EffectiveHalos Documentation

Oliver Philcox

Jul 05, 2020

Contents:

EffectiveHalos is a fast Python code providing models of the real-space matter power spectrum, based a combination of the Halo Model and Effective Field Theory, which are 1% accurate up to $k = 1h\,\mathrm{Mpc}^{-1}$, across a range of cosmologies including those with massive neutrinos. It can additionally compute accurate halo count covariances (including a model of halo exclusion), both and alone and with the matter power spectrum, featuring a model for halo exclusion.

This is based on the work of [Philcox, Spergel & Villaescusa-Navarro 2020](https://arxiv.org/abs/2004.09515), and makes use of the CLASS and FAST-PT codes for computing linear and one-loop power spectra respectively.

Authors:

- Oliver Philcox (Princeton)
- David Spergel (Princeton / CCA)
- Francesco Villaescusa-Navarro (Princeton / CCA)

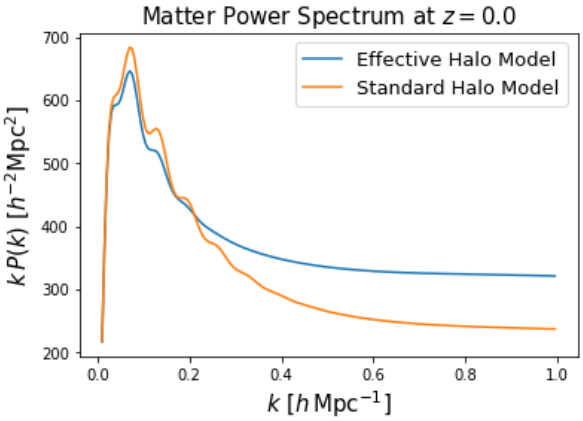
CHAPTER 1

Usage

To compute a matter power spectrum in EffectiveHalos, simply run the following:

```
from EffectiveHalos import *
import numpy as np

## Parameters
z = 0. # redshift
cs2 = 8. # effective speed of sound (should be calibrated from simulations)
R = 1. # smoothing scale (should be calibrated from simulations)
k = np.arange(0.01, 1., 0.005) # wavenumbers in h/Mpc

## Load general classes
cosmology = Cosmology(z, 'Planck18') # use Planck 2018 cosmology
mass_function = MassFunction(cosmology, 'Bhattacharya') # Bhattacharya 2010 mass␣
↪function
halo_physics = HaloPhysics(cosmology, mass_function, 'Duffy', 'NFW') # Duffy 08␣
↪concentration relation, NFW halo profiles

## Load HaloModel class
halo_model = HaloModel(cosmology, mass_function, halo_physics, k)

## Compute the power spectrum in both Effective and Standard Halo Models
power_spectrum_EHM = halo_model.halo_model(cs2, R)
power_spectrum_SHM = halo_model.halo_model(cs2, R, 'Linear', 0, 0, 0)
```

This generates an estimate for the matter power spectrum in a few seconds. Let's plot this:



A full tutorial is given on the tutorial page.

1.1 Installation

EffectiveHalos can be installed either from pip or by cloning the GitHub repository. Note that CLASS (and the classy Python wrapper) must be installed first, as described in *Dependencies*.

1.1.1 Installation via pip

We recommend that EffectiveHalos is installed via pip:

```
pip install EffectiveHalos (--user)
```

This installs the latest release of the code.

1.1.2 Installation from source

EffectiveHalos can also be installed directly from the git repository:

```
git clone https://github.com/oliverphilcox/EffectiveHalos.git
cd EffectiveHalos
python -m pip install . (--user)
```

This will install the current master branch of the git repository.

1.1.3 Dependencies

Basic Dependencies:

- numpy
- scipy
- cython (for CLASS)

CLASS

To run EffectiveHalos, we require the Boltzmann code CLASS along with its Python wrapper classy. This can be installed from the CLASS Github and is used to compute the linear power spectrum for a specified cosmology.

The basic installation follows:

```
# Clone the class repository
git clone https://github.com/lesgourg/class_public.git

# Now install
cd class_public
make clean
make
```

For further details, including the modifications required for Mac compilation, see the CLASS wiki. Note that, if a modified version of CLASS is installed (e.g. CLASS-PT) which modifies the classy wrapper, EffectiveHalos will use this instead.

FAST-PT

EffectiveHalos uses Joe McEwen's FAST-PT package to compute one-loop power spectra from the CLASS linear power spectrum. This is probably installed alongside EffectiveHalos. If not, it is easiest to install from pip:

```
pip install fast-pt (--user)
```

mcfit

EffectiveHalos uses the mcfit for integral transforms. This is probably installed alongside EffectiveHalos. If not, it is easiest to install from pip:

```
pip install mcfit (--user)
```

1.2 Python API

1.2.1 Cosmology Module

class EffectiveHalos.Cosmology.**Cosmology**(*redshift*, *name=''*, *verb=False*, *npoints=100000*, ***params*)

Class to hold the basic cosmology and CLASS attributes. This can be initialized by a set of cosmological parameters or a pre-defined cosmology.

Loaded cosmological models:

- **Planck18**: Bestfit cosmology from Planck 2018, using the baseline TT,TE,EE+lowE+lensing likelihood.
- **Quijote**: Fiducial cosmology from the Quijote simulations of Francisco Villaescusa-Navarro et al.
- **Abacus**: Fiducial cosmology from the Abacus simulations of Lehman Garrison et al.

Parameters

- **redshift** (*float*) – Desired redshift z
- **name** (*str*) – Load cosmology from a list of predetermined cosmologies (see above).
- **params** (*kwargs*) – Any other CLASS parameters. (Note that sigma8 does not seem to be supported by CLASS in Python 3).

Keyword Arguments

- **verb** (*bool*) – If true output useful messages througout run-time, default: False.
- **npoints** (*int*) – Number of points to use in the interpolators for sigma^2, default: 100000

compute_linear_power(*kh*, *kh_min=0.0*, *with_neutrinos=False*)
Compute the linear power spectrum from CLASS for a vector of input k.

If set, we remove any modes below some minimum k.

Parameters **kh** (*float, np.ndarray*) – Wavenumber or vector of wavenumbers (in h/Mpc units) to compute linear power with.

Keyword Arguments

- **kh_min** (*float*) – Value of k (in h/Mpc units) below which to set $P(k) = 0$, default: 0.
- **with_neutrinos** (*bool*) – If True, return the full matter power spectrum, else return the CDM+baryon power spectrum (which is generally used in the halo model). Default: False.

Returns Linear power spectrum in $(h^{-1}\mathrm{Mpc})^3$ units

Return type np.ndarray

dlns_dlogM_int(*logM_h*)
Return the value of $d\ln\sigma/d\log M$ using the prebuilt interpolators, which are constructed if not present.

Parameters **logM** (*np.ndarray*) – Input $\log_{10}(M/h^{-1}M_{\rm sun})$

Returns $d\ln\sigma/d\log M$

Return type np.ndarray

sigma_logM_int(*logM_h*)
Return the value of $\sigma(M,z)$ using the prebuilt interpolators, which are constructed if not present.

