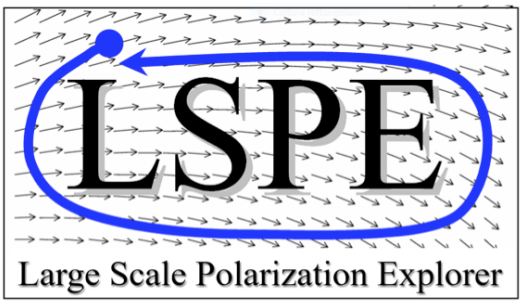
LSPE-Strip

User Guide for the functional verification pipeline

Version 1.4.0



11th May 2024

Francesco Andreetto

Contents

This document contains a guide to use the pipeline for the functional verification of the instrument LSPE-Strip that I developed form September 2022 to April 2024. I wrote part of this code during my master thesis in physics (March 2023) and I tested the current version of this pipeline on the following dataset:

- From 2nd February 2024 20:00:00 to 3rd February 2024 00:00:00;

Here you can find a quite complete report of the work but If something is not clear or could be done in an easier way feel free to contact me at francesco.andreetto97@gmail.com, I will be very happy to discuss about it!

0.1 Introduction

The document is articulated as follow:

- The first section shows step by step what to do to install all the packages to run the pipeline properly.
- The second section illustrates the procedure to run the three possible operation of the pipeline: total analysis, analysis of the bias housekeeping and analysis of the thermal sensors.
- Finally, there is a last section dedicated to every document of the GitHub repository of the pipeline.

If you are curious to see how every function is developed, I suggest you to have a look directly at the python code into the repository.

Chapter 1

Installation requirements

1. **Install anaconda.**
   To install Anaconda on a Linux distribution, you can follow these steps:

   (a) Download Anaconda: go to the Anaconda download page and find the link to download the latest version of Anaconda for Linux. You may choose either the graphical installer or the command-line installer.

   (b) Open a terminal window on your Linux system.

   (c) Navigate to the directory where the Anaconda installer is downloaded. For example use the `cd` command as follows:

   ```
   cd ~/Downloads
   ```

   (d) Make the installer executable: if you downloaded the graphical installer, you may need to make it executable. Run the following command:

   ```
   chmod +x Anaconda3-"version"-Linux-x86_64.sh
   ```

   Replace "version" with the version number you downloaded.

   (e) Run the installer using the following command:

   ```
   ./Anaconda3-"version"-Linux-x86_64.sh
   ```

   Again, replace "version" with the version number you downloaded.

   (f) Follow the installation prompts: the installer will guide you through the installation process. You'll be asked to review the license agreement, specify the installation location (press Enter to accept the default location), and whether you want to add Anaconda to your system's PATH. It's recommended to add Anaconda to the PATH.

   (g) Activate Anaconda: after the installation is complete, you may need to restart your terminal or run:

   ```
   source ~/.bashrc
   ```

   This will activate the Anaconda distribution.

(h) Test the installation by running:

```
conda list
```

This should display a list of installed packages if Anaconda was installed successfully.

You should now have Anaconda installed on your Linux system. Remember that the specific commands may vary based on the Anaconda version you downloaded, so adjust them accordingly.

2. **Create an environment where to install the new packages**.
To create a new Anaconda environment, you can use the `conda create` command. Here's the basic syntax:

```
conda create --name your_environment_name python=your_python_version
```

Replace `your_environment_name` with the desired name for your new environment and `your_python_version` with the Python version you want to use. If you don't specify a Python version, conda will use the default version.

For example, to create a new environment named *"LSPE-Strip"* with Python 3.8, you would run:

```
conda create --name LSPE-Strip python=3.8
```

After creating the environment, you can activate it using the following command:

```
conda activate LSPE-Strip
```

Now you are inside the *"LSPE-Strip"* environment, and you can install packages specific to this environment without affecting your system-wide Python installation.

Whenever you want to work within this environment, you need to activate it. To deactivate the environment and return to the base environment, you can use:

```
conda deactivate
```

I report also two other useful commands concerning the environments.

You can list all your environments with:

```
conda env list
```

This will show a list of all environments on your system, and the currently active environment will be marked with an asterisk (*).

Finally, to remove an environment, you can use the following command:

```
conda env remove --name your_environment_name
```

Replace `your_environment_name` with the name of the environment you want to remove. This will permanently delete the specified environment.

3. **Install striptease**.
   Go to the official GitHub page of the LSPE collaboration and follow the instructions to download the library striptease.

4. **Install pipeline**.
   Enter the striptease dir you cloned at the previous step. Then dowload or clone the code of the pipeline for the functional verification from my GitHub repository. Once you have done this, copy the files `official_pipeline.py` and `op_runner.py` in the striptease directory, and rename the `StripThesis` dir that contains the other modules as `pipeline`.

5. **Last step: dataset index.**
   You are almost ready to run the pipeline. What you need now is a dataset index. To get it, open the terminal and go to the *striptease* directory. Now run the command:

   ```
   ./build_hdf5_database.py hdf5_dataset_path
   ```

   where you have to replace `hdf5_dataset_path` with the actual path in which you placed your *hdf5* files containing the dataset of Strip. This command will create a dataset index in that directory and you will be able to use the dataset present in the directory. If you add or remove some of the files from the directory you will have to re-run the command once again to update the dataset index.

Chapter 2

The three operations of the Pipeline

The pipeline for the functional verification of the LSPE-Strip instrument can operate in three different ways, depending on which kind of analysis we are interested in.
The first operation is called **"tot"**; as the name suggests it is the most complete of the operations. It can analyze:

- All the responses of the 4 diodes of all the 55 polarimeters of Strip (scientific outputs);
- The Housekeeping Bias Parameters of these polarimeters, such as currents, voltages, offsets, biasing modalities for the LNAs, switching conditions of the phase switches;
- The Thermal Sensors displaced into the instrument monitoring the Temperature of the frames, of the polarimeters and of other components;

The second and the third operations can be considered "sub-operation" as they constitute only a part of the total operation, respectively the Housekeeping Analysis (**"pol_hk"**) and the Thermal Analysis (**"thermal_hk"**).

2.1 Run the pipeline

To run the pipeline in one of the three operations open the striptease directory from the command line (shell). Then, choose one of the two modality:

1. Call directly the official pipeline with the command

   ```
   python official_pipeline.py
   ```

   followed by all the **positional arguments** and the **flags** of the specific operation (See the following three sections to see how to set those parameters).

