The Astronomical Journal, 156:123 (19pp), 2018 September https://doi.org/10.3847/1538-3881/aabc4f

The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package*

The Astropy Collaboration, A. M. Price-Whelan[1], B. M. Sipőcz[44], H. M. Günther[2], P. L. Lim[3], S. M. Crawford[4], S. Conseil[5], D. L. Shupe[6], M. W. Craig[7], N. Dencheva[3], A. Ginsburg[8], J. T. VanderPlas[9], L. D. Bradley[3], D. Pérez-Suárez[10], M. de Val-Borro[11]
(Primary Paper Contributors),
T. L. Aldcroft[12], K. L. Cruz[13,14,15,16], T. P. Robitaille[17], E. J. Tollerud[3]
(Astropy Coordination Committee),
and
C. Ardelean[18], T. Babej[19], Y. P. Bach[20], M. Bachetti[21], A. V. Bakanov[98], S. P. Bamford[22], G. Barentsen[23], P. Barmby[18], A. Baumbach[24], K. L. Berry[98], F. Biscani[25], M. Boquien[26], K. A. Bostroem[27], L. G. Bouma[1], G. B. Brammer[3], E. M. Bray[98], H. Breytenbach[4,28], H. Buddelmeijer[29], D. J. Burke[12], G. Calderone[30], J. L. Cano Rodríguez[98], M. Cara[3], J. V. M. Cardoso[23,31,32], S. Cheedella[33], Y. Copin[34,35], L. Corrales[36,99], D. Crichton[37], D. D'Avella[3], C. Deil[38], É. Depagne[4], J. P. Dietrich[39,40], A. Donath[38], M. Droettboom[3], N. Earl[3], T. Erben[41], S. Fabbro[42], L. A. Ferreira[43], T. Finethy[98], R. T. Fox[98], L. H. Garrison[12], S. L. J. Gibbons[44], D. A. Goldstein[45,46], R. Gommers[47], J. P. Greco[1], P. Greenfield[3], A. M. Groener[48], F. Grollier[98], A. Hagen[49,50], P. Hirst[51], D. Homeier[52], A. J. Horton[53], G. Hosseinzadeh[54,55], L. Hu[56], J. S. Hunkeler[3], Ž. Ivezić[57], A. Jain[58], T. Jenness[59], G. Kanarek[3], S. Kendrew[60], N. S. Kern[45], W. E. Kerzendorf[61], A. Khvalko[98], J. King[38], D. Kirkby[62], A. M. Kulkarni[63], A. Kumar[64], A. Lee[65], D. Lenz[66], S. P. Littlefair[67], Z. Ma[68], D. M. Macleod[69], M. Mastropietro[70], C. McCully[54,55], S. Montagnac[71], B. M. Morris[57], M. Mueller[72], S. J. Mumford[73], D. Muna[74], N. A. Murphy[12], S. Nelson[7], G. H. Nguyen[75], J. P. Ninan[50], M. Nöthe[76], S. Ogaz[3], S. Oh[1], J. K. Parejko[57], N. Parley[77], S. Pascual[78], R. Patil[98], A. A. Patil[79], A. L. Plunkett[80], J. X. Prochaska[81], T. Rastogi[98], V. Reddy Janga[82], J. Sabater[83], P. Sakurikar[84], M. Seifert[98], L. E. Sherbert[3], H. Sherwood-Taylor[98], A. Y. Shih[85], J. Sick[86], M. T. Silbiger[98], S. Singanamalla[87], L. P. Singer[88,89], P. H. Sladen[90], K. A. Sooley[98], S. Sornarajah[98], O. Streicher[91], P. Teuben[92], S. W. Thomas[44], G. R. Tremblay[12], J. E. H. Turner[93], V. Terrón[94], M. H. van Kerkwijk[95], A. de la Vega[37], L. L. Watkins[3], B. A. Weaver[96], J. B. Whitmore[97], J. Woillez[61], and V. Zabalza[98]
(Astropy Contributors)

[1] Department of Astrophysical Sciences, Princeton University, Princeton, NJ 08544, USA; coordinators@astropy.org
[2] Kavli Institute for Astrophysics and Space Research, Massachusetts Institute of Technology, 70 Vassar St., Cambridge, MA 02139, USA
[3] Space Telescope Science Institute, 3700 San Martin Dr., Baltimore, MD 21218, USA
[4] South African Astronomical Observatory, P.O. Box 9, Observatory 7935, Cape Town, South Africa
[5] Univ Lyon, Univ Lyon1, Ens de Lyon, CNRS, Centre de Recherche Astrophysique de Lyon UMR5574, F-69230, Saint-Genis-Laval, France
[6] Caltech/IPAC, 1200 E. California Blvd, Pasadena, CA 91125, USA
[7] Department of Physics and Astronomy, Minnesota State University Moorhead, 1104 7th Ave S, Moorhead, MN 56563, USA
[8] National Radio Astronomy Observatory, 1003 Lopezville Rd, Socorro, NM 87801, USA
[9] eScience Institute, University of Washington, 3910 15th Ave NE, Seattle, WA 98195, USA
[10] University College London/Research IT Services, Gower St, Bloomsbury, London WC1E 6BT, UK
[11] Astrochemistry Laboratory, NASA Goddard Space Flight Center, 8800 Greenbelt Road, Greenbelt, MD 20771, USA
[12] Harvard-Smithsonian Center for Astrophysics, 60 Garden St., Cambridge, MA, 02138, USA
[13] Department of Physics and Astronomy, Hunter College, City University of New York, 695 Park Avenue, New York, NY 10065, USA
[14] Physics, Graduate Center of the City University of New York, New York, NY, USA
[15] Department of Astrophysics, American Museum of Natural History, New York, NY, USA
[16] Center for Computational Astropyhsics, Flatiron Institute, 162 Fifth Avenue, New York, NY 10010, USA
[17] Aperio Software Ltd., Headingley Enterprise and Arts Centre, Bennett Road, Leeds, LS6 3HN, UK
[18] Department of Physics & Astronomy, University of Western Ontario, 1151 Richmond St, London ON N5X4H1 Canada
[19] Department of Theoretical Physics & Astrophysics, Masaryk University, Kotlarska 2, 61137 Brno, Czech Republic
[20] Department of Physics and Astronomy, Seoul National University, Gwanak-gu, Seoul 08826, Republic of Korea
[21] INAF-Osservatorio Astronomico di Cagliari, via della Scienza 5, I-09047, Selargius, Italy
[22] School of Physics & Astronomy, University of Nottingham, University Park, Nottingham NG7 2RD, UK
[23] NASA Ames Research Center, Moffett Field, CA 94043, USA
[24] Heidelberg University, Kirchhoff Institut for Physics, Im Neuenheimer Feld 227, D-69116 Heidelberg, Germany
[25] Max-Planck-Institut für Astronomie, Königstuhl 17, D-69117 Heidelberg, Germany
[26] Unidad de Astronomía Fac. Cs. Básicas, Universidad de Antofagasta, Avda.U. de Antofagasta 02800, Antofagasta, Chile
[27] Department of Physics, UC Davis, 1 Shields Ave, Davis, CA, 95616, USA
[28] Department of Astronomy, University of Cape Town, Private Bag X3, Rondebosch 7701, South Africa
[29] Leiden Observatory, Leiden University, P.O. Box 9513, 2300 RA, Leiden, The Netherlands
[30] Istituto Nazionale di Astrofisica, via Tiepolo 11 Trieste, Italy
[31] Universidade Federal de Campina Grande, Campina Grande, PB 58429-900, Brazil
[32] Bay Area Environmental Research Institute, Petaluma, CA 94952, USA
[33] Department of Physics, Virginia Tech, Blacksburg, VA 24061, USA
[34] Université de Lyon, F-69622, Lyon, France

* The author list has three parts: the authors that made significant contributions to the writing of the paper in order of contribution, the four members of the Astropy Project coordination committee, and contributors to the Astropy Project in alphabetical order. Their position in the author list does not correspond to contributions to the Astropy Project as a whole. A more complete list of contributors to the core package can be found in the package repository and at the Astropy team webpage.

[35] Université de Lyon 1, Villeurbanne; CNRS/IN2P3, Institut de Physique Nucléaire de Lyon, France
[36] University of Wisconsin—Madison, 475 North Charter Street, Madison, WI 53706, USA
[37] Department of Physics and Astronomy, Johns Hopkins University, Baltimore, MD 21218, USA
[38] Max-Planck-Institut für Kernphysik, P.O. Box 103980, D-69029 Heidelberg, Germany
[39] Faculty of Physics, Ludwig-Maximilians-Universität, Scheinerstr. 1, D-81679 Munich, Germany
[40] Excellence Cluster Universe, Boltzmannstr. 2, D-85748 Garching b. München, Germany
[41] Argelander-Institut für Astronomie, Auf dem Hügel 71, D-53121 Bonn, Germany
[42] National Research Council Herzberg Astronomy & Astrophysics, 4071 West Saanich Road, Victoria, BC, Canada
[43] Instituto de Matemática Estatística e Física—IMEF, Universidade Federal do Rio Grande—FURG, Rio Grande, RS 96203-900, Brazil
[44] Institute of Astronomy, University of Cambridge, Madingley Road, Cambridge, CB3 0HA, UK
[45] Department of Astronomy, UC Berkeley, 501 Campbell Hall #3411, Berkeley, CA 94720, USA
[46] Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, USA
[47] Scion, Private Bag 3020, Rotorua, New Zealand
[48] Drexel University, Physics Department, Philadelphia, PA 19104, USA
[49] Vizual.ai, 3600 O'Donnell St, Suite 250, Baltimore, MD 21224, USA
[50] Dept of Astronomy and Astrophysics, Pennsylvania State University, University Park, PA 16802, USA
[51] Gemini Observatory, 670 N. Aohoku Pl, Hilo, HI 96720, USA
[52] Zentrum für Astronomie der Universität Heidelberg, Landessternwarte, Königstuhl 12, D-69117 Heidelberg, Germany
[53] Australian Astronomical Observatory, 105 Delhi Road, North Ryde NSW 2113, Australia
[54] Las Cumbres Observatory, 6740 Cortona Drive, Suite 102, Goleta, CA 93117-5575, USA
[55] Department of Physics, University of California, Santa Barbara, CA 93106-9530, USA
[56] Imperial College London, Kensington, London SW7 2AZ, UK
[57] Department of Astronomy, University of Washington, Seattle, WA 98155, USA
[58] BITS PILANI/Computer Science, Pilani Campus, Rajasthan, India
[59] Large Synoptic Survey Telescope, 950 N. Cherry Ave., Tucson, AZ, 85719, USA
[60] European Space Agency, Space Telescope Science Institute, 3700 San Martin Dr., Baltimore, MD 21218, USA
[61] European Southern Observatory, Karl-Schwarzschild-Straße 2, D-85748 Garching bei München, Germany
[62] Department of Physics and Astronomy, University of California, Irvine, CA 92697, USA
[63] College of Engineering Pune/Department of Computer Engineering and IT, Shivajinagar, Pune 411005, India
[64] Delhi Technological University, India
[65] Department of Physics, University of Berkeley, California, CA94709, USA
[66] Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109, USA
[67] Department of Physics & Astronomy, University of Sheffield, Sheffield, S3 7RH, UK
[68] Department of Physics and Astronomy, University of Missouri, Columbia, Missouri, 65211, USA
[69] Cardiff University, Cardiff CF24 3AA, UK
[70] Department of Physics and Astronomy, Ghent University, Krijgslaan 281, S9, B-9000 Gent, Belgium
[71] Puy-Sainte-Réparade Observatory, France
[72] Department of Mathematics, Brown University, 151 Thayer Street, Providence, RI 02912, USA
[73] SP^{2}RC, School of Mathematics and Statistics, The University of Sheffield, UK
[74] Center for Cosmology and Astroparticle Physics, The Ohio State University, 191 West Woodruff Avenue, Columbus, OH 43210, USA
[75] VNU-HCMC, University of Natural Sciences/Faculty of IT, 227 Nguyen Van Cu St., Ward 4, District 5, Ho Chi Minh City, Vietnam
[76] Experimental Physics 5, TU Dortmund, Otto-Hahn-Str. 4, D-44227 Dortmund, Germany
[77] University of Reading, Whiteknights Campus, Reading RG6 6BX, UK
[78] Departamento de Astrofisica, Universidad Complutense de Madrid, Madrid, Spain
[79] Pune Institute of Computer Technology, Pune 411043, India
[80] European Southern Observatory, Av. Alonso de Córdova 3107, Vitacura, Santiago, Chile
[81] Astronomy & Astrophysics, UC Santa Cruz, 1156 High St., Santa Cruz, CA 95064 USA
[82] Indian Institute of Technology, Mechanical Engineering, Kharagpur, India
[83] Institute for Astronomy (IfA), University of Edinburgh, Royal Observatory, Blackford Hill, EH9 3HJ Edinburgh, UK
[84] IIIT-Hyderabad, India
[85] NASA Goddard Space Flight Center, 8800 Greenbelt Road, Greenbelt, MD 20771, USA
[86] AURA/LSST, 950 N Cherry Ave, Tucson, 85719, USA
[87] Microsoft Research, Redmond, WA 98053, USA
[88] Astroparticle Physics Laboratory, NASA Goddard Space Flight Center, 8800 Greenbelt Road, Greenbelt, MD 20771, USA
[89] Joint Space-Science Institute, University of Maryland, College Park, MD 20742, USA
[90] Zentrum für Astronomie der Universität Heidelberg, Astronomisches Rechen-Institut, Mönchhofstraße 12-14, D-69120 Heidelberg, Germany
[91] Leibniz Institute for Astrophysics Potsdam (AIP), An der Sternwarte 16, D-14482 Potsdam, Germany
[92] Astronomy Department, University of Maryland, College Park, MD 20742, USA
[93] Gemini Observatory, Casilla 603, La Serena, Chile
[94] Institute of Astrophysics of Andalusia (IAA-CSIC), Granada, Spain
[95] Department of Astronomy & Astrophysics, University of Toronto, 50 Saint George Street, Toronto, ON M5S 3H4, Canada
[96] National Optical Astronomy Observatory, 950 N. Cherry Ave., Tucson, AZ 85719, USA
[97] Centre for Astrophysics and Supercomputing, Swinburne University of Technology, Hawthorn, VIC 3122, Australia

Received 2018 January 8; revised 2018 March 27; accepted 2018 March 29; published 2018 August 24

[98] The Astropy Project.
[99] Einstein Fellow.

