LSS Documentation

M. S. Suryan Sivadas

Dec 21, 2021

CONTENTS:

CHAPTER

ONE

LARGE-SCALE STRUCTURE

A light-weight module for computations related to large scale structure formation in cosmology. This contains

1. Transfer function fits (*Transfer*).
2. Mass-function fits (*MassFunction*).
3. Large scale linear bias functions (*Bias*)
4. An object for computations related to large-scale structure formation, *CosmoStructure*.

This module is intented to be a *single file module*, for almost all computations related to the large-scale structure formation in the universe. The main tool is the *CosmoStructure* object. It can be used for computing the linear power spectra, variance or the halo mass function under some specified cosmology.

class lss2.**Transfer**

Definitions of various transfer functions. Available functions are listed in the *available* dictionary. Use its keys to select a specific form. Alternatively, use the . operator to access the specific function, like *Transfer.modelEisenstein98* etc. For example,

```
>>> Transfer.available.keys() # available keys
dict_keys(['eisenstein98', 'eisenstein98_zb', 'eisenstein98_mdm', 'sugiyama95'])
>>> f = Transfer.available['eisenstein98'].f # select a function
>>> callable(f)
True
>>> f == Transfer.modelEisenstein98
True
```

Notes

Transfer functions available are Eisenstein-Hu with and without baryon oscillations [R1], with mixed dark-matter [R2] and the BBKS with Sugiyama correction [R3].

References

[R1], [R2], [R3]

Methods

class Data (*f*, *z_dep*, *model*)

Methods

f
Alias for field number 0

model
Alias for field number 2

z_dep
Alias for field number 1

modelEisenstein98(*h: float*, *Om0: float*, *Ob0: float*, *Tcmb0: float*) → float
Matter transfer function given by Eisentein and Hu (1998), with baryon oscillations.

Parameters **k: array_like**

Wavenumbers in Mpc/h

h: float

Hubble parameter in 100 km/sec/Mpc at present

Om0: float

Normalized matter density at present

Ob0: float

Normalized baryon density at present

Tcmb0: float

Present CMB temperature in Kelvin

Returns T: array_like

Value of transfer function. Has the same shape as *k*

Examples

```
>>> Transfer.modelEisenstein98(k = [1.e-4, 1., 1.e+4], Om0 = 0.3, Ob0 = 0.05,␣
↪Tcmb0 = 2.725, h = 0.7)
array([9.99906939e-01, 4.59061096e-03, 2.18246719e-10])
```

modelEisenstein98_mixedDarkMatter(*z: float*, *h: float*, *Om0: float*, *Ob0: float*, *Onu0: float*, *Ode0: float*, *Nnu: float*, *Tcmb0: float*, *_zisDz: bool = False*) → float
Matter transfer function given by Eisentein and Hu (1998), with mixed dark-matter.

NOTE: **This is not tested.**

Parameters **k: array_like**

Wavenumbers in Mpc/h

z: float

Redshift or growth at that time, based on the value of *_zisDz* argument.

h: float

Hubble parameter in 100 km/sec/Mpc at present

Om0: float

Normalized matter density at present

Ob0: float

Normalized baryon density at present

Onu0: float

Normalized massive neutrino (hot dark-matter) density at present

Ode0: float

Normalized dark energy density at present

Nnu: float

Number of massive nuetrinos.

Tcmb0: float

Present CMB temperature in Kelvin

_zisDz: bool, optional

If true, this takes the value of z as **growth**, $D(z)$. Otherwise take it as redshift. This is used internaly for using growth factor definitions consistently. It is False by default.

Returns T: array_like

Value of transfer function. Has the same shape as *k*

Examples

TODO

modelEisenstein98_zeroBaryon(*h: float*, *Om0: float*, *Ob0: float*, *Tcmb0: float*)

Matter transfer function given by Eisentein and Hu (1998), with without baryon oscillations.