2. Call the pipeline using another program called *op_runner.py* that uses a TOML file containing all the positional arguments and the flags. If you downloaded correctly the pipeline repository you should have the 3 TOML files to run the operations (one file per operation) in the following path: */config/validation_config/*. Hence, if you want to run the **"tot"** analysis, for example, you have to open the corresponding TOML file (*config_tot.toml*), modify the flags that you want (see next sections) and than run the pipeline from the striptease directory as follows:

   ```
   python op_runner.py pipeline/config/validation_config/config_tot.toml
   ```

I suggest to use the second approach because it allows to have a better control on all the flags set for the analysis. The result will be the same but to me it's more handy to use a TOML file instead of writing an infinite epic poem of flags on the shell. But that is up to you ;)

2.2 Total Analysis

The Total Analysis parses the LSPE-Strip instrument, produce a series of plot and tables and collects them into some nice reports in format Markdown and `csv` . This operation can be run using the function **"tot"**. This operation, by activating the proper flags, allows to do the following actions:

- Single Polarimeter Analysis:
  - **Analysis of the Scientific Outputs** of one or more polarimeters.
    These Scientific Outputs are parsed in three different sub-set: Even Samples (E), Odd Samples (O) and All Samples (A), due to their different behaviours. The Scientific Outputs are of two kind: "DEM" and "PWR". For each polarimeter parsed (55 in total) we have four different exits, hence four different measures: "Q1", "Q2", "U1", "U2" for each kind of Output. The pipeline can plot the timeline of the Scientific Outputs, their Root Mean Square (RMS) and their Power Spectra Distribution (PSD) using a Fast Fourier Transformed method (FFT). Finally, the code can look for anomalous spikes into the datasets;
  - **Analysis of the Scientific Data**.
    To reduce the $\frac{1}{f}$ noise, the instrument acts on the measures performing an operation called "first demodulation", hence producing the Scientific Outputs. Then, the code must operate a "second demodulation" to produce the Scientific Data, "TOT PWR" (from the Outputs of type "PWR") and "DEMODULATED" (from "DEM"). This operation is needed to avoid the *leakage* from Temperature to polarization, which occurs when a variation in intensity I appears, for a certain fraction, as a variation of the Stokes parameters Q and U. The pipeline can plot the timeline of the Scientific Data, their Root Mean Square (RMS) and their Power Spectra Distribution (PSD) using a Fast Fourier Transformed method (FFT);
  - **Analysis of the Housekeeping Parameters** of the polarimeters.
    The pipeline analyzes the statistics of the Housekeeping Bias parameters and create a table containing info of the dataset (min value, max value, mean value, std deviation, Nan percentage, 5 percentile and 95 percentile). Then, it can plot the Housekeeping Measures.
- Global Analysis
  - **Analysis of the Thermal Sensors** of Strip. The pipeline analyzes the statistics of the Thermal measures and create a table containing info of the dataset (min value, max value, mean value, std deviation, Nan percentage, 5 percentile and 95 percentile). Then, it can plot the Thermal Measures and their Power Spectra Distribution (PSD) using a Fast Fourier Transformed method (FFT). Finally, the code can look for anomalous spikes into the datasets;
  - **Correlations**.
    The pipeline can produce correlations plots and correlation matrices of all kind of measures analyzed above:
    - TS vs TS
    - HK vs HK
    - HK vs TS

- Scientific Outputs vs TS
- Scientific Outputs vs HK
- Scientific Outputs vs Scientific Outputs

Also, using the **cross correlation** flag (see below), it is possible to create some big correlation matrices with 55x55 entries. Here every row/column correspond to a specific exit ("Q1", "Q2", "U1", "U2") of each of the 55 polarimeters of Strip.

- **Warnings**. The pipeline collects a series of warnings during the procedure. These warnings are then printed into the reports. The warnings concern:
  - *Sampling problems* (holes) during the acquisition of Scientific Outputs, Housekeeping Bias Parameters and Thermal Sensors measures;
  - *Time warnings* (when the Timestamps array and the dataset array don't have the same dimensions);
  - *High correlation values*;
  - *Spikes* in dataset or in the FFT of the measures;

The operation **"tot"** requires 5 positional arguments that are mandatory:

1. **subcommand** = "tot";
2. **path_file** (*str*): location of the data file and hdf5 file index (without the name of the file);
3. **start_datetime** (*str*): starting datetime of the analysis. Format: "YYYY-MM-DD hh:mm:ss"
4. **end_datetime** (*str*): ending datetime of the analysis. Format: "YYYY-MM-DD hh:mm:ss"
5. **name_pol** (*str*): name of the polarimeter. If more than one write them spaced into ' '.

Then, there are 24 optional **flags** that can be activated:

1. **eoa** (*str*): states which scientific data analyze. Even samples (e), odd samples (o), all samples (a). Default: 'EOA'.
2. **smooth** (*int*): Smoothing length used to flatter the data. smooth=1 equals no smooth. Default: 1.
3. **window** (*int*): Used to convert the array of the data into a matrix with "window" elements per row. Default: 1.
4. **scientific** (*bool*): If True, compute the double demodulation and analyze the scientific data. Default: False.
5. **rms** (*bool*): If True, compute the RMS on the scientific output and data. Default: False.
6. **thermal_sensors** (*bool*): If True, the code analyzes the Thermal Sensors of Strip. Default: False.
7. **housekeeping** (*bool*): If True, the code analyzes the Housekeeping parameters of the Polarimeters. Default: False.
8. **fft** (*bool*): If True, the code computes the power spectra of the scientific data. Default: False.
9. **nperseg** (*int*): number of elements of the array of scientific data on which the FFT is calculated. Default: 256.
10. **nperseg_thermal** (*int*): number of elements of thermal measures on which the FFT is calculated. Default: 256.
11. **spike_data** (*bool*): If True, the code will look for spikes in Sci-data. Default: False
12. **spike_fft** (*bool*): If True, the code will look for spikes in FFT. Default: False

13. **sam_tolerance** (*float*): the acceptance sampling tolerances of the Scientific Output. Default: 0.005
14. **ts_sam_exp_med** (*float*): the exp sampling delta between two consecutive timestamps of TS. Default: 60.
15. **ts_sam_tolerance** (*float*): the acceptance sampling tolerances of the TS. Default: 1.
16. **hk_sam_exp_med** (*dict*): the exp sampling delta between two consecutive timestamps of HK. In the order: Currents (I), Voltages (V), Offsets (O), Biasing Modality (M) and Phase Switch condition (P). Default: [1.4, 1.4, 60., 60., 60.].
17. **hk_sam_tolerance** (*dict*): the acceptance sampling tolerances of the HK parameters. In the order: Currents (I), Voltages (V), Offsets (O), Biasing Modality (M) and Phase Switch condition (P). Default: [0.1, 0.1, 0.5, 0.5, 0.5].
18. **corr_plot** (*bool*): If True, compute the correlation plot of the even-odd and scientific data. Default: False.
19. **corr_mat** (*bool*): If True, compute the correlation matrices of the even-odd and scientific data. Default: False.
20. **corr_t** (*float*): LimSup for the corr value between two dataset: if overcome, a warning is produced. Default: 0.4.
21. **cross_corr** (*bool*): If True, compute the 55x55 correlation matrices between all the polarimeters. Default: False.
22. **output_report_dir** (*str*): Path from the pipeline dir to the dir that contains the reports of the analysis. Default:'../RESULTS/PIPELINE/Reports'
23. **output_plot_dir** (*str*): Path from the pipeline dir to the dir that contains the plots of the analysis. Default:'../RESULTS/PIPELINE'
24. **report_to_plot** (*str*): Path from the Report dir to the dir that contain the plots of the analysis. Default:'../..'

