March 12, 2022

Draft version March 12, 2022
Typeset using LaTeX/*show your work!* **modern** style in AASTeX631

The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package

Astropy Collaboration

Submitted to ApJ

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 summarize key features in the core package as of the recent major release, version 5.0, and provide major updates for the project. We then discuss supporting a broader ecosystem of inter-operable packages, including connections with several astronomical observatories and missions. We also revisit the future outlook of the Astropy Project and the current status of Learn Astropy. We conclude by raising and discussing the current and future challenges facing the project.

Keywords: Astrophysics - Instrumentation and Methods for Astrophysics — methods: data analysis — methods: miscellaneous

1. INTRODUCTION

Author: *Adrian Price-Whelan*

The `Python` programming language is a high-level, interpreted (as opposed to compiled) programming language that has become an industry standard across many computational domains, technological sectors, and fields of research. Despite claims to the contrary (Portegies Zwart 2020), `Python` enables scalable, time- and energy-

Corresponding author: Astropy Coordination Committee
coordinators@astropy.org

The author list has two parts: the authors that made significant contributions to the writing of the paper in order of contribution, followed by contributors to the Astropy Project in alphabetical order. **The 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.

efficient code execution (e.g., Augier et al. 2021) with a focus on code readability, ease of use, and interoperability with other languages. Over the last decade, `Python` has grown enormously in popularity to become a dominant programming language in the astronomical and broader scientific communities. For example, Figure 1 shows the number of yearly full-text mentions of `Python` as compared to a few other programming languages in refereed articles in the astronomical literature, demonstrating its nearly exponential growth in popularity. The rapid adoption of `Python` by astronomy researchers, students, observatories, and technical staff combined with an associated increase in awareness and interest about open-source software tools is contributing to a paradigm shift in the way research is done, data is analyzed, and results are shared in astronomy and beyond.

One of the factors that has led to its rapid ascent in popularity in scientific contexts has been the significant, volunteer-driven effort behind developing community-oriented open-source software tools and fostering communities of users and developers that have grown around these efforts. Today, a broad and feature-diverse "ecosystem" of packages exists in the `Python` scientific computing landscape: Roughly ordered from general-use to domain-specific, this landscape now includes packages that provide core numerical analysis functionality like `numpy` (Harris et al. 2020) and `scipy` (Jones et al. 2001–), visualization frameworks like `matplotlib` (Hunter 2007), machine learning and data analysis packages like `tensorflow` (Abadi et al. 2015), `pymc3` (Salvatier et al. 2016), and `emcee` (Foreman-Mackey et al. 2013), domain-specific libraries like `yt` (Turk et al. 2011), `plasmapy` (Community et al. 2021), `sunpy` (The SunPy Community et al. 2020), `Biopython` (Cock et al. 2009), and `sympy` (?) (to name a few in each category). The `astropy` (Astropy Collaboration et al. 2013, 2018) core package began in this vein, as an effort to consolidate the development of commonly-used functionality needed to perform astronomical research into a community-developed `Python` package.

The `astropy` core package was one of the first large, open-source `Python` packages developed for astronomy and provides, among other things, software functionality for reading and writing astronomy-specific data formats (e.g., FITS), transforming and representing astronomical coordinates, and representing and propagating physical units in code. An early description of the core functionality in `astropy` can be found in the first Astropy paper (Astropy Collaboration et al. 2013) or in detail in the core package documentation.[1] The `astropy` core package is now largely stable in that the software interface does not change without sufficient and significant motivation, and the addition of new features into the core package has slowed significantly as compared to the first years of its development. This is largely driven by the fact that the core package now represents just one piece of the broader astronomy `Python` context, and much new feature development is now happening in more specialized

[1] https://docs.astropy.org/

Figure 1. Yearly full-text mentions of programming languages (indicated in the figure legend) in refereed publications in the astronomical literature database in the Astrophysics Data System (ADS; ?). `Python` has rapidly become the dominant programming language mentioned in refereed articles over the last 10 years.

packages that are expanding the capabilities of the Astropy ecosystem by building on top of the foundations laid by the `astropy` core package. Because of this natural expansion, the name Astropy has grown in scope beyond a single `Python` library to become "the Astropy Project."

The Astropy Project is a community effort that represents the union of the `astropy` core package, the ecosystem of astronomy-specific software tools that are interoperable with `astropy` (Astropy Affiliated Packages), *and* the community of users, developers, and maintainers that participate in Astropy efforts. However, there is no institution responsible for managing the Astropy Project, for funding or maintaining its development, or sustaining it into the future: The Project is maintained and coordinated largely by volunteers. While new Astropy-affiliated packages are being developed that expand upon the core functionality in the `astropy` package, representing a natural expansion of the Astropy Project ecosystem, the needs of and challenges faced by the Project are evolving. In particular, the transition from focusing our energy on development and maintenance of a single core package, to instead sustaining the core package and fostering the development of the community and its expansion has been a key issue faced by the Astropy Project in the last several years.

In this Article, we briefly describe recent key updates in the `astropy` core package since the last Astropy paper ("Paper II"; Astropy Collaboration et al. 2018), major updates in the governance, contributor base, and funding of the Project, and discuss some of the future plans and challenges faced by the Astropy Project.

2. MAJOR UPDATES TO THE ASTROPY CORE PACKAGE

2.1. *New Long-term Support (LTS) Version: v5.0*

Author: *Tom Robitaille*

Major versions of the core package – that is to say versions which add and/or modify functionality – are released approximately every six months, and are then maintained with releases that fix issues, until the next major version is released. However, every two years a major release is designated as a long-term support (LTS) release which continues to be maintained for up to two years (Tollerud 2013). The motivation for LTS releases is to provide longer-term stable versions of astropy that users requiring a high level of stability can make use of if they do not always need the latest features (this could include for example telescope pipelines and so on). The 5.0 release of the core package was designated as LTS, and since it was released at the end of 2021, it will be maintained until the end of 2023.

2.2. *Highlighted Feature Development*

Author: *Nathaniel Starkman, Marten van Kerkwijk*

The `astropy` core package is mature and stable, in that many features have been part of the core package for years with purposefully few changes to the software interface. This maturity and stability allows for the broader astronomy `Python` community to rely upon the `astropy` core package and build specialized packages within the Astropy ecosystem. Even many non-astronomical `Python` libraries have come to rely on `astropy`. The relationship between Astropy and the `Python`-using community is reciprocal, with the evolving needs of the community and the maturation of the broader scientific `Python` ecosystem driving much of the development of the `astropy` core package.

Within `astropy`, development may be roughly split into a few categories: new features added, intra and inter-package interoperability, improvements to precision, accuracy, and reproducibility, and performance enhancements. We discuss these in turn.

Additionally, some features have been moved from the core package, either because they are of more general use outside of `astropy`, or because they are too specialized and belong in an Astropy-affiliated package. An example of the former is the copy of the IAU Standards Of Fundamental Astronomy (SOFA) software Collection[2] (Hohenkerk 2011) that Astropy carried, as well as the `Python` wrappers for it. Since this is basic infrastructure, with a release cycle set by SOFA, it made more sense to create separate packages that serve as `astropy` dependencies: `ERFA` (Tollerud et al. 2021) and `PyERFA` (van Kerkwijk et al. 2021). An example of something that was too specialized was a Virtual Observatory sub-package; the Simple Application Message Protocol made sense to keep (`astropy.samp`), but other parts more properly belonged in `astroquery` (see Sect. 4.3).