Abstract

The Astropy Project supports and fosters the development of open-source and openly developed Python packages that provide commonly needed functionality to the astronomical community. A key element of the Astropy Project is the core package astropy, which serves as the foundation for more specialized projects and packages. In this article, we provide an overview of the organization of the Astropy project and summarize key features in the core package, as of the recent major release, version 2.0. We then describe the project infrastructure designed to facilitate and support development for a broader ecosystem of interoperable packages. We conclude with a future outlook of planned new features and directions for the broader Astropy Project.

Key words: methods: data analysis – methods: miscellaneous – methods: statistical – reference systems

1. Introduction

All modern astronomical research makes use of software in some way. Astronomy, as a field, has thus long supported the development of software tools for astronomical tasks, such as scripts that enable individual scientific research, software packages for small collaborations, and data reduction pipelines for survey operations. Some software packages are, or were, supported by large institutions and intended for a wide range of users. These packages therefore typically provide some level of documentation and user support or training. Other packages are developed by individual researchers or research groups, are then typically used by smaller groups for more domain-specific purposes, and may have less extensive user support. For both packages meant for wider distribution and for scripts specific to particular research projects, a library that addresses common astronomical tasks simplifies the software development process by encouraging the reuse of common functions. The users of such a library then also benefit from a community and ecosystem built around a shared foundation. The Astropy project has grown to become such a community for Python astronomy software, and the astropy core package has become this shared foundation.

The development of the astropy core package began as a largely community-driven effort to standardize core functionality for astronomical software in Python. In this way, its genesis differs from, but builds upon, many substantial and former astronomical software development efforts that were commissioned or initiated through large institutional support, such as IRAF (developed at NOAO; Tody 1993), MIDAS (developed at ESO; Banse et al. 1988), or Starlink (originally developed by a consortium of UK institutions and now maintained by the East Asian Observatory; Disney & Wallace 1982; Currie et al. 2014). More recently, community-driven efforts have seen significant success in the astronomical sciences (e.g., Turk et al. 2011).

Python[100] is an increasingly popular, general-purpose programming language that is available under a permissive open-source software license and is free of charge for all major operating systems. This programming language has become especially popular in the quantitative sciences, where researchers must simultaneously conduct research, perform data analysis, and develop software. A large part of this success owes itself to the vibrant community of developers and a continuously growing ecosystem of tools, web services, and stable, well-developed packages that enable easier collaboration on software development, easier writing and sharing of software documentation, and continuous testing and validation of software. While dedicated libraries provide support for array representation and arithmetic (numpy; Van der Walt et al. 2011), a wide variety of functions for scientific computing (scipy; Jones et al. 2001), and publication-quality plotting (matplotlib; Hunter 2007), tens of thousands of other high-quality and easy-to-use packages are available to help with tasks that are not specific to astronomy but might be performed in the course of astronomical research, e.g., interfacing with databases, or statistical inferences. More recently, the development and mainstream adoption of package managers such as Anaconda[101] has significantly streamlined the installation process for many libraries, lowering the barriers to entry for new users.

The Astropy Project aims to provide an open-source and open-development core package (astropy) and an ecosystem of *affiliated packages* that support astronomical functionality in the Python programming language. "Open development" describes a process where anybody with an internet connection can suggest changes to the source code and contribute their opinion when new features, bug fixes or other code changes, governance, or any other aspect of the development process is discussed (see Sections 2.2 and 2.3 of how this is organized in practice). The astropy core package is now a feature-rich library of sufficiently general tools and classes that supports the development of more specialized code. An example of such functionality is reading and writing FITS files: it would be time-consuming and impractical for multiple groups to implement the FITS standard (Pence et al. 2010) and maintain software for such a general-purpose need. Another example of such a common task is in dealing with representations of and transformations between astronomical coordinate systems.

The Astropy Project aims to develop and provide high-quality code and documentation according to the best practices in software development. The project makes use of different tools and web services to reach those goals without central institutional oversight. The first public release of the astropy package is described in Astropy Collaboration et al. (2013). Since then, the astropy package has been used in hundreds of projects and the scope of the package has grown considerably. At the same time, the scientific community contributing to the project has grown tremendously and an ecosystem of packages supporting or affiliated with the astropy core has developed. In this paper, we describe the current status of the Astropy community and the astropy core package and discuss goals for future development.

We start by describing the way the Astropy Project functions and is organized in Section 2. We then describe the main software efforts developed by the Astropy Project itself: a core package called astropy (Section 3) and several separate packages that help maintain the infrastructure for testing and documentation (Section 4). We end with a short vision for the

[100] https://www.python.org/

[101] https://anaconda.org/

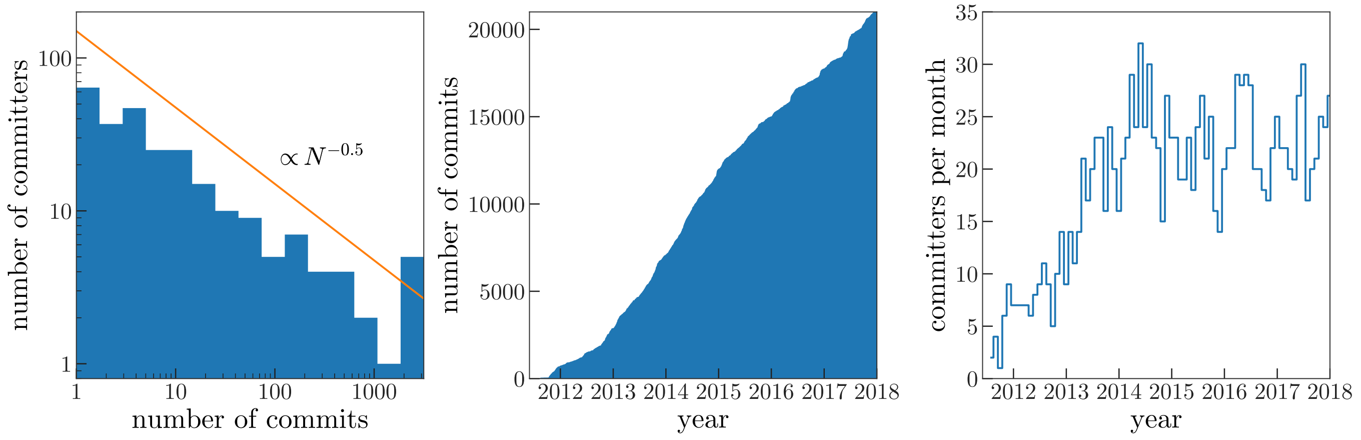

Figure 1. Left panel: distribution of number of commits per committer. Middle panel: cumulative number of commits to the `astropy` core package over time. Right panel: number of unique committers per month to the `astropy` core package.

future of Astropy and astronomical software in general in Section 5. The full paper, including the code to produce the figures, is available in a `GitHub` repository (Price-Whelan et al. 2018).[102]

This article is not intended as an introduction to `astropy`, nor does it replace the `astropy` documentation. Instead, it describes the way the Astropy community is organized and the current state of the `astropy` package.

2. Organization and Infrastructure

2.1. Coordination of Astropy

From its inception, Astropy has required coordination to ensure the project as a whole and its coding efforts are consistent and reasonably efficient. While many `Python` projects adopt a "Benevolent Dictator For Life" (BDFL) model, Astropy has instead opted for a *coordination committee*. This is partly due to the nature of the project as a large-scale collaboration between many contributors with many interests, and partly due to the sheer amount of work that needs to get done. For the latter reason, the project has expanded the committee from three to four members starting in 2016.

For resolving disagreements about the `astropy` core package or other Astropy-managed code, the coordination committee primarily acts to work toward consensus, or when consensus is difficult to achieve, generally acts as a "tie-breaker." The committee also oversees affiliated package applications to ensure that they are in keeping with Astropy's vision and philosophy,[103] as well as the associated procedures. Additionally, the committee oversees the assignment of roles (primarily driven by already-existing contributions), and increasingly has acted as the "face" of the Project, providing contact with organizations like NumFOCUS (the body that holds any potential funding in trust for Astropy) and the American Astronomical Society (AAS).

2.2. Astropy Development Model

Code is contributed to the `astropy` core package or modified through "pull requests" (via `GitHub`[104]) that often contain several `git` commits. Pull requests may fix bugs, implement new features, or improve or modify the infrastructure that supports the development and maintenance of the package. Individual pull requests are generally limited to a single conceptual addition or modification, to make code review tractable. Pull requests that modify or add code to a specific subpackage must be reviewed and approved by one of the subpackage maintainers before they are merged into the core codebase. Bugs and feature requests are reported via the `GitHub` issue tracker and labeled with a set of possible labels that help classify and organize the issues. The development workflow is detailed in the `astropy` documentation.[105]

As of version 2.0, `astropy` contains 212,244 lines of code[106] contributed by 232 unique contributors over 19,270 `git` commits. Figure 1, left, shows the distribution of total number of commits per contributor as of early 2018. The relative flatness of this distribution (as demonstrated by its log-log slope of −0.5) shows that the `astropy` core package has been developed by a broad contributor base. A leading group of six developers have each added over 1000 commits to the repository, and ∼20 more core contributors have contributed at least 100 commits. However, the distribution of contribution level (number of commits) continues from 100 down to a single commit. In this sense, the development of the core package has been a true community effort and is not dominated by a single individual. It is also important to note that the number of commits is only a rough metric of contribution, as any single commit could be a critical fix in the package or merely a fix for a typographical error. Figure 1, middle, shows the number of commits as a function of time since the genesis of the `astropy` core package. The package is still healthy: new commits are and have been contributed at a steady rate throughout its existence. Figure 1, right, shows that, ≈20–25 people contribute to the core package each month. While we would like for this number to grow, this demonstrates that the core package is still being developed and maintained by a substantial group of contributors.

2.3. APEs—Astropy Proposals for Enhancement

The Astropy project has a formal mechanism to propose significant changes to the core package (e.g., re-writing the

[102] Codebase: https://github.com/astropy/astropy-v2.0-paper.
[103] http://docs.astropy.org/en/stable/development/vision.html
[104] https://github.com/astropy/astropy/
[105] *How to make a code contribution*, http://docs.astropy.org/en/latest/development/workflow/development_workflow.html.
[106] This line count includes comments, as these are often as important for maintainability as the code itself. Without comments there are 142,197 lines of code.

coordinates subpackage; Tollerud et al. 2014), to plan out major new features (e.g., a new file format; Aldcroft 2015), or to institute new organization-wide policies (e.g., adopting a code of conduct; Cruz et al. 2015). This mechanism is called “Astropy Proposal for Enhancement” (APE) and is modeled after the “Python Enhancement Proposals” (PEP) that guide the development of the `Python` programming language. In an APE, one or more authors describe in detail the proposed changes or additions, including a rationale for the changes, how these changes will be implemented, and in the case of code, what the interface will be (Greenfield 2013). The APEs are discussed and refined by the community before much work is invested into a detailed implementation; anyone is welcome to contribute to these discussions during the open consideration period. APEs are proposed via pull requests on a dedicated GitHub repository[107]; therefore, anyone can read the proposed APEs and leave in-line comments. When a community consensus emerges, the APEs are accepted and become the basis for future work. In cases where consensus cannot be reached, the Astropy coordination committee may decide to close the discussion and make an executive decision based on the community input on the APE in question.

2.4. Concept of Affiliated Packages

A major part of the Astropy Project is the concept of “Affiliated Packages.” An affiliated package is an astronomy-related `Python` package that is not part of the `astropy` core package, but has requested to be included as part of the Astropy Project’s community. These packages support the goals and vision of Astropy of improving code re-use, interoperability, and embracing good coding practices such as testing and thorough documentation.

Affiliated packages contain functionality that is more specialized, have license incompatibilities, or have external dependencies (e.g., GUI libraries) that make these packages more suitable to be separate from the `astropy` core package. Affiliated packages may also be used to develop substantial new functionality that will eventually be incorporated into the `astropy` core package (e.g., astropy.visualization.wcsaxes). New functionality benefits from having a rapid development and release cycle that is not tied to that of the `astropy` core (Section 2.5). These projects may also have less stringent requirements for style, testing, or development as compared to the core package.

