# Dark Emulator

*Release 0.0.0*

**Takahiro Nishimichi, Hironao Miyatake, Sunao Sugiyama**

**Apr 14, 2023**

# CONTENTS:

Dark emulator is a cosmology code for calculating summary statistics of large scale structure constructed as a part of Dark Quest Project (https://darkquestcosmology.github.io/). The "dark_emulator" python package enables fast and accurate computations of halo clustering quantities. The current version supports the halo mass function, halo-matter cross-correlation, and halo auto-correlation as a function of halo masses, redshift, separations and cosmological models.

See the *darkemu module tutorial notebook* for an introductory set of examples of how to use the DarkEmulator package. This is usually the fastest way to learn how to use it and quickly see some of the capabilities. When you use the HOD module, you can quickly learn the module with *model_hod module tutorial notebook*.

In order to install dark emulator package, use pip:

```
pip install dark_emulator
```

or use conda:

```
conda install -c nishimichi dark_emulator
```

CHAPTER

ONE

# DARK_EMULATOR PACKAGE MODULES

## 1.1 dark_emulator package

### 1.1.1 Subpackages

#### dark_emulator.darkemu package

#### Submodules

#### dark_emulator.darkemu.de_interface module

**class** dark_emulator.darkemu.de_interface.**base_class**

Bases: `object`

The base class of dark emulator. This holds all the individual emulator class objects for different statistical quantities. By passing to the base class object, the cosmological paramters in all the lower-level objects are updated.

**Parameters**

**cparam** (*numpy array*) – Cosmological parameters $(\omega_b, \omega_c, \Omega_{de}, \ln(10^{10}A_s), n_s, w)$

**cosmo**

A class object dealing with the cosmological parameters and some basic cosmological quantities such as expansion and linear growth.

**Type**

class cosmo_class

**pkL**

A class object that takes care of the linear matter power spectrum

**Type**

class pklin_gp

**g1**

A class object that takes care of the large-scale bias as well as the BAO damping

**Type**

class gamma1_gp

**xi_cross**

A class object that takes care of the halo-matter cross correlation function

**Type**
class cross_gp

`xi_auto`

A class object that takes care of the halo-halo correlation function

**Type**
class auto_gp

`massfunc`

A class object that takes care of the halo mass function

**Type**
class hmf_gp

`xiNL`

A class object that takes care of the nonlinear matter correlation function (experimental)

**Type**
class xinl_gp

`Dgrowth_from_a`($a$)

Compute the linear growth factor, D_+, at scale factor a. Normalized to unity at z=0.

**Parameters**
**a** – scale factor normalized to unity at present.

**Returns**
linear growth factor

**Return type**
float

`Dgrowth_from_z`($z$)

Compute the linear growth factor, D_+, at redshift z. Normalized to unity at z=0.

**Parameters**
**z** – redshift

**Returns**
linear growth factor

**Return type**
float

`dens_to_mass`(*dens*, *redshift*, *nint=20*, *integration='quad'*)

Convert the cumulative number density to the halo mass threshold for the current cosmological model at redshift z.

**Parameters**

- **dens** (*float*) – halo number density in $(h^{-1}\mathrm{Mpc})^{-3}$
- **redshift** (*float*) – redshift
- **nint** (*int, optional*) – number of sampling points in log(M) used for interpolation
- **integration** (*str, optional*) – type of integration (default: “quad”, “trapz” is also supported)

**Returns**
mass threshold in $[h^{-1}M_{\odot}]$

**Return type**
float

`f_from_a`($a$)

Compute the linear growth rate, $f = \mathrm{d}\ln D_+/\mathrm{d}\ln a$, at scale factor a.

**Parameters**
**a** – scale factor normalized to unity at present.

**Returns**
linear growth rate

**Return type**
float

`f_from_z`($z$)

Compute the linear growth rate, $f = \mathrm{d}\ln D_+/\mathrm{d}\ln a$, at redshift z.

**Parameters**
**z** – redshift

**Returns**
linear growth rate

**Return type**
float

`get_DeltaSigma`(*R2d*, *logdens*, *redshift*)

Compute the halo-galaxy lensing signal, the excess surface mass density, $\Delta\Sigma(R; n_h)$, for a mass threshold halo sample specified by the corresponding cumulative number density.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $h^{-1}\mathrm{Mpc}$
- **logdens** (*float*) – Logarithm of the cumulative halo number density taken from the most massive, $\log_{10}[n_h/(h^{-1}\mathrm{Mpc})^3]$
- **redshift** (*float*) – redshift at which the lens halos are located

**Returns**
excess surface mass density in $[hM_\odot \mathrm{pc}^{-2}]$

**Return type**
numpy array

`get_DeltaSigma_mass`(*R2d*, *M*, *redshift*)

Compute the halo-galaxy lensing signal, the excess surface mass density, $\Delta\Sigma(R; M)$, for halos with mass $M$.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $h^{-1}\mathrm{Mpc}$
- **M** (*float*) – Halo mass in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the lens halos are located

**Returns**
excess surface mass density in $[hM_\odot \mathrm{pc}^{-2}]$

**Return type**
numpy array

**get_DeltaSigma_massthreshold**(*R2d*, *Mthre*, *redshift*)

Compute the halo-galaxy lensing signal, the excess surface mass density, $\Delta\Sigma(R; > M_{\rm th})$, for a mass threshold halo sample.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $h^{-1}{\rm Mpc}$
- **Mthre** (*float*) – Minimum halo mass threshold in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the lens halos are located

**Returns**
excess surface mass density in $[hM_\odot{\rm pc}^{-2}]$

**Return type**
numpy array

**get_Sigma**(*R2d*, *logdens*, *redshift*)

Compute the surface mass density, $\Sigma(R; n_h)$, for a mass threshold halo sample specified by the corresponding cumulative number density.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $h^{-1}{\rm Mpc}$
- **logdens** (*float*) – Logarithm of the cumulative halo number density taken from the most massive, $\log_{10}[n_h/(h^{-1}{\rm Mpc})^3]$
- **redshift** (*float*) – redshift at which the lens halos are located

**Returns**
surface mass density in $[hM_\odot{\rm pc}^{-2}]$

**Return type**
numpy array

**get_Sigma_mass**(*R2d*, *M*, *redshift*)

Compute the surface mass density, $\Sigma(R; M)$, for halos with mass $M$.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $h^{-1}{\rm Mpc}$
- **M** (*float*) – Halo mass in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the lens halos are located

**Returns**
surface mass density in $[hM_\odot{\rm pc}^{-2}]$

**Return type**
numpy array

**get_Sigma_massthreshold**(*R2d*, *Mthre*, *redshift*)

Compute the surface mass density, $\Sigma(R; > M_{\rm th})$, for a mass threshold halo sample.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $h^{-1}{\rm Mpc}$
- **Mthre** (*float*) – Minimum halo mass threshold in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the lens halos are located

**Returns**
surface mass density in $[hM_{\odot}\mathrm{pc}^{-2}]$

**Return type**
numpy array

`get_bias`(*logdens*, *redshift*)

Compute the linear bias for a mass threshold halo sample specified by the corresponding cumulative number density.

**Parameters**

- **logdens** (*float*) – Logarithm of the cumulative halo number density taken from the most massive, $\log_{10}[n_h/(h^{-1}\mathrm{Mpc})^3]$
- **redshift** (*float*) – redshift at which the lens halos are located

**Returns**
linear bias factor

**Return type**
float

`get_bias_mass`(*M*, *redshift*)

Compute the linear bias for halos with mass $M$.

**Parameters**

- **M** (*float*) – Halo mass in $[h^{-1}M_{\odot}]$
- **redshift** (*float*) – redshift at which the lens halos are located

**Returns**
linear bias factor

**Return type**
float

`get_bias_massthreshold`(*Mth*, *redshift*)

Compute the linear bias, $b(> M_{\mathrm{th}})$, for a mass threshold halo sample.

**Parameters**

- **Mth** (*float*) – Halo mass threshold in $[h^{-1}M_{\odot}]$
- **redshift** (*float*) – redshift at which the lens halos are located

**Returns**
linear bias factor

**Return type**
float

`get_cosmology`()

Obtain the cosmological parameters currently set to the emulator.

**Returns**
Cosmological parameters $(\omega_b, \omega_c, \Omega_{de}, \ln(10^{10}A_s), n_s, w)$

**Return type**
numpy array

**get_f_HMF**(*redshift*)

Compute the multiplicity function $f(\sigma)$, defined through $dn/dM = f(\sigma)\bar{\rho}_m/M d\ln\sigma^{-1}/dM$.

**Parameters**
**redshift** (*float*) – redshift

**Returns**

tuple containing:

mass(numpy array): $M_{200b}$

mass variance(numpy array): $\sigma(M_{200b})$

multiplicity function(numpy array): $f(\sigma)$

**Return type**
(tuple)

**get_nhalo**(*Mmin*, *Mmax*, *vol*, *redshift*)

Compute the mean number of halos in a given mass range and volume.

**Parameters**

- **Mmin** (*float*) – Minimum halo mass in $[h^{-1}M_\odot]$
- **Mmax** (*float*) – Maximum halo mass in $[h^{-1}M_\odot]$
- **vol** (*float*) – Volume in $[(h^{-1}\mathrm{Mpc})^3]$

**Returns**
Number of halos

**Return type**
float

**get_nhalo_tinker**(*Mmin*, *Mmax*, *vol*, *redshift*)

Compute the mean number of halos in a given mass range and volume based on the fitting formula by Tinker et al. (ApJ 688 (2008) 709).

**Parameters**

- **Mmin** (*float*) – Minimum halo mass in $[h^{-1}M_\odot]$
- **Mmax** (*float*) – Maximum halo mass in $[h^{-1}M_\odot]$
- **vol** (*float*) – Volume in $[(h^{-1}\mathrm{Mpc})^3]$

**Returns**
Number of halos

**Return type**
float

**get_phh**(*ks*, *logdens1*, *logdens2*, *redshift*)

Compute the halo-halo power spectrum $P_{hh}(k; n_1, n_2)$ between 2 mass threshold halo samples specified by the corresponding cumulative number densities.

**Parameters**

- **ks** (*numpy array*) – Wavenumbers in $[h\mathrm{Mpc}^{-1}]$
- **logdens1** (*float*) – Logarithm of the cumulative halo number density of the first halo sample taken from the most massive, $\log_{10}[n_1/(h^{-1}\mathrm{Mpc})^3]$

- **logdens2** (*float*) – Logarithm of the cumulative halo number density of the second halo sample taken from the most massive, $\log_{10}[n_2/(h^{-1}\mathrm{Mpc})^3]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
halo power spectrum in $[(h^{-1}\mathrm{Mpc})^3]$

**Return type**
numpy array

**get_phh_mass**(*ks*, *M1*, *M2*, *redshift*)

Compute the halo-halo power spectrum $P_{hh}(k; M_1, M_2)$ between 2 halo samples with mass $M_1$ and $M_2$.

**Parameters**

- **ks** (*numpy array*) – Wavenumbers in $[h\mathrm{Mpc}^{-1}]$
- **M1** (*float*) – Halo mass of the first sample in $[h^{-1}M_\odot]$
- **M2** (*float*) – Halo mass of the second sample in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
halo power spectrum in $[(h^{-1}\mathrm{Mpc})^3]$

**Return type**
numpy array

**get_phh_massthreshold**(*ks*, *Mthre*, *redshift*)

Compute the halo-halo auto power spectrum $P_{hh}(k; > M_{\rm th})$ for a mass threshold halo sample.

**Parameters**

- **ks** (*numpy array*) – Wavenumbers in $[h\mathrm{Mpc}^{-1}]$
- **Mthre** (*float*) – Minimum halo mass threshold in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
halo power spectrum in $[(h^{-1}\mathrm{Mpc})^3]$

**Return type**
numpy array

**get_phm**(*ks*, *logdens*, *redshift*)

Compute the halo-matter cross power spectrum $P_{hm}(k; n_h)$ for a mass threshold halo sample specified by the corresponding cumulative number density.

**Parameters**

- **ks** (*numpy array*) – Wavenumbers in $[h\mathrm{Mpc}^{-1}]$
- **logdens** (*float*) – Logarithm of the cumulative halo number density of the halo sample taken from the most massive, $\log_{10}[n_h/(h^{-1}\mathrm{Mpc})^3]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
Halo-matter cross power spectrum in $[(h^{-1}\mathrm{Mpc})^3]$

**Return type**
numpy array

**get_phm_mass**(*ks*, *M*, *redshift*)

Compute the halo-matter cross power spectrum $P_{hm}(k; M)$ for halos with mass $M$.

**Parameters**

- **ks** (*numpy array*) – Wavenumbers in $[h\mathrm{Mpc}^{-1}]$
- **M** (*float*) – Halo mass in $[h^{-1}M_{\odot}]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
Halo-matter cross power spectrum in $[(h^{-1}\mathrm{Mpc})^3]$

**Return type**
numpy array

**get_phm_massthreshold**(*ks*, *Mthre*, *redshift*)

Compute the halo-matter cross power spectrum $P_{hm}(k; > M_{\mathrm{th}})$ for a mass threshold halo sample.

