PostPIC Documentation

Release 0.0.0

Stephan Kuschel

Jul 27, 2018

Contents

Contents:

CHAPTER 1

postpic

[![Build Status](https://travis-ci.org/skuschel/postpic.svg?branch=master){}](https://travis-ci.org/skuschel/postpic) [![PyPI version](https://badge.fury.io/py/postpic.png){}](http://badge.fury.io/py/postpic) [![Documentation Status](https://readthedocs.org/projects/postpic/badge/?version=latest){}](https://postpic.readthedocs.org/)

postpic is a open-source post processor for Particle-in-cell (PIC) simulations written in python. If you are doing PIC Simulations (likely for you PhD in physics. . .) you need tools to provide proper post-processing to create nice graphics from many GB of raw simulation output data – regardless of what simulation code you are using.

1.1 Idea of postpic

The basic idea of postpic is to calculate the plots you are interested in just from the basic data the Simulation provides. This data includes electric and magnetic fields and a tuple (weight, x, y, z, px, py, pz) for every macro-particle. Anything else you want to look at (for example a spectrum at your detector) should just be calculated from these values. This is exactly what postpic can do for you, and even more:

postpic has a unified interface for reading the required simulation data. If the simulation code of your choice is not supported by postic, this is the perfect opportunity to add a new datareader.

Additionally postpic can plot and label your plot automatically. This makes it easy to work with and avoids mistakes. Currently matplotlib is used for that but this is also extensible.

1.2 Installation

Postpic can be used with python2 and python3. However the usage of python3 is recommended.

Users should install the latest version directly from github using pip (python package manager):

pip install –user git+https://github.com/skuschel/postpic.git

The latest *release* is also available ib the python package index [pypi](https://pypi.python.org/pypi/postpic/), thus it can be installed by using the python package manager pip:

pip install postpic

Developers should clone the git repository (or their fork of it) and install it using

python setup.py develop –user

This command will link the current folder to global python scope, such that changing the code will immediately update the installed package.

After installing you should be able to import it into any python session using *import postpic*.

Postpic's main functions should work but there is still much work to do and lots of documentation missing. If postpic awakened your interest you are welcome to contribute. Even if your programming skills are limited there are various ways how to contribute and adopt postpic for your particular research. Read [CONTRIBUTING.md](../master/CONTRIBUTING.md).

1.3 Postpic in Science

If you use postpic for your research and present or publish results, please show your support for the postpic project and its [contributers](https://github.com/skuschel/postpic/graphs/contributors) by:

- Add a note in the acknowledgements section of your publication.
- Drop a line to one of the core developers including a link to your work to make them smile (there might be a public list in the future).

1.4 License

Postpic is released under GPLv3+. See [http://www.gnu.org/licenses/gpl-howto.html{]}(http://www.gnu.org/licenses/gpl-howto.html) for further information.

CHAPTER 2

Contributing to the PostPic code base

Any help contributing to the PostPic project ist greatly appreciated! Feel free to contact any of the developers or ask for help using the [Issues](https://github.com/skuschel/postpic/issues) Page.

2.1 Why me?

because you are using it!

2.2 How to contribute?

Reporting bugs or asking questions works with a GitHub account simply on the [Issues](https://github.com/skuschel/postpic/issues) page.

For any coding you need to be familiar with [git](http://git-scm.com/). Its a distributed version control system created by Linus Torvalds (and more importantly: he is also using it for maintaining the linux kernel). There is a nice introduction to git at try.github.io/, but in general you can follow the bootcamp section at [https://help.github.com/{}](https://help.github.com/) for your first steps.

One of the most comprehensive guides is probably [this book](http://git-scm.com/doc). Just start reading from the beginning. It is worth it!

The Workflow

Adding a feature is often triggered by the personal demand for it. Thats why production ready features should propagte to master as fast as possible. Everything on master is considered to be production ready. We follow the [github-flow](http://scottchacon.com/2011/08/31/github-flow.html) describing this very nicely.

