
SEYFERT

Release 1.3.7dev

Luca Paganin, Marco Bonici, Stefano Davini

Jun 05, 2021

Contents:

1	Cosmology modules	3
1.1	Cosmology	3
1.1.1	seyfert.cosmology.cosmology Module	3
1.1.1.1	Classes	3
1.1.1.2	Class Inheritance Diagram	10
1.2	Physical parameters	10
1.2.1	seyfert.cosmology.parameter Module	10
1.2.1.1	Classes	10
1.2.1.2	Class Inheritance Diagram	16
1.3	Power Spectrum	16
1.3.1	seyfert.cosmology.power_spectrum Module	16
1.3.1.1	Classes	16
1.3.1.2	Class Inheritance Diagram	19
1.4	Boltzmann solver	19
1.4.1	seyfert.cosmology.boltzmann_solver Module	19
1.4.1.1	Classes	19
1.4.1.2	Class Inheritance Diagram	22
1.5	Redshift Density	22
1.5.1	seyfert.cosmology.redshift_density Module	22
1.5.1.1	Functions	22
1.5.1.2	Classes	23
1.5.1.3	Class Inheritance Diagram	27
1.6	Bias	27
1.6.1	seyfert.cosmology.bias Module	27
1.6.1.1	Classes	27
1.6.1.2	Class Inheritance Diagram	35
1.7	Weight functions	36
1.7.1	seyfert.cosmology.weight_functions Module	36
1.7.1.1	Functions	36
1.7.1.2	Classes	36
1.7.1.3	Class Inheritance Diagram	44
1.8	Angular power spectra	44
1.8.1	seyfert.cosmology.c_ells Module	44
1.8.1.1	Functions	44
1.8.1.2	Classes	45
1.8.1.3	Class Inheritance Diagram	52

2 Indices and tables	53
Python Module Index	55
Index	57

SEYFERT (SurvEY FishEr foRecast Tool) is a code to perform forecast of cosmological parameters measurement for several cosmological probes. Up to now are included:

- Weak Lensing
- Spectroscopic Galaxy Clustering
- Photometric Galaxy Clustering
- Void Clustering

The code is written in Python in a modular way. It is actually interfaced with the Boltzmann solver [CAMB](#), although other codes can be easily added. The code can be installed executing:

```
./installer.sh
```


1.1 Cosmology

1.1.1 `sefirt.cosmology.cosmology` Module

Module hosting the cosmology class and its HDF5 file-interface. The Cosmology possesses cosmological parameters values and it can compute all relevant cosmological functions, like the Hubble parameter and the co-moving distance. For now accepted cosmological parameters are:

- w_0 , CPL dark energy equation of state intercept
- w_a , CPL dark energy equation of state slope
- h , the dimensionless Hubble constant
- $\sum m_\nu$, the sum of the neutrino mass eigenstates
- σ_8
- n_s , the scalar spectral index
- Ω_m , the cold matter density parameter
- Ω_b , the baryon density parameter
- Ω_{DE} , the dark energy density parameter

1.1.1.1 Classes

`AbstractH5FileIO([root])`

<code><i>Cosmology</i></code> ([params, flat, model_name, ...])	Class for managing the cosmology.
<code><i>CosmologyError</i></code>	

`H5Cosmology`(**kwargs)

`H5PowerSpectrum`(**kwargs)

continues on next page

Table 1 – continued from previous page

<code>Path(*args, **kwargs)</code>	PurePath subclass that can make system calls.
<code>PhysicalParameter([name, fiducial, ...])</code>	
<code>PowerSpectrum([power_spectrum_config, ...])</code>	Class to manage the matter power spectrum. It can be used to calculate the linear and non linear power spectra,
<code>PowerSpectrumConfig(**kwargs)</code>	

Cosmology

class `seyfert.cosmology.cosmology.Cosmology`(*params=None, flat=None, model_name=None, z_grid=None, cosmology_name=None*)

Bases: object

Class for managing the cosmology.

Parameters

- **params** (Optional[Dict[str, *PhysicalParameter*]]) – a dictionary with the cosmological parameters, stored as *PhysicalParameter* instances.
- **z_grid** (Optional[ndarray]) – a `np.ndarray` storing the redshift grid on which to compute functions.
- **dimensionless_hubble_array** – a `np.ndarray` storing the values of the dimensionless Hubble parameter $E(z)$ sampled on the grid.
- **dimensionless_comoving_distance_array** – a `np.ndarray` storing the values of the comoving distance $r(z)$ sampled on the grid.
- **power_spectrum** – the matter power spectrum, instance of *PowerSpectrum*.

Attributes Summary

<i>E_z</i>	rtype ndarray
<i>H0</i>	Hubble constant in $\text{km s}^{-1} \text{Mpc}^{-1}$
<i>H_z</i>	rtype ndarray
<i>OmDE</i>	Dark energy density parameter Ω_{DE}
<i>OmK</i>	Curvature density parameter Ω_k
<i>Omb</i>	Baryon density parameter Ω_b
<i>Omm</i>	Cold matter density parameter Ω_m
<i>cosmo_pars_current</i>	Current values of the cosmological parameters
<i>growth_factor_z</i>	The growth factor $D(z)$ as a function of redshift.
<i>h</i>	Dimensionless Hubble constant
<i>k_grid</i>	rtype ndarray
<i>mnu</i>	Sum of the neutrino mass eigenstates $\sum m_\nu$ in eV

continues on next page

Table 2 – continued from previous page

<i>ns</i>	Scalar spectral index n_s
<i>r_tilde_z</i>	rtype ndarray
<i>r_z</i>	rtype ndarray
<i>sigma8</i>	σ_8 parameter
<i>transfer_function_k</i>	rtype ndarray
<i>w0</i>	Dark energy w_0 CPL parameter
<i>wa</i>	Dark energy w_a CPL (slope) parameter

Methods Summary

<i>checkParameters()</i>	Method checking values of cosmological parameters, returning error if values are invalid.
<i>computeComovingDistance(z)</i>	Method to compute the comoving distance as a function of the redshift.
<i>computeDimensionlessComovingDistance(z)</i>	Method to compute the dimensionless comoving distance at a given redshift, computed according the formula:
<i>computeDimensionlessHubbleParameter(z)</i>	Method to compute the dimensionless Hubble parameter as a function of the redshift.
<i>computeHubbleParameter(z)</i>	Method to compute the Hubble parameter as a function of the redshift.
<i>computeReciprocalDimensionlessHubbleParameter(z)</i>	Shortcut method to compute integrals of the reciprocal of $E(z)$
<i>computeSigmaR(R)</i>	Method to compute the variance of the filtered matter density fluctuations.
<i>evaluateOverRedshiftGrid()</i>	Method which evaluates the Hubble parameter and the comoving distance on the redshift grid.
<i>evaluatePowerSpectrum(workdir, ...)</i>	Method computing the power spectrum for the given cosmology.
<i>fromHDF5(file[, root, load_power_spectrum])</i>	rtype <i>Cosmology</i>
<i>loadFromHDF5(file[, root, load_power_spectrum])</i>	rtype None
<i>saveToHDF5(file[, root, save_power_spectrum])</i>	rtype None
<i>sigmaR(R, P_lin, k)</i>	Function to compute the value σ_R of the matter fluctuation variance filtered at a given radius value R

Attributes Documentation

E_z

Return type ndarray

H0

Hubble constant in $\text{km s}^{-1} \text{Mpc}^{-1}$

Return type float

H_z

Return type ndarray

OmDE

Dark energy density parameter Ω_{DE}

Return type float

OmK

Curvature density parameter Ω_k

Return type float

OmB

Baryon density parameter Ω_b

Return type float

OmM

Cold matter density parameter Ω_m

Return type float

cosmo_pars_current

Current values of the cosmological parameters

Return type Dict[str, float]

growth_factor_z

The growth factor $D(z)$ as a function of redshift. The convention is that this growth factor is a decreasing function of z .

Return type ndarray

h

Dimensionless Hubble constant

Return type float

k_grid

Return type ndarray

mnu

Sum of the neutrino mass eigenstates $\sum m_\nu$ in eV

Return type float

ns

Scalar spectral index n_s

Return type float

r_tilde_z

Return type ndarray

r_z

Return type ndarray

sigma8

σ_8 parameter

Return type float

transfer_function_k

Return type ndarray

w0

Dark energy w_0 CPL parameter

Return type float

wa

Dark energy w_a CPL (slope) parameter

Return type float

Methods Documentation

checkParameters()

Method checking values of cosmological parameters, returning error if values are invalid.

Return type None

computeComovingDistance(z)

Method to compute the comoving distance as a function of the redshift.

Return type Union[float, ndarray]

computeDimensionlessComovingDistance(z)

Method to compute the dimensionless comoving distance at a given redshift, computed according the formula:

$$\tilde{r}(z) = \int_0^z \frac{dz}{E(z)}$$

Parameters **z** (Union[float, ndarray]) – the value of the redshift at which to compute the comoving distance

Return type Union[float, ndarray]

Returns the value of the comoving distance at the given redshift(s) **z**

computeDimensionlessHubbleParameter(z)

Method to compute the dimensionless Hubble parameter as a function of the redshift.

Return type Union[float, ndarray]

computeHubbleParameter(z)

Method to compute the Hubble parameter as a function of the redshift.

Return type Union[float, ndarray]

computeReciprocalDimensionlessHubbleParameter(z)

Shortcut method to compute integrals of the reciprocal of $E(z)$

Return type Union[float, ndarray]

computeSigmaR(R)

Method to compute the variance of the filtered matter density fluctuations. It computes the variance using a top-hat filter in Fourier space with at a given radius R

Parameters R (Union[int, float, ndarray]) – radius or radii grid, units are Mpc

Return type Union[float, ndarray]

Returns the value(s) of σ_R at given radius value(s)

evaluateOverRedshiftGrid()

Method which evaluates the Hubble parameter and the comoving distance on the redshift grid.

Return type None

evaluatePowerSpectrum(*workdir*, *power_spectrum_config*)

Method computing the power spectrum for the given cosmology. It will make a call to the Boltzmann code selected for the computation.

Parameters

- **workdir** (Union[str, Path]) – working directory
- **power_spectrum_config** (PowerSpectrumConfig) – power spectrum configuration, instance of PowerSpectrumConfig.