**Parameters**

- **ks** (*numpy array*) – Wavenumbers in $[h\mathrm{Mpc}^{-1}]$
- **Mthre** (*float*) – Minimum halo mass threshold in $[h^{-1}M_{\odot}]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
Halo-matter cross power spectrum in $[(h^{-1}\mathrm{Mpc})^3]$

**Return type**
numpy array

**get_pklin**(*k*)

Compute the linear matter power spectrum at z=0.

**Parameters**
**k** (*numpy array*) – Wavenumbers in $[h\mathrm{Mpc}^{-1}]$

**Returns**
Linear power spectrum at wavenumbers given in the argument k.

**Return type**
numpy array

**get_pklin_from_z**(*k*, *z*)

get_pklin_z

Compute the linear matter power spectrum.

**Parameters**

- **k** (*numpy array*) – Wavenumbers in $[h\mathrm{Mpc}^{-1}]$
- **z** (*float*) – redshift

**Returns**
Linear power spectrum at wavenumbers given in the argument k.

**Return type**
numpy array

**get_pknl**($k$, $z$)

Compute the nonlinear matter power spectrum. Note that this is still in a development phase, and the accuracy has not yet been fully evaluated.

**Parameters**

- **k** (*numpy array*) – Wavenumbers in $[h\mathrm{Mpc}^{-1}]$
- **z** (*float*) – redshift

**Returns**
Nonlinear matter power spectrum at wavenumbers given in the argument k.

**Return type**
numpy array

**get_sd**($z$)

Compute the root mean square of the linear displacement, $\sigma_d$, for the current cosmological model at redshift z.

**Parameters**
**z** (*float*) – redshift

**Returns**
$\sigma_d$

**Return type**
float

**get_sigma8**(*logkmin=-4*, *logkmax=1*, *nint=100*)

Compute $\sigma_8$ for the current cosmology.

**Parameters**

- **logkmin** (*float, optional*) – log10 of the minimum wavenumber for the integral (default=-4)
- **logkmin** – log10 of the maximum wavenumber for the integral (default=1)
- **nint** (*int, optional*) – Number of samples taken for the trapz integration (default=100)

**Returns**
$\sigma_8$

**Return type**
float

**get_wauto**(*R2d*, *logdens1*, *logdens2*, *redshift*)

Compute the projected halo-halo correlation function $w_{hh}(R; n_1, n_2)$ for 2 mass threshold halo samples specified by the corresponding cumulative number densities.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $[h^{-1}\mathrm{Mpc}]$
- **logdens1** (*float*) – Logarithm of the cumulative halo number density of the first halo sample taken from the most massive, $\log_{10}[n_1/(h^{-1}\mathrm{Mpc})^3]$
- **logdens2** (*float*) – Logarithm of the cumulative halo number density of the second halo sample taken from the most massive, $\log_{10}[n_2/(h^{-1}\mathrm{Mpc})^3]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
projected halo correlation function in $[h^{-1}\mathrm{Mpc}]$

**Return type**
numpy array

**get_wauto_cut**(*R2d*, *logdens1*, *logdens2*, *redshift*, *pimax*, *integration='quad'*)

Compute the projected halo-halo correlation function $w_{hh}(R; n_1, n_2)$ for 2 mass threshold halo samples specified by the corresponding cumulative number densities. Unlike get_wauto, this function considers a finite width for the radial integration, from $-\pi_{\max}$ to $\pi_{\max}$.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $[h^{-1}\mathrm{Mpc}]$
- **logdens1** (*float*) – Logarithm of the cumulative halo number density of the first halo sample taken from the most massive, $\log_{10}[n_1/(h^{-1}\mathrm{Mpc})^3]$
- **logdens2** (*float*) – Logarithm of the cumulative halo number density of the second halo sample taken from the most massive, $\log_{10}[n_2/(h^{-1}\mathrm{Mpc})^3]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated
- **pimax** (*float*) – $\pi_{\max}$ for the upper limit of the integral

**Returns**
projected halo correlation function in $[h^{-1}\mathrm{Mpc}]$

**Return type**
numpy array

**get_wauto_mass**(*R2d*, *M1*, *M2*, *redshift*)

Compute the projected halo-halo correlation function $w_{hh}(R; M_1, M_2)$ for 2 mass threshold halo samples.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $[h^{-1}\mathrm{Mpc}]$
- **M1** (*float*) – Halo mass of the first sample in $[h^{-1}M_\odot]$
- **M2** (*float*) – Halo mass of the second sample in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
projected halo correlation function in $[h^{-1}\mathrm{Mpc}]$

**Return type**
numpy array

**get_wauto_mass_cut**(*R2d*, *M1*, *M2*, *redshift*, *pimax*)

Compute the projected halo-halo correlation function $w_{hh}(R; M_1, M_2)$ for 2 mass threshold halo samples. Unlike get_wauto_mass, this function considers a finite width for the radial integration, from $-\pi_{\max}$ to $\pi_{\max}$.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $[h^{-1}\mathrm{Mpc}]$
- **M1** (*float*) – Halo mass of the first sample in $[h^{-1}M_\odot]$
- **M2** (*float*) – Halo mass of the second sample in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated
- **pimax** (*float*) – $\pi_{\max}$ for the upper limit of the integral

**Returns**
projected halo correlation function in $[h^{-1}\mathrm{Mpc}]$

**Return type**
numpy array

**get_wauto_massthreshold**(*R2d*, *Mthre*, *redshift*)

Compute the projected halo-halo correlation function $w_{hh}(R; > M_{\rm th})$ for a mass threshold halo sample.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $[h^{-1}\mathrm{Mpc}]$
- **Mthre** (*float*) – Minimum halo mass threshold in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
projected halo correlation function in $[h^{-1}\mathrm{Mpc}]$

**Return type**
numpy array

**get_wauto_masthreshold_cut**(*R2d*, *Mthre*, *redshift*, *pimax*, *integration='quad'*)

get_wauto_massthreshold_cut

Compute the projected halo-halo correlation function $w_{hh}(R; > M_{\rm th})$ for a mass threshold halo sample. Unlike get_wauto_massthreshold, this function considers a finite width for the radial integration, from $-\pi_{\rm max}$ to $\pi_{\rm max}$.

**Parameters**

- **R2d** (*numpy array*) – 2 dimensional projected separation in $[h^{-1}\mathrm{Mpc}]$
- **Mthre** (*float*) – Minimum halo mass threshold in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated
- **pimax** (*float*) – $\pi_{\rm max}$ for the upper limit of the integral

**Returns**
projected halo correlation function in $[h^{-1}\mathrm{Mpc}]$

**Return type**
numpy array

**get_xiauto**(*xs*, *logdens1*, *logdens2*, *redshift*)

Compute the halo-halo correlation function, $\xi_{\rm hh}(x; n_1, n_2)$, bwtween 2 mass threshold halo samples specified by the corresponding cumulative number densities.

**Parameters**

- **xs** (*numpy array*) – Separations in $[h^{-1}\mathrm{Mpc}]$
- **logdens1** (*float*) – Logarithm of the cumulative halo number density of the first halo sample taken from the most massive, $\log_{10}[n_1/(h^{-1}\mathrm{Mpc})^3]$
- **logdens2** (*float*) – Logarithm of the cumulative halo number density of the second halo sample taken from the most massive, $\log_{10}[n_2/(h^{-1}\mathrm{Mpc})^3]$
- **redshift** (*float*) – Redshift at which the correlation function is evaluated

**Returns**
Halo correlation function

**Return type**
numpy array

`get_xiauto_mass`(*xs*, *M1*, *M2*, *redshift*)

Compute the halo-halo correlation function, $\xi_{\rm hh}(x; M_1, M_2)$, between 2 halo samples with mass $M_1$ and $M_2$. :param xs: Separations in $[h^{-1}{\rm Mpc}]$ :type xs: numpy array :param M1: Halo mass of the first sample in $[h^{-1}M_\odot]$ :type M1: float :param M2: Halo mass of the second sample in $[h^{-1}M_\odot]$ :type M2: float :param redshift: Redshift at which the correlation function is evaluated :type redshift: float

**Returns**
Halo correlation function

**Return type**
numpy array

`get_xiauto_massthreshold`(*xs*, *Mthre*, *redshift*)

Compute the halo-halo correlation function, $\xi_{\rm hh}(x; > M_{\rm th})$, for a mass threshold halo sample.

**Parameters**

- **xs** (*numpy array*) – Separations in $[h^{-1}{\rm Mpc}]$
- **Mthre** (*float*) – Minimum halo mass threshold in $[h^{-1}M_\odot]$
- **redshift** (*float*) – Redshift at which the correlation function is evaluated

**Returns**
Halo correlation function

**Return type**
numpy array

`get_xicross`(*xs*, *logdens*, *redshift*)

Compute the halo-matter cross correlation function $\xi_{hm}(x; n_h)$ for a mass threshold halo sample specified by the corresponding cumulative number density.

**Parameters**

- **xs** (*numpy array*) – Separations in $[h^{-1}{\rm Mpc}]$
- **logdens** (*float*) – Logarithm of the cumulative halo number density of the halo sample taken from the most massive, $\log_{10}[n_h/(h^{-1}{\rm Mpc})^3]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
Halo-matter cross correlation function

**Return type**
numpy array

`get_xicross_mass`(*xs*, *M*, *redshift*)

Compute the halo-matter cross correlation function $\xi_{hm}(x; M)$ for halos with mass $M$.

**Parameters**

- **xs** (*numpy array*) – Separations in $[h^{-1}{\rm Mpc}]$
- **M** (*float*) – Halo mass in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
Halo-matter cross correlation function

**Return type**
numpy array

`get_xicross_massthreshold`(*xs*, *Mthre*, *redshift*)

Compute the halo-matter cross correlation function $\xi_{hm}(x; > M_{\rm th})$ for a mass threshold halo sample.

**Parameters**

- **xs** (*numpy array*) – Separations in $[h^{-1}{\rm Mpc}]$
- **Mthre** (*float*) – Minimum mass threshold of a halo sample in $[h^{-1}M_\odot]$
- **redshift** (*float*) – redshift at which the power spectrum is evaluated

**Returns**
Halo-matter cross correlation function

**Return type**
numpy array

`get_xilin`(*xs*)

Compute the linear matter correlation function at z=0.

**Parameters**
**xs** (*numpy array*) – Separations in $[h^{-1}{\rm Mpc}]$

**Returns**
Correlation function at separations given in the argument xs.

**Return type**
numpy array

`get_xinl`(*xs*, *redshift*)

Compute the nonlinear matter correlation function. Note that this is still in a development phase, and the accuracy has not yet been fully evaluated.

**Parameters**
**xs** (*numpy array*) – Separations in $[h^{-1}{\rm Mpc}]$

**Returns**
Correlation function at separations given in the argument xs.

**Return type**
numpy array

`mass_to_dens`(*mass_thre*, *redshift*, *integration='quad'*)

Convert the halo mass threshold to the cumulative number density for the current cosmological model at redshift z.

**Parameters**

- **mass_thre** (*float*) – mass threshold in $h^{-1}M_\odot$
- **redshift** (*float*) – redshift
- **integration** (*str, optional*) – type of integration (default: “quad”, “trapz” is also supported)

**Returns**
halo number density in $[(h^{-1}{\rm Mpc})^{-3}]$

**Return type**
float

`set_cosmology`(*cparam*)

Let the emulator know the cosmological parameters. This interface passes the 6 parameters to all the class objects used for the emulation of various halo statistics.

The current version supports wCDM cosmologies specified by the 6 parameters as described below. Other parameters are automatically computed:

$\Omega_m = 1 - \Omega_{de}$,

$h = \sqrt{(\omega_b + \omega_c + \omega_\nu)/\Omega_m}$,

where the neutrino density is fixed by $\omega_\nu = 0.00064$ corresponding to the mass sum of 0.06 eV.

**Parameters**

**cparam** (*numpy array*) – Cosmological parameters $(\omega_b, \omega_c, \Omega_{de}, \ln(10^{10} A_s), n_s, w)$

## Module contents

## dark_emulator.model_hod package

## Submodules

## dark_emulator.model_hod.hod_interface module

dark_emulator.model_hod.hod_interface.**binave_array**(*x*, *y*, *dlnx*, *D=2*, *nbin=100*)

Assumes dlnx << np.diff(x). Performs the forward bin average in dimension D. ::math:

```
\bar{y} = \frac{1}{d\ln x} \int_{\ln x}^{\ln x+d\ln x} x^D y(x)
```

**class** dark_emulator.model_hod.hod_interface.**darkemu_x_hod**(*config=None*)

Bases: *base_class*

This class holds cosmological parameters (see `set_cosmology()`), HOD parameters, and other galaxy parameters (see `set_galaxy()`), and computes galaxy-galaxy lensing, galaxy-galaxy clustering signal, and related correlation functions based on these parameters. This class can be initialized through a dictionary that specifies the following configurations. With the default values, one can get $\Delta\Sigma$ and $w_p$ with an enough accuracy for the HSC S16A analysis.