Parameters **k: array_like**

Wavenumbers in Mpc/h

h: float

Hubble parameter in 100 km/sec/Mpc at present

Om0: float

Normalized matter density at present

Ob0: float

Normalized baryon density at present

Tcmb0: float

Present CMB temperature in Kelvin

Returns T: array_like

Value of transfer function. Has the same shape as *k*

Examples

```
>>> Transfer.modelEisenstein98_zeroBaryon(k = [1.e-4, 1., 1.e+4], Om0 = 0.3,␣
↪Ob0 = 0.05, Tcmb0 = 2.725, h = 0.7)
array([9.99899458e-01, 4.55592401e-03, 2.14698256e-10])
```

modelSugiyama95 (*h: float*, *Om0: float*, *Ob0: float*, *Tcmb0: float = None*) → float

Matter transfer function given by Bardeen et al.(1986), with correction given by Sugiyama(1995).

Parameters **k: array_like**

Wavenumbers in Mpc/h

h: float

Hubble parameter in 100 km/sec/Mpc at present

Om0: float

Normalized matter density at present

Ob0: float

Normalized baryon density at present

Tcmb0: float

Present CMB temperature in Kelvin (not used)

Returns T: array_like

Value of transfer function. Has the same shape as *k*

Examples

```
>>> Transfer.modelSugiyama95(k = [1.e-4, 1., 1.e+4], Om0 = 0.3, Ob0 = 0.05,␣
↪Tcmb0 = 2.725, h = 0.7)
array([9.98671614e-01, 4.65026996e-03, 2.03316613e-10])
```

class lss2.**MassFunction**

Definitions of various halo-mass function fits. Available functions are listed in the *available* dictionary. Use its keys to select a specific form. Alternatively, use the *dot* operator to access the specific function, like *MassFunction.modelTinker08* etc. For example,

```
>>> MassFunction.available.keys() # available keys
dict_keys(['press74', 'sheth01', 'tinker08'])
>>> f = MassFunction.available['tinker08'].f # select a function
>>> callable(f)
True
>>> f == MassFunction.modelTinker08
True
```

Notes

Available massfunction models are the Press-Schechter [R4], Sheth et al [R5] and Tinker et al [R6].

References

[R4], [R5], [R6]

Methods

class Data (*f*, *mdef*, *z_dep*)

Methods

f
Alias for field number 0

mdef
Alias for field number 1

z_dep
Alias for field number 2

modelPress74 ()
Fitting function found by Press and Schechter in 1974.

Parameters **sigma** : array_like

Mass variance

Returns **f** : array_like

Value of the fitting function

Examples

```
>>> MassFunction.modelPress74([0.7, 0.8, 0.9, 1.0])
array([0.10553581, 0.18231359, 0.25834461, 0.32457309])
```

modelSheth01 ()
Fitting function found by Sheth et. al. in 2001.

Parameters **sigma** : array_like

Mass variance

Returns **f** : array_like

Value of the fitting function

Examples

```
>>> MassFunction.modelSheth01([0.7, 0.8, 0.9, 1.0])
array([0.1107284 , 0.16189185, 0.2061881 , 0.24155146])
```

modelTinker08 (*Delta: float*, *z: float*)
Fitting function found by J. Tinker et. al. in 2008[1].

Parameters **sigma: array_like**

Sigma values (square root of variance)

Delta: float

Overdensity with respect to mean density of the background.

z: float

redshift

Returns f: array_like

Values of fit. Has the same size as *sigma*.

Examples

```
>>> MassFunction.modelTinker08([0.7, 0.8, 0.9, 1.0], 200, 0.)
array([0.12734265, 0.19005856, 0.24294619, 0.28320827])
```

class lss2.**Bias**

Definitions of various large scale linear halo bias functions. Available functions are listed in the *available* dictionary. Use its keys to select a specific form. Alternatively, use the *dot* operator to access the specific function, like *Bias.modelTinker10* etc.

Notes

Available models are Tinker et al, Cole et al and Sheth et al (for references, see [R7] [R8] and [R9]).

References

New in version 1.1.

[R7], [R8], [R9]

Examples

```
>>> Bias.available.keys() # available keys
dict_keys(['cole89', 'sheth01', 'tinker10'])
>>> f = Bias.available['tinker10'].f # select a function
>>> callable(f)
True
>>> f == Bias.modelTinker10
True
```

Methods

class Data(*f*, *mdef*, *z_dep*)

Methods

f
Alias for field number 0

mdef
Alias for field number 1

z_dep
Alias for field number 2

modelCole89()
Bias function given by Cole & Kaiser (1989) and Mo & White (1996).

