cosmicfishpie

Release 0.1.3

S. Casas, M. Martinelli, S. Pamuk, Sabarish V.M.

Feb 26, 2024

GETTING STARTED

CHAPTER

ONE

INSTALLATION

cosmicfishpie supports Python >= 3.8.

1.1 Installing with `pip`

cosmicfishpie is available on PyPI. Just run

```
pip install cosmicfishpie
```

1.2 Installing from source

To install **cosmicfishpie** from source, first clone the repository:

```
git clone https://github.com/santiagocasas/cosmicfishpie.git
cd cosmicfishpie
```

Then run

```
pip install -e .
```

CHAPTER

TWO

OVERVIEW

CHAPTER

THREE

COSMICFISHPIE PACKAGE

3.1 Subpackages

3.1.1 cosmicfishpie.CMBsurvey package

Submodules

cosmicfishpie.CMBsurvey.CMB_cov module

CMB MAIN

This is the main engine for CMB Fisher Matrix.

class cosmicfishpie.CMBsurvey.CMB_cov.**CMBCov**(*cosmopars*, *print_info_specs=False*)

Bases: object

compute_covmat()

compute_derivs(*print_info_specs=False*)

get_covmat(*noisy_cls*)

Data covariance

Parameters
noisy_cls (dict) – dictionary containing cls with noise

Returns

- list
- data covariance matrix

getcls(*allpars*)

getclsnoise(*cls*)

Noise value

Parameters
cls (dict) – cls dictionary

Returns
Noise value[ell,bin]

Return type
array

Note: Implements the following equation:

$$N_\ell^X = ...$$

sum_inv_squares(*arr*)

cosmicfishpie.CMBsurvey.CMB_obs module

CLS

This module contains cls calculations (only LSS atm).

class cosmicfishpie.CMBsurvey.CMB_obs.**ComputeCls**(*cosmopars*, *print_info_specs=False*)

Bases: object

compute_all()

computecls()

Cls computation

Parameters

- **ell** (float) – multipole
- **X** (str) – first observable
- **Y** (str) – second observable
- **i** (int) – first bin
- **j** (int) – second bin

Returns
Value of Cl

Return type
float

print_numerical_specs()

Module contents

CMB Module

Provide some basic description of the module computing CMB stuff.

3.1.2 cosmicfishpie.LSSsurvey package

Submodules

cosmicfishpie.LSSsurvey.photo_cov module

LSS MAIN

This is the main engine for LSS Fisher Matrix.

class cosmicfishpie.LSSsurvey.photo_cov.**PhotoCov**(*cosmopars*, *photopars*, *IApars*, *biaspars*, *fiducial_Cls=None*)

Bases: object

compute_covmat()

Computes the fiducial covariance matrix for the Fisher matrix.

Returns

- dict – a dictionary with all the auto and cross correlation fiducial angular power spectra with noise added to it
- list – A list of pandas.DataFrame objects that store the covariance matrix for each multipole

compute_derivs(*print_info_specs=False*)

computes the derivatives of the angular power spectrum needed to compute the Fisher matrix

Returns
a dictionary containing the derivatives of the angular power spectrum at the fiducial for all free parameters

Return type
dict

get_covmat(*noisy_cls*)

Computes the covariance matrix from the noisy angular power spectrum

Parameters
noisy_cls (dict) – a dictionary with all the auto and cross correlation angular power spectra with noise added to it

Returns
A list of pandas.DataFrame objects that store the covariance matrix for each multipole

Return type
list

getcls(*allpars*)

Function to calculate the angular power spectrum

Parameters
allpars (dict) – a dictionary with all parameters needed to compute the angular power spectrum

Returns
a dictionary with all the auto and cross correlation angular power spectra

Return type
dict

getclsnoise(*cls*)

Obtain the angular power spectrum with noise

Parameters
cls (dict) – a dictionary with all the auto and cross correlation angular power spectra

Returns
a dictionary with all the auto and cross correlation angular power spectra with noise added to it

Return type
dict

Note: Implements the following equation:

$$N_{ij}^{YX}(\ell) = \frac{\sigma_X^2}{\bar{n}_i}\,\delta_{XY}\,\delta_{ij}$$
$$\sigma_L = \sigma_\epsilon \quad , \quad \sigma_G = 1$$

---

cosmicfishpie.LSSsurvey.photo_obs module

CLS

This module contains cls calculations (only LSS atm).

class cosmicfishpie.LSSsurvey.photo_obs.**ComputeCls**(*cosmopars*, *photopars*, *IApars*, *biaspars*, *print_info_specs=False*, *fiducial_cosmo=None*)

Bases: object

P_limber()

Calculates the Limber-approximated power spectrum. This is done for a range of redshift values and multipoles. Will add MG effects to WL and(!) GCph if asked for.

Note: This does not return the power spectra. The results are stored in the attribute *Pell*

---

clsintegral(*obs1*, *obs2*, *bin1*, *bin2*, *hub*)

function to obtain the angular power spectrum as an function of the multipole.

Parameters

- **obs1** (str) – name of the observable of the redshift bin bin1 (GC or WL)
- **obs2** (str) – name of the observable of the redshift bin bin2 (GC or WL)
- **bin1** (int) – index of the redshift bin at which the observable obs1 is measured
- **bin2** (int) – index of the redshift bin at which the observable obs2 is measured
- **hub** (callable) – function that should be passed a numpy.ndarray of redshifts returns the hubble expansion rate at these redshifts.

Returns
Function that receives an array multipole and returns the angular power spectrum for these multipoles.

Return type
callable

compute_all()

Main function to compute the angular power spectrum. Will first compute the limber approximated angular power spectrum and the window functions for both probes. From that it will obtain the angular power spectrum.

Note: This function does not return the calculated quantities. Rather they are found in new attributes of the object.

The result of the Limber approximated power spectra are found in the new attributes *Pell* and *sqrtPell*

The window functions are found in the new attributes *GC* if galaxy clustering is asked for, and *WL* if cosmic sheer is asked for

The angular power spectrum is found in the new attribute *result*

`compute_kernels()`

function to compute all the needed window functions.

Note: This does not return the window functions. They are found in the new attributes *GC* if galaxy clustering is asked for, and *WL* if cosmic sheer is asked for

---

`computecls()`

Function to compute the angular power spectrum for all observables, redshift bins and multipoles.

Returns
a dictionary containing all auto and cross correlation angular power spectra. Its keys are formatted to have an array of the power spectra in 'X ixY j'. Also the multipoles for which the angular power spectra were computed for are found in 'ells'.

Return type
`dict`

Note: Implements the following equation:

$$C_{i,j}^{X,Y}(\ell) = c \int \mathrm{d}z \frac{W_i^X(z) W_j^Y(z)}{H(z) r^2(z)} P_{\delta\delta}\left[\frac{\ell + 1/2}{r(z)}, z\right]$$

`galaxy_kernel(z, i)`

Calculates the GCph kernel function

Parameters

- **z** (`numpy.ndarray`) – array containing the redshifts for which the kernel should be computed for
- **i** (`int`) – index of the redshift bin

Returns
Value of the galaxy clustering kernel at redshift z for bin i

Return type
`numpy.ndarray`

Note: Implements the following equation:

$$W_i^{GCph} = b(z) \frac{n_i(z)}{\bar{n}(z)} H(z)$$

---

genwindow(*z*, *obs*, *i*)

generic function to obtain the window functions for the different observables.

Parameters

- **z** (numpy.ndarray) – array of redshifts for which the window function should be computed
- **obs** (str) – name of the observable (GC or WL)
- **i** (int) – integer of the redshift bin the window should be computed for

Returns
Values of the window function for the observable obs, for redshift bin i, and at redshifts z.

Return type
numpy.ndarray

integral_efficiency(*i*)

computes the integral that enters the lensing kernel for a given redshift bin

Parameters
i (int) – index of the redshift bin for which the lensing efficiency should be calculated

Returns
callable function that receives a numpy.ndarray of requested redshifts and returns the lensing efficiency for the i-th bin as a numpy.ndarray

Return type
callable

lensing_efficiency()

computes the integral that enters the lensing kernel for all redshift bins

Returns
list of callable functions that give the lensing efficiency for each bin

Return type
list

lensing_kernel(*z*, *i*)

Computes the photometric cosmic shear kernel function

Parameters

- **z** (numpy.ndarray) – array containing the redshifts for which the kernel should be computed for
- **i** (int) – index of the redshift bin

Returns
Value of the cosmic shearing kernel at redshift z for bin i

Return type
numpy.ndarray

Note: Implements the following equation:

$$W_i^{WL} = W_i^{IA} + \frac{3}{2}\left(\frac{H_0}{c}\right)^2 \Omega_{m,0}(1+z) r(z) \int_z^{z_{\max}} dz' \frac{n_i(z')}{\bar{n}(z)} \left[1 - \frac{r(z)}{r(z')}\right]$$

---

print_numerical_specs()

prints the numerical specifications of the internal computations

sqrtP_limber()

Calculates the square root of the Limber-approximated power spectrum for weak lensing (WL) and galaxy clustering photometric (GCph) observables. This is done for a range of redshift values and multipoles. Depending on the tracer asked for in 'GCph_Tracer' will use 'matter' or 'clustering' observable GCph. Will add MG effects to WL if asked for.

Note: This does not return the power spectra. The results are stored in the attribute *sqrtPell*

---

cosmicfishpie.LSSsurvey.photo_obs.**faster_integral_efficiency**(*i*, *ngal_func*, *comoving_func*, *zarr*)

function to do the integration over redshift that shows up in the lensing kernel

Parameters

- **i** (int) – index of the redshift bin for which the lensing kernel should be computed
- **ngal_func** (callable) – callable function that returns the number density distribution of galaxies. Should be a function of the redshift bin index i as an int and the redshift z as a numpy.ndarray
- **comoving_func** (callable) – callable function that returns the comoving distance. Should beb a function of the redshift z as a numpy.ndarray
- **zarr** (numpy.ndarray) – 1d array of all redshifts z that should be used as integration points.

Returns

callable function that receives a numpy.ndarray of requested redshifts and returns the lensing efficiency for the i-th bin as a numpy.ndarray

Return type

callable

cosmicfishpie.LSSsurvey.photo_obs.**memo_integral_efficiency**(*i*, *ngal_func*, *comoving_func*, *z*, *zint_mat*, *diffz*)

function to do the integration over redshift that shows up in the lensing kernel

Parameters

- **i** (int) – index of the redshift bin for which the lensing kernel should be computed
- **ngal_func** (callable) – callable function that returns the number density distribution of galaxies. Should be a function of the redshift bin index i as an int and the redshift z as a numpy.ndarray
- **comoving_func** (callable) – callable function that returns the comoving distance. Should be a function of the redshift z as a numpy.ndarray
- **z** (numpy.ndarray) – 1d array of all redshifts z that the result of the integral should use as interpolated over.
- **zint_mat** (numpy.ndarray) – 2d array of redshifts z that the integral should use as integration points. The first row must coincide with z.
- **diffz** (numpy.ndarray) – 2d array of the separation of integration points

Returns

callable function that receives a numpy.ndarray of requested redshifts and returns the lensing efficiency for the i-th bin as a numpy.ndarray

Return type
callable

Note: This function is deprecated. Use *faster_integral_efficiency* instead.

---

cosmicfishpie.LSSsurvey.photo_window module

LSS_window

This module returns the window functions for LSS surveys.

class cosmicfishpie.LSSsurvey.photo_window.**GalaxyPhotoDist**(*photopars*)

Bases: object

dNdz(*z*)

unnormalized dN/dz(z)

Parameters
z (numpy.ndarray) – array of redshifts at which to compute the galaxy distribution

Returns
unnormalized theoretical galaxy redshift distribution

Return type
numpy.ndarray

Note: Implements the following parametrization:

$$\frac{\mathrm{d}N}{\mathrm{d}z} = \left(\frac{z}{z_b}\right)^2 \exp\left[-\left(\frac{z}{z_0}\right)^{n_\gamma}\right]$$

---

n_i(*z*, *i*)

Function to compute the unnormalized dN/dz(z) with a window picking function applied to it

Parameters

- **z** (float) – Redshift
- **i** (int) – index of the redshift bin

Returns
binned distribution without photometric redshift errors

Return type
float

ngal_photoz(*z*, *i*)

Function to compute the binned galaxy redshift distribution convolved with photometric redshift errors n^{ph}_i(z)

Parameters

- **z** (float) – redshift at which to compute the distribution
- **i** (int) – index of the redshift bin

Returns
binned galaxy distribution convolved with photometric redshift errors

Return type
float

Note: Implements the following equation:

$$p_{ph}(z_p|z) = \frac{1-f_{out}}{\sqrt{2\pi}\sigma_b(1+z)} \exp\left\{-\frac{1}{2}\left[\frac{z-c_b z_p - z_b}{\sigma_b(1+z)}\right]^2\right\} + \frac{f_{out}}{\sqrt{2\pi}\sigma_0(1+z)} \exp\left\{-\frac{1}{2}\left[\frac{z-c_0 z_p - z_0}{\sigma_0(1+z)}\right]^2\right\}$$

---

norm()

n^{ph}_i(z)

Parameters

- **z** (float) – redshift at which to compute the function
- **i** (integer) – redshift bin index

Returns
normalized binned galaxy distribution convolved with photoz errors

Return type
float

norm_ngal_photoz(*z*, *i*)

n^{ph}_i(z)

Parameters

- **z** (array) – redshift at which to compute the function
- **i** (integer) – redshift bin index

Returns
normalized binned galaxy distribution convolved with photoz errors

Return type
float

cosmicfishpie.LSSsurvey.spectro_cov module

CLS

This module contains cls calculations (only LSS atm).

class cosmicfishpie.LSSsurvey.spectro_cov.**SpectroCov**(*fiducialpars*, *fiducial_specobs=None*, *bias_samples=['g', 'g']*)

Bases: object

P_noise_21(*z*, *k*, *mu*, *temp_dim=True*, *beam_term=False*)

Compute the shotnoise of the 21 centimeter intensity mapping probe

Parameters

- **z** (float, numpy.ndarray) – Redshift at which the noise is to be computed
- **k** (float, numpy.ndarray) – Wavenumber
- **mu** (float, numpy.ndarray) – The cosine of angle between the wavevector and the line-of-sight direction.

