AGU100 ADVANCING EARTH AND SPACE SCIENCE

JGR

Journal of Geophysical Research: Space Physics

TECHNICAL REPORTS: METHODS

10.1029/2018JA025877

Key Points:
- A wealth of Python analysis tools exists to support space physics research
- Open-source tools aid reproducible science
- Collaborative programming is essential for analysis tools to keep pace with scientific progress

Correspondence to:
A. G. Burrell,
angeline.burrell@nrl.navy.mil

Citation:
Burrell, A. G., Halford, A., Klenzing, J., Stoneback, R. A., Morley, S. K., Annex, A. M., et al. (2018). Snakes on a spaceship—An overview of Python in heliophysics. *Journal of Geophysical Research: Space Physics*, *123*, 10,384–10,402. https://doi.org/10.1029/2018JA025877

Received 10 JUL 2018
Accepted 11 NOV 2018
Accepted article online 16 NOV 2018
Published online 14 DEC 2018

Snakes on a Spaceship—An Overview of Python in Heliophysics

A. G. Burrell[1,2], A. Halford[3], J. Klenzing[4], R. A. Stoneback[2], S. K. Morley[5], A. M. Annex[6], K. M. Laundal[7], A. C. Kellerman[8], D. Stansby[9], and J. Ma[1,10]

[1]Space Science Division, U.S. Naval Research Laboratory, Washington, DC, USA, [2]William B. Hanson Center for Space Sciences, The University of Texas at Dallas, Richardson, TX, USA, [3]Space Sciences Department, The Aerospace Corporation, Chantilly, VA, USA, [4]ITM Physics Laboratory/Code 675, Goddard Space Flight Center, Greenbelt, MD, USA, [5]Space Science and Applications, Los Alamos National Laboratory, Los Alamos, NM, USA, [6]Department of Earth and Planetary Sciences, Johns Hopkins University, Baltimore, MD, USA, [7]Birkeland Centre for Space Science, University in Bergen, Bergen, Norway, [8]Department of Earth Planetary and Space Sciences, University of California, Los Angeles, CA, USA, [9]Imperial College London, London, UK, [10]Facebook AI Research, Menlo Park, CA, USA

Abstract Computational analysis has become ubiquitous within the heliophysics community. However, community standards for peer review of codes and analysis have lagged behind these developments. This absence has contributed to the reproducibility crisis, where inadequate analysis descriptions and loss of scientific data have made scientific studies difficult or impossible to replicate. The heliophysics community has responded to this challenge by expressing a desire for a more open, collaborative set of analysis tools. This article summarizes the current state of these efforts and presents an overview of many of the existing Python heliophysics tools. It also outlines the challenges facing community members who are working toward the goal of an open, collaborative, Python heliophysics toolkit and presents guidelines that can ease the transition from individualistic data analysis practices to an accountable, communalistic environment.

Plain Language Summary As computers have become more powerful and better at solving complex mathematical equations, space scientists have relied more and more on computational tools. Community standards for peer review of computer codes and similar analysis tools have not kept pace with the development of these technologies. This lag in community accountability has contributed to a crisis in the scientific literature, where it has been hard or even impossible to verify past studies. Space scientists have responded to this challenge with a desire for open, shared analysis tools. This article summarizes the current state of these efforts and presents an overview of many of the existing Python analysis tools used in space physics. It also outlines the challenges facing scientists who are working toward the goal of an open, shared, Python space science toolkit and presents guidelines that can ease the transition from private to public data analysis practices.

1. Introduction

The proliferation of observed and modeled data within the field of heliophysics, a field that encompasses solar, magnetospheric, and upper atmospheric studies within the solar system, has vastly expanded the possibilities for science investigation. These ever-expanding archives (e.g., the Coupling, Energetics and Dynamics of Atmospheric Regions [CEDAR] Madrigal database, the National Aeronautics and Space Administration [NASA] Coordinated Data Analysis Web (CDAWeb), and the Near Earth Space data infrastructure for e-science (ESPAS)) are accessible and searchable but (appropriately) do not provide many tools for identifying specific case studies or performing data analysis. This absence has contributed to the reproducibility crisis, where the results of scientific studies have been shown to be difficult or even impossible to replicate (Gil et al., 2016; Peng, 2011). Additionally, as the heliophysics community moves toward solving more complex interdisciplinary problems through data science techniques, a common infrastructure capable of handling diverse data sets is required (McGranaghan et al., 2017).

The challenges presented by large, distinct data sets and unreproducible results may be surmounted using currently available tools and techniques. Other scientific communities have tackled these challenges, providing an example for the heliophysics community to follow. For example, the Incorporated Research

Institutions for Seismology (IRIS) provides services that include batch data downloads and searchable lists of institution- and community-developed software (IRIS: Data services, 2018). This article presents a framework for creating, maintaining, and sharing these tools within the space physics community. It begins in section 2 by discussing the benefits of open-source tools in general, and Python (van Rossum, 1995) in particular, to the scientific community. Section 3 then presents a summary of currently available heliophysics Python packages. Finally, a framework for future community development is presented in section 4. The desire for an overarching framework (inspired by successful projects in other disciplines, such as AstroPy Astropy Collaboration et al., 2013) has been repeatedly expressed in space physics community meetings and workshops (e.g., Burrell et al., 2017; Stoneback et al., 2017).

2. Upholding Mertonian Norms

The modern, Western scientific ethos can be described by the Mertonian norms of universalism, communalism, disinterestedness, and organized skepticism (Merton, 1957). Universalism speaks to the expectation that scientific results will be evaluated on their own merit. Communalism asserts that scientific knowledge belongs to the entire community, not an individual or single institution. Disinterestedness espouses the rejection of personal gain, in terms of both prestige and income. Organized skepticism requires scientists to be critical of new and old ideas presented by themselves and others in the community. These norms, intended to characterize the social aspects of scientific culture, show a strong subscription across disciplines. The vast majority of scientists believe, to a great extent, that these norms should be upheld (Anderson et al., 2010). All of these norms support the adoption of the open-source philosophy, which refers to things that have been made publicly accessible for people to use, modify, and share.

Making science and scientific analysis open-source has a wide range of benefits. It encourages organized skepticism and addresses the reproducibility crisis (Gil et al., 2016), since all community members would have all of the necessary information available to independently evaluate and build upon past work (Peng, 2011). With proper citation, it brings scientific work that is currently being performed behind the scenes into the open where it can be appropriately evaluated and recognized, reducing opportunities for methodological plagiarism. It also reduces gatekeeping (upholding communalism and universalism), since data and analysis tools would not be hidden behind a paywall.

The reduction of gatekeeping has both ethical and practical benefits, as reduced gatekeeping has been shown to increase diversity within the field (Murray et al., 2018). Although the benefits of a diverse scientific community are obvious, research has shown that diverse working groups produce higher quality and better-cited research (AlShebli et al., 2018; Jehn et al., 1999; Nielsen et al., 2017). These practical benefits also improve the working lives of individual researcher, leading to improved well-being and job satisfaction (Choi, 2017; Kanter, 2008).

Even though open science both fits with the Western scientific ethos and has been shown to help researchers succeed through increased acknowledgement for traditionally unacknowledged work, increased citations, media attention, funding opportunities, and potential collaborations (McKiernan et al., 2016), there is still considerable resistance across the community toward adopting these practices. Opponents of open-source practices in particular (as open-source practices are the focus of this paper) give a range of reasons to keep scientific analysis and data private, including plagiarism, misuse of open science products, risks to career advancement, previous lack of publication opportunities, giving up an edge in funding opportunities, lack of funding for open-source software in space sciences, the possibility of producing a new set of gatekeepers with additional requirements for scientific productivity, and the capturing of academic labor output by commercial interests (e.g., Longo & Drazen, 2016; Lancaster, 2016; Tyfield, 2013). There is also a trend for many researchers to agree with open code policies in principle, but not in practice (Shamir et al., 2013). The adoption of open-source software practices can be hampered by a lack of familiarity with available tools and by concerns about short-term productivity. Although the impacts of the open-source movement on research and individual scientists are ongoing, its early impacts are positive (McKiernan et al., 2016, and references therein). Following the steps and procedures outlined in section 4 will help the space physics community avoid many of the potential negative effects of open-source science.

Open-source science is most easily disseminated when scientists program their tools in an open-source language. While there are many options for open-source languages, this paper focuses on the applicability of Python to space science. Python is a popular option for building scientific data analysis tools, because it is

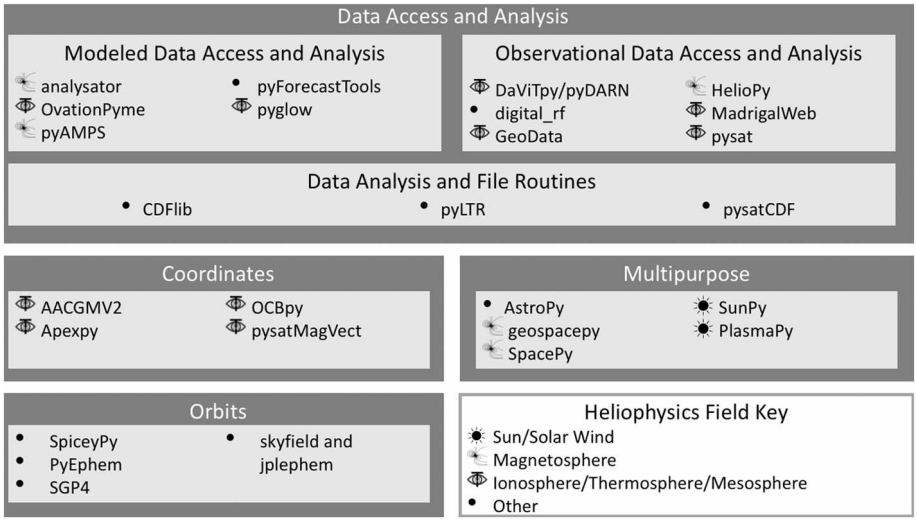

Figure 1. Community-developed heliophysics packages, grouped by field and purpose.

a community-driven, open-source language, with a broad spectrum of well-developed packages. Two features make Python stand out as a language: the relative maturity of Python for scientific applications and the widespread use of Python outside of academia.

Scientific programming in Python typically builds off of the NumPy (Oliphant, 2006) and SciPy (Jones et al., 2001) libraries, which form a mature foundation for scientific applications. A rich ecosystem of software packages that build on the *scientific stack* of Python (NumPy and SciPy) are available; this paper describes the current heliophysics ecosystem. In addition, legacy codes in languages like Fortran (FORmula TRANslation, Backus, 1998) and C Ritchie et al. (1996) are easily wrapped in Python. Many of the models discussed in section A2 utilize this functionality to make empirical models more widely available to the community. Python is a high-level language that natively handles many of the required computational tasks (such as memory management and compilation) without requiring any instruction from the developer. This lowers the barrier for new scientific programmers and allows scientists to focus on their algorithms and analysis rather than the details of computer programming. Finally, SciPy provides the functionality to read proprietary data sets, including those written by the Interactive Data Language (IDL; Harris Geospatial Solutions, Inc. 2018) and Matrix Laboratory (MATLAB; MATLAB, 2018), allowing for a smooth transition in working environments that use legacy codes and data sets.

