hmf Documentation

Release 1.8.0

Steven Murray

February 19, 2016

Contents

hmf is a python application that provides a flexible and simple way to calculate the Halo Mass Function for any input cosmology, redshift, dark matter model, virial overdensity or several other variables. Addition of further variables should be simple.

It is also the backend to HMFcalc, the online HMF calculator.

CHAPTER 1

Installation

hmf is built on several other packages, most of which will be familiar to the scientific python programmer. All of these dependencies *should* be automatically installed when installing *hmf*, except for one. Explicitly, the dependencies are numpy, scipy, scitools, cosmolopy and emcee.

You will only need *emcee* if you are going to be using the fitting capabilities of *hmf*. The final, optional, library is pycamb, which can not be installed using pip currently. To install pycamb:

```
cd <Directory that pycamb source will live in>
git clone https://github.com/steven-murray/pycamb
cd pycamb
[sudo] python setup.py install [--get=www.address-where-camb-code-lives.org]
```

The final command gives the option of automatically downloading and compiling CAMB while installing pycamb. It cannot be done more automatically at this point due to licensing. Alternatively, if one does not know the location of the camb downloads, go to camb.info and follow the instructions. Download the source directory to your pycamb folder, and untar it there. Then use ‘‘python setup.py install” and it should work.

Note: At present, versions of CAMB post March 2013 are not working with *pycamb*. Please use earlier versions until further notice.

Finally the *hmf* package needs to be installed: `pip install hmf`. If you want to install the latest build (not necessarily stable), grab it here.

CHAPTER 2

Basic Usage

hmf can be used interactively (for instance in ipython) or in a script and is called like this:

```
>>> from hmf import MassFunction
>>> hmf = MassFunction()
>>> mass_func = hmf.dndlnm
>>> mass_variance = hmf.sigma
>>> ...
```

This will return a Sheth-Mo-Tormen (2001) mass function between $10^{10} - 10^{15} M_{\odot}$, at $z = 0$ for the default PLANCK cosmology. Cosmological parameters may be passed to the initialiser, `MassFunction()`

To change the parameters (cosmological or otherwise), one should use the *update()* method, if a MassFunction() object already exists. For example

```
>>> hmf = MassFunction()
>>> hmf.update(omegab = 0.05,z=10) #update baryon density and redshift
>>> cumulative_mass_func = hmf.ngtm
```

Note: Older versions of *hmf* used the class called *Perturbations()* rather than *MassFunction()*.

Please check the more in-depth user-guide for more details, or even the API documentation.

CHAPTER 3

User Guide

Look here for more details concerning the usage in general.

CHAPTER 4

API Documentation

4.1 API

We here list the modules within the *hmf* package, and the objects in each.

4.1.1 hmf

This is the primary module for user-interaction with the `hmf` package.

The module contains a single class, *MassFunction*, which wraps almost all the functionality of `hmf` in an easy-to-use way.

class `hmf.hmf.`**`MassFunction`**(*Mmin=10*, *Mmax=15*, *dlog10m=0.01*, *mf_fit=<class 'hmf.fitting_functions.Tinker08'>*, *delta_h=200.0*, *delta_wrt='mean'*, *cut_fit=True*, *z2=None*, *nz=None*, *_fsig_params={}*, *delta_c=1.686*, *filter=<class 'hmf.filters.TopHat'>*, *filter_params={}*, ***transfer_kwargs*)

An object containing all relevant quantities for the mass function.

The purpose of this class is to calculate many quantities associated with the dark matter halo mass function (HMF). The class is initialized to form a cosmology and takes in various options as to how to calculate all further quantities.

All required outputs are provided as `@property` attributes for ease of access.

Contains an update() method which can be passed arguments to update, in the most optimal manner. All output quantities are calculated only when needed (but stored after first calculation for quick access).

Quantities related to the transfer function can be accessed through the `transfer` property of this object.

Parameters **Mmin** : float

Minimum mass at which to perform analysis [units $\log_{10} M_\odot h^{-1}$].

