cluster-lensing: A Python PACKAGE FOR GALAXY CLUSTERS AND MISCENTERING

Jes Ford[1], Jake VanderPlas[1]
Draft version April 20, 2016

ABSTRACT

We describe a new open source package for calculating properties of galaxy clusters, including NFW halo profiles with and without the effects of cluster miscentering. This pure-Python package, cluster-lensing, provides well-documented and easy-to-use classes and functions for calculating cluster scaling relations, including mass-richness and mass-concentration relations from the literature, as well as the surface mass density $\Sigma(R)$ and differential surface mass density $\Delta\Sigma(R)$ profiles, probed by weak lensing magnification and shear, respectively. Galaxy cluster miscentering is especially a concern for stacked weak lensing shear studies of galaxy clusters, where offsets between the assumed and the true underlying matter distribution This software has been developed and released in a public GitHub repository, and is licensed under the permissive free MIT license. The cluster-lensing package can be downloaded through the Python Package Index, https://pypi.python.org/pypi/cluster-lensing, or directly from GitHub, at https://github.com/jesford/cluster-lensing. Full documentation is available at http://jesford.github.io/cluster-lensing/.

Subject headings: methods: data analysis – methods: numerical – galaxies: clusters: general – gravitational lensing: weak – dark matter

1. INTRODUCTION

Clusters of galaxies are the largest gravitationally collapsed structures to have formed in the history of the universe. As such, they are interesting both from a cosmological as well as an astrophysics perspective. In the former case, the galaxy cluster number density as a function of mass (the cluster mass function) is a probe of cosmological parameters including the fractional matter density Ω_m and the normalization of the matter power spectrum σ_8. Astrophysically, the deep potential wells of galaxy clusters are environments useful for testing theories of general relativity, galaxy evolution, and gas and plasma physics, among other things (Voit 2005).

The common thread among these diverse investigations is the requisite knowledge of the mass of the galaxy cluster, which is largely composed of its invisible dark matter halo. Although many techniques exist for estimated the total mass of these systems, weak lensing has emerged as somewhat of a gold standard, since it is sensitive to the mass itself, and not to the dynamical state or other biased tracers of the underlying mass. Scaling relations between weak lensing derived masses, and other observables, including richness, X-ray luminosity and temperature, for examples, are typically calibrated from large surveys and extrapolated to clusters for which gravitational lensing measurements are impossible or unreliable. Since weak lensing masses are often considered the "true" masses, against which other estimates are compared (e.g. Leauthaud et al. 2010; von der Linden et al. 2014; Hoekstra et al. 2015), it is paramount that cluster masses from weak lensing modeling are as unbiased as possible.

For stacked weak lensing measurements of galaxy clusters, an important source of bias in fitting a mass model is the inclusion of the effect of miscentering offsets. Miscentering occurs when the center of the mass distribution, the dark matter halo, does not perfectly coincide with the assumed center around which tangential shear (or magnification) profiles are being measured. Candidate centers for galaxy clusters are necessarily chosen from observational proxies, and often include a single galaxy, such as the brightest or most massive cluster galaxy, or the centroid of some extended quantity like the peak of X-ray emission or average of galaxy positions (George et al. 2012). The particular choice of center may be offset from the true center due to interesting physical processes such as recent mergers and cluster evolution, or simply due to misidentification of the proxy of interest (Johnston et al. 2007).

The miscentering effect on the stacked weak lensing profile can be included in a proper modeling of the measurement, as done in Johnston et al. (2007); Mandelbaum et al. (2010); Oguri & Takada (2011); George et al. (2012); Sehgal et al. (2013); Oguri (2014); Ford et al. (2014, 2015); Simet et al. (2016). The inclusion of this effect commonly assumes a form for the distribution of offsets, such as a Rayleigh distribution in radius (which represents a 2D Gaussian in the plane of the sky), which is convolved with the centered profile. Software for performing the integrations for the miscentered weak lensing profiles was developed in order to produce results in Ford et al. (2014, 2015), and has recently been publicly released to the astronomical community.[2]

When many different gravitational lenses are stacked, as is usually necessary to increase signal-to-noise for weak lensing measurements, care must be taken in the interpretation of the average signal. The issue here is that the (differential) surface mass density is not a linear function of the mass, so the average of many stacked profiles does not directly yield the average mass of the lens sample. Care must be taken to consider the underlying distribution of cluster masses as well as the redshifts of lenses and sources, all of which affect the amplitude of the measured lensing profile. One approach to this is to use a so-called

[1] eScience Institute, University of Washington
contact: jesford@uw.edu

[2] https://github.com/jesford/cluster-lensing

composite-halo approach (e.g. Hildebrandt et al. 2011; Ford et al. 2012, 2014, 2015; Simet et al. 2016), where profiles are calculated for all individual lens objects and then averaged together to create a model that can be fit to the measurement. The `ClusterEnsemble()` class discussed in Section 2.3 is designed with this approach in mind.