Parameters **logM** (*np.ndarray*) – Input $\log_{10}(M/h^{-1}M_{\rm sun})$

Returns $\sigma(M,z)$

Return type np.ndarray

1.2.2 Mass Function Module

class EffectiveHalos.MassFunction.**MassFunction**(*cosmology*, *mass_function_name='Bhattacharya'*, *verb=False*, ***mass_params*)
Class to hold a mass function for halos and associated bias.

Implemented Mass Functions:

- **Sheth-Tormen**: Sheth & Tormen 1999 analytic mass function. This assumes virialized halos, and uses the critical density from Nakamura & Suto 1997.
- **Tinker**: Tinker et al. 2010, eq. 8. This assume a spherical overdensity, the value of which can be specified.
- **Crocce**: Crocce et al. 2009, eq. 22. Calibrated from Friends-of-Friends halos with linking length 0.2
- **Bhattacharya**: Bhattacharya et al. 2010, eq. 12. Calibrated from Friends-of-Friends halos with linking length 0.2

Implemented Bias Functions:

- **Sheth-Tormen** Sheth-Tormen 2001, eq. 8 Associated to the 'Sheth-Tormen' mass function.
- **Tinker**: Tinker et al. 2010, eq. 15. Associated to the 'Tinker' mass function.
- **Crocce**: Peak-background split derivation from the 'Crocce' mass function of eq. 22, Crocce et al. 2009.
- **Bhattacharya**: Bhattacharya et al. 2010, eq. 18. Associated to the 'Bhattacharya' mass function.

Parameters

- **cosmology** (Cosmology) – Class instance containing relevant cosmological information
- **mass_function_name** (*str*) – Mass function to use (see above), default: 'Crocce'
- **mass_param** (*kwargs*) – Any additional parameters to pass to the class. These include: - tinker_overdensity: (Only for the Tinker mass function): spherical overdensity defining halos, default: 200

Keyword Arguments **verb** (*bool*) – If true output useful messages througout run-time, default: False.

linear_halo_bias(*m_h*)

Compute the linear halo bias, from the peak background split.

Associated bias functions are available for each mass function, and more can be user-defined. See the class description for details of the loaded parametrizations.

Parameters **m_h** (*np.ndarray*) – Array of masses in $h^{-1}M_{\rm sun}$ units.

Returns Linear Eulerian halo bias (dimensionless)

Return type np.ndarray

mass_function(*m_h*)

Return the mass function, equal to the number density of halos per unit logarithmic mass interval. This assumes the existence of a universal mass function, with

$$dn/d\log_{10} M = f(\sigma(M))\frac{\bar{\rho}_M}{M}\frac{d\ln\sigma(M)}{d\log_{10}(M)}$$

where f is the universal mass function, $\bar{\rho}_M$ is the mean matter density at redshift zero and $\sigma^2(M)$ is the overdensity variance on spheres with a radius given by the Lagrangian radius for mass M.

For details of the available mass function parametrizations, see the class description.

Note: For efficiency, we return the mass function $\frac{dn}{d\log_{10}(M)}$ rather than the standard form $\frac{dn}{dM}$.

Parameters **m_h** (*np.ndarray*) – Array of masses in $h^{-1}M_{\rm sun}$ units.

Returns Mass function, $dn/d\log_{10}(M/h^{-1}M_{\rm sun})$ in $h^3{\rm Mpc}^{-3}$ units

Return type np.ndarray

second_order_bias(*m_h*)

Compute the second order Eulerian bias, defined as $\frac{4}{21}b_1^L + \frac{1}{2}b_2^L$ where b_1^L and b_2^L are the Lagrangian bias parameters.

Associated bias functions are available for each mass function, and more can be user-defined. See the class description for details of the loaded parametrizations.

Args: - m_h: Mass in $h^{-1}M_{\rm sun}$ units.

Returns Quadratic Eulerian halo bias (dimensionless)

Return type np.ndarray

1.2.3 Halo Physics Module

class EffectiveHalos.HaloPhysics.**HaloPhysics**(*cosmology*, *mass_function*, *concentration_name='Duffy'*, *profile_name='NFW'*, *min_logM_h=6*, *max_logM_h=17*, *npoints=100000*, *halo_overdensity=200*, *verb=False*)

Class to hold halo model quantities and relevant integrals.

Implemented Concentration Functions:

- **Duffy**: Duffy et al. (2008) for virial-density haloes (second section in Table 1)

Implemented Halo Profiles:

- **NFW**: Navarro, Frenk & White (1997) universal halo profile.

Parameters

- **cosmology** (Cosmology) – Instance of the Cosmology class containing relevant cosmology and functions.
- **mass_function** (MassFunction) – Instance of the MassFunction class, containing the mass function and bias
- **concentration_name** (*str*) – Concentration parametrization to use (see above), default: 'Duffy'
- **profile_name** (*str*) – Halo profile parametrization to use (see above), default: 'NFW'
- **hyperparams** (*kwargs*) – Any additional parameters to pass to the class (see below).

Keyword Arguments

- **min_logM_h** (*float*) – Minimum halo mass in $\log_{10}(M/h^{-1}M_{\rm sun})$, default: 6
- **max_logM_h** (*float*) – Maximum halo mass in $\log_{10}(M/h^{-1}M_{\rm sun})$, default: 17
- **npoints** (*int*) – Number of sampling points in mass for $\sigma(M)$ interpolation and mass integrals, default: 1e5
- **halo_overdensity** (*float*) – Characteristic halo overdensity in units of background matter density. Can be a fixed value or 'virial', whereupon the virial collapse overdensity relation of Bryan & Norman 1998 to construct this. Default: 200.
- **verb** (*bool*) – If true output useful messages througout run-time, default: False.

halo_concentration(*m_h*)

Compute the halo concentration $c = r_{\rm virial}/r_{\rm scale}$.

For details of the available concentration parametrizations, see the class description.

Parameters **m_h** (*np.ndarray*) – Mass in $h^{-1}M_{\rm sun}$ units.

Returns Array of concentration parameters.

Return type np.ndarray

halo_profile(*m_h*, *kh*, *norm_only=False*)

Compute the halo profile function in Fourier space; $\rho(k|m) = \frac{m}{\bar{\rho}_M} u(k|m)$ where $\bar{\rho}_M$ is the background matter density and $u(k|m)$ is the halo profile.

We assume halos have a virial collapse overdensity here, based on the parametrization of Bryan & Norman 1998.

For details of the available profile parametrizations, see the class description.

Parameters

- **m_h** (*np.ndarray*) – Mass in $h^{-1}M_{\rm sun}$ units.
- **kh** (*np.ndarray*) – Wavenumber in h/Mpc units.
- **norm_only** (*bool*) – Boolean, if set, just return the normalization factor $m/\bar{\rho}_M$, default: False

Returns Halo profile $\rho(k|m)$ or $m/\bar{\rho}_M$, if the norm_only parameter is set.