Mmax : float

Maximum mass at which to perform analysis [units $\log_{10} M_\odot h^{-1}$].

dlog10m : float

log10 interval between mass bins

mf_fit : str or callable, optional, default `"SMT"`

A string indicating which fitting function to use for $f(\sigma)$

Available options:

1. `'PS'`: Press-Schechter form from 1974
2. `'ST'`: Sheth-Mo-Tormen empirical fit 2001 (deprecated!)
3. `'SMT'`: Sheth-Mo-Tormen empirical fit from 2001
4. `'Jenkins'`: Jenkins empirical fit from 2001
5. `'Warren'`: Warren empirical fit from 2006
6. `'Reed03'`: Reed empirical from 2003
7. `'Reed07'`: Reed empirical from 2007
8. `'Tinker'`: Tinker empirical from 2008
9. `'Watson'`: Watson empirical 2012
10. `'Watson_FoF'`: Watson Friend-of-friend fit 2012
11. `'Crocce'`: Crocce 2010
12. `'Courtin'`: Courtin 2011
13. `'Angulo'`: Angulo 2012
14. `'Angulo_Bound'`: Angulo sub-halo function 2012
15. `'Bhattacharya'`: Bhattacharya empirical fit 2011
16. `'Behroozi'`: Behroozi extension to Tinker for high-z 2013

    Alternatively, one may define a callable function, with the signature `func(self)`, where `self` is a *MassFunction* object (and has access to all its attributes). This may be passed here.

delta_wrt : str, {`"mean"`, `"crit"`}

Defines what the overdensity of a halo is with respect to, mean density of the universe, or critical density.

delta_h : float, optional, default `200.0`

The overdensity for the halo definition, with respect to `delta_wrt`

user_fit : str, optional, default `""`

A string defining a mathematical function in terms of x, used as the fitting function, where x is taken as σ. Will only be applicable if `mf_fit == "user_model"`.

cut_fit : bool, optional, default `True`

Whether to forcibly cut $f(\sigma)$ at bounds in literature. If false, will use whole range of M.

delta_c : float, default `1.686`

The critical overdensity for collapse, δ_c

kwargs : keywords

These keyword arguments are sent to the *hmf.transfer.Transfer* class.

Included are all the cosmological parameters (see the docs for details).

Attributes

H0	
M	
Mmax	
Mmin	
N_nu	
N_nu_massive	
cosmolopy_dict	Collect parameters into a dictionary suitable for cosmolopy.
cs2_lam	
cut_fit	
delta_c	
delta_h	
delta_halo	Overdensity of a halo w.r.t mean density
delta_k	Dimensionless power spectrum, $\Delta_k = \frac{k^3 P(k)}{2\pi^2}$
delta_wrt	
dlnk	
dlog10m	
dndlnm	The differential mass function in terms of natural log of M, `len=len(M)` [units $h^3 Mpc^{-3}$]
dndlog10m	The differential mass function in terms of log of M, `len=len(M)` [units $h^3 Mpc^{-3}$]
dndm	The number density of haloes, `len=len(M)` [units $h^4 M_\odot^{-1} Mpc^{-3}$]
filter	
filter_mod	
filter_params	
force_flat	
fsigma	The multiplicity function, $f(\sigma)$, for *mf_fit*.
growth	The growth factor $d(z)$
h	
how_big	Size of simulation volume in which to expect one halo of mass M, `len=len(M)` [units $Mpch^{-1}$]
kr_warning	
lnk	
lnk_max	
lnk_min	
lnsigma	Natural log of inverse mass variance, `len=len(M)`
mean_dens	
mean_dens_z	Mean density of universe at redshift z
mf_fit	
n	
n_eff	Effective spectral index at scale of halo radius, `len=len(M)`
ngtm	The cumulative mass function above M, `len=len(M)` [units $h^3 Mpc^{-3}$]
nonlinear_delta_k	Dimensionless nonlinear power spectrum, $\Delta_k = \frac{k^3 P_{\rm nl}(k)}{2\pi^2}$
nonlinear_power	Non-linear log power [units Mpc^3/h^3]
nu	The parameter :math:‘
nz	
omegab	
omegab_h2	
omegac	
omegac_h2	
omegak	
omegam	
omegam_z	Density parameter at redshift of this instance.
omegan	

Continued on next page

Table 4.1 – continued from previous page

omegav	
power	Normalised log power spectrum [units Mpc^3/h^3]
pycamb_dict	Collect parameters into a dictionary suitable for pycamb.
radii	The radii corresponding to the masses *M*
rho_gtm	Mass density in haloes >*M*, len=len(M) [units $M_\odot h^2 Mpc^{-3}$]
rho_ltm	Mass density in haloes <*M*, len=len(M) [units $M_\odot h^2 Mpc^{-3}$]
sigma	The mass variance at z, len=len(M)
sigma_8	
t_cmb	
takahashi	
tau	
transfer_fit	
transfer_options	
w	
y_he	
z	
z2	
z_reion	