The widespread use of Python outside of the immediate heliophysics community has a number of advantages. Commonly used techniques such as signal processing have a large support network, allowing the heliophysics community to benefit from outside expertise. It ensures continued language development and support for new technology, since its popularity throughout the global programming community provides the critical mass needed for such investment. Python has been broadly adopted as a teaching language (Fangohr, 2004), allowing the heliophysics community to build on the budding expertise of computationally literate students. Additionally, the use of Python in our community provides students with transferable skills for industry careers (TIOBE Index, 2018; PYPL PopularitY of programming language, 2018).

3. Overview of Current Packages

Heliophysics is a diverse research community, with research interests reaching from the Sun to the lower thermosphere of solar planets and methods encompassing active experiments, ground- and space-based observations, modeling, and theory. An enterprise, this vast requires a wide range of tools. Figure 1 outlines the types of Python packages currently available and their regions of specialization, while Table 1 supplies their licensing information, package location, and the section in Appendix A that contains a description of each package. Although not a complete list of heliophysics Python packages, this figure shows the community-written packages commonly used at the time of publication, as determined through an international survey designed to gage community involvement in collaborative space physics python projects (Burrell et al., 2018). This survey was distributed to six space physics mailing lists and received 223 responses

Table 1
License, Description Section, and Location of Alphabetically Ordered Community-Developed Heliophysics Packages

Name	License	Section	Location
AACGMV2	MIT	A4.1	https://github.com/aburrell/aacgmv2
analysator	GPL-2	A2.1	https://github.com/fmihpc/analysator
apexpy	MIT	A4.2	https://github.com/aburrell/apexpy
Astropy	BSD	A6.1	http://www.astropy.org
CDFlib	MIT	A3.1	https://github.com/MAVENSDC/cdflib
DaViTPy/pyDARN	GPL-3.0	A1.1	https://github.com/vtsuperdarn/davitpy
digital_rf	BSD	A1.2	https://github.com/MITHaystack/digital_rf
GeoData	MIT	A1.3	https://github.com/jswoboda/GeoData
geospacepy	MIT	A6.2	https://github.com/lkilcommons/geospacepy-lite
HelioPy	GPL-3.0	A1.4	http://docs.heliopy.org
jplephem	MIT	A5.4	https://pypi.org/project/jplephem
MadrigalWeb	MIT	A1.5	http://cedar.openmadrigal.org
OCBpy	BSD-3-Clause	A4.3	https://github.com/aburrell/ocbpy
OvationPyme	LGPL-3.0	A2.2	https://github.com/lkilcommons/OvationPyme
PlasmaPy	BSD+Patent	A6.5	http://www.plasmapy.org
pyAMPS	MIT	A2.3	https://github.com/klaundal/pyAMPS
PyEphem	LGPL	A5.2	http://rhodesmill.org/pyephem/index.html
pyForecastTools	BSD-3-Clause	A2.4	https://github.com/drsteve/PyForecastTools
pyglow	MIT	A2.5	https://github.com/timduly4/pyglow
pyLTR	BSD-3-Clause	A3.2	https://github.com/jma127/pyltr
pysat	BSD-3-Clause	A1.6	https://github.com/rstoneback/pysat
pysatCDF	BSD-3-Clause	A3.3	https://github.com/rstoneback/pysatcdf
pysatMagVect	BSD-3-Clause	A4.4	https://github.com/rstoneback/pysatmagvect
SGP4	MIT	A5.3	https://pypi.org/project/sgp4
skyfield	MIT	A5.4	http://rhodesmill.org/skyfield
SpacePy	PSF	A6.3	https://github.com/spacepy/spacepy
SpiceyPy	MIT	A5.1	https://github.com/AndrewAnnex/SpiceyPy
SunPy	MIT	A6.4	https://sunpy.org

Note. License acronyms are defined in section 4.2.

(195 complete and 28 partial) from heliophysicists of all career stages across 28 countries and six continents. These project descriptions are also limited to include only those that are all free and open-source software (FOSS).

At the time of publication, software development within the space physics community is largely an individual effort. For example, the survey results reported that 92.9% of respondents wrote their own analysis code and only 18% contributed to community packages. This has lead to some overlap of functionality in the community-written packages.

Common areas of overlap include file-handling routines, time handling utilities, legacy model implementations, and coordinate transformations. For example, there are three different packages that load NASA common data format (CDF) files and three different packages that use the International Geomagnetic Reference Magnetic Field (IGRF; Thébault et al., 2015). The first instance of overlap is a function of early development and different focuses within heliophysics. SpacePy (see section A6.3) was developed first and provides full CDF library support (reading and writing) along with many other tools for magnetospheric physics. Python Satellite Data Analysis Toolkit CDF (pysatCDF; see section A3.3) was developed as a stand-alone python CDF reader to provide a streamlined user experience by developers in the ionospheric community. CDFlib (see section A3.1) was developed recently and contains a pure Python CDF reader and writer (as opposed to a Python wrapper for the NASA CDF C library).

The second example of overlap is less avoidable. The IGRF is used in two of the coordinate transformation packages (AACGMV2 in section A4.1 and pysatMagVect in section A4.4), the coordinate transformations within SpacePy, and the modeled data access and analysis package, pyglow (see section A2.5). The IGRF implementation within the coordinate packages and SpacePy is contained internally, and it would not be desirable for them to use a common python IGRF python package. The pyglow package is the only one of these packages to provide the user with the direct output from IGRF. While the pyglow implementation may not be ideal for all users, since the package contains many other models and was not developed by the authors of IGRF, it does provide a unique tool needed by the heliophysics community.

Apart from obvious overlap in package functionality and model implementations, there are also instances of conceptual overlap between the packages listed in Figure 1 and Table 1. For example, the packages that focus on the data access and analysis of observations all contain routines to read and load the observations into a python object that may be easily used for data analysis. However, this overlap is not necessarily a problem. Several of these packages focus on a single instrument network, allowing the experts to deal with the specific peculiarities in that data set (e.g., DaViTpy/pyDARN in section A1.1). Others, such as HelioPy and pysat (section A1.6), focus on the instruments in different subdisciplines of heliophysics. The packages with broader scopes can benefit from the targeted work done by the more focused packages. pysat sets a good example of this practice; it uses pysatCDF to read CDF files, DaViTpy/pyDARN to read Super Dual Auroral Radar Network (SuperDARN) data, pysatMagVect for magnetic vector transformations, pyglow for model access, SGP4 and pyEphem for satellite orbit propagation, madrigalWeb for downloading data from Madrigal, and AACGMv2 and apexpy for magnetic coordinates.

4. Toward a Heliophysics Framework

The broad scope of heliophysics, the diverse nature of heliophysics data sets, the lack of funding for scientific software development, and the current tradition of individualism in data analysis all present challenges for the development of a useful Python toolkit for space physics. Despite these challenges, the community has expressed a desire to create a unified framework, similar to Astropy (described in section A6.1), for space science Python tools (Burrell et al., 2017). As the community works toward this unified framework, there are steps that can be taken to reduce duplicated efforts and increase the utility of existing packages. These steps include establishing a steering committee, centralizing information about all Python heliophysics projects (as discussed in section 4.1), providing guidelines for making these projects accessible and connectible (as described in section 4.2), ensuring that scientists receive appropriate recognition for their work (as discussed in section 4.3), encouraging best practices for scientists and developers (as discussed in section 4.4), and holding scientific software to the same standards that scientific theses are held (as discussed in section 4.5). Engaging in these practices will improve the trustworthiness of scientific analysis (Miller, 2006; Kanewala & Bieman, 2014) and address many of the concerns raised by opponents of the open-source philosophy described in section 2.

4.1. Centralization

As with many international, interdisciplinary, collaborative efforts the best way to centralize is online. To this end, a website has been set up to act as a hub for heliophysics Python packages at http://www.heliopython.org. Currently, this website focuses on providing a place to easily locate heliophysics Python packages and community developers. Python packages are not hosted at this website, allowing developers to determine which hosting service best serves their own needs. To be included on this website, package authors submit a pull request on the website's GitHub page and follow the guidelines (which currently include ensuring that the software is citable and FOSS). The code should also provide a useful tool for heliophysics research, falling into one of the categories outlined in Figure 1. Other websites also attempt to curate subsets of heliophysics software (not limited to Python packages), including the CEDAR wiki (http://cedarweb.vsp.ucar.edu/wiki/index.php/Community:Software).

Another goal of the website is to involve new community members in development. By laying out where work has already been done, new developers can dedicate their time to areas in need of improvement, rather than reinventing the wheel. Increasing community involvement will also help maintain active development, keep packages up to date, and improve communication between scientists involved with model development, instrument development, data analysis, and space weather products.

These community organization efforts are recent and could benefit from following the example of more mature scientific community organizations, such as the IRIS. The IRIS consortium aims to advance discov-

ery, research, and education in seismology and has a large membership from research groups across the globe. Their website provides access to data, derived data products, sponsored and community-developed software, education resources, archives of posters and presentations, annual reports, and much more (https://www.iris.edu/hq/). The scope of involvement at IRIS is larger than that currently envisioned by the heliophysics Python community, but following their example and involving national and international professional organizations would increase community involvement, encourage collaboration, and reduce instances of overlapping software development.

4.2. Accessibility

Software licenses are used to set the terms on which the software may be used, modified, or distributed. There are three types of software licenses: FOSS (which may be further divided into permissive and copyleft), proprietary, and hybrid (Morin et al., 2012). FOSS licenses are typically used by academic institutions, since they best espouse the scientific ethos by fostering collaboration, improving reproducibility, and aiding the peer-review process.

Permissive FOSS licenses ensure the widest possible distribution and adaptation of the software, place the fewest restrictions on users, and ease the incorporation of the code by others. The two most common permissive FOSS licenses are the modified Berkeley Software Development (BSD) license (The BSD license, 2018) and the Massachusetts Institute of Technology license (2018). These licenses allow commercial use, distribution, modification, and private use as long as the original developers are attributed and are not held liable (there is no warranty). Table 1 shows that 22 of the 27 packages have a permissive FOSS license (the Python Software Foundation license is a BSD-style permissive FOSS license).

Copyleft licenses were created to assure that the benefits of FOSS are maintained in all future derivatives of the software. The GNU (GNU's Not Unix!) General Public License (GPL; Free Software Foundation, 2007) is the best-known, strong copyleft license. These licenses explicitly describe the treatment of patents related to the software and require that any computer code linked to the licensed software has the same license. The remaining packages in Table 1 have either a GPL or Lesser GPL license.

When choosing a software license, it is important to involve your local technology transfer office. However, for new and existing heliophysics projects to eventually form a cohesive framework, they must all have compatible licenses (Morin et al., 2012). The adaptation of permissive FOSS licenses best supports this goal and so should be strongly considered.