Return type np.ndarray

1.2.4 Mass Integrals Module

class EffectiveHalos.MassIntegrals.**MassIntegrals**(*cosmology*, *mass_function*, *halo_physics*, *kh_vector*, *min_logM_h=6.001*, *max_logM_h=16.999*, *npoints=10000*, *verb=False*, *m_low=-1*)

Class to compute and store the various mass integrals of the form

$$I_p^{q_1,q_2}(k_1,...k_p) = \int n(m) b^{(q_1)}(m) b^{(q_2)} \frac{m^p}{\rho_M^p} u(k_1|m)..u(k_p|m) dm$$

which are needed to compute the power spectrum model. Here $b^{(q)}$ is the q-th order bias (with $b^0 = 1$), $u(k|m)$ is the normalized halo profile and $n(m)$ is the mass function.

All integrals are performed via Simpson's rule over a specified mass range, and are simply returned if they are already computed.

For the $I_1^{1,0} = I_1^1$ integral, the integral must be corrected to ensure that we recover the bias consistency relation $I_1^1 \to 1$ as $k \to 0$.

This requires an infinite mass range, so we instead approximate;

$$I_1^1(k)_{\rm corrected} = I_1^1(k) + (1 - I_1^1(0)) \frac{u(k|m_m in)}{u(k|0)}$$

for normalized halo profile $u(k|m)$.

Note that this can also be used to compute ${}_iJ_p^{q_1,q_2}$ and ${}_iK_p^{q_1,q_2}[f]$ type integrals required for the exclusion counts covariance.

Parameters

- **cosmology** (Cosmology) – Instance of the Cosmology class containing relevant cosmology and functions.
- **mass_function** (MassFunction) – Instance of the MassFunction class, containing the mass function and bias.
- **halo_physics** (HaloPhysics) – Instance of the HaloPhysics class, containing the halo profiles and concentrations.
- **kh_vector** (*np.ndarray*) – Array (or float) of wavenumbers in $h{\rm Mpc}^{-1}$ units from which to compute mass integrals.

Keyword Arguments

- **min_logM_h** (*float*) – Minimum mass in $\log_{10}(M/h^{-1}M_{\rm sun})$ units, default: 6.001.
- **max_logM_h** (*float*) – Maximum mass in $\log_{10}(M/h^{-1}M_{\rm sun})$ units, default: 16.999.
- **npoints** (*int*) – Number of logarithmically spaced mass grid points, default: 10000.
- **verb** (*bool*) – If true output useful messages througout run-time, default: False.

compute_I_00()
Compute the I_0^0 integral, if not already computed.

Returns Value of I_0^0

Return type float

compute_I_01()
Compute the I_0^1 integral, if not already computed.

Returns Value of I_0^1

Return type float

compute_I_02()
Compute the I_0^2 integral, if not already computed.

Returns Value of I_0^2

Return type float

compute_I_10(*apply_correction=False*)
Compute the I_1^0 integral, if not already computed. Also apply the correction noted in the class header if required.

When computing ${}_iJ_1^1$ type integrals (over a finite mass bin), the correction should *not* be applied.

Keyword Arguments **apply_correction** (*bool*) – Whether to apply the correction in the class header to ensure the bias consistency relation is upheld.

Returns Array of I_1^0 values for each k.

Return type float

compute_I_11(*apply_correction=True*)
Compute the $I_1^1(k)$ integral, if not already computed. Also apply the correction noted in the class header if required.

When computing ${}_iJ_1^1$ type integrals (over a finite mass bin), the correction should *not* be applied.

Keyword Arguments **apply_correction** (*bool*) – Whether to apply the correction in the class header to ensure the bias consistency relation is upheld.

Returns Array of I_1^1 values for each k.

Return type np.ndarray

compute_I_111()
Compute the $I_1^{1,1}(k)$ integral, if not already computed.

Returns Array of $I_1^{1,1}$ values for each k.

Return type np.ndarray

compute_I_12(*apply_correction=True*)
Compute the $I_1^2(k)$ integral, if not already computed. Also apply the correction noted in the class header if required.

When computing ${}_iJ_1^2$ type integrals (over a finite mass bin), the correction should *not* be applied.

Keyword Arguments **apply_correction** (*bool*) – Whether to apply the correction in the class header to ensure the bias consistency relation is upheld.

Returns Array of I_1^2 values for each k.

Return type np.ndarray

compute_I_20()
Compute the $I_2^0(k, k)$ integral, if not already computed. Note that we assume both k-vectors are the same here.

Returns Array of I_2^0 values for each k.

Return type np.ndarray

compute_I_21()
Compute the $I_2^1(k, k)$ integral, if not already computed. Note that we assume both k-vectors are the same here.

Returns Array of I_2^1 values for each k.

Return type np.ndarray

compute_K_PTheta_11(*alpha*, *PTheta_interpolator*)
Compute the $K_2^1[Pst\Theta](k)$ integral, if not already computed. Θ is the Fourier transform of the exclusion window function which is convolved with the power spectrum.

Parameters

- **alpha** (*float*) – Dimensionless ratio of exclusion to Lagrangian halo radius
- **PTheta_interpolator** (*interp1d*) – Interpolator for the non-linear $S(R_{\rm ex})$ function

Returns Array of $K_1^1[Pst\Theta](k)$ values for each k.

Return type np.ndarray

compute_K_S_01(*alpha*, *S_L_interpolator*)
Compute the $K_0^1[S](k)$ integral, if not already computed. S is the integral of the 2PCF windowed by the halo exclusion function of radius $R_{\rm ex}$.

Parameters

- **alpha** (*float*) – Dimensionless ratio of exclusion to Lagrangian halo radius
- **S_L_interpolator** (*interp1d*) – Interpolator for the linear $S(R_{\rm ex})$ function

Returns Array of $K_0^1[S](k)$ values for each k.

Return type np.ndarray

compute_K_S_21(*alpha*, *S_NL_interpolator*)
Compute the $K_2^1[S](k)$ integral, if not already computed. S is the integral of the 2PCF windowed by the halo exclusion function of radius $R_{\rm ex}$. Note this function uses the non-linear form.

Parameters

- **alpha** (*float*) – Dimensionless ratio of exclusion to Lagrangian halo radius
- **S_NL_interpolator** (*interp1d*) – Interpolator for the non-linear $S(R_{\rm ex})$ function

Returns Array of $K_2^1[S](k)$ values for each k.

Return type np.ndarray

compute_K_Theta_01 (*alpha*)
Compute the $K_0^1[\Theta](k)$ integral, if not already computed. $Theta$ is the Fourier transform of the exclusion window function.

Parameters **alpha** (*float*) – Dimensionless ratio of exclusion to Lagrangian halo radius

Returns Array of $K_0^1[\Theta](k)$ values for each k.

Return type np.ndarray

compute_K_Theta_10 (*alpha*)
Compute the $K_1^0[\Theta](k)$ integral, if not already computed. $Theta$ is the Fourier transform of the exclusion window function.

Parameters **alpha** (*float*) – Dimensionless ratio of exclusion to Lagrangian halo radius

Returns Array of $K_1^0[\Theta](k)$ values for each k.

Return type np.ndarray

compute_K_V_11 (*alpha*)
Compute the $K_1^1[V](k)$ integral, if not already computed. **:mathrm:'V'** is the volume of the exclusion window function.