Methods

cosmo_update(**kwargs)	
update(**kwargs)	Update the class optimally with given arguments.

delta_halo
Overdensity of a halo w.r.t mean density

dndlnm
The differential mass function in terms of natural log of *M*, len=len(M) [units $h^3 Mpc^{-3}$]

dndlog10m
The differential mass function in terms of log of *M*, len=len(M) [units $h^3 Mpc^{-3}$]

dndm
The number density of haloes, len=len(M) [units $h^4 M_\odot^{-1} Mpc^{-3}$]

fsigma
The multiplicity function, $f(\sigma)$, for *mf_fit*. len=len(M)

how_big
Size of simulation volume in which to expect one halo of mass M, len=len(M) [units $Mpch^{-1}$]

lnsigma
Natural log of inverse mass variance, len=len(M)

mean_dens_z
Mean density of universe at redshift z

n_eff
Effective spectral index at scale of halo radius, len=len(M)

Notes

This function, and any derived quantities, can show small non-physical 'wiggles' at the 0.1% level, if too coarse a grid in ln(k) is used. If applications are sensitive at this level, please use a very fine k-space grid.

Uses eq. 42 in Lukic et. al 2007.

ngtm
The cumulative mass function above *M*, `len=len(M)` [units $h^3 Mpc^{-3}$]

In the case that *M* does not extend to sufficiently high masses, this routine will auto-generate `dndm` for an extended mass range. If `cut_fit` is True, and this extension is invalid, then a power-law fit is applied to extrapolate to sufficient mass.

In the case of the Behroozi fit, it is impossible to auto-extend the mass range except by the power-law fit, thus one should be careful to supply appropriate mass ranges in this case.

nu

The parameter :math:'

u = left(rac{delta_c}{sigma} ight)^2', `len=len(M)`

omegam_z
Density parameter at redshift of this instance.

radii
The radii corresponding to the masses *M*

rho_gtm
Mass density in haloes >*M*, `len=len(M)` [units $M_\odot h^2 Mpc^{-3}$]

In the case that *M* does not extend to sufficiently high masses, this routine will auto-generate `dndm` for an extended mass range. If `cut_fit` is True, and this extension is invalid, then a power-law fit is applied to extrapolate to sufficient mass.

In the case of the Behroozi fit, it is impossible to auto-extend the mass range except by the power-law fit, thus one should be careful to supply appropriate mass ranges in this case.

rho_ltm
Mass density in haloes <*M*, `len=len(M)` [units $M_\odot h^2 Mpc^{-3}$]

Note: As of v1.6.2, this assumes that the entire mass density of halos is encoded by the `mean_density` parameter (ie. all mass is found in halos). This is not explicitly true of all fitting functions (eg. Warren), in which case the definition of this property is somewhat inconsistent, but will still work.

In the case that *M* does not extend to sufficiently high masses, this routine will auto-generate `dndm` for an extended mass range. If `cut_fit` is True, and this extension is invalid, then a power-law fit is applied to extrapolate to sufficient mass.

In the case of the Behroozi fit, it is impossible to auto-extend the mass range except by the power-law fit, thus one should be careful to supply appropriate mass ranges in this case.

sigma
The mass variance at z, `len=len(M)`

4.1.2 transfer

This module contains a single class, *Transfer*, which provides methods to calculate the transfer function, matter power spectrum and several other related quantities.

class hmf.transfer.**Transfer**(*z=0.0*, *lnk_min=-18.420680743952367*, *lnk_max=9.9034875525361272*, *dlnk=0.05*, *transfer_fit=<class 'hmf.transfer.CAMB'>*, *transfer_options={}*, *takahashi=True*, ***kwargs*)

Neatly deals with different transfer functions and their routines.

The purpose of this class is to calculate transfer functions, power spectra and several tightly associated quantities using many of the available fits from the literature.

Importantly, it contains the means to calculate the transfer function using the popular CAMB code, the Eisenstein-Hu fit (1998), the BBKS fit or the Bond and Efstathiou fit (1984). Furthermore, it can calculate non-linear corrections using the halofit model (with updated parameters from Takahashi2012).

The primary feature of this class is to wrap all the methods into a unified interface. On top of this, the class implements optimized updates of parameters which is useful in, for example, MCMC code which covers a large parameter-space. Calling the *nonlinear_power* does not re-evaluate the entire transfer function, rather it just calculates the corrections, improving performance.