Affiliated packages are listed on the main Astropy website and advertised to the community through Astropy mailing lists; a list of current affiliated packages is included in Table 1. Becoming an affiliated package is a good way for new and existing packages to gain exposure while promoting Astropy’s high standard for code and documentation quality. This process of listing and promoting affiliated packages is one way in which the Astropy Project tries to increase code re-use in the astronomical community.

Packages can become affiliated with Astropy by applying for this status on a public mailing list. The coordination committee (Section 2.1) reviews such requests and issues recommendations for the improvement of a package, where applicable.

[107] https://github.com/astropy/astropy-APEs

2.5. Release Cycle and Long-term Support

The `astropy` package has a regular release schedule consisting of new significant releases every six months, with bugfix releases as needed (Tollerud 2013). The major releases contain new features or any significant changes, whereas the bugfix releases only contain fixes to code or documentation but no new features. Some versions are additionally designated as “Long-term support” (LTS) releases, which continue to receive bug fixes for two years following the release with no changes to the API. The LTS versions are ideal for pipelines and other applications in which API stability is essential. The latest LTS release (version 2.0) is also the last one that supports `Python` 2; it will receive bug fixes until the end of 2019 (Robitaille 2017).

The version numbering of the `astropy` core package reflects this release scheme: the core package version number uses the form `x.y.z`, where “x” is advanced for LTS releases, “y” for non-LTS feature releases, and “z” for bugfix releases. This is similar to Semantic Versioning.[108]

The released versions of the `astropy` core package are available from several of the `Python` distributions for scientific computing (e.g., Anaconda) and from the `Python` Package Index (PyPI).[109] Effort has been made to make `astropy` available and easily installable across all platforms; the package is constantly tested on different platforms as part of a suite of continuous integration tests.

2.6. Support of Astropy

The Astropy Project, as of the version 2.0 release, does not receive any direct financial support for the development of `astropy`. Development of the software, all supporting materials, and community support are provided by individuals who work on the Astropy Project in their own personal time, by individuals or groups contributing to Astropy as part of a research project, or contributions from institutions that allocate people to work on Astropy. A list of organizations that have contributed to Astropy in this manner can be found in the Acknowledgments.

Different funding models have been proposed for support of Astropy (e.g., Muna et al. 2016), but a long-term plan for sustainability has not yet been established. The Astropy Project has the ability to accept financial contributions from institutions or individuals through the NumFOCUS[110] organization. NumFOCUS has, to date, covered the direct costs incurred by the Astropy Project.

2.7. Reuse of the Scientific Python Ecosystem

The Astropy Project is built on a philosophy of building from the existing Python scientific ecosystem wherever possible. Many of the enabling technologies like `numpy`, `scipy`, `matplotlib`, or `cython`, are necessary as the core underpinnings of Astropy packages. Often, this means using these packages as “building blocks” (e.g., the ubiquitous use of `numpy` arrays throughout `astropy`). In other cases, this means wrappers around more general algorithms that make them more convenient for astronomy use cases (e.g., the model-fitting in the `astropy.modeling` subpackage

[108] https://semver.org

[109] See the installation documentation for more information: http://docs.astropy.org/en/stable/install.html.

[110] NumFOCUS is a 501(c)(3) nonprofit that supports and promotes world-class, innovative, open-source scientific computing.

discussed in Section 3.7). Sometimes these simple wrappers evolve into more complex implementations that address astronomically relevant use cases the general tool does not support (e.g., `astropy.convolution`, see Section 3.8). As a broad rule, the Project explicitly encourages re-use of code where possible.

However, the boundaries of when this re-use is called for is often ambiguous. Some of the examples above only evolved after significant debate in the community over whether these algorithms were sufficient. Other times, even apparently general functionality did not exist in the wider ecosystem that met the Astropy community's needs, and hence the functionality had to be developed wholly from the developer resources available in the community (e.g., `astropy.units`, or `astropy.table` when it began). At the same time, however, the Astropy Project has provided the wider community with myriad bug fixes to the enabling technologies listed above, as well as the testing and documentation architecture. Functionality is only "extracted" from Astropy to other packages after careful consideration that includes considering the impact on the maintenance and support of Astropy.

3. Astropy Core Package Version 2.0

The Astropy Project aims to provide `Python`-based packages for all tasks that are commonly needed in a large subset of the astronomical community. At the foundation is the `astropy` core package, which provides general functionality (e.g., coordinate transformations, reading and writing astronomical files, and units) or base classes for other packages to utilize for a common interface (e.g., NDData). In this section, we highlight new features introduced or substantially improved since version 0.2 (previously described in Astropy Collaboration et al. 2013). The `astropy` package provides a full log of changes[111] over the course of the entire project and more details about individual subpackages are available in the documentation.[112] Beyond what is mentioned below, most subpackages have seen improved performance since the release of the version 0.2 package.

3.1. Units

The astropy.units subpackage adds support for representing units and numbers with associated units—"quantities"—in code. Historically, quantities in code have often been represented simply as numbers, with units implied or noted via comments in the code because of considerations about speed: having units associated with numbers inherently adds overhead to numerical operations. In astropy.units, Quantity objects extend `numpy` array objects and have been designed with speed in mind.

As of `astropy` version 2.0, units and quantities, prevalent in most of its subpackages, have become a key concept for using the package as a whole. Units are intimately entwined in the definition of astronomical coordinates; thus, nearly all functionality in the astropy.coordinates subpackage (see Section 3.3) depends on them. Most `astropy` subpackages have been made compatible with `Quantity` objects, although they are not always required.

[111] https://github.com/astropy/astropy/blob/stable/CHANGES.rst
[112] http://docs.astropy.org/en/stable/

The motivation and key concepts behind this subpackage were described in detail in the previous paper (Astropy Collaboration et al. 2013). Therefore, we primarily highlight new features and improvements here.

3.1.1. Interaction with `numpy` Arrays

Quantity objects extend `numpy.ndarray` objects and therefore work well with many of the functions in `numpy` that support array operations. For example, Quantity objects with angular units can be directly passed in to the trigonometric functions implemented in `numpy`. The units are internally converted to radians, which is what the `numpy` trigonometric functions expect, before being passed to `numpy`.

3.1.2. Logarithmic Units and Magnitudes

By default, taking the logarithm of a Quantity object with non-dimensionless units intentionally fails. However, some well-known units are actually logarithmic quantities, where the logarithm of the value is taken with respect to some reference value. Examples include astronomical magnitudes, which are logarithmic fluxes, and decibels, which are more generic logarithmic ratios of quantities. Logarithmic, relative units are now supported in astropy.units.

3.1.3. Defining Functions that Require Quantities

When writing code or functions that expect Quantity objects, we often want to enforce that the input units have the correct physical type. For example, we may want to require only length-type Quantity objects. astropy.units provides a tool called quantity_input() that can perform this verification automatically to avoid repetitive code.

3.2. Constants

The astropy.constants subpackage provides a selection of physical and astronomical constants as Quantity objects (see Section 3.1). A brief description of this package was given in Astropy Collaboration et al. (2013). In version 2.0, the built-in constants have been organized into modules for specific versions of the constant values. For example, physical constants have `codata2014` (Mohr et al. 2016) and `codata2010` versions. Astronomical constants are organized into `iau2015` and `iau2012` modules to indicate their sources (resolutions from the International Astronomical Union, IAU). The `codata2014` and `iau2015` versions are combined into the default constant value version: `astropyconst20`. For compatibility with `astropy` version 1.3, `astropyconst13` is available and provides access to the adopted versions of the constants from earlier versions of `astropy`. To use previous versions of the constants as *units* (e.g., solar masses), the values have to be imported directly; with version 2.0, astropy.units uses the `astropyconst20` versions.

Astronomers using astropy.constants should take particular note of the constants provided for Earth, Jupiter, and the Sun. Following IAU 2015 Resolution B3 (Mamajek et al. 2015), nominal values are now given for mass parameters and radii. The nominal values will not change even as "current best estimates" are updated.

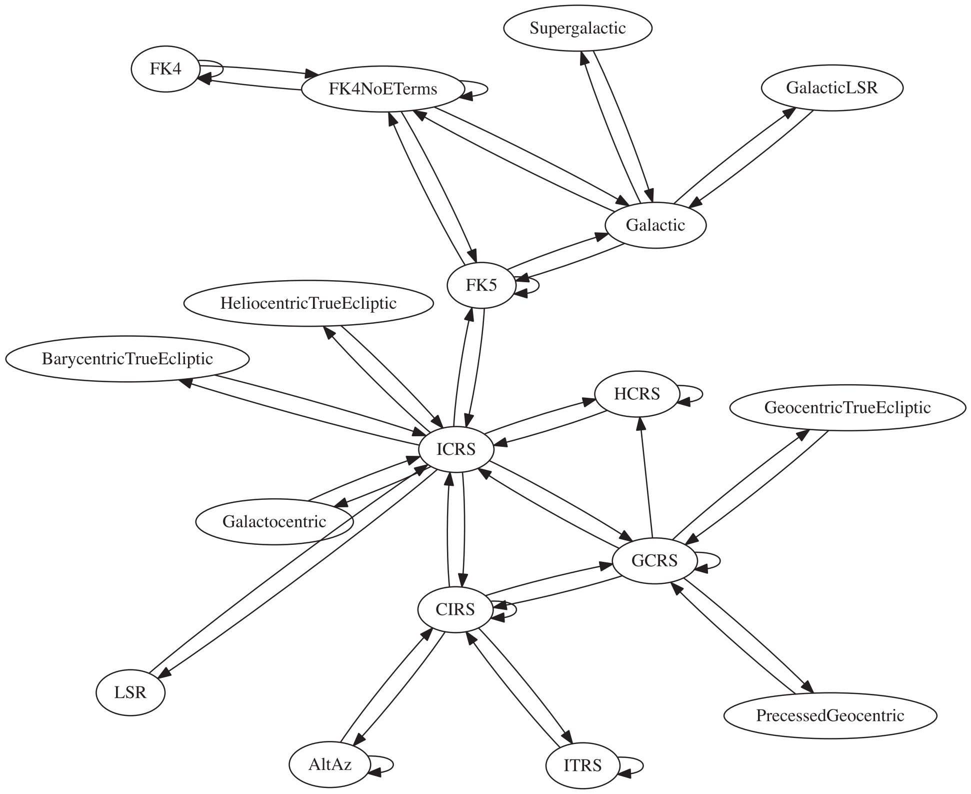

Figure 2. The full graph of possible reference frame transformations implemented in astropy.coordinates. Arrows indicate transformations from one frame to another. Arrows that point back to the same frame indicate self-transformations that involve a change of reference frame parameters (e.g., equinox).

3.3. Coordinates

The astropy.coordinates subpackage is designed to support representing and transforming celestial coordinates and—new in version 2.0—velocities. The framework heavily relies on the astropy.units subpackage, and most inputs to objects in this subpackage are expected to be Quantity objects. Some of the machinery also relies on the Essential Routines of Fundamental Astronomy (ERFA) `C` library for some of the critical underlying transformation machinery (Tollerud et al. 2017), which is based on the Standards Of Fundamental Astronomy (SOFA) effort (Hohenkerk 2011).

A key concept behind the design of this subpackage is that coordinate *representations* and *reference systems/frames* are independent of one another. For example, a set of coordinates in the International Celestial Reference System (ICRS) reference frame could be represented as spherical (right ascension, declination, and distance from solar system barycenter) or Cartesian coordinates (x, y, z with the origin at barycenter). They can therefore change representations independent of being transformed to other reference frames (e.g., the Galactic coordinate frame).

The classes that handle coordinate representations (the `Representation` classes) act like three-dimensional vectors and thus support vector arithmetic. The classes that represent reference systems and frames (the `Frame` classes) internally use `Representation` objects to store the coordinate data—that is, the `Frame` classes accept coordinate data, either as a specified `Representation` object, or using short-hand keyword arguments to specify the components of the coordinates. These preferred representation and short-hand component names differ between various astronomical reference systems. For example, in the ICRS frame, the spherical angles are right ascension (`ra`) and declination (`dec`), whereas in the Galactic frame, the spherical angles are Galactic longitude (`l`) and latitude (`b`). Each `Frame` class defines its own component names and preferred `Representation` class. The frame-specific component names map to corresponding components on the underlying `Representation` object that internally stores the coordinate data. For most frames, the preferred representation is spherical, although this is determined primarily by their common use in the astronomical community.

Many of the `Frame` classes also have attributes specific to the corresponding reference system that allow the user to specify the frame. For example, the Fifth Fundamental Catalogue (FK5) reference system requires specifying an equinox to determine the reference frame. If required, these additional frame attributes must be specified, along with the coordinate data, when a `Frame` object is created. Figure 2 shows the network of possible reference frame transformations as currently implemented in astropy.coordinates. Custom user-implemented `Frame` classes that define transformations to any reference frame in this graph can then be transformed to any of the other connected frames.

The typical user does not usually have to interact directly with the `Frame` or `Representation` classes. Instead, astropy.coordinates provides a high-level interface to representing astronomical coordinates through the SkyCoord class, which was designed to provide a single class that accepts a wide range of possible inputs. It supports coordinate data in any coordinate frame in any representation by internally using the `Frame` and `Representation` classes.

In what follows, we briefly highlight key new features in astropy.coordinates.