Another important aspect of accessibility is making the source code publicly available (Shamir et al., 2013). The centralization efforts described in section 4.1 can only succeed if the community makes their code available online. Few projects have the funds to set up and maintain an individual repository in perpetuity. Code sharing directories such as GitHub (https://github.com) and RunMyCode.org (Stodden et al., 2012) allow scientists to share, archive, and distribute software at no cost, while also providing a platform for collaborative work.

Sometimes the analysis code for a particular project may not have a scope large enough for its authors to justify creating a repository and DOI for it. In such cases, it may be possible to publish source code as supplemental information in the article that shares the results from that project. This use of supplemental information is explicitly encouraged by some space physics journals, including the American Geophysical Union journals (American Geophysical Union, 2018). However, other journals (including European Geophysical Union journals) require that computer program code be deposited in a repository with a persistent identifier such as a DOI (Annales Geophysicae, 2018).

4.3. Attribution

A common argument against open-source data and software is the danger of plagiarism. When dealing with Python packages, there are two aspects of attribution that should be considered: citations and collaborations. Citations deal with providing scientists with credit for their products (whether they be data sets, software, or theories), and collaborations deal with the ethics of building upon another person's work.

Citations are at the core of academic culture. Most heliophysics journals now require citations to data sets and software, discouraging plagiarism and encouraging communalism. Properly citing FOSS used to obtain and analyze data provides an incentive for other scientists to provide quality FOSS. Since the advent of the digital object identifier (DOI), it is possible to cite software projects and data sets. One service that offers DOIs for software packages is Zenodo (Nielsen & Smith, 2014). Unlike journal articles, which will not change after

publication, software and data sets often have different versions. Zenodo deals with this challenge by allowing the developers to set up a DOI for a package in general and have separate DOIs for different versions.

Before software DOIs, it was common to provide attribution by entering into a collaboration with the person who created the data set that was used or provided the analysis software. This could be an active or an in-name only collaboration, where the work already done was acknowledge by offering a coauthorship. This is a valid method of attribution, though it should not replace proper referencing as described above. This type of consideration should also be extended when beginning a new heliophysics Python project. For example, before creating a Python package for an existing model, it is best practice to contact the model's author. This ensures that the resulting product is of the highest quality and makes it easier to maintain the package when future versions of the model are released.

Another option is for project authors to write a methods paper. This follows the tradition of instrument papers of heliophysics and has the advantage of allowing the project authors to describe the scope and purpose of their project, as well as outline plans for future development. Like instrument papers, software papers may be published in either dedicated journals (such as the Journal of Open Source Software Smith, 2018) or those with a larger scope (such as the Journal of Geophysical Research: Space Physics American Geophysical Union, 2018). When using software that has published methods papers, both the paper and the software DOI should be cited.

Because different people are often involved in each part of a project, this ensures that the efforts of all involved are recognized. For example, an article that uses apexpy should cite Richmond (1995), who describes the coordinate systems; Emmert et al. (2010), which outlines the computational process used to smoothly represent modified apex and quasi-dipole coordinates; and van der Meeren et al. (2018), the Python implementation of Emmert et al. (2010)'s code. In order to ensure that citations are correctly made, it is useful for the software developers to provide this information in much the same way that data and instrument teams currently do.

4.4. Best Practices

Space scientists historically have not received formal education in computer programming. This deficit means that most of the people developing scientific analysis software are not aware of the best practices involved in writing code, providing documentation, and working with collaborators. This should not be a barrier to publishing code. If it is good enough to produce reliable scientific results, it is both good enough for peer consumption and of interest to the scientific community (Barnes, 2010; Shamir et al., 2013). However, good coding practices (like a well-written paper) improve both adaptation by others and reproducibility. To encourage more scientists to contribute code to projects, this section provides an overview of the most relevant of these best practices.

Best practices for coding include supporting the current standard version of the programming language, adhering to the style guide for the programming language, commenting, using descriptive variables, and reducing in-code duplication. At the time of publication, Python 2 is a legacy version with limited future support (Peterson, 2008), and Python 3 is undergoing active development. It is possible to write code compatible to both Python 2 and Python 3; this is the strategy that has been followed in most of the packages described in Appendix A.

Style guidelines for Python are described in Python Enhancement Proposal (PEP) 8 (van Rossum et al., 2013). PEPs are community proposed design documents used to inform the Python community about a new feature, process, or environment. PEP 8 is a style guide for Python code that will help developers write programs that are suitable for community development and incorporation into other packages.

Commenting and using descriptive variables ensure that a new user will be able to follow the coded algorithm and find references to the literature when certain methods or constants are used. For example, when coding up a theory or a model that is published in a journal, it is useful to cite papers with page or equation numbers as comments when they are coded. Comments should be included for every function, defined constant, and algorithm stage in the software. This low-level documentation prevents confusion over units, sources of empirically determined values, and reasons why a particular method was implemented.

Python provides two ways to comment code, through traditional source code comments and through docstrings. Docstrings are literal strings that describe a Python package, function, class, or method and occur as the first statement in one of these objects. Well-written docstrings are an important element of good Python code, since standard use dictates that they contain a summary of the object that they are contained in. This

practice allows docstrings to improve the maintainability and clarity of different routines and speed up the learning curve for new users. They are an improvement over standard comments because they are accessible at runtime and so do not require the user to open up the source code. The standard conventions for writing docstrings are described in PEP 257 (Goodger & van Rossum, 2001).

Variable names are another potential source of elucidation or confusion. Best practice dictates using descriptive variables or ensuring that the variable name indicates what it is. For example, iterative counts are typically have either single letter, lower case names (e.g., *i*, *j*, and *k*), or names that reflect their purpose (e.g., *counter*, *inum*, and *ion_count*).

Documentation is very important, since it ensures that changes to the code (which will affect scientists abilities to replicate results in the future) can be easily identified. This is vital for reproducibility as scientists move onto other projects, making their expertise less accessible to their former collaborators and future investigators. Comments and docstrings are examples of low-level documentation, providing details about specific parts of a software package. However, their dispersion throughout a package, lack of cohesion, and specificity make them insufficient to serve as an overarching (or high-level) document. High-level documentation should provide general information about the project, an installation guide, useful references to appropriate literature, tutorials, and more. Adding to high-level documentation is an excellent way for new users to contribute to software packages, since they have a perspective that the project developers lack.

The SunPy project is an excellent example of documented software. They provide general information about the project, an installation guide, community and developer guidelines, and extensive downloadable tutorials (available at the URL shown in Table 1). Not all projects will be able to support such extensive online documentation. Free online documentation is supported through a variety of websites, including readthedocs.org. This website and other available tools often take advantage of internal, low-level documentation by turning all the docstrings in a package into a manual (Brandl & The Sphinx team, 2018; numpydoc maintainers, 2018), providing a structure around which more extenstive documentation may be written.

Duplication within a package, defined as the existence of code that is either identical to another section of code or has the same intent and structure, happens naturally during the development of complex code. Often this happens unintentionally, when the developer does not yet realize how useful a particular portion of the algorithm will be. Other times this happens intentionally, with the developer deciding that the abstraction necessary to remove the duplication will negatively impact functionality. In general, duplication is problematic, as it makes software packages harder to maintain. Multiple instances of the same structure make algorithms more difficult to analyze and increases the likelihood of not fixing a known bug, since finding all instances of the bug will be more difficult (Kanewala & Bieman, 2014). Thus, best practice dictates that in-code duplication should be removed or managed through documentation.

4.5. Rigor

Producing scientific analysis tools is an integral part of modern heliophsyics analysis and so should be treated with the same level of skepticism as the results that it produces. Peer-reviewing code is one way to ensure that computational methodology is treated with the same level of rigor as other scientific analysis methods. Code review may be performed in-house, with collaborating developers checking each others code (Kelly et al., 2011). It may also be performed as a part of the publishing process. Journals dedicated to publishing and reviewing software (e.g., Smith, 2018) have review criteria that require the implementation of many of the best practices outlined above, including licensing, documentation, and attribution. In addition, the reviewers also ensure that the software under consideration is functional.

To be functional, software must conclusively address a demonstrated need. In science, finding a need for a software tool is not difficult. However, testing the ability of the software to fill that need can be, since it requires that the developers determine the applicable range of conditions their algorithm can support and construct tests that convincingly demonstrate expected outcomes. Common software-testing methods include unit, integration, system, and acceptance testing.

Unit tests are an application of component testing, a method that tests the smallest unit of an application. Python has native and third-party frameworks for unit testing that allow each object in a program to be subjected to a series of tests, ensuring it behaves as expected under a variety of operating conditions. Unit tests are most practically implemented as code is written, discouraging in-code duplication and reducing instances

when analysis results are contaminated by software bugs. The adoption of widespread unit testing promotes the use of short methods with outputs that are readily verifiable.

Even when each individual component of a program is working well, problems may still be encountered when they are brought together. Integration testing ensures that disparate parts of a code behave as expected when they invoke each other and pass data among themselves. The same frameworks that Python uses for unit testing may also be used for integration testing, since the difference between unit and integration testing is the scope of the tests rather then the implementation method.

System testing is a type of black-box testing that evaluates a computing system's compliance with the requirements of a software program. This type of testing ensures that the program is behaving as expected in its current environment. There are many different types of system tests; some that are commonly used by scientific developers include installation testing (ensuring dependencies are installed and up to date) and regression testing (rerunning integration and unit tests to ensure the software still performs as expected).

Acceptance testing is testing performed by users. This type of test can be performed locally, but it is also common to release a package as an *alpha* or *beta* version for this purpose. Acceptance testing is common in heliophysics, where the small size of the community encourages contact between users and developers. Repositories such as GitHub also assist with this level of testing, providing a channel for users to notify developers of problems they encounter when using a software package.

Best practice dictates unit, integration, and system tests should be automated, allowing an inexperienced user to run the software with confidence. There are several commercial services that will automatically test appropriately configured repositories (e.g., TRAVIS CI, GMBH, 2018; Appveyor Systems Inc. 2018). Many offer these services for free to smaller, FOSS projects. These tools promote efficient use of developer time and help users determine which packages are suitable for their research and development environment.

Despite the dangers of using untested software (e.g., Miller, 2006), scientific software is largely untested or undertested. The reasons behind this have been attributed to inherent difficulties in applying tests to scientific software and the culture around scientific software development (Kanewala & Bieman, 2014). The greatest technical challenge facing scientific software testing is the lack of an oracle (or known truth) to test against (Kelly et al., 2011). Techniques such as metamorphic, property based, and *golden run* testing may overcome some of these testing challenges, though more research from the software engineering community is needed (Kanewala & Bieman, 2014, and references therein).