To update parameters optimally, use the update() method. All output quantities are calculated only when needed (but stored after first calculation for quick access).

Parameters **lnk_min** : float

Defines min log wavenumber, *k* [units $hMpc^{-1}$].

lnk_max : float

Defines max log wavenumber, *k* [units $hMpc^{-1}$].

dlnk : float

Defines log interval between wavenumbers

z : float, optional, default `0.0`

The redshift of the analysis.

wdm_mass : float, optional, default `None`

The warm dark matter particle size in *keV*, or `None` for CDM.

transfer_fit : str, { `"CAMB"`, `"EH"`, `"bbks"`, `"bond_efs"`}

Defines which transfer function fit to use. If not defined from the listed options, it will be treated as a filename to be read in. In this case the file must contain a transfer function in CAMB output format.

Scalar_initial_condition : int, {1,2,3,4,5}

(CAMB-only) Initial scalar perturbation mode (adiabatic=1, CDM iso=2, Baryon iso=3,neutrino density iso =4, neutrino velocity iso = 5)

lAccuracyBoost : float, optional, default `1.0`

(CAMB-only) Larger to keep more terms in the hierarchy evolution

AccuracyBoost : float, optional, default `1.0`

(CAMB-only) Increase accuracy_boost to decrease time steps, use more k values, etc.Decrease to speed up at cost of worse accuracy. Suggest 0.8 to 3.

w_perturb : bool, optional, default `False`

(CAMB-only)

transfer__k_per_logint : int, optional, default `11`

(CAMB-only) Number of wavenumbers estimated per log interval by CAMB Default of 11 gets best performance for requisite accuracy of mass function.

transfer__kmax : float, optional, default `0.25`

(CAMB-only) Maximum value of the wavenumber. Default of 0.25 is high enough for requisite accuracy of mass function.

ThreadNum : int, optional, default `0`

(CAMB-only) Number of threads to use for calculation of transfer function by CAMB. Default 0 automatically determines the number.

takahashi : bool, default `True`

Whether to use updated HALOFIT coefficients from Takahashi+12

wdm_model : WDM subclass or string

The WDM transfer function model to use

kwargs : keywords

The `**kwargs` take any cosmological parameters desired, which are input to the *hmf.cosmo.Cosmology* class. *hmf.Perturbations* uses a default parameter set from the first-year PLANCK mission, with optional modifications by the user. Here is a list of parameters currently available (and their defaults in *Transfer*):

sigma_8 [0.8344] The normalisation. Mass variance in top-hat spheres with $R = 8Mpch^{-1}$

n [0.9624] The spectral index

w [-1] The dark-energy equation of state

cs2_lam [1] The constant comoving sound speed of dark energy

t_cmb [2.725] Temperature of the CMB

y_he [0.24] Helium fraction

N_nu [3.04] Number of massless neutrino species

N_nu_massive [0] Number of massive neutrino species

delta_c [1.686] The critical overdensity for collapse

H0 [67.11] The hubble constant

h [`H0/100.0`] The hubble parameter

omegan [0] The normalised density of neutrinos

omegab_h2 [0.022068] The normalised baryon density by `h**2`

omegac_h2 [0.12029] The normalised CDM density by `h**2`

omegav [0.6825] The normalised density of dark energy

omegab [`omegab_h2/h**2`] The normalised baryon density

omegac [`omegac_h2/h**2`] The normalised CDM density

force_flat [False] Whether to force the cosmology to be flat (affects only `omegav`)

default [`"planck1_base"`] A default set of cosmological parameters

Attributes

`H0`	
`N_nu`	
`N_nu_massive`	
`cosmolopy_dict`	Collect parameters into a dictionary suitable for cosmolopy.
`cs2_lam`	
`delta_k`	Dimensionless power spectrum, $\Delta_k = \frac{k^3 P(k)}{2\pi^2}$
`dlnk`	
`force_flat`	
`growth`	The growth factor $d(z)$
`h`	
`lnk`	
`lnk_max`	
`lnk_min`	
`mean_dens`	
`n`	
`nonlinear_delta_k`	Dimensionless nonlinear power spectrum, $\Delta_k = \frac{k^3 P_{\rm nl}(k)}{2\pi^2}$
`nonlinear_power`	Non-linear log power [units Mpc^3/h^3]
`omegab`	
`omegab_h2`	
`omegac`	
`omegac_h2`	
`omegak`	
`omegam`	
`omegan`	
`omegav`	
`power`	Normalised log power spectrum [units Mpc^3/h^3]
`pycamb_dict`	Collect parameters into a dictionary suitable for pycamb.
`sigma_8`	
`t_cmb`	
`takahashi`	
`tau`	
`transfer_fit`	
`transfer_options`	
`w`	
`y_he`	
`z`	
`z_reion`	

