PlasmaPy Vision Statement

This document contains the original vision statement for PlasmaPy. This vision statement has been superseded by PlasmaPy's online documentation (including its contributor guide), PlasmaPy Enhancement Proposals (PLEPs), and PlasmaPy's website.

About PlasmaPy

PlasmaPy is a community-developed and community-driven free and open source Python package that provides common functionality required for plasma physics in a single, reliable codebase.

Motivation

In recent years, researchers in many different scientific disciplines have worked together to develop core Python packages such as Astropy, SunPy, and SpacePy. These packages provide core functionality, common frameworks for data visualization and analysis, and educational tools for their respective scientific disciplines. We believe that a similar cooperatively developed package for plasma physics will greatly benefit our field. In this document, we lay out our vision for PlasmaPy: a community-developed and community-driven open source core Python software package for plasma physics.

There is considerable need in plasma physics for open, general purpose software framework using modern best practices for scientific computing. As most scientific programmers are largely self-taught, software often does not take advantage of these practices and is instead written in a rush to produce results for the next research paper. The resulting code is often difficult to read and maintain, the documentation is usually inadequate, and tests are typically implemented late in the development process if at all. Legacy code is often written in low level languages such as Fortran, which typically makes compiling and installing packages difficult and frustrating, especially if it calls external libraries. It is also unusual to share code, and access to major software is often restricted in some way, resulting in many different programs and packages which do essentially the same thing but with little or no interoperability. These factors lead to research that is difficult to reproduce, and present a significant barrier to entry for new users.

The plasma physics community is slowly moving in the open source direction. Several different types of packages and software have been released under open source licenses, including the UCLA PIC codes, PICCANTE, EPOCH, VPIC, PIConGPU, WARP, the FLASH framework, Athena, and PENCIL. These projects are built as individual packages, are written in different programming languages, and often have many dependencies on specific packages. Python packages such as Astropy, SunPy, and SpacePy have had notable success providing open source alternatives to legacy code in related fields. We are grateful to these communities for their hard work, and hope to build upon their accomplishments for the field of plasma physics.

An end user might not always be interested in a complicated powerpack to perform one specific task on supercomputers. She might also be interested in performing some basic plasma physics calculations, running small desktop scale simulations to test preliminary ideas (e.g., 1D MHD/PIC or test particles), or even comparing data from two different sources (simulations vs. spacecraft). Such tasks require a central platform. This is where PlasmaPy comes in.

PlasmaPy Community Code of Conduct

Please see PlasmaPy's code of conduct.

Organizational Structure

The Coordinating Committee (CC) will oversee the PlasmaPy project and code development. The CC will ensure that roles are being filled, facilitate community-wide communication, coordinate and delegate tasks, manage the project repository, oversee the code review process, regulate intercompatibility between different subpackages, seek funding mechanisms, facilitate compromises and cooperation, enforce the code of conduct, and foster a culture of appreciation.

The Community Engagement Committee (CEC) will be responsible for organizing conferences, trainings, and workshops; maintaining and moderating social media groups and accounts; overseeing PlasmaPy's website; and communicating with the PlasmaPy and plasma physics communities. The CEC will facilitate partnerships with groups such as Software Carpentry.

Each subpackage will have lead and deputy coordinators who will guide and oversee the development of that subpackage.

The Accessibility Coordinator will work to ensure that the PlasmaPy codebase, documentation, and practices are accessible to disabled students and scientists. Additional roles include the Webpage Maintainer, the Release Coordinator, and the Testing Coordinator.

The work undertaken by each of these groups and coordinators should be done openly and transparently, except where confidentiality is needed. We will strive to have multiple subfields from plasma physics in each committee. Major decisions should ideally be made by general consensus among the PlasmaPy community, but when consensus is not possible then the committees may decide via majority vote. Much of this section is following the organizational structure of Astropy.

Development Procedure

The initial developers of PlasmaPy will create a flexible development roadmap that outlines and prioritizes subpackages to be developed. The developers will survey existing open source Python packages in plasma physics. Priority will be given to determining how data will be stored and structured. Developers will break up into small groups to work on different subpackages. These small groups will communicate regularly and work towards interoperability and common coding practices.

Because Python is new to many plasma physicists, community engagement is vital. The CEC will arrange occasional informal trainings early in the project that are directed towards the initial developers.