- **temp_dim** (bool) – If true the Temperature terms is in units of Kelvin^2
- **beam_term** (bool) – If true will add the beam term to the computation of the power spectrum

Returns
Additional shotnoise of the 21 cm intensity mapping

Return type
float, numpy.ndarray

Tsys_func(*z*)

Calculates Tsys in mK

Parameters
z (float, numpy.ndarray) – Redshift at which to compute Tsys

Returns
Tsys at z in milli Kelvin

Return type
float, numpy.ndarray

cov(*ibin*, *k*, *mu*)

Function to calculate the covariance the galaxy clustering probe

Parameters

- **ibin** (int) – Index of the redshift bin for which the covariance is to be computed
- **k** (float, numpy.ndarray) – Wavenumber
- **mu** (float, numpy.ndarray) – The cosine of angle between the wavevector and the line-of-sight direction.

Returns
Covariance of the Galaxy clustering probe

Return type
float, numpy.ndarray

d_volume(*ibin*)

Calculates the comoving volume of a redshift bin

Parameters
i (int) – Index of the survey redshift bin

Returns
Comoving volume of the redshift bin

Return type
float

n_density(*ibin*)

calculate the comoving number density of the probe

Parameters
i (int) – Index of the survey redshift bin

Returns
comoving number density of the probe

Return type
float

veff(*ibin*, *k*, *mu*)

calculate the effective volume entering the covariance of the galaxy clustering probe

Parameters

- **i** (int) – Index of the survey redshift bin
- **k** (float, numpy.ndarray) – wave number at which the effective volume should be calculated
- **mu** (float, numpy.ndarray) – The cosine of angle between the wavevector and the line-of-sight direction.
- **Returns** –
- **float** – The effective volume for a given wavenumber, angle and redshift
- **numpy.ndarray** – The effective volume for a given wavenumber, angle and redshift

veff_21cm(*ibin*, *k*, *mu*)

calculate the effective volume entering the covariance of the line intensity mapping probe

Parameters

- **i** (int) – Index of the survey redshift bin
- **k** (float, numpy.ndarray) – wave number at which the effective volume should be calculated
- **mu** (float, numpy.ndarray) – The cosine of angle between the wavevector and the line-of-sight direction.
- **Returns** –
- **float** – The effective volume for a given wavenumber, angle and redshift
- **numpy.ndarray** – The effective volume for a given wavenumber, angle and redshift

veff_XC(*ibin*, *k*, *mu*)

calculate the effective volume entering the covariance of the cross correlation of galaxy clustering and intensity mapping

Parameters

- **i** (int) – Index of the survey redshift bin
- **k** (float, numpy.ndarray) – wave number at which the effective volume should be calculated
- **mu** (float, numpy.ndarray) – The cosine of angle between the wavevector and the line-of-sight direction.
- **Returns** –
- **float** – The effective volume for a given wavenumber, angle and redshift
- **numpy.ndarray** – The effective volume for a given wavenumber, angle and redshift

volume_bin(*zi*, *zj*)

Calculates the comoving volume of a spherical shell

Parameters

- **zi** (float, numpy.ndarray) – Redshift of the inner sphere
- **zj** (float, numpy.ndarray) – Redshift of the outer sphere

Returns
Volume of the comoving spherical shell between zj and zi

Return type
float, numpy.ndarray

volume_survey(*ibin*)

Calculates the survey volume of a redshift bin

Parameters
i (int) – Index of the survey redshift bin

Returns
survey volume of the redshift bin

Return type
float

class cosmicfishpie.LSSsurvey.spectro_cov.**SpectroDerivs**(*z_array*, *pk_kmesh*, *pk_mumesh*, *fiducial_spectro_obj*, *bias_samples=['g', 'g']*)

Bases: object

compute_derivs(*freeparams={}*)

Calls the common derivative engine to compute the derivatives of the observed power spectrum

Parameters
freeparams (dict, *optional*) – A dictionary containing the names and step sizes for all parameters you want to vary. Will default to the global free params if not passed.

Returns
A dictionary containing lists of derivatives of the observed power spectrum for each redshift bin and parameter

Return type
dict

dlnpobs_dcosmop(*zi*, *k*, *mu*, *par*)

This is a deprecated function! It was used to compute the compute the derivatives of the power spectrum with respect to the cosmological parameters internally in the cosmicfishpie.LSSsurvey.spectro_cov.SpectroDerivs . Use now the common derivative engine at cosmicfishpie.fishermatrix.derivatives.derivatives

Parameters

- **zi** (float, numpy.ndarray) – The redshifts values of interest.
- **k** (float) – The wavenumber for which the power spectrum should be computed
- **mu** (float, numpy.ndarray) – The cosine of angle between the wavevector and the line-of-sight direction.
- **par** (str) – Name of the parameter with regards to which the derivative should be computed for

Returns
Derivative of the observed power spectrum with regard to the passed parameter

Return type
float, numpy.ndarray

dlnpobs_dnuisp(*zi*, *k*, *mu*, *par*)

This is a deprecated function! It was used to compute the compute the derivatives of the power spectrum with respect to nuisance parameters in the cosmicfishpie.LSSsurvey.spectro_cov.SpectroDerivs . Use now the common derivative engine at cosmicfishpie.fishermatrix.derivatives.derivatives

Parameters

- **zi** (float, numpy.ndarray) – The redshifts values of interest.
- **k** (float) – The wavenumber for which the power spectrum should be computed
- **mu** (float, numpy.ndarray) – The cosine of angle between the wavevector and the line-of-sight direction.
- **par** (str) – Name of the parameter with regards to which the derivative should be computed for

Returns
Derivative of the observed power spectrum with regard to the passed parameter

Return type
float, numpy.ndarray

dlnpobs_dp(*zi*, *k*, *mu*, *par*)

This is a deprecated function! It was used to compute the compute the derivatives of the power spectrum internally in the cosmicfishpie.LSSsurvey.spectro_cov.SpectroDerivs . Use now the common derivative engine at cosmicfishpie.fishermatrix.derivatives.derivatives

Parameters

- **zi** (float, numpy.ndarray) – The redshifts values of interest.
- **k** (float) – The wavenumber for which the power spectrum should be computed
- **mu** (float, numpy.ndarray) – The cosine of angle between the wavevector and the line-of-sight direction.
- **par** (str) – Name of the parameter with regards to which the derivative should be computed for

Returns
Derivative of the observed power spectrum with regard to the passed parameter

Return type
float, numpy.ndarray

exact_derivs(*par*)

Compute the exact log derivative of the Power spectrum with respect to the shotnoise using chain rule

Parameters
par (str) – String name of the Shotnoise parameter for which the exact derivative should be computed from

Returns
A dictionary containing the exact log derivative with respect to the shotnoise parameter

Return type
dict

get_obs(*allpars*)

function to obtain the power spectrum of the observable

Parameters
allpars (dict) – A dictionary containing all relevant parameters to compute the observed power spectrum

Returns
A dictionary containing the observed power spectrum on the k and mu grid for all redshift bins

Return type
dict

initialize_obs(*allpars*)

kronecker_bins(*par*, *zmids*, *zi*)

function to figure out what bin a parameter corresponds and compares to the redshift passed

Parameters

- **par** (str) – String name of a parameter. The name should end with '_i' where i marks the bin it corresponds to
- **zmid** (numpy.ndarray) – List of the centers for the redshift bin
- **zi** (float) – redshift for which we want to find the bin

Returns
returns 1 if passed redshift is in the bin corresponding to the parameter. Returns 0 else wise

Return type
int

cosmicfishpie.LSSsurvey.spectro_obs module

CLS

This module contains cls calculations (only LSS atm).

class cosmicfishpie.LSSsurvey.spectro_obs.**ComputeGalIM**(*cosmopars*, *fiducial_cosmopars=None*, *spectrobiaspars=None*, *IMbiaspars=None*, *PShotpars=None*, *fiducial_cosmo=None*, *use_bias_funcs=True*, *bias_samples=['I', 'I']*)

Bases: *ComputeGalSpectro*

Omega_HI(*z*)

Temperature(*z*)

obtaining the temperature (T^2(z)) for the Power Spectrum (PHI(z))

alpha_SD()

beta_SD(*z*, *k*, *mu*)

lnpobs_IM(*z*, *k*, *mu*, *bsi_z=None*, *bsj_z=None*)

observed_P_HI(*z*, *k*, *mu*, *bsi_z=None*, *bsj_z=None*, *si='I'*, *sj='I'*)

set_IM_specs()

theta_b(*zz*)

class cosmicfishpie.LSSsurvey.spectro_obs.**ComputeGalSpectro**(*cosmopars*, *fiducial_cosmopars=None*, *spectrobiaspars=None*, *spectrononlinearpars=None*, *PShotpars=None*, *fiducial_cosmo=None*, *bias_samples=['g', 'g']*, *use_bias_funcs=True*)

Bases: object

BAO_term(*z*)

Calculates the BAO term. This is the rescaling of the fourier volume by the AP-effect

Parameters
z (float, numpy.ndarray) – The redshifts of interest

Returns
Value of BAO term at redshifts z

Return type
float, numpy.ndarray

Note: Implements the following equation:

$$\mathrm{BAO} = q_\perp^2 \, q_\parallel$$

FingersOfGod(*z*, *k*, *mu*, *mode='Lorentz'*)

Calculates the Fingers of God effect in redshift-space power spectra.

Parameters

- **z** (float, numpy.ndarray) – The redshifts values of interest.
- **k** (float) – The wavenumber for which the suppression should be computed
- **mu** (float, numpy.ndarray) – The cosine of angle between the wavevector and the line-of-sight direction.
- **mode** (str, *optional*) – A string parameter indicating the model to use. Defaults to 'Lorentz'.

Returns
The calculated FoG term, which is 1 if either FoG_switch is False or linear_switch is True. Otherwise, it depends on the specified mode.

Return type
float, numpy.ndarray

Note: If mode is "Lorentz" this implements following equation

..math:

```
\mathrm{FoG} = \frac{1}{1+\left[f\,\sigma_p\,\mu^2\right]^2}
```

P_ThetaTheta_Moments(*zz*, *moment=0*)

Calculates the angular power spectrum moments of the velocity divergence field, also known as the Theta field.

Parameters

- **zz** (float) – The redshift value at which to calculate the power spectrum.
- **moment** (int) – An integer indicating the order of the moment to calculate. Default is 0.

Returns
The power spectrum moment of the velocity divergence field.

Return type
float

activate_terms()

Update which modelling effects should be taken into consideration

bterm_fid(*z*, *bias_sample='g'*)

Calculates the fiducial bias term at a given redshift z, of either galaxies or intensity mapping.

Parameters:

z
[float, numpy.ndarray] The redshifts value at which to evaluate the bias term.

bias_sample
[str, optional] Specifies whether to compute the galaxy ('g') or intensity mapping ('I') bias term. (default='g')

Returns:

: float The value of the bias term at *z*.

dewiggled_pdd(*z*, *k*, *mu*)

This function calculates the normalized dewiggled power spectrum.

Parameters

- **z** (float, numpy.ndarray) – The redshifts values of interest.
- **k** (float) – The wavenumber for which the power spectrum should be computed
- **mu** (float, numpy.ndarray) – The cosine of angle between the wavevector and the line-of-sight direction.

Returns
The the mildly non-linear (dewiggled) power spectrum.

Return type
float, numpy.ndarray

Note: If the config asks for only linear spectrum this just returns the power spectrum normalized with either 1 or 1/sigma8^2

---

k_units_change(*k*)

Function that rescales the k-array, when the kmax-kmin integration units are fixed in h/Mpc, while the rest of the code is defined in 1/Mpc.

Parameters
k (float, numpy.ndarray) – wavenumbers in units of h sample/Mpc to be rescaled

Returns
wavenumbers in un units of h ref/Mpc

Return type
float, numpy.ndarray

kaiserTerm(*z*, *k*, *mu*, *b_i=None*, *just_rsd=False*, *bias_sample='g'*)

Computes the Kaiser redshift-space distortion term.