Methods

`cosmo_update`(**kwargs)	
`update`(**kwargs)	Update the class optimally with given arguments.

delta_k
Dimensionless power spectrum, $\Delta_k = \frac{k^3 P(k)}{2\pi^2}$

growth
The growth factor $d(z)$

This is calculated (see Lukic 2007) as

$$d(z) = \frac{D^+(z)}{D^+(z=0)}$$

where

$$D^+(z) = \frac{5\Omega_m}{2}\frac{H(z)}{H_0}\int_z^\infty \frac{(1+z')dz'}{[H(z')/H_0]^3}$$

and

$$H(z) = H_0\sqrt{\Omega_m(1+z)^3 + (1-\Omega_m)}$$

nonlinear_delta_k
Dimensionless nonlinear power spectrum, $\Delta_k = \frac{k^3 P_{\rm nl}(k)}{2\pi^2}$

nonlinear_power
Non-linear log power [units Mpc^3/h^3]

Non-linear corrections come from HALOFIT (Smith2003) with updated parameters from Takahashi2012.

This code was heavily influenced by the HaloFit class from the *chomp* python package by Christopher Morrison, Ryan Scranton and Michael Schneider (https://code.google.com/p/chomp/). It has been modified to improve its integration with this package.

power
Normalised log power spectrum [units Mpc^3/h^3]

update(***kwargs*)
Update the class optimally with given arguments.

Accepts any argument that the constructor takes

hmf.transfer.**get_transfer**(*name*, *t*)
A function that chooses the correct Profile class and returns it

4.1.3 cosmo

class hmf.cosmo.**Cosmology**(*default='planck1_base'*, *force_flat=False*, ***kwargs*)
A class that nicely deals with cosmological parameters.

Most cosmological parameters are merely input and exposed as attributes in the class. However, more complicated relations such as the interrelation of omegab, omegac, omegam, omegav for example are dealt with in a more robust manner.

The secondary purpose of this class is to provide simple mappings of the parameters to common python cosmology libraries (for now just *cosmolopy* and *pycamb*). It has been found by the authors that using more than one library becomes confusing in terms of naming all the parameters, so this class helps deal with that.

Note: There are an incredible number of possible combinations of input parameters, many of which could easily be inconsistent. To this end, this class raises an exception if an inconsistent parameter combination is input, eg. h = 1.0, omegab = 0.05, omegab_h2 = 0.06.

Note: *force_flat* is provided for convenience to ensure a flat cosmology. In nearly all cases (except where it would be quite perverse to do so) this will modify omegav if it is otherwise inconsistent. Eg. if `omegam = 0.3, omegav = 0.8, force_flat = True` is passed, the omegav will modified to 0.7.

Parameters **default** : str, {`"planck1_base"`}

Defines a set of default parameters, based on a published set from WMAP or Planck. These defaults are applied in a smart way, so as not to override any user-set parameters.

Current options are

1. `"planck1_base"`: The cosmology of first-year PLANCK mission (with no lensing or WP)

force_flat : bool, default `False`

If `True`, enforces a flat cosmology ($\Omega_m + \Omega_\lambda = 1$). This will modify `omegav` only, never `omegam`.

****kwargs :**

The list of available keyword arguments is as follows:

1. `sigma_8`: The normalisation. Mass variance in top-hat spheres with $R = 8Mpch^{-1}$
2. `n`: The spectral index
3. `w`: The dark-energy equation of state
4. `cs2_lam`: The constant comoving sound speed of dark energy
5. `t_cmb`: Temperature of the CMB
6. `y_he`: Helium fraction
7. `N_nu`: Number of massless neutrino species
8. `N_nu_massive`:Number of massive neutrino species
9. `z_reion`: Redshift of reionization
10. `tau`: Optical depth at reionization
11. `delta_c`: The critical overdensity for collapse
12. `h`: The hubble parameter
13. `H0`: The hubble constant
14. `omegan`: The normalised density of neutrinos
15. `omegam`: The normalised density of matter
16. `omegav`: The normalised density of dark energy
17. `omegab`: The normalised baryon density
18. `omegac`: The normalised CDM density
19. `omegab_h2`: The normalised baryon density by `h**2`
20. `omegac_h2`: The normalised CDM density by `h**2`

Note: The reason these are implemented as *kwargs* rather than the usual arguments, is because the code can't tell *a priori* which combination of density parameters the user will input.

