Void Analysis User Documentation

Stephen Stopyra

June 27, 2025

Abstract

Documentation for the void analysis pipeline.

Contact: Stephen Stopyra (svstopyra@googlemail.com)

1 Introduction

This is the user documentation for the `void_analysis` pipeline, a collection of scripts and pipelines designed for computing anti-halo void catalogues from pairs of simulations, and analysing the output.

Anti-halos are a definition of cosmological voids that places voids and halos on equal footing: instead of using the morphology of the cosmic web (as is done with void finders such as `VIDE`, Sutter et al. (2015)), in the anti-halo approach voids are modelled as clusters in an "anti-universe" reverse simulation, obtained from the original simulation by reversing the sign of the density contrast in the initial conditions. This transforms under-dense regions into over-dense regions and vice versa. After running the reversed initial conditions to redshift $z = 0$, a halo finder can be run to identify clusters, and the dark matter particles corresponding to a halo in the reverse simulation correspond to voids in the original, forward simulation. The method was first outlined by Pontzen et al. (2016).

The purpose of this code is to analyse forward/reverse simulation pairs, typically from initial conditions inferred by `BORG` (Jasche & Lavaux (2019)), and produce an anti-halo void catalogue from these. Since the initial conditions of our Universe can only be known approximately, in the Bayesian framework of `BORG` it is necessary to draw many samples from the posterior distribution of possible initial conditions. Therefore, generating an anti-halo catalogue of the present day Universe requires combining multiple possible halo catalogues into a single catalogue. This is the purpose of the void analysis code presented here, which was used to produce the catalogue in Stopyra et al. (2024).

1.1 Useful links

- Anti-halo void catalogue of the local super-volume, at Zenodo: `https://zenodo.org/records/10160612`
- Antihalo analysis github repository: `https://github.com/svstopyra/void_analysis`
- Aquila Cloud link for the anti-halo catalogue simulations: `https://cloud.aquila-consortium.org/s/bE4dKyJq6wZJ93D`
- Void Catalogue paper (Stopyra et al. (2024)): `https://doi.org/10.1093/mnras/stae1251`

The main body of the code can be found at the gihub repository above. To reproduce the void catalogues presented in Stopyra et al. (2024), however, the simulation data will be needed. The actual void catalogue has already been released on Zenodo, but the underlying simulations are available via the Aquila Cloud (see link above). For problems with accessing the data via the Aquila Cloud link, please contact the Guilhem Lavaux (guilhem.lavaux@iap.fr), but please direct any questions about the content of the files or code to Stephen Stopyra (svstopyra@googlemail.com).

While the void catalogue construction is the main focus of the code, aspects of this package were also used in the following papers:

- "How to build a catalogue of linearly evolving cosmic voids", Stopyra et al. (2021a)
- "Quantifying the rarity of the local super-volume" Stopyra et al. (2021b)
- "Towards accurate field-level inference of massive cosmic structures" Stopyra et al. (2023)

1.2 License

The code is released under the GNU public license v3, but if you make use of the code for scientific purposes, please cite the void catalogue paper, Stopyra et al. (2024).

1.3 Acknowledgements

I would like to thank the many discussions with my collaborators, over the years in helping to make this work possible. In particular, my co-authors on the works that used this code-base, Hiranya Peiris, Andrew Pontzen, and Jens Jasche.

While conducting the research that led to the development of this code over many years, I was also generously supported by: the Göran Gustafsson Foundation for Research in Natural Sciences and Medicine; the UCL Cosmoparticle Initiative, the European Union's Horizon 2020 research and innovation programme under grant agreement No. 818085 GMGalaxies; the Royal Society under research grants CEC19\100037, RGF\EA\181010, and URF\R\180003.

Much of the technical development and testing was carried out using computing resources provided by: the Research Capital Investment Fund (RCIF) provided by United Kingdom Research and Innovation (UKRI); the Sunrise HPC facility supported by the Technical Division at the Department of Physics, Stockholm University; and the Swedish National Infrastructure for Computing (SNIC) at Linköping University, partially funded by the Swedish Research Council through grant agreement no. 2018-05973. Some of the results of this code make use of the healpy and HEALPix package[1].