3.3.1. Local Earth Coordinate Frames

In addition to representing celestial coordinates, `astropy` now supports specifying positions on the Earth in a number of different geocentric systems with the `EarthLocation` class. With this, `astropy` now supports Earth-location-specific coordinate systems such as the altitude-azimuth (`AltAz`) or horizontal system. Transformations between `AltAz` and any Barycentric coordinate frame also requires specifying a time using the Time class from `astropy.time`. With this new functionality, many of the common tasks associated with observation planning can now be completed with `astropy` or the Astropy-affiliated package `astroplan` (Morris et al. 2018).

3.3.2. Proper Motion and Velocity Transformations

In addition to positional coordinate data, the `Frame` classes now also support velocity data. As the default representation for most frames is spherical, most of the `Frame` classes expect proper motion and radial velocity components to specify the velocity information. The names of the proper motion components all start with `pm` and adopt the same longitude and latitude names as the positional components. Transforming coordinates with velocity data is also supported, but in some cases the transformed velocity components have limited accuracy because the transformations are done numerically instead of analytically. The low-level interface for specifying and transforming velocity data is currently experimental. As such, in version 2.0, only the `Frame` classes (and not the SkyCoord class) support handling velocities.

3.3.3. Solar System Ephemerides

Also new is support for computing ephemerides of major solar system bodies and outputting the resulting positions as coordinate objects. These ephemerides can be computed either using analytic approximations from ERFA or from downloaded JPL ephemerides (the latter requires the `jplephem`[113] optional dependency and an internet connection).

3.3.4. Accuracy of Coordinate Transformations

In order to check the accuracy of the coordinate transformations in astropy.coordinates, we have created a set of benchmarks that we use to compare transformations between a set of coordinate frames for a number of packages.[114] Because no package can be guaranteed to implement all transformations to arbitrary precision and some transformations are sometimes subject to interpretation of standards (particularly in the case of Galactic coordinates), we do not designate any of the existing packages as the "ground truth" but instead compare each tool to all other tools. The benchmarks are thus useful beyond the Astropy Project because they allow all of the tools to be compared to all other tools. The tools included in the benchmark at the moment include the `astropy` core package, Kapteyn (Terlouw & Vogelaar 2015), NOVAS (Barron et al. 2011), PALpy (Jenness & Berry 2013), PyAST (a wrapper for AST, described in Berry et al. 2016), PyTPM,[115] PyEphem (Rhodes 2011), and pySLALIB (a Python wrapper for SLALIB, described in Wallace 1994).

The benchmarks are meant to evolve over time and include an increasing variety of cases. At the moment, the benchmarks are set up as follows—we have generated a standard set of 1000 pairs of random longitudes/latitudes that we use in all benchmarks. Each benchmark is then defined using an input and output coordinate frame, using all combinations of FK4, FK5, Galactic, ICRS, and Ecliptic frames. For now, we set the epoch of observation to J2000. We also set the frame to J2000 (for FK5 and Ecliptic) and B1950 (for FK4). In the future, we plan to include a larger variety of epochs and equinoxes, as well as tests of conversion to/from Altitude/Azimuth. For each benchmark, we convert the 1000 longitudes/latitudes from the input/output frame with all tools and quantify the comparison by looking at the median, mean, maximum, and standard deviation of the absolute separation of the output coordinates from each pair of tools.

Figure 3 visualizes the relative accuracy of the conversion from FK4 to Galactic coordinates for all pairs of tools that implement this transformation. In this figure, the color of the cell indicates the maximum difference (in arcseconds) between the two tools over the 1000 longitude-latitude pairs tested. This figure shows, for example, that `astropy`, Kapteyn, and PyTPM agree with sub-milliarcsecond differences (light colors, small differences), while PALpy, pySLALIB, and PyAST also agree among themselves. However, there is an offset of around $0\farcs2$ between the two groups. Finally, PyEphem disagrees with all other packages by $0\farcs4$–$0\farcs8$ (darker colors, large differences). These values are only meant to be illustrative and will change over time as the benchmarks are refined and the packages updated.

3.4. Time

The astropy.time subpackage focuses on supporting timescales (e.g., UTC, TAI, UT1) and time formats (e.g., Julian date, modified Julian date) that are commonly used in astronomy. This functionality is needed, for example, to calculate barycentric corrections or sidereal times. astropy.time is currently built on the ERFA (Tollerud et al. 2017) C library, which replicates the Standards of Fundamental Astronomy (SOFA; Hohenkerk 2011) but is licensed under a three-clause BSD license. The package was described in detail in Astropy Collaboration et al. (2013) and has remained stable for the last several versions of `astropy`. Thus, in what follows, we only highlight significant changes or new features since the previous Astropy paper.

3.4.1. Barycentric and Heliocentric Corrections

Detailed eclipse or transit timing requires accounting for light traveltime differences from the source to the observatory because of the Earth's motion. It is therefore common to instead convert times to the solar system barycenter or heliocenter where the relative timing of photons is standardized. With the location of a source on the sky (i.e., a SkyCoord object), the location of an observatory on Earth (i.e., an EarthLocation object), and time values as Time objects, the time corrections to shift to the solar system barycenter or heliocenter can now be computed with astropy.time using the `light_travel_time` method of a Time object.

[113] https://github.com/brandon-rhodes/python-jplephem
[114] http://www.astropy.org/coordinates-benchmark/summary.html
[115] https://github.com/phn/pytpm

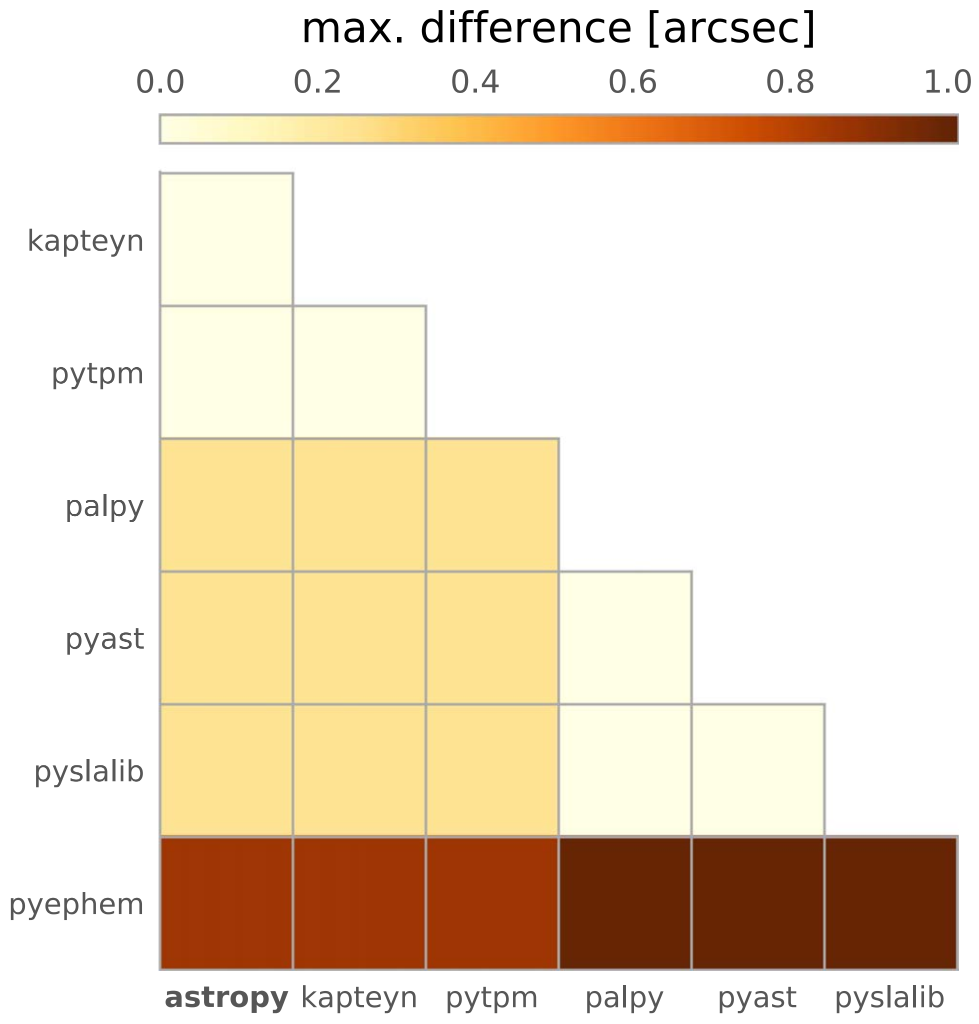

Figure 3. Comparison matrix of the maximum difference between longitude-latitude values in a set of 1000 random points transformed from FK4 to Galactic with the different packages. Darker colors (larger differences) are more significant disagreements.

3.5. Data Containers

3.5.1. `nddata`

The astropy.nddata subpackage provides three types of functionality: an abstract interface for representing generic arbitrary-dimensional data sets intended primarily for subclassing by developers of other packages, concrete classes building on this interface, and utilities for manipulating these kind of data sets.

The `NDDataBase` class provides the abstract interface for gridded data with attributes for accessing metadata, the world coordinate system (WCS), uncertainty arrays matched to the shape of the data, and other traits. Building on this interface, the `NDData` class provides a minimal working implementation for storing `numpy` arrays. These classes serve as useful base classes for package authors wishing to develop their own classes for specific use cases and as containers for exchanging gridded data.

The classes `NDDataRef`, `NDDataArray`, and `CCDData` extend the base storage functionality with options to do basic arithmetic (addition, subtraction, multiplication, and division), including error propagation in limited cases, and slicing of the data set based on grid coordinates that appropriately handles masking, errors, and units (if present). Additionally, the `CCDData` class also provides reading and writing from and to FITS files, and uses data structures from `astropy`, like `WCS`, to represent the file contents abstractly.

The http://docs.astropy.org/en/stable/nddata/utils.html `astropy.nddata.utils` module provides utilities that can operate on either plain `numpy` arrays or any of the classes in the astropy.nddata subpackage. It features a class for representing two-dimensional image cutouts, allowing one to easily link pixels in the cutout to those in the original image or vice versa, to convert between world and pixel coordinates in the cutout, and to overlay the cutout on images. Functions to enlarge or reduce an image by doing block replication or reduction are also provided.

3.5.2. Tables

The astropy.table subpackage provides functionality for representing and manipulating heterogeneous data. In some respects, this is similar to `numpy` record arrays (Van der Walt et al. 2011) or `pandas DataFrame` objects (McKinney 2010), but with modifications for astronomical data. Most notably, tables from astropy.table allow for table or column metadata and can handle vectors/arrays or arbitrary objects (with suitable column-like characteristics) as table entries. The subpackage was described in detail in Astropy Collaboration et al. (2013). Thus, in what follows, we only summarize key new features or updates to astropy.table since the previous Astropy paper. These are support for grouped table operations, table concatenation, and using array-valued `astropy` objects as table columns.

A table can contain data that naturally form groups; for example, it may contain multiple observations of a few sources at different points in time and in different bands. We may want to split the table into groups based on the combination of source observed and the band, after which we combine the results for each combination of source and band in some way (e.g., finding the mean or standard deviation of the fluxes or magnitudes over time) or filter the groups based on user-defined criteria. These kinds of grouping and aggregation operations are now fully supported by Table objects.

Table objects can now be combined in several different ways. If two tables have the same columns, we may want to stack them "vertically" to create a new table with the same columns but all rows. If two tables are row-matched but have distinct columns, we may want to stack them "horizontally" to create a new table with the same rows but all columns. For other situations, more generic table concatenation or joining are also possible when two tables share some columns.

The Table object now allows array-valued Quantity, celestial coordinate (SkyCoord), and date/time (Time) objects to be used as columns. It also provides a general way for other user-defined array-like objects to be used as columns. This makes it possible, for instance, to easily represent catalogs of sources or time series in Astropy, while having both the benefits of the Table object (e.g., accessing specific rows/columns or groups of them and combining tables) as well as, for example, the SkyCoord or the Time classes (e.g., converting the coordinates to a different frame or accessing the date/time in the desired timescale).

Additionally, there is now interoperability between `Table` and `pandas DataFrame` objects using the `Table.to_pandas()` and `Table.from_pandas()` methods. `Table.to_pandas()` is, however, limited to a subset of possible `Table` objects, because several of the features outlined above cannot be represented as `pandas` data frames. While these features might be contributed to or combined with `pandas` at a later date, the architectural differences that enable these areas are significant, so this would be a major undertaking that would require substantial investment from the core maintainers and communities of both packages.

3.6. *`io`*

The `astropy.io` subpackages provide support for reading and writing data to a variety of ASCII and binary file formats, such as a wide range of ASCII data table formats, FITS, HDF5, and VOTable. It also provides a unified interface for reading and writing data with these different formats using the astropy.table subpackage. For many common cases, this simplifies the process of file input and output (I/O) and reduces the need to master the separate details of all the I/O packages within `astropy`. The file interface allows transparent compression of the `gzip`, `bzip2`, and `lzma` (`.xz`) formats; the latter two require the Python installation to have been compiled with support the respective libraries.

3.6.1. *ASCII*

One of the problems when storing a table in an ASCII format is preserving table metadata such as comments, keywords and column data types, units, and descriptions. The newly defined *Enhanced Character Separated Values* (ECSV, Aldcroft 2015) format makes it possible to write a table to an ASCII-format file and read it back with no loss of information. The ECSV format has been designed to be both human-readable and compatible with most simple CSV readers.