- Background about clusters and weak lensing.
- NFW halos (Navarro et al. 1997; Wright & Brainerd 2000)
- composite-NFW fits for weak lensing (Ford et al. 2012, 2014, 2015)
- What is new = miscentering (Johnston et al. 2007; George et al. 2012; Ford et al. 2014, 2015)

2. DESCRIPTION OF THE CODE

In this section we describe each of the individual modules and a few of the functions available in the `cluster-lensing` package. Much of this content comes directly from pages within the online documentation.[3] Throughout the modules, dimensionful quantities are labelled as such by means of the `astropy.units` package (Astropy Collaboration et al. 2013).

- Purpose and general use.
- Relation to existing code
- `SurfaceMassDensity()` class, generic to all NFW halos
- `ClusterEnsemble()` class
- mass-richness functions
- mass-concentration functions
- We use units from the `astropy.units` package (Astropy Collaboration et al. 2013).

2.1. `nfw.py`

The `nfw.py` module contains a single class called `SurfaceMassDensity()`, which computes the surface mass density $\Sigma(R)$ and the differential surface mass density $\Delta\Sigma(R)$ using the class methods `sigma_nfw()` and `deltasigma_nfw()`, respectively. These profiles are calculated according to the analytical formulas first derived by Wright & Brainerd (2000), assuming the spherical NFW model, and can be applied to any dark matter halo: *this module is not specific to galaxy clusters.*

The 3-dimensional density profile of an NFW halo is given by

$$\rho(r) = \frac{\delta_c \rho_{\rm crit}}{(r/r_s)(1 + r/r_s)^2}, \tag{1}$$

where r_s is the cluster scale radius, δ_c is the characteristic halo overdensity, and $\rho_{\rm crit} = \rho_{\rm crit}(z)$ is the critical energy density of the universe at the lens redshift. These three parameters[4] must be specified when instantiating the class `SurfaceMassDensity()`, via the arguments `rs`, `delta_c`, and `rho_crit`, respectively. The units on `rs` are assumed to be Mpc, `delta_c` is dimensionless, and

---

[3] http://jesford.github.io/cluster-lensing/
[4] or, in the case of calculating multiple NFW halos at once, three array-like objects representing each of these parameters

`rho_crit` is in $M_\odot/{\rm Mpc}/{\rm pc}^2$, although the actual inclusion of the `astropy.units` on these variables is optional. The user will probably also want to choose the radial bins for the calculation, which are specified via the keyword argument `rbins`, in Mpc. The surface mass density is the integral along the line-of-sight of the 3-dimensional density:

$$\Sigma(R) = 2 \int_0^\infty \rho(R, y){\rm d}y. \tag{2}$$

Here R is the projected radial distance (in the plane of the sky).

We can adopt the dimensional radius $x \equiv R/r_s$ and, following from Wright & Brainerd (2000), show that:

$$\Sigma(x) = 2r_s \delta_c \rho_{\rm crit} f(x), \tag{3}$$

where $f(x) =$

$$\begin{cases} \frac{1}{x^2-1}\left(1 - \ln\left[\frac{1}{x} + \sqrt{\frac{1}{x^2}-1}\right]/\sqrt{1-x^2}\right), & \text{for } x < 1; \\ 1/3, & \text{for } x = 1; \\ \frac{1}{x^2-1}\left(1 - \arccos(1/x)/\sqrt{x^2-1}\right), & \text{for } x > 1. \end{cases} \tag{4}$$

The differential surface mass density probed by shear is calculated from the definition

$$\Delta\Sigma(x) \equiv \overline{\Sigma}(< x) - \Sigma(x), \tag{5}$$

where

$$\overline{\Sigma}(< x) = \frac{2}{x^2} \int_0^x \Sigma(x')x'{\rm d}x'. \tag{6}$$