- **fft_num** (*int*): Sampling in fftlog in unit of 2048 (default: 8).
- **fft_logrmin_1h** (*float*): Minimum $\log_{10}(r/[h^{-1}\mathrm{Mpc}])$ used in internal 1-halo term calculation by fftlog (default: -5.0).
- **fft_logrmax_1h** (*float*): Maximum $\log_{10}(r/[h^{-1}\mathrm{Mpc}])$ used in internal 1-halo term calculation by fftlog (default: 3.0).
- **fft_logrmin_2h** (*float*): Minimum $\log_{10}(r/[h^{-1}\mathrm{Mpc}])$ used in internal 2-halo term calculation by fftlog (default: -3.0).
- **fft_logrmax_2h** (*float*): Maximum $\log_{10}(r/[h^{-1}\mathrm{Mpc}])$ used in internal 2-halo term calculation by fftlog (default: 3.0).
- **M_int_logMmin** (*float*): Minimum $\log_{10}(M_{\mathrm{halo}}/[h^{-1}\mathrm{M}_\odot])$ used in the integration across halo mass (default: 12.0).
- **M_int_logMax** (*float*): Maximum $\log_{10}(M_{\mathrm{halo}}/[h^{-1}\mathrm{M}_\odot])$ used in the integration across halo mass (default: 15.9).
- **M_int_k** (*int*): Sampling in the integration across halo mass which sets $2^{\mathrm{M\_int\_k}}$ (default: 5).

- **c-M_relation** (*str*): Concentration-mass relation used for satellite distribution when NFW is used (see `set_galaxy()`; default: 'diemer15'). The concentration is internally computed using colossus, and a user can use a model listed in `concentration models` in this webpage.

**Parameters**
**config** (*dict*) – a dictionary to specify configurations

**get_ds**(*rp*, *redshift*, *dlnrp=0.0*)

Compute weak lensing signal $\Delta\Sigma(r_{\rm p})$.

**Parameters**

- **rp** (*numpy array*) – 2 dimensional projected separation in $h^{-1}{\rm Mpc}$
- **redshift** (*float*) – redshift at which the lens galaxies are located
- **dlnrp** (*float*) – width of bin averaging in logarithmic scale. If dlnrp=0, no bin average.

**Returns**
excess surface density in $hM_{\odot}{\rm pc}^{-2}$

**Return type**
numpy array

**get_ds_cen**(*rp*, *redshift*, *dlnrp=0.0*)

Compute weak lensing signal of (centered) central galaxies $\Delta\Sigma_{\rm cen}(r_{\rm p})$.

**Parameters**

- **rp** (*numpy array*) – 2 dimensional projected separation in $h^{-1}{\rm Mpc}$
- **redshift** (*float*) – redshift at which the lens galaxies are located
- **dlnrp** (*float*) – width of bin averaging in logarithmic scale. If dlnrp=0, no bin average.

**Returns**
excess surface density of (centered) central galaxies in $hM_{\odot}{\rm pc}^{-2}$

**Return type**
numpy array

**get_ds_cen_off**(*rp*, *redshift*, *dlnrp=0.0*)

Compute weak lensing signal of off-centered central galaxies $\Delta\Sigma_{\rm off-cen}(r_{\rm p})$.

**Parameters**

- **rp** (*numpy array*) – 2 dimensional projected separation in $h^{-1}{\rm Mpc}$
- **redshift** (*float*) – redshift at which the lens galaxies are located
- **dlnrp** (*float*) – width of bin averaging in logarithmic scale. If dlnrp=0, no bin average.

**Returns**
excess surface density of off-centered central galaxies in $hM_{\odot}{\rm pc}^{-2}$

**Return type**
numpy array

**get_ds_sat**(*rp*, *redshift*, *dlnrp=0.0*)

Compute weak lensing signal of satellite galaxies $\Delta\Sigma_{\rm sat}(r_{\rm p})$.

**Parameters**

- **rp** (*numpy array*) – 2 dimensional projected separation in $h^{-1}{\rm Mpc}$

- **redshift** (*float*) – redshift at which the lens galaxies are located
- **dlnrp** (*float*) – width of bin averaging in logarithmic scale. If dlnrp=0, no bin average.

**Returns**
excess surface density of satellite galaxies in $hM_{\odot}\mathrm{pc}^{-2}$

**Return type**
numpy array

**get_ng**(*redshift*)

Compute galaxy abundance $n_g$.

**Parameters**
**redshift** (*float*) – redshift at which the galaxies are located

**Returns**
galaxy abundance in $h^3\mathrm{Mpc}^{-3}$

**Return type**
float

**get_ng_cen**(*redshift*)

Compute abundance of central galaxies $n_{g,\mathrm{cen}}$.

**Parameters**
**redshift** (*float*) – redshift at which the central galaxies are located

**Returns**
abundance of central galaxies in $h^3\mathrm{Mpc}^{-3}$

**Return type**
float

**get_wp**(*rp*, *redshift*, *pimax=None*, *rsd=False*, *dlnrp=0.0*)

Compute projected galaxy auto-correlation function $w_{\mathrm{p}}(r_{\mathrm{p}})$.

**Parameters**

- **r_p** (*numpy array*) – 2 dimensional separation in $h^{-1}\mathrm{Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located
- **pi_max** (*float*) – The range of line of sight integral $\pi_{\mathrm{max}}$ in $h^{-1}\mathrm{Mpc}$. If None, the projection is performed using the zeroth order Bessel function, i.e., $\pi_{\mathrm{max}} = \infty$ (default=None).
- **rsd** (*bool*) – if True, redshift space distortion is incorporated into the model (default=False).
- **dlnrp** (*float*) – width of bin averaging in logarithmic scale. If dlnrp=0, no bin average.

**Returns**
projected galaxy auto-correlation function in $h^{-1}\mathrm{Mpc}$

**Return type**
numpy array

**get_wp_1hcs**(*rp*, *redshift*, *dlnrp=0.0*)

Compute projected 1-halo correlation function between central and satellite galaxies $w^{\mathrm{1h}}_{\mathrm{p,cen-sat}}(r_{\mathrm{p}})$. Note that the line-of-sight integration is performed using the zeroth order Bessel function, i.e., , $\pi_{\mathrm{max}} = \infty$.

**Parameters**

- **r_p** (*numpy array*) – 2 dimensional separation in $h^{-1}\mathrm{Mpc}$

- **redshift** (*float*) – redshift at which the galaxies are located
- **dlnrp** (*float*) – width of bin averaging in logarithmic scale. If dlnrp=0, no bin average.

**Returns**
projected 1-halo correlation function between central and satellite galaxies in $h^{-1}\mathrm{Mpc}$

**Return type**
numpy array

**get_wp_1hss**(*rp*, *redshift*, *dlnrp=0.0*)

Compute projected 1-halo correlation function between satellite galaxies $w^{\mathrm{1h}}_{\mathrm{p,sat-sat}}(r_{\mathrm{p}})$. Note that the line-of-sight integration is performed using the zeroth order Bessel function, i.e., , $\pi_{\mathrm{max}} = \infty$.

**Parameters**

- **r_p** (*numpy array*) – 2 dimensional separation in $h^{-1}\mathrm{Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located
- **dlnrp** (*float*) – width of bin averaging in logarithmic scale. If dlnrp=0, no bin average.

**Returns**
projected 1-halo correlation function between satellite galaxies in $h^{-1}\mathrm{Mpc}$

**Return type**
numpy array

**get_wp_2hcc**(*rp*, *redshift*, *dlnrp=0.0*)

Compute projected 2-halo correlation function between central galaxies $w^{\mathrm{2h}}_{\mathrm{p,cen-cen}}(r_{\mathrm{p}})$. Note that the line-of-sight integration is performed using the zeroth order Bessel function, i.e., , $\pi_{\mathrm{max}} = \infty$.

**Parameters**

- **r_p** (*numpy array*) – 2 dimensional separation in $h^{-1}\mathrm{Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located
- **dlnrp** (*float*) – width of bin averaging in logarithmic scale. If dlnrp=0, no bin average.

**Returns**
projected 2-halo correlation function between central galaxies in $h^{-1}\mathrm{Mpc}$

**Return type**
numpy array

**get_wp_2hcs**(*rp*, *redshift*, *dlnrp=0.0*)

Compute projected 2-halo correlation function between central and satellite galaxies $w^{\mathrm{2h}}_{\mathrm{p,cen-sat}}(r_{\mathrm{p}})$. Note that the line-of-sight integration is performed using the zeroth order Bessel function, i.e., , $\pi_{\mathrm{max}} = \infty$.

**Parameters**

- **r_p** (*numpy array*) – 2 dimensional separation in $h^{-1}\mathrm{Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located
- **dlnrp** (*float*) – width of bin averaging in logarithmic scale. If dlnrp=0, no bin average.

**Returns**
projected 2-halo correlation function between central and satellite galaxies in $h^{-1}\mathrm{Mpc}$

**Return type**
numpy array

**get_wp_2hss**(*rp*, *redshift*, *dlnrp=0.0*)

Compute projected 2-halo correlation function between satellite galaxies $w_{\rm p,sat-sat}^{\rm 2h}(r_{\rm p})$. Note that the line-of-sight integration is performed using the zeroth order Bessel function, i.e., , $\pi_{\rm max} = \infty$.

**Parameters**

- **r_p** (*numpy array*) – 2 dimensional separation in $h^{-1}{\rm Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located
- **dlnrp** (*float*) – width of bin averaging in logarithmic scale. If dlnrp=0, no bin average.

**Returns**
projected 2-halo correlation function between satellite galaxies in $h^{-1}{\rm Mpc}$

**Return type**
numpy array

**get_xi_gg**(*r*, *redshift*)

Compute galaxy auto-correlation function $\xi_{\rm gg}(r)$.

**Parameters**

- **r** (*numpy array*) – 3 dimensional separation in $h^{-1}{\rm Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located

**Returns**
galaxy auto-correlation function

**Return type**
numpy array

**get_xi_gg_1hcs**(*r*, *redshift*)

Compute 1-halo correlation function between central and satellite galaxies $\xi_{\rm cen-sat}^{\rm 1h}(r)$.

**Parameters**

- **r** (*numpy array*) – 3 dimensional separation in $h^{-1}{\rm Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located

**Returns**
1-halo correlation function between central and satellite galaxies

**Return type**
numpy array

**get_xi_gg_1hss**(*r*, *redshift*)

Compute 1-halo correlation function between satellite galaxies $\xi_{\rm sat-sat}^{\rm 1h}(r)$.

**Parameters**

- **r** (*numpy array*) – 3 dimensional separation in $h^{-1}{\rm Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located

**Returns**
1-halo correlation function between satellite galaxies

**Return type**
numpy array

`get_xi_gg_2hcc`(*rp*, *redshift*)

Compute 2-halo correlation function between central galaxies $\xi^{\rm 2h}_{\rm cen-cen}(r)$.

**Parameters**

- **r** (*numpy array*) – 3 dimensional separation in $h^{-1}{\rm Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located

**Returns**
2-halo correlation function between central galaxies

**Return type**
numpy array

`get_xi_gg_2hcs`(*rp*, *redshift*)

Compute 2-halo correlation function between central and satellite galaxies $\xi^{\rm 2h}_{\rm cen-sat}(r)$.

**Parameters**

- **r** (*numpy array*) – 3 dimensional separation in $h^{-1}{\rm Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located

**Returns**
2-halo correlation function between central and satellite galaxies

**Return type**
numpy array

`get_xi_gg_2hss`(*rp*, *redshift*)

Compute 2-halo correlation function between satellite galaxies $\xi^{\rm 2h}_{\rm sat-sat}(r)$.

**Parameters**

- **r** (*numpy array*) – 3 dimensional separation in $h^{-1}{\rm Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located

**Returns**
2-halo correlation function between satellite galaxies

**Return type**
numpy array

`get_xi_gm`(*r*, *redshift*)

Compute correlation function between galaxies and dark matter $\xi_{\rm gm}(r)$.

**Parameters**

- **r** (*numpy array*) – 3 dimensional separation in $h^{-1}{\rm Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located

**Returns**
correlation function between galaxies and dark matter

**Return type**
numpy array

`get_xi_gm_cen`(*r*, *redshift*)

Compute correlation function between (centered) central galaxies and dark matter $\xi_{\rm gm,cen}(r)$.

**Parameters**

- **r** (*numpy array*) – 3 dimensional separation in $h^{-1}\mathrm{Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located

**Returns**
correlation function between (centered) central galaxies and dark matter

**Return type**
numpy array

**get_xi_gm_cen_off**(*r*, *redshift*)

Compute correlation function between off-centered central galaxies and dark matter $\xi_{\mathrm{gm,off-cen}}(r)$.

