Best practices of software development for computational research

Fraternali group retreat

Marius Kaušas

2018-09-01

Introduction

- The following material is based on the community-oriented primer on software development:

Github link:
`https://github.com/choderalab/software-development`

- The primer incorporates best practises for modern computational chemistry, but tools and scope of the primer are equally applicable to a general audience of computational researchers who interested in software development.

Introduction

- The purpose of this talk is not to go over the details of *Object-Orientated Programming* paradigm, but to give a general overview of software development.

- The programming language used here is Python, but I believe many topics will be important for any other language you decide to use in your work.

Philoshopy and scope of the primer

Objectives of this primer are the following:

- Identify key time saving concepts and provide recommendations that has been found most useful for developing computation code research.

- Avoid high "software entropy" or a situation where it is difficult to connect software components due to the difference in structure, coding philosophy, approach to documentation, testing and deployment strategy.

- Facilitate collaborations.

- Minimize barriers for new users. *msmbuilder* and *pyemma* provide a similar interface as *scikit-learn* of how models are fit to the data.

- Encourage interoperability. Ensure data flow between each of the software components to avoid constructing "script bridges" every time.

Overview of topics

- Licensing guidelines
- Structuring your project
- Version control
- Python coding conventions
- Unit testing
- Continuous integration
- Documentation
- Optimization
- Packagin and deployment

Licensing guidelines

- **Permissive:** Places no restriction on the use of the code in derivative works, including commercialization. Often only requires citation.

- **Copyleft:** Requires that derivative works also be open source.

Permissive license

A popular permissive license, MIT license, has the following form:

```
MIT License

Copyright (c) [year] [fullname]

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

Figure 1: *MIT License* terms.

There are a few more permissive licenses with minor modifications: *Apache License 2.0*, *BSD Lisence*.

Copyleft license

- It uses a copyright concept to require developers of derivative works **also** open source the derivative work.

- Can dissuade commercial software vendors from relying on your software.

Copyleft license

A popular copyleft license is GNU General Public License:

- Derivative works must not only be open-source, but must also carry a compatible license.

GNU General Public License:
`https://choosealicense.com/licenses/gpl-3.0/`

Other copyleft licenses include: *Lesser GNU Public License*, *GNU Affero General Public License*.

More material on lincenses

Choose a lincese: `https://choosealicense.com/`

A Quick Guide to Software Licensing for the Scientist-Programmer: `https://doi.org/10.1371/journal.pcbi.1002598`

The Legal Framework for Reproducible Scientific Research: Licensing and Copyright `https://doi.org/10.1109/MCSE.2009.19`

Structuring your project: cookiecutter

Cookiecutter - a command-line utility that creates projects from cookiecutters (or project templates). There are dozens of different flavours. `https://github.com/audreyr/cookiecutter`

- Specific tamplate used in the primer:
`https://github.com/choderalab/cookiecutter-compchem`

Structuring your project: cookiecutter-compchem