Return type None

classmethod fromHDF5(*file*, *root*='/', *load_power_spectrum*=True)

Return type *Cosmology*

loadFromHDF5(*file*, *root*='/', *load_power_spectrum*=True)

Return type None

saveToHDF5(*file*, *root*='/', *save_power_spectrum*=True)

Return type None

static sigmaR(R , P_{lin} , k)

Function to compute the value σ_R of the matter fluctuation variance filtered at a given radius value R

Parameters

- R (Union[int, float]) – the radius of the filter
- P_{lin} (ndarray) – the linear matter power spectrum as a function of the wavenumber k
- k (ndarray) – the wavenumber grid k

Return type float

Returns the value of σ_R

CosmologyError

exception seyfert.cosmology.cosmology.CosmologyError

H5Cosmology

class seyfert.cosmology.cosmology.H5Cosmology(**kwargs)
Bases: seyfert.file_io.hdf5_io.AbstractH5FileIO

Methods Summary

<i>readBuildingData()</i>	rtype None
<hr/>	
<i>writeObjectToFile(obj)</i>	rtype None
<hr/>	
<i>writeToObject(obj)</i>	rtype None

Methods Documentation

readBuildingData()

Return type None

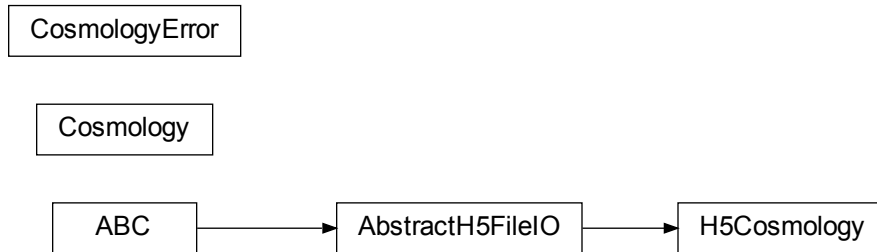
writeObjectToFile(obj)

Return type None

writeToObject(obj)

Return type None

1.1.1.2 Class Inheritance Diagram



1.2 Physical parameters

1.2.1 `seyfert.cosmology.parameter` Module

1.2.1.1 Classes

`GenericDictInterface([data_dict])`

`ParameterError`

`Path(*args, **kwargs)` PurePath subclass that can make system calls.

`PhysParJSONEncoder(*[, skipkeys, ...])`

`PhysicalParameter([name, fiducial, ...])`

`PhysicalParametersCollection([...])`

ParameterError

exception `seyfert.cosmology.parameter.ParameterError`

PhysicalParameter

```
class seyfert.cosmology.parameter.PhysicalParameter(name=None, fiducial=None,
                                                    current_value=None, kind=None, probe=None,
                                                    is_free_parameter=None, stem_factor=1.0,
                                                    description="", units=None,
                                                    derivative_method='SteM')
```

Bases: object

Attributes Summary

COSMO_PAR_STRING

NUISANCE_PAR_STRING

is_cosmological

rtype bool

is_galaxy_bias_parameter

rtype bool

is_nuisance

rtype bool

Methods Summary

computeSteMValues(stem_eps_arr)

rtype ndarray

computeValueForDisplacement(eps)

rtype float

createCosmologicalParameter(**kwargs)

rtype *PhysicalParameter*

createNuisanceParameter(**kwargs)

rtype *PhysicalParameter*

fromJSON(json_file)

rtype *PhysicalParameter*

from_dict(data)

rtype *PhysicalParameter*

resetCurrentValueToFiducial()

continues on next page

Table 7 – continued from previous page

<code>to_JSON()</code>	rtype str
<code>to_dict()</code>	rtype Dict[str, Union[str, float, bool]]
<code>updateValueForDisplacement(eps)</code>	

Attributes Documentation

`COSMO_PAR_STRING` = 'CosmologicalParameter'

`NUISANCE_PAR_STRING` = 'NuisanceParameter'

`is_cosmological`

Return type bool

`is_galaxy_bias_parameter`

Return type bool

`is_nuisance`

Return type bool

Methods Documentation

`computeSteMValues(stem_eps_arr)`

Return type ndarray

`computeValueForDisplacement(eps)`

Return type float

`classmethod createCosmologicalParameter(**kwargs)`

Return type *PhysicalParameter*

`classmethod createNuisanceParameter(**kwargs)`

Return type *PhysicalParameter*

`classmethod fromJSON(json_file)`

Return type *PhysicalParameter*

`classmethod from_dict(data)`

Return type *PhysicalParameter*

`resetCurrentValueToFiducial()`

`to_JSON()`

Return type `str`

`to_dict()`

Return type `Dict[str, Union[str, float, bool]]`

`updateValueForDisplacement(eps)`

PhysicalParametersCollection

```
class seyfert.cosmology.parameter.PhysicalParametersCollection(base_stem_disps=None,
                                                              is_universe_flat=None,
                                                              **kwargs)

Bases: seyfert.base_structs.generic_dict.GenericDictInterface[str, seyfert.cosmology.
parameter.PhysicalParameter]
```

Attributes Summary

<code>cosmo_pars_current_values</code>	rtype <code>Dict[str, float]</code>
<code>cosmo_pars_fiducials</code>	rtype <code>Dict[str, float]</code>
<code>cosmological_parameters</code>	rtype <code>Dict[str, PhysicalParameter]</code>
<code>free_cosmo_pars_fiducials</code>	rtype <code>Dict[str, float]</code>
<code>free_cosmological_parameters</code>	rtype <code>Dict[str, PhysicalParameter]</code>
<code>free_physical_parameters</code>	rtype <code>Dict[str, PhysicalParameter]</code>
<code>nuisance_parameters</code>	rtype <code>Dict[str, PhysicalParameter]</code>
<code>params</code>	rtype <code>Dict[str, PhysicalParameter]</code>

Methods Summary

<i>computePhysParSTEMValues</i> ([dvar])	rtype ndarray
<i>fromJSON</i> (file[, only_cosmological])	rtype <i>PhysicalParametersCollection</i>
<i>from_dict_list</i> (dict_list[, only_cosmological])	rtype <i>PhysicalParametersCollection</i>
<i>getFreeNuisanceParametersForProbe</i> (probe)	rtype Dict[str, <i>PhysicalParameter</i>]
<i>getNuisanceParametersForProbe</i> (probe)	rtype Dict[str, <i>PhysicalParameter</i>]
<i>getParamsDictFromDictList</i> (dict_list[, ...])	rtype Dict[str, <i>PhysicalParameter</i>]
<i>loadStemDisplacements</i> (base_stem_disp)	rtype None
<i>readJSON</i> (file[, only_cosmological])	rtype None
<i>resetPhysicalParametersToFiducial</i> ()	rtype None
<i>updatePhysicalParametersForDvarStep</i> ([dvar, step])	rtype None
<i>writeJSON</i> (file)	rtype None

Attributes Documentation

cosmo_pars_current_values

Return type Dict[str, float]

cosmo_pars_fiducials

Return type Dict[str, float]

cosmological_parameters

Return type Dict[str, *PhysicalParameter*]

free_cosmo_pars_fiducials

Return type Dict[str, float]

free_cosmological_parameters

Return type Dict[str, *PhysicalParameter*]

free_physical_parameters

Return type Dict[str, *PhysicalParameter*]

nuisance_parameters

Return type Dict[str, *PhysicalParameter*]

params

Return type Dict[str, *PhysicalParameter*]

Methods Documentation

computePhysParSTEMValues(*dvar=None*)

Return type ndarray

classmethod fromJSON(*file, only_cosmological=False*)

Return type *PhysicalParametersCollection*

classmethod from_dict_list(*dict_list, only_cosmological=False*)

Return type *PhysicalParametersCollection*

getFreeNuisanceParametersForProbe(*probe*)

Return type Dict[str, *PhysicalParameter*]

getNuisanceParametersForProbe(*probe*)

Return type Dict[str, *PhysicalParameter*]

static getParamsDictFromDictList(*dict_list, only_cosmological=False*)

Return type Dict[str, *PhysicalParameter*]

loadStemDisplacements(*base_stem_disp*)

Return type None

readJSON(*file, only_cosmological=False*)

Return type None

resetPhysicalParametersToFiducial()

Return type None

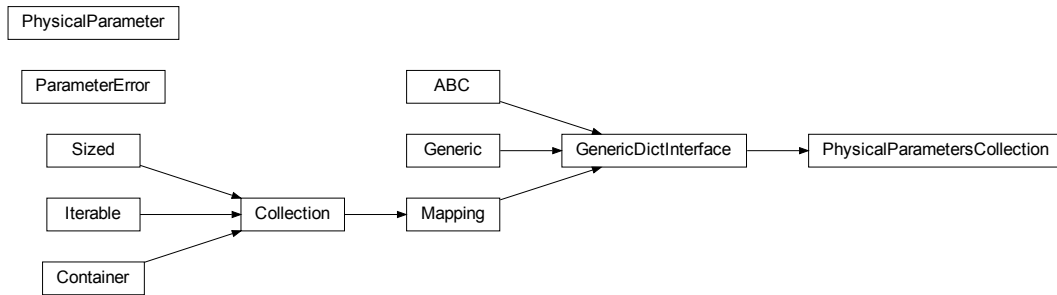
updatePhysicalParametersForDvarStep(*dvar=None, step=None*)

Return type None

`writeJSON(file)`

Return type None

1.2.1.2 Class Inheritance Diagram



1.3 Power Spectrum

1.3.1 seyfert.cosmology.power_spectrum Module

Module hosting PowerSpectrum class and H5PowerSpectrum, its HDF5 file-interface class.

1.3.1.1 Classes

<hr/> <code>AbstractH5FileIO([root])</code> <hr/>	
<hr/> <code>H5PowerSpectrum(**kwargs)</code> <hr/>	
<hr/> <code>Path(*args, **kwargs)</code>	PurePath subclass that can make system calls. <hr/>
<hr/> <code>PhysicalParameter([name, fiducial, ...])</code> <hr/>	
<hr/> <code>PowerSpectrum([power_spectrum_config, ...])</code>	Class to manage the matter power spectrum. It can be used to calculate the linear and non linear power spectra, <hr/>
<hr/> <code>PowerSpectrumConfig(**kwargs)</code> <hr/>	

H5PowerSpectrum

```
class seyfert.cosmology.power_spectrum.H5PowerSpectrum(**kwargs)
    Bases: seyfert.file_io.hdf5_io.AbstractH5FileIO
```

Methods Summary

```
writeObjectToFile(obj)
```

rtype None

```
writeToObject(obj)
```

rtype None

Methods Documentation

```
writeObjectToFile(obj)
```

Return type None

```
writeToObject(obj)
```

Return type None

PowerSpectrum

```
class seyfert.cosmology.power_spectrum.PowerSpectrum(power_spectrum_config=None,
                                                       cosmo_pars=None)
```

Bases: object

Class to manage the matter power spectrum. It can be used to calculate the linear and non linear power spectra, using a given boltzmann code (e.g. CAMB), or as a container storing all the relevant information about the matter power spectrum. In order to compute the power spectra an instance the class ExternalBoltzmannSolver is used. Methods for doing file I/O are present, and the file format employed is HDF5 (.h5), since it allows to store binary data with high compression options.