We can rewrite the differential surface mass density in the form in which it is computed in `nfw.py`:

$$\Delta\Sigma(x) = r_s \delta_c \rho_{\rm crit} g(x), \tag{7}$$

where $g(x) =$

$$\begin{cases} \left[\frac{4/x^2 + 2/(x^2-1)}{\sqrt{1-x^2}}\right]\ln\left(\frac{1+\sqrt{(1-x)/(1+x)}}{1-\sqrt{(1-x)/(1+x)}}\right) \\ \quad + \frac{4}{x^2}\ln\frac{x}{2} - \frac{2}{(x^2-1)}, & \text{for } x < 1; \\ (10/3) + 4\ln(1/2), & \text{for } x = 1; \\ \left[\frac{8}{x^2\sqrt{x^2-1}} + \frac{4}{(x^2-1)^{3/2}}\right]\arctan\sqrt{\frac{x-1}{1+x}} \\ \quad + \frac{4}{x^2}\ln\frac{x}{2} - \frac{2}{(x^2-1)}, & \text{for } x > 1. \end{cases} \tag{8}$$

Running `sigma_nfw()` or `deltasigma_nfw()`, with only a specification of halo properties `rs`, `delta_c`, `rho_crit`, and radial bins `rbins`, will lead to the calculation of halo profiles according to Equations 3 and 5 outlined above.

```
from clusterlensing import SurfaceMassDensity
rbins = [0.1, 0.5, 1.0, 2.0, 4.0]  # Mpc
smd = SurfaceMassDensity(rs=[0.1],
                         rho_crit=[0.2],
                         delta_c=[9700.],
                         rbins=rbins)
sigma = smd.sigma_nfw()
# surface mass density with default units
sigma[0]
<Quantity [ 129.33333333, 11.64751032, 3.33992059,
0.89839601, 0.23327149] solMass / pc2>
# surface mass density with no units
sigma[0].value
array([ 129.33333333, 11.64751032, 3.33992059,
0.89839601, 0.23327149])
```

These are the standard centered NFW profiles, under the assumption that the peak of the halo density distribution perfectly coincides with the identified halo center. This may not be a good assumption, however, and the user can instead run these calculations for miscentered halos by specifying the optional input parameter `offsets`. This parameter sets the width of a distribution of centroid offsets, assuming a 2-dimensional Gaussian distribution on the sky. This offset distribution is equivalent to, and implemented in code as, a uniform distribution in angle and a Rayleigh probability distribution in R:

$$P(R_{\rm off}) = \frac{R_{\rm off}}{\sigma_{\rm off}^2} \exp\left[-\frac{1}{2}\left(\frac{R_{\rm off}}{\sigma_{\rm off}}\right)^2 \right]. \qquad (9)$$

The parameter `offsets` is equivalent to $\sigma_{\rm off}$ in this equation.

```
from clusterlensing import SurfaceMassDensity
rbins = [0.1, 0.5, 1.0, 2.0, 4.0]
# single miscentered halo profile
smd = SurfaceMassDensity(rs=[0.1],
                         rho_crit=[0.2],
                         delta_c=[9700.],
                         rbins=rbins,
                         offsets=[0.3])
sigma = smd.sigma_nfw()
sigma[0]
<Quantity [ 38.60655298, 17.57285034, 4.11253461,
0.93809627, 0.23574031] solMass / pc2>
# example calculating multiple profiles
smd = SurfaceMassDensity(rs=[0.1,0.2,0.2],
                         rho_crit=[0.2,0.2,0.2],
                         delta_c=[9700,9700,9000],
                         rbins=rbins,
                         offsets=[0.3,0.3,0.3])
sigma = smd.sigma_nfw()
sigma
<Quantity [[  38.60655298,   17.57285034,
4.11253461,    0.93809627,    0.23574031],
         [ 181.91820855,   92.86651598,
27.34020647,    6.94677803,    1.81488253],
         [ 168.79009041,   86.16480864,
25.36720188,    6.44546415,
1.68391163]] solMass / pc2>
```