```
.
├── LICENSE                        <- License file
├── README.md                      <- Description of project which GitHub will render
├── appveyor.yml                   <- AppVeyor config file for Windows testing (if chosen)
├── {{repo_name}}
│   ├── __init__.py                <- Basic Python Package import file
│   ├── {{first_module_name}}.py   <- Starting packge module
│   ├── data                       <- Sample additional data (non-code) which can be packaged
│   │   ├── README.md
│   │   └── look_and_say.dat
│   ├── tests                      <- Unit test directory with sample tests
│   │   ├── __init__.py
│   │   └── test_{{repo_name}}.py
│   └── _version.py                <- Automatic version control with Versioneer
├── devtools                       <- Deployment, packaging, and CI helpers directory
│   ├── README.md
│   ├── conda-recipe               <- Conda build and deployment skeleton
│   │   ├── bld.bat
│   │   ├── build.sh
│   │   └── meta.yaml
│   └── travis-ci
│       └── install.sh
├── docs                           <- Documentation template folder with many settings already filled in
│   ├── Makefile
│   ├── README.md                  <- Instructions on how to build the docs
│   ├── _static
│   ├── _templates
│   ├── conf.py
│   ├── index.rst
│   └── make.bat
├── setup.cfg                      <- Near-master config file to make house INI-like settings for Coverage,
├── setup.py                       <- Your package's setup file for installing with additional options that
├── versioneer.py                  <- Automatic version control with Versioneer
├── .github                        <- GitHub hooks for user contrubtion and pull request guides
│   ├── CONTRIBUTING.md
│   └── PULL_REQUEST_TEMPLATE.md
├── .codecov.yml                   <- Codecov config to help reduce its verbosity to more reasonable levels
├── .gitignore                     <- Stock helper file telling git what file name patterns to ignore when a
└── .travis.yml                    <- Travis-CI config file for Linux and OSX testing
```

Figure 2: A *cookiecutter* template.

Version control

Git system version control allows:

- Tracking version of files.
- Connecting and sharing code between computers using repositories .
- Easy mirroring and branching of code bases.

Neat tutorial on Git: `http://rogerdudler.github.io/git-guide/`

Workflow comparisons: `https://www.atlassian.com/git/tutorials/comparing-workflows`

Python coding conventions or PEP8

- Keep your code simple and clean. *(Ugh! easy to say...)*

- Write robust and extensible code.

- Use a moder environment, such as Python 3. The Python 2 gets a terminated support in 2020. `http://python3statement.org/`

- Use an editor, such as PyCharm or Atom.

- Adhere to a coding convetions, and stick with it. `https://www.python.org/dev/peps/pep-0008/`. PyCharm and Atom have automatic PEP8 detection.

- Document your code: `https://www.python.org/dev/peps/pep-0257/`

Unit testing

Few programs that can be used to automate the testing of the code: *pytest*, *nose2*, *unittest*.

Unit testing

An example of a simple test:

```
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5
```

To execute it:

```
$ pytest
========================== test session starts ==========================
platform linux -- Python 3.x.y, pytest-3.x.y, py-1.x.y, pluggy-0.x.y
rootdir: $REGENDOC_TMPDIR, inifile:
collected 1 item

test_sample.py F                                                  [100%]

=================================== FAILURES ===================================
_____________________________ test_answer ______________________________

    def test_answer():
>       assert inc(3) == 5
E       assert 4 == 5
E        +  where 4 = inc(3)