Attributes

H0	
N_nu	
N_nu_massive	
cosmolopy_dict	Collect parameters into a dictionary suitable for cosmolopy.
cs2_lam	
force_flat	
h	
mean_dens	
n	
omegab	
omegab_h2	
omegac	
omegac_h2	
omegak	
omegam	
omegan	
omegav	
pycamb_dict	Collect parameters into a dictionary suitable for pycamb.
sigma_8	
t_cmb	
tau	
w	
y_he	
z_reion	

Methods

cosmo_update(**kwargs)

cosmolopy_dict
Collect parameters into a dictionary suitable for cosmolopy.

Returns dict

Dictionary of values appropriate for cosmolopy

pycamb_dict
Collect parameters into a dictionary suitable for pycamb.

Returns dict

Dictionary of values appropriate for pycamb

4.1.4 fitting_functions

class hmf.fitting_functions.**Angulo**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Class representing a Angulo mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Angulo *[R1]* form is:

$$f_{\rm Ang}(\sigma) = A\left[\left(\frac{e}{\sigma}\right)^b + c\right]\exp(\frac{d}{\sigma^2})$$

References

arXiv:1203.3216v1

[R1]

Methods

`fsigma`(cut_fit)

class hmf.fitting_functions.**Bhattacharya**(***model_parameters*)

Class representing a Bhattacharya mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Bhattacharya *[R2]* form is:

$$f_{\rm Btc}(\sigma) = f_{\rm SMT}(\sigma)(\nu\sqrt{a})^q$$

References

http://labs.adsabs.harvard.edu/ui/abs/2011ApJ...732..122B

[R2]

Methods

fsigma(cut_fit)	Calculate $f(\sigma)$ for Bhattacharya form.

fsigma(*cut_fit*)

Calculate $f(\sigma)$ for Bhattacharya form.

Bhattacharya, S., et al., May 2011. ApJ 732 (2), 122. http://labs.adsabs.harvard.edu/ui/abs/2011ApJ...732..122B

Note: valid for $10^{11.8} M_\odot < M < 10^{15.5} M_\odot$

Returns **vfv** : array_like, len=len(pert.M)

The function :math:‘f(sigma)equiv

u f(

u)‘ defined on `pert.M`

class `hmf.fitting_functions.`**`Courtin`**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Class representing a Courtin mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Courtin *[R3]* form is:

$$f_{\rm Ctn}(\sigma) = A\sqrt{2a/\pi}\nu\exp(-a\nu^2/2)(1+(a\nu^2)^{-p})$$

References

http://doi.wiley.com/10.1111/j.1365-2966.2010.17573.x

[R3]

Methods

`fsigma`(cut_fit)	

class hmf.fitting_functions.**Crocce**(***model_parameters*)

Class representing a Crocce mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Crocce *[R4]* form is:

$$f_{\rm Cro}(\sigma) = A\left[\left(\frac{e}{\sigma}\right)^b + c\right]\exp(\frac{d}{\sigma^2})$$

References

http://doi.wiley.com/10.1111/j.1365-2966.2009.16194.x

[R4]

Methods

`fsigma`(cut_fit)

class hmf.fitting_functions.**FittingFunction**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Base-class for a halo mass function fit

This class should not be called directly, rather use a subclass which is specific to a certain fitting formula.

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Methods

fsigma(cut_fit)	Calculate $f(\sigma) \equiv \nu f(\nu)$.

fsigma(*cut_fit*)
Calculate $f(\sigma) \equiv \nu f(\nu)$.

Parameters **cut_fit** : bool

Whether to cut the fit at bounds corresponding to the fitted range (in mass or corresponding unit, not redshift). If so, values outside this range will be set to `NaN`.