The miscentered surface mass density profiles are given by the centered profiles (Equations 3 and 5), convolved with the offset distribution (Equation 9). We follow the offset halo formalism first written down by Yang et al. (2006), and applied to cluster miscentering by, *e.g.* Johnston et al. (2007); George et al. (2012); Ford et al. (2014, 2015); Simet et al. (2016). Specifically, we calculate the offset surface mass density $\Sigma^{\rm off}$ as follows:

$$\Sigma^{\rm off}(R) = \int_0^\infty \Sigma(R|R_{\rm off})\ P(R_{\rm off})\ {\rm d}R_{\rm off} \qquad (10)$$

$$\Sigma(R|R_{\rm off}) = \frac{1}{2\pi}\int_0^{2\pi} \Sigma(r){\rm d}\theta \qquad (11)$$

Here $r = \sqrt{R^2 + R_{\rm off}^2 - 2RR_{\rm off}\cos(\theta)}$ and θ is the azimuthal angle (Yang et al. 2006). The $\Delta\Sigma^{\rm off}$ profile is calculated from $\Sigma^{\rm off}$, in analogy with Equations 5 and 6.

```
from clusterlensing import SurfaceMassDensity
rbins = [0.1, 0.5, 1.0, 2.0, 4.0]
# perfectly centered DeltaSigma profile
smd = SurfaceMassDensity(rs=[0.1],
                         rho_crit=[0.2],
                         delta_c=[9700.],
```

```
                         rbins=rbins)
deltasigma = smd.deltasigma_nfw()
deltasigma[0]
<Quantity [ 108.78445455, 25.47093418, 10.29627483,
3.71631903, 1.23840727] solMass / pc2>
# miscentered DeltaSigma profile
rbins = [0.1, 0.5, 1.0, 2.0, 4.0]
smd = SurfaceMassDensity(rs=[0.1],
                         rho_crit=[0.2],
                         delta_c=[9700.],
                         rbins=rbins,
                         offsets=[0.3])
deltasigma = smd.deltasigma_nfw()
deltasigma[0]
<Quantity [ 0.71370144, 9.35821817, 8.90118561,
3.6475417, 1.23610325] solMass / pc2>
```

2.2. `cofm.py`

The `cofm.py` module currently contains three functions, each of which calculates halo concentration from mass, redshift, and cosmology, according to a prescription given in the literature. These functions are `c_DuttonMaccio()` (for calculations following Dutton & Macciò 2014), `c_Duffy()` (following Duffy et al. 2008), and `c_Prada()` (for Prada et al. 2012). Halo mass-concentration relations are an area of active research, and there have been discrepancies between results from different observations and simulations, and disagreement surrounding the best choice of model (see *e.g.* Dutton & Macciò 2014; Klypin et al. 2016). We do not aim to join this discussion here, but focus on outlining the functionality provided by the **cluster-lensing** package, for calculating these different concentration values.

All three functions require two input parameters (scalars or array-like inputs), which are the halo redshift(s) `z` and the halo mass(es) `m`. Specifically, the latter is according to the M_{200} mass definition, in units of solar masses (optionally with corresponding `astropy.units`). M_{200} is the mass interior to a sphere of radius r_{200}, within which the average density is $200\rho_{\rm crit}(z)$.

The default cosmology used is from the measurements by the Planck Collaboration et al. (2014), which is imported from the module `astropy.cosmology.Planck13`. However, the user can specify alternative cosmological parameters. For calculating concentration according to either the Duffy et al. (2008) or the Dutton & Macciò (2014) prescription, the only cosmological parameter required is the Hubble parameter, which can be passed into `c_Duffy()` or `c_DuttonMaccio()` as the keyword argument `h`. For the Prada et al. (2012) concentration, the user would want to specify `Om_M` and `Om_L` (the fractional energy densities of matter and the cosmological constant) in addition to `h`, in the call to `c_Prada()`.