Parameters

- **z** (float, numpy.ndarray) – Redshifts of interest
- **k** (float, numpy.ndarray) – Wave numbers at which to calculate the linear RSD
- **mu** (float, numpy.ndarray) – cosine of angles between line of sight and the wavevector.
- **b_i** (float, numpy.ndarray, *optional*) – galaxy bias at Redshifts z
- **just_rsd** (bool, *optional*) – If True, returns only the RSD term. Otherwise, computes the full Kaiser term. Defaults to False.
- **bias_sample** (str, *optional*) – Bias term to use from self.bterm_fid(). Possible values: 'g': galaxies, 'I': intensity mapping. Defaults to 'g'.

Returns
The computed Kaiser term for redshift space distortions.

Note: Implements the following equation:

$$\mathrm{RSD} = \left(b_i + f\,\mu^2\right)$$

---

kmu_alc_pac(*z*, *k*, *mu*)

Function rescaling k and mu with the Alcock-Paczynski effect

Parameters

- **z** (numpy.ndarray, float) – redshift
- **k** (numpy.ndarray, float) – wavevector
- **mu** (numpy.ndarray, float) – cosine of angle between line of sight and the wavevector

Return type
numpy.ndarray, float

Note: Implements the following equation:

$$k^{obs} = k\,\sqrt{\left(q_\parallel \mu\right)^2 + \left(1-\mu^2\right) q_\perp^2}$$
$$\mu^{obs} = \mu\, q_\parallel\, \sqrt{\left(q_\parallel \mu\right)^2 + \left(1-\mu^2\right) q_\perp^2}^{-1}$$

---

kpar(*z*, *k*, *mu*)

Computes the parallel projection of a wavevector. Takes into acount AP-effect

Parameters

- **z** (float, numpy.ndarray) – The redshift of interest.
- **k** (float, numpy.ndarray) – wavenumbers at which to compute the power spectrum. Must be in units of Mpc^-1/h.
- **mu** (float, numpy.ndarray) – cosine of the angel between the line of sight and the wavevector

Returns

Observed parallel projection of wavevector onto the line of sight with AP-effect corrected for

kper(*z*, *k*, *mu*)

Computes the perpendicular projection of a wavevector. Takes into acount AP-effect

Parameters

- **z** (float, numpy.ndarray) – The redshift of interest.
- **k** (float, numpy.ndarray) – wavenumbers at which to compute the power spectrum. Must be in units of Mpc^-1/h.
- **mu** (float, numpy.ndarray) – cosine of the angel between the line of sight and the wavevector

Returns

Observed perpendicular projection of wavevector onto the line of sight with AP-effect corrected for

lnpobs_gg(*z*, *k*, *mu*, *b_i=None*)

This function calculates the natural logarithm of the observed galaxy power spectrum.

Parameters

- **z** (float, numpy.ndarray) – The redshifts values of interest.
- **k** (float) – The wavenumber for which the power spectrum should be computed
- **mu** (float, numpy.ndarray) – The cosine of angle between the wavevector and the line-of-sight direction.
- **b_i** (float, numpy.ndarray, *optional*) – Redshift dependant values of the galaxy bias

Returns

The observed galaxy power spectrums natural logarithm

Return type

float, numpy.ndarray

normalized_pdd(*z*, *k*)

This function normalizes the power spectrum to have a variance smoothed over 8 Mpc/h of 1. This is to cancel out possible terms with σ_8 in the RSD. :type z: :param z: The redshift at which to compute the normalized power spectrum :type z: float, numpy.ndarray :type k: :param k: Wavenumber at which to compute the normalized power spectrum :type k: float, numpy.ndarray

Returns

The Normalized power spectrum

Return type

float, numpy.ndarray

Note: This is not really a normalisation if there is no σ_8 terms inside of the RSD (Kaiserterm). It is then canceled out automatically

normalized_pnw(*z*, *k*)

This function normalizes the power spectrum with the BAO wiggles subtracted from to have a variance smoothed over 8 Mpc/h of roughly 1. This is to cancel out possible terms with σ_8 in the RSD. :type z: :param z: The redshift at which to compute the normalized 'no-wiggle' power spectrum :type z: `float`, `numpy.ndarray` :type k: :param k: Wavenumber at which to compute the normalized 'no-wiggle' power spectrum :type k: `float`, `numpy.ndarray`

Returns
The Normalized 'no-wiggle' power spectrum

Return type
`float`, `numpy.ndarray`

Note: This is not really a normalisation if there is no σ_8 terms inside of the RSD (Kaiserterm). It is then canceled out automatically

observed_Pgg(*z*, *k*, *mu*, *b_i=None*)

This function calculates the observed galaxy power spectrum.

Parameters

- **z** (`float`, `numpy.ndarray`) – The redshifts values of interest.
- **k** (`float`) – The wavenumber for which the power spectrum should be computed
- **mu** (`float`, `numpy.ndarray`) – The cosine of angle between the wavevector and the line-of-sight direction.
- **b_i** (`float`, `numpy.ndarray`, *optional*) – Redshift dependant values of the galaxy bias

Returns
The observed galaxy power spectrum

Return type
`float`, `numpy.ndarray`

Note: In presence of all modeling terms, this function implements the following equation:

$$P_{gg}^{obs} = q_\perp^2 \, q_\parallel \, \mathrm{RSD}^2 \, \mathrm{FoG} \, \frac{P_{dw}^{obs}}{\sigma_8^2} \, \mathrm{Err} + P_{shot}$$

qparallel(*z*)

Function implementing q parallel of the Alcock-Paczynski effect

Parameters
z (`numpy.ndarray`) – list of redshifts for which the q parallel should be computed

Returns
redshift dependant value of q parallel

Return type
`numpy.ndarray`

qperpendicular(*z*)

Function implementing q perpendicular of the Alcock-Paczynski effect

Parameters
z (numpy.ndarray) – list of redshifts for which the q perpendicular should be computed

Returns
redshift dependant value of q perpendicular

Return type
numpy.ndarray

set_internal_kgrid()

Updates the internal grid of wavenumbers used in the computation

set_spectro_specs()

Updates the spectroscopic redshift error

sigmapNL(*zz*)

Function to calculate the variance of the velocity dispersion

Parameters
zz (float) – The redshift value at which to calculate the variance.

Returns
Calculates the variance of the pairwise velocity dispersion. Enters into the FOG effect.

Return type
float

sigmavNL(*zz*, *mu*)

Function to calculate the variance of the displacement field

Parameters
zz (float) – The redshift value at which to calculate the variance.

Returns
Calculates the variance of the displacement field. Enters into the dewiggling weight to obtain the mildly nonlinear power spectrum

Return type
float

spec_err_z(*z*, *k*, *mu*)

Function to compute the scale dependant suppression of the observed power spectrum due to the spectroscopic redshift error

Parameters

- **z** (float, numpy.ndarray) – The redshifts of interest.
- **k** (float, numpy.ndarray) – wavenumbers at which to compute the power spectrum suppression.
- **mu** (float, numpy.ndarray) – cosine of the angel between the line of sight and the wavevector.

Returns
Suppression of the observed power spectrum due to the error on spectroscopic redshift determination.

Return type
float, numpy.ndarray

Note: Implements the following equation:

$$\mathrm{Err} = \exp\left[-\sigma_\parallel^2\, k^2\, \mu^2 - \sigma_\perp^2\, k^2\, \left(1-\mu^2\right)\right].$$

Module contents

LSS Module

Provide some basic description of the theory module.

3.1.3 cosmicfishpie.analysis package

Submodules

cosmicfishpie.analysis.colors module

class cosmicfishpie.analysis.colors.**bash_colors**

Bases: object

This class contains the necessary definitions to print to bash screen with colors. Sometimes it can be useful and nice!

Variables

- **HEADER** – ANSI color for light purple.
- **OKBLUE** – ANSI color for blue.
- **OKGREEN** – ANSI color for green.
- **WARNING** – ANSI color for yellow.
- **FAIL** – ANSI color for red.
- **BOLD** – ANSI code for bold text.
- **UNDERLINE** – ANSI code for underlined text.
- **ENDC** – ANSI code to restore the bash default.

blue(*string*)

Function that returns a string that can be printed to bash in `cosmicfish_pylib.colors.bash_colors.OKBLUE` color.

Parameters
string (*string*) – input string.

Returns
the input string with the relevant ANSI code at the beginning and at the end.

Return type
string

bold(*string*)

Function that returns a string that can be printed to bash in `cosmicfish_pylib.colors.bash_colors.BOLD` color.

Parameters
string (*string*) – input string.

Returns
the input string with the relevant ANSI code at the beginning and at the end.

Return type
string

fail(*string*)

Function that returns a string that can be printed to bash in `cosmicfish_pylib.colors.bash_colors.FAIL` color.

Parameters
string (*string*) – input string.

Returns
the input string with the relevant ANSI code at the beginning and at the end.

Return type
string

green(*string*)

Function that returns a string that can be printed to bash in `cosmicfish_pylib.colors.bash_colors.OKGREEN` color.

Parameters
string (*string*) – input string.

Returns
the input string with the relevant ANSI code at the beginning and at the end.

Return type
string

header(*string*)

Function that returns a string that can be printed to bash in `cosmicfish_pylib.colors.bash_colors.HEADER` color.

Parameters
string (*string*) – input string.

Returns
the input string with the relevant ANSI code at the beginning and at the end.

Return type
string

underline(*string*)

Function that returns a string that can be printed to bash in `cosmicfish_pylib.colors.bash_colors.UNDERLINE` color.

Parameters
string (*string*) – input string.

Returns
the input string with the relevant ANSI code at the beginning and at the end.

Return type
string

warning(*string*)

Function that returns a string that can be printed to bash in `cosmicfish_pylib.colors.bash_colors.WARNING` color.

Parameters
string (*string*) – input string.

Returns
the input string with the relevant ANSI code at the beginning and at the end.

Return type
string

BOLD = '\x1b[1m'

ANSI code for bold text.

ENDC = '\x1b[0m'

ANSI code to restore the bash default.

FAIL = '\x1b[91m'

ANSI color for red.

HEADER = '\x1b[95m'

ANSI color for light purple.

OKBLUE = '\x1b[94m'

ANSI color for blue.

OKGREEN = '\x1b[92m'

ANSI color for green.

UNDERLINE = '\x1b[4m'

ANSI code for underlined text.

WARNING = '\x1b[93m'

ANSI color for yellow.

cosmicfishpie.analysis.colors.**nice_colors**(*num*)

This function returns a color from a colormap defined below according to the number entered.

Parameters
num (`int` or `float`) – input number. Can be an integer or float. Notice that the colormap contains only a small numbers of colors. Even if the input is a float the output will still be one of the few colors in the colormap.

Returns
tuple of `float` containing the three RGB coordinates of the color.

Return type
tuple

cosmicfishpie.analysis.fisher_derived module

class cosmicfishpie.analysis.fisher_derived.**fisher_derived**(*derived_matrix=None*, *param_names=None*, *derived_param_names=None*, *param_names_latex=None*, *derived_param_names_latex=None*, *fiducial=None*, *fiducial_derived=None*, *file_name=None*)

Bases: object

This class contains the relevant code to define a matrix that contains the relevant information to reparametrize a Fisher matrix. Generally this is a rectangular matrix containing the Jacobian of the transformation from the original Fisher to the derived one.

Variables

- **derived_matrix** – Numpy array containing the Jacobian of the transformation between the Fisher matrix and the derived Fisher matrix. Passed to the constructor of by file.
- **path** – Absolute path of the input Jacobian matrix. Computed at initialization if passing a file.
- **name** – Name of the input Jacobian matrix. Computed at initialization if passing a file.
- **indir** – Absolute path of the directory containing the input Jacobian matrix. Computed at initialization if passing a file.
- **num_params** – Number of base parameters of the Jacobian matrix.
- **num_derived** – Number of derived parameters.
- **param_names** – Names of the base parameters. Used as the identifier of the parameters. Initialized, if possible, through a .paramnames file.
- **param_names_latex** – LaTeX names of the base parameters.
- **param_fiducial** – Numpy array with the values of the fiducial of the base parameters. Passed to the constructor or by file.
- **derived_param_names** – Names of the derived parameters. Used as the identifier of the parameters. Initialized, if possible, through a .paramnames file.
- **derived_param_names_latex** – LaTeX names of the derived parameters.
- **derived_param_fiducial** – Numpy array with the values of the fiducial of the derived parameters. Passed to the constructor or by file.

add_derived(*fisher_matrix*, *preserve_input=False*)

This function computes the derived fisher_matrix given an input Fisher matrix based on the Jacobian contained in derived_matrix.

Parameters

- **fisher_matrix** (cosmicfish_pylib.fisher_matrix.fisher_matrix) – input Fisher matrix that will be used as a base for the derived Fisher matrix.
- **preserve_input** (bool) – wether to preserve input parameters in the output Fisher. Default to false because it might lead to strage results if not used properly.

Returns

output Fisher matrix with derived parameters.

