Cosmotile

Steven Murray

May 02, 2024

CONTENTS

Create cosmological lightcones from coeval simulations.

This algorithm is taken from the code in https://github.com/piyanatk/cosmotile, but is repackaged and re-tooled.

CHAPTER

ONE

FEATURES

- Fast tiling of finite, periodic cosmic simulations onto arbitrary angular coordinates.
- Generate different realizations by translation and rotation.

CHAPTER

TWO

INSTALLATION

You can install *Cosmotile* via pip from PyPI:

```
$ pip install cosmotile
```

CHAPTER

THREE

USAGE

Please see the *Command-line Reference* for details.

CHAPTER

FOUR

CONTRIBUTING

Contributions are very welcome. To learn more, see the *Contributor Guide*.

CHAPTER

FIVE

LICENSE

Distributed under the terms of the *MIT license*, *Cosmotile* is free and open source software.

CHAPTER

SIX

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

CHAPTER

SEVEN

CREDITS

This project was generated from @cjolowicz's Hypermodern Python Cookiecutter template.

The algorithm used in this repository is derived from the `cosmotile` module in https://github.com/nithyanandan/AstruUtils, which was later modularised in https://github.com/piyanatk/cosmotile.

CHAPTER

EIGHT

ACKNOWLEDGMENTS

If you find `cosmotile` useful in your project, please star this repository and, if applicable, cite https://arxiv.org/abs/1708.00036.

8.1 Usage

A simple example of using cosmotile is as follows:

```
import cosmotile
import numpy as np

# Create an artificial co-eval "simulation" that is periodic on its boundaries.
# In this case, the simulation is simply zeros with ones on all three planes:
box = np.zeros((100, 100, 100))
box[0] = 1
box[:, 0] = 1
box[:, :, 0] = 1

# cosmotile interpolates a regular, periodic box onto an arbitrary set of angular
# coordinates on a single spherical shell (to get a full 'lightcone', just re-call
# the function for different shell radii).
# Evaluate at a latitude of zero, around the full longitude.
lat = np.zeros(1000)
lon = np.linspace(0, 2*np.pi, 1000, endpoint=False)

lc_slice = cosmotile.make_lightcone_slice(
    coeval = box,
    coeval_res = 1.0,         # resolution of each cell of the box (in cMpc)
    latitude=lat,
    longitude=lon,
    distance_to_shell = 100,  # the radius of the shell (in cMpc)
)

# To get a lightcone between 100-200 Mpc:
distances = np.linspace(100, 200, 101)
lc = np.zeros((len(lat), len(distances)))
```

8.2 Reference

8.2.1 cosmotile

Cosmotile.

cosmotile.**apply_rsds**(*field*, *los_displacement*, *distance*, *n_subcells=4*)

Apply redshift-space distortions to a field.

Notes

To ensure that we cover all the slices in the field after the velocities have been applied, we extrapolate the densities and velocities on either end by the maximum velocity offset in the field. Then, to ensure we don't pick up cells with zero particles (after displacement), we interpolate the slices onto a finer regular grid (in comoving distance) and then displace the field on that grid, before interpolating back onto the original slices.

Parameters

- **field** (*ndarray*) – The field to apply redshift-space distortions to, shape (nslices, ncoords).
- **los_displacement** (*ndarray*) – The line-of-sight "apparent" displacement of the field, in pixel coordinates. Equal to `v / H(z) / cell_size`. Positive values are towards the observer, shape `(nslices, ncoords)`.
- **distance** (*ndarray*) – The comoving distance to each slice in the field, in units of the cell size. shape (nslices,).
- **n_subcells** (*int*)

Return type
ndarray

cosmotile.**get_distance_to_shell_from_redshift**(*z*, *cell_size*, *cosmo=FlatLambdaCDM(name='Planck18', H0=<Quantity 67.66 km / (Mpc s)>, Om0=0.30966, Tcmb0=<Quantity 2.7255 K>, Neff=3.046, m_nu=<Quantity [0., 0., 0.06] eV>, Ob0=0.04897)*)

Get a distance to a shell, in units of cell size, from a given redshift.

Parameters

- **z** (*float*) – The redshift
- **cell_size** (*Quantity*) – The resolution of the coeval simulation, in comoving units.
- **cosmo** (*FLRW*) – The astropy cosmology.

Returns
The distance, in units of pixels, to the shell.

Return type
distance

cosmotile.**make_healpix_lightcone_slice**(*nside*, *order='ring'*, ***kwargs*)

Create a healpix lightcone slice in angular coordinates.

This is a simple wrapper around *make_lightcone_slice()* that sets up angular co-ordinates from a healpix grid.