The `c_DuttonMaccio()` calculation of concentration is done according to the power-law

$$\log_{10} c_{200} = a + b\log_{10}(M_{200}/[10^{12}h^{-1}M_\odot]), \qquad (12)$$

where

$$a = 0.52 + 0.385 \cdot \exp[-0.617\, z^{1.21}], \qquad (13)$$

$$b = -0.101 + 0.206z. \qquad (14)$$

The above three equations map to Equations 7, 11, and 10, respectively in Dutton & Macciò (2014). The values in these expressions were determined from simulations

of halos between $0 < z < 5$, spanning over 5 orders of magnitude in mass, and shown to match observational measurements of low-redshift galaxies and clusters (Dutton & Macciò 2014). This concentration-mass relation is the default one used by the `clusters.py` module

```
from clusterlensing import cofm
# single 10**14 Msun halo at z=1
cofm.c_DuttonMaccio(0.1, 1e14)
array([ 5.13397936])
# example with multiple halos
cofm.c_DuttonMaccio([0.1, 0.5], [1e14, 1e15])
array([ 5.13397936,  3.67907305])
```

The concentration calculation in `c_Duffy()` is

$$c_{200} = A \cdot (M_{200}/M_{\text{pivot}})^B \cdot (1+z)^C, \quad (15)$$

where

$$\{A, B, C\} = \{5.71, -0.084, -0.47\}, \quad (16)$$

$$M_{\text{pivot}} = 2 \times 10^{12} \ h^{-1} M_\odot. \quad (17)$$

Equation 15 in this work maps to Equation 4 in Duffy et al. (2008). The values for A, B, and C can be found in Table 1 of that work, where they are specific to the "full" (relaxed and unrelaxed) sample of simulated NFW halos, spanning the redshift range $0 < z < 2$. M_{pivot} can be found in the caption of Table 1 as well. One caveat with this relation is that the cosmology used in creating the Duffy et al. (2008) simulations was that of the now outdated WMAP5 experiment (Komatsu et al. 2009).

```
from clusterlensing import cofm
# default cosmology, h=0.6777
cofm.c_Duffy([0.1, 0.5], [1e14, 1e15])
array([ 4.06126115,  2.89302767])
# with h=1
cofm.c_Duffy([0.1, 0.5], [1e14, 1e15], h=1)
array([ 3.93068341,  2.80001099])
```

The `c_Prada()` concentration calculation is much more complex, and written in terms of $\sigma(M_{200}, x_{\text{p}})$, the rms fluctuation of the density field. The Prada et al. (2012) halo concentration is given by[5]

$$c_{200} = 2.881 B_0(x_{\text{p}}) \left[\left(\frac{B_1(x_{\text{p}})\sigma(M_{200}, x_{\text{p}})}{1.257} \right)^{1.022} + 1 \right]$$
$$\times \exp\left(\frac{0.06}{[B_1(x_{\text{p}})\sigma(M_{200}, x_{\text{p}})]^2} \right). \quad (18)$$

The cosmology and redshift dependence is encoded by the variable x_{p}, which is

$$x_{\text{p}} = \left(\frac{\Omega_{\Lambda,0}}{\Omega_{\text{m},0}} \right)^{1/3} (1+z)^{-1}. \quad (19)$$

The functions within Equation 18 are as follows:

$$\sigma(M_{200}, x_{\text{p}}) = D(x_{\text{p}}) \frac{16.9 y_{\text{p}}^{0.41}}{1 + 1.102 y_{\text{p}}^{0.2} + 6.22 y_{\text{p}}^{0.333}} \quad (20)$$

$$y_{\text{p}} \equiv \frac{10^{12} h^{-1} M_\odot}{M_{200}} \quad (21)$$

[5] we use the subscript "p" to distinguish some variables in the equations from Prada et al. (2012) from those in the current work

$$D(x_{\text{p}}) = \frac{5}{2} \left(\frac{\Omega_{\text{m},0}}{\Omega_{\Lambda,0}} \right)^{1/3} \frac{\sqrt{1 + x_{\text{p}}^3}}{x_{\text{p}}^{3/2}} \int_0^{x_{\text{p}}} \frac{x^{3/2} \mathrm{d}x}{(1 + x^3)^{3/2}} \quad (22)$$

$$B_0(x_{\text{p}}) = \frac{c_{\min}(x_{\text{p}})}{c_{\min}(1.393)} \quad (23)$$

$$B_1(x_{\text{p}}) = \frac{\sigma_{\min}^{-1}(x_{\text{p}})}{\sigma_{\min}^{-1}(1.393)} \quad (24)$$

$$c_{\min}(x_{\text{p}}) = 3.681 + 1.352 \left[\frac{1}{\pi} \arctan[6.948(x_{\text{p}} - 0.424)] + \frac{1}{2} \right] \quad (25)$$

$$\sigma_{\min}^{-1}(x_{\text{p}}) = 1.047 + 0.599 \left[\frac{1}{\pi} \arctan[7.386(x_{\text{p}} - 0.526)] + \frac{1}{2} \right] \quad (26)$$