The astropy.io.ascii subpackage now includes a significantly faster Cython/C engine for reading and writing ASCII files. This is available for most of the common formats. It also offers some additional features like parsing of different exponential notation styles, such as commonly produced by Fortran programs. On average, the new engine is about four to five times faster than the corresponding pure-`Python` implementation and is often comparable to the speed of the `pandas` (McKinney 2010) ASCII file interface. The fast reader has a parallel processing option that allows harnessing multiple cores for input parsing to achieve even greater speed gains. By default, `read()` and `write()` will attempt to use the fast Cython/C engine when dealing with compatible formats. Certain features of the full read / write interface are unavailable in the fast version, in which case the reader will by default fall back automatically to the pure-`Python` version.

The astropy.io.ascii subpackage now provides the capability to read a table within an HTML file or web URL into an `astropy` Table object. A Table object can now also be written out as an HTML table.

3.6.2. *FITS*

The astropy.io.fits subpackage started as a direct port of the PyFITS project (Barrett & Bridgman 1999). Therefore, it is pretty stable, with mostly bug fixes but also a few new features and performance improvements. The API remains mostly compatible with PyFITS, which is now deprecated in favor of `astropy`.

Command-line scripts are now available for printing a summary of the HDUs in FITS file(s) (fitsinfo) and for printing the header information to the screen in a human-readable format (fitsheader).

FITS files are now loaded *lazily* by default, i.e., an object representing the list of HDUs is created but the data are not loaded into memory until requested. This approach should provide substantial speed-ups when using the convenience functions (e.g., getheader() or getdata()) to get an HDU that is near the beginning in a file with many HDUs.

3.6.3. *HDF5*

The `astropy.io.misc.hdf5` subpackage provides support for binary read and write access to files in the *Hierarchical Data Format* (HDF5), if the `h5py` package is installed. Astropy table I/O is offered transparently through `Table.read()` and `Table.write()`, analogously to the other auto-detected formats. The keyword `path='group/subgroup/data set'` specifies the path to the data inside the file's hierarchical structure.

3.7. *Modeling*

The astropy.modeling subpackage provides a framework for representing analytical models and performing model evaluation and parameter fitting. The main motivation for this functionality was to create a framework that allows arbitrary combination of models to support the Generalized World Coordinate System (GWCS) package.[116] The current FITS

[116] https://github.com/spacetelescope/gwcs

WCS specification lacks the flexibility to represent arbitrary distortions and does not meet the needs of many types of current instrumentation. The fact that the `astropy` modeling framework now supports propagating units also makes it a useful tool for representing and fitting astrophysical models within data analysis tools.

Models and fitters are independent of each other: a model can be fit with different fitters and new fitters can be added without changing existing models. The framework is designed to be flexible and easily extensible. The goal is to have a rich set of models, but also facilitate creating new ones, if necessary.

3.7.1. Single Model Definition and Evaluation

Models are defined by their parameters and initialized by providing values for them. The names of the parameters are stored in a list, `Model.param_names`. Parameters are complex objects. They store additional information—default value, default unit, and parameter constraints. Parameter values and constraints can be updated by assignment. Supported constraints include `fixed` and `tied` parameters, as well as `bounds` on parameter values. The framework also supports models for which the number of parameters and their names are defined by another argument. A typical example is a polynomial model defined by its degree. A model is evaluated by calling it as a function.

If an analytical inverse of a model exists, it can be accessed by calling `Model.inverse`. In addition, `Model.inverse` can be assigned another model that represents a computed inverse.

Another useful settable property of models is `Model.bounding_box`. This attribute sets the domain over which the model is defined. This greatly improves the efficiency of evaluation when the input range is much larger than the characteristic width of the model itself.

3.7.2. Model Sets

Using astropy.modeling provides an efficient way to set up the same type of model with many different sets of parameter values. This creates a model set that can be efficiently evaluated. For example, in PSF (point-spread function) photometry, all objects in an image will have a PSF of the same functional form, but with different positions and amplitudes.

3.7.3. Compound Models

Models can be combined using arithmetic expressions. The result is also a model, which can further be combined with other models. Modeling supports arithmetic ($+$, $-$, *, $/$, and **), join (&), and composition ($|$) operators. The rules for combining models involve matching their inputs and outputs. For example, the composition operator, $|$, requires the number of outputs of the left model to be equal to the number of inputs of the right one. For the join operator, the total number of inputs must equal the sum of number of inputs of both the left and the right models. For all arithmetic operators, the left and the right models must have the same number of inputs and outputs. An example of a compound model could be a spectrum with interstellar absorption. The stellar spectrum and the interstellar extinction are represented by separate models, but the observed spectrum is fitted with a compound model that combines both.

3.7.4. Fitting Models to Data

The astropy.modeling subpackage also provides several fitters that are wrappers around some of the `numpy` and `scipy.optimize` functions and provide support for specifying parameter constraints. The fitters take a model and data as input and return a copy of the model with the optimized parameter values set. The goal is to make it easy to extend the fitting framework to create new fitters. The optimizers available in `astropy` version 2.0 are Levenberg–Marquardt (`scipy.optimize.leastsq`), Simplex (`scipy.optimize.fmin`), SLSQP (`scipy.optimize.slsqp`), and LinearLSQFitter (`numpy.linalg.lstsq` which provides exact solutions for linear models).

Modeling also supports a plugin system for fitters, which allows using the `astropy` models with external fitters. An example of this is `SABA`,[117] which is a bridge between Sherpa (Doe et al. 2007), and astropy.modeling, to bring the Sherpa fitters into `astropy`.

3.7.5. Creating New Models

If arithmetic combinations of existing models are not sufficient, new model classes can be defined in different ways. The astropy.modeling package provides tools to turn a simple function into a full-featured model, but it also allows extending the built-in model class with arbitrary code.

3.7.6. Unit Support

The astropy.modeling subpackage now supports the representation, evaluation, and fitting of models using Quantity objects, which attach units to scalar values or arrays of values. In practice, this means that one can, for example, fit a model to data with units and get parameters that also have units out, or initialize a model with parameters with units and evaluate it using input values with different but equivalent units. For example, the blackbody model (BlackBody1D) can be used to fit observed flux densities in a variety of units and as a function of different units of spectral coordinates (e.g., wavelength or frequency).

3.8. Convolution

The astropy.convolution subpackage implements *normalized convolution* (e.g., Knutsson & Westin 1993), an image reconstruction technique in which missing data are ignored during the convolution and replaced with values interpolated using the kernel. An example is given in Figure 4. In `astropy` versions $\leqslant$1.3, the direct convolution and Fast Fourier Transform (FFT) convolution approaches were inconsistent, with only the latter implementing normalized convolution. As of version 2.0, the two methods now agree and include a suite of consistency checks.

3.9. Visualization

The astropy.visualization subpackage provides functionality that can be helpful when visualizing data. This includes a framework (previously the standalone astropy.visualization.wcsaxes package) for plotting astronomical images with coordinates via `matplotlib`, functionality related to image normalization (including both scaling and stretching), smart

[117] https://github.com/astropy/saba

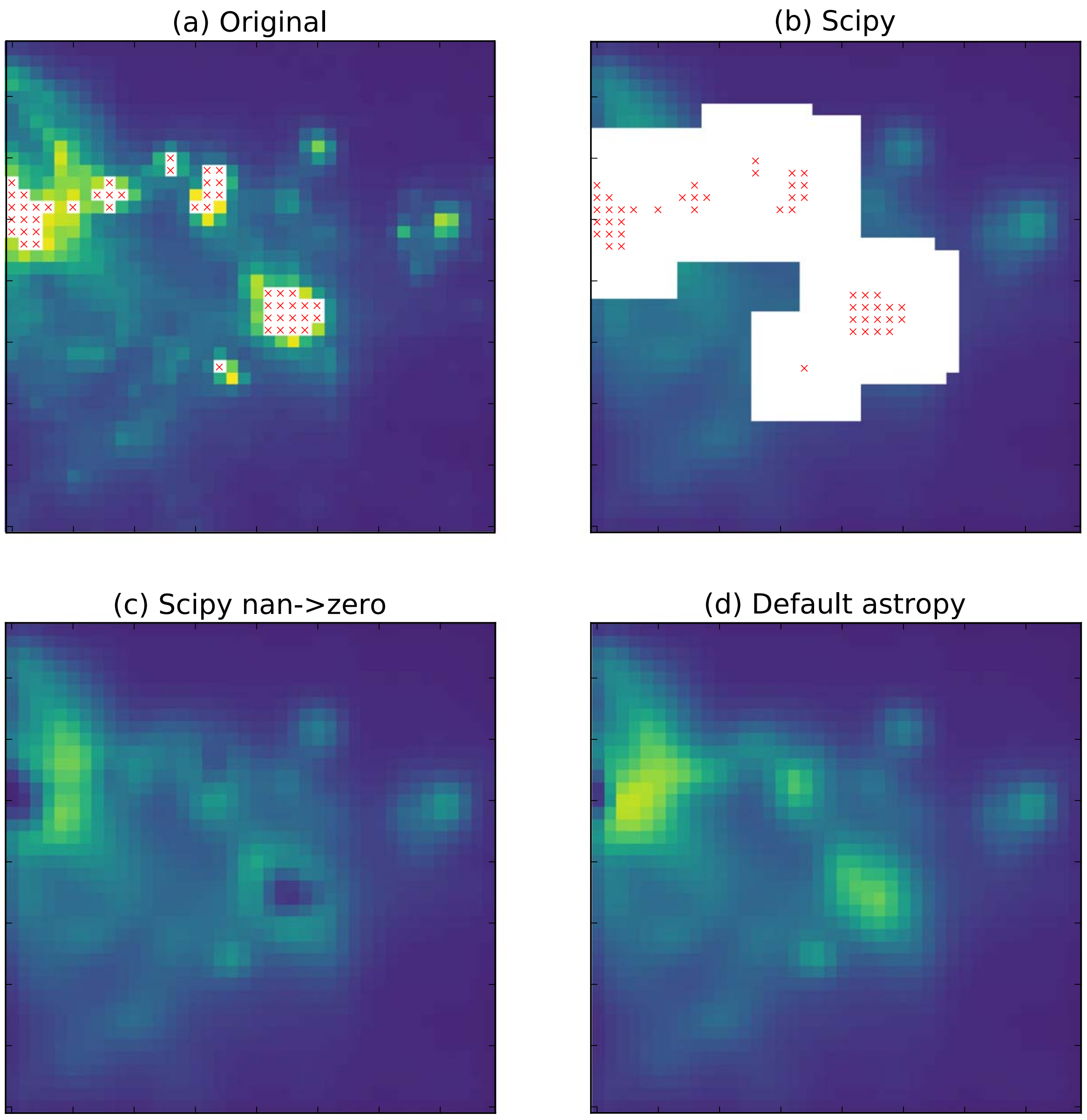

Figure 4. An example showing different modes of convolution available in the `Python` ecosystem. Each red *x* signifies a pixel that is set to `NaN` in the original data (top left). If the data are convolved with a Gaussian kernel on a 9 × 9 grid using `scipy`'s direct convolution (top right), any pixel within range of the original `NaN` pixels is also set to `NaN`. The bottom left panel shows what happens if the `NaN`s are set to zero first: the original `NaN` regions are depressed relative to their surroundings. Finally, the bottom right panel shows `astropy`'s convolution behavior, where the missing pixels are replaced with values interpolated from their surroundings using the convolution kernel.

histogram plotting, red-green-blue (RGB) color image creation from separate images, and custom plotting styles for `matplotlib`.

3.9.1. Image Stretching and Normalization

Using astropy.visualization provides a framework for transforming values in images (and more generally, any arrays), typically for the purpose of visualization. The two main types of transformations are normalization and stretching of image values.

Normalization transforms the image values to the range [0, 1] using lower and upper limits ($v_{\min}$, $v_{\max}$),

$$y = \frac{x - v_{\min}}{v_{\max} - v_{\min}}, \qquad (1)$$

where x represents the values in the original image.

Stretching transforms the image values in the range [0, 1] again to the range [0, 1], using a linear or nonlinear function:

$$z = f(y). \qquad (2)$$

Several classes are provided for automatically determining intervals (e.g., using image percentiles) and for normalizing values in this interval to the [0, 1] range.

`Using matplotlib` allows a custom normalization and stretch to be used when displaying data by passing in a normalization object. The astropy.visualization package also provides a normalization class that wraps the interval and stretches objects into a normalization object that `matplotlib` understands.