Return type
cosmicfish_pylib.fisher_matrix.fisher_matrix

`get_derived_matrix()`

Returns
the derived Jacobian matrix.

`get_derived_param_fiducial()`

Returns
the derived parameter fiducial values.

`get_derived_param_names()`

Returns
the derived parameters names.

`get_derived_param_names_latex()`

Returns
the LaTeX version of the derived parameters names.

`get_param_fiducial()`

Returns
the base parameter fiducial values.

`get_param_names()`

Returns
the base parameter names.

`get_param_names_latex()`

Returns
the LaTeX version of the base parameter names.

`load_paramnames_from_file`(*file_name=None*)

Loads the paramnames array, of a derived Fisher matrix, from a file

Parameters
file_name (string) – (optional) file name and path of the parameter names file. If file_name is None this reads the file self.name+.paramnames.

cosmicfishpie.analysis.fisher_matrix module

class cosmicfishpie.analysis.fisher_matrix.**fisher_matrix**(*fisher_matrix=None*, *param_names=None*, *param_names_latex=None*, *fiducial=None*, *file_name=None*, *name=''*)

Bases: object

This class contains the relevant code to define a fisher matrix and basic operations on it.

Variables

- **fisher_cutoff** – cutoff for the spectrum of the Fisher matrix. Parameter of the class. Starts at 10**(-9) but, if needed, is fixed during computations.

- **fisher_spectrum** – maximum condition number allowed for the Fisher matrix. Worse constrained modes that go above this value will be flattened to this value.
- **fisher_matrix** – numpy array with the fisher matrix. Passed to the constructor of by file.
- **path** – absolute path of the input Fisher matrix. Computed at initialization if passing a file or just an empty string.
- **name** – name of the input Fisher matrix. Computed at initialization if passing a file or just an empty string.
- **indir** – absoulte path of the directory of the input Fisher matrix. Computed at initialization if passing a file or just an empty string.
- **num_params** – number of parameters of the input Fisher matrix. Computed at initialization.
- **fisher_eigenvalues** – eigenvalues of the input Fisher matrix. Computed at initialization or by the PCA function.
- **fisher_eigenvectors** – eigenvectors of the input Fisher matrix. Computed at initialization or by the PCA function.
- **fisher_matrix_inv** – inverse of the input Fisher matrix. Computed at initialization or by inverse_fisher_matrix.
- **param_names** – name of the parameters of the input Fisher matrix. Passed to the constructor of by file.
- **param_names_latex** – LaTeX name of the parameters of the Fisher matrix. Passed to the constructor of by file.
- **param_fiducial** – numpy array with the values of the fiducial parameters. Passed to the constructor of by file.
- **param_names_dict** – a dictionary that maps parameter names to numbers and vice versa.

__add__(*other*)

Addition operator (+). Safeguarded agains adding Fisher matrices with different parameters and different fiducials. The addition will add parameters with the same name and append parameters with different names. Notice that if a parameter is in one of the two Fisher matrices but not in the other it will be assumed independent from the other.

__eq__(*other*)

Equality check operator (==). Ensures equality in all properties of the Fisher matrix. Notice that also name, path and indir are checked.

__ne__(*other*)

Non-equality operator (!=). Simply implemented as the inverse of the equality operator.

PCA()

This function performs the principal component analysis of the Fisher matrix returning its eigenvalues and its eigenvectors. As of now it just works just as a wrapper for numpy.

Returns
a `list` with (eigenvalues, eigenvectors) as `numpy.array`.

Return type
`list` of `numpy.array`

check_symmetric()

Assert if the Fisher matrix is symmetric or not :returns: a `bool` :rtype: `bool`

determinant()

This function returns the determinant of the Fisher matrix.

Returns
a float with the determinant of the Fisher matrix.

get_confidence_bounds(*confidence_level=0.6827*, *marginal=True*, *cache=False*)

Computes the marginal 1D confidence bounds on the Fisher parameters

Parameters

- **confidence_level** (float in [0,1]) – (optional) C.L. of the bounds. Default 68%.
- **cache** (*bool*) – (optional) wether to use cached results or compute everything again

get_fiducial(*name*)

Returns the fiducial of the parameter called name.

Parameters
name (string or a list of string) – input name or list of names of the parameters.

Returns
the fiducial or a list of fiducials corresponding to the parameter names.

Return type
float or a list of float.

get_fisher_eigenvalues()

Returns
the eigenvalues of the Fisher matrix as a numpy array.

get_fisher_eigenvectors()

Returns
the eigenvectors of the Fisher matrix.

get_fisher_inverse()

Returns
the inverse of the Fisher matrix.

get_fisher_matrix()

Returns
the fisher matrix as a numpy array.

get_param_fiducial()

Returns
the fiducial values of the parameters of the Fisher matrix.

get_param_index(*name*)

Returns the index of a parameter as specified by his name. Notice that indices starts at 0.

Parameters
name (string or a list of string) – input name or list of names of the parameters.

Returns
the index of the parameter or a list of numbers.

Return type
int or a list of int

get_param_name(*number*)

Returns the name of the parameter corresponding to the given number.

Parameters
number (int or a list of int) – number of the parameter or list of numbers. Notice that parameters are numbered starting from 1.

Returns
the name or a list of names of the parameters.

Return type
string or a list of string.

get_param_name_latex(*name*)

Returns the Latex name of the parameter called name.

Parameters
name (string or a list of string) – input name or list of names of the parameters.

Returns
the Latex name or a list of Latex names corresponding to the parameter names.

Return type
string or a list of string.

get_param_names()

Returns
the parameter names of the Fisher matrix.

get_param_names_latex()

Returns
the parameter names, in LaTeX format of the Fisher matrix.

get_param_number(*name*)

Returns the number of a parameter as specified by his name. Notice this differs from get_param_index becasue number = index+1.

Parameters
name (string or a list of string) – input name or list of names of the parameters.

Returns
the index of the parameter or a list of numbers.

Return type
int or a list of int

inverse_fisher_matrix()

Invert the Fisher matrix.

Returns
a matrix containing the inverse of the Fisher matrix.

Return type
numpy.array

load_paramnames_from_file(*file_name=None*)

Loads the paramnames array from a file.

Parameters
file_name (default: None) – (optional) file name and path of the parameter names file. If file_name is None this reads the file self.name+.paramnames.

make_symmetric(*method=None*)

Transforms a non-symmetric matrix into a symmetric matrix

protect_degenerate(*cache=True*)

Protects the Fisher matrix against degeneracies. Modifies the spectrum to ensure that the absolute value of the eigenvalues is bounded. This will make the Fisher matrix strictly positive definite. It will modify the magnitude of the worst constrained parameter combinations without modifying the degeneracies directions.

Parameters
cache (*bool*) – (optional) wether to use cached results or compute everything again

save_paramnames_to_file(*file_name=None*)

Saves the paramnames to a file.

Parameters
file_name (default: None) – (optional) file name and path of the parameter names file. If file_name is None this saves the file self.name+.paramnames.

save_to_file(*file_name*, *simple_header=False*, *file_format='.txt'*)

Saves the fisher matrix to a file. Notice that the file name has to be specified to avoid overwriting an existing fisher matrix.

Parameters
file_name – file name and path of the output fisher matrix. The file extension gets automatically added as is not needed.

set_fiducial(*fiducial*)

Function sets a new fiducial substituting the old one.

Parameters
fiducial – list containing the new fiducial.

set_fisher_matrix(*fisher_matrix*)

Function sets a new fisher matrix substituting the old one. Notice that this will reset parameter names, latex parameter names and fiducial values.

Parameters
fisher_matrix (numpy.array) – numpy.array containing the input Fisher matrix.

set_param_names(*param_names*)

Function sets a new list of param names substituting the old one. Notice that latex parameter names will be reset.

Parameters
param_names (list of string) – list containing the new parameter names.

set_param_names_latex(*param_names_latex*)

Function sets a new list of LaTeX param names substituting the old one.

Parameters
param_names_latex – list containing the new LaTeX parameter names.

translate_param_names(*trans_param_dict*)

Function substituting individual param names, effectively translating between names.

Parameters
trans_param_dict (dict of string) – dict containing the equivalence between names

cosmicfishpie.analysis.fisher_operations module

cosmicfishpie.analysis.fisher_operations.**eliminate_columns_rows**(*fisher_matrix*, *indexes*)

This function eliminates the row and columns corresponding to the given indexes from the Fisher matrix. It also deletes all the other informations like the names of the parameters. Notice that the index corresponding to the first parameter is zero.

Parameters

- **fisher_matrix** (cosmicfish_pylib.fisher_matrix.fisher_matrix) – input Fisher matrix
- **indexes** (list of int) – list of integers with the indexes to delete from the Fisher matrix

Returns
A Fisher matrix with the columns and rows deleted

Return type
cosmicfish_pylib.fisher_matrix.fisher_matrix

cosmicfishpie.analysis.fisher_operations.**eliminate_parameters**(*fisher_matrix*, *names*)

This function eliminates the row and columns corresponding to the given parameter name from the Fisher matrix. It also deletes all the other informations like the names of the parameters.

Parameters

- **fisher_matrix** (cosmicfish_pylib.fisher_matrix.fisher_matrix) – input Fisher matrix
- **names** (list of string) – list of names of the parameters to delete from the Fisher matrix

Returns
A Fisher matrix with the parameters deleted

Return type
cosmicfish_pylib.fisher_matrix.fisher_matrix

cosmicfishpie.analysis.fisher_operations.**information_gain**(*fisher_1*, *fisher_2*, *fisher_prior*, *units=0.6931471805599453*, *stat=True*)

This function computes the Fisher approximation of Kullback-Leibler information gain. For the details of the formula we refer to the CosmicFish notes.

Parameters

- **fisher_1** (cosmicfish_pylib.fisher_matrix.fisher_matrix) – first input Fisher matrix
- **fisher_2** (cosmicfish_pylib.fisher_matrix.fisher_matrix) – second input Fisher matrix
- **fisher_prior** (cosmicfish_pylib.fisher_matrix.fisher_matrix) – input Fisher matrix with the prior information.
- **units** (float) – Units of information gain. Optional by default in Bits.
- **stat** (logical) – wether to output the expected value and variance

Returns
a float with the information gain.

Return type
float

cosmicfishpie.analysis.fisher_operations.**marginalise**(*fisher_matrix*, *names*, *update_names=True*)

This function marginalises a Fisher matrix over all parameters but the ones in names. The new Fisher matrix will have the parameters specified in names, in the order specified by names. The calculation is performed in the numerically stable way.

Parameters

- **fisher_matrix** (cosmicfish_pylib.fisher_matrix.fisher_matrix) – input Fisher matrix
- **names** (list of string) – list of names of the parameters of the output Fisher matrix, in the order that will appear in the output Fisher matrix. All other parameters will be marginalized over.

Returns

A Fisher matrix with the marginalized parameters

Return type

cosmicfish_pylib.fisher_matrix.fisher_matrix

cosmicfishpie.analysis.fisher_operations.**marginalise_over**(*fisher_matrix*, *names*)

This function marginalises a Fisher matrix over the parameters in names. The new Fisher matrix will not have the parameters specified in names. The calculation is performed in the numerically stable way.

Parameters

- **fisher_matrix** (cosmicfish_pylib.fisher_matrix.fisher_matrix) – input Fisher matrix
- **names** (list of string) – list of names of the parameters over which the Fisher will be marginalised.

Returns

A Fisher matrix with the names parameters marginalized.

Return type

cosmicfish_pylib.fisher_matrix.fisher_matrix

cosmicfishpie.analysis.fisher_operations.**reshuffle**(*fisher_matrix*, *names*, *update_names=True*)

This function reshuffles a Fisher matrix. The new Fisher matrix will have the parameters specified in names, in the order specified by names. Can be used to delete parameters, change their order or extract the Fisher for some parameters without marginalizing over the others.

Parameters

- **fisher_matrix** (cosmicfish_pylib.fisher_matrix.fisher_matrix) – input Fisher matrix
- **names** (list of string) – list of names of the parameters that are desired in the output Fisher matrix, in the desired order.

Returns

A Fisher matrix with the new parameters

Return type

cosmicfish_pylib.fisher_matrix.fisher_matrix

cosmicfishpie.analysis.fisher_plot_analysis module

class cosmicfishpie.analysis.fisher_plot_analysis.**CosmicFish_FisherAnalysis**(*fisher_list=None*, *fisher_path=None*, *search_fisher_guess=False*, *with_derived=True*)

Bases: `object`

This class takes care of handeling a set of Fisher matrices with plotting in mind. This class is meant to hold a list of Fisher matrices and have defined on this list a set of vectorized operations. For now no caching is implemented.

Variables

- **fisher_list** – list of Fisher matrices `cosmicfish_pylib.fisher_matrix.fisher_matrix`
- **fisher_name_list** – list of names of the Fisher matrices. The names are used as the unique identifier of the Fisher matrix. No double name is enforced.

add_fisher_matrix(*fisher*)

Add a set of Fisher matrices to the already existing set. Rejects Fisher matrices if the name is double i.e. the name is the unique identifier of the Fisher matrix. Checks wether the elements that are passed are really Fisher matrices.