In order of appearance above, beginning with our Equation 18, these equations correspond to Equations 14-17, 13, 23a, 23b, 12, 18a, 18b, 19, 20 in Prada et al. (2012). The numerical values in these equations were obtained empirically from the simulations described in that work.

```
from clusterlensing import cofm
cofm.c_Prada([0.1, 0.5], [1e14, 1e15])
array([ 5.06733941,  5.99897362])
cofm.c_Prada([0.1, 0.1, 0.1], [1e13, 1e14, 1e15])
array([ 5.71130928, 5.06733941, 5.30163572])
```

The last code example demonstrates the controversial feature of the Prada et al. (2012) mass-concentration relation – an upturn in concentration values for the highest mass halos – in opposition to the canonical view that higher mass halos have lower concentrations (Navarro et al. 1996, 1997; Jing 2000; Bullock et al. 2001).

2.3. `clusters.py`

The `clusters.py` module is designed to provide a catalog-level tool for calculating, tracking, and updating galaxy cluster properties and profiles, through structuring data from multiple clusters as an updatable Pandas Dataframe, and providing an intelligent interface to the other modules discussed in Sections 2.1 and 2.2. This module contains a single class `ClusterEnsemble()`, as well as three functions, `mass_to_richness()`, `richness_to_mass()`, and `calc_delta_c()`.

The function `calc_delta_c()` takes a single input parameter, the cluster concentration `c200` (*e.g.* as calculated by one of the functions in `cofm.py`), and returns the characteristic halo overdensity:

$$\delta_{\text{c}} = \left(\frac{200}{3} \right) \frac{c_{200}^3}{\ln(1 + c_{200}) - c_{200}/(1 + c_{200})}. \quad (27)$$

Both input and output are dimensionless here. For example, to convert a concentration value of $c_{200} = 5$ to δ_{c}, you could do:

```
from clusterlensing.clusters import calc_delta_c
calc_delta_c(5)
8694.8101906193315
```

The pair of functions `mass_to_richness()` and `richness_to_mass()`, as their names imply, perform conversions between cluster mass and richness. The only required input parameter to `mass_to_richness()` is the `mass`, and likewise the only required input to `richness_to_mass()` is `richness`. The calculations assume a power-law form for the relationship between these variables:

$$M_{200} = M_0 \left(\frac{N_{200}}{20} \right)^\beta. \qquad (28)$$

Here M_0 is the normalization, which defaults to 2.7×10^{13}, but can be changed in the call to either function by setting the `norm` keyword argument. The power-law slope $\beta = 1.4$ by default, but can be set by specifying the optional `slope` input parameter. When these functions are invoked by the `ClusterEnsemble()` class, they are applied to the particular mass definition M_{200}, and assume units of $M_\odot$. However the functions themselves do not assume a mass definition or unit, and can be generalized to any parameter (or type of richness) that has a power-law relationship with mass.

```
from clusterlensing.clusters import \
mass_to_richness, richness_to_mass
richness_to_mass(50)
97382243648736.9
mass_to_richness(97382243648736.9)
50.0
# specify other power-law parameters
richness_to_mass(20, slope=1.5, norm=1e14)
100000000000000.0
```

The `ClusterEnsemble()` class creates, modifies and tracks a Pandas DataFrame containing the properties and attributes of many galaxy clusters at once. When given a new or updated cluster property, it calculates and updates all dependent cluster properties, treating each cluster (row) in the DataFrame as an individual object. This makes it easy to calculate the $\Sigma(R)$ and $\Delta\Sigma(R)$ weak lensing profiles for many different mass clusters at different redshifts, for example. In contrast to using the `SurfaceMassDensity()` class discussed in Section 2.1, the user only needs to specify the cluster redshifts and either of the mass or richness. If richness is supplied, then mass is calculated from it, assuming the form of Equation 28 (which is customizable); if mass is specified instead, than the inverse relation is used to calculate richness. In either case the changes are propagated to any dependent variables:

```
from clusterlensing import ClusterEnsemble
z = [0.1,0.2,0.3]
c = ClusterEnsemble(z)
n200 = [20, 20, 20]
c.n200 = n200
# display cluster dataframe
c.dataframe
        z   n200          m200        r200
c200        delta_c         rs
0  0.1     20  2.700000e+13  0.612222  5.839934
12421.201995  0.104834
1  0.2     20  2.700000e+13  0.591082  5.644512
11480.644557  0.104718
2  0.3     20  2.700000e+13  0.569474  5.442457
10555.781440  0.104636
# specify mass directly
c.m200 = [1e13, 1e14, 1e15]
c.dataframe
        z         n200          m200        r200
c200        delta_c         rs
```

```
0   0.1       9.838141  1.000000e+13   0.439664
6.439529  15599.114356   0.068276
1   0.2      50.956400  1.000000e+14   0.914520
4.979102    8612.362538   0.183672
2   0.3     263.927382  1.000000e+15   1.898248
3.886853    4947.982895   0.488377
```