Parameters

- **k_grid** – the wavenumber grid over which the power spectra are evaluated
- **z_grid** – the redshift grid over which the power spectra are evaluated
- **lin_p_mm_z_k** – the linear matter power spectrum
- **nonlin_p_mm_z_k** – the nonlinear matter power spectrum

Attributes Summary

<i>growth_factor_z</i>	Returns the growth factor from the linear matter power spectrum as $D(z) = \text{Plin}(z, k_{\text{min}}) / \text{Plin}(0, k_{\text{min}})$.
------------------------	---

Methods Summary

<i>evaluateLinearAndNonLinearPowerSpectra</i> (workdir)	Method for calling the selected Boltzmann solver.
<i>fromHDF5</i> (file[, root])	rtype <i>PowerSpectrum</i>
<i>getResultsFromMatterPowerGenerator</i> ()	Method for read and store into attributes the Boltzmann solver results.
<i>loadFromHDF5</i> (file[, root])	rtype None
<i>saveToHDF5</i> (file[, root])	rtype None

Attributes Documentation

growth_factor_z

Returns the growth factor from the linear matter power spectrum as $D(z) = \text{Plin}(z, k_{\text{min}}) / \text{Plin}(0, k_{\text{min}})$. The scale dependency of the growth is not taken into account. :rtype: ndarray :return: np.ndarray

Methods Documentation

evaluateLinearAndNonLinearPowerSpectra(workdir)

Method for calling the selected Boltzmann solver.

Return type None

classmethod fromHDF5(file, root='')

Return type *PowerSpectrum*

getResultsFromMatterPowerGenerator()

Method for read and store into attributes the Boltzmann solver results.

Return type None

loadFromHDF5(file, root='')

Return type None

saveToHDF5(file, root='')

Return type None

1.3.1.2 Class Inheritance Diagram



1.4 Boltzmann solver

1.4.1 seyfert.cosmology.boltzmann_solver Module

1.4.1.1 Classes

<code>ABC()</code>	Helper class that provides a standard way to create an ABC using inheritance.
<code><i>CAMB</i>BoltzmannSolver(*args)</code>	
<code><i>CLASS</i>BoltzmannSolver(*args)</code>	
<code><i>External</i>BoltzmannSolver(workdir, config, ...)</code>	
<code>Path(*args, **kwargs)</code>	PurePath subclass that can make system calls.
<code>PhysicalParameter([name, fiducial, ...])</code>	
<code>PowerSpectrumConfig(**kwargs)</code>	

CAMBoltzmannSolver

```

class seyfert.cosmology.boltzmann_solver.CAMBoltzmannSolver(*args)
    Bases: seyfert.cosmology.boltzmann_solver.ExternalBoltzmannSolver

```

Methods Summary

<i>computeCAMBCosmologicalBasis()</i>	rtype Dict
<i>computeCAMBRedshiftGroupsNumber</i> (n_redshifts)	rtype int
<i>evaluateLinearAndNonLinearPowerSpectra()</i>	
<i>finetuneScalarAmpFromSigma8()</i>	rtype None
<i>getTransferFunctionFromResults</i> (camb_results)	rtype ndarray
<i>readCAMBIniFileToDict</i> (ini_file)	rtype Dict[str, str]
<i>run</i> ([get_growth, get_transfer_func])	
<i>writeDictToIniFile</i> (ini_dict, file)	rtype None
<i>writeUpdatedCAMBIniFile()</i>	

Methods Documentation

computeCAMBCosmologicalBasis()

Return type Dict

static **computeCAMBRedshiftGroupsNumber**(n_redshifts)

Return type int

evaluateLinearAndNonLinearPowerSpectra()

finetuneScalarAmpFromSigma8()

Return type None

static **getTransferFunctionFromResults**(camb_results)

Return type ndarray

static **readCAMBIniFileToDict**(ini_file)

Return type Dict[str, str]

run(get_growth=True, get_transfer_func=True)


```
static writeDictToIniFile(ini_dict, file)
```

Return type None

```
writeUpdatedCAMBIniFile()
```

CLASSBoltzmannSolver

```
class seyfert.cosmology.boltzmann_solver.CLASSBoltzmannSolver(*args)
    Bases: seyfert.cosmology.boltzmann_solver.ExternalBoltzmannSolver
```

Methods Summary

<code>computeCLASSCosmologicalBasis()</code>	rtype Dict
--	------------

<code>evaluateLinearAndNonLinearPowerSpectra()</code>

<code>run()</code>

Methods Documentation

```
computeCLASSCosmologicalBasis()
```

Return type Dict

```
evaluateLinearAndNonLinearPowerSpectra()
```

```
run()
```

ExternalBoltzmannSolver

```
class seyfert.cosmology.boltzmann_solver.ExternalBoltzmannSolver(workdir, config,
                                                                    cosmological_parameters)
    Bases: abc.ABC
```

Methods Summary

<code>evaluateLinearAndNonLinearPowerSpectra()</code>

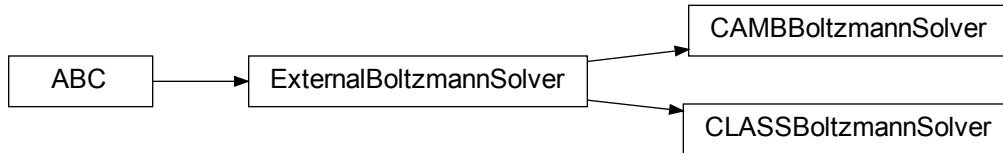
<code>run()</code>

Methods Documentation

abstract `evaluateLinearAndNonLinearPowerSpectra()`

abstract `run()`

1.4.1.2 Class Inheritance Diagram



1.5 Redshift Density

1.5.1 `seyfert.cosmology.redshift_density` Module

1.5.1.1 Functions

`compute_photoXspectro_shotnoise_ij(nph, nsp)`

rtype ndarray

`get_bins_overlap(z1_l, z1_r, z2_l, z2_r)`

rtype Tuple[float, float]

`jit([signature_or_function, locals, target, ...])`

This decorator is used to compile a Python function into native code.

`compute_photoXspectro_shotnoise_ij`

`seyfert.cosmology.redshift_density.compute_photoXspectro_shotnoise_ij(nph, nsp)`

Return type ndarray

get_bins_overlap

```
seyfert.cosmology.redshift_density.get_bins_overlap(zl_l, zl_r, z2_l, z2_r)
```

Return type Tuple[float, float]

1.5.1.2 Classes

`AbstractH5FileIO([root])`

DensityError

H5Density(**kwargs)

`Path(*args, **kwargs)`

PurePath subclass that can make system calls.

RedshiftDensity([probe])

`TypeError(msg[, loc, highlighting])`

A type inference failure.

DensityError

exception seyfert.cosmology.redshift_density.DensityError

H5Density

class seyfert.cosmology.redshift_density.H5Density(**kwargs)
 Bases: seyfert.file_io.hdf5_io.AbstractH5FileIO

Methods Summary

writeObjectToFile(obj)

rtype None

writeToObject(obj)

rtype None

Methods Documentation

`writeObjectToFile(obj)`

Return type None

`writeToObject(obj)`

Return type None

RedshiftDensity

`class seyfert.cosmology.redshift_density.RedshiftDensity(probe=None)`

Bases: object

Attributes Summary

<code>n_bins</code>	rtype int
<code>shot_noise</code>	rtype ndarray
<code>z_bin_centers</code>	rtype ndarray
<code>z_max</code>	rtype float
<code>z_min</code>	rtype float

Methods Summary

<code>computeBinNormFactor(bin_idx)</code>	rtype float
<code>computeInstrumentResponse(z_p, z)</code>	rtype Union[float, ndarray]
<code>computeNormalizedDensityAtBinAndRedshift(i, z)</code>	rtype Union[float, ndarray]
<code>computeSurfaceDensityAtBin(i)</code>	rtype float

continues on next page

Table 22 – continued from previous page

<i>computeTotalGalaxyNumber</i> ([integ_method])	rtype float
<i>convolvedNdzdOmegaWithInstrumentResponse</i> (z, i)	rtype Union[float, ndarray]
<i>evaluate</i> (z_grid)	rtype None
<i>evaluateBinNormFactors</i> ()	rtype None
<i>evaluateSurfaceDensity</i> ()	
<i>fromHDF5</i> (file[, root])	rtype <i>RedshiftDensity</i>
<i>interpolateInput</i> ()	rtype None
<i>loadFromHDF5</i> (file[, root])	rtype None
<i>modifiedGaussianResponse</i> (z_p, z[, z_mean, ...])	rtype Union[float, ndarray]
<i>saveToHDF5</i> (file[, root])	rtype None
<i>setUp</i> ()	rtype None

Attributes Documentation

n_bins

Return type int

shot_noise

Return type ndarray

z_bin_centers

Return type ndarray

z_max

Return type float

z_min

Return type float

Methods Documentation

computeBinNormFactor(*bin_idx*)

Return type float

computeInstrumentResponse(*z_p*, *z*)

Return type Union[float, ndarray]

computeNormalizedDensityAtBinAndRedshift(*i*, *z*)

Return type Union[float, ndarray]

computeSurfaceDensityAtBin(*i*)

Return type float

computeTotalGalaxyNumber(*integ_method*='simps')

Return type float

convolvedNdzdOmegaWithInstrumentResponse(*z*, *i*)

Return type Union[float, ndarray]

evaluate(*z_grid*)

Return type None

evaluateBinNormFactors()

Return type None

evaluateSurfaceDensity()

static fromHDF5(*file*, *root*='/')

Return type *RedshiftDensity*

interpolateInput()

Return type None

loadFromHDF5(*file*, *root*='/')

Return type None

static modifiedGaussianResponse(*z_p*, *z*, *z_mean*=None, *sigma*=None, *c*=None, *amplitude*=None)

Return type Union[float, ndarray]

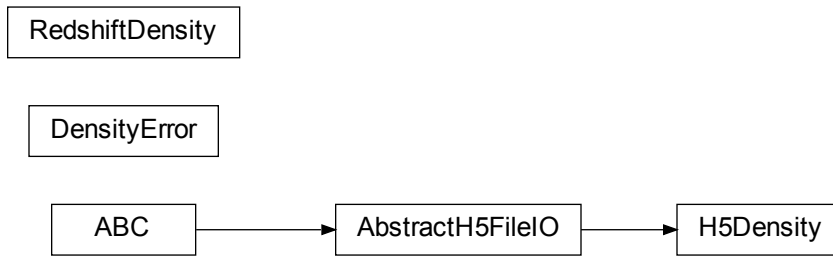
saveToHDF5(*file*, *root*='/')

Return type None

`setUp()`

Return type None

1.5.1.3 Class Inheritance Diagram



1.6 Bias

1.6.1 seyfert.cosmology.bias Module

Module hosting the bias class and the implemented bias models.