**Parameters**

- **r** (*numpy array*) – 3 dimensional separation in $h^{-1}\mathrm{Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located

**Returns**
correlation function between off-centered central galaxies and dark matter

**Return type**
numpy array

**get_xi_gm_sat**(*r*, *redshift*)

Compute correlation function between satellite galaxies and dark matter $\xi_{\mathrm{gm,sat}}(r)$.

**Parameters**

- **r** (*numpy array*) – 3 dimensional separation in $h^{-1}\mathrm{Mpc}$
- **redshift** (*float*) – redshift at which the galaxies are located

**Returns**
correlation function between satellite galaxies and dark matter

**Return type**
numpy array

**set_cosmology**(*cparams*)

Let the emulator know the cosmological parameters. This interface passes the 6 parameters to all the class objects used for the emulation of various halo statistics.

The current version supports wCDM cosmologies specified by the 6 parameters as described below. Other parameters are automatically computed:

$\Omega_m = 1 - \Omega_{de}$,

$h = \sqrt{(\omega_b + \omega_c + \omega_\nu)/\Omega_m}$,

where the neutrino density is fixed by $\omega_\nu = 0.00064$ corresponding to the mass sum of 0.06 eV.

**Parameters**
**cparam** (*numpy array*) – Cosmological parameters $(\omega_b, \omega_c, \Omega_{de}, \ln(10^{10}A_s), n_s, w)$

**set_galaxy**(*gparams*)

This method sets galaxy parameter through a dictionary. See Miyatake et al (2021) for the definition of galaxy parameters. Here is the list of keys.

- HOD parameters:
  - **logMmin** (*float*): Central HOD parameter, $\log M_{\mathrm{min}}$
  - **sigma_sq** (*float*): Central HOD parameter, $\sigma^2$

- **logM1** (*float*): Satellite HOD parameter, $\log M_1$
- **alpha** (*float*): Satellite HOD parameter, $\alpha$
- **kappa** (*float*): Satellite HOD parameter, $\kappa$

- off-centering parameters:
  - **poff** (*float*): Fraction of off-centered galaxies, $p_{\rm off}$
  - **Roff** (*float*): Characteristic scale of off-centered galaxies with respect to $R_{\rm 200m}$, $R_{\rm off}$
- satellite distribution
  - **sat_dist_type** (*float*): Profile of satellite distribution. Valid values are 'emulator' or 'NFW'. When 'NFW', concentration is specified in **config** parameter (see `dark_emulator.model_hod.hod_interface.darkemu_x_hod`)
- incompleteness parameters
  - **alpha_inc** (*float*): Incompleteness parameter, $\alpha_{\rm inc}$
  - **logM_inc** (*float*): Incompleteness parameter, $\log M_{\rm inc}$

**Parameters**
**gparams** (*dict*) – a dictionary to specify galaxy parameters

### Module contents

### dark_emulator.pyfftlog_interface package

### Submodules

### dark_emulator.pyfftlog_interface.pyfftlog_class module

### dark_emulator.pyfftlog_interface.pyfftlog_funcs module

### Module contents

## 1.1.2 Module contents

CHAPTER

# TWO

# DARKEMU MODULE TUTORIAL NOTEBOOK

```
%load_ext autoreload
%autoreload 2
%pylab inline
import logging
mpl_logger = logging.getLogger('matplotlib')
mpl_logger.setLevel(logging.WARNING)
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.size'] = 18
plt.rcParams['axes.linewidth'] = 1.5
plt.rcParams['xtick.major.size'] = 5
plt.rcParams['ytick.major.size'] = 5
plt.rcParams['xtick.minor.size'] = 3
plt.rcParams['ytick.minor.size'] = 3
plt.rcParams['xtick.top'] = True
plt.rcParams['ytick.right'] = True
plt.rcParams['xtick.minor.visible'] = True
plt.rcParams['ytick.minor.visible'] = True
plt.rcParams['xtick.direction'] = 'in'
plt.rcParams['ytick.direction'] = 'in'
plt.rcParams['figure.figsize'] = (10,6)
```

```
from dark_emulator import darkemu
```

```
emu = darkemu.base_class()
```

```
initialize cosmo_class
Initialize pklin emulator
initialize propagator emulator
Initialize sigma_d emulator
initialize cross-correlation emulator
initialize auto-correlation emulator
Initialize sigmaM emulator
initialize xinl emulator
```

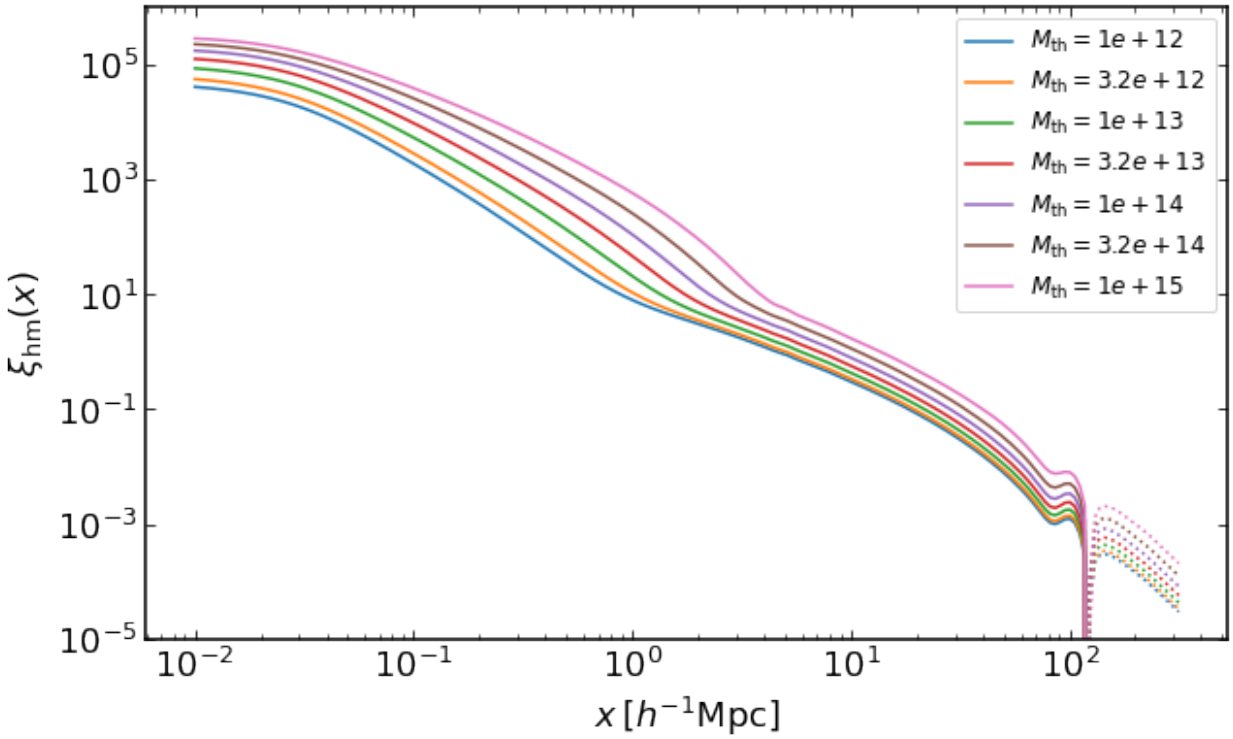
## 2.1 how to set cosmology

```
cparam = np.array([0.02225,0.1198,0.6844,3.094,0.9645,-1.])
emu.set_cosmology(cparam)
```

## 2.2 how to plot halo-mass cross correlation for mass threshold halo samples

```
rs = np.logspace(-2,2.5,200)

plt.figure(figsize=(10,6))

z = 0

for i, Mmin in enumerate(np.logspace(12,15,7)):
    xihm = emu.get_xicross_massthreshold(rs,Mmin,z)
    plt.loglog(rs,xihm,color="C{}".format(i),label='$M_\mathrm{th}=%0.2g$' %Mmin)
    plt.loglog(rs,-xihm,':',color="C{}".format(i))
plt.legend(fontsize=12)
plt.ylim(0.00001,1000000)
plt.xlabel("$x\,[h^{-1}\mathrm{Mpc}]$")
plt.ylabel("$\\xi_\mathrm{hm}(x)$")
```

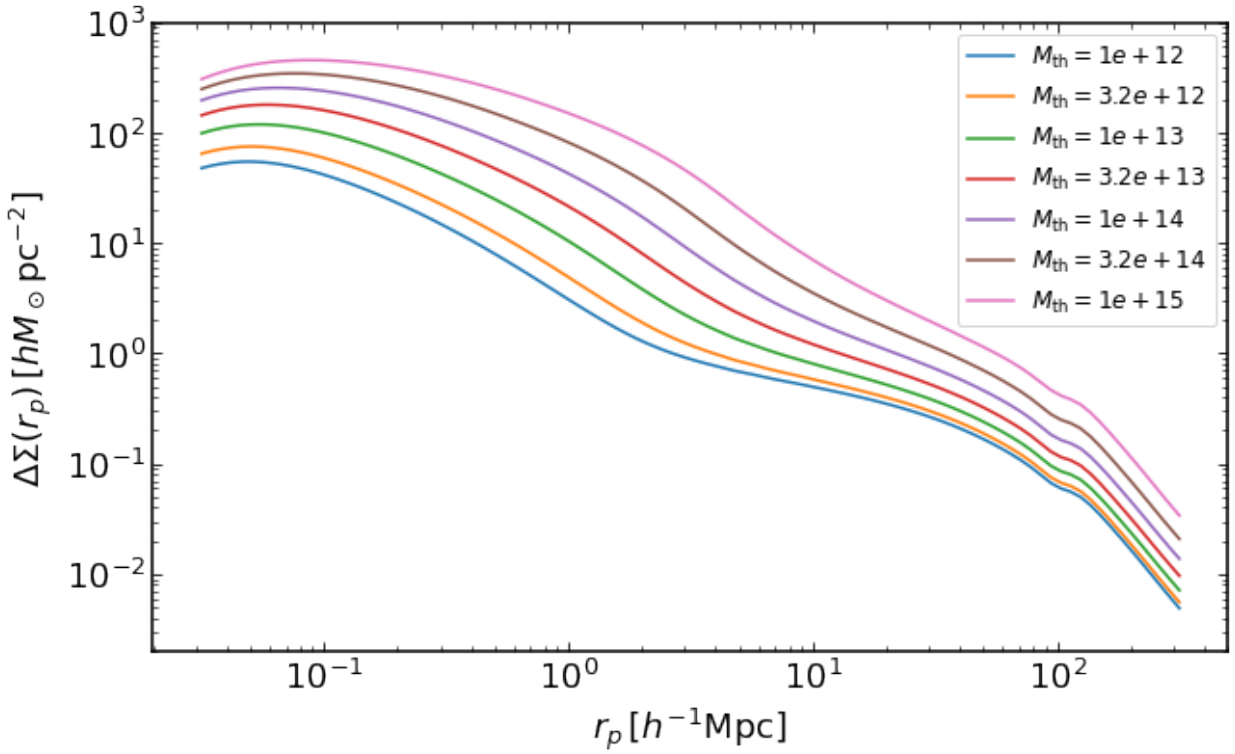
```
Text(0, 0.5, '$\xi_\mathrm{hm}(x)$')
```



## 2.3 how to plot DeltaSigma(R) for a mass threshold halo samples

```
rs = np.logspace(-1.5,2.5,100)

plt.figure(figsize=(10,6))

z = 0

for i, Mmin in enumerate(np.logspace(12,15,7)):
    dsigma = emu.get_DeltaSigma_massthreshold(rs,Mmin,z)
    plt.loglog(rs,dsigma,label='$M_\mathrm{th}=%0.2g$' %Mmin)
plt.legend(fontsize=12)
plt.ylim(0.002,1000)
plt.xlabel("$r_p\,[h^{-1}\mathrm{Mpc}]$")
plt.ylabel("$\Delta\Sigma(r_p)\,[h M_\odot \mathrm{pc}^{-2}]$")
```