Returns **vfv** : array_like, `len=len(self.M)`

The function f(sigma).

class `hmf.fitting_functions.`**Jenkins**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Class representing a Jenkins mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Jenkins *[R5]* form is:

$$f_{\rm Jenkins}(\sigma) = A\exp(-|\ln\sigma^{-1}+b|^c)$$

References

http://doi.wiley.com/10.1046/j.1365-8711.2001.04029.x

[R5]

Methods

`fsigma`(cut_fit)

class `hmf.fitting_functions.`**PS**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Class representing a Press-Schechter mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Press-Schechter *[R6]* form is:

$$f_{\rm PS}(\sigma) = \sqrt{\frac{2}{\pi}}\nu \exp(-0.5\nu^2)$$

References

http://adsabs.harvard.edu/full/1974ApJ...187..425P

[R6]

Methods

`fsigma`(cut_fit)

class `hmf.fitting_functions.`**`Peacock`**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Class representing a Peacock mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Peacock *[R7]* form is:

$$f_{\rm Pck}(\sigma) = \nu \exp(-c\nu^2)(2cd\nu + ba\nu^{b-1})/d^2$$

References

http://adsabs.harvard.edu/abs/2007MNRAS.379.1067P

[R7]

Methods

`fsigma`(cut_fit)

class `hmf.fitting_functions.`**`Reed03`**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Class representing a Reed03 mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Reed03 *[R8]* form is:

$$f_{\rm R03}(\sigma) = f_{\rm SMT}(\sigma)\exp\left(-\frac{c}{\sigma\cosh^5(2\sigma)}\right)$$

References

http://adsabs.harvard.edu/abs/2003MNRAS.346..565R

[R8]

Methods

`fsigma`(cut_fit)

class `hmf.fitting_functions.`**`Reed07`**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Class representing a Reed07 mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Reed07 *[R9]* form is:

$$f_{\rm R07}(\sigma) = A\sqrt{2a/\pi}\left[1 + (\frac{1}{a\nu^2})^p + 0.6G_1 + 0.4G_2\right]\nu\exp(-ca\nu^2/2 - \frac{0.03\nu^{0.6}}{(n_{\rm eff}+3)^2})$$

References

http://adsabs.harvard.edu/abs/2007MNRAS.374....2R

[R9]

Methods

`fsigma`(cut_fit)

class `hmf.fitting_functions.`**`SMT`**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Class representing a Sheth-Mo-Tormen mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Sheth-Mo-Tormen *[R10]* form is:

$$f_{\rm SMT}(\sigma) = A\sqrt{2a/\pi}\nu\exp(-a\nu^2/2)(1 + (a\nu^2)^{-p})$$

References

http://doi.wiley.com/10.1046/j.1365-8711.2001.04006.x

[R10]

Methods

`fsigma`(cut_fit)

class `hmf.fitting_functions.`**`Tinker08`**(***model_parameters*)

Class representing a Tinker08 mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Tinker08 *[R11]* form is:

$$f_{\rm Tkr}(\sigma) = A(\frac{\sigma}{b}^{-a} + 1)\exp(-c/\sigma^2)$$

References

http://iopscience.iop.org/0004-637X/688/2/709

[R11]

Methods

`fsigma`(cut_fit)

class `hmf.fitting_functions.`**`Warren`**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Class representing a Warren mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Warren *[R12]* form is:

$$f_{\rm Warren}(\sigma) = A\left[\left(\frac{e}{\sigma}\right)^b + c\right]\exp(\frac{d}{\sigma^2})$$

References

http://adsabs.harvard.edu/abs/2006ApJ...646..881W

[R12]

Methods

`fsigma`(cut_fit)

class hmf.fitting_functions.**Watson**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Class representing a Watson mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The Watson *[R13]* form is:

$$f_{\rm WatS}(\sigma) = \Gamma A \left((\frac{\beta}{\sigma}^{\alpha} + 1 \right) \exp(-\gamma/\sigma^2)$$

References

http://arxiv.org/abs/1212.0095

[R13]

Methods

`fsigma`(cut_fit)	
gamma()	Calculate Γ for the Watson fit.

gamma()

Calculate Γ for the Watson fit.

class hmf.fitting_functions.**Watson_FoF**(*M*, *nu2*, *delta_c*, *sigma=None*, *n_eff=None*, *lnsigma=None*, *z=0*, *delta_halo=200*, *cosmo=None*, *omegam_z=None*, ***model_parameters*)

Class representing a WatsonFoF mass function fit

Parameters **M** : array

A vector of halo masses [units M_sun/h]

nu2 : array

A vector of peak-heights, δ_c^2/σ^2 corresponding to `M`

z : float, optional

The redshift.

delta_halo : float, optional

The overdensity of the halo w.r.t. the mean density of the universe.

cosmo : `cosmo.Cosmology` instance, optional

A cosmology. Default is the default provided by the `cosmo.Cosmology` class. Not required if `omegam_z` is passed.

omegam_z : float, optional

A value for the mean matter density at the given redshift `z`. If not provided, will be calculated using the value of `cosmo`.