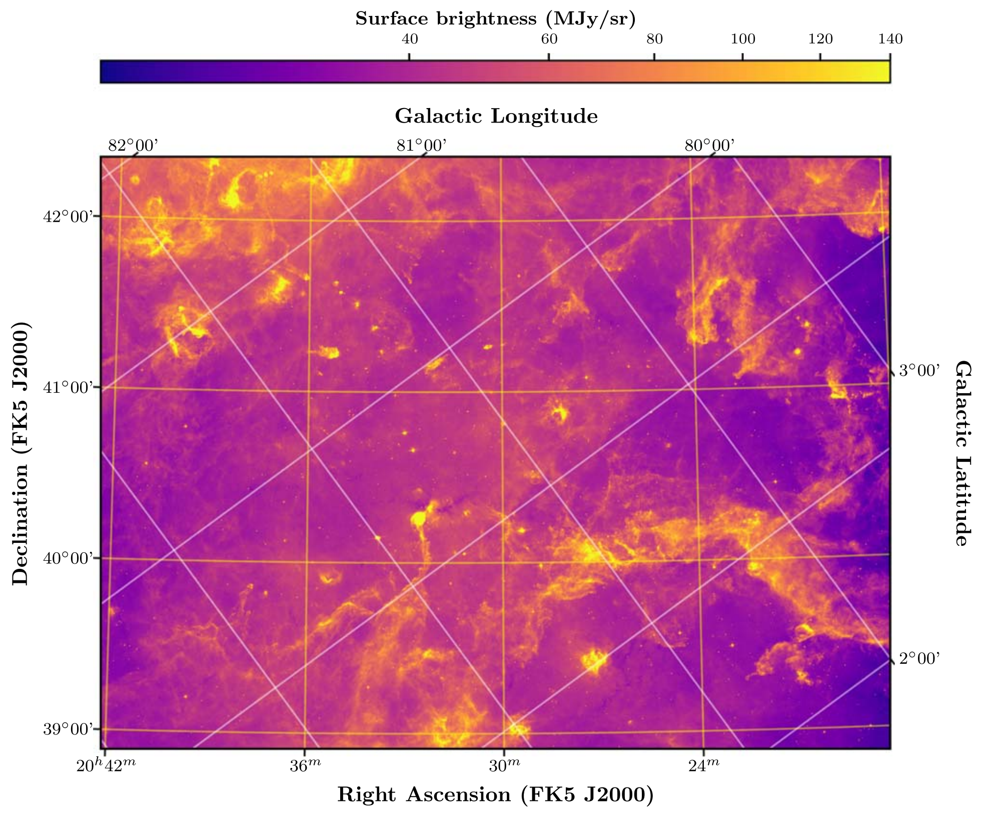

Figure 5. An example of a figure made using the astropy.visualization.wcsaxes subpackage, using *Spitzer*/IRAC 8.0 μm data from the Cygnus-X *Spitzer* Legacy survey (Beerer et al. 2010).

3.9.2. Plotting Image Data with World Coordinates

Astronomers dealing with observational imaging commonly need to make figures with images that include the correct coordinates and optionally display a coordinate grid. The challenge, however, is that the conceptual coordinate axes (such as longitude/latitude) need not be lined up with the pixel axes of the image. The astropy.visualization.wcsaxes subpackage implements a generalized way of making figures from an image array and a WCS object that provides the transformation between pixel and world coordinates.

World coordinates can be, for example, right ascension and declination, but can also include, for example, velocity, wavelength, frequency, or time. The main features from this subpackage include the ability to control which axes to show which coordinate on (e.g., showing longitude ticks on the top and bottom axes and latitude on the left and right axes), controlling the spacing of the ticks either by specifying the positions to use or providing a tick spacing or an average number of ticks that should be present on each axis, setting the format for the tick labels to ones commonly used by astronomers, controlling the visibility of the grid/graticule, and overlaying ticks, labels, and/or grid lines from different coordinate systems. In addition, it is possible to pass data with more than two dimensions and slice on-the-fly. Last but not least, it is also able to define non-rectangular frames such as, for example, Aitoff projections.

This subpackage differs from `APLpy` (Robitaille & Bressert 2012), in that the latter focuses on providing a very high-level interface to plotting that requires very few lines of code to get a good result, whereas astropy.visualization.wcsaxes defines an interface that is much closer to that of `matplotlib` (Hunter 2007). This enables significantly more advanced visualizations.

An example of a visualization made with astropy.visualization.wcsaxes is shown in Figure 5. This example illustrates the ability to overlay multiple coordinate systems and customize which ticks/labels are shown on which axes around the image. This also uses the image stretching functionality from Section 3.9.1 to show the image in a square-root stretch (automatically updating the tick positions in the colorbar).

3.9.3. Choosing Histogram Bins

The astropy.visualization subpackage also provides a histogram function, which is a generalization of `matplotlib`'s histogram function, to allow for a more flexible specification of histogram bins. The function provides several methods of automatically tuning the histogram bin size. It has a syntax identical to `matplotlib's` histogram function, with the exception of the `bins` parameter, which allows specification of one of four different methods for automatic bin selection: "blocks," "knuth," "scott," or "freedman."

3.9.4. Creating Color RGB Images

Lupton et al. (2004) describe an "optimal" algorithm for producing RGB composite images from three separate high-dynamic range arrays. The astropy.visualization subpackage provides a convenience function to create such a color image. It also includes an associated set of classes to provide alternate scalings. This functionality was contributed by developers from the Large Synoptic Survey Telescope (LSST) and serves as an

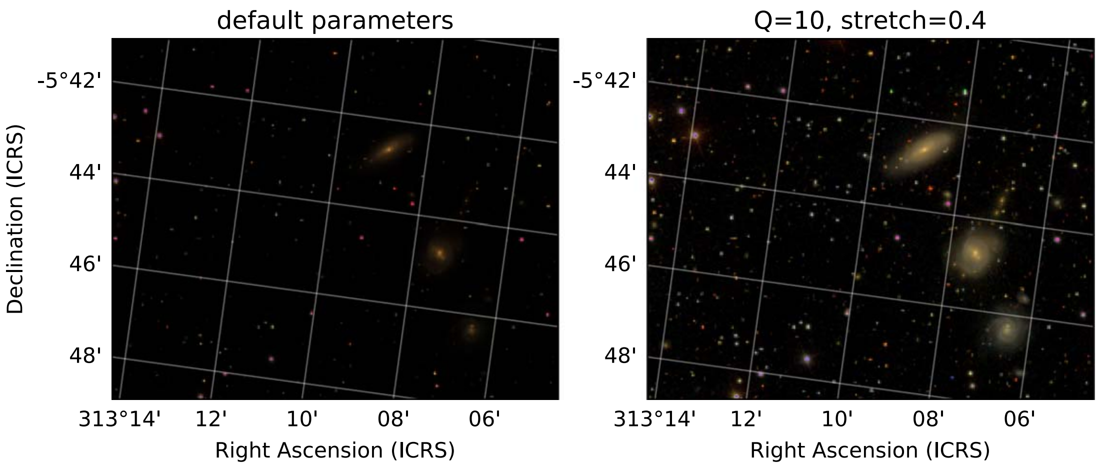

Figure 6. An RGB color image of the region near the Hickson 88 group constructed from SDSS images and the astropy.visualization tools. This example uses astropy.visualization.wcsaxes to display the sky coordinate grid, and the http://docs.astropy.org/en/stable/api/astropy.visualization.make_lupton_rgb.html `make_lupton_rgb()` function to produce the RGB image from three SDSS filter images (g, r, i). The image on the left shows the image with the default parameters, whereas the image on the right has parameters set to show a greater dynamical range.

example of a contribution to Astropy from a more traditional engineering organization (Jenness et al. 2016).

The Sloan Digital Sky Survey (SDSS) SkyServer color images were made using a variation on this technique. As an example, in Figure 6, we show an RGB color image of the Hickson 88 group, centered near NGC 6977.[118] This image was generated from SDSS images using the astropy.visualization tools.

3.10. Cosmology

The astropy.cosmology subpackage contains classes for representing different cosmologies and functions for calculating commonly used quantities such as look-back time and distance. The subpackage was described in detail in Astropy Collaboration et al. (2013). The default cosmology in `astropy` version 2.0 is given by the values in Planck Collaboration et al. (2016).

3.11. Statistics

The astropy.stats package provides statistical tools that are useful for astronomy and are either not found in or extend the available functionality of other `Python` statistics packages, such as `scipy` (Jones et al. 2001) or `statsmodels` (Seabold & Perktold 2010). The astropy.stats package also contains a range of functionality used by many different disciplines in astronomy. It is not a complete set of statistical tools, but rather a still growing collection of useful features.

3.11.1. Robust Statistical Estimators

Robust statistics provide reliable estimates of basic statistics for complex distributions that largely mitigate the effects of outliers. The astropy.stats package includes several robust statistical functions that are commonly used in astronomy, such as sigma clipping methods for rejecting outliers, median absolute deviation functions, and biweight estimators, which have been used to calculate the velocity dispersion of galaxy clusters (Beers et al. 1990).

[118] http://skyserver.sdss.org/dr13/en/tools/chart/navi.aspx?ra=313.12381&dec=-5.74611

3.11.2. Circular Statistics

Astronomers often need to compute statistics of quantities evaluated on a circle, such as sky direction or polarization angle. A set of circular statistical estimators based on Jammalamadaka & Sengupta (2001) are implemented in astropy.stats. These functions provide measurements of the circular mean, variance, and moment. All of these functions work with both `numpy.ndarrays` (assumed to be in radians) and Quantity objects. In addition, the subpackage includes tests for Rayleigh Test, `vtest`, and a function to compute the maximum likelihood estimator for the parameters of the von Mises distribution.

3.11.3. Lomb–Scargle Periodograms

Periodic analysis of unevenly spaced time series is common across many subfields of astronomy. The astropy.stats package now includes several efficient implementations of the Lomb–Scargle periodogram (Lomb 1976; Scargle 1982) and several generalizations, including floating mean models (Zechmeister & Kürster 2009), truncated Fourier models (Bretthorst 2003), and appropriate handling of heteroscedastic uncertainties. Importantly, the implementations make use of several fast and scalable computational approaches (e.g., Press & Rybicki 1989; Palmer 2009), and thus can be applied to much larger data sets than Lomb–Scargle algorithms available in, e.g., `scipy.stats` (Jones et al. 2001). Much of the Lomb–Scargle code in `astropy` has been adapted from previously published open-source code (VanderPlas et al. 2012; VanderPlas & Ivezic 2015). Users should be aware that correct interpretation of periodogram results involves some subtleties; for a thorough discussion of this issue, see (VanderPlas 2018, in press).

3.11.4. Bayesian Blocks and Histogram Binning

The astropy.stats package also includes an implementation of *Bayesian Blocks* (Scargle et al. 2013), an algorithm for analysis of breakpoints in nonperiodic astronomical time-series. One interesting application of Bayesian Blocks is its use in determining optimal histogram binnings, particularly binnings with unequal bin sizes. This code was adapted, with several improvements, from the `astroML` package (VanderPlas et al. 2012). An

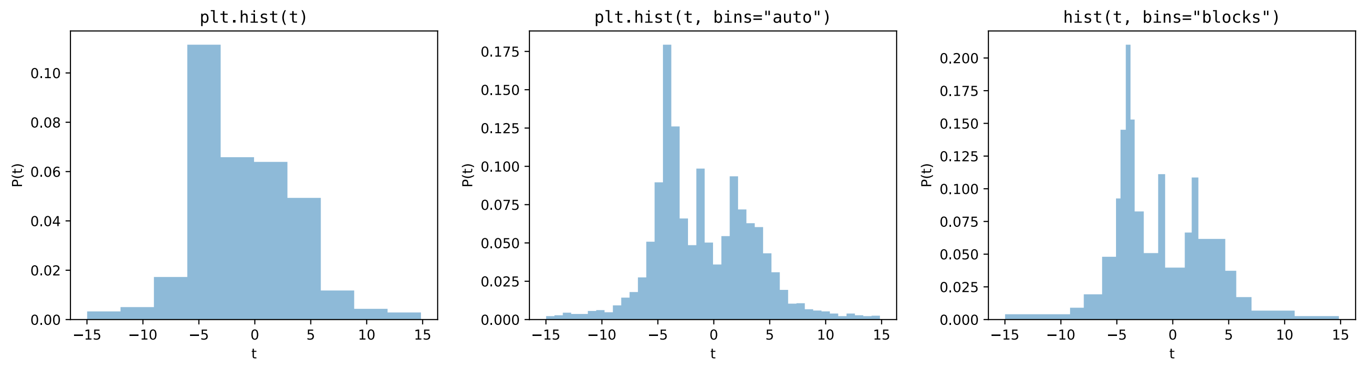

Figure 7. Three approaches to a 1D histogram. Left: a standard histogram using `matplotlib`'s default of 10 bins. Center: a histogram with the number of equal-width bins determined automatically using `numpy`'s `bins = 'auto'`. Right: a histogram created by `astropy`, with irregularly spaced bins computed via the Bayesian Blocks algorithm. Compared to regularly spaced bins, the irregular bin widths give a more accurate visual representation of features in the data set at various scales.

example of a histogram fit using the Bayesian Blocks algorithm is shown in the right panel of Figure 7.

4. Infrastructure for Astropy Affiliated Packages

In addition to astronomy-specific packages and libraries, the Astropy Project also maintains and distributes several general-purpose infrastructure packages that assist with the maintenance and upkeep of the `astropy` core package and other affiliated packages. The following sections describe the most widely used infrastructure packages developed by the Astropy Project.

4.1. Package Template

Astropy provides a package template—as a separate `GitHub` repository, `astropy/package-template`[119]—that aims to simplify setting up packaging, testing, and documentation builds for developers of affiliated packages or `astropy`-dependent packages. Any `Python` package can make use of this ready-to-go package layout, setup, installation, and `Sphinx` documentation build infrastructure that was originally developed for the `astropy` core package and affiliated packages maintained by the Astropy Project. The package template also provides a testing framework, template configurations for continuous integration services, and `Cython` build support.