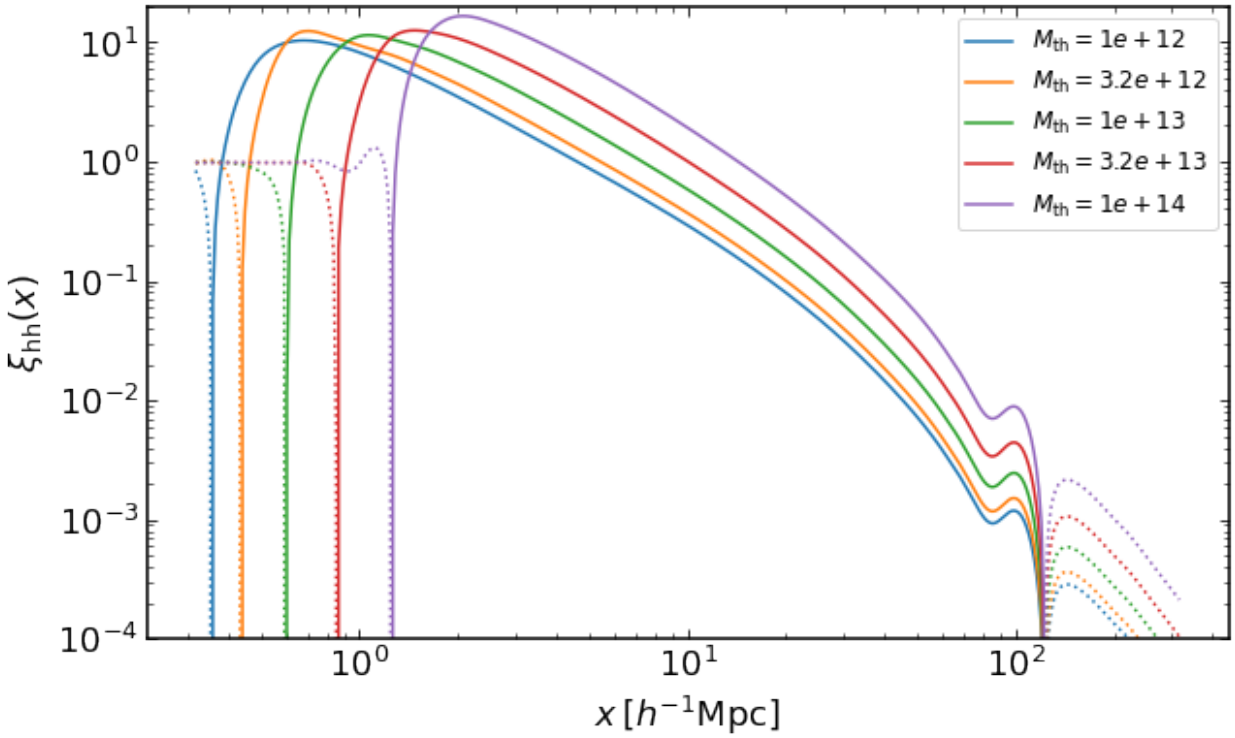
```
Text(0, 0.5, '$\Delta\Sigma(r_p)\,[h M_\odot \mathrm{pc}^{-2}]$')
```



## 2.4 how to plot halo-halo correlation for mass threshold halo samples

```
rs = np.logspace(-0.5,2.5,400)

plt.figure(figsize=(10,6))

z = 0

for i, Mmin in enumerate(np.logspace(12,14,5)):
    xih = emu.get_xiauto_massthreshold(rs,Mmin,z)
    plt.loglog(rs,xih,color="C{}".format(i),label='$M_\mathrm{th}=%0.2g$' %Mmin)
    plt.loglog(rs,-xih,':',color="C{}".format(i))
plt.legend(fontsize=12)
plt.ylim(0.0001,20)
plt.xlabel("$x\,[h^{-1}\mathrm{Mpc}]$")
plt.ylabel("$\\xi_\mathrm{hh}(x)$")
```

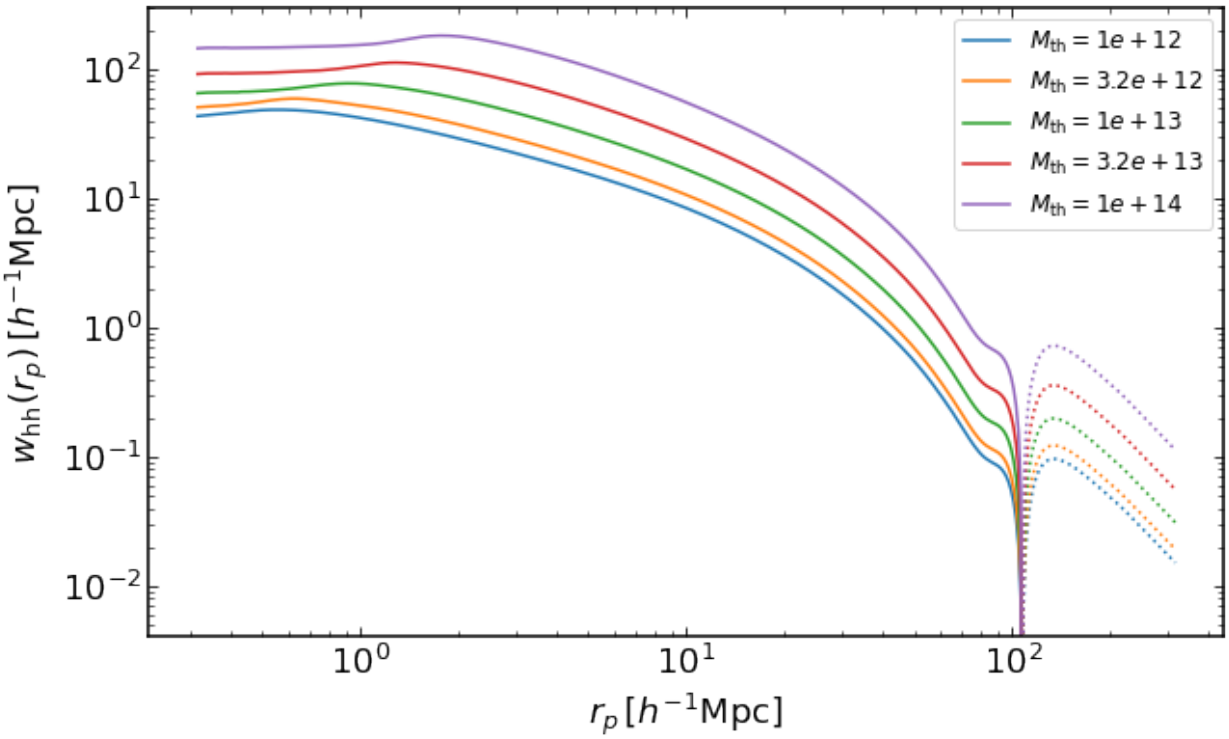
```
Text(0, 0.5, '$\xi_\mathrm{hh}(x)$')
```



## 2.5 how to plot halo-halo projected correlation function for mass threshold halo samples

```
rs = np.logspace(-0.5,2.5,400)

z = 0

plt.figure(figsize=(10,6))

for i, Mmin in enumerate(np.logspace(12,14,5)):
    wh = emu.get_wauto_massthreshold(rs,Mmin,z)
    plt.loglog(rs,wh,color="C{}".format(i),label='$M_\mathrm{th}=%0.2g$' %Mmin)
    plt.loglog(rs,-wh,':',color="C{}".format(i))
plt.legend(fontsize=12)
plt.xlabel("$r_p\,[h^{-1}\mathrm{Mpc}]$")
plt.ylabel("$w_\mathrm{hh}(r_p)\,[h^{-1}\mathrm{Mpc}]$")
```

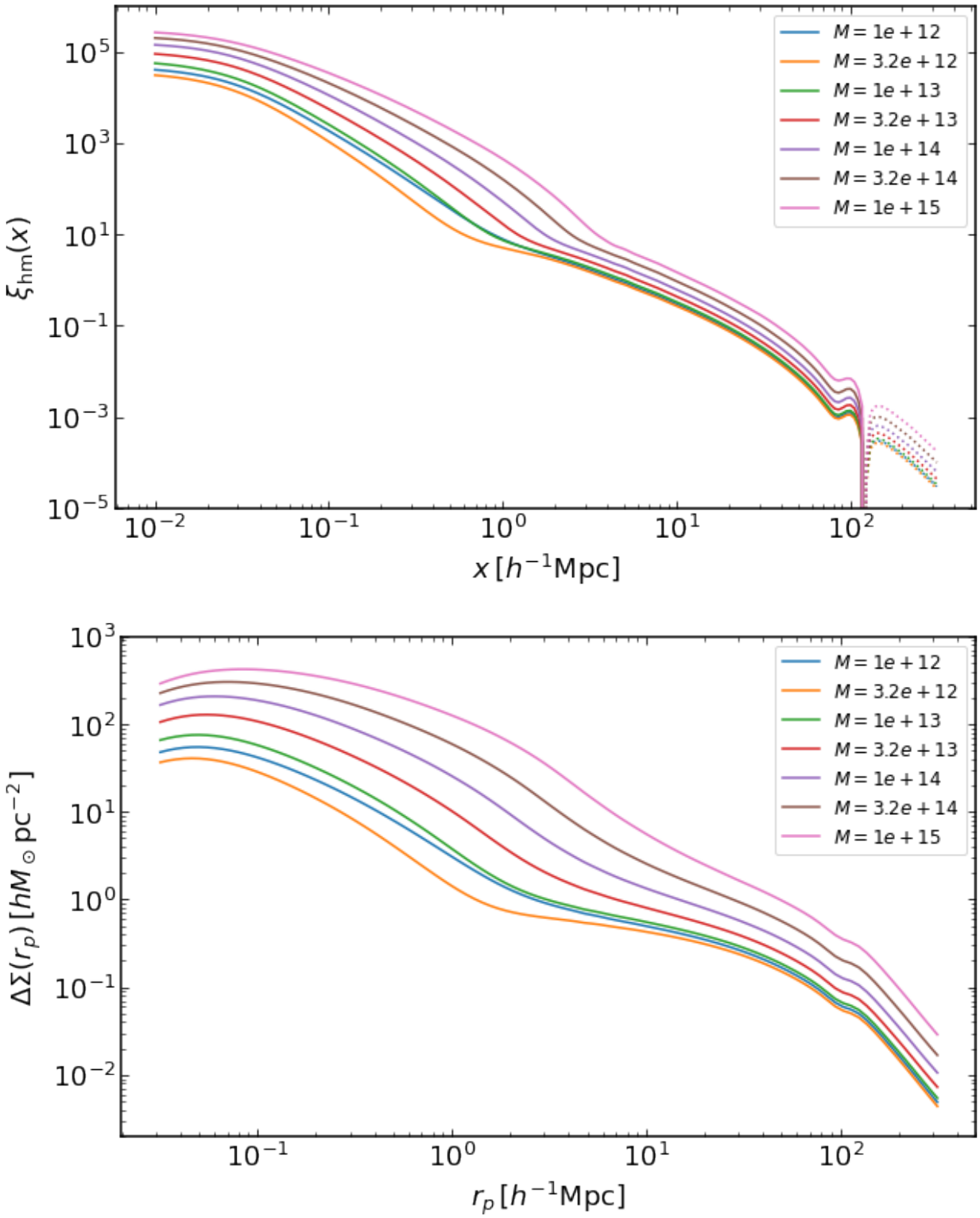
```
Text(0, 0.5, '$w_\mathrm{hh}(r_p)\,[h^{-1}\mathrm{Mpc}]$')
```

