BAGGInS: Bayesian Analysis of Galaxy-Galaxy Interactions in Simulations

Release 1.0

Alex Rawlings

Oct 21, 2024

CONTENTS:

CHAPTER

ONE

PYTHON API

1.1 BAGGInS

Here you can find a list of available functions and classes in the BAGGInS package.

1.1.1 _backend package

1.1.1.1 Submodules

1.1.1.2 _backend.Logging module

_backend.Logging.**setup_logger**(*name*, *console_level='WARNING'*, *logfile=None*, *file_level='WARNING'*, *capture_warnings=True*)

Set up a logger instance

Parameters

- **name** (*str*) – name of logger
- **console_level** (*str, optional*) – logging level for console printing, by default "WARNING"
- **logfile** (*path-like, optional*) – file to print logs to, by default None
- **file_level** (*str, optional*) – logging level for file printing, by default "WARNING"
- **capture_warnings** (*bool, optional*) – redirect all warnings to the logger, by default True

Returns

CustomLogger – logger

1.1.1.3 _backend.States module

class _backend.States.**PublishingState**

Bases: _State

__init__()

Simple object to store the state of a plotting-style setting

class _backend.States.**TmpDirRegister**(*basename*)

Bases: object

__init__(*basename*)

Class to hold a register of temporary directories. This means that temporary directories are particular to the given invocation of baggins, and can be cleaned up at exit without affecting the temporary directories of other invocations running concurrently.

Parameters
basename (*str, path-like*) – base temporary directory path to which a timestamp will be appended

`make_new_dir()`

Make a new temporary directory, and add it to the register

`property register`

1.1.1.4 Module contents

_backend.**setup_logger**(*name*, *console_level='WARNING'*, *logfile=None*, *file_level='WARNING'*, *capture_warnings=True*)

Set up a logger instance

Parameters

- **name** (*str*) – name of logger
- **console_level** (*str, optional*) – logging level for console printing, by default "WARNING"
- **logfile** (*path-like, optional*) – file to print logs to, by default None
- **file_level** (*str, optional*) – logging level for file printing, by default "WARNING"
- **capture_warnings** (*bool, optional*) – redirect all warnings to the logger, by default True

Returns
CustomLogger – logger

1.1.2 analysis package

1.1.2.1 Subpackages

analysis.analysis_classes package

Submodules

analysis.analysis_classes.BHBinary module

analysis.analysis_classes.BHBinaryData module

analysis.analysis_classes.Brownian module

analysis.analysis_classes.BrownianData module

analysis.analysis_classes.ChildSim module

analysis.analysis_classes.ChildSimData module

analysis.analysis_classes.CoreKick module

analysis.analysis_classes.GaussianProcesses module

analysis.analysis_classes.GrahamModels module

analysis.analysis_classes.HDF5Base module

1.1.2.2 Submodules

1.1.2.3 analysis.analyse_ketju module

1.1.2.4 analysis.analyse_snap module

1.1.2.5 analysis.general module

1.1.2.6 analysis.masks module

analysis.masks.**get_all_id_masks**(*snap*, *family='stars'*)

Return a dict of masks that mask the chosen particles by ID number to a specific galaxy, assuming each galaxy has a single SMBH in it.

Parameters

- **snap** (*pygad.Snapshot*) – snapshot to mask
- **family** (*str, optional*) – particle type we want to mask, by default 'stars'

Returns

- **masks** (*dict*)
- *keys corresponding to the host galaxy SMBH particle ID, values to pygad.*
- *snapshot.masks.IDMask objects*

analysis.masks.**get_all_radial_masks**(*snap*, *radius*, *centre=None*, *id_masks=None*, *family='stars'*)

Return a dict of masks that mask the chosen particles by radial distance to the desired centre. Either a ball mask or a shell mask may be constructed. Additionally, the option to initially filter particles by ID to constrain the mask to those particles belonging to a given galaxy is possible.

Parameters

- **snap** (*pygad.Snapshot*) – snapshot to mask
- **radius** (*float, tuple*) – radius to constrain the particles to - can be either a number to construct a ball mask, or a tuple of (inner_radius, outer_radius) to construct a shell mask
- **centre** (*pygad.UnitArr, optional*) – centre from which the radial measurements should be made, by default None
- **id_mask** (*dict, optional*) – ID masks to constrain particles to a given galaxy, by default None

- **family** (*str, optional*) – particle type to construct the mask for, by default None (all particle used)

Returns

masks (*dict*) – pygad mask objects, with keys corresponding to the host galaxy SMBH particle ID number

Raises

ValueError – centre method is invalid

analysis.masks.**get_binding_energy_mask**(*snap*, *energy=None*, *id_mask=None*, *family=None*)

Mask particles into bins of binding energy. Assumes that we are in centre of mass coordinates (including velocity): no centring is done.