2.2.1 Command Line usage

The first approach to run the pipeline requires just a command line from the shell. Since all the optional flags have default values, you can run a "default version" of the pipeline from the command line as follows:

```
python official_pipeline.py tot '/mnt/storage/data/HDF5' '2023-03-13 00:00:00' '2023-03-13 00:10:00' 'B0'
```

With this specific line, the code will operate a tot analysis on a dataset saved into the directory "data_test". The analysis is done on 10 minutes of the measures of the polarimeter "B0" collected in the dataset of the 13th of March 2023. Since all the other flags are in their default values (all set to False), the code will not analyze Thermal Sensors, Bias Housekeeping parameters of the polarimeter, RMS, FFT and correlations.

2.2.2 Toml file usage

If you want to use the second approach, here you can find the instruction to use the TOML file. For the first operation, the TOML file is called *config_tot.toml* and it is structured as follows:

```
# Positional Arguments (mandatory)
subcommand = 'tot'
path_file = '/mnt/storage/data/HDF5'
start_datetime='2024-04-02 20:00:00'
end_datetime='2024-04-02 23:59:59'
```

```
name_pol= "all" # Set ‘all’ to parse all 55 polarimeters

# Flags (optional)
# TOML
toml_file_path = "config/validation_config/config_tot.toml"
# TS Analysis
thermal_sensors = "true"
ts_sam_exp_med=20.0
ts_sam_tolerance=1.0
# HK Analysis
housekeeping = "true"
hk_sam_exp_med = "1.4, 1.4, 60.0, 60.0, 60.0"
hk_sam_tolerance = "0.1, 0.1, 0.5, 0.5, 0.5"
# Correlation Analysis
corr_plot= "true"
corr_mat= "true"
cross_corr= "true" # Set "true" for cross correlations
corr_t=0.4

# Scientific Analysis
even_odd_all=‘AOE’
scientific="true"
rms="true"
smooth=1
window=10
sam_tolerance=0.005
# Spectral Analysis
fourier="true"
# To reach a min freq of 10^-4 these are the nperseg requested
nperseg=500000
nperseg_thermal=500
# Spike Analysis
spike_data="true"
spike_fft="true"
# Output dir for plots and reports
output_plot_dir=‘../RESULTS/PIPELINE’
output_report_dir=‘../RESULTS/PIPELINE/Reports’
report_to_plot =‘../..’
```

If you want to enable a specific part of the analysis you just have to modify the proper flag in the TOML file, save it, and than run the code as follows:

```
python op_runner.py pipeline/config/validation_config/config_tot.toml
```

2.3 Analysis of the Bias Housekeeping Parameters

The Analysis of the Bias Housekeeping Parameters can be considered a sub-operation of the Total one. In fact, this operation parses only the Bias Housekeeping of the specified polarimeters, analyzing the statistics and plotting the measures. If requested with the specific flag, the code can produce also some correlation plots and matrices within the Housekeeping Parameters. All plots, the tables and the warnings produced with the code are saved into some nice reports in format Markdown and `csv` . The operation **"pol_hk"** requires 5 positional arguments that are mandatory:

1. **subcommand** = "pol_hk";
2. **path_file** (*str*): location of the data file and hdf5 file index (without the name of the file);
3. **start_datetime** (*str*): starting datetime of the analysis. Format: "YYYY-MM-DD hh:mm:ss"
4. **end_datetime** (*str*): ending datetime of the analysis. Format: "YYYY-MM-DD hh:mm:ss"
5. **name_pol** (*str*): name of the polarimeter. If more than one write them spaced into ' '.

Then, there are 8 optional **flags** that can be activated:

1. **hk_sam_exp_med** (*dict*): the exp sampling delta between two consecutive timestamps of HK. In the order: Currents (I), Voltages (V), Offsets (O), Biasing Modality (M) and Phase Switch condition (P). Default: [1.4, 1.4, 60., 60., 60.].
2. **hk_sam_tolerance** (*dict*): the acceptance sampling tolerances of the HK parameters.In the order: Currents (I), Voltages (V), Offsets (O), Biasing Modality (M) and Phase Switch condition (P). Default: [0.1, 0.1, 0.5, 0.5, 0.5].
3. **corr_plot** (*bool*): If True, compute the correlation plot of the HK. Default: False.
4. **corr_mat** (*bool*): If True, compute the correlation matrices of the HK. Default: False.
5. **corr_t** (*float*): LimSup for the corr value between two dataset: if overcome, a warning is produced. Default: 0.4.
6. **output_report_dir** (*str*): Path from the pipeline dir to the dir that contains the reports of the analysis. Default:'../RESULTS/PIPELINE'
7. **output_plot_dir** (*str*): Path from the pipeline dir to the dir that contains the plots of the analysis. Default:'../RESULTS/PIPE
8. **report_to_plot** (*str*): Path from the Report dir to the dir that contain the plots of the analysis. Default:'../..'

Once again, the code can be run directly from the command line, in the default version or setting the flags:

```
python official_pipeline.py pol_hk '/mnt/storage/data/HDF5' '2023-03-13 00:00:00' '2023-03-13 00:10:00' 'B0'
```

or using a TOML file. For this second operation the TOML file is called *config_pol_hk.toml* and it will appear with the following structure:

```
# Positional Arguments (mandatory)
subcommand = 'pol_hk'
path_file = '/mnt/storage/data/HDF5'
start_datetime='2024-02-02 20:00:00'
end_datetime='2024-02-02 23:59:59'
name_pol= ''all" # Set 'all' to parse all 55 polarimeters
```

```
# Flags (optional)
# TOML
toml_file_path = "pipeline/config/validation_config/config_pol_hk.toml"
# HK Analysis
hk_sam_exp_med = "1.4, 1.4, 60.0, 60.0, 60.0"
hk_sam_tolerance = "0.1, 0.1, 0.5, 0.5, 0.5"
# Correlation Analysis
corr_plot= "true"
corr_mat= "true"
corr_t=0.4
# Output dir for plots and reports
output_plot_dir='../RESULTS/PIPELINE'
output_report_dir='../RESULTS/PIPELINE/Reports'
report_to_plot ='../..'
```

If you want to enable a specific part of the analysis you just have to modify the proper flag in the TOML file, save it, and than run the code as follows:

```
python op_runner.py config/validation_config/config_pol_hk.toml
```

2.4 Analysis of the Thermal Sensors

The Analysis of the Thermal Sensors can be considered a sub-operation of the Total one. In fact, this operation parses only the Thermal Measures of a specific time period, analyzing their statistics, plotting the measures and, if requested, plotting their FFT. Enabling the specific flag, the code can produce also some correlation plots and matrices within the Thermal Sensors of both the states of the Multiplexer 0 and 1. All plots, tables and warnings produced with the code are saved into some nice reports in format Markdown and `csv` .
The operation **"pol_hk"** requires 4 positional arguments that are mandatory:

