PlasmaPy: an open source community-developed Python package for plasma physics

The PlasmaPy Community, Nicholas A. Murphy,[1] Andrew J. Leonard,[2] Dominik Stańczak, Pawel M. Kozlowski,[3] Samuel J. Langendorf,[4] Colby C. Haggerty,[5] Jasper P. Beckers, Stuart J. Mumford,[6] Tulasi N. Parashar,[7] and Yi-Min Huang[8]

[1]Harvard-Smithsonian Center for Astrophysics, [2]Aperio Software, [3]West Virginia University, [4]Los Alamos National Laboratory, [5]University of Chicago, [6]University of Sheffield, [7]University of Delaware, [8]Princeton University

April 30, 2018

plasmapy

Introduction

- In recent years, researchers in several different subfields of physics and astronomy have collaboratively developed core Python packages such as Astropy[1] and SunPy[2]
- These packages provide core functionality, common frameworks for data analysis and visualization, and educational tools
- A similar open source package for plasma physics would greatly benefit our field
- **We are developing PlasmaPy: a community-developed and community-driven open source core Python package for plasma physics**

[1] Astropy Collaboration (2018)
[2] SunPy Community (2015)

Current status of scientific programming in plasma physics

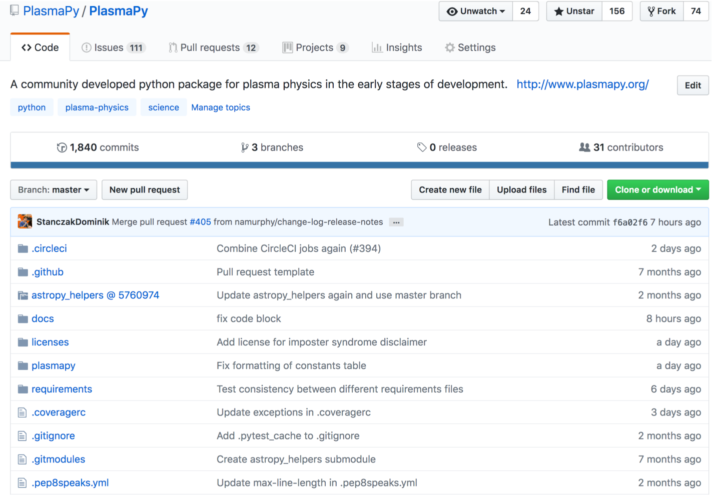
- Major codes often use low-level languages such as Fortran
- Programmers are often self-taught
- Code is often difficult to read
- Compiling and installing codes is difficult and time-consuming
- Different codes lack interoperability
- Documentation is usually inadequate
- Access to major codes is often restricted in some way
- It is somewhat unusual to share code
- Many versions of software do essentially the same thing
- Research is difficult to reproduce

There is a considerable need for an open, general purpose shared software package for plasma physics that uses modern best practices for scientific programming.

Why choose Python?

- Free and open source
- High-level, interpreted language
- Programming style emphasizes readability
- Can "glue" together software written in different languages
- Can reach near-compiled speeds using packages such as Numba and Cython, or by calling compiled routines
- Well-developed numerical and scientific analysis packages
- Active user community
- Can learn from and collaborate with ongoing highly successful projects such as Astropy and SunPy
- Will help students learn programming skills that will be useful in finding employment outside of plasma physics

PlasmaPy is an open source Python 3.6+ package for plasma physics in the early stages of development



The long-term goal of the PlasmaPy community is to facilitate a fully open source Python ecosystem for plasma physics.

PlasmaPy's first development release is version `0.1.0`

- Version `0.1.0` is a prototype and a preview, and not yet recommended for production work
  - Significant changes to the application programming interface (API) will occur during the first few development releases
- Rather, version `0.1.0` serves as an invitation to plasma students and scientists to collaboratively develop a community-wide shared software package
- PlasmaPy is available on the Python Package index (PyPI) and may be installed into an existing scientific Python 3.6 environment[3] by running

```
pip install plasmapy
```

[3] We recommend using an Anaconda Python environment.

