Forward Modeling of Large-scale Structure: An Open-source Approach with Halotools

Andrew P. Hearin1,2,3,4, Duncan Campbell2, Erik Tollerud2,3, Peter Behroozi4,5, Benedikt Diemer6, Nathan J. Goldbaum6,7, Elise Jennings4,8, Alexie Leauthaud9,10,11, Yao-Yuan Mao10,11, Surhud More9,12, John Parejko12, Manodeep Sinha13,14, Brigitta Sipőcz15,16, and Andrew Zentner11

1 Department of Astronomy, Yale University, P.O. Box 208101, New Haven, CT, USA
2 Yale Center for Astronomy & Astrophysics, Yale University, New Haven, CT, USA
3 Space Telescope Science Institute, Baltimore, MD 21218, USA
4 Department of Astronomy, University of California, Berkeley, Berkeley CA 94720, USA
5 Institute for Theory and Computation, Harvard-Smithsonian Center for Astrophysics, 60 Garden Street, Cambridge, MA 02138, USA
6 National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 1205 W. Clark Street, Urbana, IL 61801, USA
7 Center for Particle Astrophysics, Fermi National Accelerator Laboratory MS209, P.O. Box 500, Kirk Road & Pine Street, Batavia, IL 60510-0500, USA
8 Kavli Institute for Cosmological Physics, Enrico Fermi Institute, University of Chicago, Chicago, IL 60637, USA
9 Kavli IPMU (WPI), UTIAS, The University of Tokyo, Kashiwa, Chiba 277-8583, Japan
10 Kavli Institute for Particle Astrophysics and Cosmology and Department of Physics, Stanford University, Stanford, CA 94305, USA
11 Department of Physics and Astronomy and Pittsburgh Particle Physics, Astrophysics, and Cosmology Center (PIT C PACC), University of Pittsburgh, Pittsburgh, PA 15260, USA
12 University of Washington, Dept. of Astronomy, Box 351580, Seattle, WA 98195, USA
13 Department of Physics & Astronomy, 6301 Stevenson Center, Vanderbilt University, Nashville, TN 37235, USA
14 Centre for Astrophysics & Supercomputing, Swinburne University of Technology, 1 Alfred Street, Hawthorn, VIC 3122, Australia
15 Centre for Astrophysics Research, University of Hertfordshire,College Lane, Hatfield, AL10 9AB, UK
16 Institute of Astronomy, University of Cambridge, Madingley Road, Cambridge, CB3 0HA, UK
17 Argonne National Laboratory, Argonne, IL 60439, USA

Received 2016 June 13; revised 2017 July 28; accepted 2017 August 8; published 2017 October 20

Abstract

We present the first stable release of Halotools (v0.2), a community-driven Python package designed to build and test models of the galaxy-halo connection. Halotools provides a modular platform for creating mock universes of galaxies starting from a catalog of dark matter halos obtained from a cosmological simulation. The package supports many of the common forms used to describe galaxy-halo models: the halo occupation distribution, the conditional luminosity function, abundance matching, and alternatives to these models that include effects such as environmental quenching or variable galaxy assembly bias. Satellite galaxies can be modeled to live in subhalos or to follow custom number density profiles within their halos, including spatial and/or velocity bias with respect to the dark matter profile. The package has an optimized toolkit to make mock observations on a synthetic galaxy population—including galaxy clustering, galaxy–galaxy lensing, galaxy group identification, RSD multipoles, void statistics, pairwise velocities and others—allowing direct comparison to observations. Halotools is object-oriented, enabling complex models to be built from a set of simple, interchangeable components, including those of your own creation. Halotools has an automated testing suite and is exhaustively documented on http://halotools.readthedocs.io, which includes quickstart guides, source code notes and a large collection of tutorials. The documentation is effectively an online textbook on how to build and study empirical models of galaxy formation with Python.

Key words: cosmology: theory – galaxies: halos – galaxies: statistics – large-scale structure of universe

1. Introduction

Empirical modeling of the galaxy-halo connection has become a mature subfield of galaxy formation studies. The foundation of this subfield is the halo model (Ma & Fry 2000; Seljak 2000), which gives an approximate description of the large-scale density field by supposing that all matter is bound within collapsed regions called dark matter halos (see Cooray & Sheth 2002; Mo et al. 2010, for reviews). The abundance, internal structure, and spatial distribution of dark matter halos can be predicted via simulations with high precision (e.g., Tinker et al. 2008, 2010; Bhattacharya et al. 2011, 2013; Diemer & Kravtsov 2015; Heitmann et al. 2016). Because the centers of dark matter halos are the natural sites of galaxy formation (White & Rees 1978), then quantitative predictions for the statistical distribution of galaxies are enabled by knowledge of how galaxy properties are connected to the properties of their underlying halos.