5. Summary

The heliophysics community has expressed a desire to create a unified framework, similar to Astropy, for space science Python tools. This article has outlined some of the challenges facing those of us working toward this goal and presented guidelines that can ease the transition from individualistic and proprietary coding practices to a communalistic environment that takes advantage of each individual's expertise. Many of the existing tools have been outlined as well, and a central location for further information about these Python projects has been identified.

Currently, much of this effort has been undertaken by individuals and small groups with little communication between the parties. This has led to significant overlap between certain packages. For instance, considering only the packages summarized in this article, there are three different CDF readers. This overlap is to be expected at the beginning of a community development effort. There also tends to be more duplication of effort between projects written by different heliophysics disciplines. While combining these packages into a single framework may be desirable, it is more practical to create packages for each subdiscipline that play well together and rely on a common framework for truly universal tasks, such as access to file reading utilities and the calculation of plasma parameters.

Following the best practices in software development and citations is an important step for creating an environment where collaborative software can flourish. Although space scientists have not historically been educated in software development and may not be aware of the best practices to follow when developing software, these practices have immediate benefits that make their adoption advantageous. By adopting these practices in current and future work, the heliophysics community can reduce instances of unreproducible research.

Making science and scientific analysis open source, although not always possible due to institutional constraints, is consistent with scientific ethics and can increase the ease and quality of scientific research. FOSS for scientific analysis makes it possible to fully replicate past scientific analysis, reduces the monetary barrier to participation caused by propriety software, and allows all stages of scientific analysis to receive appropriate acknowledgement. Ultimately, open-source analysis leads to an increase in diversity within the scientific community, increased creativity, and better-cited research.

Appendix A: Description of Current Packages

The detailed descriptions provided here are intended to introduce the heliophysics community to the range of available tools and act as a reference for the packages whose scope is small enough that a paper dedicated to their description is not currently a feasible option.

A1. Observational Data Access and Analysis

Experimental and observational scientists in the heliophysics community rely on ground- and space-based data from a wide variety of instruments. Each of these data sets has their own quirks and standards. The Python packages in this section support data access and analysis for one or more observational data sets.

A1.1. DaViTPy/pyDARN

SuperDARN consists of high-frequency (HF) coherent scatter radars distributed over the northern and southern high latitudes and midlatitudes (Chisham et al., 2007; Greenwald et al., 1995). This network monitors the plasma convection over the poles through backscatter from field-aligned ionospheric irregularities, facilitates studies of magnetosphere-ionosphere interactions, and provides important ionospheric specifications. The Python package pyDARN (currently provided as part of DaViTpy, but also being rewritten as a more focused Python package) provides tools to retrieve, load, analyze, and visualize the SuperDARN backscatter (Sterne et al., 2017).

The core functionality of DaViTpy, pyDARN, sets out to provide the necessary tools to download, read, and plot the SuperDARN data. Currently, there are routines to read the custom-formatted SuperDARN data files that are produced from raw data using the Radar Software Toolkit (SuperDARN Data Analysis Working Group. Participating members et al., 2018), obtain radar hardware information, download the files from one of the data mirrors (defaulting to the Virginia Tech mirror), perform some higher-level processing, and create basic plots (such as range-time-intensity plots and maps of the radar fields of view). DaViTpy also provides coordinate conversion tools, contains tools to perform coordinated studies with satellites and incoherent scatter radars, has a ray tracing tool, and provides Python implementations of several useful models. This additional functionality is useful but will not be included in pyDARN. Modeling and coordinate system packages are better supported by existing packages (e.g., AACGMV2 and pyglow; described in sections A4.1 and A2.5, respectively); ray tracing tools have uses beyond the SuperDARN community and so are better developed as stand-alone packages; and coordinated studies may be performed through packages such as pysat (described in section A1.6), whose focus is providing a common framework for multiple data sets and has already successfully integrated DaViTpy.

A1.2. Digital_rf

The Digital Radio Frequency (RF) project established a disk storage and archival format for radio signals. It uses the Hierarchical Data Format 5 (HDF5), a software package and a file format that can be read by any programming language (Koziol & Robinson, 2018), to define a self-documenting file format for radio frequency data. The Python package, digital_rf, contains routines for reading, writing, and processing radio frequency data using this format. The digital_rf project also has C and MATLAB implementations. This package may be referenced using the URL provided in Table 1.

A1.3. GeoData

GeoData is a software package implemented in Python and MATLAB, which plots and interpolates data from a variety of space physics sources. The main goal of this package is to simplify the plotting and processing of geophysical data, specifically data provided by the CEDAR Madrigal database. To support the data analysis, coordinate transformations between several geographic systems are provided. The processing flow is streamlined by outputting data into HDF5 files. This package may be referenced using the URL provided in Table 1.

A1.4. HelioPy

HelioPy Stansby et al. (2018) is a Python library for heliospheric and planetary physics, whose primary goal is to make it easy to download and import common data sets. It uses CDFlib (see section A3.1) to handle CDF files and is set up to download and ingest a wide variety of solar and satellite data.

At the time of publication this included magnetometer and Solar Wind Ion Composition Spectrometer data from the Advanced Composition Explorer (Stone et al., 1998) spacecraft, magnetometer, Cluster Ion Spectrometry, and Plasma Electron And Current Experiment data from Cluster, particle and magnetic field data from Helios, International Monitoring Platform, the Magnetospheric MultiScale mission, Ulysses, and Wind (Acuña et al., 1995), as well as magnetometer data from the Time History of Events and Macroscale Interactions during Substorms (Angelopoulos, 2009); Acceleration, Reconnection, Turbulence, and Electrodynamics of the Moon's Interaction with the Sun (Angelopoulos, 2010); Cassini Dougherty et al. (2004); and Mercury Surface, Space Environment, GEochemistry, and Ranging (Anderson et al., 2007) missions. Sunspot numbers are also included, and there are plans to include Deep Space Climate Observatory, NASA/Goddard Space Flight Center OMNI, Solar Orbiter (Müller et al., 2013), and Parker Solar Probe (Fox et al., 2016) data.

As well as importing data, HelioPy builds upon the SpiceyPy package (described in section A5.1) to provide an accessible interface for performing orbital calculations. It has also implemented a framework to perform transformations between some common coordinate systems. Future goals for HelioPy involve building upon the Astropy package (described in section A6.1) to provide data with physical units attached, easy methods for transforming between a wider range of coordinate systems, and expanding methods for importing data.

A1.5. MadrigalWeb

The CEDAR Madrigal Database is an online resource for archiving and retrieving many heliophysics data sets. This data can be accessed remotely using Python scripts, through functions provided by the MadrigalWeb package. MadrigalWeb allows users to explore the available experiments and instruments, download the data in several file formats, calculate a range of derived parameters, and perform some instrument-specific coordinate conversions. This package may be referenced using the URL provided in Table 1.

A1.6. pysat

The pysat (Stoneback, Burrell, et al., 2018; Stoneback, Spence, et al., 2018) is a high-level package intended to form a common ground for all packages and data sources in space science. To make this possible, pysat hides the tedious file and data handling behind a single consistent object interface in a class object called *Instrument*. The Instrument object features robust data and metadata handling, generalized data iteration, on-the-fly orbit breakdown, and a versatile system for modifying data. These features enable the creation of instrument independent routines that can operate on the varied dimensionality found across space science.

Pysat is currently being used as a framework for processing the Ion Velocity Measurements (IVM) for the upcoming NASA Ionospheric Connection (ICON) Explorer satellite (Immel et al., 2018) as well as the National Oceanic and Atmospheric Administration (NOAA) Formosa Satellite (Formosat)-7/Constellation Observing System for Meteorology, Ionosphere, and Climate (COSMIC) -2 Constellation. Several instruments from the Communication/Navigation Outage Forecasting System (C/NOFS) satellite (the IVM, Vector Electric Field Instrument (VEFI), and Planar Langmuir Probe (PLP)) (de la Beaujardière & C/NOFS Science Definition Team, 2004); NASA/GSFC OMNI data, SuperDARN grid data (SuperDARN Data Analysis Working Group. Participating members et al., 2018); SuperMAG magnetometer data and indices (Gjerloev, 2009); Formosat-3/COSMIC Global Positioning System (GPS) occultation data (Liou et al., 2007); the Republic of China Satellite (ROCSAT) -1/Formosat-1 IVM (Su et al., 1999); Dst; Kp; the Defense Meteorological Satellite Program (DMSP) IVM (Gorney, 1987; Sun et al., 2018); the Floating Point Measurement Unit (FPMU) on the International Space Station (ISS) (Barjatya et al., 2009); and Thermosphere Ionosphere Mesosphere Energetics Dynamics Solar Extreme ultraviolet Experiment (TIMED-SEE) (Woods et al., 2005) are also currently supported. The most recent release includes a *Constellation* class that allows simultaneous processing of heterogeneous groups of Instruments. The Constellation support was developed by undergraduate computer science students for their senior project. Upcoming versions of pysat will feature support for both Pandas (McKinney, 2010) and xarray (Hoyer & Hamman, 2017) data formats, improving support for multidimensioned data sets.

A2. Modeled Data Access and Analysis

The heliophysics community uses first principles and empirical models for a variety of purposes, including theoretical studies and space weather forecasts. Modeling studies frequently face reproducibility challenges,

since the data are often not made publicly available and many models are not FOSS. The Python packages in this section begin to address these issues by providing access to documented versions of common heliophysics models, as well as standard analysis tools.

A2.1. Analysator

Analysator is an analysis tool developed for Vlasiator, a six-dimensional Vlasov theory-based simulation that focuses on fundamental plasma processes within the near-Earth space environment (von Alfthan et al., 2014). It began as a file reader for the Vlasiator output and has evolved to include analysis and visualization tools. Analysator facilitates studies of particle paths, pitch angle distributions, velocity distributions, and more. More details about the capabilities of this package may be found at the URL provided in Table 1.

A2.2. OvationPyme

OvationPyme is a translation of the Ovation Prime model written in IDL. The Ovation Prime model (Newell et al., 2002) predicts the total electron and ion energies and number fluxes precipitated into the upper atmosphere, as well as the characteristic energy of the precipitation (assuming a Maxwellian distribution). This model is based on observations from the DMSP Special Sensor Precipitating Electron and Ion Spectrometer (SSJ)4/5 particle detectors, which are sensitive to particles in the 30 eV–30 keV energy ranges (Newell et al., 1996). This package may be referenced using the URL provided in Table 1.

A2.3. pyAMPS

pyAMPS (Laundal & Toresen, 2018) is a Python interface for the Average Magnetic field and Polar current System (AMPS) model (Laundal et al., 2018). AMPS is an empirical model of the ionospheric current system and magnetic field, which takes inputs of the solar wind velocity, the interplanetary magnetic field, the dipole tilt, and the $F_{10.7}$ index. The primary model output is the average, large-scale, ionospheric magnetic field disturbances at any location in near-Earth space for the selected set of input parameters. Ionospheric currents are then derived from the magnetic field. pyAMPS includes functions to calculate field-aligned currents, horizontal currents, and estimates of associated ground magnetic field perturbations on a grid. It also includes functions to calculate a time series of model magnetic field perturbations (e.g., along satellite tracks). The empirical model is derived from magnetic field measurements from the Swarm (Friis-Christensen et al., 2006) and Challenging Minisatellite Payload (Reigber et al., 2002) satellites.