PlasmaPy is open source for open and reproducible science

- Some software packages in plasma physics are described as open source, but do not meet the definition set by the Open Source Initiative (OSI) or use an OSI-approved license
- PlasmaPy is under the permissive **BSD 3-clause license** with OSI-approved language to protect against software patents
  - Using a permissive license maximizes compatibility with software under different licenses
  - Permissively licensed code may be incorporated into both proprietary and copyleft software
- Creative works besides source code are usually under Creative Commons licenses
  - The **CC BY 4.0** license allows works to be shared and adapted as long as attribution is given to the original work
  - The **CC BY-SA 4.0** license allows works to be shared and adapted with attribution if derivative works are shared under the same license

PlasmaPy is using best practices for scientific computing[4] to ensure that code is easy-to-use and maintainable

- Simple and intuitive API
- Readable and consistent style (PEP 8 standard)
- Embed documentation in code
- Use modular, object-oriented programming
- Version control with git with useful commit messages
- Avoid prematurely optimizing code
- Use semantic versioning
- Continuous integration testing and test coverage checks
- Issue tracking and code review using GitHub
- Adopt a code of conduct and work toward a welcoming and inclusive community

[4] Many of these practices are described by Wilson et al., "Best Practices for Scientific Computing," PLOS Biology **12**, e1001745 (2014).

Organizational development in PlasmaPy's first year

- Set up communication channels
  - Matrix/Gitter channel for real-time text-based chats
  - Biweekly video conferences
  - Email list
- Chose a license and added protections against software patents
- Wrote PlasmaPy's vision statement
- Adopted a code of conduct
- Developed a guide for new contributors
- Appointed the Coordinating Committee
- Started the PlasmaPy Enhancement Proposals repository
- Started a development roadmap

PlasmaPy is well-documented and well-tested

- Each pull request undergoes continuous integration testing with Travis CI and AppVeyer
- Automated test coverage checks with Coveralls show which lines of code are not covered by tests
- PlasmaPy's online documentation is hosted on Read the Docs after being built using Sphinx
  - We use the numpydoc docstring format
- CircleCI test builds the documentation for each pull request
- PlasmaPy's website was created using Nikola and is hosted using GitHub Pages
- Created initial website using Nikola and GitHub Pages
- PlasmaPy's entire code development history is openly available on our GitHub repository

Code development began in earnest in April 2017

- Created `atomic` and `constants` subpackages to access physical data
- Developed `physics` subpackage to calculate plasma parameters, including dielectric tensor components
- Created `physics.transport` module to calculate transport/collision parameters
- Created `mathematics` subpackage for commonly used analytical functions
- Started a `diagnostics` subpackage with initial functionality for analyzing Langmuir probe data
- Developed prototype base classes in `classes` subpackage, including particle pusher functionality
- Created the `utils` subpackage with helper functionality and custom exceptions
- Began using test/import functionality from `astropy-helpers`

PlasmaPy uses the astropy.units package for units

This package creates Quantity objects with attached units.

```
>>> from astropy import units as u
>>> distance = 44 * u.imperial.mile
>>> time = 30 * u.minute
>>> distance / time
<Quantity 88.0 mi / h>
>>> (distance/time).to(u.m/u.s)
<Quantity 39.33952 m / s>
>>> (1.21 * u.GW).cgs
<Quantity 1.21e+16 erg / s>
>>> 2 * u.m / u.s + 4 * u.m / u.s ** 2
UnitConversionError: Can only apply 'add' function to quantities
with compatible dimensions
```

Built-in equivalencies can handle non-standard unit conversions commonly used in plasma physics:[5]

```
>>> kT = 1.2 * units.keV
>>> kT.to(u.K, equivalencies=u.temperature_energy())
<Quantity 13925426.47248121 K>
```

[5] Code inside PlasmaPy uses SI units to avoid confusion and for consistency with established international practices.