The above examples also demonstrate that cluster masses are converted to concentrations and to characteristic halo overdensities. This assumes the default mass-concentration relation of the `c_DuttonMaccio()` form, or the user can instead specify another of the relations by setting the keyword `cm="Prada"` or `cm="Duffy"`, when the `ClusterEnsemble()` object is instantiated. Cosmology can also be specified upon instantiation, by setting the `cosmology` keyword to any of `"WMAP9"`, `"WMAP7"`, or `"WMAP5"`:

```
c = ClusterEnsemble(z, cm="Duffy",
                cosmology="WMAP5")
c.n200 = [20, 30, 40]
c.dataframe
        z   n200          m200        r200
c200        delta_c         rs
0   0.1     20  2.700000e+13   0.599910  4.520029
6920.955951   0.132723
1   0.2     30  4.763120e+13   0.702040  4.136873
5678.897592   0.169703
2   0.3     40  7.125343e+13   0.775889  3.851601
4849.836498   0.201446
```

Instead of using the `dataframe` attribute, which retrieves the Pandas DataFrame object itself, it might be useful to use the `show()` method, which prints additional information to the screen, including assumptions of the mass-richness relation:

```
c.show()

Cluster Ensemble:

        z   n200          m200        r200
c200        delta_c         rs

0   0.1     20  2.700000e+13   0.599910  4.520029
6920.955951   0.132723
1   0.2     30  4.763120e+13   0.702040  4.136873
5678.897592   0.169703
2   0.3     40  7.125343e+13   0.775889  3.851601
4849.836498   0.201446

Mass-Richness Power Law:
M200 = norm * (N200 / 20) ^ slope
   norm: 2.7e+13 solMass
   slope: 1.4

# alter mass-richness parameters
# and show the updated table
c.massrich_norm = 3e13
c.massrich_slope = 1.5
c.show()

Cluster Ensemble:
        z   n200          m200        r200
c200        delta_c         rs
0   0.1     20  3.000000e+13   0.621353  4.480202
6784.805438   0.138689
1   0.2     30  5.511352e+13   0.737028  4.086481
5526.615129   0.180358
2   0.3     40  8.485281e+13   0.822406  3.795500
4696.109606   0.216679

Mass-Richness Power Law:
M200 = norm * (N200 / 20) ^ slope
   norm: 3e13 solMass
   slope: 1.5
```

This example also demonstrates how the slope or normalization of the mass-richness relation can be altered, and the changes propagate from richness through to mass and other variables.

Then all the ingredients are in place to calculate halo profiles by invoking the `calc_nfw()` method, which interfaces to the `sigma_nfw()` and `deltasigma_nfw()` methods of the `SurfaceMassDensity()` class, and passes it the required inputs $\{r_s, \rho_{crit}, \delta_c\}$ for all the clusters behind the scenes. ρ_{crit} is calculated at every cluster redshift using the (default `astropy.cosmology.Planck13` or user-specified) cosmological model.

```python
import numpy as np
# create some logarithmic bins:
rmin, rmax = 0.1, 5. # Mpc
rbins = np.logspace(np.log10(rmin),
                    np.log10(rmax),
                    num = 8)
# calculate the profiles:
c.calc_nfw(rbins=rbins)
# profiles now exist as attributes:
c.sigma_nfw
<Quantity [[ 128.97156123, 62.58323349,
27.01073105, 10.60607722, 3.88999449,
1.36360964, 0.46464366, 0.15563814],
[ 132.13989867, 64.10484454, 27.66159293,
10.85990257, 3.98265113, 1.39599118,
0.47565695, 0.15932308], [ 135.62272115,
65.782882, 28.38138702, 11.14121765,
4.08549675, 1.43196834, 0.48790043,
0.16342108]] solMass / pc2>
```

```python
c.deltasigma_nfw
<Quantity [[ 105.3190568 , 72.43842908,
43.74538085, 23.44005481, 11.37085955,
5.10385452, 2.16011364, 0.87479771],
[ 107.98098357, 74.25022426, 44.82825347,
24.01505305, 11.64776118, 5.22744541,
2.21219956, 0.89582394], [ 110.88173507,
76.23087398, 46.01581348, 24.64741078,
11.95297965, 5.36391529, 2.26978998,
0.91909571]] solMass / pc2>
```