In short:

> 0. [Fork](https://help.github.com/articles/fork-a-repo) the PostPic repo to your own GitHub account. 0. Clone from your fork to your local computer. 0. Create a branch whose name tells what you do. Something like *codexy-reader* or *fixwhatever*,... is a good choice. Do NOT call it *issue42*. Git history should be clearly readable without external information. If its somehow unspecific in the worst case call it *dev*

or even commit onto your *master* branch. 0. Implement a new feature/bugfix/documentation/whatever commit to your local repository. It is highly recommended that the new features will have test cases. 0. KEEP YOUR FORK UP TO DATE! Your fork is yours, only. So you have to update it to whatever happens in the main repository. To do so add the main repository as a second remote with

git remote add upstream git@github.com:skuschel/postpic.git

and pull from it regularly with

git pull –rebase upstream master

0. Make sure all tests are running smoothly (the *run-tests.py* script also involves pep8 style verification!) Run *run-tests.py* before EVERY commit! 0. push to your fork and create a [pull request](https://help.github.com/articles/using-pull-requests/) EARLY! Even if your feature or fix is not yet finished, create the pull request and start it with *WIP:* or *[WIP]* (work-in-progress) to show its not yet ready to merge in. But the pull request will

- trigger travis.ci to run the tests whenever you push
- show other people what you work on
- ensure early feedback on your work

Coding and general remaks

- Make sure, that the *run-tests.py* script exits without error on EVERY commit. To do so, it is HIGHLY RECOMMENDED to add the *pre-commit* script as the git pre-commit hook. For instructions see [pre-commit](../master/pre-commit).
- The Coding style is according to slightly simplified pep8 rules. This is included in the *run-tests.py* script. If that script runs without error, you should be good to ~~go~~ commit.
- Add the GPLv3+ licence notice on top of every new file. If you add a new file you are free to add your name as a author. This will let other people know that you are in charge if there is any trouble with the code. This is only useful if the file you provide adds functionality like a new datareader. Thats why the *__init__.py* files typically do not have a name written. In doubt, the git revision history will always show who added which line.

What to contribute?

Here is a list for your inspiration:

- Add Documentation and usage examples.
- Report bugs at the [Issues](https://github.com/skuschel/postpic/issues) page.
- Fix bugs from the [Issues](https://github.com/skuschel/postpic/issues) page.
- Add python docstrings to the codebase.
- Add new features.
- Add new datareader for additional file formats.
- Add test cases
- ...

CHAPTER 3

Getting started

The following script should just show an example how to get started using the postpic postprocessor. This script uses the dummy reader (thus auto generated random data). Thats why there is no input file needed to read the simulation data from.

```
#!/usr/bin/env python
#
# This file is part of postpic.
#
# postpic is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# postpic is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with postpic. If not, see <http://www.gnu.org/licenses/>.
#
# Copyright Stephan Kuschel 2015
#

def main():
    import numpy as np
    import postpic as pp

    # postpic will use matplotlib for plotting. Changing matplotlibs backend
    # to "Agg" makes it possible to save plots without a display attached.
    # This is necessary to run this example within the "run-tests" script
    # on travis-ci.
    import matplotlib; matplotlib.use('Agg')
```

(continues on next page)

(continued from previous page)

```
    # choose the dummy reader. This reader will create fake data for testing.
    pp.chooseCode('dummy')

    dr = pp.readDump(3e5)  # Dummyreader takes a float as argument, not a string.
    # set and create directory for pictures.
    savedir = '_examplepictures/'
    import os
    if not os.path.exists(savedir):
        os.mkdir(savedir)

    # initialze the plotter object.
    # project name will be prepended to all output names
    plotter = pp.plotting.plottercls(dr, outdir=savedir, autosave=True, project=
↪'simpleexample')

    # we will need a refrence to the MultiSpecies quite often
    from postpic.particles import MultiSpecies

    # create MultiSpecies Object for every particle species that exists.
    pas = [MultiSpecies(dr, s) for s in dr.listSpecies()]

    if True:
        # Plot Data from the FieldAnalyzer fa. This is very simple: every line␣
↪creates one plot
        plotter.plotField(dr.Ex())  # plot 0
        plotter.plotField(dr.Ey())  # plot 1
        plotter.plotField(dr.Ez())  # plot 2
        plotter.plotField(dr.energydensityEM())  # plot 3

        # Using the MultiSpecies requires an additional step:
        # 1) The MultiSpecies.createField method will be used to create a Field object
        # with choosen particle scalars on every axis
        # 2) Plot the Field object
        optargsh={'bins': [300,300]}
        for pa in pas:
            # create a Field object nd holding the number density
            nd = pa.createField('x', 'y', optargsh=optargsh, simextent=True)
            # plot the Field object nd
            plotter.plotField(nd, name='NumberDensity')    # plot 4
            # if you like to keep working with the just created number density
            # yourself, it will convert to an numpy array whenever needed:
            arr = np.asarray(nd)
            print('Shape of number density: {}'.format(arr.shape))

            # more advanced: create a field holding the total kinetic energy on grid
            ekin = pa.createField('x', 'y', weights='Ekin_MeV', optargsh=optargsh,␣
↪simextent=True)
            # The Field objectes can be used for calculations. Here we use this to
            # calculate the average kinetic energy on grid and plot
            plotter.plotField(ekin / nd, name='Avg Kin Energy (MeV)')  # plot 5
            # use optargsh to force lower resolution
            # plot number density
            plotter.plotField(pa.createField('x', 'y', optargsh=optargsh),␣
↪lineoutx=True, lineouty=True)  # plot 6
            # plot phase space
            plotter.plotField(pa.createField('x', 'p', optargsh=optargsh))  # plot 7
```