The atomic subpackage provides functional and object-oriented interfaces to particle data

Instances of the Particle class may be used to represent individual atoms, ions, or elementary particles.

```
>>> from plasmapy.atomic import *

>>> alpha = Particle("He-4++")
>>> alpha.mass
<Quantity 6.64465709e-27 kg>
>>> electron = Particle("e-")
>>> electron.charge
<Quantity -1.60217662e-19 C>
>>> electron.is_category(require={"lepton", "fermion"})
True
>>> ~electron  # find antiparticle with invert operator
Particle("e+")
```

We can calculate the released energy from a nuclear reaction.

```
>>> nuclear_reaction_energy("D + T -> alpha + n").to('MeV')
<Quantity 17.58932778 MeV>
```

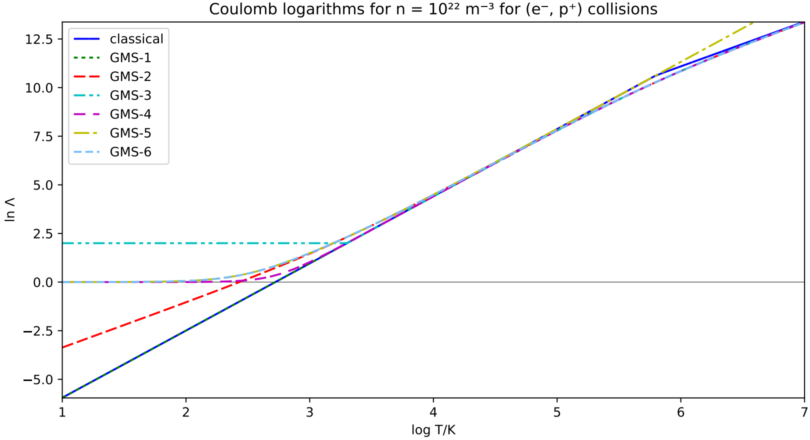
The `physics` subpackage provides functions to calculate plasma parameters and dielectric tensor components

```
>>> from plasmapy.physics import *

>>> Debye_length(n_e = 1e15 * u.m ** -3, T_e = 6e6 * u.K)
<Quantity 0.00534541 m>

>>> inertial_length(5e19 * u.m ** -3, particle='D+')
<Quantity 0.04553085 m>

>>> upper_hybrid_frequency(0.2 * u.T, n_e = 5e19 * u.m ** -3)
<Quantity 4.00459419e+11 rad / s>

>>> B = 2 * u.T
>>> species = ['e-', 'D+']
>>> n = [1e18 * u.m ** -3, 1e18 * u.m ** -3]
>>> omega = 3.7e9 * (2 * pi) * (u.rad / u.s)
>>> L, R, P = cold_plasma_permittivity_LRP(B, species, n, omega)
>>> L
<Quantity 0.63333549>
>>> R
<Quantity 1.41512254>
>>> P
<Quantity -4.8903104>
```

The `transport` subpackage provides functions to calculate collision parameters and transport coefficients

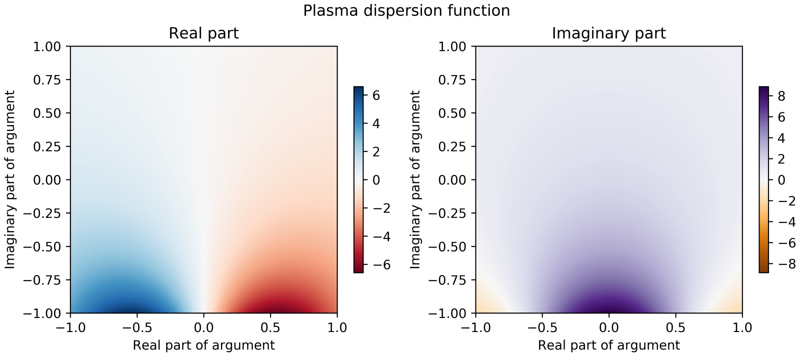
```
>>> from plasmapy.transport import *
>>> T = 1 * u.MK
>>> n = 5e15 * u.m ** -3
>>> particles = ('e-', 'p+')
>>> collision_frequency(T, n, particles)
<Quantity 443.02775451 Hz>
>>> coupling_parameter(T, n, particles)
<Quantity 4.60608476e-06>

>>> T_e, n_e = 0.6 * u.keV, 1e16 * u.cm ** -3
>>> T_p, n_p = 0.8 * u.keV, 1e16 * u.cm ** -3
>>> braginskii = ClassicalTransport(T_e, n_e, T_p, n_p, 'p+')
>>> braginskii.ion_thermal_conductivity()
<Quantity 132961.01785222 W / (K m)>
>>> braginskii.electron_viscosity() # Eq 2.25-2.27 in Braginskii (1965)
<Quantity [0.02734206, 0.02733305, 0.02733305, 0., 0.] Pa s>
```