Parameters **alpha** (*float*) – Dimensionless ratio of exclusion to Lagrangian halo radius

Returns Array of $K_1^1[V](k)$ values for each k.

Return type np.ndarray

compute_K_V_20 (*alpha*)
Compute the $K_2^0[V](k)$ integral, if not already computed. **:mathrm:'V'** is the volume of the exclusion window function.

Parameters **alpha** (*float*) – Dimensionless ratio of exclusion to Lagrangian halo radius

Returns Array of $K_2^0[V](k)$ values for each k.

Return type np.ndarray

1.2.5 Halo Model Module

class EffectiveHalos.HaloModel.**HaloModel**(*cosmology*, *mass_function*, *halo_physics*, *kh_vector*, *kh_min=0*, *verb=False*)
Class to compute the non-linear power spectrum from the halo model of Philcox et al. 2020.

The model power is defined as

$$P_{\rm model} = I_1^1(k)^2 P_{\rm NL}(k) W^2(kR) + I_2^0(k,k)$$

where I_p^q are mass function integrals defined in the MassIntegrals class, $P_{\rm NL}$ is the 1-loop non-linear power spectrum from Effective Field Theory and $W(kR)$ is a smoothing window on scale R.

Parameters

- **cosmology** (Cosmology) – Instance of the Cosmology class containing relevant cosmology and functions.
- **mass_function** (MassFunction) – Instance of the MassFunction class, containing the mass function and bias.
- **halo_physics** (HaloPhysics) – Instance of the HaloPhysics class, containing the halo profiles and concentrations.

- **kh_vector** (*np.ndarray*) – Vector of wavenumbers (in h/Mpc units), for which power spectrum will be computed.

Keyword Arguments

- **kh_min** – Minimum k vector in the simulation (or survey) region in h/Mpc units. Modes below kh_min are set to zero, default: 0.
- **verb** (*bool*) – If true, output useful messages througout run-time, default: False.

compute_one_loop_only_power()

Compute the one-loop SPT power from the linear power spectrum in the Cosmology class. This returns the one-loop power evaluated at the wavenumber vector specfied in the class initialization. When first called, this computes an interpolator function, which is used in this and subsequent calls.

Returns Vector of 1-loop power $P_{1-\mathrm{loop}}(k)$ for the input k-vector.

Return type np.ndarray

compute_resummed_linear_power()

Compute the IR-resummed linear power spectrum, using the linear power spectrum in the Cosmology class.

The output power is defined by

$$P_{\mathrm{lin,IR}}(k) = P_{\mathrm{lin,nw}}(k) + P_{\mathrm{lin,w}}(k)e^{-k^2\Sigma^2}$$

where 'nw' and 'w' refer to the no-wiggle and wiggle parts of the linear power spectrum and Σ^2 is the BAO damping scale (computed in the _prepare_IR_resummation function)

If already computed, the IR resummed linear power is simply returned.

Returns Vector of IR-resummed linear power $P_{\mathrm{lin,IR}}(k)$ for the input k-vector.

Return type np.ndarray

compute_resummed_one_loop_power()

Compute the IR-resummed linear-plus-one-loop power spectrum, using the linear power spectrum in the Cosmology class.

The output power is defined by

$$P_{\mathrm{lin+1,IR}}(k) = P_{\mathrm{lin,nw}}(k) + P_{1-\mathrm{loop,nw}}(k) + e^{-k^2\Sigma^2}[P_{\mathrm{lin,w}}(k)(1+k^2\Sigma^2) + P_{1-\mathrm{loop,w}}(k)]$$

where 'nw' and 'w' refer to the no-wiggle and wiggle parts of the linear / 1-loop power spectrum and $Sigma^2$ is the BAO damping scale (computed in the _prepare_IR_resummation function)

Returns Vector of IR-resummed linear-plus-one-loop power $P_{\mathrm{lin+1,IR}}(k)$ for the input k-vector.

Return type np.ndarray

halo_model(*cs2*, *R*, *pt_type='EFT'*, *pade_resum=True*, *smooth_density=True*, *IR_resum=True*, *include_neutrinos=True*, *return_terms=False*)

Compute the non-linear halo-model power spectrum to one-loop order, with IR corrections and counterterms. Whilst we recommend including all non-linear effects, these can be optionally removed with the Boolean parameters.

This is similar to the 'non_linear_power()' function, but includes the halo mass integrals, and is the *complete* model of the matter power spectrum at one-loop-order in our approximations. Note that the function requires two free parameters; the smoothing scale R and the effective squared sound-speed c_s^2, which cannot be predicted from theory. (Note that $c_s^2 < 0$ is permissible).

For massive neutrino cosmologies, we assume that the matter power spectrum is given by a mass-fraction-weighted sum of the halo model CDM+baryon power spectrum, linear neutrino spectrum and

linear neutrino cross CDM+baryon spectrum. This is a good approximation in practice (i.e. including halo-model effects only for the CDM+baryon component.) The function can return either the halo-model CDM+baryon power spectrum (suitable for comparison to CDM+baryon power spectra) or the combined CDM+baryon+neutrino power spectrum (suitable for comparison to matter spectra) using the 'include_neutrinos' flag. When 'include_neutrinos' is specified the model has three components; the weighted two-halo CDM+baryon part, the weighted one-halo CDM+baryon part and the weighted linear neutrino and cross spectra. The sum of all three and the first two individually are returned by the 'return_terms' command.

For further details, see the class description.

Parameters

- **cs2** (*float*) – Squared-speed-of-sound counterterm c_s^2 in $(h^{-1}\mathrm{Mpc})^2$ units. (Unused if pt_type is not "EFT")
- **R** (*float*) – Smoothing scale in $h^{-1}Mpc$. This is a free parameter of the model. (Unused if smooth_density = False)

Keyword Arguments

- **pt_type** (*str*) – Which flavor of perturbation theory to adopt. Options 'EFT' (linear + 1-loop + counterterm), 'SPT' (linear + 1-loop), 'Linear', default: 'EFT'
- **pade_resum** (*bool*) – If True, use a Pade resummation of the counterterm $k^2/(1+k^2)P_{\rm lin}$ rather than $k^2 P_{\rm lin}(k)$, default: True
- **smooth_density** (*bool*) – If True, smooth the density field on scale R, i.e. multiply power by W(kR)^2, default: True
- **IR_resum** (*bool*) – If True, perform IR resummation on the density field to resum non-perturbative long-wavelength modes, default: True
- **include_neutrinos** (*bool*) – If True, return the full power spectrum of CDM+baryons+neutrinos (with the approximations given above). If False, return only the CDM+baryon power spectrum. This has no effect in cosmologies without massive neutrinos, default: True.
- **return_terms** (*bool*) – If True, return the one- and two-halo CDM+baryon halo-model terms in addition to the combined power spectrum model, default: False

Returns Non-linear halo model power spectrum $P_{\rm halo}$ evaluated at the input k-vector. np.ndarray: One-halo power spectrum term (if return_terms is true) np.ndarray: Two-halo power spectrum term (if return_terms is true)

Return type np.ndarray

non_linear_power(*cs2*, *R*, *pt_type='EFT'*, *pade_resum=True*, *smooth_density=True*, *IR_resum=True*, *include_neutrinos=True*)

Compute the non-linear power spectrum to one-loop order, with IR corrections and counterterms. Whilst we recommend including all non-linear effects, these can be optionally removed with the Boolean parameters. Setting (pt_type='Linear', pade_resum=False, smooth_density=False, IR_resum = False) recovers the standard halo model prediction.