1. **subcommand** = "thermal_hk";
2. **path_file** (*str*): location of the data file and hdf5 file index (without the name of the file);
3. **start_datetime** (*str*): starting datetime of the analysis. Format: "YYYY-MM-DD hh:mm:ss"
4. **end_datetime** (*str*): ending datetime of the analysis. Format: "YYYY-MM-DD hh:mm:ss"

Then, there are 12 optional **flags** that can be activated:

1. **status** (*int*): status of the multiplexer of the TS to analyze: 0, 1 or 2 (which stands for both 0 and 1). Default: 2.
2. **fft** (*bool*): If True, the code computes the power spectra of the scientific data. Default: False.
3. **nperseg_thermal** (*int*): number of elements of thermal measures on which the FFT is calculated. Default: 256.
4. **spike_data** (*bool*): If True, the code will look for spikes in Scientific data. Default: False
5. **spike_fft** (*bool*): If True, the code will look for spikes in FFT. Default: False
6. **ts_sam_exp_med** (*float*): the exp sampling delta between two consecutive timestamps of TS. Default: 60.
7. **ts_sam_tolerance** (*float*): the acceptance sampling tolerances of the TS. Default: 1.
8. **corr_plot** (*bool*): If True, compute the correlation plot of the even-odd and scientific data. Default: False.
9. **corr_mat** (*bool*): If True, compute the correlation matrices of the even-odd and scientific data. Default: False.
10. **corr_t** (*float*): LimSup for the corr value between two dataset: if overcome, a warning is produced. Default: 0.4.
11. **output_report_dir** (*str*): Path from the pipeline dir to the dir that contains the reports of the analysis. Default:'../RESULTS/PIPELINE'
12. **output_plot_dir** (*str*): Path from the pipeline dir to the dir that contains the plots of the analysis. Default:'../RESULTS/PIPE
13. **report_to_plot** (*str*): Path from the Report dir to the dir that contain the plots of the analysis. Default:'../..'

Once again, the code can be run directly from the command line, in the default version or setting the flags:

```
python official_pipeline.py thermal_hk '/mnt/storage/data/HDF5' '2023-03-13 00:00:00' '2023-03-13 00:10:00'
```

or using a TOML file. For this third and last operation the TOML file is called *config_thermal_hk.toml* and it will appear with the following structure:

```
# Positional Arguments (mandatory)
subcommand = 'thermal_hk'
path_file = '/mnt/storage/data/HDF5'
start_datetime='2024-02-02 20:00:00'
end_datetime='2024-02-02 20:00:00'

# Flags (optional)
# TOML
toml_file_path = "config/validation_config/config_thermal_hk.toml"
# TS Analysis
ts_sam_exp_med=20.0
ts_sam_tolerance=1.0
# Status
status = 2
# Correlation Analysis
corr_plot= "true"
corr_mat= "true"
corr_t=0.4
# Spectral Analysis
fourier="true"
# To reach a min freq of 10^-4 this is the nperseg requested
nperseg_thermal=500
# Spike Analysis
spike_data="true"
spike_fft="true"
# Output dir for plots and reports
output_plot_dir='../RESULTS/PIPELINE'
output_report_dir='../RESULTS/PIPELINE/Reports'
report_to_plot ='../..'
```

If you want to enable a specific part of the analysis you just have to modify the proper flag in the TOML file, save it, and than run the code as follows:

```
python op_runner.py config/validation_config/config_thermal_hk.toml
```

Chapter 3

The code of the pipeline

In this chapter I will briefly illustrate the content of every file that compose the pipeline for the functional verification of the LSPE-Strip instrument. If you want to dig deeper in the code, I suggest you to have a direct look at the files in the GitHub repository of the pipeline.

3.1 op_runner.py

This program is used to run the new version of the pipeline for functional verification of LSPE-STRIP (2024) using TOML files passed by the command line. The programm **op_runner.py** contains the function `load_toml_config` which accepts as parameter the path of a TOML file and loads the arguments of the TOML file into a dict. Using this function and the module `Argparse`, the code extracts the arguments from the TOML file and builds the command to call the pipeline.

3.2 official_pipeline.py

This program is the core of the whole work. It contains all the operations to parse the dataset and produce the reports containing all the plots, the tables, the warnings and the results of the analysis. This python code is divided in five parts:

1. In the first part, I used the module `Argparse` to create a common parser that collects the parameters which are common to all the three operations (**"tot"**, **"pol_hk"** and **"thermal_hk"**) and three sub-parser, one for each operation, with their specific parameters;

2. In the second part, all the parameters collected in the parsers are analyzed to control their coherence with their future usage in the analysis;

3. In the third part, the code collects and prepares all the information to produce the reports;

4. In the fourth part, the code calls one of the three operations as specified by the user. These operations produces many little Markdown reports divided into sections: General Information, Warnings, HK Analysis, TS Analysis, EOA Outputs Analysis, Scientific Data Analysis, Correlations Matrices and Plots;

5. In the last part, the heading of the reports is produced and the Markdown Reports of the sections of the analysis are merged into one.

3.3 polarimeter.py

This file contains the Class `Polarimeter`. Part of this code was used in my bachelor thesis (2020) and master thesis (2023). Now you can use this Class with the new version of the pipeline for functional verification of LSPE-STRIP (2024). The idea is to create a virtual object that represents the real polarimeter, a receiver of microwaves that produces 4 outputs from 4 pin-diode. The constructor of the class accepts 4 parameters:

1. **name_pol** (*str*): name of the polarimeter;
2. **path_file** (*str*): location of the data file and hdf5 file index (without the name of the file);
3. **start_datetime** (*str*): start time (of the analysis);
4. **end_datetime** (*str*): end time (of the analysis);
5. **output_plot_dir** (*str*): Path from the pipeline dir to the dir that contains the plots of the analysis.

3.3.1 Members of Polarimeter

The members of the Class `Polarimeter` are:

- **name** (*str*): the name of the polarimeter
- **ds** (*Datastorage*): a Datastorage from the path of the file;
- **STRIP_SAMPLING_FREQ** (*int*): Sampling Frequency of Strip. Std value = 100 Hz;
- **norm_mode** (*int*): Normalization Modality of the samples: 0 (output vs index), 1 (output vs time in s), 2 (output vs time in Julian Date JHD);
- **date** (*list*): Julian Date MJD [start, end];
- **gdate** (*list*): Gregorian Date (in string format) [start, end];
- **date_dir** (*str*): Directory where to save all plot for a given analysis;
- **times** (*list*): Timestamps list;
- **data** (*dict*): Dictionary for scientific outputs PWR and DEM. It contains two dict: `dem` and `power` ("DEM": `dem`, "PWR": `power`);
- **hk_list** (*dict*): dict with four keys (V, I, O, M, P), each associated to one of the HK parameters: tensions (V), currents (I), offset (O), biasing modality (M) and Phase Switch condition (P). Each key is linked to a list with the names of all the Housekeeping parameters;
- **hk** (*dict*): Dictionary for Housekeeping Analysis. It contains four dictionaries: tensions (V), currents (I), offset (O), biasing modality (M) and Phase Switch condition (P);
- **hk_t** (*dict*): Dictionary containing four lists of timestamps, one per each HK parameter (V, I, O, M, P).
- **warnings** (*dict*): Dictionary containing 5 lists, each containing warning messages of a specific ambit, tables and synthetic results of the analysis: `time_warning`, `sampling_warning`, `corr_warning`, `eo_warning`, `spike_warning`.