Parameters **nu: array_like**

Peak height, defined as δ_c/σ.

Returns b: array_like

Value of bias function.

Examples

```
>>> Bias.modelCole89([1., 1.5, 2., 2.5])
array([1.        , 1.74119306, 2.77886333, 4.11301083])
```

modelSheth01()

Bias function given by Sheth et al. (2001).

Parameters **nu: array_like**

Peak height, defined as δ_c/σ.

Returns b: array_like

Value of bias function.

Examples

```
>>> Bias.modelSheth01([1., 1.5, 2., 2.5])
array([1.07578897, 1.66258103, 2.47024337, 3.49037188])
```

modelTinker10(*Delta: float*, *z: float*)

Bias function given by Tinker et al. (2010).

Parameters **nu: array_like**

Peak height, defined as δ_c/σ.

Delta: float

Overdensity with respect to mean density of the background.

z: float

redshift

Returns b: array_like

Value of bias function.

Examples

```
>>> Bias.modelTinker10([1., 1.5, 2., 2.5], 200, 0.)
array([0.96550128, 1.54189048, 2.41182243, 3.60186066])
```

class lss2.**CosmoStructure**(*, *flat: bool = True*, *heavy_nu: bool = False*, *Om0: float = Ellipsis*, *Ob0: float = Ellipsis*, *Ode0: float = Ellipsis*, *sigma8: float = Ellipsis*, *n: float = Ellipsis*, *h: float = Ellipsis*, *Nnu: int = Ellipsis*, *Mnu: float = Ellipsis*, *Tcmb0: float = 2.725*, *psmodel: str = 'eisenstein98'*, *hmfmodel: str = 'tinker08'*, *biasmodel: str = 'tinker10'*, *gfmodel: str = 'exact'*)

An object for storing a radiationless Λ-MDM or Λ-CDM cosmology and do calculations related to large-scale

structure formation. It can be used for computing power spectrum and halo-mass functions. For the details of equations used, refer any books on cosmology and structure formation (eg., *Galaxy Formation and Evolution* by Mo and White).

Parameters **flat: bool, optional**

Specify the space geometry. By default, a flat cosmology is used.

heavy_nu: bool, optional

Specify if the model has heavy neutrinos (i.e., mixed dark-matter model). Default is True (cold dark-matter only).

Om0: float

Normalized matter density at present.

Ob0: float

Normalized baryon density at present.

Ode0: float

Normalized dark-energy density. It is a **required** parameter for non-flat cosmologies.

sigma8: float

Variance at the scale of 8 Mpc/h at present.

n: float

Tilt of the power spectrum or spectral index.

h: float

Hubble parameter in 100 km/sec/Mpc at present.

Tcmb0: float, optional

Present CMB temperature in Kelvin. Default is 2.725 K.

psmodel: str, optional

Power spectrum model. Allowed values are *eisenstein98* (for Eisenstein and Hu with BAO, default), *eisenstein98_zb* (EH without BAO) and *sugiyama95* (or *bbks* for BBKS with Sugiyama correction).

hmfmodel: str, optional

Halo mass-function model. Allowed values are *press74* (for fit by Press and Schechter, 1974), *sheth01* (Sheth et al, 2001) and *tinker08* (Tinker et al, 2008).

biasmodel: str, optional

Linear halo-bias model. Allowed values *cole89* (Cole & Kaiser, 1989), *sheth01* (Sheth et al, 2001) and *tinker10* (Tinker et al, 2010).

gfmodel: str, optional

Growth function model. Allowed values are *exact* (default, use the integral form) and *carroll92* (use the fit by Carroll et al, 1992).

Notes

1. All the parameters are keyword arguments. Some has default values, while others are required.

2. This use `scipy.integrate.simps()` for k space integrations and `scipy.integrate.quad()` for time integration. Integration settings can be updated by calling *`quadOptions()`* with new values of *n* - number of points, *kmin* and *kmax* - minimum and maximum *k* limits.