Including all relevant effects, this is defined as

$$P_{\rm NL}(k,R,c_s^2) = [P_{\rm lin}(k) + P_{\rm 1-loop}(k) + P_{\rm counterterm}(k;c_s^2)]W(kR)$$

where

$$P_{\rm counterterm}(k;c_s^2) = -c_s^2\frac{k^2}{(1+k^2)}P_{\rm lin}(k)$$

is the counterterm, and IR resummation is applied to all spectra.

This computes the relevant loop integrals if they haven't already been computed. The function returns $P_{\rm NL}$ given smoothing scale R and effective squared sound-speed c_s^2.

For massive neutrino cosmologies, we assume that the matter power spectrum is given by a mass-fraction-weighted sum of the non-linear CDM+baryon power spectrum, linear neutrino spectrum and linear neutrino cross CDM+baryon spectrum. This is a good approximation for the halo model spectra (i.e. including non-linear effects only for the CDM+baryon component.) The function can return either the non-linear CDM+baryon power spectrum or the combined CDM+baryon+neutrino power spectrum using the 'include_neutrinos' flag.

Parameters

- **cs2** (*float*) – Squared-speed-of-sound counterterm c_s^2 in $(h^{-1}{\rm Mpc})^2$ units. (Unused if pt_type is not "EFT")
- **R** (*float*) – Smoothing scale in $h^{-1}Mpc$. This is a free parameter of the model. (Unused if smooth_density = False)

Keyword Arguments

- **pt_type** (*str*) – Which flavor of perturbation theory to adopt. Options 'EFT' (linear + 1-loop + counterterm), 'SPT' (linear + 1-loop), 'Linear', default: 'EFT'
- **pade_resum** (*bool*) – If True, use a Pade resummation of the counterterm $k^2/(1+k^2)P_{\rm lin}$ rather than $k^2 P_{\rm lin}(k)$, default: True
- **smooth_density** (*bool*) – If True, smooth the density field on scale R, i.e. multiply power by W(kR)^2, default: True
- **IR_resum** (*bool*) – If True, perform IR resummation on the density field to resum non-perturbative long-wavelength modes, default: True
- **include_neutrinos** (*bool*) – If True, return the full power spectrum of CDM+baryons+neutrinos (with the approximations given above). If False, return only the CDM+baryon power spectrum. This has no effect in cosmologies without massive neutrinos, default: True.

Returns Non-linear power spectrum $P_{\rm NL}$ evaluated at the input k-vector.

Return type np.ndarray

1.2.6 Counts Covariance Module

class EffectiveHalos.CountsCovariance.**CountsCovariance**(*cosmology*, *mass_function*, *halo_physics*, *kh_vector*, *mass_bins*, *volume*, *kh_min=0*, *pt_type='EFT'*, *pade_resum=True*, *smooth_density=True*, *IR_resum=True*, *npoints=1000*, *verb=False*)

Class to compute the covariance of cluster counts and the non-linear power spectrum using the halo model of Philcox et al. 2020. We provide routines for both the N_i-N_j and N_i-$P(k)$ covariance where N_i is the halo count in a mass bin defined by $[m_{{\rm low},i}, m_{{\rm high},i}]$

In the Effective Halo Model, the covariance between X and Y is defined as

$$\mathrm{cov}(X,Y) = \mathrm{cov}_{\rm intrinsic}(X,Y) + \mathrm{cov}_{\rm exclusion}(X,Y) + \mathrm{cov}_{\rm super-sample}(X,Y).$$

The full expressions for the cluster auto-covariance and cross-covariance with the power spectrum are lengthy but can be found in Philcox et al. (2020). These depend on mass function integrals, I_p^q, ${}_iJ_p^q$ and ${}_iK_p^q[f]$ which are computed in the MassIntegrals class for mass bin i, P_{NL} is the 1-loop non-linear power spectrum from Effective Field Theory and $W(kR)$ is a smoothing window on scale R.

Parameters

- **cosmology** (Cosmology) – Class containing relevant cosmology and functions.
- **mass_function** (MassFunction) – Class containing the mass function and bias.
- **halo_physics** (HaloPhysics) – Class containing the halo profiles and concentrations.
- **kh_vector** (*np.ndarray*) – Vector of wavenumbers (in h/Mpc units), for which power spectra will be computed.
- **mass_bins** (*np.ndarray*) – Array of mass bin edges, in $h^{-1}M_{\rm sun}$ units. Must have length N_bins + 1.
- **volume** – Volume of the survey in $(h^{-1}\mathrm{Mpc})^3$.

Keyword Arguments

- **kh_min** – Minimum k vector in the simulation (or survey) region in h/Mpc units. Modes below kh_min are set to zero, default 0.
- **pt_type** (*str*) – Which flavor of perturbation theory to adopt. Options 'EFT' (linear + 1-loop + counterterm), 'SPT' (linear + 1-loop), 'Linear', default: 'EFT'
- **pade_resum** (*bool*) – If True, use a Pade resummation of the counterterm $k^2/(1+k^2)P_{\rm lin}$ rather than $k^2P_{\rm lin}(k)$, default: True
- **smooth_density** (*bool*) – If True, smooth the density field on scale R, i.e. multiply power by W(kR)^2, default: True
- **IR_resum** (*bool*) – If True, perform IR resummation on the density field to resum non-perturbative long-wavelength modes, default: True
- **npoints** (*int*) – Number of mass bins to use in numerical integration, default: 1000
- **verb** (*bool*) – If true output useful messages througout run-time, default: False.

NN_covariance(*cs2*, *R*, *alpha*, *sigma2_volume=-1*, *use_exclusion=True*, *use_SSC=True*)

Compute the full covariance matrix of cluster counts N_i, N_j as defined in the class description.

An important parameter is $\sigma^2(V)$, the variance of the (linear) density field across the survey or simulation box region. If this is not specified, it will be computed from the volume of the survey, assuming spherical symmetry. Note that this is rarely a valid assumption in practice.

Furthermore, note that the c_s^2 and R parameters have only a minor impact on the covariances here, whilst the $lpha$ parameter is important, since it controls halo exclusion.

Using the parameters 'use_exclusion' and 'use_SSC' the user can choose which parts of the covariance should be returned.

Parameters

- **cs2** (*float*) – Squared-speed-of-sound c_s^2 counterterm in $(h^{-1}\mathrm{Mpc})^2$ units. This should be set by fitting the power spectrum model. (Unused if pt_type is not "EFT")
- **R** (*float*) – Smoothing scale in :math:h^{-1}'mathrm{Mpc}'. This should be set by fitting the power spectrum model. (Unused if smooth_density = False)

- **alpha** (*float*) – Dimensionless ratio of the halo exclusion radius to the halo Lagrangian radius. (Unused if use_exclusion = False)

Keyword Arguments

- **sigma2_volume** (*float*) – The variance of the linear density field across the survey. This will be computed from the survey volume, assuming isotropy, if not provided. (Unused if use_SSC = False)
- **use_excluson** (*bool*) – Whether to include the halo exclusion terms, default: True
- **use_SSC** (*bool*) – Whether to include the super-sample covariance (SSC) terms, default: True

Returns Two-dimensional array of $\mathrm{cov}(N_i, N_j)$ with shape (N_bins, N_bins) for N_bins mass bins.