3.3.2 Methods of Polarimeter

The methods of the Class `Polarimeter` are:

- **Load_Pol**: Load all dataset in the polarimeter. All the exits "Q1", "Q2", "U1", "U2" of all types "DEM" and "PWR".
- **Load_X**: Load only a specific type of dataset "PWR" or "DEM" in the polarimeter.
  Parameter: **type** (*str*) "PWR" or "DEM";
- **Load_Times**: Load the timestamps in the polarimeter and put to 0 the `Strip Sampling Frequency`.
  Parameter: **range** (*Time*) is an array-like object containing the `Time` objects: `start_date` and `end_date`.
- **Date_Update**: Calculates and returns the new Gregorian date in which the experience begins, given a number of samples that must be skipped from the beginning of the dataset.
  Parameters:
  - **n_samples** (*int*): number of samples that must be skipped;
  - **modify** (*bool*):
    - *True*: The beginning date is definitely modified and provided.
    - *False*: A copy of the beginning date is modified and provided.
- **Clip_Values**: Data cleansing. Scientific Outputs with value zero at the beginning and at the end are removed from the dataset. Control that a channel ("Q1", "Q2", "U1" or "U2") doesn't turn on before the others.
- **STRIP_SAMPLING_FREQUENCY_HZ**: Calculate the `Strip Sampling Frequency` by dividing the number of output saved during a period of time. Std value = 100. It depends on the electronics hence it's the same for all polarimeters. Note: it must be defined before time normalization.
- **Norm**: Timestamps Normalization.
  Parameter: **norm_mode** (*int*) can be set in three ways:
  - 0) the output is expressed in function of the number of samples
  - 1) the output is expressed in function of the time in s from the beginning of the experience
  - 2) the output is expressed in function of the number of the Julian Date JHD
- **Prepare**: Prepare the polarimeter in two steps: 1. Calculate Strip Sampling Frequency 2. Normalize timestamps
  Parameter: **norm_mode** (*int*) can be set in three ways:
  - 0) the output is expressed in function of the number of samples
  - 1) the output is expressed in function of the time in s from the beginning of the experience
  - 2) the output is expressed in function of the number of the Julian Date JHD
- **Demodulation**: Calculate the Scientific data DEMODULATED or TOTAL POWER at 50Hz. Timestamps are chosen as mean of the two consecutive times of the DEM/PWR data.
  Parameters:
  - **exit** (*str*) "Q1", "Q2", "U1", "U2";
  - **type** (*str*) of data "DEM" or "PWR";

  - **begin**, **end** (*int*): indexes of the data that have to be considered

- **Load_HouseKeeping**: Load all House-Keeping parameters using the module `load_hk`, taking the names from the list in the constructor.

- **Norm_HouseKeeping**: Check if the Timestamps array and the House-keeping data array have the same length. Normalize all House-Keeping's timestamps putting one every 1.4 seconds from the beginning of the dataset. Return a list of problematic HK.

- **Analyse_HouseKeeping**: Analise the following HouseKeeping parameters: I Drain, I Gate, V Drain, V Gate, Offset. See `self.hk_list` in the constructor. Calculate the mean the std deviation.

- **HK_table**: Create a dictionary containing a string and a list to produce a table of Housekeeping results. The string contains the code for Markdown reports, the list is used for `csv` reports.
  In the table there are the following info:

  1. HK-Parameter name
  2. Max value
  3. Min value
  4. Mean value
  5. Standard deviation
  6. NaN percentage

  The HouseKeeping parameters included are: I Drain, I Gate, V Drain, V Gate, Offset.
  Parameters: **results** (*dict*): contains the info about HK analysis obtained with Analyze_Housekeeping

- **HK_Sampling_Table**: Create a dictionary with the info of the housekeeping parameter sampling. The dictionary has two keys "md" and "csv" - each contains a list with the info to create the relative report The current code produces a table with the following information:

  1. HK-Parameter name
  2. Number of sampling jumps
  3. Median jump
  4. Expected median jump
  5. The 5th percentile
  6. The 95th percentile

  The HouseKeeping parameters included are: I Drain, I Gate, V Drain, V Gate, Offset.
  Parameters:

  - **sam_exp_med** ("dict"): contains the exp sampling delta between two consecutive timestamps of the HK;
  - **sam_tolerance** ("dict"): contains the acceptance sampling tolerances of the HK parameters: I,V,O;

- **Plot_Housekeeping**: Plot all the acquisitions of the chosen HouseKeeping parameters of the polarimeter.
  Parameters:

  - **hk_kind** (*str*): defines the HK to plot.

    - V - Drain Voltage and Gate Voltage;
    - I - Drain Current and Gate Current;
    - O - Offsets;
    - M - Biasing Modality (Open Loop - Closed Loop);
    - P - Phase Switch condition.
  - **show** (*bool*):
    - True - show the plot and save the figure;
    - False - save the figure only.

- **Plot_Band**: Plot the bands of the 4 exits "PWR"/"DEM" or "TOT POWER"/"DEMODULATED" of the Polarimeter.
  Parameters:
  - **type** (*str*) of data "DEM" or "PWR";
  - **demodulated** (*bool*):
    - True - demodulated data are computed;
    - False - even, odd, all samples are plotted;
  - **file_path** (*str*): Path of the data file.hdf5 (including its name)
  - **output_path** (*str*): Path of the dir where the band plots are saved;
  - **s_start** (*int*): Number of seconds from the tag acquisition to the beginning of first band;
  - **s_duration** (*int*): Duration of the first band in seconds;
  - **f_i** (*float*): initial frequency at which the band starts;
  - **f_f** (*float*): final frequency at which the band arrives;
  - **binning** (*bool*):
    - True - bin the dataset loaded;
    - False - no binning;
  - **binning_length** (*int*): number of elements on which the mean in the binning is calculated;
  - **show** (*bool*):
    - True - show the plot and save the figure;
    - False - save the figure only.

- **Plot_Output**: Plot the 4 exits PWR or DEM of the Polarimeter.
  Parameters:
  - **type** (*str*) of data "DEM" or "PWR";
  - **begin**, **end** (*int*): indexes of the data that have to be considered;
  - **show** (*bool*):
    - True - show the plot and save the figure;
    - False - save the figure only.