Parameters

fisher_list (`cosmicfish_pylib.fisher_matrix.fisher_matrix` or `list` of `cosmicfish_pylib.fisher_matrix.fisher_matrix`) – Fisher matrix or list of Fisher matrices.

compute_ellipse(*params1=None*, *params2=None*, *confidence_level=0.6827*, *names=None*, *num_points=100*)

Function that computes the (2D) ellipses for a given parameters combination. Returns a dictionary with all the meaningul information about the ellipses.

Parameters

- **params1** (a `string` or a `list` of `string`) – name of the first parameter or list of names of parameters.
- **params2** (default: `None`) – name of the second parameter or list of names of parameters.
- **confidence_level** (`float`) – (optional) Confidence Level of the bounds. Default 68%.
- **names** (a `string` or a `list` of `string`) – names of the Fisher matrices.
- **num_points** (`int`) – number of (x,y) points.

Returns

a dictionary mapping name and parameters to a tuple of: [x, y, [fiducial_x, fiducial_y, coeff_a, coeff_b, theta_0]]

Return type

`dict`

compute_gaussian(*params=None*, *confidence_level=0.6827*, *names=None*, *num_points=100*, *normalized=False*, *nice_bounds=True*)

Function that computes the (1D) gaussian distribution of a given parameter. Returns a dictionary with all the meaningul information about the gaussian.

Parameters

- **params** (a string or a list of string) – name of the parameter or list of names of parameters.
- **confidence_level** (float) – (optional) Confidence Level of the bounds. Default 68%.
- **names** (a string or a list of string) – names of the Fisher matrices.
- **num_points** (int) – number of (x,y) points.
- **normalized** (bool) – wether the distribution is normalized or not.
- **nice_bounds** (bool) – wether the x range is properly rounded to be nice or not.

Returns
a dictionary mapping name and parameter to a tuple of: [x, y, [fiducial,sigma]]

Return type
dict

compute_plot_range(*params=None*, *confidence_level=0.6827*, *range_factors={'default': 1.0}*, *names=None*, *nice=True*, *dimensions=1*)

Function that computes a meaningfull plot range for the plots involving the specified parameters and the specified Fisher names.

Parameters

- **params** (a string or a list of string) – name of the parameter or list of names of parameters.
- **confidence_level** (float) – (optional) Confidence Level of the bounds. Default 68%.
- **names** (a string or a list of string) – names of the Fisher matrices.
- **nice** (bool) – wether the number is properly rounded to be nice.

Returns
a dictionary of name and bounds

Return type
dict

delete_fisher_matrix(*names=None*)

Delete the fisher matrix or the fisher matrices in names from the Fisher list.

Parameters
names (a string or a list of string) – names of the Fisher matrices to delete.

get_fisher_list()

Returns
the list fisher matrices.

get_fisher_matrix(*names=None*)

Returns the list of Fisher matrices corresponding to the given names.

Parameters
names (a string or a list of string) – names of the Fisher matrices.

Returns
a list containing the desired Fisher matrices. Notice if the name is not found in the list no error is risen and and the entrance is just ignored.

Return type
a list of cosmicfish_pylib.fisher_matrix.fisher_matrix

`get_fisher_name_list()`

Returns
the list fisher matrices names. These are the unique identifiers of the list.

`get_parameter_latex_names(names=None)`

Returns a dictionary mapping parameter names and latex parameter names.

Parameters
names (a `string` or a `list` of `string`) – names of the Fisher matrices.

Returns
a dictionary containing parameter names and the LaTeX parameter names.

Return type
`dict`

`get_parameter_list(names=None)`

Returns the list of parameter names of all the matrices identified in names.

Parameters
names (a `string` or a `list` of `string`) – names of the Fisher matrices.

Returns
a list containing the parameter names.

Return type
a `list` of `string`

`marginalise(params, names=None, update_names=True)`

Marginalise all the Fisher matrices over all the parameters that are not in params.

Parameters

- **params** (a `string` or a `list` of `string`) – list of names of the parameters of the output Fisher matrix, in the order that will appear in the output Fisher matrix. All other parameters will be marginalized over.
- **names** (a `string` or a `list` of `string`) – names of the Fisher matrices.

Returns
a new Fisher list with the reshuffled Fishers

Return type
a `cosmicfish_pylib.fisher_plot_analysis.CosmicFish_FisherAnalysis`

`reshuffle(params, names=None, update_names=True)`

Reshuffles all the Fisher matrices.

Parameters

- **params** (a `string` or a `list` of `string`) – parameters to reshuffle.
- **names** (a `string` or a `list` of `string`) – names of the Fisher matrices.

Returns
a new Fisher list with the reshuffled Fishers

Return type
a `cosmicfish_pylib.fisher_plot_analysis.CosmicFish_FisherAnalysis`

search_fisher_path(*fisher_path*, *search_fisher_guess=False*, *with_derived=True*)

Searches a path for fisher matrices. Will detect wether fisher_path contains directly the paths to the Fisher files or folder. If a list of folders is passed all the folders will be searched, first for Fisher matrices then for derived Fisher matrices.

Parameters

- **fisher_path** (string or list of string) – path or list of paths to search for Fisher matrices. If this contains Fisher matrices files those are imported as well.
- **search_fisher_guess** (bool) – wether to guess the Fisher matrix name or not. Guessing assumes that the Fisher matrices have 'fisher_matrix' and '.dat' in the name as happens with Fisher matrices produced with CosmicFish.
- **with_derived** (bool) – wether to search for derived Fisher matrices to add to the base Fisher that are found.

cosmicfishpie.analysis.fisher_plotting module

class cosmicfishpie.analysis.fisher_plotting.**fisher_plotting**(***options*)

Bases: object

This class uses the cosmicfish_pylib classes to generate contour plots using getdist

compare_errors(*options={}*)

get_FoM(*ind*)

load_gaussians()

matrix_ratio(*r_fishers_list=None*, *tick_labels=None*, *plot_title=None*, *ratio_mat=None*, *filename=None*, *savefig=True*)

param_limits_bounds(*axis_custom_factors=None*)

plot_fisher(***kwargs*)

Generates a triangle plot based on loaded gaussian data and specified parameters.

Parameters

****kwargs** – Keyword arguments for customizing the plot. axis_custom_factors (optional): Custom factors for axis limits. Default is None. filled (optional): Boolean value indicating whether contour plots should be filled or not. Default is True. contour_args (optional): List of dictionaries specifying contour plot arguments. Default is [{'alpha':0.9}]. legend_loc (optional): Location of the legend in the plot. Default is 'upper right'. dpi (optional): Dots per inch for saving the plot to a file. Default is 300. file_format (optional): File format for saving the plot. Default is '.pdf'. marker_color (optional): Color of the axis markers. Default is 'black'. axes_fontsize (optional): Font size for the axes labels. Default is 20. legend_fontsize (optional): Font size for the legend labels. Default is 20. figure_legend_frame (optional): Frame thickness for the figure legend. Default is 20. axes_labelsize (optional): Font size for the axes tick labels. Default is 20. figure_facecolor (optional): Facecolor of the figure. Default is 'white'.

Returns

None

Raises

None –

Usage:

instance_name.plot_fisher(axis_custom_factors=create_factors(),
filled=True, contour_args=[{'alpha':0.7}], legend_loc='lower left', dpi=150, file_format='.png', marker_color='red', axes_fontsize=16, legend_fontsize=18, figure_legend_frame=10, axes_labelsize=14)

read_fisher_matrices()

cosmicfishpie.analysis.plot_comparison module

cosmicfishpie.analysis.plot_comparison.**add_colorbar**(*im*, *aspect=30*, *pad_fraction=0.5*, ***kwargs*)

Add a vertical color bar to an image plot.

cosmicfishpie.analysis.plot_comparison.**calc_y_range**(*axis*, *yrang=None*)

cosmicfishpie.analysis.plot_comparison.**matrix_plot**(*matrix*, *xlabel='Ratio'*, *ticklabels=None*, *filename='matrixplot.png'*, *figsize=(9, 9)*, *colormap=<matplotlib.colors.ListedColormap object>*, *savefig=True*, *dpi=200*)

cosmicfishpie.analysis.plot_comparison.**og_plot_shades**(*ax*, *x_arr*, *x_names*, *lighty_arr=None*, *darky_arr=None*, *mats_labels=None*, *lightdark_names=['marg.', 'unmarg.']*, *cols=[]*, *plotdark=True*, *plotlight=True*, *yrang=None*, *x_limpad=0.2*, *fish_leg_loc='upper left'*, *LW=2*, *colordark='darkgrey'*, *colorlight='lightgrey'*, *alpha=0.7*, *light_hatch='/'*, *patches_legend_loc='upper right'*, *patches_legend_fontsize=16*, *dots_legend_fontsize=20*, *ylabelfontsize=20*, *ncol_legend=None*, *legend_title_fontsize=None*, *legend_title=None*, *y_label='Differences'*, *yticklabsize=18*, *xticklabsize=18*, *xtickfontsize=15*, *xticksrotation=0*)

cosmicfishpie.analysis.plot_comparison.**plot_shades**(*ax*, *x_arr*, *x_names*, *lighty_arr=None*, *darky_arr=None*, *mats_labels=None*, *lightdark_names=['marg.', 'unmarg.']*, *cols=[]*, *plotdark=True*, *plotlight=True*, *yrang=None*, *x_limpad=0.2*, *fish_leg_loc='upper left'*, *LW=2*, *colordark='darkgrey'*, *colorlight='lightgrey'*, *alpha=0.7*, *light_hatch='/'*, *patches_legend_loc='upper right'*, *patches_legend_fontsize=16*, *dots_legend_fontsize=20*, *ylabelfontsize=20*, *ncol_legend=None*, *legend_title_fontsize=None*, *legend_title=None*, *y_label='Differences'*, *yticklabsize=18*, *xticklabsize=18*, *xtickfontsize=15*, *xticksrotation=0*)

cosmicfishpie.analysis.plot_comparison.**ploterrs**(*fishers_list*, *fishers_name*, *parstoplot=None*, *parsnames_latex=None*, *marginalize_pars=True*, *plot_style='original'*, *outpath-file='/home/docs/checkouts/readthedocs.org/user_builds/cosmicfishpie/c*
plot_marg=True, *plot_unmarg=True*, *yrang=None*, *figsize=(10, 6)*, *fish_leg_loc='lower left'*, *dpi=400*, *savefig=True*, *y_label='Errors'*, *ncol_legend=None*, *legend_title_fontsize=None*, *legend_title=None*, *yticklabsize=20*, *xticklabsize=15*, *patches_legend_fontsize=20*, *dots_legend_fontsize=20*, *xtickfontsize=18*, *ylabelfontsize=20*, *xticksrotation=0*, *save_error=False*, *transform_latex_dict={}*, *figure_title=''*)

cosmicfishpie.analysis.plot_comparison.**process_fish_errs**(*fishers_list*, *fishers_name*, *parstoplot=None*, *parsnames_latex=None*, *marginalize_pars=True*, *print_errors=True*, *transform_latex_dict={}*)

cosmicfishpie.analysis.utilities module

cosmicfishpie.analysis.utilities.**CosmicFish_write_header**(*name*)

This function prints to screen the CosmicFish header. To be called at the beginning of the applications.

Parameters

name – string that contains the name of the program. This will be printed along the CosmicFish header.

cosmicfishpie.analysis.utilities.**confidence_coefficient**(*confidence_level*, *dimensions=1*)

This function returns the number of sigmas given a confidence level. See page 815 of Numerical Recipes, Press et al., 2007 Uses the inverse CDF of the chi squared distribution :type confidence_level: :param confidence_level: desired confidence level. Between 0 and 1. :type confidence_level: `float` :return: the coefficient (number of sigmas) for the desired confidence level. :rtype: `float`

cosmicfishpie.analysis.utilities.**find_nearest**(*array*, *value*)

This function finds the index of the element in array which is nearest to ValueError :type array: :param array: an array of numbers :type value: :param value: a number :return: the index in array nearest to value :rtype: int

cosmicfishpie.analysis.utilities.**grouper**(*n*, *iterable*, *fillvalue=None*)

This small function regroups a list in sub lists of n elements

Parameters

- **n** – an element or a list of elements
- **iterable** – input list
- **fillvalue** (default: `None`) – value to put to fill if no element is present

Returns

a list of list containing grouped elements

Return type

list

cosmicfishpie.analysis.utilities.**make_list**(*elements*)

Checks if elements is a list. If yes returns elements without modifying it. If not creates and return a list with elements inside.

Parameters
elements – an element or a list of elements

Returns
a list containing elements if elements is not a list, elements otherwise.

Return type
list

cosmicfishpie.analysis.utilities.**mant_exp_to_num**(*mant_exp*)

This function returns a float built with the given (base 10) mantissa and exponent.

Parameters
mant_exp (*tuple*) – (mantissa, exponent) a tuple of two int with the mantissa and the exponent of the input number.

Returns
output number built as mantissa*10**exponent.