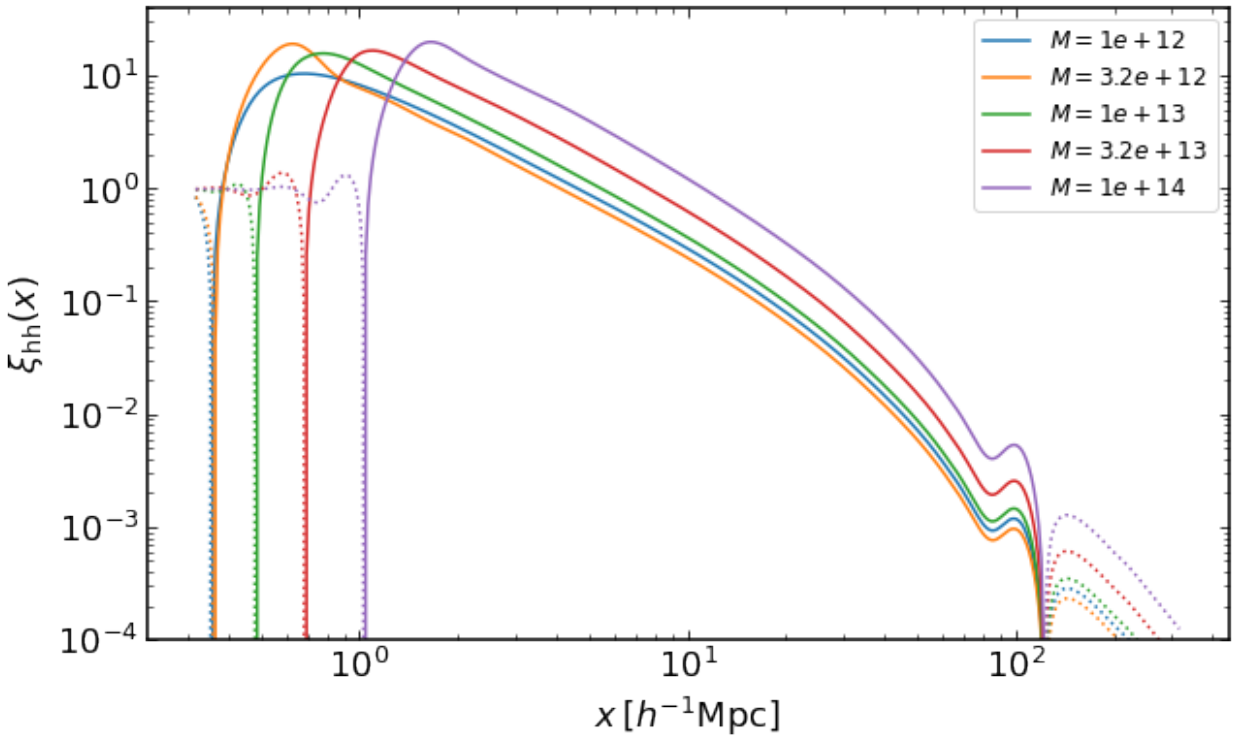


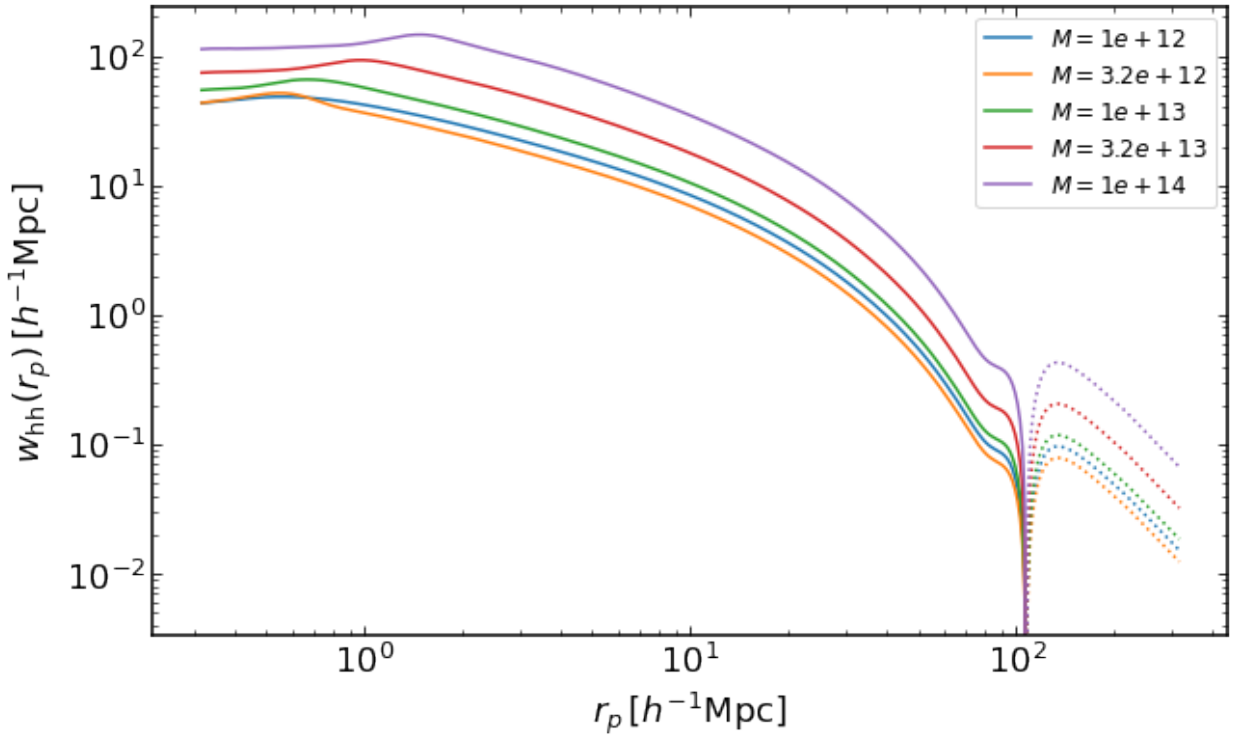
## 2.6 Same as before, but for halos with fixed masses instead of mass threshold samples.

```
rs = np.logspace(-2,2.5,200)
plt.figure(figsize=(10,6))
for i, M in enumerate(np.logspace(12,15,7)):
    xihm = emu.get_xicross_mass(rs,M,z)
    plt.loglog(rs,xihm,color="C{}".format(i),label='$M=%0.2g$' %M)
    plt.loglog(rs,-xihm,':',color="C{}".format(i))
plt.legend(fontsize=12)
plt.ylim(0.00001,1000000)
plt.xlabel("$x\,[h^{-1}\mathrm{Mpc}]$")
plt.ylabel("$\\xi_\mathrm{hm}(x)$")

rs = np.logspace(-1.5,2.5,100)
plt.figure(figsize=(10,6))
for i, M in enumerate(np.logspace(12,15,7)):
    dsigma = emu.get_DeltaSigma_mass(rs,M,z)
    plt.loglog(rs,dsigma,label='$M=%0.2g$' %M)
plt.legend(fontsize=12)
plt.ylim(0.002,1000)
plt.xlabel("$r_p\,[h^{-1}\mathrm{Mpc}]$")
plt.ylabel("$\Delta\Sigma(r_p)\,[h M_\odot \mathrm{pc}^{-2}]$")

rs = np.logspace(-0.5,2.5,400)
plt.figure(figsize=(10,6))
for i, M in enumerate(np.logspace(12,14,5)):
    xih = emu.get_xiauto_mass(rs,M,M,z)
    plt.loglog(rs,xih,color="C{}".format(i),label='$M=%0.2g$' %M)
    plt.loglog(rs,-xih,':',color="C{}".format(i))
plt.legend(fontsize=12)
plt.ylim(0.0001,40)
plt.xlabel("$x\,[h^{-1}\mathrm{Mpc}]$")
plt.ylabel("$\\xi_\mathrm{hh}(x)$")

rs = np.logspace(-0.5,2.5,400)
plt.figure(figsize=(10,6))
for i, M in enumerate(np.logspace(12,14,5)):
    wh = emu.get_wauto_mass(rs,M,M,z)
    plt.loglog(rs,wh,color="C{}".format(i),label='$M=%0.2g$' %M)
    plt.loglog(rs,-wh,':',color="C{}".format(i))
plt.legend(fontsize=12)
plt.xlabel("$r_p\,[h^{-1}\mathrm{Mpc}]$")
plt.ylabel("$w_\mathrm{hh}(r_p)\,[h^{-1}\mathrm{Mpc}]$")
```

```
Text(0, 0.5, '$w_\mathrm{hh}(r_p)\,[h^{-1}\mathrm{Mpc}]$')
```


$M = 1e+12$
$M = 3.2e+12$
$M = 1e+13$
$M = 3.2e+13$
$M = 1e+14$
$M = 3.2e+14$
$M = 1e+15$
$\xi_{\rm hm}(x)$
$x\,[h^{-1}{\rm Mpc}]$
$\Delta\Sigma(r_p)\,[hM_\odot\,{\rm pc}^{-2}]$
$r_p\,[h^{-1}{\rm Mpc}]$

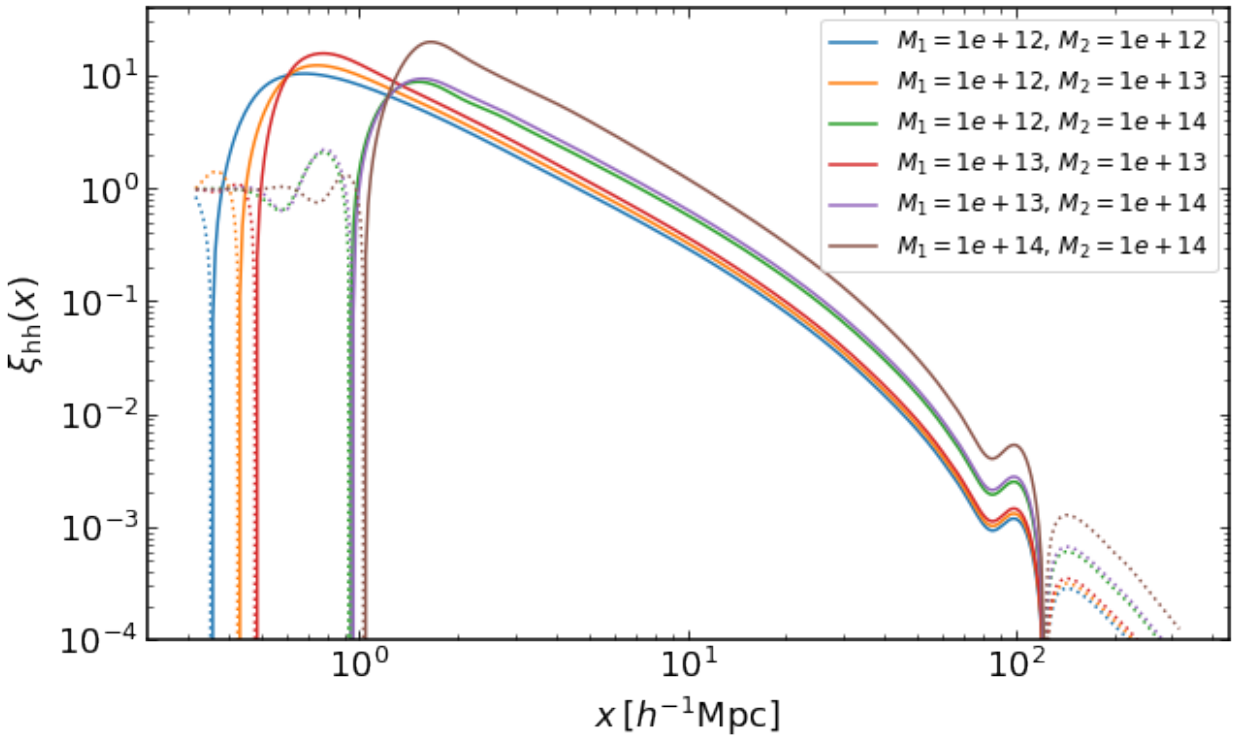

$M = 1e+12$
$M = 3.2e+12$
$M = 1e+13$
$M = 3.2e+13$
$M = 1e+14$
$\xi_{\rm hh}(x)$
$x\,[h^{-1}{\rm Mpc}]$

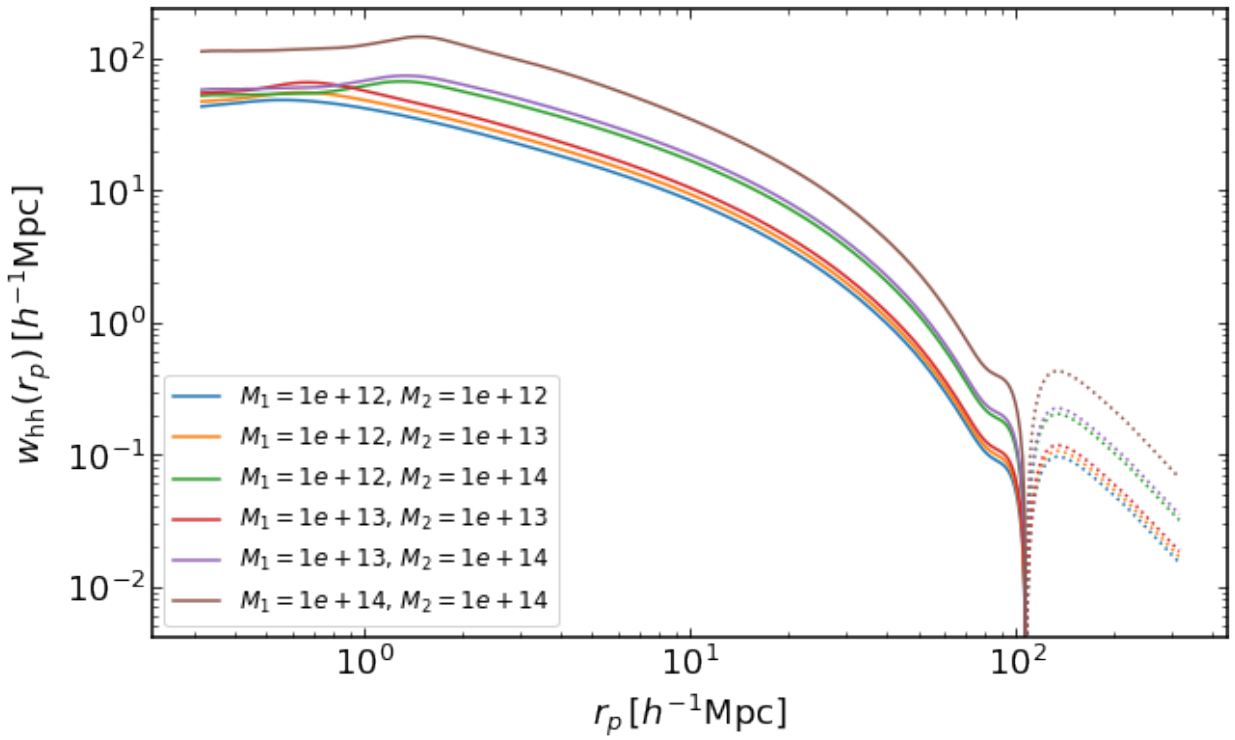

$M = 1e+12$
$M = 3.2e+12$
$M = 1e+13$
$M = 3.2e+13$
$M = 1e+14$
$w_{\rm hh}(r_p)\,[h^{-1}{\rm Mpc}]$
$r_p\,[h^{-1}{\rm Mpc}]$