A2.4. PyForecastTools

PyForecastTools is a Python package providing implementations of a wide variety of metrics for model validation and forecast verification (Morley, 2018). The metrics include a generic forecast skill score, comparison metrics, scale- and order-dependent biases, a symmetric signed bias, different measures of accuracy, and common error estimates. A key feature in this package is the inclusion of classes for contingency table analyses with multiple methods for estimating confidence intervals on scores. These metrics and contingency tables simplify and illuminate the model validation process, making it more accessible to new users and improving the ability of the scientific community to critically analyze model outputs and forecasts.

A2.5. pyglow

Pyglow is a Python package wrapping multiple empirical ionosphere-thermosphere models, including the Horizontal Wind Model (HWM; Drob et al., 2008; Drob et al., 2015; Hedin, Fleming, et al., 1993; Hedin, Schmidlin, et al., 1993), the International Reference Ionosphere (IRI; Bilitza et al., 2014, 2017), IGRF, the Naval Research Laboratory (NRL) Mass Spectrometer Incoherent Scatter Radar (MSIS) Exobase (NRLMSISE) -00 model (Picone, 2002), and an airglow model (Chartier et al., 2015). To ensure the most up-to-date versions are used at the time of installation, pyglow retrieves the source code (for the non-Python model implementations) from the official distribution sites at the time of installation. Pyglow also includes a package to download and archive the geophysical indices used to drive these models: Ap, Kp, $F_{10.7}$, Dst, and AE. This package may be referenced using the URL provided in Table 1.

A3. Data Analysis and File Routines

Sometimes data analysis methods and file formats reach beyond disciplines. The Python packages described in this section are used by the heliophysics community. They have kept their scope small, though, to better serve multiple scientific fields.

A3.1. CDFlib

CDFlib is a Python package for reading and writing CDF files. Unlike other CDF file-handling packages, CDFlib is a pure Python implementation that does not require any compiled C or Fortran code. This makes installing

the package on different operating systems and platforms very easy. This package may be referenced using the URL provided in Table 1.

A3.2. pyLTR

pyLTR is a Python learning-to-rank (LTR) toolkit. LTR is a supervised machine learning method that trains a model to rank lists of data points rather than score single data points. This is useful for applications such as information retrieval and data mining (Li, 2011). pyLTR provides ranking models, evaluation metrics, tools to load and sort data, and other relevant utilities. pyLTR's goal is to be as full featured as the prevailing open-source LTR library, RankLib (implemented in Java; Arnold et al., 2000), while being significantly easier to use. As such, it contains built-in support for data libraries like NumPy and Pandas. This package may be referenced using the URL provided in Table 1.

A3.3. pysatCDF

pysatCDF Stoneback, Depew, et al. (2018) provides a Python interface to the NASA CDF C library through an intermediate Fortran layer. To enable ease of access, pysatCDF includes the NASA library and couples the build system for the CDF C library with Python installation tools. This makes pysatCDF a self-contained package that is easy to install. pysatCDF also supports the same data access mechanisms present in SpacePy's CDF routines to enable cross-package interoperability (SpacePy is described in section A6.3). pysatCDF operates independently of pysat (described in section A1.6), though it also features routines designed to simplify the integration of science data sets into pysat. For example, the NASA CDAWeb-hosted mission support within pysat relies upon pysatCDF.

A4. Coordinates

Coordinate systems are used to order data in a sensible fashion. In complex and coupled systems, there is often not a single best way to do this. As a result, there are a plethora of coordinate systems used within heliophysics. This section outlines several packages that focus on calculating coordinate transformations between geographic and geomagnetic systems.

A4.1. AACGMV2

Corrected geomagnetic (CGM) coordinates are defined in terms of the intersection between the local IGRF field line and the dipole equatorial plane. The CGM longitude is the centered dipole longitude of this intersection (see Laundal & Richmond, 2017, for a review). CGM latitude is the latitude where a dipole field line with this radius intersects a sphere at $1R_{\oplus}$ (Earth radius). This coordinate conversion requires magnetic field line tracing with the IGRF and is thus relatively computation heavy. Early implementations, based on look-up tables, only allowed for conversions of points at ground. When this limitation was later removed, CGM coordinates became better known as altitude-adjusted corrected geomagnetic (AACGM) coordinates.

AACGM coordinates were originally developed for the purpose of comparing ground-based radar backscatter measurements from high-latitude locations in both hemispheres. Originally known as the Polar Anglo-American Conjugate Experiment geomagnetic coordinate system (Baker & Wing, 1989), AACGM coordinates preserve latitude and longitude along magnetic field lines (as specified by IGRF). AACGMv2 is the most recent incarnation of this coordinate system, providing coefficients that can be used to obtain AACGM coordinates between 0 and 2,000 km above the surface of the earth and field line tracing for higher altitudes. These coordinates are designed to be highly accurate at polar and middle latitudes and may be undefined at the dip equator and near the South Atlantic Anomaly (Shepherd, 2014). The Python implementation of AACGMV2 provides an interface for the C code developed by Shepherd (2014). It allows conversions between geographic (or geodetic) coordinates and AACGM latitude, longitude, and local time, and it is possible to provide alternative versions of the AACGM coefficients (Burrell et al., 2018).

A4.2. apexpy

Magnetic apex coordinates are defined in a similar way as CGM/AACGM coordinates. Instead of using the intersection of the IGRF model with the dipole equatorial plane, magnetic apex coordinates are based on the field line apex, the highest point above the geoid. Apex longitude is defined as the centered dipole longitude of the field line apex. Two different definitions are used for the latitude, which separate two variants of the apex coordinates: modified apex coordinates and quasi-dipole coordinates (Richmond, 1995).

In modified apex coordinates, the latitude is defined as the latitude where a dipole field line with radius equal to the apex height intersects a sphere with radius $R_{\oplus} + h_R$, where h_R is a chosen reference height. In quasi-dipole coordinates, this sphere is replaced by $R_{\oplus} + h$, where h is the height of the point of interest.

This means that modified apex coordinates are constant along IGRF magnetic field lines, while quasi-dipole coordinates are not. If h_R or h are small, the apex coordinates are very similar to AACGM coordinates at high latitudes. In contrast to AACGM coordinates, apex coordinates are defined at low latitudes (above h_R in the case of modified apex coordinates).

apexpy (van der Meeren et al., 2018) provides functions to convert to and from apex coordinates. It also includes functions to calculate base vectors, needed to do vector calculus in these nonorthogonal coordinate systems (Richmond, 1995; Laundal & Richmond, 2017), and map electric fields along magnetic field lines to different altitudes. At apexpy's core is a wrapper for the Fortran library described in Emmert et al. (2010).

A4.3. OCBpy

High-latitude ionospheric processes interact directly with the magnetosphere and the solar wind. These interactions lead to different ionospheric behaviors in the auroral oval and the polar cap, a region where the magnetic field lines are open (connecting to the interplanetary magnetic field). Chisham (2017) developed a coordinate system for high latitudes that arranges observations relative to the Open Closed field line Boundary (OCB). This was shown to affect the formation of empirical models and statistical studies, which could otherwise inadvertently combine observations taken in the auroral oval and polar cap. OCBpy is a Python package that determines the location of data relative to the OCB from AACGM coordinates and, when appropriate, scales the measurements to reflect the influence of the cross-polar cap potential drop (Burrell & Chisham, 2018).

The coordinate transformation OCBpy performs requires the OCB location, the data location, the data value, and knowledge of how this value is related to the ionospheric electric field. OCB locations are currently provided for the northern hemisphere between May 2000 and August 2002 using observations from the FUV Imager on board the IMAGE satellite (Mende, Heetderks, Frey, Lampton, et al., 2000; Mende, Heetderks, Frey, Stock, et al., 2000). Future developments will increase the number of OCB locations, incorporate AACGMV2 (described in section A4.1) to allow conversion between geographic and OCB coordinates, and incorporate the pysat Instrument class (described in section A1.6) to allow coordinate transformations for a wide range of data sets.

A4.4. pysatMagVect

The motion of plasma in the ionosphere is constrained by the anisotropic conductivity of magnetized plasma, making perpendicular motion much more difficult than motion along magnetic field lines. To best reflect a geophysical basis for interpreting electric fields and plasma motion, pysatMagVect calculates unit vectors in the magnetic field-aligned, meridional, and zonal directions. The field-aligned direction points along the magnetic field, defined to be positive when directed from south to north. The meridional unit vector is perpendicular to the field-aligned direction and constrained to the meridional plane, the plane that contains the field line. At the magnetic equator, the meridional vector is vertical and defined to be positive when directed upward. The zonal direction completes the orthogonal set and is positive when directed toward the East.

The vector system is calculated by field line tracing and vector math. Reference code for IGRF in Fortran is coupled with SciPy's Ordinary Differential Equation numerical integrator to provide a robust and accurate field-line tracing system. The relative locations of the field line footprints in the Northern and Southern Hemispheres are compared to a specified location and used to define the magnetic meridian plane. The geomagnetic vector system may also be determined using the local magnetic field.

Under the common assumption that geomagnetic lines are equipotentials in the ionosphere, electric fields are *mapped* along field lines. This allows measurements made anywhere along a field line to be translated to another location on the field. The electric field values are not strictly maintained along the field line as the magnetic flux density changes with position. pysatMagVect (Stoneback, 2018) uses field-line tracing to determine both the geomagnetic unit vectors as well as scalars needed to determine the changes in an electric field along a geomagnetic field line.

To support these and other calculations, pysatMagVect also includes coordinate and vector transformations. Translation between Earth Centered Earth Fixed (where x lies in equatorial plane pointing from center of Earth toward 0° longitude, y similarly points toward 90° longitude, and z completes the system and is aligned with the rotation axis), as well as geographic and geodetic (WGS84) systems. Vector projections onto an ECEF or other custom basis specified in ECEF are also supported.

A5. Orbits

Space-based data from satellites and rockets requires knowledge of orbital mechanics to properly determine locations. The ephemera that contain orbital information are often difficult to read. This process becomes increasingly complex when data from multiple satellites are used together. The Python packages in this section provide tools for determining the orbital mechanics of natural and artificial satellites and can be very useful for mission planning, conjunction studies, and post multipoint analysis.

A5.1. SpiceyPy

SPICE is a geometry information system designed, built, and maintained by the Navigation and Ancillary Information Facility, acting under the directions of NASA's Planetary Science Division, to assist NASA scientists in planning and interpreting scientific observations from space-borne instruments. As FOSS, SPICE has been used internationally to assist spacecraft mission concept development, data analysis, and the correlation of instruments on multiple spacecraft. SPICE components, or kernels, have different functions that give the system its name. *S* stands for spacecraft ephemeris; *P* stands for planet, satellite, comet, asteroid, or any other target body ephemerids; *I* stands for instrument information; *C* stands for C-matrix (a matrix containing orientation information); and *E* stands for events information (which summarizes mission activities). SPICE was originally implemented in Fortran 77 but is now officially supported in C, IDL, MATLAB, and Java.