Return type
float

cosmicfishpie.analysis.utilities.**mkdirp**(*dirpath*)

This function creates the directory dirpath if it is not found :type dirpath: :param dirpath: string with the path of the directory to be created :return: None :rtype: NoneType

cosmicfishpie.analysis.utilities.**nice_number**(*num*, *mode=1*, *digits=1*)

This function returns a nice number built with num. This is useful to build the axes of a plot. The nice number is built by taking the first digit of the number.

Parameters

- **num** (float or int) – input number
- **mode** (int) – (optional) operation to use to build the nice number

  0 – use ceil
  1 – use round
  2 – use floor

- **digits** (int) – input number of digits to keep

Returns
a nice number!

Return type
float

cosmicfishpie.analysis.utilities.**num_to_mant_exp**(*num*)

This function returns the (base 10) exponent and mantissa of a number.

Parameters
num (int or float) – input number.

Returns
tuple (mantissa, exponent) of `int` containing the mantissa and the exponent of the input number.

Return type
tuple

`cosmicfishpie.analysis.utilities.`**`print_table`**(*table*)

This function prints on the screen a nicely formatted table.

Parameters
`table` – a 2D list that should be printed on the screen.

`cosmicfishpie.analysis.utilities.`**`rel_median_error`**(*array*, *percentage=True*)

This function returns the percentage difference compared to median of an array for each element :type array: :param array: numpy array of numbers :optional_param percentage: if set to True, returns percentage difference. Default: True :return: difference of each element compared to median of array :rtype: Numpy array

`cosmicfishpie.analysis.utilities.`**`significant_digits`**(*num_err*, *mode=1*, *digits=1*)

This function returns the number in num_err at the precision of error.

Parameters

- **`num_err`** (*tuple*) – (number, error) input number and error in a tuple.
- **`mode`** (`int`) – (optional) operation to use to build the number

  0 – use ceil
  1 – use round
  2 – use floor

- **`digits`** (`int`) – input number of digits to keep

Returns
a number with all the significant digits according to error

Return type
`float`

Module contents

Cosmology Module

Provide some basic description of the theory module.

3.1.4 cosmicfishpie.cosmology package

Submodules

cosmicfishpie.cosmology.cosmology module

COSMOLOGY

This module contains useful cosmological functions.

class cosmicfishpie.cosmology.cosmology.**boltzmann_code**(*cosmopars*, *code='camb'*)

Bases: object

camb_results(*camb*)

camb_setparams(*cosmopars*, *camb*)

Sets the parameters for the CAMB (Code for Anisotropies in the Microwave Background) computation.

Parameters

- **cosmopars** (dict) – A dictionary containing the cosmological parameters.
- **camb** – The CAMB object.

Returns
None

The function sets the CAMB parameters based on the provided input and performs a basis change if necessary. It then sets the matter power spectrum and additional options for the CAMB computation.

Note: This function assumes that the boltzmann_cambpars and settings dictionaries are available.

Example usage:
cosmopars = {'gamma': 0.545, 'k_per_logint': 0.1, 'kmax': 10, 'accurate_massive_neutrino_transfers': True} camb = CAMB() camb_set_params(cosmopars, camb)

changebasis_camb(*cosmopars*, *camb*)

changebasis_class(*cosmopars*, *Class*)

class_results(*Class*)

class_setparams(*cosmopars*, *Class*)

static print_camb_params(*cambpars*, *feedback=1*)

static print_class_params(*classpars*, *feedback=1*)

rescale_LP(*cambpars*, *camb*, *insigma8*)

set_cosmicfish_defaults()

Fills up default values in the cosmopars dictionary if the values are not found.

Parameters
self (object) – The instance of the class.

Returns
None

hardcoded_Neff = 3.044

hardcoded_neutrino_mass_fac = 94.07

class cosmicfishpie.cosmology.cosmology.**cosmo_functions**(*cosmopars*, *input=None*)

Bases: object

E_hubble(z)

E(z) dimensionless Hubble function

Parameters
z (float) – redshift

Returns
Dimensionless E(z) Hubble function values at the redshifts of the input redshift

Return type
float

Hubble(z, *physical=False*)

Hubble function

Parameters

- **z** (float) – redshift
- **physical** (bool) – Default False, if True, return H(z) in (km/s/Mpc).

Returns
Hubble function values (Mpc^-1) at the redshifts of the input redshift

Return type
float

Omegam_of_z(z)

Omega matter fraction as a function of redshift

Parameters
z (float) – redshift

Returns
Omega matter (total) at the redshifts of the input redshift z

Return type
float

Note: Assumes standard matter evolution Implements the following equation:

$$Omega_m(z) = Omega_{m,0} * (1+z)^3 / E^2(z)$$

Pcb(z, k, *nonlinear=False*)

Compute the power spectrum of the clustering matter species (CB) at a given redshift and wavenumber.

Parameters

- **self** – An instance of the current class.
- **z** – The redshift at which to compute the CB power spectrum.
- **k** – The wavenumber at which to compute the CB power spectrum in 1/Mpc.
- **nonlinear** (bool, optional) – If True, include nonlinear effects in the computation. Default is False.

Returns
The value of the CB power spectrum at the given redshift and wavenumber.

Pmm(*z*, *k*, *nonlinear=False*)

Compute the power spectrum of the total matter species (MM) at a given redshift and wavenumber.

Parameters

- **self** – An instance of the current class.
- **z** – The redshift at which to compute the MM power spectrum.
- **k** – The wavenumber at which to compute the MM power spectrum in 1/Mpc.
- **nonlinear** (`bool, optional`) – If True, include nonlinear effects in the computation. Default is False.

Returns
The value of the MM power spectrum at the given redshift and wavenumber.

Return type
float

SigmaMG(*z*, *k*)

angdist(*z*)

Angular diameter distance

Parameters
z (`float`) – redshift

Returns
Angular diameter distance values at the redshifts of the input redshift

Return type
`float`

cmb_power(*lmin*, *lmax*, *obs1*, *obs2*)

comoving(*z*)

Comoving distance

Parameters
z (`float`) – redshift

Returns
Comoving distance values at the redshifts of the input redshift

Return type
`float`

f_growthrate(*z*, *k=None*, *gamma=False*, *tracer='matter'*)

Growth rate in LCDM gamma approximation

Parameters
z (`float`) – redshift

Returns
Growth rate values at the redshifts of the input redshift, using self.gamma as gamma value.

Return type
`float`

Note: Implements the following equation:

$$f(z) = Omega_m(z)^\gamma$$

---

fsigma8_of_z(*z*, *k=None*, *gamma=False*, *tracer='matter'*)

Growth rate in LCDM gamma approximation

Parameters
z (float) – redshift

Returns
Growth rate values at the redshifts of the input redshift, using self.gamma as gamma value.

Return type
float

Note: Implements the following equation:

$$f(z) = Omega_m(z)^\gamma$$

---

growth(*z*, *k=None*)

Growth factor

Parameters
z (float) – redshift

Returns
Growth factor values at the redshifts of the input redshift

Return type
float

matpow(*z*, *k*, *nonlinear=False*, *tracer='matter'*)

Calculates the power spectrum of a given tracer quantity at a specific redshift and wavenumber.

Parameters

- **z** (float) – The redshift of interest.
- **k** (array_like) – An array of wavenumbers at which to compute the power spectrum. These must be in units of 1/Mpc and should be sorted in increasing order.
- **nonlinear** (bool, *optional*) – A boolean indicating whether or not to include nonlinear corrections to the matter power spectrum. The default value is False.
- **tracer** (str, *optional*) – A string indicating which trace quantity to use for computing the power spectrum. If this argument is "matter" or anything other than "clustering", the power spectrum functions *Pmm* will be used to compute the power spectrum. If the argument is "clustering", the power spectrum function *Pcb* will be used instead. The default value is "matter".

Returns
Array containing the calculated power spectrum values.

Return type
np.ndarray

Warning: If *tracer* is not "matter" or "clustering", a warning message is printed to the console saying the provided tracer was not recognized and the function defaults to using *Pmm* to calculate the power spectrum of matter.

nonwiggle_pow(*z*, *k*, *nonlinear=False*, *tracer='matter'*)

Calculate the power spectrum at a specific redshift and wavenumber, after smoothing to remove baryonic acoustic oscillations (BAO).

Parameters

- **z** – The redshift of interest.
- **k** – An array of wavenumbers at which to compute the power spectrum. Must be in units of Mpc^-1/h. Should be sorted in increasing order.
- **nonlinear** (default: `False`) – Whether to include nonlinear corrections to the matter power spectrum. Default is False.
- **tracer** (default: `'matter'`) – Which perturbations to use for computing the power spectrum. Options are 'matter' or 'clustering'. Default is 'matter'.

Returns

An array of power spectrum values corresponding to the input wavenumbers. Units are (Mpc/h)^3.

Note: This function computes the power spectrum of a given tracer quantity at a specific redshift, using the matter power spectrum function *matpow*. It then applies a Savitzky-Golay filter to smooth out the BAO features in the power spectrum. This is done by first taking the natural logarithm of the power spectrum values at a set of logarithmic wavenumbers spanning from *kmin_loc* to *kmax_loc*. The smoothed power spectrum is then returned on a linear (not logarithmic) grid of wavenumbers given by the input array *k*.

---

sigma8_of_z(*z*, *tracer='matter'*)

sigma_8

Parameters

- **z** (`float`) – redshift
- **tracer** (`String`) – either 'matter' if you want sigma_8 calculated from the total matter power spectrum or 'clustering' if you want it from the Powerspectrum with massive neutrinos substracted

Returns

The Variance of the matter perturbation smoothed over a scale of 8 Mpc/h

Return type

`float`

c = 299792.458

class cosmicfishpie.cosmology.cosmology.**external_input**(*cosmopars*, *fiducialcosmopars={}*, *external={}*, *extra_settings={}*)

Bases: `object`

calculate_interpol_results(*parameter_string='fiducial_eps_0'*)

get_param_string_from_value(*cosmopars*)

load_txt_files(*parameter_string='fiducial_eps_0'*)

cosmicfishpie.cosmology.nuisance module

Nuisance

This module contains nuisance parameter functions.

class cosmicfishpie.cosmology.nuisance.**Nuisance**

Bases: object

IA(*IApars*, *cosmo*)

Intrinsic Alignment

Parameters
z – float redshift

Returns

- float: Value of IA window at redshift z

Notes
Implements the following equation:

$$W_i^{IA} = -\frac{\mathcal{A}_{\rm IA} C_{\rm IA} \Omega_m \mathcal{F}_{\rm IA}}{D(z)} \frac{n_i(z)}{\bar{n}} \frac{H(z)}{c}$$

IM_THI_noise()

" Reads from file for a given survey

IM_bias(*z*)

IM 21cm HI bias function from http://arxiv.org/abs/2006.05996

IM_bias_at_zm()

IM_zbins()

Reads from file for a given survey

IM_zbins_mids()

bterm_z_key(*z_ind*, *z_mids*, *fiducosmo*, *bias_sample='g'*)

extra_Pshot_noise()

gcph_bias(*biaspars*, *ibin=1*)

Galaxy bias

Parameters
z (array) – redshift

Returns
Value galaxy bias at redshift z

Return type
float

gcsp_bias()

Galaxy bias for the galaxies used in spectroscopic Galaxy Clustering

Returns

- object
- Interpolating function of bias for the redshift bins given

Note: Reads from file and interpolates for a given survey

gcsp_bias_at_zm()

gcsp_dndz()

Reads from file for a given survey

gcsp_zbins()

Reads from file for a given survey

gcsp_zbins_mids()

luminosity_ratio()

Luminosity ratio

Parameters
z (float) – redshift

Returns
Value of the luminosity ratio

Return type
float

Note: Reads from file and interpolates the following quantity:

$$\frac{< L(z) >}{L_*(z)}$$

Module contents

Cosmology Module

Provide some basic description of the theory module.