****model_parameters** : unpacked-dictionary

These parameters are model-specific. For any model, list the available parameters (and their defaults) using `<model>._defaults`

Notes

The WatsonFoF *[R14]* form is:

$$f_{\rm WatF}(\sigma) = A\left[\left(\frac{e}{\sigma}\right)^b + c\right]\exp(\frac{d}{\sigma^2})$$

References

http://arxiv.org/abs/1212.0095

[R14]

Methods

`fsigma`(cut_fit)

hmf.fitting_functions.**get_fit**(*name*, ***kwargs*)

Returns the correct subclass of *FittingFunction*.

Parameters **name** : str

The class name of the appropriate fit

****kwargs :**

Any parameters for the instantiated fit (including model parameters)

4.1.5 tools

A collection of functions which do some of the core work of the HMF calculation.

The routines here could be made more 'elegant' by taking *MassFunction* or *Transfer* objects as arguments, but we keep them simple for the sake of flexibility.

`hmf.tools.`**`check_kr`**(*min_m*, *max_m*, *mean_dens*, *mink*, *maxk*)
Check the bounds of the product of k*r

If the bounds are not high/low enough, then there can be information loss in the calculation of the mass variance. This routine returns a warning indicating the necessary adjustment for requisite accuracy.

See http://arxiv.org/abs/1306.6721 for details.

`hmf.tools.`**`d_plus`**(*z*, *cdict*, *getvec=False*)
Finds the factor $D^+(a)$, from Lukic et. al. 2007, eq. 8.

Uses simpson's rule to integrate, with 1000 steps.

Parameters **z** : float

The redshift

cosmo : `hmf.cosmo.Cosmology()` object

Cosmological parameters

Returns **dplus** : float

The un-normalised growth factor.

`hmf.tools.`**`growth_factor`**(*z*, *cdict*, *getvec=False*)
Calculate $d(a) = D^+(a)/D^+(a=1)$, from Lukic et. al. 2007, eq. 7.

Parameters **z** : float

The redshift

cosmo : `hmf.cosmo.Cosmology()` object

Cosmological parameters

Returns **growth** : float

The normalised growth factor.

`hmf.tools.`**`normalize`**(*norm_sigma_8*, *unn_power*, *lnk*, *mean_dens*)
Normalize the power spectrum to a given σ_8

Parameters **norm_sigma_8** : float

The value of σ_8 to normalize to.

unn_power : array_like

The natural logarithm of the unnormalised power spectrum

lnk : array_like

The natural logarithm of the values of *k/h* at which *unn_power* is defined.

mean_dens : float

The mean density of the Universe.

Returns **power** : array_like

An array of the same length as *unn_power* in which the values are normalised to :math:‘‘sigma_8‘

normalisation : float

The normalisation constant.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

[R1] Angulo, R. E., et al., 2012.

[R2] Bhattacharya, S., et al., May 2011. ApJ 732 (2), 122.

[R3] Courtin, J., et al., Oct. 2010. MNRAS 1931

[R4] Crocce, M., et al. MNRAS 403 (3), 1353-1367.

[R5] Jenkins, A. R., et al., Feb. 2001. MNRAS 321 (2), 372-384.

[R6] Press, W. H., Schechter, P., 1974. ApJ 187, 425-438.

[R7] Peacock, J. A., Aug. 2007. MNRAS 379 (3), 1067-1074.

[R8] Reed, D., et al., Dec. 2003. MNRAS 346 (2), 565-572.

[R9] Reed, D. S., et al., Jan. 2007. MNRAS 374 (1), 2-15.

[R10] Sheth, R. K., Mo, H. J., Tormen, G., May 2001. MNRAS 323 (1), 1-12.

[R11] Tinker, J., et al., 2008. ApJ 688, 709-728.

[R12] Warren, M. S., et al., Aug. 2006. ApJ 646 (2), 881-885.

[R13] Watson, W. A., et al., Dec. 2012.

[R14] Watson, W. A., et al., Dec. 2012.

h

T

U

W