4.2. Continuous Integration Helpers

Astropy also provides a set of scripts for setting up and configuring continuous integration (CI) services as a `GitHub` repository, `astropy/ci-helpers`.[120] These tools aim to enable package maintainers to control their testing setup and installation process for various CI services through a set of environment variables. While the current development is mostly driven by the needs of the Astropy ecosystem, the actual usage of this package is extremely widespread.[121] The current tools support configuration for Travis CI[122] and Appveyor CI.[123]

[119] https://github.com/astropy/package-template/
[120] https://github.com/astropy/ci-helpers
[121] Approximately 30% of the ci-helpers contributors come from outside the astropy community, and there are currently 292 repositories on github that have ci-helpers in their travis configuration file.
[122] https://travis-ci.org/
[123] https://www.appveyor.com/

4.3. Sphinx Extensions

The documentation for many `Python` packages, including all the packages in the Astropy ecosystem, is written using the `Sphinx` documentation build system. `Sphinx` supports writing documentation using plain text files that follow a markup language called `reStructuredText` (RST). These files are then transformed into HTML, PDF, or LATEX documents during the documentation build process. For the Astropy Project, we have developed several `Sphinx` extensions that facilitate automatically generating API documentation for large projects, like the `astropy` core package. The main extension we have developed is `sphinx-automodapi`,[124] which provides a convenient single RST command to generate a set of documentation pages, listing all of the available classes, functions, and attributes in a given `Python` module.

5. The Future of the Astropy Project

Following the release of version 2.0, development on the next major version of the `astropy` core package (version 3.0) began. On top of planned changes and additions to the core package, we also plan to overhaul the Astropy educational/learning materials and further generalize the infrastructure utilities originally developed for the core package for the benefit of the community.

5.1. Future Versions of the Astropy Core and Affiliated Packages

One of the most significant changes coming in this next major release will be removing the support for `Python` 2 (Robitaille 2017): future versions of `astropy` will only support `Python` 3.5 or higher. Removing `Python` 2 support will allow the use of new features exclusive to `Python` 3, simplify the code base, and reduce the testing overhead for the package. Version 3.0 was released in February 2018.

In the next major release after version 3.0, scheduled for mid-2018, the focus will be on algorithm optimization and documentation improvement. To prepare for this release, we are subjecting the core package to testing, evaluation, and performance monitoring. As a result, less new functionality may be introduced, as a trade-off for better performance.

Beyond the core code, the Astropy Project is also further developing the Astropy-managed affiliated packages. While these may not be integrated into the `astropy` core package,

[124] http://sphinx-automodapi.readthedocs.io

these projects provide code that is useful to the astronomical community and meet the testing and documentation standards of Astropy. Some of these new efforts include an initiative to develop tools for spectroscopy (Crawford et al. 2017, `specutils`, `specreduc`, `specviz`), integration of LSST software, and support for HEALPIX projection.

5.2. Learn Astropy

The documentation of the `astropy` core package contains narrative descriptions of the package's functionality, along with detailed usage notes for functions, classes, and modules. While useful as a reference for more experienced `Python` users, it is not the proper point of entry for other users or learning environments. In the near future, we will launch a new resource for learning to use both the `astropy` core package and the many packages in the broader Astropy ecosystem, under the name *Learn Astropy*.

The new *Learn Astropy* site will present several different ways to engage with the Astropy ecosystem:

Documentation: The `astropy` and affiliated package documentation contains the complete description of a package with all requisite details, including usage, dependencies, and examples. The pages will largely remain as-is, but will be focused toward more intermediate users and as a reference resource.

Examples: These are stand-alone code snippets that live in the `astropy` documentation that demonstrate a specific functionality within a subpackage. The `astropy` core package documentation will then gain a new "index of examples" that links to all of the code or demonstrative examples within any documentation page.

Tutorials: The Astropy tutorials are step-by-step demonstrations of common tasks that incorporate several packages or subpackages. Tutorials are more extended and comprehensive than examples, may contain exercises for the users, and are generally geared towards workshops or teaching. Several tutorials already exist[125] and are being actively expanded.

Guides: These are long-form narrative, comprehensive, and conceptually focused documents (roughly one book chapter in length), providing stand-alone introductions to core packages in addition to the underlying astronomical concepts. These are less specific and more conceptual than tutorials: for example, "using `astropy` and `ccdproc` to reduce imaging data."

We encourage any users who wish to see specific material to either contribute or comment on these efforts via the Astropy mailing list or `astropy/astropy-tutorials GitHub` repository.[126]

6. Conclusion

The `astropy` package is improving in stability, breadth, and reliability while the the Astropy project is still significantly growing. As the `astropy` core package becomes more mature, several subpackages have reached stability with a rich set of features that help astronomers worldwide to perform many daily tasks, such as planning observations, analyzing data or simulation results, and writing publications. The strong emphasis that the Astropy Project puts on reliability and high coding standards helps users to trust the calculations performed with `astropy` and to publish reproducible results. At the same time, the Astropy ecosystem and core package are both growing: new functionality is still being contributed and new affiliated packages are being developed to support more specialized needs.

The Astropy Project is also spreading awareness of best practices in community-driven software development. This is important because most practicing astronomers were not explicitly taught computer science and software development, despite the fact that a substantial fraction of many astronomers' workload today is related to software use and development. The `astropy` package leads by example, showing all interested astronomers how modern tools like `git` version control or CI testing can increase the quality, accessibility, and discoverability of astronomical software without overly complicating the development cycle. Within Astropy, all submitted code is reviewed by at least one (but typically more than one) member of the Astropy community. Reviewers provide feedback to contributors, which helps to improve contributors' software development skills. As a community, Astropy follows an explicit code of conduct (Cruz et al. 2015) and treats all contributors and users with respect, provides a harassment-free environment, and encourages and welcomes new contributions from all. Thus, while the Astropy Project provides and develops software and tools essential to modern astronomical research, it also helps to prepare the current and next generation of researchers with the knowledge to adequately use, develop, and contribute to those tools within a conscientious and welcoming community.

We thank the referee for an insightful report that improved the paper. We would like to thank the members of the community who have contributed to Astropy, who have opened issues and provided feedback, and who have supported the project in a number of different ways. We would like to acknowledge Alex Conley and Neil Crighton for maintaining the astropy.cosmology subpackage.

The Astropy community is supported by and makes use of a number of organizations and services outside the traditional academic community. We thank Google for financing and organizing the Google Summer of Code (GSoC) program, that has funded several students per year to work on Astropy related projects over the summer. These students often turn into long-term contributors. We also thank NumFOCUS and the Python Software Foundation for financial support. Within the academic community, we thank institutions that make it possible for astronomers and other developers on their staff to contribute their time to the development of Astropy projects. We acknowledge the support of the Space Telescope Science Institute, Harvard–Smithsonian Center for Astrophysics, and the South African Astronomical Observatory.

The following individuals would like to recognize support for their personal contributions. H.M.G. was supported by the National Aeronautics and Space Administration through the Smithsonian Astrophysical Observatory contract SV3-73016 to MIT for Support of the *Chandra* X-Ray Center, which is operated by the Smithsonian Astrophysical Observatory for and on behalf of the National Aeronautics Space Administration under contract NAS8-03060. J.T.V. was supported by the UW eScience Institute via grants from the Moore Foundation, the Sloan Foundation, and the Washington Research Foundation. S.M.C. acknowledges the National Research Foundation of South Africa. M.V.B. was supported by NASA's Planetary

[125] http://tutorials.astropy.org/
[126] https://github.com/astropy/astropy-tutorials

Astronomy Program. T.L.A. was supported by NASA contract NAS8-03060. Support for E.J.T. was provided by NASA through Hubble Fellowship grant No. 51316.01 awarded by the Space Telescope Science Institute, which is operated by the Association of Universities for Research in Astronomy, Inc., for NASA, under contract NAS 5-26555, as well as a Giacconi Fellowship. M.B. was supported by the FONDECYT regular project 1170618 and the MINEDUC-UA projects codes ANT 1655 and ANT 1656. D.H. was supported through the SFB 881 "The Milky Way System" by the German Research Foundation (DFG). W.E.K was supported by an ESO Fellowship. C.M. is supported by NSF grant AST-1313484. S.P. was supported by grant AYA2016-75808-R (FEDER) issued by the Spanish government. J.E.H.T. was supported by the Gemini Observatory, which is operated by the Association of Universities for Research in Astronomy, Inc., on behalf of the international Gemini partnership of Argentina, Brazil, Canada, Chile, and the United States of America. Y.P.B was supported by the Korea Astronomy and Space Science Institute, under the R&D program supervised by the Ministry of Science, ICT, and Future Planning.

Furthermore, the astropy packages would not exist in their current form without a number of web services for code hosting, continuous integration, and documentation; in particular, astropy heavily relies on GitHub, Travis CI, Appveyor, CircleCI, and Read the Docs.

`Astropy` interfaces with the SIMBAD database, operated at CDS, Strasbourg, France. It also makes use of the ERFA library (Tollerud et al. 2017), which in turn derives from the IAU SOFA Collection[127] developed by the International Astronomical Union Standards of Fundamental Astronomy (Hohenkerk 2011).

Software: `astropy` (Astropy Collaboration et al. 2013), `numpy` (Van der Walt et al. 2011), `scipy` (Jones et al. 2001), `matplotlib` (Hunter 2007), `Cython` (Behnel et al. 2011), `SOFA` (Hohenkerk 2011), `ERFA` (Tollerud et al. 2017).

Appendix
List of Affiliated Packages

The Appendix comprises Table 1.

Table 1
Registry of Affiliated Packages

Package Name	Stable	PyPI Name	Maintainer	Citation
APLpy	Yes	APLpy	Thomas Robitaille and Eli Bressert	Robitaille & Bressert (2012)
Astro-SCRAPPY	Yes	astroscrappy	Curtis McCully	van Dokkum (2001)
astroML	Yes	astroML	Jake Vanderplas	Vanderplas et al. (2012), Ivezić et al. (2014)
astroplan	No	astroplan	Brett Morris	Morris et al. (2018)
astroquery	Yes	astroquery	Adam Ginsburg and Brigitta Sipocz	Ginsburg et al. (2017b)
ccdproc	Yes	ccdproc	Steven Crawford, Matt Craig, and Michael Seifert	Craig et al. (2015)
cluster-lensing	No	cluster-lensing	Jes Ford	Ford (2016)
gala	Yes	astro-gala	Adrian Price-Whelan	Price-Whelan (2017)
galpy	Yes	galpy	Jo Bovy	Bovy (2015)
gammapy	No	gammapy	Christoph Deil	Deil et al. (2017)
ginga	Yes	ginga	Eric Jeschke and Pey-Lian Lim	ejeschke et al. (2017)
Glue	Yes	glueviz	Chris Beaumont and Thomas Robitaille	Beaumont et al. (2014)
gwcs	No	gwcs	Nadia Dencheva	Dencheva et al. (2017)
Halotools	Yes	halotools	Andrew Hearin	Hearin et al. (2017)
HENDRICS	Yes	hendrics	Matteo Bachetti	Bachetti (2015)
hips	No	hips	Christoph Deil and Thomas Boch	hips developers (2018)
imexam	No	imexam	Megan Sosey	Sosey (2017)
linetools	Yes	linetools	J. Xavier Prochaska, Nicolas Tejos, and Neil Crighton	Prochaska et al. (2017)
marxs	Yes	marxs	Hans Moritz Günther	Günther et al. (2017)
naima	Yes	naima	Victor Zabalza	Zabalza (2015)
omnifit	Yes	omnifit	Aleksi Suutarinen	Suutarinen (2015)
photutils	No	photutils	Larry Bradley and Brigitta Sipocz	Bradley et al. (2017)
poliastro	No	poliastro	Juan Luis Cano Rodríguez	Rodríguez et al. (2017)
PyDL	No	pydl	Benjamin Alan Weaver	Weaver et al. (2017)
pyregion	Yes	pyregion	Jae-Joon Lee and Christoph Deil	pyregions developers (2018)
pyspeckit	Yes	pyspeckit	Adam Ginsburg	Ginsburg & Mirocha (2011)
python-cpl	No	python-cpl	Ole Streicher	Streicher & Weilbacher (2012)
PyVO	No	pyvo	Stefan Becker	Graham et al. (2014)
regions	No	regions	Christoph Deil and Johannes King	pyregions developers (2018)
reproject	Yes	reproject	Thomas Robitaille	Robitaille (2018)
sncosmo	Yes	sncosmo	Kyle Barbary	Barbary (2014)
spectral-cube	Yes	spectral-cube	Adam Ginsburg	Ginsburg et al. (2017a)
specutils	No	specutils	Nicholas Earl, Adam Ginsburg, Steve Crawford, and Erik Tollerud	specutils developers (2018)
spherical_geometry	No	spherical-geometry	Bernie Simon	
stingray	No	stingray	Daniela Huppenkothen, Matteo Bachetti, Abigail Stevens, Simone Migliari, and Paul Balm	Huppenkothen et al. (2016)
synphot	Yes	synphot	Pey Lian Lim	Lim (2016)