NEW AND PLANNED FEATURES

New features present a particular difficulty for a mature package such as `astropy`, because it is difficult to know *a priori* what the best interface will be, and hard to address all possible use cases in one round of development, even within `astropy` itself. New features come in various forms, from new submodules providing wholly new capabilities, to larger additions to existing submodules, to large rewrites of the interface of modules. This range is spanned by five main new features that entered the `astropy` core since the previous summary (Paper II). In our descriptions of each we include how we hope the new feature will evolve. Ultimately, in a user-driven project like Astropy, development depends primarily on the priorities of contributors.

Uncertainties and Distributions—Early in our careers we were likely all taught that measurements without units and uncertainties are meaningless. `astropy.units` provides the ability to associate numbers with units, and propagate these correctly, using

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

the Quantity class. However, the ability to associate and propagate uncertainties is relatively limited: in astropy.nddata, there are options to associate errors with data arrays, but these are treated as independent, i.e., covariances are not tracked.

Error propagation is difficult, and often is best approximated using Monte Carlo methods. The new astropy.uncertainties sub-package is our first step towards enabling seamless error propagation. The package allows one to generate, for each variable, randomly drawn samples in a Distribution, and then propagate these by passing them through the normal analysis, producing a Distribution of final results that can be inspected. This Distribution can be any type of array, including a Quantity. The goal is to ensure Distribution can be seamlessly used to instantiate other astropy classes as well, such as SkyCoord, Time, and Cosmology.

A plan for future development of astropy.uncertainties is to also support error propagation, tracking covariances, for the case that uncertainties are normally distributed rather than approximated by Monte-Carlo means. One implementation problem to be solved is when to *stop* tracking covariances. For instance, a simple operation such as subtracting the mean from N data points implies that all data points are now covariant with each other, i.e., one has to carry $N \times N$ covariances. For a large image, that becomes not just pointless but also prohibitive in terms of memory and CPU usage.

Masked quantities—It can be useful to mask bad data. How one supports masked data involves a number of choices, such as whether one indicates data values are bad by a separate flag or by replacing bad data with a special value, such as "Not a Number" (NaN). In astropy, both approaches have been used: a flag for masked columns in Table, mask flags and bitmaps for N-dimensional data in astropy.nddata, and replacing elements with NaN in Time. However masked quantities were poorly supported, making it difficult to use masks in a QTable (in which Quantity is used for all columns with units).

Unfortunately, the MaskedArray class from numpy only works well with plain data arrays, hiding other metadata and attributes, for example the unit of a Quantity. Consequently, a new Masked class was designed, based on a framework very similar to that of Distribution, making it easy to create masked instances of other classes. In particular, Masked can be used to create masked quantities for QTable, enabling I/O of masked data from different file types.

The masked quantities work largely without any changes in higher-level objects such as SkyCoord, but work is still underway to expose the mask in those classes, as well as to use masked arrays in Time instead of replacing data with NaN.

Time series—From sampling a continuous variable at fixed times to counting events binned into time windows, many different areas of astrophysics require the manipulation of time series. The new astropy.timeseries sub-package extends the QTable class to tables of data as a function of time, where the data can either represent samples or averages over particular time bins. The new classes offer a number of special methods to manipulate time series (folding, resampling, etc.) and to read different data formats (such as *Kepler* lightcurves, etc.).

Also part of `astropy.timeseries` are common analysis routines, including Lomb-Scargle and box-least-squares periodogrammes.

Spectral Coordinates—Measurements are often taken at specific "spectral coordinates," be they frequencies, wavelengths, or photon energies. `Quantity` can represent these and convert between them via dedicated equivalencies. The new `SpectralCoord` builds on `Quantity` by providing a more straightforward interface: baking in these equivalencies and those for Doppler velocities. Furthermore, `SpectralCoord` can be made aware of the observer and target reference frames, allowing transformation from telescope-centric (or topocentric) frames to Barycentric or Local Standard of Rest (LSRK and LSRD) velocity frames.

A High-Level Interface to World Coordinate Systems—Astronomical data are often provided alongside information about the correspondence between "real-world" and pixel coordinates. This mapping is the essence of the "World Coordinate System (WCS)" concept. From its inception, the `astropy.wcs` sub-package allowed access to WCS information provided in, e.g., FITS files, but it became clear that other WCS standards and representations had to be supported for new missions and observatories (e.g., the James Webb Space Telescope and the Rubin Observatory). To harmonize these needs, a new high-level interface was created based on a formal design first proposed in an Astropy Proposal for Enhancement (APE 14). The hope is that by having a formal design, other packages implementing WCS objects can straightforwardly modify their classes to conform to the new interface or build thin wrappers that conform. The implementation in `astropy.wcs` interacts well with other `astropy` objects such as `SkyCoord` and `Time`.

INTEROPERABILITY

Because Astropy is modular and situated at the nexus between scientific computing and astronomy, interoperability is an important focus for Astropy, both within the various `astropy` sub-packages and with other `Python` libraries. If done right, all a person notices is that code using many different python features and libraries "just works." Here, we describe a few particular efforts towards this goal.

`numpy` on Units—`Quantity` is a backbone of Astropy, leveraging the power of `numpy` and adding units. Previously, many `numpy` functions would strip a `Quantity` of its units (or fail outright), limiting `Quantity`'s usefulness. Now, advances in `numpy` function overloading (e.g., NEP 18) mean `Quantity` works with almost all `numpy` (v1.17+) functions.

In the remaining gaps, the `numpy` and `Quantity` interoperability efforts are ongoing. For some functions, such as in `numpy`'s module for manipulating structured arrays, compatibility only requires extending the existing numpy-astropy bridge frameworks. Community interest, in the form of a Feature Request (or better yet Pull Request) would be sufficient to see this compatibility completed. For a few remaining functions, discussed further in the Astropy documentation, the numpy framework does not yet allow for full interoperability with Astropy. A goal of Astropy and the scientific Python community is to enhance and implement the frameworks, allowing Quantity to propagate units seamlessly across the whole ecosystem of numpy-like projects.

astropy on Units—Units were also integrated further within astropy. For an already-defined unit-less Model, e.g. those imported from another library, astropy.modeling can now coerce units. A new module astropy.cosmology.units has been added to the cosmology sub-package for defining and collecting cosmological units and equivalencies. The unit littleh and equivalency with_H0 was moved from the main astropy.units sub-package to astropy.cosmology.units. A new unit, redshift, has been added for tracking factors of cosmological redshift. To correctly use redshift units in dimensional analysis, redshift is treated as dimensionless by default. To convert between redshift and other cosmological distance measures, e.g. CMB temperature or comoving distance, the equivalency with_redshift has also been added. This equivalency is actually a composite of other equivalencies, which may be used separately.