2 Installation

The package is not yet available via `pip`, so it is necessary to clone the git repository at `https://github.com/svstopyra/void_analysis` onto a location in your python path. For example:

```
git clone https://github.com/svstopyra/void_analysis.git
```

which can be for example cloned into the `<PATH_TO_PYTHON_LIBRARIES>/site_packages/` directory. If installing in a virtual environment, the appropriate virtual environment directory for `site_packages` should be used.

Once installed, tests can be run by running `pytest` in the `void_analysis` directory.

2.1 Dependencies

The code has been tested with python 3.10.12. Older versions may also work. It has also been tested with the most recent version of python at the time of writing (3.13.5), and everything except the mayavi visualisation code is functional. Mayavi unfortunately requires older versions of python at the time of writing (3.10 recommended). A requirements.txt file is included in the documentation subfolder, which allows for easier setup.

The following packages are required, depending on what parts of the code you wish to use.

Essential packages:

- `numpy`: (see Harris et al. (2020)) currently not supporting `numpy 2+`. Tested with version 1.26.4.
- `pynbody`: (see Pontzen et al. (2013)) for loading cosmological simulation data and halo catalogues. Install via `pip` (preferred) or clone from `https://github.com/pynbody/pynbody`.
- `scipy`: (see Virtanen et al. (2020)) mostly used for statistical and special functions, as well as bootstrapping.
- `astropy`: (see Astropy Collaboration et al. (2013, 2018, 2022)) For cosmological calculations.
- `matplotlib`: (see Hunter (2007)) For all plotting routines.
- `Corrfunc`: (Sinha & Garrison (2020))For computing correlation functions, and also used by the void stacking code for accelerated stacking.

Highly recommended (most calculations will work, but some plotting functions will be unavailable):

- `alphashape`: (see Bellock et al. (2021)) For void outlines on sky-plots.
- `seaborn`: (see Waskom (2021)) For some plotting routines, and colormaps.
- `healpy`: (see Zonca et al. (2019); Górski et al. (2005)) For producing skyplots that use `HEALPix`.
- `hmf`: (see Murray et al. (2013)) For computing theoretical halo mass functions.
- `camb`: (see Lewis & Challinor (2011))For power spectrum calculations.
- `emcee`: (see Foreman-Mackey et al. (2013)) For cosmological inference, particularly in `cosmology_inference.py`

[1]`http://healpix.sourceforge.net`

- pytest: (see Krekel et al. (2004)) For running the automated test suite.
- pytest-cov: For running the automated test suite (provides coverage statistics).
- numexpr: (see Cooke et al. (2018)) For fast evaluation of some quantities, such as void barycentres.

Optional packages (some functionality will be disabled, but core routines are not effected):

- sympy: (see Meurer et al. (2017)) For symbolic calculations, used mainly in lpt_symbolic.py.
- pillow: (see Clark (2015)) For producing GIF animation plots.
- mayavi: (see Ramachandran & Varoquaux (2011)) Only if you wish to use the 3D visualisations provided in mayavi_plot.py. This package can be quite fiddly to install, which is why the functions using it are separated into mayavi_plot.py. In particular, mayavi doesn't work on more recent versions of python.
- h5py: (see Collette et al. (2022))For loading some file types.
- nbodykit: (see Hand et al. (2018)) Only for computing power spectra, if using the get_sim_ps.py script.
- borg: (see Jasche & Lavaux (2019)) Only used by generate_genetIC_ics.py. Can be tricky to get working correctly.