test_sample.py:6: AssertionError
========================= 1 failed in 0.12 seconds =========================
```

Figure 3: A *pytest* example.

Unit testing

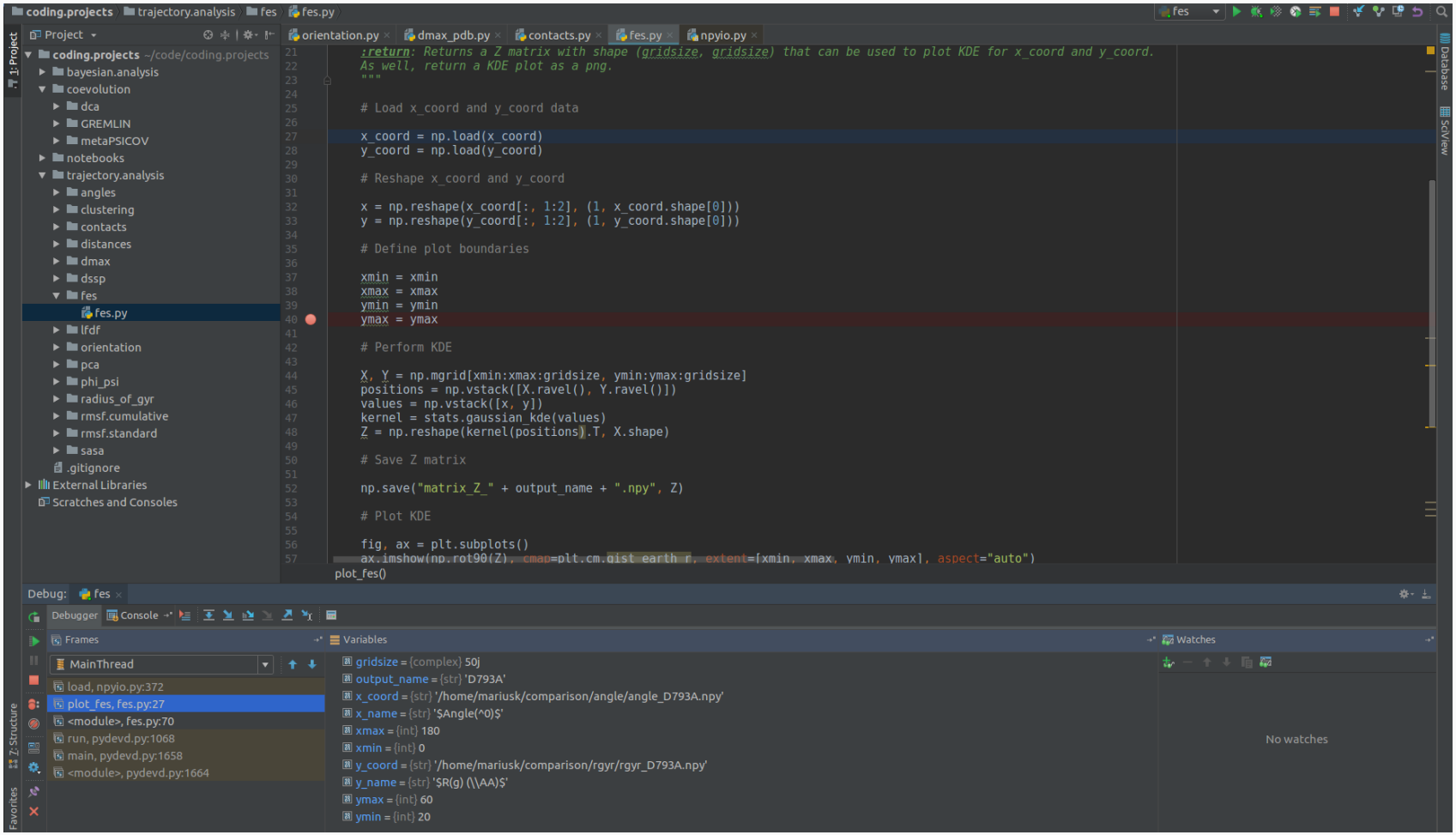

Figure 4: A *PyCharm* debugging example.

Continuous Integration

Continuous Integration (CI):

- A practice of merging in small code changes frequently - rather than merging in a large change at the end of a development cycle.

- Travis CI: language agnostic, free for open source, integration with GitHub, platform independent. `https://travis-ci.org/`

Travis CI works by cloning your GitHub repository to a new virtual environment, carries out build and testing of the code. If the tests passed, Travis CI can deploy your code to a web server or application host.

Documentation

- **GitHub Wiki:**
`https://help.github.com/articles/about-github-wikis/`

- **Sphinx:** `http://www.sphinx-doc.org/en/master/`. Various outputs (HTML, Latex, ePub, plain text), autogenerates API from source code, supports mathematic notation, source code highlighting and GitHub integration.

Documentation

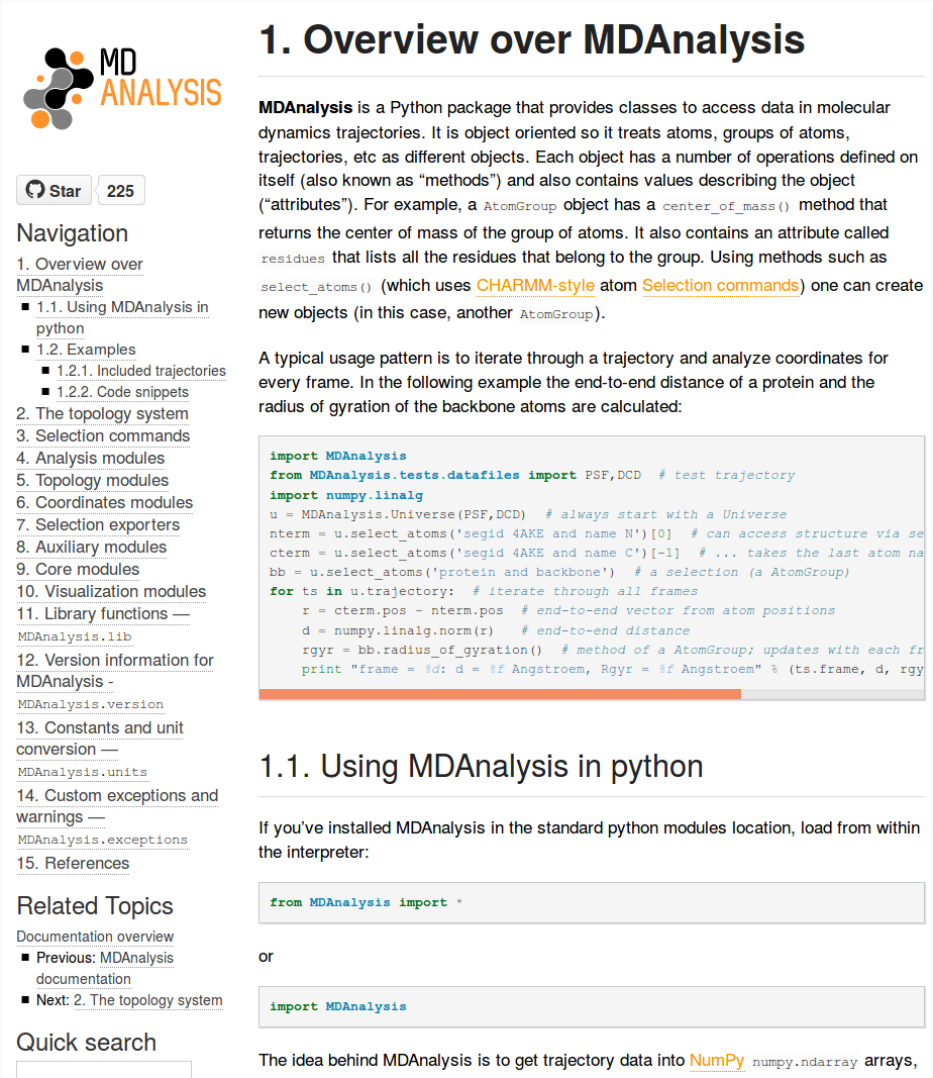

Figure 5: *MDAnalysis* documentation powered by *Sphinx*.

Optimization

Package recommendations:

- **Numpy:** powerful linear algebra, Fourier transform, random number generators, multi-dimensional arrays, sophisticated broadcasting, tools for integrating C/C++ and Fortran code.

- **ThensorFlow/Theano:** allows to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. GPU and TPU support.

-**PyCUDA/PyOpenCL:** Python way to a world of GPUs.

-**Cython:** write C extensions for Python.