The modern methods of galaxy-halo modeling has been in place for over ten years, including formulations such as the Halo Occupation Distribution (HOD, Berlind & Weinberg 2002), the Conditional Luminosity Function (CLF, Yang et al. 2003) and abundance matching (Kravtsov et al. 2004; Vale & Ostriker 2004, 2006; Conroy et al. 2006). Because of the computational efficiency and the transparency of the assumptions underlying empirical models, these are the most widely used approaches to incorporating galaxy formation physics into contemporary cosmological likelihood analyses (e.g., Cacciato et al. 2013; Reid et al. 2014). Galaxy-halo models also play a central role in uncovering observational trends that are now considered fundamental to extragalactic astronomy, such as the stellar mass-to-halo mass relation (Tinker et al. 2005; van den Bosch et al. 2007; Behroozi et al. 2010; Moster et al. 2010; Leauthaud et al. 2012) and the demographics of galaxy quenching (van den Bosch et al. 2003; Collister & Lahav 2005; Behroozi et al. 2013a; Tinker et al. 2013).
Building upon non-standard formulations of the galaxy-halo connection, recent advances in this field have seen an increase in the complexity of empirical models. For example, in the abundance matching formulation constrained in Lehmann et al. (2015), stellar mass is influenced by both present-day halo mass as well as halo concentration. Satellite galaxies in the “delayed-then-rapid” model introduced in Wetzel et al. (2012) have an explicit dependence on the time the satellites first passed within the virial radius of their host halo. In the age matching model (Hearin & Watson 2013), the star formation history of both centrals and satellites is tightly coupled to halo mass assembly history across cosmic time. These advances reflect a growing trend in galaxy-halo modeling that we only expect to continue: galaxies co-evolve together with their parent halos. While this improves the sophistication of the predictions that can be extracted from empirical models, it also creates additional technical challenges and hurdles for their implementation, especially for beginners.

The standardization of traditional models such as the HOD, CLF, and abundance matching provides a natural motivation for a correspondingly standard code base with which the observable predictions of these models can be generated. The trend of galaxy-halo models toward increasing complexity highlights the need for such a code base, particularly one that is open-source and transparent to modify and extend.

Alongside these scientific developments, over the past several years the Python programming language has become the most popular language in the field of astronomy (Momcheva & Tollerud 2015). Many aspects of Python make this language well-suited for a fast-moving scientific field: dynamic typing and automated memory management facilitate rapid development; stable, high-performance libraries such as NumPy (Van Der Walt et al. 2011) and SciPy (Jones et al. 2001–2016) relieve scientists from the need to implement common numerical routines in scientific computing; the Python community has developed a broad toolkit for transparently documenting and testing code in an automated fashion; Python users find its uncluttered syntax to be highly readable and expressive. A significant additional factor leading to the widespread use of Python in astronomy has been the development of Astropy (Astropy Collaboration et al. 2013), a project with the express goal of providing a single core package for astronomy in Python, as well as fostering interoperability between complementary Python astronomy packages.

Motivated by these trends, we present the first stable release of Halotools (v0.2), an open-source, object-oriented Python package for building and testing models of the galaxy-halo connection. Halotools is community-driven and already includes contributions from over a dozen scientists spread across numerous universities. Designed with high-speed performance in mind, the package generates mock observations of synthetic galaxy populations with sufficient speed to conduct expansive Monte Carlo Markov Chain (MCMC) likelihood analyses over a diverse and highly customizable set of models.

Halotools can be thought of in analogy to Boltzmann-solvers such as CMBFast (Seljak & Zaldarriaga 1996), CAMB (Lewis et al. 2000), and CLASS (Essinger-Hileman et al. 2014). However, rather than generating ΛCDM predictions for Cosmic Microwave Background power spectra, Halotools uses a halo catalog output of a cosmological simulation to predict large-scale structure observables such as galaxy clustering, galaxy–galaxy lensing, void abundance, redshift-space distortions, and related statistics. These predictions are forward modeled from a mock galaxy population: Halotools makes no appeal whatsoever to simulation fitting functions (e.g., Sheth et al. 2001; Tinker et al. 2005) that are commonly employed in conventional formulations of the galaxy-halo connection. As these and related fitting functions have well-established upper limits on their accuracy (e.g., Tinker et al. 2008), our forward modeling approach makes Halotools better suited to the needs of the precision cosmology program, limiting systematic uncertainty to the accuracy and cosmic variance of the underlying simulation.

Halotools is an affiliated package of Astropy. Affiliated packages are not part of the Astropy core package, but extend the core with typically more domain-specific functionality while maintaining its engineering and interface standards. Since its inception, Halotools has been written in fully public view on https://github.com/astropy/halotools, in the spirit of open science.