Examples

To create a *`CosmoStructure`* with defaults (flat cosmology with only cold dark-matter),

```
>>> cs = CosmoStructure(Om0 = 0.3, Ob0 = 0.05, sigma8 = 0.8, n = 1., h = 0.7)
>>> cs
<CosmoStructure flat = True, Om0 = 0.3, Ob0 = 0.05, Ode0 = 0.7, sigma8 = 0.8, ns␣
↪= 1, h = 0.7, Tcmb0 = 2.73 K, Mnu = 0 eV/c2; power = 'eisenstein98', hmf =
↪'tinker08', bias = 'tinker10'>
```

If not specified *cs* in the deocumentation of methods defined here correspond to this object.

Methods

`Ez` (*z: float*)
Cosmology function $E(z)$ defined as

$$E(z) = \sqrt{\Omega_{\rm m}(z+1)^3 + \Omega_{\rm k}(z+1)^2 + \Omega_{\rm de}}$$

Parameters **z: array_like**

Redshift

Returns retval: array_like

Value of $E(z)$. Has the same shape as z.

Examples

```
>>> cs.Ez([0., 1., 2.])
array([1.        , 1.76068169, 2.96647939])
```

`Odez` (*z: float*)
Normalized matter density at redshift z.

Parameters **z: array_like**

Redshift

Returns retval: array_like

Density. Has the same shape as z.

Examples

```
>>> cs.Odez([0., 1., 2.])
array([0.7       , 0.22580645, 0.07954545])
```

`Omz` (*z: float*)
Normalized matter density at redshift z.

Parameters **z: array_like**

Redshift

Returns retval: array_like

Density. Has the same shape as *z*.

Examples

```
>>> cs.Omz([0., 1., 2.])
array([0.3       , 0.77419355, 0.92045455])
```

bias (*m: float, z: float = 0.0, mdef: str = Ellipsis*)
Get the (large scale) linear halo bias function.

Parameters **m: array_like**

Mass in Msun/h

mdef: str, optional

Mass definition corresponding to the halo finder. This function is defined only for spherical overdensity (SO) mass definitions, i.e., **m*, **c* and *vir*.

z: float, optional

Redshift. Default is 0.

Returns retval: array_like

Values of bias function. Has the same size as *m*.

Examples

```
>>> cs.biasmodel
'tinker10'
>>> cs.bias([1.e+06, 1.e+09, 1.e+12], z = 0., mdef = '200m')
array([0.59202591, 0.61077932, 0.81167364])
```

New in version 1.1.

correlation (*r: float, z: float = 0.0, ignore_norm: bool = False*)
Get the (normalised) 2-point correlation function ar scale r. This is the Fourier transform of the matter power spectrum and is found by evaluating the integral

$$\xi(r) = \frac{1}{2\pi^2}\int_0^\infty P(k)\frac{\sin(kr)}{kr}k^2\mathrm{d}k$$

Parameters **r: array_like**

Radius in Mpc/h

z: float

Redshift

ignore_norm: bool

If true, give the un-normalized values.

Returns retval: array_like

Value of 2-pt. correlation function. Has the same shape as *r*.

Examples

```
>>> cs.correlation([0.01, 0.1, 1.], z = 0.)
array([79.9002375 , 27.03930878,  5.39579643])
>>> cs.correlation([0.01, 0.1, 1.], z = 1.)
array([29.90716065, 10.12098308,  2.01968049])
```

New in version 1.1.

criticalDensity(*z: float*)
Critical density in units of kg/m^3.

Parameters **z: array_like**

Redshift

Returns retval: array_like

critical density. Has the same shape as *z*.

Examples

```
>>> cs.criticalDensity([0., 1., 2.]) # in kg/m3
array([9.20390150e-27, 2.85320947e-26, 8.09943332e-26])
```

dlnsdlnm(*r: float*, *z: float = 0.0*)
Get the logarithmic derivative of σ w.r.to mass. This is 1/3 times the derivative w.r.to radius.

Parameters **r: array_like**

Radius in Mpc/h.

z: float, optional

Redshift

Returns y: array_like

Values of derivative. Has the same size as *m*.

Examples

```
>>> cs.dlnsdlnm([0.01, 0.1, 1.])
array([-0.06545157, -0.09101889, -0.13984833])
```

filt(*x: float*, *deriv: bool = False*)
Top-hat filter

fit(*sigma: float*, *mdef: str = Ellipsis*, *z: float = Ellipsis*)
Get the fitting function.