- **Jump_Plot**: Plot the Timestamps and the Delta-time between two consecutive Timestamps. Note: the Polarimeter must be Loaded but not Prepared hence DO NOT normalize the Timestamps!
  Parameters:
- **show** (*bool*):
  - True - show the plot and save the figure;
  - False - save the figure only.
- **Pol_Sampling_Table**: Return a list to produce a `csv` table that contains a description of jumps in Polarimeter Timestamps. This function store also a table in Markdown format in the member warnings of the Polarimeter (`self.warnings`).
  The current code produces a table with the following information:
  1. Name_Polarimeter
  2. Jump_Index
  3. Jump_Value [JHD]
  4. Jump_Value [s]
  5. Gregorian Date
  6. Julian Date

  Parameters:
  - **start_datetime** ("str"): start time that defines the polarimeter, format: "YYYY-MM-DD hh:mm:ss"
  - **sam_tolerance** ("float"): the acceptance sampling tolerances of the scientific output
- **Spike_Report**: Create a dictionary with the info of the spikes in Output and FFT of the Outputs. The dictionary has two keys "md" and "csv" - that contain a *list* and a *str* with the info to create the report. Look up for 'spikes' in the DEM and PWR output of the Polarimeter and in their FFT. Create a table in `md` and `csv` language in which the spikes found are listed.
  - **fft** (*bool*): if True, the code looks for spikes in the FFT;
  - **nperseg** (*int*): number of elements of the array on which the FFT is calculated.
- **spike_CSV**: Look up for 'spikes' in the DEM and PWR output of the Polarimeter. Create *list* of *str* to be written in a `csv` file in which the spikes found are listed.
- **Inversion_EO_Time**: Find the inversions between even and odd output during the sampling due to time jumps. It could be also used to find even-odd inversions given a generic vector of position defining the intervals.
  Parameters:
  - **jump_pos** (*list*): list that contains the indexes of the time jumps (obtained with the function `find_jump`).

3.4 thermalsensors.py

This file contains the Class `Thermal_Sensors`. Use this Class with the new version of the pipeline for functional verification of LSPE-STRIP (2024). The idea is to create a virtual object that represents the real Thermal Sensors, displaced around the instrument monitoring the variation of temperature during the data acquisition and analysis. The constructor of the class accepts 4 parameters:

1. **path_file** (*str*): location of the data file and hdf5 file index (without the name of the file);
2. **start_datetime** (*str*): start time (of the analysis);
3. **end_datetime** (*str*): end time (of the analysis).
4. **status** (*int*): status of the multiplexer of the TS to analyze: 0, 1 or 2 (which stands for both 0 and 1);
5. **output_plot_dir** (*str*): Path from the pipeline dir to the dir that contains the plots of the analysis.
6. **nperseg_thermal** (*str*): number of elements of thermal measures on which the FFT is calculated. Then the average of all periodograms is computed to produce the spectrogram. Changing this parameter allow to reach lower frequencies in the FFT plot: in particular, the limInf of the x-axis is fs/nperseg, where fs is the sampling frequency.

3.4.1 Members of Thermal_Sensors

The members of the Class `Thermal_Sensors` are:

- **ds** (*Datastorage*): a Datastorage from the path of the file;
- **date** (*list*): Julian Date MJD [start, end];
- **gdate** (*list*): Gregorian Date (in string format) [start, end];
- **date_dir** (*str*): Directory where to save all plot for a given analysis;
- **status** (*int*): Status of the multiplexer of the TS to analyze: 0, 1 or 2 (which stands for both 0 and 1);
- **ts_names** (*dict*): Dictionary containing the names of the TS divided in the two states: “0” and “1”. The TS in each state are divided in groups whose names suggest their positions on Strip or their functions: “TILES”, “FRAME”, “POLAR”, “100-200K”, “VERIFY”, “COLD_HEAD”;
- **ts** (*dict*): Dictionary containing all TS measures and timestamps. It has two keys:
  - **thermal_times** (*list*): Timestamps list;
  - **thermal_data** (*dict*): Dictionary for thermal measures. It contains two dict: `raw` and `calibrated`.
- **nperseg_thermal** (*str*): number of elements of thermal measures on which the FFT is calculated.
- **warnings** (*dict*): Dictionary containing 5 lists, each containing warning messages of a specific ambit, tables and synthetic results of the analysis: `time_warning`, `corr_warning`, `spike_warning`.

3.4.2 Methods of Thermal_Sensors

The methods of the Class `Thermal_Sensors` are:

- **Load_TS**: Load all Thermal Sensor measures taking the names from the list in the constructor;
- **Clean_TS**: Clean all Thermal Sensor measures removing those whose acquisition presents Nan values;
- **Norm_TS**: Check if the Timestamps array and the CALIBRATED DATA array have the same length. Normalize all Thermal Sensor's Timestamps putting one every 20 seconds from the beginning of the dataset. Return a list of problematic TS;
- **TS_Sampling_Table**: Create a dictionary with the info of the Thermal Sensors sampling. The dictionary has two keys "md" and "csv" - each containing a list with the info to create the relative report.
  The current code produces a table with the following information:
  1. TS name
  2. Number of sampling jumps
  3. Median jump
  4. Expected median jump
  5. The 5th percentile
  6. The 95th percentile

  Parameters:
  - **sam_exp_med** (*dict*): contains the exp sampling delta between two consecutive timestamps of the TS;
  - **sam_tolerance** (*dict*): contains the acceptance sampling tolerances of the TS;
- **Analyze_TS**: Analyze all Thermal Sensors' output: calculate the mean the std deviation for both raw and calibrated samples.
- **Thermal_Table**: Create a dictionary containing a *string* and a *list* to produce a table of thermal results for both raw and calibrated measures. The string contains the code for Markdown reports, the list is used for `csv` reports.
  In the table there are the following info:
  1. Sensor name;
  2. Status of acquisition (0 or 1);
  3. Group of the sensor;
  4. Expected median jump;
  5. Max value measured;
  6. Min value measured;
  7. Mean value;
  8. Standard deviation;
  9. NaN percentage.
- **Plot_TS**: Plot all the calibrated acquisitions of Thermal Sensors.
  Parameters:

  - **show** (*bool*):
    - True - show the plot and save the figure;
    - False - save the figure only.

- **Plot_FFT_TS**: Plot the FFT of the calibrated acquisitions of Thermal Sensors of the polarimeter.
  Parameters:
  - **all_in** (*bool*):
    - True - show all the FFT of TS in one plot;
    - False - show the FFT of TS in different plots for the groups.
  - **show** (*bool*):
    - True - show the plot and save the figure;
    - False - save the figure only.

- **Spike_Report**: Look up for 'spikes' in the TS output of Strip or in their FFT. Create a dictionary containing two tables in which the spikes found are listed:
  1. in `md` language (basically a str)
  2. in `csv` language (a list)

  Parameters:
  - **fft** (*bool*): if True, the code looks for spikes in the FFT.
  - **nperseg** (*int*): number of elements of the array on which the FFT is calculated.

3.5 f_strip.py

This file contains the main functions used in my bachelor thesis (2020) updated to be used in the new version of the pipeline for functional verification of LSPE-STRIP (2024). Here you can find a quick sommarium of all the functions, if you want to dig deeper in the code, I suggest you once again to have a direct look at the files in the GitHub repository of the pipeline.