## 2.7 Halo-halo correlation function for halos with 2 different masses

```
rs = np.logspace(-0.5,2.5,400)
Ms = np.logspace(12,14,3)
plt.figure(figsize=(10,6))
ii = 0
for i in range(3):
    for j in range(i,3):
        xih = emu.get_xiauto_mass(rs,Ms[i],Ms[j],z)
        plt.loglog(rs,xih,color="C{}".format(ii),label='$M_1=%0.2g,\,M_2=%0.2g$' %(Ms[i],
↪Ms[j]))
        plt.loglog(rs,-xih,':',color="C{}".format(ii))
        ii+=1
plt.legend(fontsize=12)
plt.ylim(0.0001,40)
plt.xlabel("$x\,[h^{-1}\mathrm{Mpc}]$")
plt.ylabel("$\\xi_\mathrm{hh}(x)$")

rs = np.logspace(-0.5,2.5,400)
plt.figure(figsize=(10,6))
ii = 0
for i in range(3):
    for j in range(i,3):
        wh = emu.get_wauto_mass(rs,Ms[i],Ms[j],z)
        plt.loglog(rs,wh,color="C{}".format(ii),label='$M_1=%0.2g,\,M_2=%0.2g$' %(Ms[i],
↪Ms[j]))
        plt.loglog(rs,-wh,':',color="C{}".format(ii))
        ii+=1
plt.legend(fontsize=12)
plt.xlabel("$r_p\,[h^{-1}\mathrm{Mpc}]$")
plt.ylabel("$w_\mathrm{hh}(r_p)\,[h^{-1}\mathrm{Mpc}]$")
```

```
Text(0, 0.5, '$w_\mathrm{hh}(r_p)\,[h^{-1}\mathrm{Mpc}]$')
```


$M_1 = 1e+12, M_2 = 1e+12$
$M_1 = 1e+12, M_2 = 1e+13$
$M_1 = 1e+12, M_2 = 1e+14$
$M_1 = 1e+13, M_2 = 1e+13$
$M_1 = 1e+13, M_2 = 1e+14$
$M_1 = 1e+14, M_2 = 1e+14$
$\xi_{\rm hh}(x)$
$x\,[h^{-1}{\rm Mpc}]$

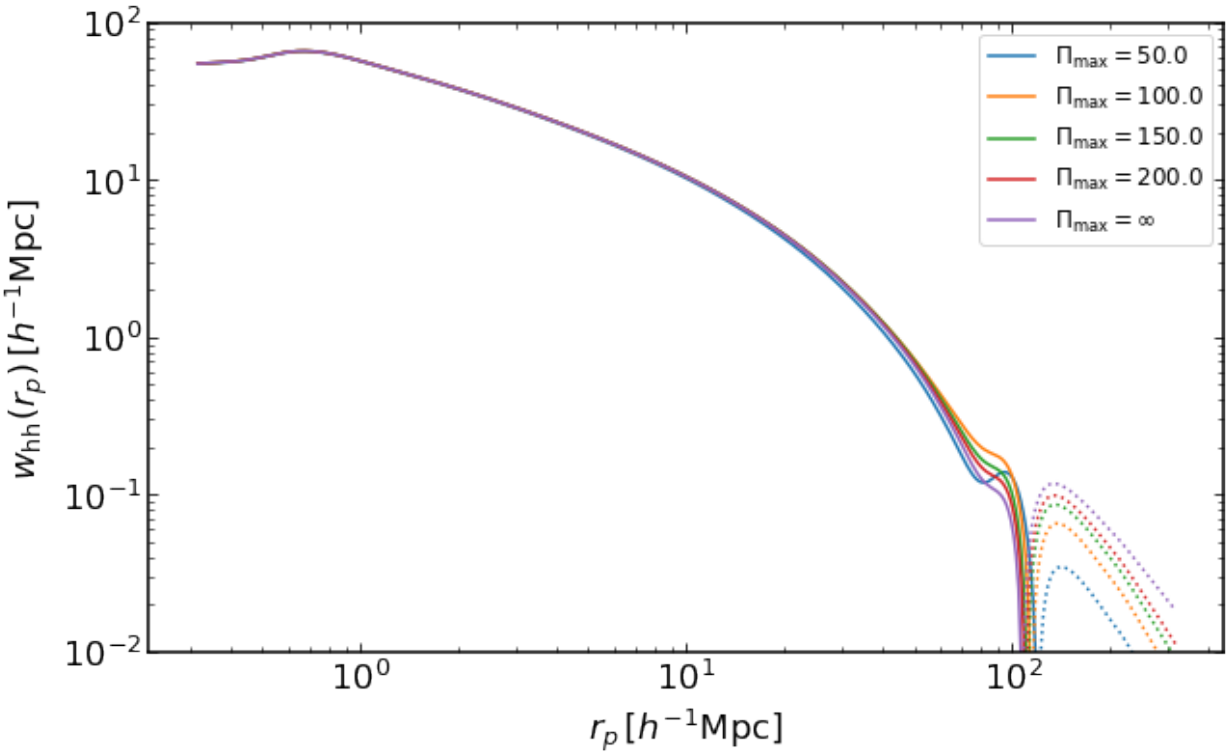

$M_1 = 1e+12, M_2 = 1e+12$
$M_1 = 1e+12, M_2 = 1e+13$
$M_1 = 1e+12, M_2 = 1e+14$
$M_1 = 1e+13, M_2 = 1e+13$
$M_1 = 1e+13, M_2 = 1e+14$
$M_1 = 1e+14, M_2 = 1e+14$
$w_{\rm hh}(r_p)\,[h^{-1}{\rm Mpc}]$
$r_p\,[h^{-1}{\rm Mpc}]$

## 2.8 Projected Halo-halo correlation function with finite projection widths

This takes more time because of an additional direct integration, which is bypassed by using pyfftlog in other routines.

```
plt.figure(figsize=(10,6))
ii = 0
M = 1e13

for i, pimax in enumerate(np.linspace(50,200,4)):
    wh = emu.get_wauto_mass_cut(rs,M,M,z,pimax)
    plt.loglog(rs,wh,color="C{}".format(i),label='$\Pi_\mathrm{max}=%.1f$' %(pimax))
    plt.loglog(rs,-wh,':',color="C{}".format(i))

wh = emu.get_wauto_mass(rs,M,M,z)
plt.loglog(rs,wh,color="C{}".format(4),label='$\Pi_\mathrm{max}=\infty$')
plt.loglog(rs,-wh,':',color="C{}".format(4))

plt.legend(fontsize=12)
plt.xlabel("$r_p\,[h^{-1}\mathrm{Mpc}]$")
plt.ylabel("$w_\mathrm{hh}(r_p)\,[h^{-1}\mathrm{Mpc}]$")
plt.ylim(0.01,100)
```

```
(0.01, 100)
```



CHAPTER

# THREE

# MODEL_HOD MODULE TUTORIAL NOTEBOOK

```
%load_ext autoreload
%autoreload 2
%pylab inline
import logging
mpl_logger = logging.getLogger('matplotlib')
mpl_logger.setLevel(logging.WARNING)
pil_logger = logging.getLogger('PIL')
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.size'] = 18
plt.rcParams['axes.linewidth'] = 1.5
plt.rcParams['xtick.major.size'] = 5
plt.rcParams['ytick.major.size'] = 5
plt.rcParams['xtick.minor.size'] = 3
plt.rcParams['ytick.minor.size'] = 3
plt.rcParams['xtick.top'] = True
plt.rcParams['ytick.right'] = True
plt.rcParams['xtick.minor.visible'] = True
plt.rcParams['ytick.minor.visible'] = True
plt.rcParams['xtick.direction'] = 'in'
plt.rcParams['ytick.direction'] = 'in'
plt.rcParams['figure.figsize'] = (10,6)
```

```
from dark_emulator import model_hod
```

```
hod = model_hod.darkemu_x_hod({"fft_num":8})
```

```
initialize cosmo_class
initialize xinl emulator
Initialize pklin emulator
initialize propagator emulator
Initialize sigma_d emulator
initialize cross-correlation emulator
initialize auto-correlation emulator
Initialize sigmaM emulator
```

## 3.1 how to set cosmology and galaxy parameters (HOD, off-centering, satellite distribution, and incompleteness)

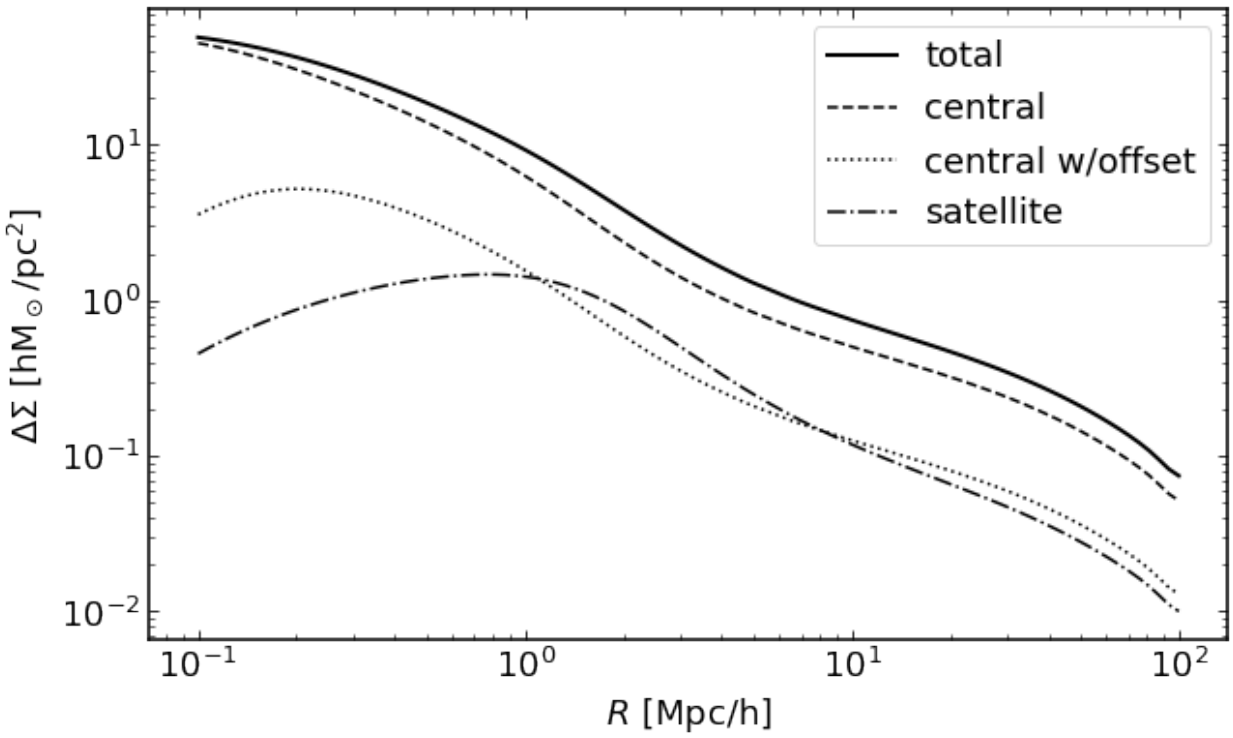
```
cparam = np.array([0.02225,0.1198,0.6844,3.094,0.9645,-1.])
hod.set_cosmology(cparam)

gparam = {"logMmin":13.13, "sigma_sq":0.22, "logM1": 14.21, "alpha": 1.13, "kappa": 1.25,
↪ # HOD parameters
          "poff": 0.2, "Roff": 0.1, # off-centering parameters p_off is the fraction of␣
↪off-centered galaxies. Roff is the typical off-centered scale with respect to R200m.
          "sat_dist_type": "emulator", # satellite distribution. Chosse emulator of NFW.
↪ In the case of NFW, the c-M relation by Diemer & Kravtsov (2015) is assumed.
          "alpha_inc": 0.44, "logM_inc": 13.57} # incompleteness parameters. For␣
↪details, see More et al. (2015)
hod.set_galaxy(gparam)
```

```
INFO:root:Got same cosmology. Keep quantities already computed.
```