There are several unofficial Python implementations of SPICE, one of which is SpiceyPy (Annex et al., 2018). SpiceyPy provides a Python interface for more than 98% of C SPICE functions, a greater percentage than are made available through officially supported IDL and MATLAB interfaces, and is thoroughly tested through the use of continuous integration services. SpiceyPy also simplifies Python to C interactions by presenting an interface that simplifies C function parameters (such as array lengths and temporary data structures) to idiomatic Python and through data type conversions of common NumPy data types (Annex, 2017).

A5.2. PyEphem

PyEphem is a Python package based on the XEphem C package (Xephem 3.7.7, 2018) that allows users to determine the position of astronomical bodies or artificial satellites with user provided orbital elements (Rhodes, 2008). These locations are available in equatorial, ecliptic, and galactic coordinates, as well as location in the sky relative to known *landmarks*, such as a constellation. Although a database of heliophysics spacecraft is not provided by PyEphem, it does provide orbital elements for a wide variety of astronomical object and long-duration artificial satellites.

A5.3. SGP4

The Simplified General Perturbations #4 (SGP4) Python package uses two-line element data for an Earth-orbiting satellite to calculate its position and velocity. The purpose of the original C++ and Python implementations of SGP4 is to foster collaboration between partners and allies by providing a high-quality, FOSS propagator compatible with data products produced by the United States Air Force Space Command Joint Space Operations Center. The Python package was regularly tested against the C++ 2010 SGP4 propagator suit to ensure that the predictions agree within 0.1 mm. This package may be referenced using the URL provided in Table 1.

A5.4. skyfield and jplephem

Skyfield and jplephem are both pure Python ephemeris packages. Jplephem uses the Jet Propulsion Laboratory (JPL) ephemeris to predict the position and velocity of a planet or other solar system body, producing plain three-dimensional vectors. Skyfield builds upon jplephem, computing positions for stars, planets, and Earth-orbiting satellites in a variety of coordinate systems. Its results are tested against the Astronomical Almanac (produced by the United States Naval Observatory and Her Majesty's Nautical Almanac Office) to within 0.5 milliarcseconds. Each of these packages may be referenced using the URLs provided in Table 1.

A6. Multipurpose

With appropriate scope, vision, and resources, it is possible to provide a maintainable Python toolkit that encompasses multiple functions. The software outlined in this section contain functionality that crosses the categories outlined in sections A1 – A5.

A6.1. Astropy

The Astropy Project is a community effort to develop a core Python package for astronomy, as well as improving the usability, interoperability, and collaboration between other astronomical Python packages (Astropy Collaboration et al., 2013). To this end, Astropy consists of a core package aimed at professional astronomers

and astrophysicists and secondary packages that may or may not have been created by the core development team. The secondary (or affiliated) packages share the goals of Astropy and often use the core package as a base.

The core Astropy package contains data structures and transformations, file handling, remote communication, computational tools, analysis utilities, and other supporting tools. The data structures contain common astronomical constants, units, and coordinates that are often useful for heliophysics research. The file handling, computational, and analysis tools are commonly used within the planetary portion of the heliophysics community, which often relies on telescope observations to observe auroral and other ionospheric emissions.

Astropy actively supports community development through affiliated packages with the aim of improving the availability of interpretable tools in the astronomical community. A template for affiliated packages is provided to encourage new developers. A complete list of affiliated Astropy packages is available at http://affiliated.astropy.org/.

A6.2. Geospacepy

Geospacepy is a small library of Python functions used for space science data analysis and can currently be referenced using the URL provided in Table 1. It includes a set of utilities for handling different time formats, some coordinate conversions, and plotting utilities for several standard data plots. It also downloads and reads in OMNI data. Geospacepy uses SpacePy (described in section A6.3) to read CDF files and PyEphem (described in section A5.2) for astrodynamical calculations.

A6.3. SpacePy

SpacePy is a Python package that contains a set of common analysis software primarily developed for the magnetospheric community (Morley et al., 2010; Morley et al., 2011). SpacePy includes tools to read from a variety of data formats including NASA CDF. A core feature is the *datamodel* module, which provides classes that allow both data and metadata to be loaded and stored in a common form from a range of sources including CDF, HDF5, and netCDF files, with full support for writing to any of these formats. Additional functionality includes conversion between different time systems, and an interface to the International Radiation Belt Environment Modeling library (Boscher et al., 2008) for computing terrestrial magnetic coordinate transformations, evaluating model magnetic fields, and tracing magnetic field lines and drift shells. SpacePy also provides a range of empirical models, statistical analysis tools, data handling tools, and plotting convenience functions.

A suite of tools to work with simulation output from components of the Space Weather Modeling Framework is included in the pybats module. Supported components include the Block-Adaptive-Tree Solar-Wind Roe-Type Upwind Scheme (BATS-R-US; De Zeeuw et al., 2000; Powell et al., 1999), Polar Wind Outflow Model (PWOM; Glocer et al., 2009), Ridley Ionosphere Model (Ridley et al., 2004), Rice Convection Model (RCM; Toffoletto et al., 2003), Ring current Atmosphere interactions Model with Self-Consistent B field (RAM-SCB; Jordanova et al., 2012), and Global Ionosphere-Thermosphere Model (GITM; Ridley et al., 2006).

SpacePy's core development team is continuing work to improve ease of installation, ease of use, and compatibility with other packages across heliophysics. The SpacePy project is planning a move to an environment inspired by *scikits* (SciPy developers, 2018), where a streamlined core SpacePy package provides key functionality for a broad range of applications and is supplemented by specialized packages that build on the core of SpacePy.

A6.4. SunPy

SunPy is a Python package for solar physics that was developed with the help and support of the global community (SunPy Community, T. et al., 2015). It provides a comprehensive data analysis environment that allows researchers to carry out their tasks with minimal effort. As a mature solar physics package, SunPy handles data acquisition (from observations and models), analysis, and plotting. It includes functions that perform common solar coordinate transformations, unit conversions, and deals with temporal parsing. SunPy also allows local user customization of the analysis environment.

A6.5. PlasmaPy

PlasmaPy is a community-developed and community-driven Python package for plasma physics (PlasmaPy Community et al., 2018). Still at an early stage of development, it aims to provide the common tools used within the field of plasma physics for theoretical and experimental analysis. It currently includes access to particle data, functions to calculate plasma parameters, dielectric tensor components, and transport coefficients.

Acknowledgments
The authors would like to acknowledge the support of all the heliophysics scientists who have participated in the CEDAR Snakes on a Spaceship workshops, the recent Snakes on a Spaceship survey, AGU meet-ups, and other community organization efforts. We acknowledge use of NASA/GSFC's Space Physics Data Facility's OMNIWeb service and OMNI data by many of these Python packages (https://omniweb.gsfc.nasa.gov/). The Python packages and data accessed by these packages may be obtained by following the links provided in Table 1. A. G. Burrell is supported by the Chief of Naval Research. D. Stansby is supported by the U.K. Science and Technology Facilities Council studentship ST/N504336/1. Contributions by S. K. Morley were performed under the auspices of the U.S. Department of Energy and partly funded by the Laboratory Directed Research and Development program (Grant 20170047DR).

References

Acuña, M. H., Ogilvie, K. W., Baker, D. N., Curtis, S. A., Fairfield, D. H., & Mish, W. H. (1995). The Global Geospace Science Program and its investigations. *Space Science Reviews*, *71*(1), 5–21.

AlShebli, B. K., Rahwan, T., & Woon, W. L. (2018). Ethnic diversity increases scientific impact. *ArXiv e-prints*.

American Geophysical Union (2018). Supporting information guidelines. https://publications.agu.org/author-resource-center/auxiliary-materials-guidelines/

Anderson, B. J., Acuña, M. H., Lohr, D. A., Scheifele, J., Raval, A., Korth, H., & Slavin, J. A. (2007). The magnetometer instrument on MESSENGER, *The MESSENGER mission to Mercury* (pp. 417–450). New York, NY: Springer.

Anderson, M. S., Ronning, E. A., De Vries, R., & Martinson, B. C. (2010). Extending the Mertonian norms: Scientists' subscription to norms of research. *The Journal of Higher Education*, *81*(3), 366–393.

Angelopoulos, V. (2009). The THEMIS mission. In J. L. Burch & V. Angelopoulos (Eds.), *The THEMIS mission* (pp. 5–34). New York, NY: Springer.

Angelopoulos, V. (2010). The ARTEMIS mission. In C. Russell & V. Angelopoulos (Eds.), *The ARTEMIS mission* (pp. 3–25). New York, NY: Springer.

Annales Geophysicae (2018). Manuscript preparation guidelines for authors. https://www.annales-geophysicae.net/for_authors/manuscript_preparation.html

Annex, A. (2017). SpiceyPy, a Python Wrapper for SPICE. In A. Annex (Ed.), *Third planetary data workshop and the planetary geologic mappers annual meeting* (Vol. 1986, pp. 7081). Flagstaff, Arizona: LPI Contributions.

Annex, A., Carcich, B., Murakami, S.-Y., Kulumani, S., de Val-Borro, M., Stefko, M., et al. (2018). Andrewannex/spiceypy: Spiceypy 2.1.2. https://doi.org/10.5281/zenodo.1291631

Appveyor Systems Inc. (2018). Continuous integration for windows and linux. https://www.appveyor.com/

Arnold, K., Gosling, J., & Holmes, D. (2000). *The java programming language* (3rd ed.). Boston, MA: Addison-Wesley Longman Publishing Co., Inc.

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., et al. (2013). Astropy: A community python package for astronomy. *Astronomy and Astrophysics*, *558*, A33.

Backus, J. (1998). The history of FORTRAN I, II, and III. *IEEE Annals of the History of Computing*, *20*(4), 68–78.

Baker, K. B., & Wing, S. (1989). A new magnetic coordinate system for conjugate studies at high latitudes. *Journal of Geophysical Research*, *94*, 9139–9143.

Barjatya, A., Swenson, C. M., Thompson, D. C., & Wright Jr., K. H. (2009). Invited article: Data analysis of the floating potential measurement unit aboard the international space station. *Review of Scientific Instruments*, *80*(4), 041301.

Barnes, N. (2010). Publish your computer code: It is good enough. *Nature*, *467*, 753.

Bilitza, D., Altadill, D., Truhlik, V., Shubin, V., Galkin, I., Reinisch, B., & Huang, X. (2017). International Reference Ionosphere 2016: From ionospheric climate to real-time weather predictions. *Space Weather*, *15*, 418–429. https://doi.org/10.1002/2016SW001593

Bilitza, D., Altadill, D., Zhang, Y., Mertens, C., Truhlik, V., Richards, P., et al. (2014). The International Reference Ionosphere 2012—A model of international collaboration. *Journal of Space Weather and Space Climate*, *4*, A07.