1.6.1.1 Classes

<code>ABC()</code>	Helper class that provides a standard way to create an ABC using inheritance.
<code>AbstractH5FileIO([root])</code>	
<code><i>Bias</i>([probe, z_bin_edges])</code>	Class managing the bias.
<code><i>BiasError</i></code>	
<code><i>BiasModel</i>([name, z_bin_edges, cosmology, ...])</code>	
<code><i>ConstantBias</i>(**kwargs)</code>	
<code><i>EuclidFlagshipGCphBias</i>(**kwargs)</code>	
<code><i>FiducialGrowthVoidBias</i>(**kwargs)</code>	
<code><i>H5Bias</i>(**kwargs)</code>	

continues on next page

Table 23 – continued from previous page

<code>Path(*args, **kwargs)</code>	PurePath subclass that can make system calls.
<code>PiecewiseBias(**kwargs)</code>	
<code>VdnVoidBias(**kwargs)</code>	

Bias

class seyfert.cosmology.bias.**Bias**(*probe=None, z_bin_edges=None*)

Bases: object

Class managing the bias.

Parameters

- **model** – instance of *BiasModel*. It defines the model with which to compute the bias as a function of the redshift.
- **nuisance_parameters** – a Dict[str, float] of nuisance parameters for the bias. It should contain the updated values of the nuisance parameters, since these may be let free to vary.
- **additional_parameters** – a Dict[str, float] of additional parameters for the bias. These are kept constant and not let free to vary as it happens for nuisance parameters

Attributes Summary

<code>has_output</code>	rtype bool
-------------------------	-------------------

Methods Summary

<code>evaluateBias(z_grid)</code>	rtype None
<code>fromHDF5(file[, root])</code>	rtype <i>Bias</i>
<code>initBiasModel(**kwargs)</code>	Method for initializing the bias model.
<code>loadFromHDF5(file[, root])</code>	rtype None
<code>saveToHDF5(file[, root])</code>	rtype None

Attributes Documentation

has_output

Return type bool

Methods Documentation

evaluateBias(*z_grid*)

Return type None

static fromHDF5(*file, root='/'*)

Return type *Bias*

initBiasModel(***kwargs*)

Method for initializing the bias model.

Parameters **kwargs** – generic keyword arguments parameters. These depend on the particular model that is being instantiated.

Return type None

loadFromHDF5(*file, root='/'*)

Return type None

saveToHDF5(*file, root='/'*)

Return type None

BiasError

exception seyfert.cosmology.bias.**BiasError**

BiasModel

class seyfert.cosmology.bias.**BiasModel**(*name=None, z_bin_edges=None, cosmology=None, nuisance_parameters=None, additional_parameters=None*)

Bases: abc.ABC

Attributes Summary

n_bins

rtype int

z_bin_centers

rtype ndarray

Methods Summary

computeBias(z_grid)**rtype** ndarray

Attributes Documentation

n_bins**Return type** int**z_bin_centers****Return type** ndarray

Methods Documentation

abstract *computeBias*(z_grid)**Return type** ndarray

ConstantBias

class seyfert.cosmology.bias.**ConstantBias**(**kwargs)Bases: *seyfert.cosmology.bias.BiasModel*

Methods Summary

computeBias(z_grid)**rtype** ndarray

Methods Documentation

computeBias(z_grid)**Return type** ndarray

EuclidFlagshipGCphBias

```
class seyfert.cosmology.bias.EuclidFlagshipGCphBias(**kwargs)
    Bases: seyfert.cosmology.bias.BiasModel
```

Attributes Summary

<i>Aph</i>	rtype float
<i>Bph</i>	rtype float
<i>Cph</i>	rtype float
<i>Dph</i>	rtype float

Methods Summary

<i>computeBias</i> (z_grid)	rtype ndarray
-----------------------------	----------------------

Attributes Documentation

Aph	Return type float
Bph	Return type float
Cph	Return type float
Dph	Return type float

Methods Documentation

`computeBias(z_grid)`

Return type ndarray

FiducialGrowthVoidBias

class seyfert.cosmology.bias.FiducialGrowthVoidBias(**kwargs)
Bases: [seyfert.cosmology.bias.BiasModel](#)

Methods Summary

<code>computeBias(z_grid)</code>	rtype ndarray
----------------------------------	----------------------

Methods Documentation

`computeBias(z_grid)`

Return type ndarray

H5Bias

class seyfert.cosmology.bias.H5Bias(**kwargs)
Bases: [seyfert.file_io.hdf5_io.AbstractH5FileIO](#)

Methods Summary

<code>writeObjectToFile(obj)</code>	rtype None
-------------------------------------	-------------------

<code>writeToObject(obj)</code>	rtype None
---------------------------------	-------------------

Methods Documentation

`writeObjectToFile(obj)`

Return type None

`writeToObject(obj)`

Return type None

PiecewiseBias

`class seyfert.cosmology.bias.PiecewiseBias(**kwargs)`
Bases: `seyfert.cosmology.bias.BiasModel`

Methods Summary

<code>computeBias(z_grid)</code>	rtype ndarray
<hr/>	
<code>getSortedBiasValues()</code>	rtype ndarray

Methods Documentation

`computeBias(z_grid)`

Return type ndarray

`getSortedBiasValues()`

Return type ndarray

VdnVoidBias

```
class seyfert.cosmology.bias.VdnVoidBias(**kwargs)
    Bases: seyfert.cosmology.bias.BiasModel
```

Attributes Summary

<i>R_max_Mpc</i>	rtype float
<hr/>	
<i>R_min_Mpc</i>	rtype float
<hr/>	
<i>delta_c0</i>	rtype float
<hr/>	
<i>delta_v0</i>	rtype float
<hr/>	
<i>n_R</i>	rtype int
<hr/>	
<i>z_grid</i>	

Methods Summary

<i>computeBias</i> (z_grid)	rtype ndarray
<hr/>	
<i>computeVdnSizeFunction</i> (R, D, x, delta_v_z, ...)	
<hr/>	
<i>computeVoidMultiplicity</i> (D, x, delta_v_z, sigmaR)	rtype ndarray
<hr/>	
<i>voidMultiplicity</i> (delta_v, D, sigma, x)	rtype float

Attributes Documentation

R_max_Mpc
Return type float

R_min_Mpc
Return type float

delta_c0

Return type float

`delta_v0`

Return type float

`n_R`

Return type int

`z_grid`

Methods Documentation

`computeBias(z_grid)`

Return type ndarray

`computeVdnSizeFunction(R, D, x, delta_v_z, sigmaR)`

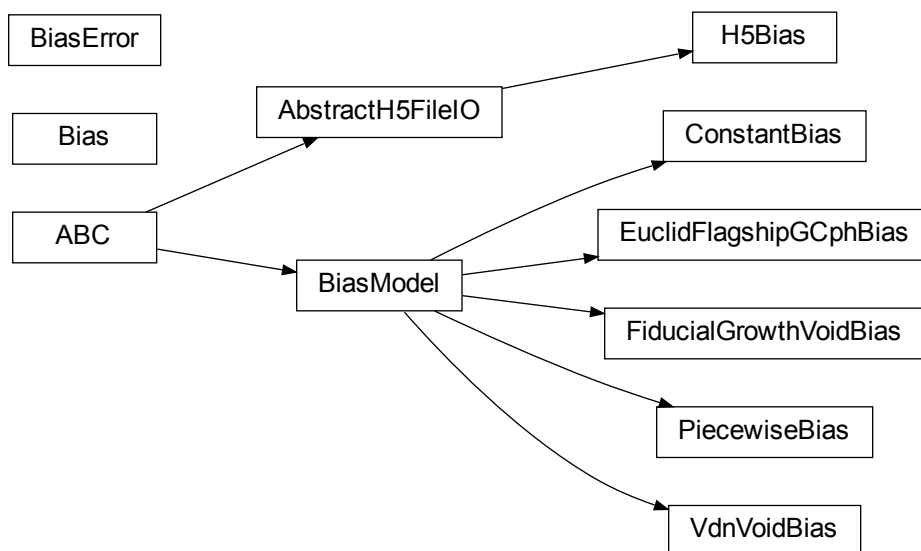
`computeVoidMultiplicity(D, x, delta_v_z, sigmaR)`

Return type ndarray

`static voidMultiplicity(delta_v, D, sigma, x)`

Return type float

1.6.1.2 Class Inheritance Diagram



1.7 Weight functions

1.7.1 seyfert.cosmology.weight_functions Module

1.7.1.1 Functions

<code>abstractmethod(funcobj)</code>	A decorator indicating abstract methods.
<code>join(a, *p)</code>	Join two or more pathname components, inserting ‘/’ as needed.
<code>probe_from_weight_function_cls_name(...)</code>	
<code>weight_cls_name_from_obs(obs_string)</code>	
<code>weight_function_for_probe(probe[, ...])</code>	
	rtype <i>WeightFunction</i>

`probe_from_weight_function_cls_name`

`seyfert.cosmology.weight_functions.probe_from_weight_function_cls_name(weight_function_name)`

`weight_cls_name_from_obs`

`seyfert.cosmology.weight_functions.weight_cls_name_from_obs(obs_string)`

`weight_function_for_probe`

`seyfert.cosmology.weight_functions.weight_function_for_probe(probe, probe_config=None, nuisance_params=None, cosmology=None, fiducial_cosmology=None, angular_config=None)`

Return type *WeightFunction*

1.7.1.2 Classes

<code>ABC()</code>	Helper class that provides a standard way to create an ABC using inheritance.
<code>AbstractH5FileIO([root])</code>	
<code>AngularConfig(**kwargs)</code>	
<code>Bias([probe, z_bin_edges])</code>	Class managing the bias.
<code>Cosmology([params, flat, model_name, ...])</code>	Class for managing the cosmology.