## 3.2 how to plot g-g lensing signal in DeltaSigma(R)

```
redshift = 0.55
r = np.logspace(-1,2,100)

plt.figure(figsize=(10,6))

plt.loglog(r, hod.get_ds(r, redshift), linewidth = 2, color = "k", label = "total")
plt.loglog(r, hod.get_ds_cen(r, redshift), "--", color = "k", label = "central")
plt.loglog(r, hod.get_ds_cen_off(r, redshift), ":", color = "k", label = "central w/
↪offset")
plt.loglog(r, hod.get_ds_sat(r, redshift), "-.", color = "k", label = "satellite")

plt.xlabel(r"$R$ [Mpc/h]")
plt.ylabel(r"$\Delta\Sigma$ [hM$_\odot$/pc$^2$]")
plt.legend()
```

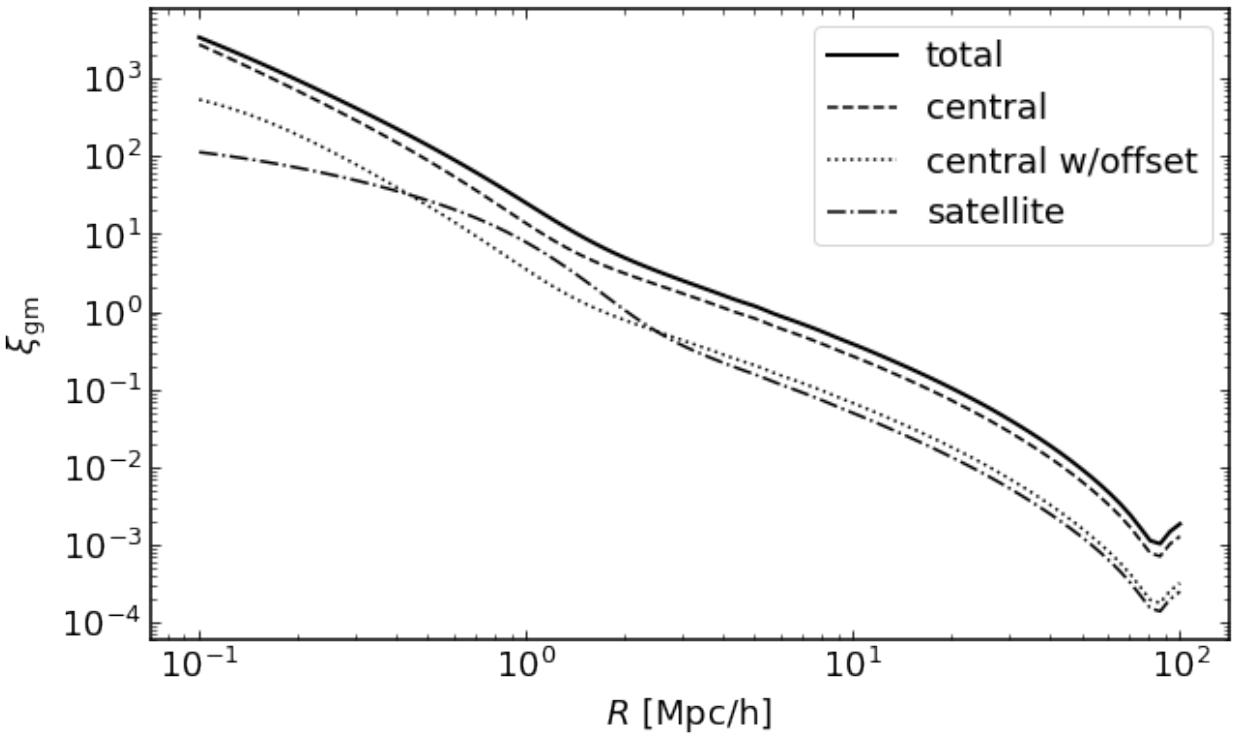
```
<matplotlib.legend.Legend at 0x7f8d07656b70>
```




## 3.3 how to plot g-g lensing signal in xi

```
redshift = 0.55
r = np.logspace(-1,2,100)

plt.figure(figsize=(10,6))

plt.loglog(r, hod.get_xi_gm(r, redshift), linewidth = 2, color = "k", label = "total")
plt.loglog(r, hod.get_xi_gm_cen(r, redshift), "--", color = "k", label = "central")
plt.loglog(r, hod.get_xi_gm_cen_off(r, redshift), ":", color = "k", label = "central w/
↪offset")
plt.loglog(r, hod.get_xi_gm_sat(r, redshift), "-.", color = "k", label = "satellite")

plt.xlabel(r"$R$ [Mpc/h]")
plt.ylabel(r"$\xi_{\rm gm}$")
plt.legend()
```

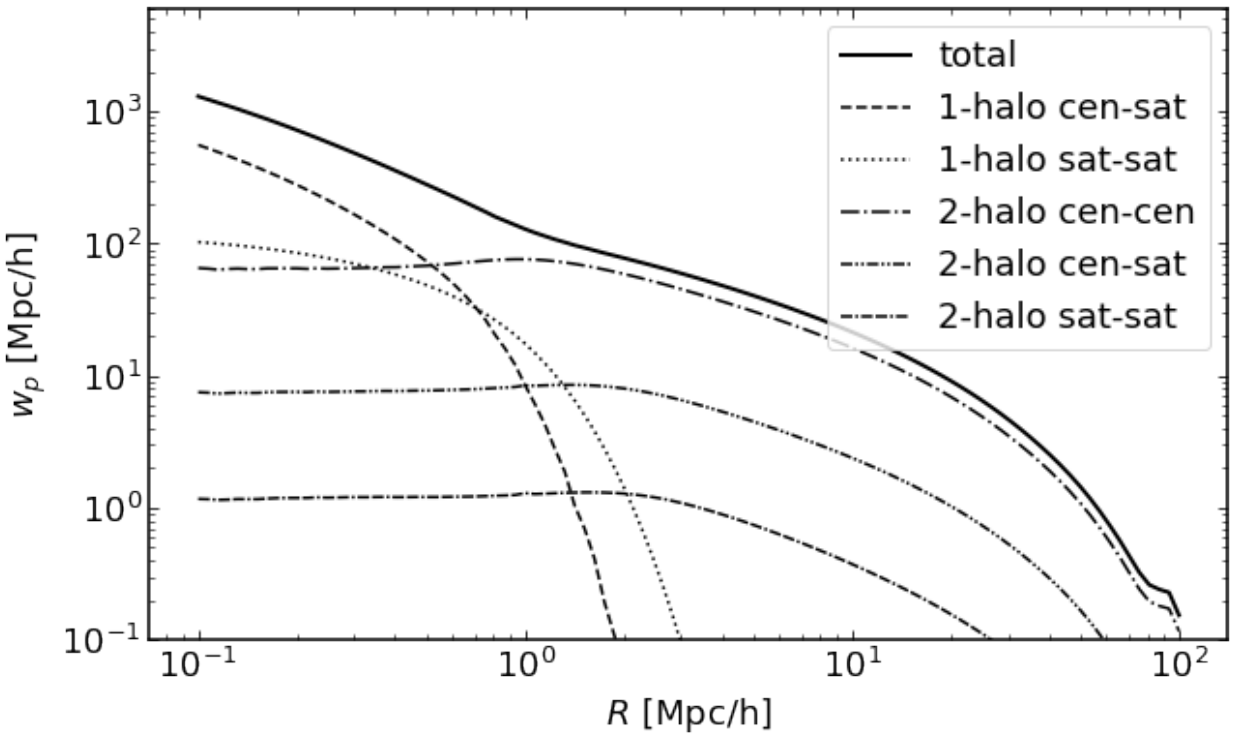
```
<matplotlib.legend.Legend at 0x7f8d095bac50>
```




## 3.4 how to plot g-g clustering signal in wp

```
redshift = 0.55
rs = np.logspace(-1,2,100)

plt.figure(figsize=(10,6))

plt.loglog(r, hod.get_wp(r, redshift), linewidth = 2, color = "k", label = "total")
plt.loglog(r, hod.get_wp_1hcs(r, redshift), "--", color = "k", label = "1-halo cen-sat")
plt.loglog(r, hod.get_wp_1hss(r, redshift), ":", color = "k", label = "1-halo sat-sat")
plt.loglog(r, hod.get_wp_2hcc(r, redshift), "-.", color = "k", label = "2-halo cen-cen")
plt.loglog(r, hod.get_wp_2hcs(r, redshift), dashes=[4,1,1,1,1,1], color = "k", label =
↪"2-halo cen-sat")
plt.loglog(r, hod.get_wp_2hss(r, redshift), dashes=[4,1,1,1,4,1], color = "k", label =
↪"2-halo sat-sat")

plt.xlabel(r"$R$ [Mpc/h]")
plt.ylabel(r"$w_p$ [Mpc/h]")
plt.legend()
plt.ylim(0.1, 6e3)
```

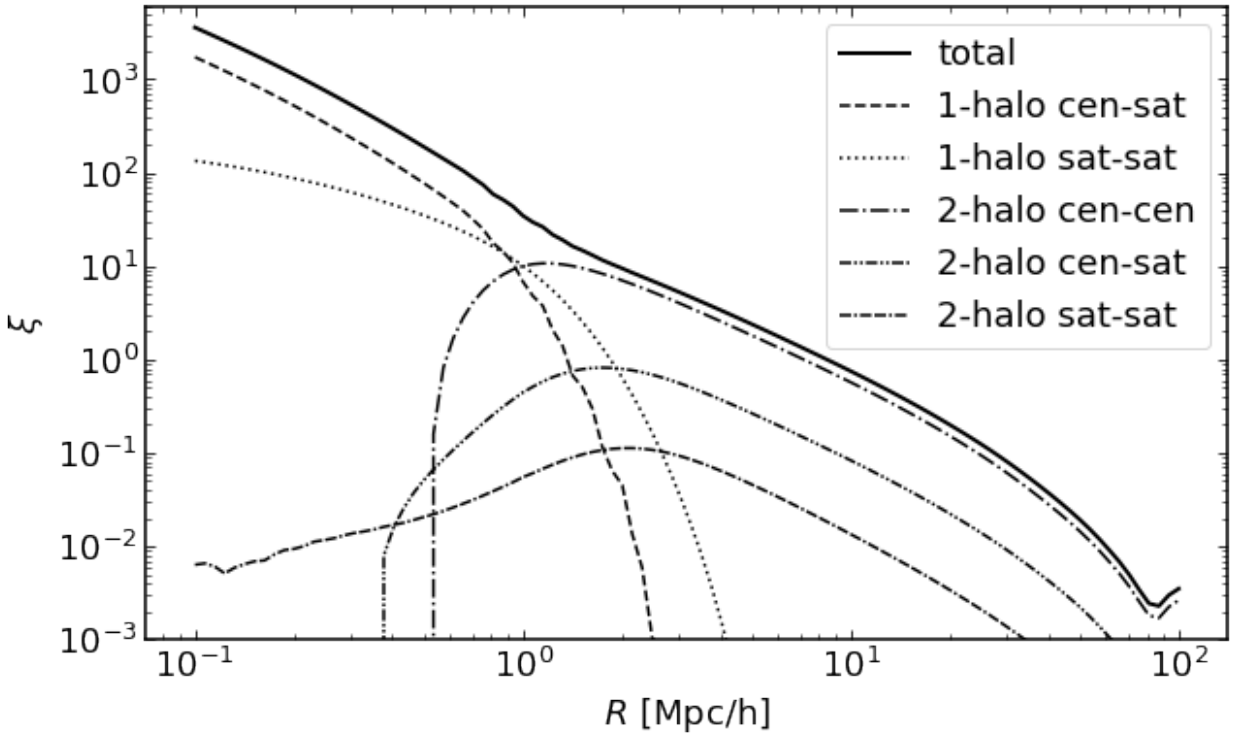
```
(0.1, 6000.0)
```




## 3.5 how to plot g-g clustering signal in xi

```
redshift = 0.55
rs = np.logspace(-1,2,100)

plt.figure(figsize=(10,6))

plt.loglog(r, hod.get_xi_gg(r, redshift), linewidth = 2, color = "k", label = "total")
plt.loglog(r, hod.get_xi_gg_1hcs(r, redshift), "--", color = "k", label = "1-halo cen-sat
↪")
plt.loglog(r, hod.get_xi_gg_1hss(r, redshift), ":", color = "k", label = "1-halo sat-sat
↪")
plt.loglog(r, hod.get_xi_gg_2hcc(r, redshift), "-.", color = "k", label = "2-halo cen-cen
↪")
plt.loglog(r, hod.get_xi_gg_2hcs(r, redshift), dashes=[4,1,1,1,1,1], color = "k", label␣
↪= "2-halo cen-sat")
plt.loglog(r, hod.get_xi_gg_2hss(r, redshift), dashes=[4,1,1,1,4,1], color = "k", label␣
↪= "2-halo sat-sat")

plt.xlabel(r"$R$ [Mpc/h]")
plt.ylabel(r"$\xi$")
plt.legend()
plt.ylim(1e-3, 6e3)
```

```
(0.001, 6000.0)
```




The dark_emulator package is maintained on github (https://github.com/DarkQuestCosmology/dark_emulator_public).

CHAPTER

FOUR

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## d

# INDEX

## M



## P

## S



## X