This paper is not intended to be viewed as code documentation, which is available online at http://halotools.readthedocs.io. Our aim here is to give a simple overview of the features of the package and to serve as a standard reference for scientists using Halotools to support their published work. We outline the basic structure of the code in Section 2, describe the development workflow in Section 3, and conclude in Section 4.

2. Package Overview

Halotools is composed almost entirely in Python with syntax that is compatible with both 2.7 and 3.x versions of the language. Bounds-checking, exception-handling, and high-level control flow are always written in pure Python. Whenever possible, performance-critical functions are parallelized using Python’s native multiprocessing module. Halotools relies heavily on vectorized functions in NumPy as a core optimization strategy. However, in many cases there is simply no memory efficient way to vectorize a calculation, and it becomes necessary to write explicit loops over large numbers of points. In such situations, care is taken to pinpoint the specific part of the calculation that is the bottleneck; that section, and that section only, is written in Cython.

Halotools is designed with a high degree of modularity, so that users can pick and choose the features that are suitable to their science applications. At the highest level, this modularity is reflected in the organization of the package into sub-packages. For Halotools v0.2, there are three major sub-packages. The sim_manager sub-package described in Section 2.1 is responsible for reducing “raw” halo catalogs into efficiently organized fast-loading hdf5 files and for creating and keeping track of a persistent memory of where the simulation data is stored on disk. The empirical_models sub-package

21 [21] Halotools is licensed under a 3-clause BSD style license—see the LICENSE.rst file. Release v0.2 is archived with (Hearin et al. 2017a). While revising this paper for publication, code development has progressed such that the latest release is v0.5, archived with (Hearin et al. 2017b).

22 Halotools is an affiliated package of Astropy.

23 Cython is a tool that compiles Python-like code into C code. See http://cython.org and Behnel et al. (2011) for further information.
described in Section 2.2 contains models of the galaxy-halo connection, as well as a flexible object-oriented platform for users to design their own models. The `mock_observables` sub-package described in Section 2.3 contains a collection of functions that generate predictions for models in a manner that can be directly compared to astronomical observations. Many of the functions in `mock_observables` should also be of general use in the analysis of halo catalogs.

Although these sub-packages are designed to work together, each individual sub-package has entirely stand-alone functionality that is intended to be useful even in the absence of the others. For example, while Halotools provides pre-processed halo catalogs that are science-ready as soon as they are downloaded, use of Halotools-provided catalogs is entirely optional and the package works equally well with alternative halo catalogs provided and processed by the user. The `empirical_models` sub-package can be used to populate mock galaxies into the halos of any cosmological simulation, where the populated halos could be identified by any algorithm. The functions in the `mock_observables` sub-package accept simple point-data as inputs and so these functions could be used to generate observational predictions for semi-analytical models that otherwise have no connection to Halotools. We outline each of these sub-packages in the sections below.

### 2.1. Managing Simulation Data

The end result of any algorithm for identifying dark matter halos in an N-body simulation is a catalog of tabular data storing the positions and properties of the halos (e.g., Klypin & Holtzman 1997, BDM; Springel et al. 2001, SUBFIND; Behroozi et al. 2013b, ROCKSTAR). One of the most tedious tasks in simulation analysis is the initial process of getting started with a halo catalog: reading large data files storing the halos (typically ASCII-formatted), making cuts on halos, adding additional columns, and storing the reduced and value-added catalog on disk for later use. In our experience with simulation analysis, one of the most common sources of errors comes from these initial bookkeeping exercises.

To help beginning users get started, the Halotools developers manage a collection of pre-processed halo catalogs that are available for download either with the `DownloadManager` class, or equivalently with the command-line script `download_additional_halocat`. Through either download method, the catalogs are automatically cached and science-ready as soon as the download completes. Halotools currently offers pre-processed halo catalogs for Rockstar-identified halos from four different simulations: bolshoi, bolshoi-planck, multi-dark, and consuelo, for which snapshots at \( z = 0, 0.5, 1 \), and 2 are available (Gottloeber & Klypin 2008; Klypin et al. 2011; McBride et al. 2011; Riebe et al. 2011). A random down-sampling of dark matter particles is also available to accompany each supported snapshot. The `sim_manager` sub-package structure also supports halo catalogs in light code format, although all pre-processed catalogs are single-snaphot data.

In addition to these pre-processed catalogs, the `sim_manager` sub-package offers tools to process and format other halo catalogs so that users can get started with full-fledged halo catalog analysis from just a few lines of code. The `RockstarHListReader` class allows users to quickly create a Halotools-formatted catalog starting from the typical ASCII output of the Rockstar halo-finder (Behroozi et al. 2013b, 2013c). Users wishing to work with catalogs of halos identified by algorithms other than Rockstar can use the `TabularAsciiReader` class to initially process their ASCII data. Both readers are built around a convenient API that uses Python’s native “lazy evaluation” functionality to select, on-the-fly, only those columns and rows that are of interest, making these readers highly memory efficient.