Return type np.ndarray

NP_covariance(*cs2*, *R*, *alpha*, *sigma2_volume=-1*, *use_exclusion=True*, *use_SSC=True*)

Compute the full covariance matrix of cluster counts and the matter power spectrum $N_i, P(k)$ as defined in the class description.

An important parameter is $\sigma^2(V)$, the variance of the (linear) density field across the survey or simulation box region. If this is not specified, it will be computed from the volume of the survey, assuming spherical symmetry. Note that this is rarely a valid assumption in practice.

Using the parameters 'use_exclusion' and 'use_SSC' the user can choose which parts of the covariance should be returned.

Parameters

- **cs2** (*float*) – Squared-speed-of-sound c_s^2 counterterm in $(h^{-1}\mathrm{Mpc})^2$ units. This should be set by fitting the power spectrum model. (Unused if pt_type is not "EFT")
- **R** (*float*) – Smoothing scale in :math:h^{-1}‘mathrm{Mpc}‘. This should be set by fitting the power spectrum model. (Unused if smooth_density = False)
- **alpha** (*float*) – Dimensionless ratio of the halo exclusion radius to the halo Lagrangian radius. (Unused if use_exclusion = False)

Keyword Arguments

- **sigma2_volume** (*float*) – The variance of the linear density field across the survey. This will be computed from the survey volume, assuming isotropy, if not provided. (Unused if use_SSC = False)
- **use_excluson** (*bool*) – Whether to include the halo exclusion terms, default: True
- **use_SSC** (*bool*) – Whether to include the super-sample covariance (SSC) terms, default: True

Returns Two-dimensional array of $\mathrm{cov}(N_i, P(k))$ with shape (N_bins, N_k) for N_bins mass bins and N_k power spectrum bins.

Return type np.ndarray

1.3 Tutorial

1.3.1 Initializing the Module

To use EffectiveHalos, first load the Cosmology module to set the cosmological parameters and redshift. The class accepts either the name of a pre-loaded cosmology, or any parameters used by CLASS. For a list of available cosmologies see the docstrings, or the full API.

Here we'll initialize with the cosmology used in the Quijote simulations at redshift zero:

[1]:
```
from EffectiveHalos import *
import numpy as np
import matplotlib.pyplot as plt

z = 0.
cosmology = Cosmology(z, name = 'Quijote', verb = True) # set verb = True to display␣
↪useful messages throughout runtime
```

```
Loading the Quijote cosmology at z = 0.00
Loading CLASS
```

Next, load the mass function. Here we'll use the prescription of Bhattacharya et al. 2010:

[2]:
```
mass_function = MassFunction(cosmology, mass_function_name = 'Bhattacharya', verb =␣
↪True)
```

```
Using fitted parameters for the Bhattacharya mass function from Quijote simulations
```

Finally, the HaloPhysics class must be initialized. This includes the halo profile and concentration. We'll use the standard NFW profiles (Navarro et al. 1997) and the halo concentration prescription of Duffy et al. 2008:

[3]:
```
halo_physics = HaloPhysics(cosmology, mass_function, concentration_name = 'Duffy',␣
↪profile_name = 'NFW', verb = True)
```

```
Creating an interpolator for sigma(M) and its derivative.
```

This class contains a number of optional arguments controlling interpolation and the mass function limits. For more information, see the module API.

1.3.2 Computing Power Spectra

In this section, we'll compute model power spectra with free parameters fitted to the mean of 100 high-resolution Quijote N-body simulations. This is provided with the module, using $k_{\rm max} = 0.8h\,{\rm Mpc}^{-1}$.

The first step is to load the simulation data, and initialize the HaloModel class:

[9]:
```
# Load simulated data
k, Pk, Pk_err = np.loadtxt('/home/ophilcox/EffectiveHalos/quijote_HR_spectra_z0.txt',␣
↪unpack = True)

# Initialize the HaloPower class
halo_model = HaloModel(cosmology, mass_function, halo_physics, k, verb = True)
```

Power spectra can be computed using the `halo_model()` function. This features a number of optional arguments which control various features of the power spectrum model. The full effective halo model is obtained by setting these to their default values.

The Effective Halo Model requires two free parameters; the effective sound speed c_s^2 and the density field smoothing scale R. Here we'll set them by comparing the model spectrum to the Quijote simulations.

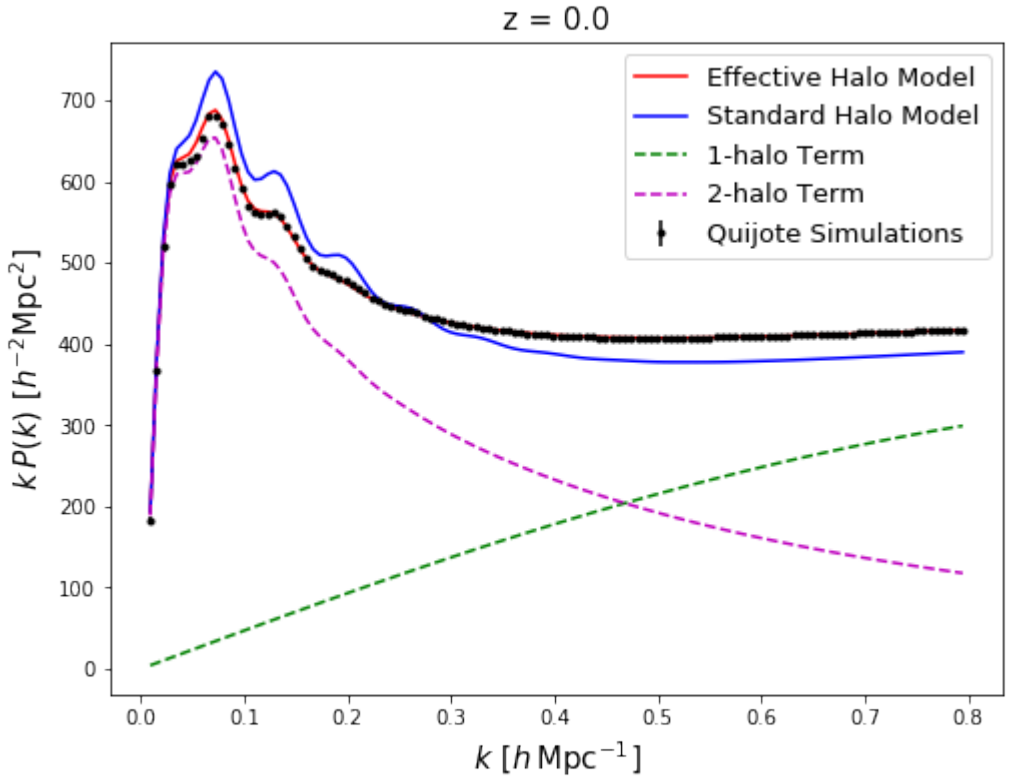
Note that power spectra computation takes a few seconds the first time it is run since numerical integrals must be computed, but negligible time for any additional runs.