Parameters **sigma: array_like**

Sigma values (square root of variance)

mdef: str, optional

Mass definition corresponding to the halo finder. This function is defined only for Fof and spherical overdensity (SO) mass definitions (i.e., *m*, *c* and *vir*).

z: float, optional

Redshift.

Note: Not all these parameters are needed for all fitting functions.

Returns f: array_like

Values of fit. Has the same size as *sigma*.

Examples

```
>>> s = np.linspace(0., 10., 21)
>>> cs.hmfmodel
'tinker08'
>>> cs.fit([0.1, 1., 10.], mdef = '200m', z = 0.)
array([4.62093863e-51, 2.83208269e-01, 2.08742594e-01])
```

growth(*z: float*, *ignore_norm: bool = False*)

Returns the normalized linear growth factor. This is **not designed for vector input**. Growth factor is calculated either by using the fit by Carroll et al (1992) or by evaluating the integral

$$D_+(z) \propto H(z) \int_z^\infty \mathrm{d}z' \frac{1+z'}{E^3(z')}$$

Parameters **z: float**

Redshift

ignore_norm: bool

Ignore the normalization if set true.

Returns retval: float

Linear growth factor.

Examples

```
>>> cs.growth(1.)
0.6118057532973097
```

lagrangianM(*r: float*)

Comoving mass corresponding to comoving lagrangian radius.

Parameters **r: array_like**

Lagrangian radius in Mpch/

Returns retval: array_like

Mass in Msun/h. Has the same shape as *r*.

Examples

```
>>> m = cs.lagrangianM([0.01, 0.1, 1.]) # in Msun/h
>>> m
array([3.4876208e+05, 3.4876208e+08, 3.4876208e+11])
```

To test the results, let's find the radius:

```
>>> cs.lagrangianR(m) # same as input `r`
array([0.01, 0.1 , 1.  ])
```

New in version 1.1.

lagrangianR(*m: float*)

Comoving lagrangian radius in Mpc/h.

Parameters **m: array_like**

Mass in Msun/h

Returns retval: array_like

Lagrangian radius. Has the same shape as *m*.

Examples

```
>>> cs.lagrangianR([1.e+6, 1.e+9, 1.e+12]) # in Mpc/h
array([0.0142066 , 0.14206603, 1.42066031])
```

massFunction(*m: float*, *mdef: str = '200m'*, *z: float = 0.0*, *form: str = 'dndm'*)

Get the halo mass-function with current settings. It is given by in terms of a fitting (or, multiplicity) function $f(\sigma)$ as

$$\frac{\mathrm{d}n}{\mathrm{d}M}\mathrm{d}M = \frac{\rho_\mathrm{m}}{M^2} f(\sigma) \left|\frac{\mathrm{d}\ln\sigma}{\mathrm{d}\ln M}\right| \mathrm{d}M$$

Parameters **m: array_like**

Mass of halo

mdef: str, optional

Mass definition corresponding to the halo finder. This function is defined only for spherical overdensity (SO) mass definitions, i.e., **m*, **c* and *vir*.

z: float, optional

Redshift. Default is 0.

form: str, optional

Format of output. Available values are *dndm* (which give $\frac{\mathrm{d}n}{\mathrm{d}M}$, default), *dndlnm* (for $\frac{\mathrm{d}n}{\mathrm{d}\log M}$) and *fsigma* (for $f(\sigma)$).

Returns retval: float

Values of mass-function in specified format. Has the same size as *m*.

Examples

```
>>> cs.massFunction([1.e+6, 1.e+9, 1.e+12], mdef = '200m', z = 0., form =
↪'fsigma')
array([0.21462692, 0.2465193 , 0.33287869])
>>> cs.massFunction([1.e+6, 1.e+9, 1.e+12], mdef = '200m', z = 0., form =
↪'dndm')
array([1.22326921e-03, 1.98075984e-09, 4.18069529e-15])
```

(continues on next page)

(continued from previous page)

```
>>> cs.massFunction([1.e+6, 1.e+9, 1.e+12], mdef = '200m', z = 0., form =
↪'dndlnm')
array([1.22326921e+03, 1.98075984e+00, 4.18069529e-03])
```

matterPowerSpectrum(*k: float*, *z: float = 0.0*, *ignore_norm: bool = False*)

Get the (normalised) linear matter power spectrum at redshift z.