Parameters

- **snap** (*pygad.Snapshot*) – snapshot to mask
- **energy** (*array-like, list of tuples, optional*) – energy bins to mask particles into - can either be a regular array, in which a "ball" mask is created, or a list of tuples, in which a "shell" mask is created. By default (None) creates a "shell" type mask from the 5% to 95% quantile of binding energy in 20 bins evenly spaced in logscale
- **id_mask** (*pygad.snapshot.masks.IDMask, optional*) – ID mask to constrain particles to a given galaxy, by default None
- **family** (*_type_, optional*) – particle type we want to mask, by default None (all)

Yields

- **mask** (*pygad.snapshots.mask.IDMask*) – mask for that energy interval
- **energy** (*array-like, list of tuples*) – energy sequence
- *str* – energy units

Raises

ValueError – invalid energy input

analysis.masks.**get_id_mask**(*snap*, *bhid*, *family='stars'*)

Obtain a mask that allows filtering of particular particles. Particles are first filtered to the host galaxy depending on their particle ID number, where it is necessarily assumed that the particle ID ordering follows:

galaxy 1 IDs –> galaxy 2 IDs

To obtain a mask of all particles in a system, the routine can be run for each particle family, and the masks combined as e.g.:

all_id_mask = star_id_mask & dm_id_mask & bh_id_mask

#TODO should this be a separate method??

Parameters

- **snap** (*pygad.Snapshot*) – snapshot to mask
- **bhid** (*int*) – SMBH id number we want to find particles around
- **family** (*str, optional*) – particle type we want to mask, by default 'stars'

Returns

pygad.snapshot.masks.IDMask – object to mask future snapshots of the same simulation

Raises

- **RuntimeError** – ID ordering is not divided by galaxy

- **RuntimeError** – only galaxy in the system

analysis.masks.**get_radial_mask**(*snap*, *radius*, *centre=None*, *id_mask=None*, *family=None*)

Create a radial-based mask, can be either a ball or a shell. The default function arguments are designed for compatability with get_all_radial_masks.

Parameters

- **snap** (*pygad.Snapshot*) – snapshot to mask
- **radius** (*float, tuple*) – radius to constrain the particles to - can be either a number to construct a ball mask, or a tuple of (inner_radius, outer_radius) to construct a shell mask
- **centre** (*pygad.UnitArr, optional*) – centre from which the radial measurements should be made, by default None
- **id_mask** (*pygad.snapshot.masks.IDMask, optional*) – ID mask to constrain particles to a given galaxy, by default None
- **family** (*str, optional*) – particle type to construct the mask for, by default None (all particle used)

Returns

mask (*pygad.snapshot.masks.BallMask*) – radial mask of particles

Raises

ValueError – radius input is invalid

1.1.2.7 analysis.orbits module

1.1.2.8 analysis.voronoi module

1.1.2.9 Module contents

1.1.3 cosmology package

1.1.3.1 Submodules

1.1.3.2 cosmology.conversions module

cosmology.conversions.**time2z**(*t*, *H0=67.36*, *pres=False*)

Estimate the redshift for a corresponding cosmic time from Carmeli 2008

Parameters

- **t** (*float*) – cosmic time since big bang [Gyr] -> pres = False time before present [Gyr] -> pres = True
- **H0** (*float, optional*) – Hubble constant in km/s/Mpc, by default 100*cosmology['h']
- **pres** (*bool, optional*) – t is given from today not Big Bang?, by default False

Returns

float – redshift

1.1.3.3 cosmology.cosmology module

cosmology.cosmology.**angular_diameter_distance**(*z*, *cosmology={'h': 0.6736, 'omega_L': 0.6847, 'omega_M': 0.3153, 'zeq': 3402}*)

Determine the angular diameter distance for a flat universe

Parameters

- **z** (*float*) – redshift

- **cosmology** (*dict, optional*) – cosmological parameters, by default cosmology

Returns
float – angular diameter distance [kpc]

cosmology.cosmology.**get_a0r**(*z*, *cosmology={'h': 0.6736, 'omega_L': 0.6847, 'omega_M': 0.3153, 'zeq': 3402}*)

Determine the value a_0*r from MBW eq. 3.106, for use in cosmological distance calculations

Parameters

- **z** (*float*) – redshift
- **cosmology** (*dict, optional*) – cosmological parameters, by default cosmology

Returns
float – a0*r [kpc]

Raises
AssertionError – redshift larger than redshift of matter radiation equality

cosmology.cosmology.**luminosity_distance**(*z*, *cosmology={'h': 0.6736, 'omega_L': 0.6847, 'omega_M': 0.3153, 'zeq': 3402}*)

Determine the luminosity distance for a flat universe

Parameters

- **z** (*float*) – redshift
- **cosmology** (*dict, optional*) – cosmological parameters, by default cosmology

Returns
float – luminosity distance [kpc]

PYTHON MODULE INDEX

INDEX

Symbols

A

C

G

L

M

P

R

S

T