continues on next page

Table 37 – continued from previous page

<i>GalaxyClusteringWeightFunction</i> (**kwargs)	
<i>H5WeightFunction</i> (**kwargs)	
<i>LensingWeightFunction</i> (**kwargs)	
<i>Path</i> (*args, **kwargs)	PurePath subclass that can make system calls.
<i>PhotometricGalaxyWeightFunction</i> (**kwargs)	
<i>PhysicalParameter</i> ([name, fiducial, ...])	
<i>ProbeConfig</i> (name[, input_data_dir, fc_xml_io])	
<i>RedshiftDensity</i> ([probe])	
<i>SpectroscopicGalaxyWeightFunction</i> (**kwargs)	
<i>VoidWeightFunction</i> (**kwargs)	
<i>WeightFunction</i> ([probe_config, ...])	
<i>WeightFunctionWithBias</i> (**kwargs)	

GalaxyClusteringWeightFunction

class seyfert.cosmology.weight_functions.**GalaxyClusteringWeightFunction**(**kwargs)
 Bases: *seyfert.cosmology.weight_functions.WeightFunctionWithBias*, abc.ABC

Methods Summary

<i>setupBias</i> ()	rtype None
---------------------	-------------------

Methods Documentation

setupBias()

Return type None

H5WeightFunction

class seyfert.cosmology.weight_functions.H5WeightFunction(**kwargs)
Bases: seyfert.file_io.hdf5_io.AbstractH5FileIO

Methods Summary

<i>readBuildingData()</i>	rtype None
<hr/>	
<i>writeObjectToFile(obj)</i>	rtype None
<hr/>	
<i>writeToObject(obj)</i>	rtype None

Methods Documentation

readBuildingData()

Return type None

writeObjectToFile(obj)

Return type None

writeToObject(obj)

Return type None

LensingWeightFunction

class seyfert.cosmology.weight_functions.LensingWeightFunction(**kwargs)
Bases: *seyfert.cosmology.weight_functions.WeightFunction*

Attributes Summary

<i>H0</i>	rtype float
<hr/>	
<i>Om</i>	rtype float
<hr/>	
<i>c_km_s</i>	rtype float

Methods Summary

<i>computeIntrinsicAlignmentContribution()</i>	rtype ndarray
<i>computeLensingEfficiency(z)</i>	rtype ndarray
<i>computeLensingEfficiencyAtBin(i, z)</i>	rtype Union[float, ndarray]
<i>evaluateOverRedshiftGrid(z_grid)</i>	rtype None

Attributes Documentation

H0

Return type float

Omm

Return type float

c_km_s

Return type float

Methods Documentation

computeIntrinsicAlignmentContribution()

Return type ndarray

computeLensingEfficiency(z)

Return type ndarray

computeLensingEfficiencyAtBin(i, z)

Return type Union[float, ndarray]

evaluateOverRedshiftGrid(z_grid)

Return type None

PhotometricGalaxyWeightFunction

class seyfert.cosmology.weight_functions.**PhotometricGalaxyWeightFunction**(**kwargs)
Bases: *seyfert.cosmology.weight_functions.GalaxyClusteringWeightFunction*

Methods Summary

*setupBias()***rtype** None

Methods Documentation

setupBias()**Return type** None

SpectroscopicGalaxyWeightFunction

class seyfert.cosmology.weight_functions.**SpectroscopicGalaxyWeightFunction**(**kwargs)
Bases: *seyfert.cosmology.weight_functions.GalaxyClusteringWeightFunction*

Methods Summary

*setupBias()***rtype** None

Methods Documentation

setupBias()**Return type** None

VoidWeightFunction

class seyfert.cosmology.weight_functions.VoidWeightFunction(**kwargs)
 Bases: *seyfert.cosmology.weight_functions.WeightFunctionWithBias*

Methods Summary

setupBias()

rtype None

Methods Documentation

setupBias()

Return type None

WeightFunction

class seyfert.cosmology.weight_functions.WeightFunction(*probe_config=None, nuisance_params=None, cosmology=None, fiducial_cosmology=None, angular_config=None*)
 Bases: abc.ABC

Attributes Summary

H_z

rtype ndarray

is_evaluated

rtype bool

n_bins

n_i_z

rtype ndarray

probe

r_tilde_z

rtype ndarray

z_bin_centers

z_bin_edges

continues on next page

Table 45 – continued from previous page

<i>z_max</i>	
<i>z_min</i>	
Methods Summary	
<i>evaluateOverRedshiftGrid</i> (z_grid)	
<i>fromHDF5</i> (file[, root])	rtype <i>WeightFunction</i>
<i>loadFromHDF5</i> (file[, root])	rtype None
<i>saveToHDF5</i> (file[, root])	rtype None
<i>setUp</i> ()	

Attributes Documentation**H_z****Return type** ndarray**is_evaluated****Return type** bool**n_bins****n_i_z****Return type** ndarray**probe****r_tilde_z****Return type** ndarray**z_bin_centers****z_bin_edges****z_max****z_min**

Methods Documentation

abstract `evaluateOverRedshiftGrid(z_grid)`

classmethod `fromHDF5(file, root='/')`

Return type *WeightFunction*

loadFromHDF5(file, root='/')

Return type `None`

saveToHDF5(file, root='/')

Return type `None`

setUp()

WeightFunctionWithBias

class `seyfert.cosmology.weight_functions.WeightFunctionWithBias(**kwargs)`

Bases: *seyfert.cosmology.weight_functions.WeightFunction*, *abc.ABC*

Methods Summary

evaluateOverRedshiftGrid(z_grid)

rtype `None`

setUp()

setupBias()

rtype `None`

Methods Documentation

evaluateOverRedshiftGrid(z_grid)

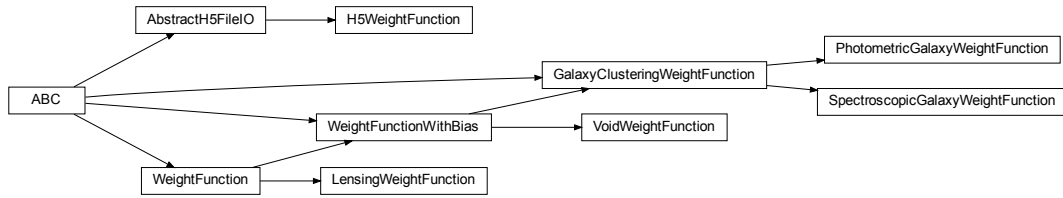
Return type `None`

setUp()

abstract `setupBias()`

Return type `None`

1.7.1.3 Class Inheritance Diagram



1.8 Angular power spectra

1.8.1 `seyfert.cosmology.c_ells` Module

Module hosting the classes for computing and storing the angular power spectra.

1.8.1.1 Functions

<code>cl_key_long_to_short(cl_long_key)</code>	rtype str
--	------------------

<code>cl_key_short_to_long(cl_short_key)</code>	rtype str
---	------------------

<code>grid_plot(x, y[, label, axes, idx_fontsize, ...])</code>	
--	--

<code>join(a, *p)</code>	Join two or more pathname components, inserting <code>'/'</code> as needed.
--------------------------	---

`cl_key_long_to_short`

`seyfert.cosmology.c_ells.cl_key_long_to_short(cl_long_key)`

Return type str

cl_key_short_to_long

seyfert.cosmology.c_ells.cl_key_short_to_long(*cl_short_key*)

Return type `str`

1.8.1.2 Classes

<code>AbstractH5FileIO([root])</code>	
<code>AngularCoefficient([probe1, probe2, kernel, ...])</code>	Class representing a particular kind of angular power spectrum (i.e.
<code>AngularCoefficientsCollector([phys_params, ...])</code>	
<code>AngularConfig(**kwargs)</code>	
<code>ClError</code>	
<code>CompositeNewtonCotesIntegrator(x, y, order)</code>	
<code>Cosmology([params, flat, model_name, ...])</code>	Class for managing the cosmology.
<code>ForecastConfig([input_xml, input_data_dir])</code>	
<code>H5Cl([probe1, probe2])</code>	
<code>H5ClCollection(**kwargs)</code>	
<code>Path(*args, **kwargs)</code>	PurePath subclass that can make system calls.
<code>PhysicalParametersCollection([...])</code>	
<code>RedshiftDensity([probe])</code>	

AngularCoefficient

```
class seyfert.cosmology.c_ells.AngularCoefficient(probe1=None, probe2=None, kernel=None,
                                                forecast_config=None, angular_config=None,
                                                full_output=False)
```

Bases: `object`

Class representing a particular kind of angular power spectrum (i.e. auto-correlation or cross-correlation. These are computed in the Limber approximation, meaning that a single redshift integral is performed. Attributes

Parameters

- **c_lij** – np 3D array (l, i, j) storing the values of the Cl tomographic matrix. Here l is the multipole
- **l_bin_centers** – the multipole bin centers where to compute the Cl, np 1D array.
- **l_bin_widths** – the multipole bin widths, np 1D array.

- **limber_power_spectrum_l_z** – np 2D array (l, z) storing the matter power spectrum evaluated in the Limber approximation, i.e. at wave-numbers $k = \frac{l+1/2}{r(z)}$. The redshift dimension is define by the redshift grid
- **cosmology** – instance of Cosmology class storing the current cosmology.
- **angular_config** (Optional[AngularConfig]) – configuration for the Cl computation, instance of AngularConfig.
- **forecast_config** (Optional[ForecastConfig]) – forecast configuration, instance of ForecastConfig.
- **kernel** (Optional[KernelFunction]) – kernel function to be integrated against the Limber power spectrum, instance of KernelFunction.

Attributes Summary

<i>integ_method</i>	rtype str
<i>is_auto_correlation</i>	rtype bool
<i>n_ell</i>	rtype int
<i>n_i</i>	rtype int
<i>n_j</i>	rtype int
<i>n_z</i>	rtype int
<i>obs_key</i>	rtype str
<i>power_spectrum</i>	Reference to the power spectrum possessed by the cosmology attribute.
<i>weight1</i>	Reference to first weight function making up the kernel
<i>weight2</i>	Reference to second weight function making up the kernel
<i>z_array</i>	Reference to the <i>z_grid</i> attribute of the possessed <i>cosmology</i> instance.

Methods Summary

<i>applyVoidsCutToLimberPowerSpectrum()</i>	rtype None
<i>computeClIntegral</i> (integrand, axis)	Method for computing a single Cl_{ij} .
<i>evaluateAngularCorrelation</i> ()	Method for evaluating and storing the Cl.
<i>evaluateLimberApproximatedPowerSpectrum()</i>	rtype None
<i>fromHDF5</i> (file, probe1, probe2[, root])	rtype <i>AngularCoefficient</i>
<i>getLimberRedshiftIntegrand</i> (i, j)	rtype ndarray
<i>getWeightFunction</i> (probe)	rtype <i>WeightFunction</i>
<i>loadCosmology</i> (pmm_file[, load_power_spectrum])	rtype None
<i>loadFromHDF5</i> (file[, probe1, probe2, root])	rtype None
<i>plot</i> ([what, axes])	
<i>saveToHDF5</i> (file[, root])	rtype None
<i>setUp</i> ()	rtype None

Attributes Documentation

integ_method

Return type str

is_auto_correlation

Return type bool

n_ell

Return type int

n_i

Return type int

n_j

Return type int

n_z

Return type `int`

obs_key

Return type `str`

power_spectrum

Reference to the power spectrum possessed by the cosmology attribute.