3 Overview of the Code

Examples of how to use the code are found in voids_paper_plots.py, which shows how to build a catalogue from scratch, and regenerate all the figures in Stopyra et al. (2024). The routines of the codebase are split across several files, which we describe now:

- antihalos.py: Routines for computing properties of anti-halo voids, such as void radii, masses, centres, and for comparing with other void definitions by examining volume overlaps.
- catalogue.py: This file contains the main class, combinedCatalogue, used for construction of an anti-halo void catalogue. See voids_paper_plots.py for usage examples.
- context.py: Contains functions relating to co-ordinates and void centres, including important functions such as the computation of the volume-weighted void barycentre with periodic boundary conditions.
- cosmology.py: Contains functions for computing cosmological quantities such as the linear growth factor, power spectra, and halo mass functions.
- cosmology_inference.py: Pipeline for inference of cosmological parameters via the Alcock-Paczynski test. Also contains functions for void velocity profile models, including up to 5th-order Lagrangian Perturbation Theory (LPT).
- halos.py: Functions for computing halo properties.
- lpt_symbolic.py: Symbolic code for computing and solving the equations of LPT up to 5th order (assuming spherical symmetry).
- massconstraintsplot.py: Code for generating the cluster mass constraints plot used in Stopyra et al. (2021b).
- mayavi_plot.py: Functions for generating 3D visualisations of voids and simulations, using mayavi.
- plot.py: Functions for generating various different plots that are useful for analysing void catalogues.
- plot_utilities.py: Utility functions for creating plots. Includes some simple binning functions.
- real_clusters.py: Functions for loading and processing data on galaxy clusters and catalogues.
- simulation_tools.py: Utility functions for loading, processing and analysing simulation data.
- snapedit.py: Utility functions for transforming and editing simulation snapshots. Includes functions for wrapping co-ordinates into the periodic box and changing co-ordinate systems for simulations, as well as generating Zel'dovich extrapolations of initial conditions (such as used for testing void linearity in Stopyra et al. (2021a)).
- stacking.py: Functions for stacking void density and velocities, both in 1D and 2D.
- survey.py: Utility functions for handling galaxy surveys, notably survey masks.

- `tools.py`: Generic utility functions useful for handling void catalogue data.
- `void_applications.py`: Examples for cosmological parameter inference with void catalogues.
- `voids_paper_plots.py`: Examples for generating figures in Stopyra et al. (2024).
- `example.py`: Worked example of how to use the catalogue code. See sec. 4.

There are additionally subfolders which contain useful scripts and utilities:

- `archived_scripts`: Old scripts, used for plots in Stopyra et al. (2021a,b, 2023), along with older code not seeing immediate use. Somewhat un-organised at the moment, but I have retained this folder as it may help with seeing some examples of the codebase in use.
- `cosmology_inference_tests`: Tests and test data for the codebase.
- `postprocessing`: Post-processing scripts used for analysing simulations. See sec. 3.1.
- `simulations`: Scripts for running simulations. See sec. 3.2.
- `test_snaps`: Low-resolution test snapshots, used for running automated tests on the void catalogue pipeline and other tests that require simulation data to run.

The remaining loose files include `pytest.ini` which is the configuration file for `pytest`, and

3.1 Postprocessing scripts

Found in the subdirectory `simulations`, these scripts are used when post-processing the resimulated pairs of forward and reversed initial conditions to extract void properties used by the analysis pipelines.

- `combine_cpu_catalogues.sh`: Shell script for combining multiple `AHF` halo catalogues into a single catalogue. This is used when running `AHF` with `MPI`, since each CPU generates its own halo catalogue.
- `get_sim_ps.py`: Script to estimate simulation power-spectra using `nbodykit`.
- `order_halo_catalogue.py`: Script to re-order an `AHF` halo catalogue(Knollmann & Knebe (2009)) in descending order of mass. Useful after combining the catalogues with `combine_cpu_catalogues.sh`, since this will produce an out-of-order catalogue.
- `process_snapshot.py`: Runs various analysis routines on a simulation-anti-simulation pair. See below for details.
- `run_post_processing.sh`: Shell script for running post-processing routines on simulations.
- `run_voz.py`: Runs Voronoi Tesselation code to obtain Voronoi volumes of particles in a simulation. Requires `VOBOZ/ZOBOV` (`https://ascl.net/1304.005`) to function. See Neyrinck et al. (2005).