Parameters **k: array_like**

Wavenumbers in h/Mpc.

z: float

Redshift

ignore_norm: bool

Ignore the normalization and return the un-normalized value.

Returns retval: array_like

Power spectrum. Has the same shape as *k*.

Examples

```
>>> cs.matterPowerSpectrum([1.e-4, 1., 1.e+4], z = 0.)
array([3.23064206e+02, 6.80942839e+01, 1.53909393e-09])
>>> cs.matterPowerSpectrum([1.e-4, 1., 1.e+4], z = 1.)
array([1.20924961e+02, 2.54881181e+01, 5.76092522e-10])
```

matterPowerSpectrumNorm()

Return the value of power spectrum norm

normalize()

Normalise the power spectrum.

parseMassDefn(*mdef: str*, *z: float*)

This convert the mass definition to overdensity values.

Parameters **mdef: str**

one of the SO mass definitions. *vir*, **m* and **c* are only choises and others will raise exception.

z: float

Redshift

Returns retval: float

Overdensity in $h^2 Msun/Mpc^3$.

Examples

```
>>> for mdef in ['200m', '500c', 'vir']:
...     print(cs.parseMassDefn(mdef, z = 0.))    # in kg/m3
...
5.522340901858673e-25
```

(continues on next page)

(continued from previous page)

```
4.601950751548894e-24
9.295940518128765e-25
```

peakHeight (*m: float*, *z: float = 0.0*)

Get the peak height corresponding to mass m. It is given by $\nu = \delta_c/\sigma(M, z)$.

Parameters **m: array_like**

Mass in Msun/h

z: float

Redshift

Returns retval: array_like

Values of peak height. Has the same size as *m*.

Examples

```
>>> cs.peakHeight([1.e+06, 1.e+09, 1.e+12], z = 0.)
array([0.1990946 , 0.34851773, 0.80123727])
```

New in version 1.1.

quadOptions (*n: int*, *kmin: float*, *kmax: float*, *renormalize: bool = True*)

Set the k-space integration options. By default, integration range is chosen as $[10^{-8}, 10^{8}]$ with 5001 function evaluations.

Parameters **n: int**

Number of function calls to make (or, size of k-space grid)

kmin: float

Lower integration limit

kmax: float

Upper integration limit

renormalize: bool, optional

If True, it will renormalize the power spectrum with given settings.

radius (*sigma: float*, *z: float = 0.0*, *rmin: float = 0.001*, *rmax: float = 10000.0*, *reltol: float = 1e-08*)

Get the radius from the sigma values. This solves for the radius using Brent's method (`scipy.optimize.brentq()`).

Parameters **sigma: array_like**

Mass variance

z: float

Redshift

rmin: float, optional

Lower bracketing limit of r. Default is 1e-3.

rmax: float, optional

Upper bracketing limit of r. Default is 1e+4.

reltol: float, optional

Relative tolerance for the result. Default is 1e-8.

Returns r: array_like

Radius in Mpc/h

Examples

```
>>> r = cs.radius([2., 1., 0.1, 0.01], z = 0.)
>>> r
array([  1.58825172,   5.71225866,  70.47066068, 330.48096125])
```

To check the result,

```
>>> s = cs.sigma(r)
>>> s
array([2.  , 1.  , 0.1 , 0.01])
>>> np.allclose(s, [2., 1., 0.1, 0.01])
True
```

New in version 1.1.

rho_de(*z: float*)

Dark-energy density in units of kg/m^3

Parameters **z: array_like**

Redshift

Returns retval: array_like

Density. Has the same shape as z.

Examples

```
>>> cs.rho_de([0., 1., 2.]) # in kg/m3
array([6.44273105e-27, 6.44273105e-27, 6.44273105e-27])
```

rho_m(*z: float*)

Matter density in units of kg/m^3

Parameters **z: array_like**

Redshift

Returns retval: array_like

Density. Has the same shape as z.

Examples

```
>>> cs.rho_m([0., 1., 2.]) # in kg/m3
array([2.76117045e-27, 2.20893636e-26, 7.45516022e-26])
```

sigma(*r: float*, *z: float = 0.0*, *ignore_norm: bool = False*, *deriv: bool = False*)

Get the square root of variance (std. deviation) or its logarithmic derivative w. r. to radius (i.e., $\mathrm{d}\log\sigma/\mathrm{d}\log r$).