Return type *PowerSpectrum*

weight1

Reference to first weight function making up the kernel

Return type *WeightFunction*

weight2

Reference to second weight function making up the kernel

Return type *WeightFunction*

z_array: `numpy.ndarray`

Reference to the `z_grid` attribute of the possessed *cosmology* instance.

Return type `ndarray`

Methods Documentation

`applyVoidsCutToLimberPowerSpectrum()`

Return type `None`

`computeClIntegral(integrand, axis)`

Method for computing a single `Cl_ij`.

Return type `ndarray`

`evaluateAngularCorrelation()`

Method for evaluating and storing the `Cl`.

Return type `None`

`evaluateLimberApproximatedPowerSpectrum()`

Return type `None`

`classmethod fromHDF5(file, probe1, probe2, root='')`

Return type *AngularCoefficient*

`getLimberRedshiftIntegrand(i, j)`

Return type `ndarray`

`getWeightFunction(probe)`

Return type *WeightFunction*

`loadCosmology(pmm_file, load_power_spectrum=True)`

Return type None

`loadFromHDF5(file, probe1=None, probe2=None, root='/')`

Return type None

`plot(what='cl', axes=None, **kwargs)`

`saveToHDF5(file, root='/')`

Return type None

`setUp()`

Return type None

AngularCoefficientsCollector

```
class seyfert.cosmology.c_ells.AngularCoefficientsCollector(phys_params=None,
                                                            cosmology=None,
                                                            fiducial_cosmology=None,
                                                            forecast_config=None,
                                                            angular_config=None,
                                                            full_output=False, niz_file=None)
```

Bases: object

Attributes Summary

<i>probes</i>	rtype List[str]
<i>probes_combinations</i>	rtype List[Tuple[str]]
<i>short_keys</i>	rtype List[str]

Methods Summary

<i>evaluateAngularCoefficients()</i>	rtype None
<i>fromHDF5(file[, root])</i>	rtype <i>AngularCoefficientsCollector</i>

continues on next page

Table 53 – continued from previous page

<code>loadFromHDF5(file[, root])</code>	rtype None
<code>saveToHDF5(file[, root])</code>	rtype None
<code>setUp([densities])</code>	

Attributes Documentation

probes

Return type List[str]

probes_combinations

Return type List[Tuple[str]]

short_keys

Return type List[str]

Methods Documentation

evaluateAngularCoefficients()

Return type None

classmethod fromHDF5(file, root='')

Return type *AngularCoefficientsCollector*

loadFromHDF5(file, root='')

Return type None

saveToHDF5(file, root='')

Return type None

setUp(densities=None)

CLError

exception seyfert.cosmology.c_ells.CLError

H5CI

```
class seyfert.cosmology.c_ells.H5CI(probe1=None, probe2=None, **kwargs)
    Bases: seyfert.file_io.hdf5_io.AbstractH5FileIO
```

Attributes Summary

absolute_main_path

rtype str

main_path_rel_to_root

rtype str

Methods Summary

readBuildingData()

rtype None

writeObjectToFile(obj)

rtype None

writeToObject(obj)

rtype None

Attributes Documentation

absolute_main_path

Return type str

main_path_rel_to_root

Return type str

Methods Documentation

readBuildingData()

Return type None

writeObjectToFile(obj)

Return type None

writeToObject(obj)

Return type None

H5ClCollection

```
class seyfert.cosmology.c_ells.H5ClCollection(**kwargs)
    Bases: seyfert.file_io.hdf5_io.AbstractH5FileIO
```

Methods Summary

writeObjectToFile(obj)

rtype None

writeToObject(obj)

rtype None

Methods Documentation

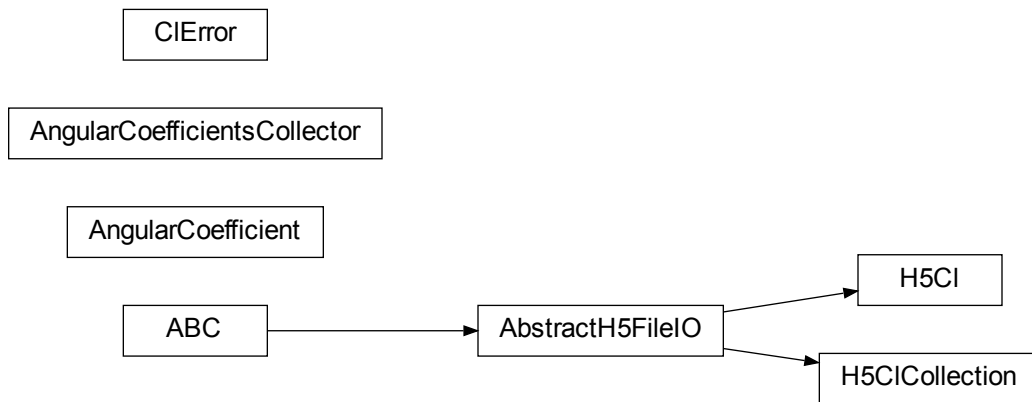
writeObjectToFile(*obj*)

Return type None

writeToObject(*obj*)

Return type None

1.8.1.3 Class Inheritance Diagram



Indices and tables

- `genindex`
- `modindex`
- `search`

S

- `seyfert.cosmology.bias`, [27](#)
- `seyfert.cosmology.boltzmann_solver`, [19](#)
- `seyfert.cosmology.c_ells`, [44](#)
- `seyfert.cosmology.cosmology`, [3](#)
- `seyfert.cosmology.parameter`, [10](#)
- `seyfert.cosmology.power_spectrum`, [16](#)
- `seyfert.cosmology.redshift_density`, [22](#)
- `seyfert.cosmology.weight_functions`, [36](#)

A

`absolute_main_path` (*seyfert.cosmology.c_ells.HSCL attribute*), 51
`AngularCoefficient` (*class in seyfert.cosmology.c_ells*), 45
`AngularCoefficientsCollector` (*class in seyfert.cosmology.c_ells*), 49
`Aph` (*seyfert.cosmology.bias.EuclidFlagshipGCphBias attribute*), 31
`applyVoidsCutToLimberPowerSpectrum()` (*seyfert.cosmology.c_ells.AngularCoefficient method*), 48

B

`Bias` (*class in seyfert.cosmology.bias*), 28
`BiasError`, 29
`BiasModel` (*class in seyfert.cosmology.bias*), 29
`Bph` (*seyfert.cosmology.bias.EuclidFlagshipGCphBias attribute*), 31

C

`c_km_s` (*seyfert.cosmology.weight_functions.LensingWeightFunction attribute*), 39
`CAMBoltzmannSolver` (*class in seyfert.cosmology.boltzmann_solver*), 19
`checkParameters()` (*seyfert.cosmology.cosmology.Cosmology method*), 7
`cl_key_long_to_short()` (*in module seyfert.cosmology.c_ells*), 44
`cl_key_short_to_long()` (*in module seyfert.cosmology.c_ells*), 45
`CLASSBoltzmannSolver` (*class in seyfert.cosmology.boltzmann_solver*), 21
`ClError`, 50
`compute_photoXspectro_shotnoise_ij()` (*in module seyfert.cosmology.redshift_density*), 22
`computeBias()` (*seyfert.cosmology.bias.BiasModel method*), 30
`computeBias()` (*seyfert.cosmology.bias.ConstantBias method*), 30
`computeBias()` (*seyfert.cosmology.bias.EuclidFlagshipGCphBias method*), 32
`computeBias()` (*seyfert.cosmology.bias.FiducialGrowthVoidBias method*), 32
`computeBias()` (*seyfert.cosmology.bias.PiecewiseBias method*), 33
`computeBias()` (*seyfert.cosmology.bias.VdnVoidBias method*), 35
`computeBinNormFactor()` (*seyfert.cosmology.redshift_density.RedshiftDensity method*), 26
`computeCAMBCosmologicalBasis()` (*seyfert.cosmology.boltzmann_solver.CAMBoltzmannSolver method*), 20
`computeCAMBRedshiftGroupsNumber()` (*seyfert.cosmology.boltzmann_solver.CAMBoltzmannSolver static method*), 20

`computeCLASSCosmologicalBasis()` (*seyfert.cosmology.boltzmann_solver.CLASSBoltzmannSolver method*), 21
`computeClIntegral()` (*seyfert.cosmology.c_ells.AngularCoefficient method*), 48
`computeComovingDistance()` (*seyfert.cosmology.cosmology.Cosmology method*), 7
`computeDimensionlessComovingDistance()` (*seyfert.cosmology.cosmology.Cosmology method*), 7
`computeDimensionlessHubbleParameter()` (*seyfert.cosmology.cosmology.Cosmology method*), 7
`computeHubbleParameter()` (*seyfert.cosmology.cosmology.Cosmology method*), 7
`computeInstrumentResponse()` (*seyfert.cosmology.redshift_density.RedshiftDensity method*), 26
`computeIntrinsicAlignmentContribution()` (*seyfert.cosmology.weight_functions.LensingWeightFunction method*), 39
`computeLensingEfficiency()` (*seyfert.cosmology.weight_functions.LensingWeightFunction method*), 39
`computeLensingEfficiencyAtBin()` (*seyfert.cosmology.weight_functions.LensingWeightFunction method*), 39
`computeNormalizedDensityAtBinAndRedshift()` (*seyfert.cosmology.redshift_density.RedshiftDensity method*), 26
`computePhysParSTEMValues()` (*seyfert.cosmology.parameter.PhysicalParametersCollection method*), 15
`computeReciprocalDimensionlessHubbleParameter()` (*seyfert.cosmology.cosmology.Cosmology method*), 7
`computeSigmaR()` (*seyfert.cosmology.cosmology.Cosmology method*), 8
`computeSTEMValues()` (*seyfert.cosmology.parameter.PhysicalParameter method*), 12
`computeSurfaceDensityAtBin()` (*seyfert.cosmology.redshift_density.RedshiftDensity method*), 26
`computeTotalGalaxyNumber()` (*seyfert.cosmology.redshift_density.RedshiftDensity method*), 26
`computeValueForDisplacement()` (*seyfert.cosmology.parameter.PhysicalParameter method*), 12

computeVdnSizeFunction() (*seyfert.cosmology.bias.VdnVoidBias* method), 35

computeVoidMultiplicity() (*seyfert.cosmology.bias.VdnVoidBias* method), 35

ConstantBias (*class in seyfert.cosmology.bias*), 30

convolvedNdzdOmegaWithInstrumentResponse() (*seyfert.cosmology.redshift_density.RedshiftDensity* method), 26