The `process_snapshot.py` script performs the following analyses on a simulation/anti-simulation pair. The main body of this code is found in `simulation_tools.py` under the function `processSnapshot`:

- Compute halo and anti-halo centres in their respective simulations.
- Convert output of Voronoi tesselation backend (`run_voz.py`) to physical volumes. If Voronoi volumes are not available, substitutes them with estimated volumes using `pynbody`'s spherical particle hydrodynamics (SPH) density estimation code (Pontzen et al. (2013)).
- Compute void volumes and centres, by mapping anti-halo particles from the reverse simulation to the forward simulation, and adding up Voronoi volumes to compute volume and radius, as well as volume-weighted barycentres.
- Computed stacked 1D density profiles in 30 fixed bins between 0 and 3 effective radii.
- Computes void central and average densities.

The output is saved in `pickle` format to a file named `<snapshot_name>.AHproperties.p`. The order of arrays in this file is:

1. Halo centres in the forward simulation (clusters)
2. Halo masses in the forward simulation.

3. Anti-halo centres in the reverse simulation (N.B. – not the same as void centres!)
4. Anti-halo masses
5. Voronoi cell volumes for all particles in the simulation, in units of $h^{-3}\,\mathrm{Mpc}^3$.
6. X_{void}, Void centres (in the forward simulation) after mapping anti-halo particles to the forward simulation. Defined as a volume-weighted barycentre.
7. V_{void}, total void volumes in the forward simulation (sum of the Voronoi cell volumes for each particle).
8. R_{eff}, void effective radii (radius of a sphere with volume equal to that of the void).
9. Radius bin edges used for the 1D density profile.
10. N_{parts}, particle counts in each radius bin, for all voids. Used to compute the density profile.
11. V_{parts}, sum of the Voronoi volumes for all particles in each radius bin. Volume-weighted density profile is then just $\frac{N_{\mathrm{parts}}}{V_{\mathrm{parts}}}$.
12. δ_c: Central density contrast for each void, defined as the density contrast within $R_{\mathrm{eff}}/4$.
13. $\bar{\delta}$: Average density contrast of each void, defined as Mass/Volume relative to the average density of the Universe.

3.2 Simulations

Found in the `simulations` subdirectory, these scripts are used for running simulation with `GADGET2` `Springel:2005mi`.

- `config.sh`: Run to setup configuration of the simulations on the cluster where they are being generated. Paths to software and other options can be configured here.
- `generate_genetIC_ics.py`: Script to generate initial conditions using `genetIC` (**?**, `https://github.com/pynbody/genetIC`). To use the `import_noise` function, it may be necessary to use the velocity transfer branch at `https://github.com/svstopyra/genetIC/tree/velocity_transfer` in order to import white noise generated with `BORG` (Jasche & Lavaux (2019)) into `genetIC`. At some point, this feature will hopefully be incorporated into the main branch of `genetIC`.
- `parameters_full_forward_512.params`: Parameter file used for running forward simulations with `GADGET2`.
- `parameters_full_reverse_512.params`: Parameter file for running reverse simulations.
- `run_simulations.py`: Script to run both forward and reverse simulations, and perform cleanup operations.

4 Worked Example

To illustrate the usage of the code, an example is included in the file `example.py`. This example uses only the test data found in the `test_snaps` subfolder. Note that these are unrelated random Λ-CDM simulations, so the void catalogue we will build in this example consists *only* of spurious voids. Construction of a proper catalogue requires data from `https://cloud.aquila-consortium.org/s/bE4dKyJq6wZJ93D`. Examples showing how to use this are given in `voids_paper_plots.py`.

First, we import the needed libraries:

```
from void_analysis.simulation_tools import SnapshotGroup
from void_analysis import catalogue
import os
import pynbody
```

This will require the package to be installed on your python path. The package `pynbody` is required for much of the functionality of this package to work. `pynbody` is available for installation with `pip`.