Table Mixin Columns—Within astropy and Python there are numerous ways to represent and store array-valued data. Some of the differences are historical: table.Column and io.fits.Column are not the same. Some differences are computational: numpy, cupy, and dask arrays have almost identical APIs, but are optimized for different use cases. Lastly, some differences are inherent: Quantity, Time and SkyCoord represent fundamentally different types of objects.

We aim to make it possible for any array-valued data to be used as a column in a table. A well defined protocol for mixin-columns has been developed for astropy.table, allowing the original object to be used as a column with a familiar API, and to “round-trip” through tables with no loss of data or attributes. With this protocol, it is now possible to store Astropy native objects – including Time, Quantity, and SkyCoord– within a Tableand write these to various file formats, such as FITs. For objects not already covered by the mixin protocol, functions can be registered with Table to convert any array-like object into a mixin column. As an example, the mixin functions are used to integrate dask arrays with Table, allowing cloud-stored or cluster-scale data to be used as a column in a Table.

Astropy FITS in Time—The FITS standard was extended to rigorously describe time coordinates in the World Coordinate System (WCS) framework (Rots et al. 2015). Compared to other types of coordinates in WCS, time requires more metadata: format, scale, position, reference, etc. This metadata had to be manually specified and the nuances understood by the user. astropy.io.fits could read this data as a standard `Column` but would not interpret the time-related metadata and attributes. Through the support of the Google Summer of Code 2017 program[3], the `astropy.io.fits` package now interprets time data correctly, using `Time` as a 'mixin' column. For backward compatibility with manual systems, this feature may be turned off.

Persistent Storage—Astropy's efforts to increase support for column types in a table is mirrored by efforts to expand storage format options. New formats have been added, and existing formats updated to support more column types.

Astropy now supports reading and writing tables in the American Astronomical Society Journals' Machine-Readable Table (MRT) format. This ASCII format has long been missing and as added in this year's Google Summer of Code project adds MRT to `Table` I/O. In addition, `Table` may also read from and write to ASDF, Parquet, and QDP formats.

The ECSV standard has been updated to version 1.0, adding support three additional data subtypes. First, it adds multidimensional column data (both masked and unmasked) with fixed dimensions in all table cells. Second, it adds multidimensional column data with variable-dimension arrays similar to FITS variable-length arrays. Third, it adds object-type columns with simple `Python` objects.

As mentioned previously, `dask` arrays may be used as a mixin column in `Table`. Now, `dask` may also be the data array in `FITS HDU` and if written to disk, the array will be computed while written, avoiding excessive memory use. `Table` can now be appended to an existing FITS file, and `dask` mixin columns interoperate seamlessly between the two table types.

Unified I/O architecture—`astropy.io.registry` is a powerful way to define input and output (I/O) functions for `astropy` objects, such as reading from or writing to a file, following a given standard. Astropy uses this internally for the I/O methods in `Table` and `Cosmology`, and users can register custom I/O to extend the options on these classes.

As of Astropy 5.0, the I/O registry submodule has been generalized to enable a number of new use cases. for instance, the class-based architecture allows for the creation of custom registries. One application is in `astropy.cosmology`, which has two different *kinds* of registries for `Cosmology`: one for reading and writing files, and another for converting between python objects. Using `astropy.io.registry` for custom classes means a developer does not have to create a whole new I/O system and can offer to the user a unified and familiar API.

PRECISION, ACCURACY, & REPRODUCIBILITY

[3] To learn more about this project, please see the final report, A mixin protocol for seamless interoperability

An important focus of Astropy has been to increase the precision and accuracy of its functionality, while, where appropriate, making sure results obtained using older versions of `astropy` are reproducible. Within each section below we include in our descriptions how old results may be reproduced. The primary means are with options in configuration files and settings on runtime configuration objects called `ScienceState`. Notable improvements to `astropy` have been made to `astropy.time`, `astropy.constants`, `astropy.coordinates`, and `astropy.cosmology`, and we describe each in turn.

Time—In astronomy, time accuracy down to the (nano)second is frequently important. Whether for planning observations, crunching pulsar or VLBI data, missing seconds meaningfully impact results. Before, when using `Time` in `astropy.time` leap seconds had to be manually applied; now, leap seconds are applied automatically. Moreover, `astropy` updates internal time data-files to ensure that the correct leap second adjustment is always used. For reproducibility, e.g. with an old code, `LeapSeconds` may also be manually applied.

Constants—Measurements of physical constants (and the units defined from them) improve over time. Periodically the standardized systems of units and constants are updated to reflect these improvements. For instance, in 2019 the SI system was redefined (Newell & Tiesinga 2019), with an accompanying update to the physical constants in `SI/CODATA 2018`. `astropy.constants` now defaults to use the `SI/CODATA 2018` values, with the units in `astropy.units` based on these constants. Most of `astropy` and affiliate packages build upon `astropy.units`, so this update effects the entire Astropy ecosystem.

For reproducibility, `astropy` allows the constants' (and therefore units') definitions to be rolled back to prior values, e.g. to the `SI/CODATA 2014` values. For work sensitive to the values of the fundamental constants we recommend including with the work an Astropy configuration file, which specifies the set of constants used.

Cosmology—`astropy.cosmology` contains classes for representing cosmological models. Bundled with the classes are commonly used `Cosmology` realizations, e.g., best-fit measurements from WMAP (Bennett et al. 2003) and Planck (The Planck Collaboration 2006).

When these important missions publish new measurements, `astropy.cosmology` is updated to include these results as realizations of the appropriate class (generally `FlatLambdaCDM`). The models used by WMAP and Planck do not always exactly correspond to `astropy.cosmology` classes. For instance, the Planck 2018 results (Planck Collaboration et al. 2020) include massive neutrinos in $\Omega_{matter,0}$, while in `FlatLambdaCDM` this mass contribution is stored in a separate parameter. Consequently, while some parameter values appear different from the source paper, the `Cosmology` realizations correctly reproduce the cosmological models.

Since Astropy Collaboration et al. 2018 the following cosmology realizations have been added: WMAP First Year (Spergel et al. 2003); WMAP Three Year (Spergel et al. 2007); Planck 2015 (Planck Collaboration et al. 2016); and the Planck 2018 best-fit cosmological parameters (Planck Collaboration et al. 2020).

Astropy provides a configurable default cosmology, which is used in calculations done in a cosmological context. The default `Cosmology` has been updated to the Planck 2018 parameters. For reproducibility, this cosmology may be set to old defaults, such as the WMAP Three Year values. The default cosmology is dynamically configurable using `ScienceState`, meaning a set of calculations may be run using different assumed cosmologies, and results compared between the two.