PlasmaPy has multiple methods for calculating Coulomb logarithms over a wide range of plasma parameters



The `Coulomb_logarithm` function includes the classical calculation and multiple methods from Gericke, Murillo, & Schlanges (2002). A `CouplingWarning` is issued when strong coupling effects may be important but are not accounted for.

The `mathematics` subpackage contains analytic functions that are commonly used by plasma physicists



The plasma dispersion function

$$Z(\zeta) = \pi^{-1/2} \int_{-\infty}^{\infty} \frac{\mathrm{e}^{-x^2}}{x - \zeta} \mathrm{d}x$$

may be calculated using `plasmapy.mathematics.plasma_dispersion_func`. This function is tested against results tabulated by Fried & Conte (1961).

PlasmaPy code development roadmap

- Create a `Plasma` metaclass as base data structure
- Add fluid and particle simulation capabilities
- Turbulence analysis tools
- Develop tools to analyze and interpret plasma diagnostics
- Implement an equilibrium solver
- Develop tools to analyze 3D magnetic topology
- Implement a dispersion relation solver
- Query tools for atomic and other databases

If there is functionality that you would like in PlasmaPy, we invite you to raise an issue in our GitHub repository.

What does PlasmaPy need to succeed?

- Open development
  - Low barrier to entry
  - Actively inviting new contributors
  - Open data policies for major experiments
- A welcoming and inclusive environment
  - Provide a culture of appreciation for contributors to PlasmaPy
  - Adopt a code of conduct
- A sustainable funding model[6]
  - Astropy development is mostly a volunteer, grassroots effort
  - Most work on Astropy has been done by graduate students and postdocs, with little direct funding support
  - There is a need for funding agencies and large institutions to support open development of general purpose software

[6] This issue is described thoroughly by D. Muna et al. in *The Astropy Problem* (arXiv:1610.03159)

Summary

- **We are developing PlasmaPy: a community-developed and community-driven open source core Python package for plasma physics**
  - Version 0.1.0 is available on PyPI and may be installed into a scientific Python 3.6 environment by running `pip install plasmapy`
- PlasmaPy is a collaboration among laboratory, heliospheric, space, and astrophysical plasma physicists, and is building bridges among these communities
- New contributors are welcome and can become involved by:
  - Joining our email list and conversation on Matrix/Gitter
  - Raising issues on GitHub with new ideas for code development
  - Contributing code, especially issues labeled **Good first contribution**
  - Contributing documentation
  - Becoming an early adopter and providing constructive feedback

PlasmaPy Links

- PlasmaPy's **GitHub repository** is:

  https://github.com/PlasmaPy/plasmapy

- PlasmaPy's **online documentation** is at:

  http://docs.plasmapy.org/

- We are developing our **webpage** at:

  http://www.plasmapy.org/

- Our **Matrix and Gitter channels** for real-time text-based communication are at:

  https://riot.im/app/#/room/#plasmapy:matrix.org
  https://gitter.im/PlasmaPy/Lobby

- Sign up for the **PlasmaPy email list** at:

  https://groups.google.com/d/forum/plasmapy