First, we load the simulations and organise them with the `SnapshotGroup` class:

```
# Forward simulations:
snapList = [
    pynbody.load(
        os.path.join(base_path,f"sample{k}/forward/snapshot_001")
    )
    for k in range(1,4)
]
# Reverse simulations:
snapListRev = [
    pynbody.load(
        os.path.join(base_path,f"sample{k}/reverse/snapshot_001")
    )
    for k in range(1,4)
]

# Class that handles groups of related snapshots cleanly:
snaps = SnapshotGroup(
    snapList,snapListRev,low_memory_mode=False,swapXZ  = False,
    reverse = True,remap_centres=True
)
```

The first two parts here are just loading the forward and reverse simulations from the `test_snaps` subdirectory. The final part here instantiates a SnapshotGroup class that allows us to access snapshots and their derived properties more easily. An explanation of some of the parameters:

- `low_memory_mode=False`

  instructs the code to load the halo and anti-halo catalogues, along with the postprocessing data computed by `processSnapshots.py` script in sec. 3.1. Typically, you would want to avoid doing this (by setting

  `low_memory_mode=True`

) for large simulation sets, to avoid memory problems. Here, since our examples are quite light-weight, loading them is not likely to be problematic.

- `swapXY=False`

  : If True, this would swap the X and Z co-ordinates of particles when loading the snapshots. Depending on the geometry of the simulation snapshots, this may be needed to get them in the correct units.

- `reverse=True`

  : This reflects the positions of particles about the centre of the box. Combining

  `reverse=True`

  with

  `swapXY=False`

  gives the correct loading of the simulations stored on the Aquila Cloud, to put the positions in Equatorial co-ordinates. It isn't strictly necessary here, but we perform this transformation to illustrate how it would be implemented for real data.

- `remap\_centres=True`

  ensures that the same co-ordinate transformation is applied to the halo and void positions, so we have consistent co-ordinates.

Next, we set up the arguments for the catalogue construction code:

```
# Construction of an anti-halo void catalogue:
# Parameters:
rSphere = 25 # Radius out to which to search for voids
muOpt = 0.2 # Optimal choice of \mu_R, radius ratio
```

```
rSearchOpt = 3 # Optimal choice of \mu_S, search radius ratio
NWayMatch = False # Whether to use N-way matching code.
refineCentres = True # Whether to refine centres iteratively using all samples
sortBy = "radius" # Quantity to sort catalogue by
enforceExclusive = True # Whether to apply the algorithm purging duplicate
# voids from the catalogues.
mMin = 1e11 # Minimum mass halos to include (Note, this is effectively
# over-ridden by the minimum radius threshold)
mMax = 1e16 # Maximum mass halos to include (Note, this is effectively
# over-ridden by the maximum radius threshold)
rMin = 5 # Minimum radius voids to use
rMax = 30 # Maximum radius voids to use
m_unit = snaps.snaps[0]['mass'][0]*1e10
```

The most important parameters are `muOpt` and `rSearchOpt`, which correspond to μ_R and μ_S respectively from Stopyra et al. (2024). The values given in the example here are wildly unrealistic, and chosen only to give a viable illustration of the catalogue using these low-resolution test simulations. For actual scientific cases, see the paper for the appropriate choice of parameters.

Catalogue construction is done by first instantiating an instance of the combinedCatalogue class, and then computing the final catalogue:

```
cat = catalogue.combinedCatalogue(
snaps.snap_filenames,snaps.snap_reverse_filenames,\
muOpt,rSearchOpt,rSphere,\
ahProps=snaps.all_property_lists,hrList=snaps["antihalos"],\
max_index=None,\
twoWayOnly=True,blockDuplicates=True,\
massRange = [mMin,mMax],\
NWayMatch = NWayMatch,r_min=rMin,r_max=rMax,\
additionalFilters = None,verbose=False,\
refineCentres=refineCentres,sortBy=sortBy,\
enforceExclusive=enforceExclusive
)
# Build the anti-halo catalogue
finalCat = cat.constructAntihaloCatalogue()
```

Here, we have set `additionalFilters` to `None`, but when working with actual posterior sampled simulations, this is where a signal-to-noise filter would be supplied (see examples in `voids_paper_plots.py`). Note that the final call to `constructAntihaloCatalogue` is necessary to generate an output. If this is not called, the final catalogue will be empty!