Coordinates—Being able to describe the position of objects, i.e., their coordinates, is fundamental across many knowledge domains. `astropy.coordinates` allows one to work with low-level positional and velocity data all the way to high-level coordinate objects with reference frames, atmospheric information, and more. A central feature of `astropy.coordinates` is the ability to transform data between reference frames, e.g. `ICRS` (Arias et al. 1997) to `GalacticLSR` (Schönrich et al. 2010). However, there have been some historical limitations for spatially proximate transformations, impacting the usefulness of `astropy.coordinates` for ground-based telescopes and astronometry within the solar system. Transformations in the `AltAz` frame were reasonably precise for very distant objects, but wrong by up to several arcseconds for e.g., the location of the moon. Now these transformations are much more precise, down to the milliarcsecond level. Similar precision improvements were made to the Hour Angle-Declination frame transformations. Additionally, Ecliptic frames and associated transformations have been updated to correctly reflect the true and mean terminology.

For Galactic astronomers, the `Galactocentric` frame defaults have been updated to include more recent measurements (tabulated in the `frame_defaults.references` attribute). For reproducibility, old definitions are still available.

PERFORMANCE

As features are added and corner cases dealt with, performance often suffers. Hence, pull requests that improve performance are welcomed, and were even the main goal for one release cycle. For most sub-packages, performance was improved in the `Python` code, but for some, the most time-critical pieces were rewritten in C (e.g., for convolution of images, sigma-clipping, and converting time strings to binary). Furthermore, a particular effort was made to ensure `astropy` is thread-safe, so that it can be used on supercomputer clusters.

3. MAJOR UPDATES IN THE ASTROPY PROJECT

3.1. *Project governance*

Author: *Erik Tollerud*

As part of the process of developing Astropy into a long-term sustainable product, and to improve transparency and accountability, the Project agreed to write down and formalize our governance structure (partly supported by explicit funding for this purpose - see §3.4). At the 2019 Astropy Coordination Meeting, input was gathered from participants on what governance structures existed in the associated Open Source Software communities, and what would fit well with the needs of Astropy. This led into a "retreat" planned for March 2020, but due to the COVID-19 pandemic, this became a series of virtual meetings of the "Astropy Governance Working Group". This group drafted the APE 0 document (Aldcroft et al. 2021), which was then eventually ratified and implemented by the "Astropy Governance Implementation Working Group" in Fall 2021. While the process emphasized flexibility and the ability to adapt to changing circumstances, it is expected that this is the framework Astropy's governance will operate in for at least the medium-term future.

The APE0 (Aldcroft et al. 2021) document lays out the principles of this governance structure, so we refer the reader to that document for a more thorough description. However here we highlight some key elements. While many of these principles were already de facto true or have been discussed organically (and have been discussed in earlier papers in this series), the APE0-based governance aimed to provide a single place where the community can agree as a starting point. With this in mind, it highlights the developer and user community of the Astropy Project as the ultimate sources of authority, as well as the core principle of "do-ocracy" that those who do work for the Project (be it coding, training, or other less concrete contributions) gain more influence on the outputs of the Project by virtue of their effort. APE0 adds, however, the concept of "voting members" - a self-governed part of that community who are entrusted to elect the Coordination Committee. While this committee has existed from the inception of the project, APE0 establishes a formal voting process for this committee, and explicitly outlines the rights and responsibilities of the Coordination Committee. This role is mainly to facilitate consensus and act as the decision maker when other mechanisms have failed. However, it also includes powers that either require central authority or secrets (e.g., passwords), but APE0 also charges the Committee to devolve responsibilities and seek community input on these items as often as possible.

The first Coordination Committee election under these rules took place in Fall 2021, electing a mix of prior existing and new coordination committee members, and was contested in the sense of more candidates than available slots. This suggests the process is already working to serve the long-term interests of the committee to both spread the coordination effort, and to ensure it is not dominated by the same people for as long as the Project continues. While other, more fine-grained governance improvements are planned for the future, it is clear the foundation is now in place.

3.2. *Contributor base*

Figure 2. Placeholder figure!

Author: *Adrian Price-Whelan*

Overview and statistics of contributors. Highlight changes since v2.0.

3.3. *Inclusion, Diversity, and Equity Programs*

Author: *Lía Corrales*

With support from the Moore Foundation, the Astropy Project was allocated funding to support mentoring programs. In 2020 a call was made to submit proposals for IDE (Inclusion, Diversity, and Empowerment) initiatives for project-wide consideration on Github. This process was deemed the most "open" because it allowed for community wide feedback to focus and improve proposal initiatives. Two programs were selected via this process, described below.

Outreachy: Describe the outreachy project

No author assigned

Women of Color Code (WoCCode) is a peer-mentoring network for coders from traditionally marginalized groups, most notably women of color. Participants were invited to the WoCCode Slack space and encouraged to attend monthly webinars to share skills in the context of open source software libraries. Every other month, a guest speaker was invited to talk about their career path and share a skill. Program participants were solicited in the fall of 2020, yielding 73 applications from 17 countries (48% from the United States, 37% from Africa, and 15% from remaining continents). We selected participants who we identified as having a high potential for contributing to open source projects: intermediate to advanced programming skills with a vocal interest in contributing. Of the thirty applicants invited to the program, nineteen joined the community on Slack. Participants were organized into cohorts based on interest and each cohort was assigned one of three mentors that were also selected via open application. Mentors acted as a general resource to participants, gave one webinar, and organized a hack day. WoCCode also supported registration of two participants to attend the American Astronomical Society virtual summer 2021 meeting.

Participants rated the impact of the program as very high. In the final webinar, participants reported a change in their perception of coding in general, for example, accepting coding as something they can do for fun. Participants reported generally feeling comfortable asking questions and interacting with a community where "[e]veryone's thoughts are welcomed, no one is made to feel less important" and where one can enjoy "warm interactions with likeminded people." WoCCode is continuing into 2022 by broadening participation in the Slack space and publicly advertising the guest webinar events.

In addition to the supported mentoring initiatives, the Astropy project as a whole has taken steps to examine representation of marginalized groups within the project and search for avenues of improvement.

Astropy representation at national diversity conferences: Several members of the Astropy community attended virtual conferences of the National Society of Black Physicists (NSBP) and SACNAS (focusing on Hispanic and Native American scientists across all STEM fields). Astropy representatives noted that one underlying topic came up multiple times. A major barrier for persons who come from under-represented communities is the lack of resources and expertise that are necessary to train students. Writing PEP 8 compliant software and understanding the Git and Github workflow is not part of the standard Physics and Astrophysics curriculum. Additionally, the larger Astronomnical community is still in the process of transitioning towards Python based tools, making it difficult for students not currently under advisement by some one with extensive Python expertise to get involved with the open source Python community. Projects such as Learn Astropy could have a profound impact by empowering underrepresented groups because it provides a free, searchable, and accessible introduction to Python tools for astrophysical research. Representatives also noted that offering training and teaching materials to PIs of Research Experience for Undergraduate (REU) programs throughout the United States could be helpful for encouraging advisors to teach students how to use Astropy, create their own libraries, and use version control on open source platforms like Github. Such materials could also be offered as a workshop at national diversity conferences themselves, as SACNAS solicits special session proposals each year.

NumFOCUS "Contributor Diversification & Retention" (CDR) initiative: Describe more about this.