Parameters

- **nside** (*int*) – The Nside parameter of the healpix map.
- **order** (*Literal['ring', 'nested']*) – The ordering of the pixels in the healpix map.
- **kwargs** (*Any*)

Return type
Generator

:param All other parameters are passed through to *make_lightcone_slice()*.:

cosmotile.**make_lightcone_slice**(*, *coevals*, ***kwargs*)

Create a lightcone slice in angular coordinates from two coeval simulations.

Interpolates the input coeval box to angular coordinates.

Parameters

- **coevals** (*Sequence[ndarray] | ndarray*) – An iterable of rectangular coeval simulations to interpolate to the angular coordinates. Must have three dimensions (not necessarily the same size). Each box must have the same shape, and all are assumed to be at the same coordinates. Each coeval box can be a different simulated field.
- **kwargs** (*Any*)

Return type
Generator

:param All other parameters are passed to *make_lightcone_slice_interpolator()*.:

Yields
field – Each interpolated field on the angular coordinates.

Parameters

- **coevals** (*Sequence[ndarray] | ndarray*)
- **kwargs** (*Any*)

Return type
Generator

cosmotile.**make_lightcone_slice_interpolator**(*, *latitude*, *longitude*, *distance_to_shell*, *interpolation_order=1*, *origin=None*, *rotation=None*)

Create a callable interpolator for a lightcone slice.

Parameters

- **latitude** (*ndarray*) – An array of latitude coordinates onto which to tile the box. In radians from -pi/2 to pi/2
- **longitude** (*ndarray*) – An array, same size as latitude, of longitude coordinates onto which to tile the box. In radians from 0 to 2pi.
- **distance_to_shell** (*float*) – The distance to the spherical shell onto which to interpolate, in units of the cell-size of the coeval box(es) you wish to interpolate.
- **interpolation_order** (*int*) – The order of interpolation. Must be in the range 0-5.
- **origin** (*ndarray | tuple[float, float, float] | None*) – Define the location of the centre of the spherical shell, assuming that the (0,0,0) pixel of the coeval box is at (0,0,0) in cartesian coordinates.

- **rotation** (*Rotation | None*) – The rotation by which to rotate the spherical coordinates before interpolation. This is done before shifting the origin, and is equivalent to rotating the coeval box beforing tiling it.

Returns
A callable that takes a 3D array of coeval values and returns a 2D array of interpolated values on a redshift slice.

Return type
interpolator

cosmotile.**make_lightcone_slice_vector_field**(*coeval_vector_fields*, *interpolator*)

Interpolate a 3D vector field to a lightcone slice as a line-of-sight component.

This takes a sequence of 3D vector fields, eg. the velocity field, and interpolates each component to the lightcone slice. It then computes the line-of-sight component of each interpolated vector field, where positive values are oriented towards the observer.

Parameters

- **coeval_vector_fields** (*Sequence[Sequence[ndarray]]*) – An iterable of 3D vector fields to interpolate to the lightcone slice. Each vector field must be an iterable of 3 3D arrays, each of the same shape.
- **interpolator** (*Callable[[ndarray], ndarray]*) – A callable that takes a 3D array of coeval values and returns a 2D array of interpolated values on a redshift slice. This should be created by *make_lightcone_slice_interpolator()* using the properties of the coeval vector fields.

Yields
los_component – The line-of-sight component of each interpolated vector field.

Return type
Generator

cosmotile.**transform_to_pixel_coords**(*, *comoving_radius*, *latitude*, *longitude*, *origin=None*, *rotation=None*)

Transform input spherical coordinates to pixel coordinates wrt a coeval box.

Parameters

- **comoving_radius** (*Quantity*) – The radius of the spherical coordinates (in units of the cell size).
- **latitude** (*ndarray*) – An array of latitude coordinates onto which to tile the box. In radians from -pi/2 to pi/2
- **longitude** (*ndarray*) – An array, same size as latitude, of longitude coordinates onto which to tile the box. In radians from 0 to 2pi.
- **origin** (*Quantity | tuple[float, float, float] | None*) – Define the location of the centre of the spherical shell, assuming that the (0,0,0) pixel of the coeval box is at (0,0,0) in cartesian coordinates. In units of the cell size.
- **rotation** (*Rotation | None*) – The rotation by which to rotate the spherical coordinates before interpolation. This is done before shifting the origin, and is equivalent to rotating the coeval box beforing tiling it.

Return type
ndarray

8.3 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- *Code of Conduct*

8.3.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

8.3.2 How to request a feature

Request features on the Issue Tracker.

8.3.3 How to set up your development environment

You need Python 3.8+. Install the package with development requirements:

```
$ pip install -e .
```

You can now run an interactive Python session.

8.3.4 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the *tests* directory, and are written using the pytest testing framework.

8.3.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

8.4 Contributor Covenant Code of Conduct

8.4.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

8.4.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment

- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.4.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

8.4.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

8.4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at steven.g.murray@asu.edu. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

8.4.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

8.4.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

8.5 License

PYTHON MODULE INDEX

C

INDEX