COSMO_PAR_STRING (*seyfert.cosmology.parameter.PhysicalParameter* attribute), 12

cosmo_pars_current (*seyfert.cosmology.cosmology.Cosmology* attribute), 6

cosmo_pars_current_values (*seyfert.cosmology.parameter.PhysicalParametersCollection* attribute), 14

cosmo_pars_fiducials (*seyfert.cosmology.parameter.PhysicalParametersCollection* attribute), 14

cosmological_parameters (*seyfert.cosmology.parameter.PhysicalParametersCollection* attribute), 14

Cosmology (*class in seyfert.cosmology.cosmology*), 4

CosmologyError, 9

Cph (*seyfert.cosmology.bias.EuclidFlagshipGCphBias* attribute), 31

createCosmologicalParameter() (*seyfert.cosmology.parameter.PhysicalParameter* class method), 12

createNuisanceParameter() (*seyfert.cosmology.parameter.PhysicalParameter* class method), 12

D

delta_c0 (*seyfert.cosmology.bias.VdnVoidBias* attribute), 34

delta_v0 (*seyfert.cosmology.bias.VdnVoidBias* attribute), 35

DensityError, 23

Dph (*seyfert.cosmology.bias.EuclidFlagshipGCphBias* attribute), 31

E

E_z (*seyfert.cosmology.cosmology.Cosmology* attribute), 6

EuclidFlagshipGCphBias (*class in seyfert.cosmology.bias*), 31

evaluate() (*seyfert.cosmology.redshift_density.RedshiftDensity* method), 26

evaluateAngularCoefficients() (*seyfert.cosmology.c_ells.AngularCoefficientsCollector* method), 50

evaluateAngularCorrelation() (*seyfert.cosmology.c_ells.AngularCoefficient* method), 48

evaluateBias() (*seyfert.cosmology.bias.Bias* method), 29

evaluateBinNormFactors() (*seyfert.cosmology.redshift_density.RedshiftDensity* method), 26

evaluateLamberApproximatedPowerSpectrum() (*seyfert.cosmology.c_ells.AngularCoefficient* method), 48

evaluateLinearAndNonLinearPowerSpectra() (*seyfert.cosmology.boltzmann_solver.CAMBoltzmannSolver* method), 20

evaluateLinearAndNonLinearPowerSpectra() (*seyfert.cosmology.boltzmann_solver.CLASSBoltzmannSolver* method), 21

evaluateLinearAndNonLinearPowerSpectra() (*seyfert.cosmology.boltzmann_solver.ExternalBoltzmannSolver* method), 22

evaluateLinearAndNonLinearPowerSpectra() (*seyfert.cosmology.power_spectrum.PowerSpectrum* method), 18

evaluateOverRedshiftGrid() (*seyfert.cosmology.cosmology.Cosmology* method), 8

evaluateOverRedshiftGrid() (*seyfert.cosmology.weight_functions.LensingWeightFunction* method), 39

evaluateOverRedshiftGrid() (*seyfert.cosmology.weight_functions.WeightFunction* method), 43

evaluateOverRedshiftGrid() (*seyfert.cosmology.weight_functions.WeightFunctionWithBias* method), 43

evaluatePowerSpectrum() (*seyfert.cosmology.cosmology.Cosmology* method), 8

evaluateSurfaceDensity() (*seyfert.cosmology.redshift_density.RedshiftDensity* method), 26

ExternalBoltzmannSolver (*class in seyfert.cosmology.boltzmann_solver*), 21

F

FiducialGrowthVoidBias (*class in seyfert.cosmology.bias*), 32

finetuneScalarAmpFromSigma8() (*seyfert.cosmology.boltzmann_solver.CAMBoltzmannSolver* method), 20

free_cosmo_pars_fiducials (*seyfert.cosmology.parameter.PhysicalParametersCollection* attribute), 14

free_cosmological_parameters (*seyfert.cosmology.parameter.PhysicalParametersCollection* attribute), 14

free_physical_parameters (*seyfert.cosmology.parameter.PhysicalParametersCollection* attribute), 15

from_dict() (*seyfert.cosmology.parameter.PhysicalParameter* class method), 12

from_dict_list() (*seyfert.cosmology.parameter.PhysicalParametersCollection* class method), 15

fromHDF5() (*seyfert.cosmology.bias.Bias* static method), 29

fromHDF5() (*seyfert.cosmology.c_ells.AngularCoefficient* class method), 48

fromHDF5() (*seyfert.cosmology.c_ells.AngularCoefficientsCollector* class method), 50

fromHDF5() (*seyfert.cosmology.cosmology.Cosmology* class method), 8

fromHDF5() (*seyfert.cosmology.power_spectrum.PowerSpectrum* class method), 18

fromHDF5() (*seyfert.cosmology.redshift_density.RedshiftDensity* static method), 26

fromHDF5() (*seyfert.cosmology.weight_functions.WeightFunction* class method), 43

fromJSON() (*seyfert.cosmology.parameter.PhysicalParameter* class method), 12

fromJSON() (*seyfert.cosmology.parameter.PhysicalParametersCollection* class method), 15

G

GalaxyClusteringWeightFunction (*class in seyfert.cosmology.weight_functions*), 37

get_bins_overlap() (*in module seyfert.cosmology.redshift_density*), 23

getFreeNuisanceParametersForProbe() (*seyfert.cosmology.parameter.PhysicalParametersCollection* method), 15

getLamberRedshiftIntegrand() (*seyfert.cosmology.c_ells.AngularCoefficient* method), 48

getNuisanceParametersForProbe() (seyfert.cosmology.parameter.PhysicalParametersCollection method), 15
 getParamsDictFromDictList() (seyfert.cosmology.parameter.PhysicalParametersCollection static method), 15
 getResultsFromMatterPowerGenerator() (seyfert.cosmology.power_spectrum.PowerSpectrum method), 18
 getSortedBiasValues() (seyfert.cosmology.bias.PiecewiseBias method), 33
 getTransferFunctionFromResults() (seyfert.cosmology.boltzmann_solver.CAMB Boltzmann Solver static method), 20
 getWeightFunction() (seyfert.cosmology.c_ells.AngularCoefficient method), 48
 growth_factor_z (seyfert.cosmology.cosmology.Cosmology attribute), 6
 growth_factor_z (seyfert.cosmology.power_spectrum.PowerSpectrum attribute), 18

H

h (seyfert.cosmology.cosmology.Cosmology attribute), 6
 H0 (seyfert.cosmology.cosmology.Cosmology attribute), 6
 H0 (seyfert.cosmology.weight_functions.LensingWeightFunction attribute), 39
 H5Bias (class in seyfert.cosmology.bias), 32
 H5Cl (class in seyfert.cosmology.c_ells), 51
 H5ClCollection (class in seyfert.cosmology.c_ells), 52
 H5Cosmology (class in seyfert.cosmology.cosmology), 9
 H5Density (class in seyfert.cosmology.redshift_density), 23
 H5PowerSpectrum (class in seyfert.cosmology.power_spectrum), 17
 H5WeightFunction (class in seyfert.cosmology.weight_functions), 38
 H_z (seyfert.cosmology.cosmology.Cosmology attribute), 6
 H_z (seyfert.cosmology.weight_functions.WeightFunction attribute), 42
 has_output (seyfert.cosmology.bias.Bias attribute), 29

I

initBiasModel() (seyfert.cosmology.bias.Bias method), 29
 integ_method (seyfert.cosmology.c_ells.AngularCoefficient attribute), 47
 interpolateInput() (seyfert.cosmology.redshift_density.RedshiftDensity method), 26
 is_auto_correlation (seyfert.cosmology.c_ells.AngularCoefficient attribute), 47
 is_cosmological (seyfert.cosmology.parameter.PhysicalParameter attribute), 12
 is_evaluated (seyfert.cosmology.weight_functions.WeightFunction attribute), 42
 is_galaxy_bias_parameter (seyfert.cosmology.parameter.PhysicalParameter attribute), 12
 is_nuisance (seyfert.cosmology.parameter.PhysicalParameter attribute), 12

K

k_grid (seyfert.cosmology.cosmology.Cosmology attribute), 6

L

LensingWeightFunction (class in seyfert.cosmology.weight_functions), 38
 loadCosmology() (seyfert.cosmology.c_ells.AngularCoefficient method), 48
 loadFromHDF5() (seyfert.cosmology.bias.Bias method), 29
 loadFromHDF5() (seyfert.cosmology.c_ells.AngularCoefficient method), 49
 loadFromHDF5() (seyfert.cosmology.c_ells.AngularCoefficientsCollector method), 50
 loadFromHDF5() (seyfert.cosmology.cosmology.Cosmology method), 8
 loadFromHDF5() (seyfert.cosmology.power_spectrum.PowerSpectrum method), 18
 loadFromHDF5() (seyfert.cosmology.redshift_density.RedshiftDensity method), 26
 loadFromHDF5() (seyfert.cosmology.weight_functions.WeightFunction method), 43
 loadStemDisplacements() (seyfert.cosmology.parameter.PhysicalParametersCollection method), 15

M

main_path_rel_to_root (seyfert.cosmology.c_ells.H5Cl attribute), 51
 mnu (seyfert.cosmology.cosmology.Cosmology attribute), 6
 modifiedGaussianResponse() (seyfert.cosmology.redshift_density.RedshiftDensity static method), 26

module

seyfert.cosmology.bias, 27
 seyfert.cosmology.boltzmann_solver, 19
 seyfert.cosmology.c_ells, 44
 seyfert.cosmology.cosmology, 3
 seyfert.cosmology.parameter, 10
 seyfert.cosmology.power_spectrum, 16
 seyfert.cosmology.redshift_density, 22
 seyfert.cosmology.weight_functions, 36

N

n_bins (seyfert.cosmology.bias.BiasModel attribute), 30
 n_bins (seyfert.cosmology.redshift_density.RedshiftDensity attribute), 25
 n_bins (seyfert.cosmology.weight_functions.WeightFunction attribute), 42
 n_ell (seyfert.cosmology.c_ells.AngularCoefficient attribute), 47
 n_i (seyfert.cosmology.c_ells.AngularCoefficient attribute), 47
 n_i_z (seyfert.cosmology.weight_functions.WeightFunction attribute), 42
 n_j (seyfert.cosmology.c_ells.AngularCoefficient attribute), 47
 n_R (seyfert.cosmology.bias.VdnVoidBias attribute), 35
 n_z (seyfert.cosmology.c_ells.AngularCoefficient attribute), 47
 ns (seyfert.cosmology.cosmology.Cosmology attribute), 6
 NUISANCE_PAR_STRING (seyfert.cosmology.parameter.PhysicalParameter attribute), 12
 nuisance_parameters (seyfert.cosmology.parameter.PhysicalParametersCollection attribute), 15