No author assigned

3.4. *Funding*

Author: *Aarya Patil*

With the goal of establishing a sustainable financial model for the Astropy Project, the Coordination Committee under the new governance charter (see §3.1) established a standing Finance committee. This committee oversee the planning and allocation of finances on behalf of the project. The Astropy Project accepts funds from institutions as well as individuals through NUMFOCUS[4]. Previously, no direct financial support was available for the project development, and NUMFOCUS covered most of the incurred operational costs. However, transitioning to long-term financial stability meant having an influx of funds in the form of institutional support, and hence the Astropy Project applied for and successfully acquired four grants. These grants are administered by NumFOCUS on behalf of the project, providing a "neutral" space not tied to any particular astronomy institution. The two major ones are briefly summarized below.

- Moore: The Gordon and Betty Moore Foundation awarded the Astropy project a ~$900k (US) grant in 2019 for the proposal "Sustaining and Growing the

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

Astropy Project". Over a three-year term, this grant supported sustenance of the project by providing funds for (1) the development and maintenance of the `astropy` and its infrastructure to existing project members and targeted hires, (2) the transition of long-term users to contributors and maintainers through mentorship efforts, (3) the formalization of a governance structure (see §3.1), and (4) the improvement of equity, diversity, and inclusion efforts in the project (see §3.3). Some of the money supported travel for meetings, conferences, and workshops. This generous grant helped reduce reliance on volunteer-driven maintenance of the openly-developed package and paved the way to acquire funding from federal agencies: a big milestone for the success of the project.

- NASA: A successful proposal to the NASA ROSES-2020 call[5], section E.7 (Support for Open Source Tools, Frameworks, and Libraries), granted the Astropy Project ∼$600k (US) to support its work. Funded work for a term of three years includes (1) project-wide infrastructure maintenance and improvements, (2) targeted work on areas of the Astropy core and coordinated packages that need support, (3) enhancements to the Learn Astropy ecosystem and educational materials (see §6), and (4) support for the Astropy affiliated packages. These goals support the sustenance of the project by encouraging further development of the package ecosystem and community engagement.

In addition to the two grants described above, the project benefited from a Gemini Observatory contract under its Science User Support Department, which awarded funds for development work that supports both the Astropy and DRAGONS projects. The project also received financial support from the Dunlap Seed Funding program at the University of Toronto to develop educational resources for scientific software packaging within the Learn Astropy framework. All these successful proposals show that the Astropy project is vital for the astronomical infrastructure and community at large, and strengthen its promising and sustainable future.

4. SUPPORTING THE ECOSYSTEM OF ASTRONOMICAL PYTHON SOFTWARE

4.1. *Community-oriented infrastructure*

Author: *Nicholas Earl*

The Astropy project supports the broader ecosystem by providing pre-configured infrastructure packages that the community can use to support and maintain their own software package infrastructure. These include tools to easily generate documentation and setup automated testing, as well as provide package scaffolding for new projects.

Sphinx is a common and useful tool for generating documentation for Python packages. The Astropy project maintains a default Sphinx configuration along with

[5] http://solicitation.nasaprs.com/ROSES2020

Astropy-specific extensions which can be easily added to community projects via the `sphinx-astropy` meta package. This tool provides a pre-configured Sphinx setup compatible with Astropy projects, which includes several extensions useful for generating API documentation (`sphinx-automodapi`), allowing for Numpy docstring parsing (`numpydoc`), embedded image handling (`sphinx-gallery`; `pillow`), advanced documentation testing support (`pytest-doctestplus`), and providing a custom documentation theme ideal for analysis packages (`astropy-sphinx-theme`).

Community package testing infrastructure is supported through the `pytest-astropy` meta-package, providing a unified testing framework with useful extensions compatible with both Astropy- and non-Astropy-affiliated community packages. This meta-package pulls in several `pytest` plugins to help with custom test headers (`pytest-astropy-header`), accessing remotely-hosted data files in tests (`pytest-remotedata`), interoperability with documentation (`pytest-doctestplus`), dangling file handle checking (`pytest-openfiles`), data array comparison support in tests (`pytest-arraydiff`), sub-package command-line testing support (`pytest-filter-subpackage`), improved mock object testing (`pytest-mock`), test coverage reports and measurements (`pytest-cov`), and configuring package for property-based testing (`hypothesis`).

The Astropy Package Template helps facilitate the setup and creation of new Python packages leveraging the Astropy ecosystem. This tool utilizes the Cookiecutter project to walk users through the process of creating new packages complete with documentation and testing support. Additionally, the package template generation process includes the ability to setup interoperability with GitHub, allowing for easy repository access from documentation, as well as an example GitHub Actions workflow to demonstrate the use of GitHub's continuous integration tooling.

4.2. *Affiliated packages*

Author: *Matt Craig, Brett Morris*

Highlight a few new affiliated packages and major updates to existing ones. Include a big table of all affiliated packages and references (as in v2.0 paper).

All coordinated and affiliated packages are listed on the astropy website[6]. Since Astropy Collaboration et al. (2018), there has been an expansion of affiliated packages for gravitational astrophysics including: `PyCBC` for exploring gravitational wave signals, `lenstronomy` for modeling strong gravitational lenses, `ligo.skymap` for visualizing gravitational wave probability maps, and `EinsteinPy` for general relativity and gravitational astronomy. There have been several packages added to the ecosystem related to HEALPix: `astropy-healpix` for a BSD-licensed HEALPix implementation, and `mocpy` for Multi-Order Coverage maps. `astroalign` has been introduced for astrometric registration, and `python-cpl` has been added for ESO pipelines and VLT data products. For ground-based astronomy, `baseband` has added IO capabilities for

[6] https://www.astropy.org/affiliated

VLBI, and `SpectraPy` brings slit spectroscopy to the astropy ecosystem. We have `agnpy` for AGN jets, and `statmorph` for fitting galactic morphological diagnostics. `saba` gives an interface to sherpa's fitting routines. For modeling interstellar dust extinction we have `dust_extinction`, and for extracting features from time series we have `feets`. We also have `corral` for managing data intensive parallel pipelines. `BayesicFitting` provides an interface for generic Bayesian inference, and `sbpy` Mommert et al. (2019) does calculations for asteroid and cometary astrophysics. Finally, `synphot` provides an interface for synthetic photometry.

Several of the coordinated and affiliated packages described in Astropy Collaboration et al. (2018) have had substantial improvements. `astroquery` Ginsburg et al. (2017) has added access to roughly a dozen new missions and data services, including JWST, and the project has switched to a continuous release model. Every time a change is committed to the main development branch it is published on the Python Package Index (PyPI, ref) and available for installation. Formal releases are still done a couple of times per year. `photutils` Bradley et al. (2021) released its first stable version, indicating that the API will change less frequently, and there have been several significant performance improvements. `ccdproc` Craig et al. (2015) also released a new major version, bringing better performance to some image combination operations. The `regions` package pyregions developers (2018), for manipulating ds9-style region definitions Joye & Mandel (2003), added new ways to manipulate regions and introduced new region types. `reprojects` Robitaille (2018) major new feature is a function to align and co-add images to create a mosaic; better support for parallelization was also added. `specutils`, the package that defines containers for 1D and 2D spectra Earl et al. (2021), also had its first stable release and the addition of classes to read JWST data. `stingray` has also had a major performance overhaul. The package `gammapy` Deil et al. (2017) also has major performance improvements, has unified its API in preparation for its first stable release.