[10]:
```
# Create a simple Gaussian likelihood
def likelihood(parameters):
    cs2, R = parameters
    model_Pk = halo_model.halo_model(cs2, R)
    return np.sum((model_Pk - Pk)**2. / Pk_err**2.)

# Optimize free parameters
from scipy.optimize import minimize
p0 = [1., 1.]
output = minimize(likelihood, p0)
cs2, R = output.x

print("Optimal parameters are c_s^2 = %.2f, R = %.2f"%(cs2, R))
```

```
Computing one-loop power spectrum
Non-linear BAO damping scale = 5.47 Mpc/h
Computing one-loop power spectrum
Optimal parameters are c_s^2 = 9.36, R = 1.76
```

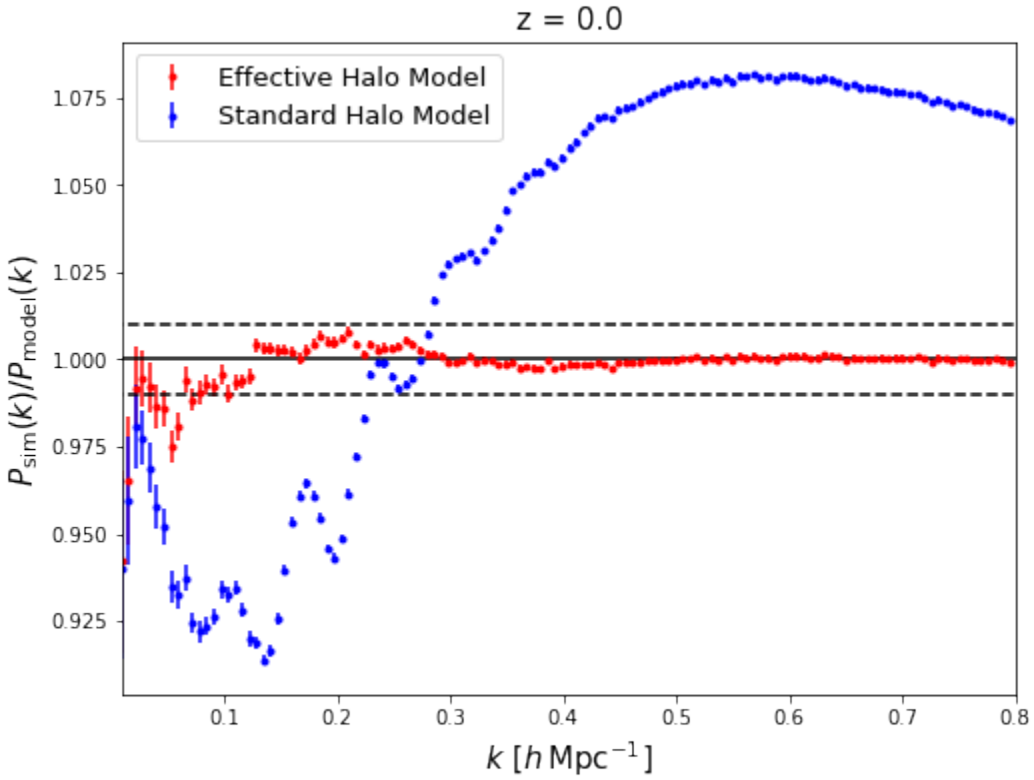
Let's plot the fitted power spectrum from the effective halo model, alongside the prediction from the standard halo model. Note that we canalso return the one- and two-halo terms separately using the `return_terms` argument of the `halo_model()` function.

[11]:
```
# Compute the spectrum using the effective halo model
power_EHM, power_1h, power_2h = halo_model.halo_model(cs2, R, return_terms = True)

# Compute the spectrum using the standard halo model
power_SHM = halo_model.halo_model(cs2, R, pt_type = 'Linear', pade_resum = False,␣
↪smooth_density = False,
                                  IR_resum = False, return_terms = False)

# Plot the results
plt.figure(figsize = (8,6))
plt.plot(k, k * power_EHM, c = 'r', label = 'Effective Halo Model')
plt.plot(k, k * power_SHM, c = 'b', label = 'Standard Halo Model')
plt.errorbar(k, k * Pk, yerr = k * Pk_err, ls = '', marker = '.', c = 'k', label =
↪'Quijote Simulations')
plt.plot(k, k * power_1h, ls = '--', c = 'g', label = '1-halo Term')
plt.plot(k, k * power_2h, ls = '--', c = 'm', label = '2-halo Term')
plt.xlabel(r'$k$ [$h\,\mathrm{Mpc}^{-1}$]', fontsize = 15)
plt.ylabel(r'$k\,P(k)$ [$h^{-2}\mathrm{Mpc}^{2}$]', fontsize = 15)
plt.title(r'z = %s'%z, fontsize = 15)
plt.legend(fontsize = 13)
plt.show()
```



To see this in more detail, let's plot the ratio:

[12]:
```
plt.figure(figsize = (8,6))
plt.errorbar(k, Pk / power_EHM, yerr = Pk_err / power_EHM, ls = '', marker = '.', c =
↪'r', label = 'Effective Halo Model')
plt.errorbar(k, Pk / power_SHM, yerr = Pk_err / power_SHM, ls = '', marker = '.', c =
↪'b', label = 'Standard Halo Model')
plt.xlabel(r'$k$ [$h\,\mathrm{Mpc}^{-1}$]', fontsize = 15)
plt.ylabel(r'$P_\mathrm{sim}(k) / P_\mathrm{model}(k)$', fontsize = 15)
plt.title(r'z = %s'%z, fontsize = 15)
plt.legend(fontsize = 13)
plt.hlines(1., 0, 1)
plt.hlines([0.99, 1.01], 0, 1, linestyles = '--')
plt.xlim([0.01, 0.8])
plt.show()
```



1.3.3 Computing Covariance Matrices

EffectiveHalos can be used to compute the covariance matrices of halo counts. This is done using the `CountsCovariance` class. Below, we compute and plot the covariance matrix for a selection of massive halos using the `NN_covariance()` function. This uses the optimal parameters c_s^2 and R found above. We will additionally set the exclusion parameter α to $1/2$.

Note that there are three contributions to the covariance; intrinsic, extrinsic and super-sample covariances. These can be turned off using the `use_SSC` and `use_exclusion` arguments. An important hyperparameter is $\sigma^2(V)$; the variance of the linear density field across the survey or simulation. This can be computed separately and passed to EffectiveHalos. If this is not present, it will be computed using CLASS, assuming that the survey is isotropic (rarely a valid assumption).

