Developer Guide for Project `HDF5_BLS`

Pierre Bouvet

January 31, 2025

Contents

1 Introduction

This document serves as the developer guide for the modules of the`HDF5_BLS` project. It provides details about the approach to defining new functions and integrating them into the project to ensure code reusability and maintainability and allow an adaptation to new needs.

2 Module Structure

The project has the following structure:

```
HDF5_BLS.
|-- conversion_PSD_modules.
    |-- ar_BLS_VIPA
|-- conversion_PSD
|-- load_data
|-- wrapper
|-- treat
```

2.1 Description of the structure

- **conversion_PSD_modules.**: The sub-package that contains all the modules for the conversion of arrays to Power Spectrum Density.
  - **ar_BLS_VIPA**: A module for angle-resolved VIPA spectrometer data.
- **conversion_PSD**: The module that contains the logic specific to each technique to ensure the data can be treated.
- **load_data**: Allows the import of data from different formats.
- **wrapper**: Contains the Wrapper object that allows the manipulation of the data and its attributes.
- **treat**: Contains the treatment functions.

3 Importing new formats

To import a new format, you need to create a new function in the `load_data` module. This function should take the file path as input and return the data and its extracted attributes. The function should have the following structure:

```
def load_file_format(filepath):
    data = # Load data from the file
    attributes = # Extract attributes from the file
    return data, attributes
```

3.1 Data format

The format of the data should be a numpy array with the last dimension being the number of spectral channels. For example: a 2D raster scan with 200 x points and 100 y points acquired with a spectrometer with 512 spectral channels would be represented as a numpy array of shape (200, 100, 512).

3.2 Attributes format

The attributes is a dictionary. The name of the keys of the dictionnary should match the list given in the `spreadsheets/attributes_v01.xlsx` file. The values should be strings.

Additionnaly, it is possible to add new attributes to the data. To do so, you need to add a new key to the attributes dictionary with the following structure:

```
    attributes['CATEGORY.Attribute'] = # Value of the new attribute
```

where `CATEGORY` is either `SPECTROMETER` (if its spectrometer related), `MEASURE` (if its measure related) or `FILEPROP` (if its file related), and `Attribute` is the name of the attribute. It is recommended to use the same name as the key in the spreadsheet, and update the spreadsheet with the new attribute. Additionnaly, we recommend to send a pull request to the repository to update the spreadsheet so as to normalize the naming convention in the community.

4 Adding a new algorithm to convert raw data in Power Spectrum Density

This project has made it easy(ish) for users to add their own algorithms to convert their data arrays in PSD arrays. This is done by imposing a certain nomenclature that will be defined hereafter.

4.1 Where to add the function

The project is made so that all modules inherent to the extraction of a PSD from an array are located in the "conversion_PSD_module" folder of the HDF5_BLS project folder. Each module is defined by the type of data it is meant to treat. For example, the module "ar_BLS_VIPA.py" is meant to treat data obtained with an angle-resolved VIPA spectrometer.

If the type of technique you use is not already defined, you can create a new module with the name of your technique. Note that this name should be the same as the one used for the "SPECTROMETER.Type" attribute in the HDF5 file.

If the type of technique you use is defined but you still want your own treatment to be accssible, you just need to add your function to the module, following the rules we will define in the following section.

4.2 Nomenclature of the arguments

The nomenclature is as follows:

- Arguments starting with "n_" are arguments that are not meant to adjust a treatment on the data array (for example the array itself).
- Arguments starting with "g_" are arguments that refer to a position on the data array (a particular pixel for example).
- Other arguments are arguments that can be adjusted by the user.

4.3 Definition of the function

Additionnaly, all the types of the arguments need to be defined in the funciton, as well as the return type. This is done the following way:

```
def function_name(n_array: np.ndarray, g_p: int, c0: float) -> tuple:
```

4.4 Doctype

As for the rest of the project, the docstrings are written in the numpydoc format. This is done the following way:

```
def function_name(n_array: np.ndarray, g_p: int, c0: float) -> tuple:
    """
    This function does something.

    Parameters
    ----------
    n_array : np.ndarray
        The array of the data.
    g_p : int
        The position on the array.
    c0 : float
        The constant.

    Returns
    -------
    tuple
        The tuple of the PSD array.
    """
```

5 Modifying the GUI

Once the functions are accessible in the library, it is interesting to allow them to be used inside the GUI of the project. The way the GUI has been developped, the functions you have added to the library are automatically integrated into the GUI. However, because the behavior of the GUI with your function might depend on the way you have implemented it, you need to add your own modifications to the GUI in order to use them properly. This section will detail how to do this.

5.1 Spirit of the GUI

Once your data are added to the GUI and the type of spectrometer has been specified in the data attributes, you can use the GUI to compute the PSD. This can be done by left clicking on one curve or group, and then clicking on `Get Power Spectrum Density`. A verification on the parameters is then performed to know wether the conversion to a PSD is possible or not. If it is, the PSD is computed and directly added to the GUI. If not, then a custom algorithm is run.

Your job here is:

1. To define the conditions that will allow the conversion to a PSD.
2. To implement the functions that are needed to get the previously mentionned conditions.
3. To implement the functions to perform the PSD computation.

5.2 Defining the conditions

Let say that you need a parameter `c0` to be able to compute the PSD. Then you will check wether this parameter is or not associated to the data in its attributes. This verification is coded in the "conversion_PSD.py" file of the HDF5_BLS project.

The nomenclature of your function will depend on the type you have given your data ("SPECTROMETER.Type" attributes), and will follow the following pattern:

```
def check_conversion_{type}(wrapper, path):
```

Three things are important here:

1. The name of the function should be `check_conversion_{type}`. where `{type}` is the type of the data as written in the "SPECTROMETER.Type" attribute. All the spaces and dashes are automatically replaced by underscores. Therefore if your type is "2-stage VIPA", your function will be `check_conversion_2_stage_vipa`.

2. This function takes two arguments: `wrapper` and `path`. The `wrapper` argument is the wrapper of the data that you want to treat, and the `path` argument is the path of the data in the wrapper. As the properties of the data are hierarchical, it is always a good practice to use the wrapper associated to the whole file and the path that from the whole file, leads to your data.

3. This function should return a boolean. If the conversion is possible, then the function should return `True`. If not, it should return `False`.

Should your function return `False`, then the GUI will automatically run the function `conversion_{type}` from the `conversion_ui.py` file, located in the HDF5_BLS_GUI folder.

5.3 The conversion user interface

This is where your implementation will really start. If you need calibration curves for defining the PSD, then it is where you will implement the function that will allow the user to select them. Same goes for any other parameter that you might need to define the PSD.

6 Contact

For questions or suggestions, please contact the maintainer at pierre.bouvet@meduniwien.ac.at.