All Halotools-formatted catalogs are Python objects storing the halo catalog itself in the form of an Astropy Table and also storing some metadata about the simulated halos. In order to build an instance of a Halotools-formatted catalog, a large collection of self-consistency checks about the halo data and metadata are performed, and an exception is raised if any inconsistency is detected. These checks are automatically carried out at the initial processing stage and also every time the catalog is loaded into memory in order to help ensure that the catalog is processed correctly and does not become corrupt over time.

The `sim_manager` sub-package allows users to cache their preprocessed halo catalogs when they are saved to disk, creating the option to load their catalogs into memory with the simple and intuitive syntax shown in Figure 1. Cached simulations are stored in the form of an hdf5 file (The HDF Group 1997–2016). This binary file format is fast-loading and permits metadata to be bound directly to the file in a transparent manner, so that the cached binary file is a self-expressive object. The `HaloTableCache` class provides an object-oriented interface for managing the cache of simulations, but users are also free to work directly with the cache log, which is a simple, human-readable text file located in the Halotools cache directory.

Using the Halotools caching system is optional in every respect. Users who prefer their own system for managing simulated data are free to do so in whatever manner they wish; they need only pass the necessary halo data and metadata to the `UserSuppliedHaloCatalog` class, and the full functionality of all sub-packages of Halotools works with the resulting object instance.

### 2.2. Empirical Models

All Halotools models of the galaxy-halo connection are contained in the `empirical_models` sub-package. Halotools models come in two categories: composite models and component models. A composite model is a complete description of the mapping(s) between dark matter halos and all properties of their resident galaxy population. A composite model provides sufficient information to populate an ensemble of halos with a Monte Carlo realization of a galaxy population. All composite models are built from a collection of independently defined component models. A component model provides a map between dark matter halos and a single property of the resident galaxy population. Example component models include the stellar-to-halo mass relation, a Navarro--Frenk--White radial profile for the satellite distribution, or the halo mass-dependence of the quenched fraction. As of v0.2, the `empirical_models`...
sub-package only contains models mapping snapshots of dark matter halos to galaxy populations at a fixed redshift. We elaborate upon the structure of Halotools models below.

2.2.1. Model Styles

Halotools composite models come in two different types: HOD-style models and subhalo-based models. In HOD-style models, there is no relationship between the abundance of satellite galaxies in a host halo and the number of subhalos in that host halo. In these models, satellite abundance in each halo is determined by a Monte Carlo realization of some analytical model. Examples of this approach to the galaxy-halo connection include the HOD (Berlind & Weinberg 2002) and CLF (Yang et al. 2003), as well as extensions of these that include additional features such as color-dependence (Tinker et al. 2013).

By contrast, in subhalo-based models, there is a one-to-one correspondence between subhalos and satellite galaxies. In these models, each host halo in the simulation is connected to a single central galaxy, and each subhalo is connected to a single satellite. Examples include traditional abundance matching (Kravtsov et al. 2004; Conroy et al. 2006), age matching (Hearin et al. 2014), and parameterized stellar-to-halo mass models (Behroozi et al. 2010; Moster et al. 2010).

2.2.2. Pre-built Models

Halotools ships with a handful of fully formed pre-built composite models, each of which has been designed around a model chosen from the literature. All pre-built models can directly populate a simulation with a mock catalog. Users need only choose the pre-built model and simulation snapshot that are appropriate for their science application and can then immediately generate a Monte Carlo realization of the model. The syntax in Figure 2 shows how to populate the Bolshoi simulation at $z = 0$ with an HOD-style model based on Leauthaud et al. (2011). All Halotools models can populate halo catalogs with mock galaxies using this same syntax, regardless of the features of the model or the selected halo catalog.

Calling the `populate_mock` method the first time creates the `mock` attribute of a model, and triggers a large amount of pre-processing that need only be done once. The algorithm used to repopulate mock catalogs takes advantage of this pre-processing, so that subsequent calls to `model.mock.populate()` are far faster. For example, repopulation of the model shown in Figure 2 based on the Leauthaud et al. (2011) model takes just a few hundred milliseconds on a modern laptop.

The behavior of all Halotools models is controlled by the `model.param_dict` attribute, a Python dictionary storing the values of the model parameters. By changing the values of the parameters in the `param_dict`, users can generate alternate mock catalogs based on the updated parameter values. The process of varying `param_dict` values and repeatedly populating mock catalogs is the typical workflow in an MCMC-type analysis conducted with Halotools.