Testing your optimization

cProfiler: `https://docs.python.org/2/library/profile.html`

line profiler: `https://github.com/rkern/line_profiler`

```
File: pystone.py
Function: Proc2 at line 149
Total time: 0.606656 s

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   149                                           @profile
   150                                           def Proc2(IntParIO):
   151     50000        82003      1.6     13.5      IntLoc = IntParIO + 10
   152     50000        63162      1.3     10.4      while 1:
   153     50000        69065      1.4     11.4          if Char1Glob == 'A':
   154     50000        66354      1.3     10.9              IntLoc = IntLoc - 1
   155     50000        67263      1.3     11.1              IntParIO = IntLoc - IntGlob
   156     50000        65494      1.3     10.8              EnumLoc = Ident1
   157     50000        68001      1.4     11.2          if EnumLoc == Ident1:
   158     50000        63739      1.3     10.5              break
   159     50000        61575      1.2     10.1      return IntParIO
```

Figure 6: A *line profiler* example

Packaging Python projects

Python-only code through PyPi: `https://packaging.python.org/tutorials/packaging-projects/`.

Conda-forge: `https://conda-forge.org/`. A community standard way to automate release builds and uploads to the Anaconda cloud for community-developed codes.

More about Open Source software

Science code manifest: `http://sciencecodemanifesto.org/`

Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research. `http://doi.org/10.5334/jors.ay`

Happy coding! Fin.