[127] http://www.iausofa.org

ORCID iDs

A. M. Price-Whelan https://orcid.org/0000-0003-0872-7098
S. M. Crawford https://orcid.org/0000-0002-8969-5229
A. Ginsburg https://orcid.org/0000-0001-6431-9633
J. T. VanderPlas https://orcid.org/0000-0002-9623-3401
L. D. Bradley https://orcid.org/0000-0002-7908-9284
M. de Val-Borro https://orcid.org/0000-0002-0455-9384
K. L. Cruz https://orcid.org/0000-0002-1821-0650
T. P. Robitaille https://orcid.org/0000-0002-8642-1329
E. J. Tollerud https://orcid.org/0000-0002-9599-310X
M. Bachetti https://orcid.org/0000-0002-4576-9337
P. Barmby https://orcid.org/0000-0003-2767-0090
G. B. Brammer https://orcid.org/0000-0003-2680-005X
L. H. Garrison https://orcid.org/0000-0002-9853-5673
D. Homeier https://orcid.org/0000-0002-8546-9128
G. Hosseinzadeh https://orcid.org/0000-0002-0832-2974
T. Jenness https://orcid.org/0000-0001-5982-167X
S. Kendrew https://orcid.org/0000-0002-7612-0469
N. S. Kern https://orcid.org/0000-0002-8211-1892
C. McCully https://orcid.org/0000-0001-5807-7893
B. M. Morris https://orcid.org/0000-0003-2528-3409
S. Pascual https://orcid.org/0000-0002-9351-6051
A. L. Plunkett https://orcid.org/0000-0002-9912-5705
J. X. Prochaska https://orcid.org/0000-0002-7738-6875
L. P. Singer https://orcid.org/0000-0001-9898-5597
P. Teuben https://orcid.org/0000-0003-1774-3436
G. R. Tremblay https://orcid.org/0000-0002-5445-5401
A. de la Vega https://orcid.org/0000-0002-6219-5558
L. L. Watkins https://orcid.org/0000-0002-1343-134X
J. Woillez https://orcid.org/0000-0002-2958-4738

References

Aldcroft, T. 2015, Astropy Proposal for Enhancement 6: Enhanced Character Separated Values table format (APE 6), Zenodo, doi:10.5281/zenodo.1043901
Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33
Bachetti, M. 2015, MaLTPyNT: Quick look timing analysis for NuSTAR data, Astrophysics Source Code Library, ascl:1502.021
Banse, K., Ponz, D., Ounnas, C., Grosbol, P., & Warmels, R. 1988, in Instrumentation for Ground-Based Optical Astronomy, ed. L. B. Robinson (New York: Springer-Verlag), 431
Barbary, K. 2014, sncosmo v0.4.2, Zenodo, doi:10.5281/zenodo.11938
Barrett, P. E., & Bridgman, W. T. 1999, in ASP Conf. Ser. 172, Astronomical Data Analysis Software and Systems VIII, ed. D. M. Mehringer, R. L. Plante, & D. A. Roberts (San Francisco, CA: ASP), 483
Barron, E. G., Kaplan, G. H., Bangert, J., et al. 2011, BAAS, 43, 344.14
Beaumont, C., Robitaille, T., Borkin, M., & Goodman, A. 2014, glueviz v0.4: multidimensional data exploration, Zenodo, doi:10.5281/zenodo.13866
Beerer, I. M., Koenig, X. P., Hora, J. L., et al. 2010, ApJ, 720, 679
Beers, T. C., Flynn, K., & Gebhardt, K. 1990, AJ, 100, 32
Behnel, S., Bradshaw, R., Citro, C., et al. 2011, CSE, 13, 31
Berry, D. S., Warren-Smith, R. F., & Jenness, T. 2016, A&C, 15, 33
Bovy, J. 2015, ApJS, 216, 29
Bradley, L., Sipocz, B., Robitaille, T., et al. 2017, astropy/photutils: v0.4, Zenodo, doi:10.5281/zenodo.1039309
Bretthorst, G. L. 2003, in Statistical Challenges in Astronomy. Third Statistical Challenges in Modern Astronomy (SCMA III) Conf., ed. E. D. Feigelson & G. J. Babu (New York: Springer), 309
Craig, M. W., Crawford, S. M., Deil, C., et al. 2015, ccdproc: CCD data reduction software, Astrophysics Source Code Library, ascl:1510.007
Crawford, S., Earl, N., Ginsburg, A., & Tollerud, E. 2017, Vision for Astropy Spectroscopic Tools (APE 13), Zenodo, doi:10.5281/zenodo.1117943
Cruz, K., Hagen, A., Robitaille, T., Tollerud, E., & Villaume, A. 2015, Astropy Proposal for Enhancement 8: Astropy Community Code of Conduct (APE 8), Zenodo, doi:10.5281/zenodo.1043913
Currie, M. J., Berry, D. S., Jenness, T., et al. 2014, in ASP Conf. Ser. 485, Astronomical Data Analysis Software and Systems XXIII, ed. N. Manset & P. Forshay (San Francisco, CA: ASP), 391
Deil, C., Zanin, R., Lefaucheur, J., et al. 2017, arXiv:1709.01751
Dencheva, N., Sipocz, B., Jones, C., et al. 2017, spacetelescope/gwcs: GWCS v0.8.0, Zenodo, doi:10.5281/zenodo.1041790
Disney, M. J., & Wallace, P. T. 1982, QJRAS, 23, 485
Doe, S., Nguyen, D., Stawarz, C., et al. 2007, in ASP Conf. Ser. 376, Astronomical Data Analysis Software and Systems XVI, ed. R. A. Shaw, F. Hill, & D. J. Bell (San Francisco, CA: ASP), 543
ejeschke, Lim, P. L., Rajul, et al. 2017, ejeschke/ginga: Ginga v2.6.6, Zenodo, doi:10.5281/zenodo.1040969
Ford, J. 2016, cluster-lensing: v0.1.2, Zenodo, doi:10.5281/zenodo.51370
Ginsburg, A., & Mirocha, J. 2011, PySpecKit: Python Spectroscopic Toolkit, Astrophysics Source Code Library, ascl:1109.001
Ginsburg, A., Robitaille, T., Koch, E., et al. 2017a, radio-astro-tools/spectral-cube: Version 0.4.1 release, Zenodo, doi:10.5281/zenodo.1013994
Ginsburg, A., Sipocz, B., Parikh, M., et al. 2017b, astropy/astroquery: v0.3.6 with fixed license, Zenodo, doi:10.5281/zenodo.826911
Graham, M., Plante, R., Tody, D., & Fitzpatrick, M. 2014, PyVO: Python access to the Virtual Observatory, Astrophysics Source Code Library, ascl:1402.004
Greenfield, P. 2013, Astropy Proposal for Enhancement 1: APE Purpose and Process (APE 1), Zenodo, doi:10.5281/zenodo.1043886
Günther, H. M., Frost, J., & Theriault-Shay, A. 2017, AJ, 154, 243
Hearin, A. P., Campbell, D., Tollerud, E., et al. 2017, AJ, 154, 190
hips developers 2018, hips—Python library to handle HiPS data, https://github.com/hipspy/hips
Hohenkerk, C. 2011, SchpJ, 6, 11404
Hunter, J. D. 2007, CS&E, 9, 90
Huppenkothen, D., Bachetti, M., Stevens, A. L., Migliari, S., & Balm, P. 2016, Stingray: Spectral-timing software, Astrophysics Source Code Library, ascl:1608.001
Ivezić, Ž., Connolly, A., Vanderplas, J., & Gray, A. 2014, Statistics, Data Mining and Machine Learning in Astronomy (Princeton, NJ: Princeton Univ. Press)
Jammalamadaka, S. R., & Sengupta, A. 2001, Topics in Circular Statistics (Singapore: World Scientific)
Jenness, T., & Berry, D. S. 2013, in ASP Conf. Ser. 475, Astronomical Data Analysis Software and Systems XXII, ed. D. N. Friedel (San Francisco, CA: ASP), 307
Jenness, T., Bosch, J., Owen, R., et al. 2016, Proc. SPIE, 9913, 99130G
Jones, E., Oliphant, T., Peterson, P., et al. 2001, SciPy: Open source scientific tools for Python, http://www.scipy.org/
Knutsson, H., & Westin, C. F. 1993, in Proc. IEEE Conf. on Computer Visionand Pattern Recognition (New York: IEEE), 515
Lim, P. L., et al. 2016, synphot User's Guide (Baltimore, MD: STScI), https://synphot.readthedocs.io/en/latest/
Lomb, N. R. 1976, Ap&SS, 39, 447
Lupton, R., Blanton, M. R., Fekete, G., et al. 2004, PASP, 116, 133
Mamajek, E. E., Prsa, A., Torres, G., et al. 2015, arXiv:1510.07674
McKinney, W. 2010, in Proc. 9th Python in Science Conf., ed. S. van der Walt & J. Millman, 51
Mohr, P. J., Newell, D. B., & Taylor, B. N. 2016, RvMP, 88, 035009
Morris, B. M., Tollerud, E., Sipőcz, B., et al. 2018, AJ, 155, 128
Muna, D., Alexander, M., Allen, A., et al. 2016, arXiv:1610.03159
Palmer, D. M. 2009, ApJ, 695, 496
Pence, W. D., Chiappetti, L., Page, C. G., Shaw, R. A., & Stobie, E. 2010, A&A, 524, A42
Planck Collaboration, Ade, P. A. R., Aghanim, N., et al. 2016, A&A, 594, A13
Press, W. H., & Rybicki, G. B. 1989, ApJ, 338, 277
Price-Whelan, A., Crawford, S., Sipocz, B., et al. 2018, astropy/astropy-v2.0-paper: final draft, Zenodo, doi:10.5281/zenodo.1211397
Price-Whelan, A. M. 2017, JOSS, 2, doi:10.21105/joss.00388
Prochaska, J. X., Tejos, N., Crighton, N., et al. 2017, linetools/linetools: Third Minor Release, Zenodo, doi:10.5281/zenodo.1036773
pyregions developers 2018, regions—ds9 region parser for python, https://github.com/astropy/pyregion
Rhodes, B. C. 2011, PyEphem: Astronomical Ephemeris for Python, Astrophysics Source Code Library, ascl:1112.014
Robitaille, T. 2017, Astropy Proposal for Enhancement: Roadmap for Python 3-only support (APE 10), Zenodo, doi:10.5281/zenodo.1038587

Robitaille, T. 2018, reproject: astronomical image reprojection in Python, Zenodo, doi:10.5281/zenodo.1162674
Robitaille, T., & Bressert, E. 2012, APLpy: Astronomical Plotting Library in Python, Astrophysics Source Code Library, ascl:1208.017
Rodríguez, J. L. C., Hidalgo, A., Márquez, A. L., et al. 2017, poliastro/poliastro: poliastro 0.8.0 (OSCW edition), Zenodo, doi:10.5281/zenodo.1058988
Scargle, J. D. 1982, ApJ, 263, 835
Scargle, J. D., Norris, J. P., Jackson, B., & Chiang, J. 2013, ApJ, 764, 167
Seabold, S., & Perktold, J. 2010, in 9th Python in Science Conf., ed. S. van der Walt & J. Millman
Sosey, M. 2017, imexam version 0.8.0 release, Zenodo, doi:10.5281/zenodo.1042809
specutils developers 2018, specutils—Affiliated package for 1D spectral operations, https://github.com/astropy/specutils
Streicher, O., & Weilbacher, P. M. 2012, in ASP Conf. Ser. 461, Astronomical Data Analysis Software and Systems XXI, ed. P. Ballester, D. Egret, & N. P. F. Lorente (San Francisco, CA: ASP), 853
Suutarinen, A. 2015, omnifit: v0.2.1, Zenodo, doi:10.5281/zenodo.35536
Terlouw, J. P., & Vogelaar, M. G. R. 2015, Kapteyn Package, version 2.3 (Groningen: Kapteyn Astronomical Institute)
Tody, D. 1993, in ASP Conf. Ser. 52, Astronomical Data Analysis Software and Systems II, ed. R. J. Hanisch, R. J. V. Brissenden, & J. Barnes (San Francisco, CA: ASP), 173
Tollerud, E. 2013, Astropy Proposal for Enhancement 2: Astropy Release Cycle and Version Numbering (APE 2), Zenodo, doi:10.5281/zenodo.1043888
Tollerud, E., Pascual, S., Nair, P., et al. 2017, liberfa/erfa: v1.4.0, Zenodo, doi:10.5281/zenodo.1021149
Tollerud, E., Price-Whelan, A., Aldcroft, T., & Robitaille, T. 2014, Astropy Proposal for Enhancement 5: Coordinates Subpackage Plan (APE 5), Zenodo, doi:10.5281/zenodo.1043897
Turk, M. J., Smith, B. D., Oishi, J. S., et al. 2011, ApJS, 192, 9
Van der Walt, S., Colbert, S. C., & Varoquaux, G. 2011, CSE, 13, 22
van Dokkum, P. G. 2001, PASP, 113, 1420
VanderPlas, J. 2018, ApJS, arXiv:1703.09824
VanderPlas, J., Connolly, A., Ivezić, Ž., & Gray, A. 2012, in Conf. on Intelligent Data Understanding (CIDU), ed. K. Das et al. (New York: IEEE), 47
VanderPlas, J. T., & Ivezic, Z. 2015, arXiv:1502.01344
Wallace, P. T. 1994, in ASP Conf. Ser. 61, Astronomical Data Analysis Software and Systems III, ed. D. R. Crabtree, R. J. Hanisch, & J. Barnes (San Francisco, CA: ASP), 481
Weaver, B. A., Barbary, K., Bradley, K., et al. 2017, weaverba137/pydl: PyDL v0.6.0, Zenodo, doi:10.5281/zenodo.1095151
Zabalza, V. 2015, Proc. ICRC, 922
Zechmeister, M., & Kürster, M. 2009, A&A, 496, 577