platelib Documentation

Release 0.1.5-alpha

Emil Dandanell Agerschou

May 10, 2019

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 Summary

platelib is an attempt to make common tasks when working with kinetic platereader, especially amyloid aggregation, data easy and compatible with Python.

1.2 Disclaimer

The state of this repository is one of very early development, the code is not elegant, there most likely are bugs so **Use with caution!**

1.3 Read the docs

A brief overview is given below, for more detailed information see the docs directory and subfolders, or even better have a look at the source :)

CHAPTER

TWO

INSTALLATION

2.1 Prerequisites

- python
- pip
- git (optional)

2.2 Download

Either download the source distribution directly or use:

```
git https://github.com/edager/platelib
```

2.3 Install

Go to directory where the platelib source file is (`platelib/dist/` if `git` was used) and run the following in the terminal:

```
pip install platelib-X.X.tar.gz
```

Where `X.X` should be replaced by the version that was downloaded.

2.4 Upgrade

Go to directory where the platelib source file is (`platelib/dist/` if `git` was used) and run the following in the terminal:

```
pip install --upgrade platelib-X.X.tar.gz
```

Where `X.X` should be replaced by the version that was downloaded.

2.5 Uninstall

Simply go to the terminal and run:

```
pip uninstall platelib
```

CHAPTER

THREE

SUPPORT

- **For questions related to usage:**
  - Stack Overflow using the tags `python` and `platelib`
- **For questions related to bugs or enhancements:**
  - GitHub using the issue system

CHAPTER

FOUR

COOKBOOK

4.1 Reading in data

The main functionality of `platelib` is the `read_plate` function that allows for reading in platereader data from kinetic experiments into a common framework namely into the `Plate_data` class. The following examples assumes for simplicity that `platelib` has been imported using:

```
from platelib import *
```

If an equal number of replicates per sample were prepared this can be specified (default is `3`):

```
p = read_plate('path/to/file', replicates=5)
```

It can be specified which direction the replicates were loaded onto the plate where `'hori'` (horizontal) means towards increasing numbers and `'vert'` is towards increasing letters (default is `'hori'`):

```
p = read_plate('path/to/file', rep_direction='vert')
```

NOTE that the replicates have to be next to each other!

Alternatively it can be specified which wells contains replicates:

```
p = read_plate('path/to/file', named_samples=[['B03', 'D07'], ['B02', 'E06', 'G12']]
```

Data from Tecan platereaders can be read in as (default is `'bmg'`):

```
p = read_plate('path/to/file', platereader='tecan')
```

NOTE that this functionality has not been fully tested yet!

As well as from BMG platereaders either where the data has prior been transposed `True` such that well data are in column format or in row format `False` (default is `True`):

```
p = read_plate('path/to/file', transposed=False)
```

Note that it's automatically detected if several measurements (*e.g.*) were made per time-point (see *Accessing data*)

The time unit can also be specified which as either `'seconds'`, `'minutes'`, `'hours'`, or `'days'` will carry along into indexes if exported and to unit of x-axis if plotted (default is `'hours'`):

```
p = read_plate('path/to/file',time_unit='days')
```

4.2 Accessing data

The `Plate_data` class allows for different ways of accessing the data

Through index:

```
p[1]
```

Through index slice:

```
p[::3]
```

Through well name:

```
p['B02']
```

Through list of well names:

```
p[['B02','C03','D04']]
```

Retrieved as a pandas.DataFrame with wellnames as column names and time points as index:

```
df = Plate_data.to_a_dataframe()
```

Or as a (C)omma (S)eperated (V)aribles file with the first line being (time unit +) well names and the first column are the time points:

```
Plate_data.to_a_csv('path/to/file.csv')
```

4.3 Plotting data

The data is plotted according to replicates, and subtitles can be added (default is *None*):

```
p.plot(titles=['condition 1', 'conditions 2'])
```

It can be specified whether all plots should have its own y-axis, whether all plots should have the same (default is *True*):

```
p.plot(sharey='False')
```

If several measurements were made per time-point it can be specified whether all measurements should be plotted or not (default is *True*):

```
p.plot(plot_multi='False')
```

CHAPTER

FIVE

PLATELIB

5.1 platelib package

5.1.1 Submodules

5.1.2 platelib.fitfun module

5.1.3 platelib.plateread module

5.1.4 Module contents

CHAPTER

SIX

CONTRIBUTE

Contributions are more than welcome, please raise an issue on the github page highlighting the bug/extension/compatibilities before doing a pull request.

6.1 More tools

Apart from the tools listed in *Installation* the following is needed:

- Unix-like system
- git
- pandoc

6.2 Building

You have made some wicked cool changes to the source code or the documentation that you want to share with the world, awesome!

Now there's just a few steps before they can be incorporated into the platelib master branch

6.2.1 Changing the version number

The versioning scheme of platelib should be done in reasonable accordance with the so called Semantic versioning where X.Y.Z should be read as MAJOR.MINOR.PATCH.

The version number has to be changed in the two files `setup.py` and `docs/source/conf.py`

6.2.2 Create new source distribution

Go to the docs folder and run:

```
./full_make.sh
```

If no errors occurred it can be uploaded to your local branch and a pull request can be made.

6.3 Planned improvements

This is as much a wish-list as literally planned improvements:

- Plotting
  - Plotting of data from several plates in some sensible way.
- Fitting
  - Local fitting of traces in plate
  - Global fitting of traces in plate
- Statistical analysis
  - Goodness-of-fit
  - Variance along traces, among replicates, and between conditions
- Python 3.X compatibility
- PyPI availability