2.2.3. User-built Models

The `empirical_models` sub-package provides far more functionality than a simple set of pre-built composite models that are ready to generate mock catalogs “out of the box.” Halotools has special factory classes that allow users to build their own models connecting galaxies to the dark matter halos that host them. These factories are the foundation of the object-oriented platform that Halotools users can exploit to design

```python
>>> from halotools.sim_manager import CachedHaloCatalog
>>> halocat = CachedHaloCatalog(simname='bolshoi', redshift=0.5)
View the first ten halos in the catalog
>>> print(halocat.halo_table[0:10])

Inspect some of the halo catalog metadata
>>> print(halocat.Lbox, halocat.particle_mass)
```

Figure 1. Loading a cached halo catalog into memory.

```python
>>> from halotools.sim_manager import CachedHaloCatalog
>>> halocat = CachedHaloCatalog(simname='bolshoi', redshift=0)

>>> from halotools.empirical_models import PrebuiltHodModelFactory
>>> model = PrebuiltHodModelFactory('leauthaud11')

>>> model.populate_mock(halocat)

View the first ten galaxies in the catalog
>>> print(model.mock.galaxy_table[0:10])
```

Figure 2. Populating a mock galaxy catalog.
their own models of the galaxy-halo connection. This model-building platform is the centerpiece of the empirical_models sub-package.

Users choose between a set of component models of the galaxy population and compose them together into a composite model using the appropriate Halotools factory class; HOD-style models are built by the HODModelFactory class, subhalo-based models are built by the SubhaloModelFactory. Composing different collections of components together gives users a large amount of flexibility to construct highly complex models of galaxy evolution. There are no limits on the number of component models that can be chosen, nor on the number or kind of galaxy population(s) that make up the universe in the user-defined composite model. Figure 3 shows a cartoon example of the factory design of an HOD-style model.

In choosing component models, users are not restricted to the set of features that ship with the Halotools package. Users are free to write their own component models and use the Halotools factories to build the composite, to write just one new component model and include it in a collection of Halotools-provided components, or anywhere in between. This way, users mostly interested in a specific feature of the galaxy population can focus exclusively on developing code for that one feature and use existing Halotools components to model the remaining features. The factory design pattern also makes it simple to swap out individual features while keeping all other model aspects identical, facilitating users to ask targeted science questions about galaxy evolution and answer these questions via direct computation.

2.3. Mock Observations

In the analysis of halo and (mock) galaxy catalogs, many of the same calculations are performed over and over again.

1. How many pairs of points are separated by some distance $r$?
2. What is the two-point correlation function of some sample of points?
3. What are the host halo masses of some sample of subhalos?
4. What is the local environmental density of some collection of galaxies?

It is common to calculate the answers to these and other similar questions in an MCMC-type analysis, when high-performance is paramount. Even outside of the context of likelihood analyses, the sheer size of present-day cosmological simulations presents a formidable computational challenge to evaluate such functions in a reasonable runtime. There is also the notorious complicating nuisance of properly accounting for the periodic boundary conditions of a simulation. Much research time has been spent by many different researchers writing their own private versions of these calculations, writing code that is not extensible, as it was developed making hard assumptions that are only applicable to the immediate problem at hand.

The mock_observables sub-package is designed to remedy this situation. This sub-package contains a large collection of functions that are commonly encountered when analyzing halo and galaxy catalogs, including:

1. the many variations of two-point correlation functions, (a) three-dimensional correlation function $\xi(r)$,
2. marked correlation functions $M(r)$,
3. friends-of-friends group identification,
4. group aggregation calculations, e.g., calculating the total stellar mass of galaxies of a common group $M_{\text{star}}$,
5. isolation criteria, e.g., identifying those galaxies without a more massive companion inside some search radius,
6. pairwise velocity statistics, e.g., the line-of-sight velocity dispersion as a function of projected distance $\sigma_{\text{los}}(r_p)$,
7. void probability function $P_{\text{void}}(r)$,
8. etc.

The mock_observables sub-package contains heavily optimized implementations of all of the above functions, as well as a variety of others. Figure 4 provides a visual demonstration of the diversity of the available options.

Every function in mock_observables has a stable, user-friendly API that is consistently applied across the package. The documentation of all functions contains an explicit example of how to call the function, and in many cases there is a step-by-step tutorial in the documentation showing how the function might be used in a typical analysis. Considerable effort has been taken to write mock_observables to be modular, so that users can easily borrow the algorithm patterns to write their own variation on the provided calculations.