4.3. *Connections with data archives*

Author: *Adam Ginsburg*

Astroquery Ginsburg et al. (2019) is the astropy-coordinated package for interacting with online archives of astronomical and related data. It contains over 50 modules for querying astronomical databases, large and small. In particular, since Astropy Collaboration et al. (2018), significant contributions to `astroquery` have come from several of the major archives, including the European Space Agency (ESA), the Mikulski Archive for Space Telescopes (MAST) at the Space Telescope Science Institute (STScI), the Infrared Science Archive at the NASA Infrared Processing and Analysis Center (IRSA at NASA IPAC), the Canadian Astronomical Data Center (CADC), and the Atacama Large Millimeter/Submillimeter Array archive (ALMA). These contributions represent a formal acknowledgement of the utility of a centralized tool suite for archive interaction.

The widespread usage of astroquery is apparent from the range of keywords represented in the journal articles which cite Ginsburg et al. (2019); everything from astroids to galaxies is represented. Additionally Astroquery functionality has been built into several special purpose Python packages, such as LightKurve Lightkurve Collaboration et al. (2018) and SORA Gomes-Júnior et al. (2022)]. More than 4,000 repositories on GitHub make use of astroquery in some way, an astroquery features in a large number of tutorials on various facets of astronomical analysis.

Many of the existing and newly-contributed tools rely on Virtual Observatory (VO) tools. These use the underlying package pyvo, which has also recently become an astropy-coordinated package. While many or all of the functions provided in astroquery can be achieved through direct use of VO tools implemented in pyvo, the astroquery interfaces more closely resemble the web interfaces more familiar to users.

4.4. *Connections with Observatories and Missions*

Brief summary of efforts in observatory- or mission-driven development that have contributed to Astropy, and vice versa.

4.4.1. *James Webb Space Telescope*

Author: *Larry Bradley*

The James Webb Space Telescope (JWST) is a 6.5-meter space-based infrared telescope that will provide unprecedented resolution and sensitivity from 0.6 – 28 microns. JWST will enable a broad range of scientific investigations from exoplanets and their atmospheres to the formation of galaxies in the very early universe. Its four key scientific goals are to study the first light from stars and galaxies, the assembly and evolution of galaxies, the birth of stars and protoplanetary systems, and planetary systems and the origins of life.

The telescope launched on an Ariane 5 rocket on 2021 December 25 from Kourou, French Guiana. After a series of successful deployments, including the sunshield and primary and secondary mirrors, JWST reached its orbit around the L2 Lagrange point on 2022 January 24. Commissioning of the telescope optics and science instruments will occur from January until the end of June 2022, when science operations are scheduled to begin.

Software developers at the Space Telescope Science Institute (STScI), the operations center of JWST, have been developing Python-based tools for JWST since 2010 (starting with the JWST Calibration Reference Data System) and have provided major contributions to Astropy from its inception. The JWST instrument calibration pipelines, exposure-time calculators, and data analysis tools are all written in Python and depend on the astropy core package and some coordinated and affiliated packages. For the astropy core package, the JWST mission has provided extensive contributions to the astropy.modeling, astropy.units, astropy.coordinates, astropy.wcs, astropy.io.fits, astropy.io.votable, `astropy.stats`, `astropy.visualization`, and `astropy.convolution` subpackages as well as to the general package infrastructure and maintenance.

Likewise, JWST developers have provided significant contributions to the `photutils` (Bradley et al. 2021), `specutils` (Earl et al. 2021), and `regions` (Bradley et al. 2022) coordinated packages and the `gwcs` (Dencheva et al. 2021) and `synphot` (STScI Development Team 2018) affiliated packages. For example, development of the `photutils` coordinated package for source detection and photometry has largely been led by JWST contributions. JWST developers have also made significant contributions to the `specutils` coordinated package, which is used for analyzing spectroscopic data, and the `regions` coordinated package, which is used to handle geometric regions. The `gwcs` affiliated package for generalized world coordinate systems was created specifically to handle the complex world coordinate systems needed for JWST spectroscopic data. The `synphot` affiliated package for synthetic photometry was created at STScI and is a dependency of the JWST exposure-time calculators. Further, the `ASDF` (Advanced Scientific Data Format) package (Greenfield et al. 2015), a next-generation interchange format for scientific data, was initially developed at STScI to serialize JWST WCS objects along with `astropy` models, units, and coordinates. Over its mission lifetime, JWST will continue its support in developing and maintaining these critically dependent packages.

Gemini Author: *Looking for volunteers!*

Cherenkov Telescope Array Author: *Axel Donath, Maximilian Nöthe*

The Cherenkov Telescope (CTA) will be the next generation very-high-energy gamma-ray observatory. CTA will improve over the current generation of imaging atmospheric Cherenkov telescopes (IACTs) by a factor of five to ten in sensitivity and will be able to observe the whole sky from a combination of two sites: a northern site in La Palma, Spain, and a southern one in Paranal, Chile. CTA will be able to observe gamma rays in a broad energy range from around 20 GeV to over 300 TeV using three different types of telescopes, in total over 100 telescopes are planned at the two sites. CTA will also be the first open gamma-ray observatory.

The data analysis pipeline is developed as open source software and essentially split in two domains:

1. In the low-level analysis, the properties of the recorded air-shower events have to be estimated from the raw data. The raw data consists of very short ($\sim 40 - -100\,\mathrm{ns}$) videos recorded with the fast and sensitive cameras of the telescopes. This includes the energy, particle type and direction of origin of the particle that induced the air shower and the time the shower was recorded.

2. In the higher-level analysis, these reconstructed event lists are used together with some characterization of the instrument response to perform the actual scientific analysis. This software will be delivered as CTA science tools to the future users of the Observatory.

A prototype for the low-level analysis is `ctapipe` (Nöthe et al. 2021), a python package developed to perform all the necessary tasks to from the raw data of Cherenkov telescopes to the reconstructed event lists.

The high-level analysis (or CTA science tools) will be based on the astropy affiliated package Gammapy (Deil et al. 2017).

Both Gammapy and `ctapipe` are using astropy heavily, mainly for units, times, coordinate transformations, tables and FITS IO.

As CTA will record gamma-ray events with a rate of up to 10 000 events per second, it needs to perform a large number of coordinate transformations. To enable this, CTA member M. Nöthe contributed a major performance improvement for large numbers of coordinates with different observation times, based on earlier work by B. Winkel.

Together with Gammapy maintainer A. Donath and former maintainer C. Deil, a total of 107 merged pull requests were contributed to astropy.

Rubin Observatory Author: *Looking for volunteers!*

LIGO/Virgo/KAGRA Author: *Leo Singer*

5. FUTURE PLANS FOR THE ASTROPY PROJECT

5.1. *Roadmap*

Author: *Clara Brasseur*

The Astropy Roadmap[7] is a document in the astropy-project repository that captures high level actionable items that the Astropy project aims to undertake to improve the health and stability of the project. It is a static document that is revisited regularly at the Astropy coordination meetings, to keep track of progress and write new versions as needed. There is a related project board linked to the Roadmap document, that holds specific issues and efforts related to Roadmap items. The project board is a living document that is continually updated as work is planned, assigned, and completed.