- **binning_function**: Operates a binning of the data_array by doing a mean of a number of samples equal to bin_length.
  Parameters:
  - **data_array** (*str*): array-like object;
  - **binning_length** (*int*): number of elements on which the mean is calculated.

- **tab_cap_time**: Create a new file `.csv` and write the caption of a tabular. This specific function creates a tabular that collects the jumps in the dataset (JT).
  Parameters:
  - **pol_name** (*str*): Name of the polarimeter;
  - **file_name** (*str*): Name of the file to create and in which insert the caption;
  - **output_dir** (*str*): Name of the dir where the `csv` file must be saved

- **pol_list**: Create a list of the polarimeters present in the datafile
  Parameters:
  - **path_dataset** (*str*): Path comprehensive of the name of the dataset file;
- **mean_cons**: Calculate consecutive means between the elements of an array. The mean on each couple of samples of even-odd index is computed.
  Parameters:
  - **v** (*list*): is an array-like object;
- **diff_cons**: Calculate consecutive differences between the elements of an array. The difference between each couple of samples of even-odd index is computed.
  Parameters:
  - **v** (*list*): is an array-like object;
- **mob_mean**: Calculate a mobile mean on a number of elements given by `smooth_len`, used to smooth plots.
  Parameters:
  - **v** (*list*): is an array-like object;
  - **smooth_len** (*int*): number of elements on which the mobile mean is calculated.
- **demodulate_array**: Demodulation over an array. Calculate the double demodulation at 50Hz of the dataset provided.
  Parameters:
  - **array** (*str*): array-like dataset;
  - **type** (*str*) of data "DEM" or "PWR";
- **demodulation**: Calculate the double demodulation at 50Hz of the dataset provided. Timestamps are chosen as mean of the two consecutive times of the DEM/PWR data.
  Parameters:
  - **dataset** (*dict*): dictionary containing the dataset with the output of a polarimeter;
  - **timestamps** (*list*): list containing the Timestamps of the output of a polarimeter;
  - **exit** (*str*) "Q1", "Q2", "U1", "U2";
  - **type** (*str*) of data "DEM" or "PWR";
  - **begin**, **end** (*int*): interval of dataset that has to be considered.
- **rolling_window**: Accepts a vector and return a matrix with:
  - A number of element per row fixed by the parameter window;
  - The first element of the row j is the j element of the vector.

  Parameters:
  - **v** (*list*): is an array-like object;
  - **window** (*int*): see above.

- **RMS**: Calculate the RMS of a vector using the rolling window
  Parameters:
  - **data** (*dict*): is a dictionary with four keys (exits) of a particular type "DEM" or "PWR";
  - **window** (*int*): number of elements on which the RMS is calculated;
  - **exit** (*str*) "Q1", "Q2", "U1", "U2";
  - **eoa** (*str*) flag used to calculate RMS for:
    * all samples (`eoa`=0), can be used for Demodulated and Total Power scientific data (50Hz);
    * odd samples (`eoa`=1);
    * even samples (`eoa`=2).
  - **begin**, **end** (*int*): interval of dataset that has to be considered.
- **EOA**: return a string with the letters E, O and A.
  Parameters:
  - **even**, **odd**, **all** (*int*): If these variables are different from zero, this function returns a string that contains the corresponding letters.
- **eoa_values**: Return a list in which each element is a tuple of 3 values. Those values can be 0 or 1 depending on the letters (e, o, a) provided. Note: if a letter is present in the `eoa_str`, then that letter will assume both value 0 and 1. Only 0 otherwise.
  Parameters:
  - **eoa_str** (*str*): string of 0, 1, 2 or 3 letters from a combination of the letters e, o and a.
- **letter_combo**: Return a list in which each element is a combination of E, O, A letters.
  Parameters:
  - **in_str** (*str*): generic string of max 3 letters.
- **find_spike**: Look up for 'spikes' in a given array. Calculate the median and the mad and uses those to discern spikes.
  Parameters:
  - **v** (*list*): is an array-like object;
  - **type** (*str*): if "DEM" look for spikes in two sub arrays (even and odd output) if "FFT" select only spike up;
  - **threshold** (*int*): value used to discern what a spike is;
  - **n_chunk** (*int*): number of blocks in which the array is divided. On every block the median is computed to find spikes.
- **select_spike**:
  Parameters:
  - **spike_idx** (*list*): is an array-like object containing the indexes of the spikes present in the s array;
  - **s** (*list*): is an array-like object that contains spikes;
  - **freq** (*list*): is an array-like object that contains the frequency corresponding to the s values.

- **find_jump**: Find the 'jumps' in a given Time astropy object: the samples should be consequential with a fixed growth rate. Hence, their consecutive differences should have an expected median within a certain tolerance. Returns 'jumps' a dictionary containing five keys:

  1. **n** (*int*): is the number of jumps found;
  2. **idx** (*int*): index of the jump in the array;
  3. **value** (*float*): is the value of the jump in JHD;
  4. **s_value** (*float*): is the value of the jump in seconds;
  5. **median_ok** (*bool*): True if there is no jump in the vector, False otherwise.

  Parameters:

  - **v** (*list*): is a `Time` object from astropy (i.e. `Polarimeter.times`);
  - **exp_med** (*float*): expected median (in seconds) of the `TimeDelta` between two consecutive values of v;
  - **tolerance** (*float*): threshold number of seconds over which a `TimeDelta` is considered as an error.

- **get_tags_from_iso**: Get the tags in a given time interval contained in a file dir.
  Parameters:

  - **dir_path** (*str*): Path of the data dir;
  - **start_time** (*str*): start time in iso format;
  - **end_time** (*str*): end time in iso format;

- **get_tags_from_file**: Get the tags form a given file.
  Parameters:

  - **file_path** (*str*): Path of the data file;

- **get_tag_times**: Find the start-time and the end-time of a given tag. Return a list containing 4 elements: start and end time in mjd and iso format.
  Parameters:

  - **file_path** (*str*): Path of the data file;
  - **file_tag** (*str*): Name of the tag of a specific subset of data (i.e. of a test).

- **dir_format**: Take a string and return a new string changing white spaces into underscores, ":" into "-" and removing ".000"
  Parameters:

  - **old_string** (*str*)

- **csv_to_json**: Convert a `csv` file into a `json` file.
  Parameters:

  - **csv_file_path** (*str*): path of the `csv` file that have to be converted;
  - **json_file_path** (*str*): path of the `json` file converted.

- **merge_report**: Merge together all the `md` report files into a single `md` report file.
  Parameters:

  - **md_reports_path** (*str*): path of the `md` files that have to be merged;
  - **total_report_path** (*str*): path of the `md` file merged.

- **name_check**: Check if the names of the polarimeters in the list are wrong: not the same as the polarimeters of Strip.
  Parameters:
  - **names** (*list*): list of the names of the polarimeters.

