plottr

Wolfgang Pfaff

Apr 05, 2022

CONTENTS:

1 Basic usage **3**

2 Working principles of plottr **5**
- 2.1 Nodes and Flowcharts 5
- 2.2 Data formats 5

3 Customization examples **7**
- 3.1 Creating a custom node 7
- 3.2 Creating a custom app 7

4 Indices and tables **9**

Todo: a quick description of what you can do with plottr, and a screenshot, or better a gif, showing it in action.

CHAPTER

ONE

BASIC USAGE

Content: a quick tour of how to look at different types of data (qcodes, ddh5, datadict directly using jupyter).

TBD.

CHAPTER

TWO

WORKING PRINCIPLES OF PLOTTR

This section documents how plottr works internally. It should enable anyone who's interested in writing their own components to get the basic ideas and find their way through the code.

2.1 Nodes and Flowcharts

2.2 Data formats

The main format we're using within plottr is the `DataDict`. While most of the actual numeric data will typically live in numpy arrays (or lists, or similar), they don't typically capture easily arbitrary metadata and relationships between arrays. Say, for example, we have some data `z` that depends on two other variables, `x` and `y`. This information has be stored somewhere, and numpy doesn't offer readily a solution here. There are various extensions, for example xarray or the MetaArray class. Those however typically have a grid format in mind, which we do not want to impose. Instead, we use a wrapper around the python dictionary that contains all the required meta information to infer the relevant relationships, and that uses numpy arrays internally to store the numeric data. Additionally we can story any other arbitrary meta data.

A DataDict container (a *dataset*) can contain multiple *data fields* (or variables), that have values and can contain their own meta information. Importantly, we distinct between independent fields (the *axes*) and dependent fields (the *data*).

Despite the naming, *axes* is not meant to imply that the *data* have to have a certain shape (but the degree to which this is true depends on the class used). A list of classes for different shapes of data can be found below.

The basic structure of data conceptually looks like this (we inherit from *dict*)

```
{
    'data_1' : {
        'axes' : ['ax1', 'ax2'],
        'unit' : 'some unit',
        'values' : [ ... ],
        '__meta__' : 'This is very important data',
        ...
    },
    'ax1' : {
        'axes' : [],
        'unit' : 'some other unit',
        'values' : [ ... ],
        ...,
    },
    'ax2' : {
        'axes' : [],
        'unit' : 'a third unit',
```

(continues on next page)

(continued from previous page)

```
        'values' : [ ... ],
        ...,
    },
    '__globalmeta__' : 'some information about this data set',
    '__moremeta__' : 1234,
    ...
}
```

In this case we have one dependent variable, `data_1`, that depends on two axes, `ax1` and `ax2`. This concept is restricted only in the following way:

- a dependent can depend on any number of independents
- an independent cannot depend on other fields itself
- any field that does not depend on another, is treated as an axis

Note that meta information is contained in entries whose keys start and end with double underscores. Both the DataDict itself, as well as each field can contain meta information.

In the most basic implementation, the only restriction on the data values is that they need to be contained in a sequence (typically as list, or numpy array), and that the length of all values in the data set (the number of *records*) must be equal. Note that this does not preclude nested sequences!

2.2.1 Relevant data classes

DataDictBase The main base class. Only checks for correct dependencies. Any requirements on data structure is left to the inheriting classes. The class contains methods for easy access to data and metadata.

DataDict The only requirement for valid data is that the number of records is the same for all data fields. Contains some tools for expansion of data.

MeshgridDataDict For data that lives on a grid (not necessarily regular).

For more information, see the API documentation.

CHAPTER

THREE

CUSTOMIZATION EXAMPLES

Here we want to show some ideas on how plottr could be customized. can include other files (scripts, etc) if necessary. Maybe this file should be moved into a subdirectory.

3.1 Creating a custom node

to be done.

3.2 Creating a custom app

to be done.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search