All items in the Roadmap have been agreed to be priorities for the Astropy Project, and are color-coded based on resources (both time/effort and developers/expertise) needed to complete the item. Green items are well underway, have sufficient resources/support and a plan in place for completion. Orange items are well defined, and work for aquiring sufficient resources underway. Red items do not yet have a plan for implementation and need more resources. Add some examples of high priority items from the roadmap in a new paragraph here The Astropy Roadmap originates from the March 2021 Astropy Coordination meeting where it was first drafted before being handed off to a newly formed Astropy Roadmap working group for completion. The Roadmap in its currant form was adopted via pull request in December 2021.

addition

6. LEARN ASTROPY

Author: *Lía Corrales, David Shupe, Kelle Cruz + Learn team*

[7] https://github.com/astropy/astropy-project/blob/main/roadmap/roadmap.md

6.0.1. *Current status and scope*

Learn Astropy is an umbrella term that acknowledges the broad educational efforts made by the Astropy Project, which are led by the Learn Team. The efforts focus on developing online content and workshops covering astronomy-specific coding tasks in Python. As introduced in Astropy Collaboration et al. (2018), there are four different types of Learn Astropy content: *tutorials*, consisting of Jupyter Notebook lessons that are published in HTML format online; *guides*, which are a series of lessons providing a foundational resource for performing certain type of astronomical analyses; *examples*, which are snippets of code that showcase a short task that can be performed with Astropy or an affiliated package; and *documentation*, which is contained within the code base. This categorization drives content development, infrastructure choices, and the appearance of the Learn Astropy website. The Learn Team meets weekly to work on creating, expanding, improving these educational resources.

Learn Astropy website: The Learn Team re-launched the main website and search interface for Learn Astropy in 2021 with a new infrastructure platform, built around full-text search and interactive filtering functionality, with the goal of making content discoverable as the Learn Astropy content catalog expands. This work has been supported in part by a grant from the Dunlap Institute. We have adopted Algolia, a search-as-a-service cloud platform, to store the full-text and metadata records of Learn Astropy's content. The new Learn Astropy website is a JavaScript (Gatsby/React) application that uses the Algolia service to power its search and filtering user interface. Our Python-based application, Learn Astropy Librarian, populates data into the Algolia service. We tuned the Librarian around specific content formats (such as Jupyter Notebook-based tutorial pages and Jupyter Book-based guides) to more accurately index content and heuristically extract metadata. A consequence of the new platform is that we now maintain and compile content separately from the website application itself, enabling new content types. Tutorials, which are written as Jupyter Notebooks, are now compiled into their own Learn Astropy sub-site using `nbcollection`. Guides, which utilize the Jupyter Book build infrastructure, are also deployed as separate websites using GitHub pages. This architecture opens future possibilities of indexing third-party content, hosted elsewhere, such as on institutional websites.

Tutorials: Currently, we have # of tutorials, spanning a wide range of astronomical topics and common tasks.

Guides: *ccdproc* Jupyter book has been completed.

Workshops: The Project has been conducting workshops at winter meetings of the American Astronomical Society since AAS 225 in January 2015. Up to the start of the coronavirus pandemic, these were full-day in-person workshops with as many as 90 participants and a dozen facilitators from the project. During the pandemic, these workshops were moved to an online format and split into basic and advanced sessions. Additionally, beginning with the AAS 238 online meeting, the workshops have been expanded to Summer AAS meetings. The Learn team finds that the workshop audience is best found as AAS meetings as opposed to more general Python meetings.

The Astropy Project provided another mode of community engagement at AAS Meetings 235 and 237 by organizing a NumFOCUS Sponsored Projects booth in the AAS Exhibit Hall. Funding for the exhibit hall was provided alternately by NumFOCUS and later by the Moore Foundation funding (**check this**). The booth hosted a series of Q&A special sessions during AAS 235 and webinars during the virtual AAS 237 meetings, to provide the general astronomy community information and access to experts on a variety of open source astronomical tools.

6.0.2. *Learn vision for the future*

The Learn plan going forward is to continue to improve the tutorial website and facilate new content; to index third-party tutorials; and to look for supportable opportunities to expand the reach of Astropy workshops beyond the American Astronomical Society meetings.

6.1. *Current and Future Challenges*

7. COMMUNITY ENGAGEMENT

- **User forums:** The Astropy Project has historically maintained a number of avenues for users to seek help or gain access to the developer community. This includes the Astropy mailing list, the Python in Astronomy Facebook group, and the Astropy Slack workspace. The Moore Foundation grant allows this Slack space to be on a paid plan; additionally NumFOCUS has negotiated a special rate for open-source projects. While the Slack and Astropy-dev mailing lists are primarily used to discuss the project direction and updates, it was noted that the use of a Facebook community could present a barrier to open source. The Astropy Project identified a need for a public, archived, searchable, and easily-to-navigate interface for users to ask for help Accordingly, we have commissioned a Discourse site which is more open than the Facebook group and more user-oriented than Astropy Slack. A benefit of the Moore Foundation grant is that Astropy developers are able to invoice as independent contractors the time they spend helping users on these forums.
- Attracting new contributors when the code has become quite complex,
- Contributor to maintainer mentoring,
- Long-term / sustained funding for maintaining infrastructure,

- ...

We would like to thank the members of the community who have contributed to Astropy, who have opened issues and provided feedback, and have supported the project in a number of different ways.

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 severals 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 that astronomers and other developers on their staff can contribute their time to the development of Astropy projects. We would like acknowledge the support of the Space Telescope Science Institute, Harvard–Smithsonian Center for Astrophysics, and the South African Astronomical Observatory.

Supported by the international Gemini Observatory, a program of NSF's NOIRLab, which is managed by the Association of Universities for Research in Astronomy (AURA) under a cooperative agreement with the National Science Foundation, on behalf of the Gemini partnership of Argentina, Brazil, Canada, Chile, the Republic of Korea, and the United States of America.

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, Azure Pipelines, 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. 2021), which in turn derives from the IAU SOFA Collection[8] developed by the International Astronomical Union Standards of Fundamental Astronomy (Hohenkerk 2011).

Software: `astropy` (Astropy Collaboration et al. 2013, 2018), `numpy` (Harris et al. 2020), `scipy` (Jones et al. 2001–), `matplotlib` (Hunter 2007), `Cython` (Behnel et al. 2011), `photutils` (Bradley et al. 2021), `specutils` (Earl et al. 2021), `regions` (Bradley et al. 2022), `gwcs` (Dencheva et al. 2021), `synphot` (STScI Development Team 2018), `ASDF` (Greenfield et al. 2015).