3.1.5 cosmicfishpie.fishermatrix package

Submodules

cosmicfishpie.fishermatrix.config module

cosmicfishpie.fishermatrix.config.**init**(*options={}*, *specifications={}*, *observables=None*, *freepars=None*, *extfiles=None*, *fiducialpars=None*, *biaspars=None*, *photopars=None*, *IApars=None*, *PShotpars=None*, *Spectrobiaspars=None*, *spectrononlinearpars=None*, *IMbiaspars=None*, *surveyName='Euclid'*, *cosmoModel='w0waCDM'*, *latexnames=None*)

This class is to handle the configuration for the fishermatrix computation as well as the fiducial parameters. It then gives access to all global variables

Parameters

- **options** (dict, *optional*) – A dictionary that contains the global options for the calculation of the fishermatrix. A list of all possible keys are found below
- **specifications** (dict, *optional*) – A dictionary containing the survey specifications. Defaults to the specifications in the *.yaml* of the survey specifications
- **observables** (list, *optional*) – A list of strings for the different observables
- **freepars** (dict, *optional*) – A dictionary containing all cosmological parameters to be varied and their corresponding rel. step sizes
- **extfiles** (dict, *optional*) – A dictionary containing the path to the external files as well as how all the names of the files in the folder correspond to the cosmological quantities, units etc.
- **fiducialpars** (dict, *optional*) – A dictionary containing the fiducial cosmological parameters
- **biaspars** (dict, *optional*) – A dictionary containing the specifications for the galaxy biases of the photometric probe
- **photopars** (dict, *optional*) – A dictionary containing specifications for the window function's galaxy distribution
- **IApars** (dict, *optional*) – A dictionary containing the specifications for the intrinsic alignment effect in cosmic shear
- **PShotpars** (dict, *optional*) – A dictionary containing the values of additional shotnoise of the spectroscopic probes
- **Spectrobiaspars** (dict, *optional*) – A dictionary containing the specifications for the galaxy biases of the spectroscopic probe
- **spectrononlinearpars** (dict, *optional*) – A dictionary containing the values of the non linear modeling parameters of the spectroscopic probe
- **IMbiaspars** (dict, *optional*) – A dictionary containing the specifications for the galaxy biases of the spectroscopic intensity mapping probe
- **surveyName** (str, *optional*) – String of the name of the survey for which the forecast is done. Defaults to Euclid with optimistic specifications
- **cosmoModel** (str, *optional*) – A string of the name of the cosmological model used in the calculations. Defaults to flat "w0waCDM" cosmology
- **latexnames** (dict, *optional*) – A dictionary that contains the Latex names of the cosmological parameters
- **Options** –
- **-------** –
- **camb_path** (str) – Path to camb. Defaults to the camb in your current environment
- **specs_dir** (str) – Path to the survey specifications. Defaults to the *survey_specifications* folder in the home directory of cosmicfishpie
- **survey_names** (str) – String of the names of the survey. Defaults to the name passed in the parameter *surveyName*
- **derivatives** (str) – String of the name of the derivative method. Either *3PT*, *4PT_FWD*, *STEM* or *POLY*. Defaults to *3PT*
- **nonlinear** (bool) – If True will do nonlinear corrections in the computation of the different observables. Defaults to True

- **nonlinear_photo** (bool) – If True will use the nonlinear power spectrum when calculation the angular power spectrum of the photometric probe. Defaults to True
- **bfs8terms** (bool) – If True will expand the observed power spectrum with σ_8 to match the IST:F recipe. Defaults to True
- **vary_bias_str** (str) – The root of the name of the bias parameters that should be varied in the spectroscopic probe. Defaults to 'lnb'
- **AP_effect** (bool) – If True the Alcock-Paczynsk effect will be considered in the spectroscopic probe. Defaults to True
- **FoG_switch** (bool) – If True the finger of god effect will be modelled in the observed power spectrum of the spectroscopic probe. Defaults to True
- **GCsp_linear** (bool) – If True there will be no nonlinear modelling used in the spectroscopic probe. Defaults to False
- **fix_cosmo_nl** (bool) – If True and the nonlinear modeling parameters are not varied, then they will be fixed to the values computed in the fiducial cosmology. Else they will be recomputed in each sample cosmology. Defaults to True
- **Pshot_nuisance_fiducial** (float) – Value of the fiducial additional shotnoise of the spectroscopic probe. Defaults to 0.0
- **pivot_z_IA** (float) – Redshift on which the power law dependance in the eNLA model of intrinsic alignment should be normalized to. Defaults to 0.0
- **accuracy** (int) – Global rescaling of the amount of points that are used in internal calculations or interpolations for the probes. Defaults to 1
- **feedback** (int) – Number indicating the verbosity of the output. Higher numbers generally mean more output. Defaults to 2
- **activateMG** (bool) – If True will also consider modified gravity in the calculations of the observables. Defaults to False
- **external_activateMG** (bool) – If True while reading the external Files will also look for files specific to modified gravity models
- **cosmo_model** (str) – A string of the name of the cosmological model used in the calculations. Defaults to what was passed in the parameter *cosmoModel*
- **outroot** (str) – The name of the output files are always starting with CosmicFish_v<Version number>_<outroot>. Defaults to 'default_run'
- **code** (str) – String of the method to obtain the cosmological functions such as the power spectrum. Either 'camb', 'class' or 'external'. Defaults to 'camb'
- **memorize_cosmo** (bool) – If True will save and load cosmologies that have already been computed in the cache. Defaults to False
- **results_dir** (str) – Name of the folder in which the results should be saved. If the folder does not exist will create a new one. Defaults to './results'
- **boltzmann_yaml_path** (str) – Path to the configurations for the Einstein-Boltzmann solvers. Defaults to the *boltzmann_yaml_files* folder in the home directory of cosmicfishpie
- **class_config_yaml** (str) – Path to the configurations for class. Defaults to '<boltzmann_yaml_path>/class/default.yaml'
- **camb_config_yaml** (str) – Path to the configurations for camb. Defaults to '<boltzmann_yaml_path>/camb/default.yaml'

- **fishermatrix_file_extension** (str) – Specifies in what kind of file the result Fisher matrix should be saved. Defaults to '.txt'
- **savgol_polyorder** (float) – Order of the Savitzky-Golay filter. Defaults to 3 matching the IST:F recipe
- **savgol_width** (float) – Width of the Savitzky-Golay filter. Defaults to ~1.359
- **savgol_internalsamples** (float) – How many points on a logarithmic k axis should be taken to apply the Savitzky-Golay filter to. Defaults to 800
- **savgol_internalkmin** (float) – Lowest wavenumber that should be used when calculating when applying the Savitzky-Golay filter. Defaults to 1e-3. Together with the other defaults this is matching the IST:F recipe
- **eps_cosmopars** (float) – The default rel. step size of the cosmological parameters if none have been passed. Defaults to 1e-2
- **eps_gal_nuispars** (float) – The default rel. step size of the bias parameters if none have been passed. Defaults to 1e-4
- **GCsp_Tracer** (str) – What power spectrum should be used as the underlying power spectrum the spectroscopic probe's galaxy clustering traces. Either 'matter' for the total matter spectrum or 'clustering' for CDM+baryons. Defaults to 'matter'
- **GCph_Tracer** (str) – What power spectrum should be used as the underlying power spectrum the photometric probe's galaxy clustering traces. Either 'matter' for the total matter spectrum or 'clustering' for CDM+baryons. Defaults to 'matter'
- **ShareDeltaNeff** (bool) – If True, the variation of the cosmological parameter Neff is understood as a rescaling of the decoupling temperature of neutrinos. If False any additional Neff is accounted for as additional massless relics.

cosmicfishpie.fishermatrix.config.**settings**

A dictionary containing all the global options passed as well as the default values of the ones not passed

Type
dict, global

cosmicfishpie.fishermatrix.config.**external**

A dictionary containing all paths to the external files, how all the names of the files in the folder correspond to the cosmological quantities, the units etc. Will be None if no external files are given

Type
dict, global

cosmicfishpie.fishermatrix.config.**input_type**

String of the method to obtain the cosmological functions such as the power spectrum. Either 'camb', 'class' or 'external'

Type
str, global

cosmicfishpie.fishermatrix.config.**specs**

A dictionary containing the survey specifications

Type
dict, global

cosmicfishpie.fishermatrix.config.**boltzmann_classpars**

A dictionary containing the configuration, precision parameters, and fixed cosmological parameters for class

Type
dict, global

cosmicfishpie.fishermatrix.config.**boltzmann_cambpars**

A dictionary containing the configuration, precision parameters, and fixed cosmological parameters for camb

Type
dict, global

cosmicfishpie.fishermatrix.config.**survey_equivalence**

Part of the Parser, will correspond the passed survey to the name of a specifications file

Type
callable, global

cosmicfishpie.fishermatrix.config.**obs**

A list of strings for the different observables

Type
list, global

cosmicfishpie.fishermatrix.config.**freeparams**

A dictionary containing all names and the corresponding rel. step size for all parameters

Type
dict, global

cosmicfishpie.fishermatrix.config.**fiducialparams**

A dictionary containing all fiducial values for the cosmological parameters

Type
dict, global

cosmicfishpie.fishermatrix.config.**fiducialcosmo**

An instance of *cosmo_functions* of the fiducial cosmology, this contains all the cosmological functions and quantities computed from them

Type
cosmicfishpie.cosmology.cosmology.cosmo_functions, global

cosmicfishpie.fishermatrix.config.**biasparams**

a dictionary containing the specifications for the galaxy biases of the photometric probe

Type
dict, global

cosmicfishpie.fishermatrix.config.**photoparams**

A dictionary containing specifications for the window function's galaxy distribution of the photometric probe

Type
dict, global

cosmicfishpie.fishermatrix.config.**IAparams**

A dictionary containing the specifications for the intrinsic alignment effect in cosmic shear of the photometric probe

Type
dict, global

cosmicfishpie.fishermatrix.config.**PShotparams**

A dictionary containing the values of the additional shot noise per bin dictionary containing the values of the additional shot noise per bin for the spectroscopic probe

Type
dict, global

cosmicfishpie.fishermatrix.config.**Spectrobiasparams**

A dictionary containing the specifications for the galaxy biases of the spectroscopic probe

Type
dict, global

cosmicfishpie.fishermatrix.config.**Spectrononlinearparams**

A dictionary containing the values of the non linear modeling parameters entering FOG and the dewiggling weight per bin for the spectroscopic probe

Type
dict, global

cosmicfishpie.fishermatrix.config.**IMbiasparams**

A dictionary containing the specifications for the line intensity biases of the spectroscopic probe

Type
dict, global

cosmicfishpie.fishermatrix.config.**latex_names**

A dictionary with all cosmological + nuisance parameters and their corresponding name for the LaTeX labels.

Type
dict, global

cosmicfishpie.fishermatrix.cosmicfish module

MAIN

This is the main engine of CosmicFish.

class cosmicfishpie.fishermatrix.cosmicfish.**FisherMatrix**(*options={}*, *specifications={}*, *observables=None*, *freepars=None*, *extfiles=None*, *fiducialpars=None*, *biaspars=None*, *photopars=None*, *spectrononlinearpars=None*, *IApars=None*, *surveyName='Euclid'*, *cosmoModel='w0waCDM'*, *latexnames=None*, *parallel='False'*, *parallel_cpus=4*)

Bases: object

CMB_fishermatrix(*noisy_cls=None*, *covmat=None*, *derivs=None*, *cmb_obj=None*)

compute(*max_z_bins=None*)

This function will compute the Fisher information matrix and export using the specified settings.

Parameters
max_z_bins (int) – For the spectroscopic probe what the highest redshift bin that should be considered in the computation of the Fisher

Returns
A instance of fisher_matrix containing the calculated Fisher matrix as well as parameter names, settings, etc

Return type
cosmicfishpie.analysis.fisher_matrix

compute_binned_derivs(*z_arr*)

This computes the derivatives of the observed power spectrum for all redshift bins

Parameters
z_arr (list, numpy.ndarray) – List of the redshift bin centers of the probe

Returns
A dictionary containing lists of derivatives of the observed power spectrum for each redshift bin and varied parameter

Return type
dict

eliminate_zbinned_freepars(*nmax*)

This helper function is there to find any parameters passed that correspond to no zbin. It will then remove them from the varied parameters list

Parameters
nmax (int) – Index of the highest redshift bin that should be considered

export_fisher(*fishmat*, *totaltime=None*)

Will print the fisher matrix as well as specifications and the parameter names to files. To change location and filenames for this check the settings dictionary of cosmicfishpie.fishermatrix.config

Parameters

- **fishmat** (numpy.ndarray) – Computed Fisher matrix
- **totaltime** (float, *optional*) – Total time needed to compute the fisher. Will print time information if passed

Returns
A instance of fisher_matrix containing the calculated Fisher matrix as well as parameter names, settings, etc

Return type
cosmicfishpie.analysis.fisher_matrix

fish_integrand(*zi*, *k*, *mu*, *pi*, *pj*)

Helper function to calculate the integrand that enters the computation of the Fisher elements.

Parameters

- **zi** (int) – index of the redshift bin
- **k** (numpy.ndarray) – grid of wavenumbers used in the calculation. Has to be the internal kmesh
- **mu** (numpy.ndarray) – grid of observation angles used in the calculation. Has to be the internal mumesh
- **pi** (str) – name of the first parameter
- **pj** (str) – name of the second parameter

Returns
Integrant that enters the Fisher matrix element computation. The shape matches (shape of the kmesh) x (shape of mumesh)

Return type
numpy.ndarray

fisher_calculation(*zi*, *p_i*, *p_j*)

This helper function calculates a singular element of the fisher matrix

Parameters

- **zi** (int) – index of the redshift bin
- **p_i** (str) – name of the first parameter
- **p_j** (str) – name of the second parameter

Returns
The Fisher matrix element computed from the derivative with respect to two parameters and a given redshift bin for a spectroscopic probe

Return type
float

fisher_integral(*integrand*)

Function to calculate the integral that enters the computation of the Fisher elements. Uses a simpson integration on the mu and k grid

Parameters
integrand (numpy.ndarray) – Integrant that enters the Fisher matrix element computation. The shape needs to match (shape of the kmesh) x (shape of mumesh)

Returns
The Fisher matrix element computed from the derivative with respect to two parameters at a given redshift bin for a spectroscopic probe

Return type
float

fisher_per_bin(*ibin*)

This helper function contains the Fisher matrix of a spectroscopic probe for a single redshift bin

Parameters
ibin (int) – index of the redshift bin

Returns
Fisher matrix of a spectroscopic probe for a single redshift bin

Return type
numpy.ndarray

photo_LSS_fishermatrix(*noisy_cls=None*, *covmat=None*, *derivs=None*, *lss_obj=None*)

Compute the Fisher matrix of a photometric probe.