3. EXAMPLES

Some more detailed examples.

4. FUTURE DEVELOPMENT

Plans for the future.

5. SUMMARY

Summary goes here.

REFERENCES

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33 [2]
Bullock, J. S., Kolatt, T. S., Sigad, Y., et al. 2001, MNRAS, 321, 559 [2.2]
Duffy, A. R., Schaye, J., Kay, S. T., & Dalla Vecchia, C. 2008, MNRAS, 390, L64 [2.2, 2.2]
Dutton, A. A., & Macciò, A. V. 2014, MNRAS, 441, 3359 [2.2, 2.2]
Ford, J., Hildebrandt, H., Van Waerbeke, L., et al. 2014, MNRAS, 439, 3755 [1, 2.1]
—. 2012, ApJ, 754, 143 [1]
Ford, J., Van Waerbeke, L., Milkeraitis, M., et al. 2015, MNRAS, 447, 1304 [1, 2.1]
George, M. R., Leauthaud, A., Bundy, K., et al. 2012, ApJ, 757, 2 [1, 2.1]
Hildebrandt, H., Muzzin, A., Erben, T., et al. 2011, ApJ, 733, L30 [1]
Hoekstra, H., Herbonnet, R., Muzzin, A., et al. 2015, MNRAS, 449, 685 [1]
Jing, Y. P. 2000, ApJ, 535, 30 [2.2]
Johnston, D. E., Sheldon, E. S., Wechsler, R. H., et al. 2007, ArXiv e-prints, astro-ph/0709.1159, arXiv:0709.1159 [1, 2.1]
Klypin, A., Yepes, G., Gottlöber, S., Prada, F., & Heß, S. 2016, MNRAS, 457, 4340 [2.2]
Komatsu, E., Dunkley, J., Nolta, M. R., et al. 2009, ApJS, 180, 330 [2.2]

Leauthaud, A., Finoguenov, A., Kneib, J.-P., et al. 2010, ApJ, 709, 97 [1]
Mandelbaum, R., Seljak, U., Baldauf, T., & Smith, R. E. 2010, MNRAS, 405, 2078 [1]
Navarro, J. F., Frenk, C. S., & White, S. D. M. 1996, ApJ, 462, 563 [2.2]
—. 1997, ApJ, 490, 493 [1, 2.2]
Oguri, M. 2014, MNRAS, 444, 147 [1]
Oguri, M., & Takada, M. 2011, Phys. Rev. D, 83, 023008 [1]
Planck Collaboration, Ade, P. A. R., Aghanim, N., et al. 2014, A&A, 571, A16 [2.2]
Prada, F., Klypin, A. A., Cuesta, A. J., Betancort-Rijo, J. E., & Primack, J. 2012, MNRAS, 423, 3018 [2.2, 2.2, 5, 2.2, 2.2]
Sehgal, N., Addison, G., Battaglia, N., et al. 2013, ApJ, 767, 38 [1]
Simet, M., McClintock, T., Mandelbaum, R., et al. 2016, ArXiv e-prints, arXiv:1603.06953 [1, 2.1]
Voit, G. M. 2005, Reviews of Modern Physics, 77, 207 [1]
von der Linden, A., Allen, M. T., Applegate, D. E., et al. 2014, MNRAS, 439, 2 [1]
Wright, C. O., & Brainerd, T. G. 2000, ApJ, 534, 34 [1, 2.1, 2.1]
Yang, X., Mo, H. J., van den Bosch, F. C., et al. 2006, MNRAS, 373, 1159 [2.1, 2.1]