New code and edits should be submitted as a pull request to the development branch of the PlasmaPy repository on GitHub. The pull request will undergo a code review by the subpackage maintainers and/or the CC, who will provide suggestions on how the contributor may update the pull request. Subpackage maintainers will generally be responsible for deciding on pull requests with minor changes, while pull requests with major changes should be decided jointly by the subpackage maintainers and the CC. The CC and CEC will develop a friendly guide on how users may contribute new code to PlasmaPy.

New code should conform to the PEP 8 style guide for Python code and the established coding style within PlasmaPy. New code should be submitted with documentation and tests. Documentation should be written primarily in docstrings and follow the numpydoc documentation style guide. Every new module, class and function should have an appropriate docstring. The documentation should describe the

interface and the purpose for the method, but generally not the implementation. The code itself should be readable enough to be able to explain how it works. Documentation should be updated when the code is edited. The tests should cover new functionality (especially methods with complex logic), but the tests should also be readable and easy to maintain. Existing tests should be updated when necessary [e.g., during the initial development of a new feature when the application program interface (API) is not yet stable], but with caution since this may imply loss of backwards compatibility.

Members of the PlasmaPy community may submit PlasmaPy Enhancement Proposals (PLEPs) to suggest changes such as major reorganization of a subpackage, creation of a new subpackage, non-backwards compatible changes to a stable package, or significant changes to policies and procedures related to the organization of this project. The issues list on GitHub will generally be more appropriate for changes that do not require community discussion. The CC shall maintain a GitHub repository of PLEPs. PLEPs will be made openly available for community discussion and transparency for a period of at least four weeks, during which time the proposal may be updated and revised by the proposers. The CC shall approve or decline these proposals after seeking community input. The rationale behind the decision and a summary of the community discussion shall be recorded along with the PLEP.

Programming Guidelines

Choice of Languages

PlasmaPy shall be written using Python 3. PlasmaPy shall initially guarantee compatibility with Python 3.6 and above. Python 3 is continually growing, so we will proceed on the general principle that future updates to PlasmaPy remain compatible with releases of Python that are up to two years old. Python 2.7 and below will not be supported as these versions will no longer be updated past 2020. The core package will initially be written solely in Python.

Code readability is more important than optimization, except when performance is critical. Code should be optimized only after getting it to work, and primarily for where there is a performance bottleneck. Performance-critical parts of the core package will preferably be written using Numba to achieve compiled speeds while maintaining the significant advantages of using a high level language.

Versioning

PlasmaPy will use Semantic Versioning. Releases will be given version numbers of the form `MAJOR.MINOR.PATCH`, where `MAJOR`, `MINOR`, and `PATCH` are nonnegative integers. Starting with version 1.0, `MAJOR` will be incremented when backwards incompatible changes are made, `MINOR` will be incremented when new backwards-compatible functionality is added, and `PATCH` will be incremented when backwards-compatible bug fixes are made.

Development releases will have `MAJOR` equal to zero and start at version 0.1. The API should not be considered stable during the development phase. PlasmaPy will release version 1.0 once it has a stable public API that users are depending on for production code.

All releases will be provided with release notes and change log entries, and a table will be provided that describes the stability of the public API for each PlasmaPy subpackage.

Dependencies

Dependencies have the advantage of providing capabilities that will enhance PlasmaPy and speed up its development, but the disadvantage that they can make manual installation more difficult and potentially frustrating. Package managers such as Anaconda and Homebrew greatly simplify installation of Python packages, but there will be situations where manual installation is necessary (e.g., on some supercomputers without package managers). The core package should be able to be imported using a minimal number of packages (e.g., NumPy, SciPy, and matplotlib) without getting an import error. Additional packages may be included as dependencies of the core package if there is a strong need for it, and if these packages are easily installed with currently available package managers. Subpackages may use additional dependencies when appropriate.

Affiliated Packages

We will follow the practice of Astropy by having a core package and affiliated packages. The core package will contain common tools and base functionality that most plasma physicists will need. The affiliated packages contained in separate repositories will include more specialized functionality that is needed for subfields of plasma physics. This approach will reduce the likelihood of scope creep for the core package while maintaining avenues for broader development.

Units

Multiple sets of units are used by plasma physicists. There exist some peculiarities with how units are used within plasma physics, such as how an electron-volt is typically used as a measurement of temperature. Code will be most readable and maintainable if written assuming a particular set of units, but there should be enough flexibility for people in different subfields to choose their preferred set of units. As the generally most common accepted international standard, SI base units will be utilized. We will use an existing Python module (e.g., `astropy.units` or pint) to assign units to variables and allow straightforward conversion between different systems of units.