We can now examine various properties of this mock void catalogue:

```
# Properties of the antihalo catalogue:
radii = cat.getMeanProperty("radii")
mass = cat.getMeanProperty("mass")
centres = cat.getMeanCentres()

# Properties of all the constituent halos:
all_radii = cat.getAllProperties("radii")
all_masses = cat.getAllProperties("mass")
all_centres = cat.getAllCentres()

# All halo indices:
halo_indices = cat.get_final_catalogue(short_list=False)
```

We can compute the mean radius, mass, and position of voids in the catalogue:

```
In [59]: radii
Out[59]:
[array([6.94403629, 7.86756052, 5.49556746, 5.47383931]),
array([0.53356164, 0.46516516, 0.05645594, 0.30436382])]

In [60]: mass
Out[60]:
[array([4.65081068e+13, 6.47668499e+13, 2.56901938e+13, 2.19990752e+13]),
array([9.70925872e+12, 1.16233394e+13, 2.71441559e+12, 3.02761738e+12])]

In [61]: centres
Out[61]:
array([[ -0.37999936, -14.51207332,  -9.41482562],
[ -6.68264413,   6.10391777,   7.30281434],
[ -6.94982653,  11.9624704 ,   9.11765419],
[-17.15245043, -14.39614657,   2.22468294]])
```

With the given parameters, there are four voids. The lists for `radii` and `mass` give first the values of these four quantities, and then the standard deviations of the means for each.

If we wish to examine the properties of all the anti-halos corresponding to a given void in the catalogue, across the posterior samples, we can look at the output of the `getAllProperties` and `getAllCentres` functions:

```
In [62]: all_radii
Out[62]:
array([[6.18946618,        nan, 7.69860641],
[8.86128572, 7.85348824, 6.8879076 ],
[       nan, 5.57540821, 5.4157267 ],
[5.04340386,        nan, 5.90427476]])

In [63]: all_masses
Out[63]:
array([[3.27771415e+13,            nan, 6.02390722e+13],
[8.00234783e+13, 7.79564498e+13, 3.63206216e+13],
[           nan, 2.95289571e+13, 2.18514305e+13],
[1.77173776e+13,            nan, 2.62807728e+13]])

In [64]: all_centres
Out[64]:
array([[[  5.2190234 ,  -8.68813632, -14.45268237],
[ -5.78228292,  17.55552727,  -9.00987174],
[         nan,          nan,          nan],
[-14.58387323, -17.74368193,  -1.49044033]],

[[         nan,          nan,          nan],
[ -8.29333347,   9.96858431,  11.75023604],
[ -5.3894499 ,  12.36455405,   5.33469134],
[         nan,          nan,          nan]],

[[ -5.97902213, -20.33601031,  -4.37696886],
[ -5.97231601,  -9.21235828,  19.1680787 ],
[ -8.51020317,  11.56038676,  12.90061704],
[-19.72102762, -11.04861121,   5.93980621]]])
```

When no anti-halo was found corresponding to a particular void in a given sample, the value are set to `nan`. In the final catalogue, these correspond to entries with -1:

```
In [66]: finalCat
Out[66]:
array([[ 2, -1,  1],
[ 1,  1,  2],
[-1,  2,  4],
[ 3, -1,  3]])
```

These are the halo indices in the shortened list of halos satisfying the range criteria (radius range, distance from centre of the box). If we want the indices of halos in the individual reverse simulation halo catalogues, we can compute them using:

```
In [67]: halo_indices = cat.get_final_catalogue(short_list=False)

In [68]: halo_indices
Out[68]:
array([[ 3, -1,  0],
[ 0,  1,  1],
[-1,  2,  5],
[ 4, -1,  4]])
```

with again, negative entries indicating that no halo was found.