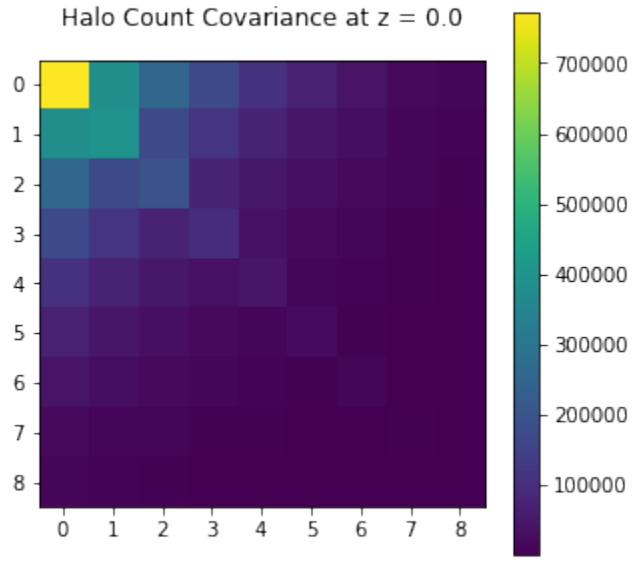
[13]:
```
# Define parameters
mass_bins = np.logspace(13, 15, 10) # mass bin limits in Msun/h units
volume = 1000.**3. # survey volume in (Mpc/h)^3
alpha = 0.5 # exclusion parameter, in range (0, 1)

# Load the class
counts_covariance = CountsCovariance(cosmology, mass_function, halo_physics, k, mass_
↪bins, volume, verb = True)

# Compute the covariance of halo counts
cov_NN = counts_covariance.NN_covariance(cs2, R, alpha, use_SSC = True, use_exclusion␣
↪= True)

# Plot the covariance
```

(continues on next page)

(continued from previous page)

```
fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot(111)
cax = ax.matshow(cov_NN)
fig.colorbar(cax)
ax.xaxis.tick_bottom()
ax.set_title(r'Halo Count Covariance at z = %s'%z);
```

```
Computing intrinsic covariance terms
Computing mass integrals
Computing exclusion covariance terms
Computing interpolation grid for non-linear S function
Computing super-sample covariance terms
Computing halo count response
Note: Variance of the linear density field sigma^2(V) not provided. This will be␣
↪computed assuming the survey volume is isotropic.
```



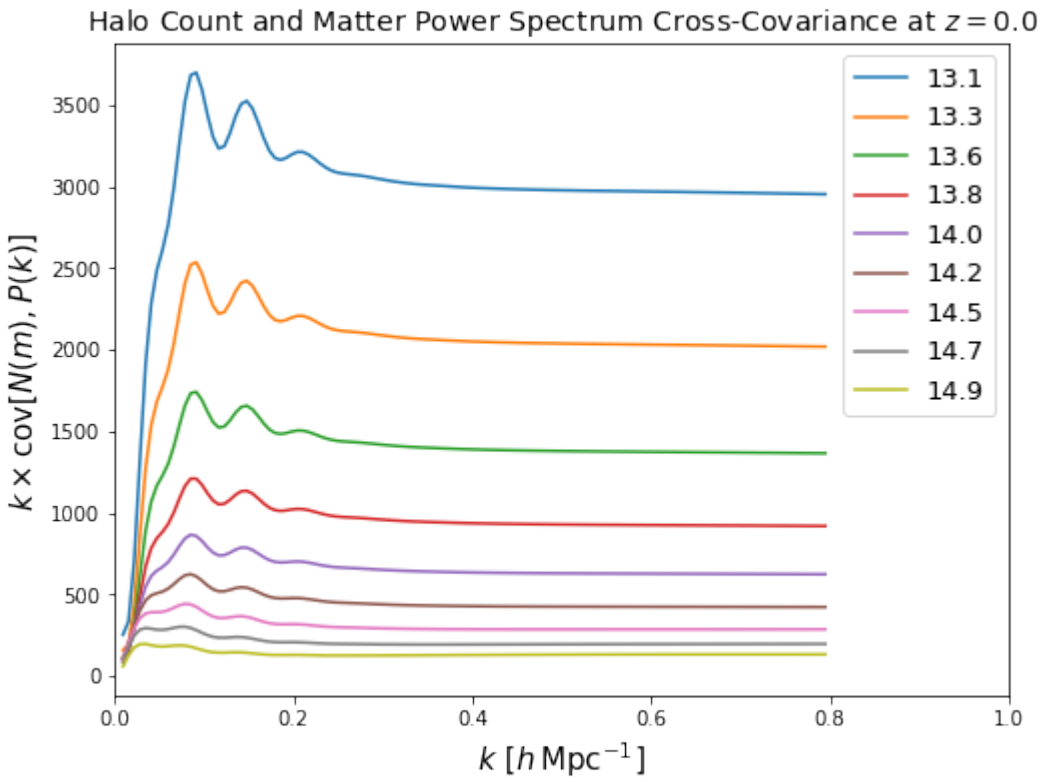
Finally, we can compute the covariance between halo counts and the matter power spectrum using the `NP_covariance()` function. We'll do this for the mass bins specfied above, including all sources of covariance.

[14]:
```
# Compute the covariance of halo counts
cov_NP = counts_covariance.NP_covariance(cs2, R, alpha, use_SSC = True, use_exclusion␣
↪= True)

# Define means of mass bins
mass_mean = 0.5*(mass_bins[1:] + mass_bins[:-1])

# Plot the output
plt.figure(figsize = (8,6))
for i in range(len(mass_mean)):
    plt.plot(k, k * cov_NP[i], label = "%.1f"%(np.log10(mass_mean[i])))
plt.ylabel(r'$k \times \mathrm{cov}\left[N(m), P(k)\right]$', fontsize = 15)
plt.xlabel(r'$k$ [$h\,\mathrm{Mpc}^{-1}$]', fontsize = 15)
plt.legend(fontsize = 13)
plt.xlim([0,1])
plt.title('Halo Count and Matter Power Spectrum Cross-Covariance at $z = %s$'%z,␣
↪fontsize = 14);
```

```
Computing intrinsic covariance terms
Computing one-loop power spectrum
Non-linear BAO damping scale = 5.47 Mpc/h
Computing one-loop power spectrum
Computing mass integrals
Constructing output covariance
Computing exclusion covariance terms
Computing mass integrals
Computing interpolation grid for non-linear S function
Computing interpolation grid for linear S function
Computing interpolation grid for P * Theta convolution
```

```
/home/ophilcox/.conda/envs/ptenv/lib/python2.7/site-packages/mcfit/mcfit.py:125:␣
↪UserWarning: The default value of lowring has been changed to False, set it to True␣
↪if you cannot reproduce previous results
  warnings.warn("The default value of lowring has been changed to False, "
/home/ophilcox/.conda/envs/ptenv/lib/python2.7/site-packages/mcfit/mcfit.py:219:␣
↪UserWarning: The default value of extrap has been changed to False, set it to True␣
↪if you cannot reproduce previous results
  warnings.warn("The default value of extrap has been changed to False, "
```

```
Constructing output covariance
Computing super-sample covariance terms
Computing power spectrum response
Note: Variance of the linear density field sigma^2(V) not provided. This will be␣
↪computed assuming the survey volume is isotropic.
```



This completes the tutorial!

Python Module Index

e

Index

L

M

N

S