REFERENCES

Abadi, M., Agarwal, A., Barham, P., et al. 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/

Aldcroft, T., Craig, M., Cruz, K., et al. 2021, Astropy Proposal for Enhancement 0: The Astropy Project Governance Charter (APE 0), doi: 10.5281/zenodo.4552791

Arias, E. F., Charlot, P., Feissel, M., & Lestrade, J. F. 1997, IERS Technical Note, 23, IV

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33, doi: 10.1051/0004-6361/201322068

Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., et al. 2018, AJ, 156, 123, doi: 10.3847/1538-3881/aabc4f

Augier, P., Bolz-Tereick, C. F., Guelton, S., & Mohanan, A. V. 2021, Nature Astronomy, 5, 334, doi: 10.1038/s41550-021-01342-y

Behnel, S., Bradshaw, R., Citro, C., et al. 2011, Computing in Science Engineering, 13, 31 , doi: 10.1109/MCSE.2010.118

Bennett, C. L., Bay, M., Halpern, M., et al. 2003, ApJ, 583, 1, doi: 10.1086/345346

Bradley, L., Sipcz, B., Robitaille, T., et al. 2021, astropy/photutils: 1.3.0, 1.3.0, Zenodo, doi: 10.5281/zenodo.5796924

Bradley, L., Deil, C., Patra, S., et al. 2022, astropy/regions: v0.5, v0.5, Zenodo, doi: 10.5281/zenodo.5826359

Cock, P. J. A., Antao, T., Chang, J. T., et al. 2009, Bioinformatics, 25, 1422, doi: 10.1093/bioinformatics/btp163

Community, P., Everson, E., Staczak, D., et al. 2021, PlasmaPy, 0.6.0, Zenodo, doi: 10.5281/zenodo.4602818

Craig, M. W., Crawford, S. M., Deil, C., et al. 2015, ccdproc: CCD data reduction software, Astrophysics Source Code Library. http://ascl.net/1510.007

Deil, C., Zanin, R., Lefaucheur, J., et al. 2017, ArXiv e-prints. https://arxiv.org/abs/1709.01751

Dencheva, N., Mumford, S., Cara, M., et al. 2021, spacetelescope/gwcs: GWCS 0.18.0, 0.18.0, Zenodo, doi: 10.5281/zenodo.5800080

Earl, N., Tollerud, E., Jones, C., et al. 2021, astropy/specutils: V1.5.0, v1.5.0, Zenodo, doi: 10.5281/zenodo.5721652

Foreman-Mackey, D., Hogg, D. W., Lang, D., & Goodman, J. 2013, PASP, 125, 306, doi: 10.1086/670067

Ginsburg, A., Sipocz, B., Parikh, M., et al. 2017, astropy/astroquery: v0.3.6 with fixed license, doi: 10.5281/zenodo.826911

Ginsburg, A., Sipőcz, B. M., Brasseur, C. E., et al. 2019, AJ, 157, 98, doi: 10.3847/1538-3881/aafc33

Gomes-Júnior, A. R., Morgado, B. E., Benedetti-Rossi, G., et al. 2022, MNRAS, 511, 1167, doi: 10.1093/mnras/stac032

Greenfield, P., Droettboom, M., & Bray, E. 2015, Astronomy and Computing, 12, 240, doi: 10.1016/j.ascom.2015.06.004

Harris, C. R., Millman, K. J., van der Walt, S. J., et al. 2020, Nature, 585, 357, doi: 10.1038/s41586-020-2649-2

Hohenkerk, C. 2011, Scholarpedia, 6, 11404, doi: 10.4249/scholarpedia.11404

Hunter, J. D. 2007, Computing In Science & Engineering, 9, 90, doi: 10.1109/MCSE.2007.55

Jones, E., Oliphant, T., Peterson, P., et al. 2001–, SciPy: Open source scientific tools for Python. http://www.scipy.org/

Joye, W. A., & Mandel, E. 2003, in Astronomical Society of the Pacific Conference Series, Vol. 295, Astronomical Data Analysis Software and Systems XII, ed. H. E. Payne, R. I. Jedrzejewski, & R. N. Hook, 489

Lightkurve Collaboration, Cardoso, J. V. d. M., Hedges, C., et al. 2018, Lightkurve: Kepler and TESS time series analysis in Python, Astrophysics Source Code Library. http://ascl.net/1812.013

Mommert, M., p. Kelley, M. S., de Val-Borro, M., et al. 2019, Journal of Open Source Software, 4, 1426, doi: 10.21105/joss.01426

Newell, D. B., & Tiesinga, E. 2019, The international system of units (SI):, Tech. rep., doi: 10.6028/nist.sp.330-2019

Nöthe, M., Kosack, K., Nickel, L., & Peresano, M. 2021, arXiv e-prints, arXiv:2110.11097. https://arxiv.org/abs/2110.11097

Planck Collaboration, Ade, P. A. R., Aghanim, N., et al. 2016, A&A, 594, A13, doi: 10.1051/0004-6361/201525830

Planck Collaboration, Aghanim, N., Akrami, Y., et al. 2020, A&A, 641, A6, doi: 10.1051/0004-6361/201833910

Portegies Zwart, S. 2020, Nature Astronomy, 4, 819, doi: 10.1038/s41550-020-1208-y

pyregions developers. 2018, regions – ds9 region parser for python, https://github.com/astropy/pyregions

Robitaille, T. 2018, reproject: astronomical image reprojection in Python, doi: 10.5281/zenodo.1162674

Rots, A. H., Bunclark, P. S., Calabretta, M. R., et al. 2015, A&A, 574, A36, doi: 10.1051/0004-6361/201424653

Salvatier, J., Wiecki, T. V., & Fonnesbeck, C. 2016, PeerJ Computer Science, 2, e55, doi: 10.7717/peerj-cs.55

Schönrich, R., Binney, J., & Dehnen, W. 2010, MNRAS, 403, 1829, doi: 10.1111/j.1365-2966.2010.16253.x

Spergel, D. N., Verde, L., Peiris, H. V., et al. 2003, ApJS, 148, 175, doi: 10.1086/377226

Spergel, D. N., Bean, R., Doré, O., et al. 2007, ApJS, 170, 377, doi: 10.1086/513700

STScI Development Team. 2018, synphot: Synthetic photometry using Astropy. http://ascl.net/1811.001

The Planck Collaboration. 2006, arXiv e-prints, astro. https://arxiv.org/abs/astro-ph/0604069

The SunPy Community, Barnes, W. T., Bobra, M. G., et al. 2020, The Astrophysical Journal, 890, 68, doi: 10.3847/1538-4357/ab4f7a

Tollerud, E. 2013, Astropy Proposal for Enhancement 2: Astropy Release Cycle and Version Numbering (APE 2), doi: 10.5281/zenodo.1043888

Tollerud, E., Pascual, S., van Kerkwijk, M. H., et al. 2021, liberfa/erfa: v2.0.0, Zenodo, doi: 10.5281/zenodo.1021148

Turk, M. J., Smith, B. D., Oishi, J. S., et al. 2011, ApJS, 192, 9, doi: 10.1088/0067-0049/192/1/9

van Kerkwijk, M. H., Tollerud, E., Valentino, A., et al. 2021, liberfa/pyerfa: v2.0.0, Zenodo, doi: 10.5281/zenodo.3940698