3. Package Development

3.1. GitHub Workflow

Halotools has been developed fully in the open since the inception of the project. Version control for the code base is managed using git.28 The public version of the code is hosted on GitHub.29 The latest stable version of the code can be

28 http://git-scm.com
29 http://www.github.com
Figure 4. Six example calculations done with Halotools demonstrating the diversity of the mock_observables sub-package. Top left: three-dimensional correlation function of mock galaxies $\xi_{gg}(r)$ split into contributions from pairs of galaxies occupying a common halo (1-halo term) and pairs in distinct halos (2-halo term). Top right: projected correlation function $w_p(r_p)$ of star-forming and quiescent galaxies, as well as their cross-correlation. Center left: galaxy lensing signal produced by galaxy clusters and galaxy groups. Center right: fraction of galaxies that are considered “isolated” according to whether there is a more massive companion within a cylinder of projected radius 500 kpc and a velocity difference of 500 km s$^{-1}$. Bottom left: mean pairwise radial velocity of galaxies in the neighborhood of a cluster BCG. Bottom right: mean mass accretion rate of lower-mass subhalos as a function of cluster-centric distance $r/R_{\text{vir}}$.
Pull Request is then tested and reviewed by the Halotools developers before it is eligible to be merged into master.

3.2. Automated Testing

Halotools includes hundreds of unit tests that are incorporated into the package via the py.test framework. These tests are typically small blocks of code that test a specific feature of a specific function. The purpose of the testing framework is both to verify scientific correctness and also to enforce that the API of the package remains stable. We also use continuous integration, a term referring to the automated process of running the entire test suite in a variety of different system configurations (e.g., with different releases of NumPy and Astropy installed or different versions of the Python language). Each time any Pull Request is submitted to the master branch of the code on GitHub, the proposed new version of the code is automatically copied to a variety of virtual environments and the entire test suite is run repeatedly in each environment configuration. The Pull Request will not be merged into the master unless the entire test suite passes in all environment configurations. We use Travis for continuous integration in Unix environments such as Linux and Mac OS X, and AppVeyor for Windows environments.

Pull Requests to the master branch are additionally subject to a requirement enforced by Coveralls. This service performs a static analysis on the Halotools code base and determines the portions of the code that are covered by the test suite, making it straightforward to identify logical branches whose behavior remains to be tested. Coveralls issues a report for the fraction of the code base that is covered by the test suite; if the returned value of this fraction is smaller than the coverage fraction of the current version of master, the Pull Request is not accepted. This ensures that test coverage can only improve as the code evolves and new features are added.

Any time a bug is found in the code, either by Halotools developers or users, a GitHub Issue is opened calling public attention to the problem. When the Halotools developers have resolved the problem, a corresponding regression test becomes part of the code base. The regression test explicitly demonstrates the specific source of the problem and contains a hyperlink to the corresponding GitHub issue. The test will fail when executed from the version of the code that had the problem and will pass in the version with the fix. Regression testing helps to make it transparent how the bug was resolved and protects against the same bug from creeping back into the repository as the code evolves.

3.3. Documentation

Documentation of the code base is generated with Sphinx and is hosted on Read the Docs (RTD) at http://halotools.readthedocs.io. The public repository https://github.com/astropy/halotools has a webhook set up so that whenever there is a change to the master branch, the documentation is automatically rebuilt to reflect the most up-to-date version of master.

Every user-facing class, method, and function in Halotools has a docstring describing its general purpose, its inputs and outputs, and also providing an explicit example usage. Each such example usage appearing in a docstring also serves as a doctest. A doctest is a code fragment appearing in the documentation that (1) demonstrates an example call to a function, and (2) executes as part of the test suite. Thus, if the API of a function changes but the documentation is not updated to reflect this change, this triggers a test failure that must be resolved prior to merging the modified code into the master. Doctests help ensure that as Halotools evolves, the code still behaves as the documentation says it does.

Docstrings for many functions with complex behavior come with a hyperlink to a separate section of the documentation in which mathematical derivations and algorithm notes are provided. The documentation also includes a large number of step-by-step tutorials and example analyses. The goal of these tutorials is more than simple code demonstration: the tutorials are intended to be a pedagogical tool illustrating how to analyze simulations and study models of the galaxy-halo connection in an efficient and reproducible manner.

4. Conclusion

We have presented the first stable release of the Halotools package (v0.2), a specialized Python package for building and testing models of the galaxy-halo connection and analyzing catalogs of dark matter halos. The core functionality of the package includes:

1. fast generation of synthetic galaxy populations using HODs, abundance matching, and related methods.
2. efficient algorithms for calculating galaxy clustering, lensing, z-space distortions, and other astronomical statistics.
3. a modular, object-oriented framework for designing galaxy evolution models.
4. end-to-end support for reducing halo catalogs and caching them as fast-loading hdf5 files.

We offer several examples below of typical use-cases for which Halotools was designed. This list is not intended to be complete, but simply to illustrate the full-featured nature of the package, which includes functionality supporting both galaxy evolution science as well as cosmological simulation analysis. The documentation hosted at https://halotools.readthedocs.io contains extensive tutorials and step-by-step guides that can be used as examples for how to carry out each of the example studies below and more.