Parameters

- **noisy_cls** (dict, *optional*) – a dictionary with all the auto and cross correlation fiducial angular power spectra with noise added to it. Will recompute from lss_obj when not passed
- **covmat** (list, *optional*) – A list of pandas.DataFrame objects that store the covariance matrix for each multipole. Will recompute from lss_obj when not passed

- **derivs** (dict, *optional*) – A dictionary containing the derivatives of the angular power spectrum at the fiducial for all free parameters. Will recompute from lss_obj when not passed
- **lss_obj** (*cosmicfishpie.LSSsurvey.photo_cov.PhotoCov*, *optional*) – This object is used to compute the ingredients of the Fisher matrix if they were not passed

Returns
The full fisher matrix for the photometric probe

Return type
numpy.ndarray

pk_LSS_Fisher(*nbins*)

This computes the Fisher matrix of a spectroscopic probe for all redshift bins

Parameters
nbins (int) – number of redshift bins the spectroscopic probe has

Returns
A list of Fisher matrices for each redshift bin

Return type
numpy.ndarray

recap_options()

This will print all the selected options into the standard output

set_pk_settings()

Function to define grids of the internal wavenumber and observation angle

Note: This is not a setter function. Rather the internal grids are constructed from the passed options and then they are stored in the attributes *Pk_kgrid* and *Pk_mugrid* as well as the combined meshes *Pk_kmesh* and *Pk_mumesh*.

cf_version = 'CosmicFish_v1.0'

cosmicfishpie.fishermatrix.cosmicfish.**ray_session**(*num_cpus=4*, *restart=True*, *shutdown=False*)

cosmicfishpie.fishermatrix.derivatives module

DERIVATIVES

This is the derivatives engine of CosmicFish.

class cosmicfishpie.fishermatrix.derivatives.**derivatives**(*observable*, *fiducial*, *special_deriv_function=None*, *freeparams={}*)

Bases: object

der_3pt_stencil(*fwd*, *bwd*, *step*)

Helper function to compute the 3PT symmetrical finite step size derivative

Parameters

- **fwd** (float, numpy.ndarray) – Observable computed at the forward step
- **fwd** – Observable computed at the backwards step

- **step** (float) – Absolute step size of the numerical derivative

Returns
Numerical derivative using the a 3 point stencil

Return type
float, numpy.ndarray

der_fwd_4pt(*fwdi*, *step*)

Helper function to compute the 4PT forward finite step size derivative

Parameters

- **fwdi** (list, numpy.ndarray) – Observable computed at the fiducial and at equally spaced points in the forward direction
- **step** (float) – Absolute distance between the size of the numerical derivative

Returns
Numerical derivative using the a 4 point forward stencil

Return type
float, numpy.ndarray

derivative_3pt()

One of the possible derivative methods. Computes the numerical derivative using a finite differences 3 point symmetrical derivative

Returns
A dictionary containing the derivative of the observable for each varied parameter.

Return type
dict

Note: Implements the following equation:

$$\frac{\mathrm{d}\mathcal{O}}{\mathrm{d}\theta} = \frac{\mathcal{O}(\theta + h) - \mathcal{O}(\theta - h)}{2\,h}$$

---

derivative_forward_4pt()

One of the possible derivative methods. Computes the numerical derivative using a finite differences one-sided 4 point forward derivative. Taken from: https://web.media.mit.edu/~crtaylor/calculator.html @misc{fdcc, title={Finite Difference Coefficients Calculator}, author={Taylor, Cameron R.}, year={2016}, howpublished="url{https://web.media.mit.edu/~crtaylor/calculator.html}" }

Returns
A dictionary containing the derivative of the observable for each varied parameter.

Return type
dict

Note: Implements the following equation:

$$
\begin{aligned}
&frac\\
&mathrmd\\
&mathcalO\\
&mathrmd\\
&theta =\\
&frac-11\\
&,\\
&mathcalO(\\
&theta) + 18\\
&,\\
&mathcalO(\\
&theta + h) - 9\\
&,\\
&mathcalO(\\
&theta + 2\\
&,) + 2\\
&,\\
&mathcalO(\\
&theta + 3\\
&,h)6\\
&,h
\end{aligned}
$$

`derivative_poly()`

One of the possible derivative methods. Computes the numerical derivative using a polynomial derivative method

Returns
A dictionary containing the derivative of the observable for each varied parameter.

Return type
dict

`derivative_stem()`

One of the possible derivative methods. Computes the numerical derivative using the SteM derivative method

Returns
A dictionary containing the derivative of the observable for each varied parameter.

Return type
dict

Module contents

LSS Module

Provide some basic description of the theory module.

3.1.6 cosmicfishpie.utilities package

Submodules

cosmicfishpie.utilities.utils module

class cosmicfishpie.utilities.utils.**filesystem**

Bases: object

static git_version()

static mkdirp(*dirpath*)

This function creates the directory dirpath if it is not found :type dirpath: :param dirpath: string with the path of the directory to be created :return: None :rtype: NoneType

class cosmicfishpie.utilities.utils.**inputiniparser**(*config_dir*, *config_file='main.ini'*, *parameter_translator={}*, *fiducial_translator={}*)

Bases: object

free_epsilons(*epsilon=0.01*)

class cosmicfishpie.utilities.utils.**numerics**

Bases: object

static bisection(*array*, *value*)

Given an `array` , and given a `value` , returns an index j such that `value` is between array[j] and array[j+1]. `array` must be monotonic increasing. #j=-1 or j=len(array) is returned #to indicate that `value` is out of range below and above respectively. # From: https://stackoverflow.com/questions/2566412/find-nearest-value-in-numpy-array

static closest(*lst*, *K*)

static find_nearest(*a*, *a0*)

Element in nd array *a* closest to the scalar value *a0*

static moving_average(*data_set*, *periods=2*)

static round_decimals_up(*number*, *decimals=2*, *precision=5*)

Returns a value rounded up to a specific number of decimal places.

class cosmicfishpie.utilities.utils.**printing**

Bases: object

static debug_print(**args*, ***kwargs*)

static time_print(*feedback_level=0*, *min_level=0*, *text='Computation done in: '*, *time_ini=None*, *time_fin=None*, *instance=None*)

debug = False

Module contents

PACKAGE

Utilities for Fisher Matrix Code

3.2 Submodules

3.3 cosmicfishpie.version module

3.4 Module contents

CHAPTER

FOUR

CHANGELOG

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

4.1 Unreleased

1.0.1 : Bugfix for external file reading for release notebooks. change `is False` statements with a normal `not` to catch False-equivalents.

1.0.2 : Added documentation for the cosmicfishpie.LSSsurvey submodule

1.0.3 : Added documentation for the cosmicfishpie.fishermatrix submodule

CHAPTER

FIVE

CONTRIBUTING

Thanks for considering contributing! Please read this document to learn the various ways you can contribute to this project and how to go about doing it.

5.1 Bug reports and feature requests

5.1.1 Did you find a bug?

First, do a quick search to see whether your issue has already been reported. If your issue has already been reported, please comment on the existing issue.

Otherwise, open a new GitHub issue. Be sure to include a clear title and description. The description should include as much relevant information as possible. The description should explain how to reproduce the erroneous behavior as well as the behavior you expect to see. Ideally you would include a code sample or an executable test case demonstrating the expected behavior.

5.1.2 Do you have a suggestion for an enhancement or new feature?

We use GitHub issues to track feature requests. Before you create a feature request:

- Make sure you have a clear idea of the enhancement you would like. If you have a vague idea, consider discussing it first on a GitHub issue.
- Check the documentation to make sure your feature does not already exist.
- Do a quick search to see whether your feature has already been suggested.

When creating your request, please:

- Provide a clear title and description.
- Explain why the enhancement would be useful. It may be helpful to highlight the feature in other libraries.
- Include code examples to demonstrate how the enhancement would be used.

5.2 Making a pull request

When you're ready to contribute code to address an open issue, please follow these guidelines to help us be able to review your pull request (PR) quickly.

1. **Initial setup** (only do this once)

   If you haven't already done so, please fork this repository on GitHub.

   Then clone your fork locally with

   ```
   git clone https://github.com/USERNAME/cosmicfishpie.git
   ```

   or

   ```
   git clone git@github.com:USERNAME/cosmicfishpie.git
   ```

   At this point the local clone of your fork only knows that it came from *your* repo, github.com/USERNAME/cosmicfishpie.git, but doesn't know anything the *main* repo, https://github.com/santiagocasas/cosmicfishpie.git. You can see this by running

   ```
   git remote -v
   ```

   which will output something like this:

   ```
   origin https://github.com/USERNAME/cosmicfishpie.git (fetch)
   origin https://github.com/USERNAME/cosmicfishpie.git (push)
   ```

   This means that your local clone can only track changes from your fork, but not from the main repo, and so you won't be able to keep your fork up-to-date with the main repo over time. Therefore you'll need to add another "remote" to your clone that points to https://github.com/santiagocasas/cosmicfishpie.git. To do this, run the following:

   ```
   git remote add upstream https://github.com/santiagocasas/cosmicfishpie.git
   ```

   Now if you do `git remote -v` again, you'll see

   ```
   origin https://github.com/USERNAME/cosmicfishpie.git (fetch)
   origin https://github.com/USERNAME/cosmicfishpie.git (push)
   upstream https://github.com/santiagocasas/cosmicfishpie.git (fetch)
   upstream https://github.com/santiagocasas/cosmicfishpie.git (push)
   ```

   Finally, you'll need to create a Python 3 virtual environment suitable for working on this project. There a number of tools out there that making working with virtual environments easier. The most direct way is with the `venv` module in the standard library, but if you're new to Python or you don't already have a recent Python 3 version installed on your machine, we recommend Miniconda.

   On Mac, for example, you can install Miniconda with Homebrew:

   ```
   brew install miniconda
   ```

   Then you can create and activate a new Python environment by running:

   ```
   conda create -n cosmicfishpie python=3.9
   conda activate cosmicfishpie
   ```

   Once your virtual environment is activated, you can install your local clone in "editable mode" with

```
pip install -U pip setuptools wheel
pip install -e .[dev]
```

The "editable mode" comes from the `-e` argument to `pip`, and essential just creates a symbolic link from the site-packages directory of your virtual environment to the source code in your local clone. That way any changes you make will be immediately reflected in your virtual environment.

2. **Ensure your fork is up-to-date**

   Once you've added an "upstream" remote pointing to https://github.com/allenai/python-package-temlate.git, keeping your fork up-to-date is easy:

```
git checkout main  # if not already on main
git pull --rebase upstream main
git push
```

3. **Create a new branch to work on your fix or enhancement**

   Committing directly to the main branch of your fork is not recommended. It will be easier to keep your fork clean if you work on a separate branch for each contribution you intend to make.

   You can create a new branch with

```
# replace BRANCH with whatever name you want to give it
git checkout -b BRANCH
git push -u origin BRANCH
```

4. **Test your changes**

   Our continuous integration (CI) testing runs a number of checks for each pull request on GitHub Actions. You can run most of these tests locally, which is something you should do *before* opening a PR to help speed up the review process and make it easier for us.

   First, you should run `isort` and `black` to make sure you code is formatted consistently. Many IDEs support code formatters as plugins, so you may be able to setup isort and black to run automatically everytime you save. For example, `black.vim` will give you this functionality in Vim. But both `isort` and `black` are also easy to run directly from the command line. Just run this from the root of your clone:

```
isort .
black .
```

   Our CI also uses `ruff` to lint the code base and `mypy` for type-checking. You should run both of these next with

```
ruff check .
```

   and

```
mypy .
```

   We also strive to maintain high test coverage, so most contributions should include additions to the unit tests. These tests are run with `pytest`, which you can use to locally run any test modules that you've added or changed.

   For example, if you've fixed a bug in `cosmicfishpie/a/b.py`, you can run the tests specific to that module with

```
pytest -v tests/a/b_test.py
```

If your contribution involves additions to any public part of the API, we require that you write docstrings for each function, method, class, or module that you add. See the *Writing docstrings* section below for details on the syntax. You should test to make sure the API documentation can build without errors by running

```
make docs
```

If the build fails, it's most likely due to small formatting issues. If the error message isn't clear, feel free to comment on this in your pull request.

And finally, please update the CHANGELOG with notes on your contribution in the "Unreleased" section at the top.

After all of the above checks have passed, you can now open a new GitHub pull request. Make sure you have a clear description of the problem and the solution, and include a link to relevant issues.

We look forward to reviewing your PR!

5.2.1 Writing docstrings

We use Sphinx to build our API docs, which automatically parses all docstrings of public classes and methods using the autodoc extension. Please refer to autoc's documentation to learn about the docstring syntax.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

c

f

u

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W