References

Astropy Collaboration et al., 2013, A&A, 558, A33

Astropy Collaboration et al., 2018, AJ, 156, 123

Astropy Collaboration et al., 2022, ApJ, 935, 167

Bellock K., Godber N., Kahn P., 2021, bellockk/alphashape: v1.3.1 Release, doi:10.5281/zenodo.4697576, https://doi.org/10.5281/zenodo.4697576

Clark A., 2015, Pillow (PIL Fork) Documentation, https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

Collette A., et al., 2022, h5py/h5py: 3.7.0, doi:10.5281/zenodo.6575970, https://doi.org/10.5281/zenodo.6575970

Cooke D., Hochberg T., Alted F., Vilata I., Wiebe M., de Menten G., Valentino A., McLeod R. A., 2018, NumExpr: Fast numerical expression evaluator for NumPy, doi:10.5281/zenodo.1492916, https://doi.org/10.5281/zenodo.1492916

Foreman-Mackey D., Hogg D. W., Lang D., Goodman J., 2013, PASP, 125, 306

Górski K. M., Hivon E., Banday A. J., Wandelt B. D., Hansen F. K., Reinecke M., Bartelmann M., 2005, ApJ, 622, 759

Hand N., Feng Y., Beutler F., Li Y., Modi C., Seljak U., Slepian Z., 2018, AJ, 156, 160

Harris C. R., et al., 2020, Nature, 585, 357

Hunter J. D., 2007, Computing in Science & Engineering, 9, 90

Jasche J., Lavaux G., 2019, A&A, 625, A64

Knollmann S. R., Knebe A., 2009, ApJS, 182, 608

Krekel H., Oliveira B., Pfannschmidt R., Bruynooghe F., Laugher B., Bruhin F., 2004, pytest x.y, https://github.com/pytest-dev/pytest

Lewis A., Challinor A., 2011, CAMB: Code for Anisotropies in the Microwave Background, Astrophysics Source Code Library, record ascl:1102.026

Meurer A., et al., 2017, PeerJ Computer Science, 3, e103

Murray S. G., Power C., Robotham A. S. G., 2013, Astronomy and Computing, 3, 23

Neyrinck M. C., Gnedin N. Y., Hamilton A. J., 2005, Monthly Notices of the Royal Astronomical Society, 356, 1222

Pontzen A., Roškar R., Stinson G. S., Woods R., Reed D. M., Coles J., Quinn T. R., 2013, pynbody: Astrophysics Simulation Analysis for Python

Pontzen A., Slosar A., Roth N., Peiris H. V., 2016, Phys. Rev., D93, 103519

Ramachandran P., Varoquaux G., 2011, Computing in Science and Engineering, 13, 40

Sinha M., Garrison L. H., 2020, MNRAS, 491, 3022

Stopyra S., Peiris H. V., Pontzen A., 2021a, Monthly Notices of the Royal Astronomical Society, 500, 4173

Stopyra S., Peiris H. V., Pontzen A., Jasche J., Natarajan P., 2021b, Monthly Notices of the Royal Astronomical Society, 507, 5425

Stopyra S., Peiris H. V., Pontzen A., Jasche J., Lavaux G., 2023, Monthly Notices of the Royal Astronomical Society, 527, 1244

Stopyra S., Peiris H. V., Pontzen A., Jasche J., Lavaux G., 2024, Monthly Notices of the Royal Astronomical Society, 531, 2213

Sutter P., et al., 2015, Astron. Comput., 9, 1

Virtanen P., et al., 2020, Nature Methods, 17, 261

Waskom M. L., 2021, Journal of Open Source Software, 6, 3021

Zonca A., Singer L., Lenz D., Reinecke M., Rosset C., Hivon E., Gorski K., 2019, Journal of Open Source Software, 4, 1298