1. Constrain traditional HOD model parameters with an MCMC-type analysis of observational measurements of projected galaxy clustering \( w_p(r_p) \).
2. Build a novel empirical model of galaxy morphology and derive MCMC-type constraints on its parameters using non-traditional large-scale structure measurements such as the morphology-marked correlation function \( M(r_p) \).
3. Conduct a study of the connection between the radial profiles of dark matter subhalos and the mass accretion history of their parent halo.

\[ \text{In fact, there are two versions of documentation available, latest and stable, and one can choose on RTD which one to read.} \]
4. Build mock catalogs of galaxies that include forward-models of observational systematics such as intrinsic alignments and/or deblending errors in galaxy shapes.

We would like to conclude this paper with an invitation. As can be seen from the public record of the package development on GitHub, contributions of all kinds are warmly welcomed. This could include submitting a Pull Request of a novel feature that has been developed for a scientific publication, or simply submitting a bug report or requesting documentation clarification. It is the hope of the development team that scientists who find the package useful in their own work will also use Halotools as an outlet to share their expertise in large-scale structure in a way that can benefit the wider astronomical community.

A.P.H. would like to express profound gratitude to the Yale Center for Astronomy and Astrophysics, and Meg Urry in particular, for their carte blanche support for this long-term and extremely labor-intensive project. Without the creative freedom offered by the YCAA Prize fellowship, there would be no Halotools.

Work at Argonne National Laboratory was supported under U.S. Department of Energy contract DE-AC02-06CH11357. A.P.H. gives special thanks to Jim Ford for every last groove of Harlan County. A.P.H. is grateful to Nikhil Padmanabhan and Frank van den Bosch for invaluable and continuing guidance. Thanks to Doug Watson and Matt Becker for productive discussions at a formative stage of Halotools development, and to Yu Feng for help in optimizing the sub-package and comments on an early draft of this paper. Thanks to Andrey Kravtsov for help with establishing compatibility in Windows environments.

D.C. would like to thank Nikhil Padmanabhan and Frank van den Bosch for their patience and support while developing this package.

We thank the Astropy developers for the package-template, which has been critical to the entire process of Halotools development. We would like to thank the NumPy, SciPy, IPython, Matplotlib, GitHub, Read the Docs, Travis, AppVeyor, and Coveralls communities for their extremely useful free software products. We thank all Halotools users who have patiently helped improve the package by providing feedback and critiques from the project’s alpha-stages to the present.

A portion of this work was also supported by the National Science Foundation under grant PHYS-1066293 and the hospitality of the Aspen Center for Physics. This work was also partially funded by the U.S. National Science Foundation under grant AST 1517563. Support for E.J.T. was provided by NASA through Hubble Fellowship grant #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. P.B. was supported through program number HST-HF2-51353.001-A, provided by NASA through a Hubble Fellowship grant from STScI, which is operated by the Association of Universities for Research in Astronomy, Incorporated, under NASA contract NAS5-26555. N.J.G. is funded by NSF grant ACI-1535651 as well as by the Gordon and Betty Moore Foundation’s Data-Driven Discovery Initiative through Grant GBMF4651. E.J. is supported by Fermi Research Alliance, LLC under the U.S. Department of Energy under contract No. DEAC02-07CH11359. S.M. is supported by Grant-in-Aid for Scientific Research from the JSPS Promotion of Science (15K17600, 16H01089).

ORCID iDs
Andrew P. Hearin @ https://orcid.org/0000-0003-2219-6852
Erik Tollerud @ https://orcid.org/0000-0002-9599-310X
Benedikt Diemer @ https://orcid.org/0000-0001-9568-7287
Nathan J. Goldbaum @ https://orcid.org/0000-0001-5557-267X
Alexie Leauthaud @ https://orcid.org/0000-0002-3677-3617
Yao-Yuan Mao @ https://orcid.org/0000-0002-1200-0820
Surhud More @ https://orcid.org/0000-0002-2986-2371
Manodeep Sinha @ https://orcid.org/0000-0002-4845-1228