Parameters **r: array_like**

Radius of spherical mass (scale) in Mpc/h.

z: float

Redshift.

ignore_norm: bool

If true, give the un-normalized value of $sigma$. It is not used for derivative.

deriv: bool

If true, give the first derivative w.r.to the radius.

Returns retval: array_like

Value of σ. Has the same shape as *r*.

Examples

```
>>> cs.sigma([0.01, 0.1, 1.], z = 1.) # sigma
array([5.5609891 , 3.26767943, 1.50063023])
>>> cs.sigma([0.01, 0.1, 1.], z = 1., deriv = True) # dln(sigma)/dln(r)
array([-0.19635472, -0.27305668, -0.41954498])
```

transfer(*k: float*, *z: float = 0.0*)

Evaluate the current model of transfer function, using the current cosmology.

Parameters **k: array_like**

Wavenumbers in h/Mpc.

z: float, optional

Redshift. Used only when the transfer function need it.

_zisDz: bool, optional

If True, take the *z* parameter as growth, else as redshift. Default is False.

Returns retval: array_like

Transfer function. Has the same shape as *k*.

Examples

```
>>> cs.transfer([1.e-4, 1., 1.e+4])
array([9.99906939e-01, 4.59061096e-03, 2.18246719e-10])
```

variance(*r: float*, *z: float = 0.0*, *ignore_norm: bool = False*, *deriv: bool = False*)

Get the (normalised) value of variance of the density fluctuations, smoothed at a scale of r using a top-hat filter. It is found by evaluating the integral

$$\sigma^2(r) = \frac{1}{2\pi^2}\int_0^\infty P(k)W(kr)k^2\mathrm{d}k$$

where $W(x) = 3(\sin x - x\cos x)/x^2$ is the spherical top-hat filter in Fourier space.

Parameters **r: array_like**

Radius of spherical mass (scale) in Mpc/h.

z: float

Redshift

ignore_norm: bool

If true, give the un-normalized values.

deriv: bool

If true, give the first derivative w.r.to the radius.

Returns retval: array_like

Variance. Has the same shape as *r*.

Examples

```
>>> cs.variance([0.01, 0.1, 1.], z = 1.)
array([30.92459974, 10.67772887,  2.2518911 ])
>>> cs.variance([0.01, 0.1, 1.], z = 1., deriv = True)
array([-1214.43822506,   -58.31250423,    -1.88953923])
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[R1] Daniel J. Eisenstein and Wayne Hu. Baryonic Features in the Matter Transfer Function, arXive:astro-ph/9709112v1,, 1997.

[R2] Daniel J. Eisenstein and Wayne Hu. Power Spectra for Cold Dark Matter and its Variants, arXive:astro-ph/9710252v1,, 1997.

[R3] A. Meiksin, matrin White and J. A. Peacock. Baryonic signatures in large-scale structure, Mon. Not. R. Astron. Soc. 304, 851-864, 1999.

[R4] Houjun Mo, Frank van den Bosch, Simon White. Galaxy Formation and Evolution, Cambridge University Press, 2010.

[R5] Zarija Lukic et al. The Halo Mass Function: High-Redshift Evolution and Universality, arXive:astro-ph/0702360v2,, 14 January 2008.

[R6] Jeremy Tinker et. al. Toward a halo mass function for precision cosmology: The limits of universality, arXive:astro-ph/0803.2706v1,, 2008

[R7] Jeremy L. Tinker et al. The Large Scale Bias of Dark Matter Halos: Numerical Calibration and Model Tests. arXive:astro-ph/1001.3162v2, 2010.

[R8] H. J. Mo and Y. P. Jing and S. D. M. White. High-order correlations of peaks and haloes: a step towards understanding galaxy biasing. Mon. Not. R. Astron. Soc. 284,189-201, 1997.

[R9] Shaun Cole and Nick Kaiser. Biased clustering in the cold dark matter cosmogony, Mon. Not. R. astr. Soc., 237, 1127-1146, 1989.

PYTHON MODULE INDEX

INDEX

V

Z