Boscher, D., Bourdarie, S., Brien, P., & Guild, T. (2008). IRBEM-LIB download. https://sourceforge.net/projects/irbem/

Brandl, G., & The Sphinx team (2018). Overview—sphinx 1.8.0+ documentation. http://www.sphinx-doc.org/en/master/

Burrell, A. G., & Chisham, G. (2018). aburrell/ocbpy: Beta release. https://doi.org/10.5281/zenodo.1217177

Burrell, A. G., Kellerman, A., Halford, A., Ireland, J., Fadden, J., Piker, C., et al. (2017). Fall AGU Python meet up. Private Communication.

Burrell, A. G., Klenzing, J., & Stoneback, R. A. (2018). Python for space science, snakes on a spaceship: The return of the python. http://cedarweb.vsp.ucar.edu/wiki/images/c/c8/Snakes_welcome.pdf

Burrell, A., van der Meeren, C., & Laundal, K. M. (2018). aburrell/aacgmv2: v2.4.2. https://doi.org/10.5281/zenodo.1250727

Chartier, A. T., Makela, J. J., Liu, H., Bust, G. S., & Noto, J. (2015). Modeled and observed equatorial thermospheric winds and temperatures. *Journal of Geophysical Research: Space Physics*, *120*, 5832–5844. https://doi.org/10.1002/2014JA020921

Chisham, G. (2017). A new methodology for the development of high-latitude ionospheric climatologies and empirical models. *Journal of Geophysical Research: Space Physics*, *122*, 932–947. https://doi.org/10.1002/2016JA023235

Chisham, G., Lester, M., Milan, S. E., Freeman, M. P., Bristow, W. A., Grocott, A., et al. (2007). A decade of the Super Dual Auroral Radar Network (SuperDARN): Scientific achievements, new techniques and future directions. *Surveys in Geophysics*, *28*(1), 33–109. https://doi.org/10.1007/s10712-007-9017-8

Choi, S. (2017). Workforce diversity and job satisfaction of the majority and the minority: Analyzing the asymmetrical effects of relational demography on whites and racial/ethnic minorities. *Review of Public Personnel Administration*, *37*(1), 84–107. https://doi.org/10.1177/0734371X15623617

De Zeeuw, D. L., Gombosi, T. I., Groth, C. P. T., Powell, K. G., & Stout, Q. F. (2000). An adaptive MHD method for global space weather simulations. *IEEE Transactions on Plasma Science*, *28*(6), 1956–1965.

de la Beaujardière, O., & C/NOFS Science Definition Team (2004). C/NOFS: A mission to forecast scintillations. *Journal of Atmospheric and Solar-Terrestrial Physics*, *66*(17), 1573–1591.

Dougherty, M. K., Kellock, S., Southwood, D. J., Balogh, A., Smith, E. J., Tsurutani, B. T., et al. (2004). The Cassini magnetic field investigation. In C. T. Russell (Ed.), *The Cassini-Huygens mission* (Vol. 2, pp. 331–383). Dordrecht: Springer, Dordrecht.

Drob, D. P., Emmert, J. T., Crowley, G., Picone, J. M., Shepherd, G. G., Skinner, W., et al. (2008). An empirical model of the Earth's horizontal wind fields: HWM07. *Journal of Geophysical Research*, *113*, A12304. https://doi.org/10.1029/2008JA013668

Drob, D. P., Emmert, J. T., Meriwether, J. W., Makela, J. J., Doornbos, E., Conde, M., et al. (2015). An update to the Horizontal Wind Model (HWM): The quiet time thermosphere. *Earth and Space Science*, *2*, 301–319. https://doi.org/10.1002/2014EA000089

Emmert, J. T., Richmond, A. D., & Drob, D. P. (2010). A computationally compact representation of magnetic-apex and quasi-dipole coordinates with smooth base vectors. *Journal of Geophysical Research*, *115*, A08322. https://doi.org/10.1029/2010JA015326

Fangohr, H. (2004). A comparison of C, MATLAB, and Python as teaching languages in engineering. In M. Bubak, G. D. van Albada, P. M. A. Sloot, & J. Dongarra (Eds.), *Computational science—ICCS 2004* (pp. 1210–1217). Berlin, Heidelberg: Springer Berlin Heidelberg.

Fox, N. J., Velli, M. C., Bale, S. D., Decker, R., Driesman, A., Howard, R. A., et al. (2016). The Solar Probe Plus Mission: Humanity's first visit to our star. *Space Science Reviews*, *204*(1-4), 7–48. https://doi.org/10.1007/s11214-015-0211-6

Free Software Foundation, Inc. (2007). The GPL license. https://opensource.org/licenses/GPL-3.0

Friis-Christensen, E., Lühr, H., & Hulot, G. (2006). Swarm: A constellation to study the Earth's magnetic field. *Earth, Planets and Space*, *58*(4), 351–358. https://doi.org/10.1186/BF03351933

Gil, Y., David, C. H., Demir, I., Essawy, B. T., Fulweiler, R. W., Goodall, J. L., et al. (2016). Toward the geoscience paper of the future: Best practices for documenting and sharing research from data to software to provenance. *Earth and Space Science*, *3*, 388–415. https://doi.org/10.1002/2015EA000136

Gjerloev, J. W. (2009). A global ground-based magnetometer initiative. *EOS, Transactions American Geophysical Union*, *90*(27), 230–231.
Glocer, A., Tóth, G., Gombosi, T., & Welling, D. (2009). Modeling ionospheric outflows and their impact on the magnetosphere, initial results. *Journal of Geophysical Research*, *114*, A05216. https://doi.org/10.1029/2009JA014053
Goodger, D., & van Rossum, G. (2001). PEP 257—Docstring conventions. https://www.python.org/dev/peps/pep-0257/
Gorney, D. J. (1987). U.S. National Report to the International Union of Geodesy and Geophysics: U.S. progress in auroral research: 1983–1986. *Reviews of Geophysics*, *25*(3), 555–569.
Greenwald, R. A., Baker, K. B., Dudeney, J. R., Pinnock, M., Jones, T. B., Thomas, E. C., et al. (1995). DARN/SUPERDARN. *Space Science Reviews*, *71*(1-4), 761–796.
Harris Geospatial Solutions, Inc. (2018). Documentation center [harris geospatial docs center]. http://www.harrisgeospatial.com/docs/whatsnew.html
Hedin, A. E., Fleming, E. L., Manson, A. H., Schmidlin, F. J., Avery, S. K., Clark, R. R., et al. (1993). Empirical wind model for the middle and lower atmosphere—Part 2: Local time variations (NASA Technical Memorandum 104592) Greenbelt, MD: Goddard Space Flight Center. https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19940017389.pdf
Hedin, A. E., Schmidlin, F. J., Fleming, E. L., Avery, S. K., Manson, A. H., & Franke, S. J. (1993). Empirical wind model for the middle and lower atmosphere—Part 1: Local time average. https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19930015971.pdf
Hoyer, S., & Hamman, J. (2017). xarray: ND labeled arrays and datasets in Python. *Journal of Open Research Software*, *5*(1), 304.
Immel, T. J., England, S. L., Mende, S. B., Heelis, R. A., Englert, C. R., Edelstein, J., et al. (2018). The Ionospheric Connection Explorer Mission: Mission goals and design. *Space Science Reviews*, *214*(1), 13.
IRIS: Data services (2018). https://http://ds.iris.edu/ds/
Jehn, K. A., Northcraft, G. B., & Neale, M. A. (1999). Why differences make a difference: A field study of diversity, conflict, and performance in workgroups. *Administrative Science Quarterly*, *44*(4), 741–763. http://www.jstor.org/stable/2667054
Jones, E., Oliphant, T., Peterson, P., & The SciPy developers (2001). SciPy: Open source scientific tools for Python. http://www.scipy.org/
Jordanova, V. K., Welling, D. T., Zaharia, S. G., Chen, L., & Thorne, R. M. (2012). Modeling ring current ion and electron dynamics and plasma instabilities during a high-speed stream driven storm. *Journal of Geophysical Research*, *117*, A00L08. https://doi.org/10.1029/2011JA017433
Kanewala, U., & Bieman, J. M. (2014). Testing scientific software: A systematic literature review. *Information and Software Technology*, *56*(10), 1219–1232.
Kanter, R. M. (2008). Men and women of the corporation: New edition. Basic Books.
Kelly, D., Smith, S., & Meng, N. (2011). Software engineering for scientists. *Computational Science and Engineering*, *13*(5), 7–11.
Koziol, Q., & Robinson, D. (2018). HDF5, [Computer Software]. https://doi.org/10.11578/dc.20180330.1
Lancaster, A. (2016). Open science and its discontents. http://ronininstitute.org/open-science-and-its-discontents/1383/
Laundal, K. M., Finlay, C. C., Olsen, N., & Reistad, J. P. (2018). Solar wind and seasonal influence on ionospheric currents from swarm and champ measurements. *Journal of Geophysical Research: Space Physics*, *123*, 4402–4429. https://doi.org/10.1029/2018JA025387
Laundal, K. M., & Richmond, A. D. (2017). Magnetic coordinate systems. *Space Science Reviews*, *206*, 27–59.
Laundal, K. M., & Toresen, M. (2018). pyAMPS. https://github.com/klaundal/pyAMPS
Li, H. (2011). A short introduction to learn to rank. *IEICE Transactions on Information and Systems*, *E94-D*(10), 1854–1862.
Liou, Y.-A., Pavelyev, A. G., Liu, S.-F., Pavelyev, A. A., Yen, N., Huang, C. Y., & Fong, C.-J (2007). FORMOSAT-3/COSMIC GPS radio occultation mission: Preliminary results. *IEEE Transactions on Geoscience and Remote Sensing*, *45*, 3813–3826.
Longo, D. L., & Drazen, J. M. (2016). Data sharing. *The New England Journal of Medicine*, *374*, 276–277.
MATLAB (2018). version 9.4 (R2018a), The MathWorks Inc. ,Natick, Massachusetts.
McGranaghan, R. M., Bhatt, A., Matsuo, T., Mannucci, A. J., Semeter, J. L., & Datta Barua, S. (2017). Ushering in a new frontier in geospace through data science. *Journal of Geophysical Research: Space Physics*, *122*, 12,586–12,590. https://doi.org/10.1002/2017JA024835
McKiernan, E. C., Bourne, P. E., Brown, C. T., Buck, S., Kenall, A., Lin, J., et al. (2016). How open science helps researchers succeed. eLIFE, e16800.
McKinney, W. (2010). Data structures for statistical computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in science conference* (pp. 51–56).
Mende, S. B., Heetderks, H., Frey, H. U., Lampton, M., Geller, S. P., Abiad, R., et al. (2000). Far ultraviolet imaging from the IMAGE spacecraft. 2. wideband FUV imaging. *Space Science Reviews*, *91*(1), 271–285.
Mende, S. B., Heetderks, H., Frey, H. U., Stock, J. M., Lampton, M., Geller, S. P., et al. (2000). Far ultraviolet imaging from the IMAGE spacecraft. 3. Spectral imaging of Lyman-alpha and OI 135.6 nm. *Space Science Reviews*, *91*(1), 287–381.
Merton, R. (1957). *Social theory and social structure* (Rev. and enl.).: Glencoe, IL: Free Press.
Miller, G. (2006). A scientist's nightmare: Software problem leads to five retractions. *Science*, *314*(5807), 1856–1857.
Morin, A., Urban, J., & Sliz, P. (2012). A quick guide to software licensing for the scientist-programmer. *PLOS Computational Biology*, *8*(7), e1002598.
Morley, S. (2018). drsteve/PyForecastTools: PyForecastTools: Version 1.0.1. https://doi.org/10.5281/zenodo.1299389
Morley, S. K., Koller, J., Welling, D. T., Larsen, B. A., Henderson, M. G., & Niehof, J. T. (2011). Spacepy—A Python-based library of tools for the space sciences. In *Proceedings of the 9th Python in science conference (SciPy 2010)*, Austin, TX.
Morley, S. K., Welling, D. T., Koller, J., Larsen, B. A., & Henderson, M. G. (2010). Spacepy - a python-based library of tools for the space sciences.
Müller, D., Marsden, R. G., St. Cyr, O. C., & Gilbert, H. R. (2013). Solar Orbiter. *Solar Physics*, *285*(1-2), 25–70. https://doi.org/10.1007/s11207-012-0085-7
Murray, D., Siler, K., Lariviére, V., Chan, W. M., Collings, A. M., Raymond, J., & Sugimoto, C. R. (2018). Gender and international diversity improves equity in peer review, bioRxiv.
Newell, P. T., Lyons, K. M., & Meng, C.-I. (1996). A large survey of electron acceleration events. *Journal of Geophysical Research*, *101*(A), 2599–2614.
Newell, P. T., Sotirelis, T., Ruohoniemi, J. M., Carbary, J. F., Liou, K., Skura, J. P., et al. (2002). Ovation: Oval variation, assessment, tracking, intensity, and online nowcasting. *Annales Geophysicae*, *20*(7), 1039–1047.
Nielsen, M. W., Alegria, S., Börjeson, L., Etzkowitz, H., Falk-Krzesinski, H. J., Joshi, A., et al. (2017). Opinion: Gender diversity leads to better science. *Proceedings of the National Academy of Sciences*, *114*(8), 1740–1742.
Nielsen, L. H., & Smith, T. (2014). Zenodo overview. https://doi.org/10.5281/zenodo.8428
numpydoc maintainers (2018). numpydoc—Numpy's Sphinx extentions. https://numpydoc.readthedocs.io/en/latest/
Oliphant, T. E. (2006). *Guide to NumPy*. Trelgol Publishing.
PYPL PopularitY of programming language (2018). http://pypl.github.io/PYPL.html
Peng, R. D. (2011). Reproducible research in computational science. *Science*, *334*(6060), 1226–1227.