References
Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33
Behnel, S., Bradshaw, R., Citro, C., et al. 2011, CSE, 13, 31
Behroozi, P. S., Conroy, C., & Wechsler, R. H. 2010, ApJ, 717, 379
Behroozi, P. S., Wechsler, R. H., & Conroy, C. 2013a, ApJL, 762, L31
Behroozi, P. S., Wechsler, R. H., & Wu, H.-Y. 2013b, ApJ, 762, 109
Behroozi, P. S., Wechsler, R. H., Wu, H.-Y., et al. 2013c, ApJ, 763, 18
Berlind, A. A., & Weinberg, D. H. 2002, ApJ, 575, 587
Bhattacharya, S., Habib, S., Heitmann, K., & Vikhlinin, A. 2013, ApJ, 766, 32
Bhattacharya, S., Heitmann, K., White, M., et al. 2011, ApJ, 732, 122
Cacciato, M., van den Bosch, F. C., More, S., Mo, H., & Yang, X. 2013, ApJ, 740, 576
Collister, A. A., & Lahav, O. 2005, MNRAS, 361, 415
Conroy, C., Wechsler, R. H., & Kravtsov, A. V. 2006, ApJ, 647, 201
Cooray, A., & Sheth, R. 2002, PhR, 372, 1
Diemer, B., &Kravtsov, A. V. 2015, ApJ, 799, 108
Essinger-Hileman, T., Ali, A., Amiri, M., et al. 2014, Proc. SPIE, 9153, 915311
Gottloeber, S., & Klypin, A. 2008, arXiv:0803.4343
Hearin, A., Tollerud, E., Robitaille, T., et al. 2017a, astropy/halotools, v0.2, Zenodo, doi:10.5281/zenodo.835898
Hearin, A., Tollerud, E., Robitaille, T., et al. 2017b, astropy/halotools, v0.5, Zenodo, doi:10.5281/zenodo.835959
Hearin, A. P., & Watson, D. F. 2013, MNRAS, 435, 1313
Hearin, A. P., Watson, D. F., Becker, M. R., et al. 2014, MNRAS, 444, 729
Heitmann, K., Bingham, D., Lawrence, E., et al. 2016, ApJ, 820, 108
Jones, E., Oliphant, T., Peterson, P., et al. 2001–2016, http://www.scipy.org
Klypin, A., & Holtzman, J. 1997, arXiv:astro-ph/9712217
Klypin, A. A., Trujillo-Gomez, S., & Primack, J. 2011, ApJ, 740, 102
Kravtsov, A. V., Berlind, A. A., Wechsler, R. H., et al. 2004, ApJ, 609, 35
Leauthaud, A., Tinker, J., Bhrooozi, P. S., Busha, M. T., & Wechsler, R. H. 2011, ApJ, 738, 45
Leauthaud, A., Tinker, J., Bundy, K., et al. 2012, ApJ, 744, 159
Lehmann, B. V., Mao, Y.-Y., Becker, M. R., Skillman, S. W., & Wechsler, R. H. 2015, arXiv:1510.05651
Lewis, A., Challinor, A., & Lasenby, A. 2000, ApJ, 538, 473
Ma, C.-P., & Fry, J. N. 2000, ApJ, 543, 503
McBride, C., Berlind, A. A., Scoccimarro, R., et al. 2011, BAAS, 43, 249.07
Mo, H., van den Bosch, F. C., & White, S. 2010, Galaxy Formation and Evolution (Cambridge: Cambridge Univ. Press)
Momcheva, I., & Tollerud, E. 2015, arXiv:1507.03989
Moster, B. P., Somerville, R. S., Maullbetsch, C., et al. 2010, ApJ, 710, 903
Reid, B. A., Seo, H.-J., Leauthaud, A., Tinker, J. L., & White, M. 2014, MNRAS, 444, 476
Riebe, K., Parli, A. M., Enke, H., et al. 2011, arXiv:1109.0003
Seljak, U. 2000, MNRAS, 318, 203
Seljak, U., & Zaldarriaga, M. 1996, ApJ, 469, 437
Sheth, R. K., Mo, H. J., & Tormen, G. 2001, MNRAS, 323, 1
Springel, V., White, S. D. M., Tormen, G., & Kauffmann, G. 2001, MNRAS, 328, 726
The HDF Group 1997–2016, http://www.hdfgroup.org/HDF5
Tinker, J., Kravtsov, A. V., Klypin, A., et al. 2008, ApJ, 688, 709
Tinker, J. L., Leauthaud, A., Bundy, K., et al. 2013, ApJ, 778, 93
Tinker, J. L., Robertson, B. E., Kravtsov, A. V., et al. 2010, ApJ, 724, 878
Tinker, J. L., Weinberg, D. H., Zheng, Z., & Zehavi, I. 2005, ApJ, 631, 41
Vale, A., & Ostriker, J. P. 2004, MNRAS, 353, 189
Vale, A., & Ostriker, J. P. 2006, MNRAS, 371, 1173
van den Bosch, F. C., Yang, X., & Mo, H. J. 2003, MNRAS, 340, 771
van den Bosch, F. C., Yang, X., Mo, H. J., et al. 2007, MNRAS, 376, 841
Van Der Walt, S., Colbert, S. C., & Varoquaux, G. 2011, arXiv:1102.1523
Wetzel, A. R., Tinker, J. L., & Conroy, C. 2012, MNRAS, 424, 232
White, S. D. M., & Rees, M. J. 1978, MNRAS, 183, 341
Yang, X., Mo, H. J., & van den Bosch, F. C. 2003, MNRAS, 339, 1057