- **datetime_check**: Check if the string is in datatime format "YYYY-MM-DD hh:mm:ss" or not.
  Parameters:
  - **date** (*str*): string with the datetime.

- **date_update**: Calculates and returns the new Gregorian date in which the analysis begins, given a number of samples that must be skipped from the beginning of the dataset.
  Parameters:
  - **start_datetime** (*str*): start time of the dataset;
  - **n_samples** (*int*): number of samples that must be skipped;
  - **sampling_freq** (*int*): number of data collected per second;
  - **ms** (*bool*): if True the new Gregorian date has also milliseconds.

- **same_length**: Check if the two array are of the same length. If not, the longer becomes as long as the smaller.
  Parameters:
  - **array1, array2** (*list*): data arrays.

- **data_plot**: Generic function that create a Plot of the dataset provided.
  Parameters:
  - **pol_name** (*str*): name of the polarimeter we want to analyze;
  - **dataset** (*dict*): dictionary containing the dataset with the output of a polarimeter;
  - **timestamps** (*list*): list containing the Timestamps of the output of a polarimeter;
  - **start_datetime** (*str*): start time;
  - **end_datetime** (*str*): end time;
  - **begin, end** (*int*): interval of dataset that has to be considered;
  - **type** (*str*): defines the scientific output, "DEM" or "PWR"
  - **even, odd, all** (*int*): used to set the transparency of the dataset (0=transparent, 1=visible);
  - **demodulated** (*bool*): if True, demodulated data are computed, if False even-odd-all output are plotted;
  - **rms** (*bool*): if True, the RMS are computed;
  - **fft** (*bool*): if True, the FFT are computed;
  - **window** (*int*): number of elements on which the RMS is calculated;
  - **smooth_len** (*int*): number of elements on which the mobile mean is calculated;
  - **nperseg** (*int*): number of elements of the array of scientific data on which the FFT is calculated;

  - **output_plot_dir** (*str*): Path from the pipeline dir to the dir that contains the plots of the analysis;
  - **show** (*bool*):
    - True - show the plot and save the figure;
    - False - save the figure only.

3.6 f_correlation_strip.py

This file contains the main correlation functions used in the new version of the pipeline for functional verification of LSPE-STRIP (2024). Here you can find a quick sommarium of the functions, if you want to dig deeper in the code, I suggest you once again to have a direct look at the files in the GitHub repository of the pipeline.

- **correlation_plot**: Create a Correlation Plot of two dataset: two array, two dictionaries or one array and one dictionary. Return a dictionary of warnings that highlights which data are highly correlated.
  Parameters:
  - **list1, list2** (*list*): arrays of n1 and n2 elements;
  - **dict1, dict2** (*dict*): dictionaries with N1, N2 keys;
  - **time1, time2** (*list*): arrays of timestamps: not necessary if the dataset have same length;
  - **data_name1, data_name2** (*str*): names of the dataset. Used for titles, labels and to save the png;
  - **measure_unit1, measure_unit2** (*str*): measure units. Used for labels in the plots;
  - **start_datetime** (*str*): begin date of dataset. Used for the title of the figure and to save the png;
  - **end_datetime** (*str*): end date of dataset. Used for the title of the figure and to save the png;
  - **show** (*bool*):
    - True - show the plot and save the figure;
    - False - save the figure only.
  - **down_sampling** (*bool*):
    - True - down-sampling between arrays with mismatched length is computed;
    - False - interpolation between arrays with mismatched length is computed.
  - **corr_t** (*float*): LimSup for the corr value between two dataset: if overcome, a warning is produced;
  - **plot_dir** (*str*): path where the plots are organized in directories and saved.
- **correlation_mat**: Plot a 4x4 Correlation Matrix of two generic dictionaries (also of one with itself). Return a list of warnings that highlight which data are highly correlated.
  Parameters:
  - **dict1, dict2** (*dict*): dictionaries, dataset;
  - **data_name1, data_name2** (*str*): names of the dataset. Used for titles, labels and to save the png;
  - **start_datetime** (*str*): begin date of dataset. Used for the title of the figure and to save the png;
  - **show** (*bool*):
    - True - show the plot and save the figure;
    - False - save the figure only.

  - **corr_t** (*float*): LimSup for the corr value between two dataset: if overcome, a warning is produced;
  - **plot_dir** (*str*): path where the plots are organized in directories and saved.

- **cross_corr_mat**: Plot 55x55 matrices of every data-kind DEM/PWR of every exit combination: Q1, Q2, U1, U2. Return a `dict` of warnings that highlight which data are highly correlated.
  Parameters:
  - **path_file** (*dict*): location of the data file, it is indeed the location of the hdf5 file's index;
  - **start_datetime** (*str*): begin date of dataset. Used for the title of the figure and to save the png;
  - **end_datetime** (*str*): end date of dataset. Used for the title of the figure and to save the png;
  - **show** (*bool*):
    - True - show the plot and save the figure;
    - False - save the figure only.
  - **corr_t** (*float*): LimSup for the corr value between two dataset: if overcome, a warning is produced;
  - **plot_dir** (*str*): path where the plots are organized in directories and saved.

3.7 Total Analysis - stripop_tot.py

You can find here a scheme of the process of the Total Analysis.

1. Loading Report information;
2. Thermal Sensors Analysis:
   - Load TS;
   - Look for Spikes in TS and in FFT of TS;
   - Clean TS;
   - Normalize TS;
   - TS Tables;
   - TS Plot;
   - TS FFT Plot;
   - Create report of TS.
3. Multi Polarimeter Analysis: Cross Correlation Matrix;
4. Single Polarimeter Analysis, repeated for each `Polarimeter` requested:
   - Load HK;
   - HK Tables;
   - Create report of HK;
   - Loading Scientific Outputs;
   - Look for Spikes in Outputs and in FFT of Outputs;
   - Even Odd All Plots of Outputs, RMS and FFT

- Report Even, Odd, All;
- Scientific Analysis: Plot of demodulated data, RMS and FFT;
- Report Scientific Data;
- Correlation Plots and Matrices: Output, TS and HK.

5. Other Correlation Plots and Matrices: TS;
6. Report Correlations;
7. Create report of Warnings of the Analysis.

3.8 Analysis of the Bias Housekeeping Parameters stripop_pol_hk.py

You can find here a scheme of the process of the Housekeeping Analysis.

1. Loading Report information;
2. Loading Housekeeping Parameters;
3. HK Tables;
4. HK Normalization;
5. HK Plots;
6. Correlations Plots and Matrices: HK;
7. Create report of Housekeeping Parameters;
8. Create report of Warnings of the Analysis.

3.9 Analysis of the Thermal Sensors - stripop_thermal_hk.py

You can find here a scheme of the process of the Thermal Sensors Analysis.

1. Load TS;
2. Look for Spikes in TS and in FFT of TS;
3. Clean TS;
4. Normalize TS;
5. TS Tables;
6. TS Plot;
7. TS FFT Plot;
8. Correlations Plots and Matrices: TS;
9. Create report of TS;
10. Create report of Warnings of the Analysis.