(continues on next page)

(continued from previous page)

```
            plotter.plotField(pa.createField('x', 'gamma', optargsh=optargsh))  #␣
↪plot 8
            plotter.plotField(pa.createField('x', 'beta', optargsh=optargsh))  # plot␣
↪9

            # same with high resolution
            plotter.plotField(pa.createField('x', 'y', optargsh={'bins': [1000,1000]}
↪))  # plot 10
            plotter.plotField(pa.createField('x', 'p', optargsh={'bins': [1000,1000]}
↪))  # plot 11

            # advanced: postpic has already defined a lot of particle scalars as Px,␣
↪Py, Pz, P, X, Y, Z, gamma, beta, Ekin, Ekin_MeV, Ekin_MeV_amu, ... but if needed␣
↪you can also define your own particle scalar on the fly.
            # In case its regularly used it should be added to postpic. If you dont␣
↪know how, just let us know about your own useful particle scalar by email or adding␣
↪an issue at
            # https://github.com/skuschel/postpic/issues

            # define your own particle scalar: p_r = sqrt(px**2 + py**2)/p
            plotter.plotField(pa.createField('sqrt(px**2 + py**2)/p', 'sqrt(x**2 +␣
↪y**2)', optargsh={'bins':[400,400]}))  # plot 12

            # however, since its unknown to the program, what quantities were␣
↪calculated the axis of plot 12 will only say "unknown"
            # this can be avoided in two ways:
            # 1st: define your own ScalarProperty(name, expr, unit):
            p_perp = pp.particles.ScalarProperty('sqrt(px**2 + py**2)/p', name='p_perp
↪', unit='kg*m/s')
            r_xy = pp.particles.ScalarProperty('sqrt(x**2 + y**2)', name='r_xy', unit=
↪'m')
            # this will create an identical plot, but correcly labled
            plotter.plotField(pa.createField(p_perp, r_xy, optargsh={'bins':[400,400]}
↪))  # plot 13
            # if those quantities are reused often, teach postip to recognize them␣
↪within the string expression:
            pp.particles.particle_scalars.add(p_perp)
            #pp.particles.scalars.add(r_xy)  # we cannot execute this line, because r_
↪xy is already predefinded
            plotter.plotField(pa.createField('p_perp', 'r_xy', optargsh={'bins':[400,
↪400]}))  # plot 14



            # choose particles by their properies
            # this has been the old interface, which would still work
            # def cf(ms):
            #     return ms('x') > 0.0  # only use particles with x > 0.0
            # cf.name = 'x>0.0'
            # pa.compress(cf)
            # nicer is the new filter function, which does exactly the same:
            pa.filter('x>0')
            # plot 15, compare with plot 10
            plotter.plotField(pa.createField('x', 'y', optargsh={'bins': [1000,1000]}
↪))
            # plot 16, compare with plot 12
            plotter.plotField(pa.createField('p_perp', 'r_xy', optargsh={'bins':[400,
↪400]}))
```

(continues on next page)

(continued from previous page)

```
            plotter.plotField(dr.divE())   # plot 13

if __name__=='__main__':
    main()
```

CHAPTER 4

postpic

4.1 postpic package

4.1.1 Subpackages

postpic.datareader package

Submodules

postpic.datareader.datareader module

postpic.datareader.dummy module

postpic.datareader.epochsdf module

postpic.datareader.vsimhdf5 module

Module contents

postpic.plotting package

Submodules

postpic.plotting.plotter_matplotlib module

Module contents

4.1.2 Submodules

4.1.3 postpic.cythonfunctions module

4.1.4 postpic.datahandling module

4.1.5 postpic.helper module

CHAPTER 5

Indices and tables

- genindex
- modindex
- search