Peterson, B. (2008). PEP 373 — Python 2.7 release schedule. https://www.python.org/dev/peps/pep-0373/
Picone, J. M. (2002). NRLMSISE-00 empirical model of the atmosphere: Statistical comparisons and scientific issues. *Journal of Geophysical Research*, *107*(A12), 1468–SIA 15–16.
PlasmaPy Community, Murphy, N. A., Leonard, A. J., Staczak, D., Kozlowski, P. M., Langendorf, S. J., Haggerty, C. C., et al. (2018). PlasmaPy: An open source community-developed python package for plasma physics. https://doi.org/10.5281/zenodo.1238132
Powell, K. G., Roe, P. L., Linde, T. J., Gombosi, T. I., & De Zeeuw, D. L. (1999). A solution-adaptive upwind scheme for ideal magnetohydrodynamics. *Journal of Computational Physics*, *154*(2), 284–309.
Reigber, C., Lühr, H., & Schwintzer, P. (2002). Champ mission status. *Advances in Space Research*, *30*(2), 129–134.
Rhodes, B. C. (2008). Pyephem home page. http://rhodesmill.org/pyephem/index.html
Richmond, A. D. (1995). Ionospheric electrodynamics using magnetic apex coordinates. *Journal of Geomagnetism and Geoelectricity*, *47*(2), 191–212.
Ridley, A. J., Deng, Y., & Tóth, G. (2006). The global ionosphere-thermosphere model. *Journal of Atmospheric and Solar-Terrestrial Physics*, *68*(8), 839–864.
Ridley, A. J., Gombosi, T. I., & DeZeeuw, D. L. (2004). Ionospheric control of the magnetosphere: Conductance. *Annales Geophysicae*, *22*(2), 567–584.
Ritchie, D. M., Bergin, T. J. Jr., & Gibson, R. G. Jr. (1996). *History of programming languages—II*. New York, NY: ACM. https://doi.org/10.1145/234286.1057834
SciPy developers (2018). SciPy: Open source scientific tools for Python. https://www.scipy.org/scikits.html
Shamir, L., Wallin, J. F., Allen, A., Berriman, B., Teuben, P., Nemiroff, R. J., et al. (2013). Practices in source code sharing in astrophysics. *Astronomy and Computing*, *1*, 54–58.
Shepherd, S. G. (2014). Altitude—adjusted corrected geomagnetic coordinates: Definition and functional approximations. *Journal of Geophysical Research: Space Physics*, *119*, 7501–7521. https://doi.org/10.1002/2014JA020264
Smith, A. (2018). http://joss.theoj.org/
Stansby, D., Yatharth, & Shaw, S (2018). heliopython/heliopy: Heliopy 0.5.2. https://doi.org/10.5281/zenodo.1009079
Sterne, K. T., Burrell, A. G., Reimer, A. S., Schmidt, M., Kotyk, K., DeLarquier, S., et al. (2017). AACGM_v2 upgrade, convection map updates. https://doi.org/10.5281/zenodo.1288637
Stodden, V., Hurlin, C., & Perignon, C. (2012). RunMyCode.org: A novel dissemination and collaboration platform for executing published, SSRN.
Stone, E. C., Frandsen, A. M., Mewaldt, R. A., Christian, E. R., Margolies, D., Ormes, J. F., & Snow, F. (1998). The advanced composition explorer. *Space Science Reviews*, *86*(1-4), 1–22.
Stoneback, R. A. (2018). pysatmagvect: Nasa icon ion velocity meter support (version 0.2.0). https://doi.org/10.5281/zenodo.1299374
Stoneback, R. A., Burrell, A. G., & Klenzing, J. (2017). Snakes on a spaceship: 2 fast 2 furious. http://cedarweb.vsp.ucar.edu/wiki/index.php/2017_Workshop:Python_for_Space_Science
Stoneback, R. A., Burrell, A. G., Klenzing, J., & Depew, M. D. (2018). Pysat: Python satellite data analysis toolkit. *Journal of Geophysical Research: Space Physics*, *123*. https://doi.org/10.1029/2018JA025297
Stoneback, R. A., Depew, M. D., & Iyer, G. S. (2018). pysatcdf: Windows compatibility and improved pysat meta handling (version 0.3.0). https://doi.org/10.5281/zenodo.1217181
Stoneback, R. A., Spence, C., Depew, M. D., Hargrave, N., Burrell, A. G., Klenzing, J., et al. (2018). rstoneback/pysat: Constellation support, satellite simulations, and DMSP IVM support. https://doi.org/10.5281/zenodo.1306979
Su, S. Y., Yeh, H. C., Heelis, R. A., Wu, J., Yang, S. C., Lee, L., & Chen, H. L. (1999). The ROCSAT-1 preliminary results: Low-latitude ionospheric plasma and flow variations. *Terrestrial, Atmospheric and Oceanic Sciences*, *10*(4), 787–804.
Sun, Q., Miao, C., Duan, Q., Ashouri, H., Sorooshian, S., & Hsu, K. L. (2018). A review of global precipitation data sets: Data sources, estimation, and intercomparisons. *Reviews of Geophysics*, *56*, 79–107. https://doi.org/10.1002/2017RG000574
SunPy Community, T., Mumford, S. J., Christe, S., Pérez-Suárez, D., Ireland, J., Shih, A. Y., Inglis, A. R., et al. (2015). SunPy—Python for solar physics. *Computational Science and Discovery*, *8*(1), 014009.
SuperDARN Data Analysis Working Group. Participating members, Thomas, E. G., Ponomarenko, P. V., Bland, E. C., Burrell, A. G., Kotyk, K., Shepherd, S. G., et al. (2018). Superdarn radar software toolkit (rst) 4.1. https://doi.org/10.5281/zenodo.1143675
TIOBE Index (2018). https://www.tiobe.com/tiobe-index/
TRAVIS CI, GMBH (2018). Travis CI—Test and deploy with confidence. https://travis-ci.org/
The BSD license (2018). https://opensource.org/licenses/bsd-license.php
The MIT license (2018). https://opensource.org/licenses/mit-license.php
Thébault, E., Finlay, C. C., & Toh, H. (2015). Special issue "international geomagnetic reference field—The twelfth generation". *Earth, Planets and Space*, *67*(1), 158. https://doi.org/10.1186/s40623-015-0313-0
Toffoletto, F., Sazykin, S., Spiro, R., & Wolf, R. (2003). Inner magnetospheric modeling with the rice convection model. *Space Science Reviews*, *107*(1), 175–196. https://doi.org/10.1023/A:1025532008047
Tyfield, D. (2013). Transition to science 2.0: "Remoralizing" the economy of science. *Spontaneous Generations: A Journal for the History and Philosophy of Science*, *7*(1), 29–48.
van Rossum, G. (1995). Python tutorial, Centrum voor Wiskunde en Informatica (CWI), Amsterdam CS-R9526.
van Rossum, G., Warsaw, B., & Coghlan, N. (2013). PEP 8—Style guide for Python code. https://www.python.org/dev/peps/pep-0008/
van der Meeren, C., Burrell, A. G., & Laundal, K. M. (2018). apexpy: Apexpy version 1.0.3. https://doi.org/10.5281/zenodo.1214207
von Alfthan, S., Pokhotelov, D., Kempf, Y., Hoilijoki, S., Honkonen, I., Sandroos, A., & Palmroth, M. (2014). Vlasiator: First global hybrid-Vlasov simulations of Earth's foreshock and magnetosheath. *Journal of Atmospheric and Solar-Terrestrial Physics*, *120*, 24–35.
Woods, T. N., Eparvier, F. G., Bailey, S. M., Chamberlin, P. C., Lean, J., Rottman, G. J., et al. (2005). Solar EUV Experiment (SEE): Mission overview and first results. *Journal of Geophysical Research*, *110*, A01312. https://doi.org/10.1029/2004JA010765
Xephem 3.7.7 (2018). http://www.clearskyinstitute.com/xephem/