O

obs_key (seyfert.cosmology.c_ells.AngularCoefficient attribute), 48
 Omb (seyfert.cosmology.cosmology.Cosmology attribute), 6
 OmDE (seyfert.cosmology.cosmology.Cosmology attribute), 6
 OmK (seyfert.cosmology.cosmology.Cosmology attribute), 6
 Omm (seyfert.cosmology.cosmology.Cosmology attribute), 6
 Omm (seyfert.cosmology.weight_functions.LensingWeightFunction attribute), 39

P

`ParameterError`, 10
`params` (*seyfert.cosmology.parameter.PhysicalParametersCollection* attribute), 15
`PhotometricGalaxyWeightFunction` (class in *seyfert.cosmology.weight_functions*), 40
`PhysicalParameter` (class in *seyfert.cosmology.parameter*), 11
`PhysicalParametersCollection` (class in *seyfert.cosmology.parameter*), 13
`PiecewiseBias` (class in *seyfert.cosmology.bias*), 33
`plot()` (*seyfert.cosmology.c_ells.AngularCoefficient* method), 49
`power_spectrum` (*seyfert.cosmology.c_ells.AngularCoefficient* attribute), 48
`PowerSpectrum` (class in *seyfert.cosmology.power_spectrum*), 17
`probe` (*seyfert.cosmology.weight_functions.WeightFunction* attribute), 42
`probe_from_weight_function_cls_name()` (in module *seyfert.cosmology.weight_functions*), 36
`probes` (*seyfert.cosmology.c_ells.AngularCoefficientsCollector* attribute), 50
`probes_combinations` (*seyfert.cosmology.c_ells.AngularCoefficientsCollector* attribute), 50

R

`R_max_Mpc` (*seyfert.cosmology.bias.VdnVoidBias* attribute), 34
`R_min_Mpc` (*seyfert.cosmology.bias.VdnVoidBias* attribute), 34
`r_tilde_z` (*seyfert.cosmology.cosmology.Cosmology* attribute), 6
`r_tilde_z` (*seyfert.cosmology.weight_functions.WeightFunction* attribute), 42
`r_z` (*seyfert.cosmology.cosmology.Cosmology* attribute), 7
`readBuildingData()` (*seyfert.cosmology.c_ells.H5Cl* method), 51
`readBuildingData()` (*seyfert.cosmology.cosmology.H5Cosmology* method), 9
`readBuildingData()` (*seyfert.cosmology.weight_functions.H5WeightFunction* method), 38
`readCAMBIniFileToDict()` (*seyfert.cosmology.boltzmann_solver.CAMBoltzmannSolver* static method), 20
`readJSON()` (*seyfert.cosmology.parameter.PhysicalParametersCollection* method), 15
`RedshiftDensity` (class in *seyfert.cosmology.redshift_density*), 24
`resetCurrentValueToFiducial()` (*seyfert.cosmology.parameter.PhysicalParameter* method), 12
`resetPhysicalParametersToFiducial()` (*seyfert.cosmology.parameter.PhysicalParametersCollection* method), 15
`run()` (*seyfert.cosmology.boltzmann_solver.CAMBoltzmannSolver* method), 20
`run()` (*seyfert.cosmology.boltzmann_solver.CLASSBoltzmannSolver* method), 21
`run()` (*seyfert.cosmology.boltzmann_solver.ExternalBoltzmannSolver* method), 22

S

`saveToHDF5()` (*seyfert.cosmology.bias.Bias* method), 29
`saveToHDF5()` (*seyfert.cosmology.c_ells.AngularCoefficient* method), 49
`saveToHDF5()` (*seyfert.cosmology.c_ells.AngularCoefficientsCollector* method), 50
`saveToHDF5()` (*seyfert.cosmology.cosmology.Cosmology* method), 8
`saveToHDF5()` (*seyfert.cosmology.power_spectrum.PowerSpectrum* method), 18

`saveToHDF5()` (*seyfert.cosmology.redshift_density.RedshiftDensity* method), 26
`saveToHDF5()` (*seyfert.cosmology.weight_functions.WeightFunction* method), 43
`setUp()` (*seyfert.cosmology.c_ells.AngularCoefficient* method), 49
`setUp()` (*seyfert.cosmology.c_ells.AngularCoefficientsCollector* method), 50
`setUp()` (*seyfert.cosmology.redshift_density.RedshiftDensity* method), 27
`setUp()` (*seyfert.cosmology.weight_functions.WeightFunction* method), 43
`setUp()` (*seyfert.cosmology.weight_functions.WeightFunctionWithBias* method), 43
`setupBias()` (*seyfert.cosmology.weight_functions.GalaxyClusteringWeightFunction* method), 37
`setupBias()` (*seyfert.cosmology.weight_functions.PhotometricGalaxyWeightFunction* method), 40
`setupBias()` (*seyfert.cosmology.weight_functions.SpectroscopicGalaxyWeightFunction* method), 40
`setupBias()` (*seyfert.cosmology.weight_functions.VoidWeightFunction* method), 41
`setupBias()` (*seyfert.cosmology.weight_functions.WeightFunctionWithBias* method), 43
`seyfert.cosmology.bias` module, 27
`seyfert.cosmology.boltzmann_solver` module, 19
`seyfert.cosmology.c_ells` module, 44
`seyfert.cosmology.cosmology` module, 3
`seyfert.cosmology.parameter` module, 10
`seyfert.cosmology.power_spectrum` module, 16
`seyfert.cosmology.redshift_density` module, 22
`seyfert.cosmology.weight_functions` module, 36
`short_keys` (*seyfert.cosmology.c_ells.AngularCoefficientsCollector* attribute), 50
`shot_noise` (*seyfert.cosmology.redshift_density.RedshiftDensity* attribute), 25
`sigma8` (*seyfert.cosmology.cosmology.Cosmology* attribute), 7
`sigmaR()` (*seyfert.cosmology.cosmology.Cosmology* static method), 8
`SpectroscopicGalaxyWeightFunction` (class in *seyfert.cosmology.weight_functions*), 40

T

`to_dict()` (*seyfert.cosmology.parameter.PhysicalParameter* method), 13
`to_JSON()` (*seyfert.cosmology.parameter.PhysicalParameter* method), 13
`transfer_function_k` (*seyfert.cosmology.cosmology.Cosmology* attribute), 7

U

`updatePhysicalParametersForDvarStep()` (*seyfert.cosmology.parameter.PhysicalParametersCollection* method), 15

`updateValueForDisplacement()`
(*seyfert.cosmology.parameter.PhysicalParameter* method),
13

V

`VdnVoidBias` (class in *seyfert.cosmology.bias*), 34
`voidMultiplicity()` (*seyfert.cosmology.bias.VdnVoidBias* static
method), 35
`VoidWeightFunction` (class in *seyfert.cosmology.weight_functions*),
41

W

`w0` (*seyfert.cosmology.cosmology.Cosmology* attribute), 7
`wa` (*seyfert.cosmology.cosmology.Cosmology* attribute), 7
`weight1` (*seyfert.cosmology.c_ells.AngularCoefficient* attribute), 48
`weight2` (*seyfert.cosmology.c_ells.AngularCoefficient* attribute), 48
`weight_cls_name_from_obs()` (in module
seyfert.cosmology.weight_functions), 36
`weight_function_for_probe()` (in module
seyfert.cosmology.weight_functions), 36
`WeightFunction` (class in *seyfert.cosmology.weight_functions*), 41
`WeightFunctionWithBias` (class in
seyfert.cosmology.weight_functions), 43
`writeDictToIniFile()`
(*seyfert.cosmology.boltzmann_solver.CAMBBoltzmannSolver*
static method), 20
`writeJSON()`
(*seyfert.cosmology.parameter.PhysicalParametersCollection*
method), 15
`writeObjectToFile()` (*seyfert.cosmology.bias.H5Bias* method), 33
`writeObjectToFile()` (*seyfert.cosmology.c_ells.H5Cl* method), 51
`writeObjectToFile()` (*seyfert.cosmology.c_ells.H5ClCollection*
method), 52
`writeObjectToFile()` (*seyfert.cosmology.cosmology.H5Cosmology*
method), 9
`writeObjectToFile()`
(*seyfert.cosmology.power_spectrum.H5PowerSpectrum*
method), 17
`writeObjectToFile()`
(*seyfert.cosmology.redshift_density.H5Density* method), 24
`writeObjectToFile()`
(*seyfert.cosmology.weight_functions.H5WeightFunction*
method), 38
`writeToObject()` (*seyfert.cosmology.bias.H5Bias* method), 33
`writeToObject()` (*seyfert.cosmology.c_ells.H5Cl* method), 51
`writeToObject()` (*seyfert.cosmology.c_ells.H5ClCollection* method),
52
`writeToObject()` (*seyfert.cosmology.cosmology.H5Cosmology*
method), 9
`writeToObject()`
(*seyfert.cosmology.power_spectrum.H5PowerSpectrum*
method), 17
`writeToObject()` (*seyfert.cosmology.redshift_density.H5Density*
method), 24
`writeToObject()`
(*seyfert.cosmology.weight_functions.H5WeightFunction*
method), 38
`writeUpdatedCAMBIniFile()`
(*seyfert.cosmology.boltzmann_solver.CAMBBoltzmannSolver*
method), 21

Z

`z_array` (*seyfert.cosmology.c_ells.AngularCoefficient* attribute), 48
`z_bin_centers` (*seyfert.cosmology.bias.BiasModel* attribute), 30
`z_bin_centers` (*seyfert.cosmology.redshift_density.RedshiftDensity*
attribute), 25

`z_bin_centers` (*seyfert.cosmology.weight_functions.WeightFunction*
attribute), 42
`z_bin_edges` (*seyfert.cosmology.weight_functions.WeightFunction*
attribute), 42
`z_grid` (*seyfert.cosmology.bias.VdnVoidBias* attribute), 35
`z_max` (*seyfert.cosmology.redshift_density.RedshiftDensity* attribute),
25
`z_max` (*seyfert.cosmology.weight_functions.WeightFunction* attribute),
42
`z_min` (*seyfert.cosmology.redshift_density.RedshiftDensity* attribute),
25
`z_min` (*seyfert.cosmology.weight_functions.WeightFunction* attribute),
42