Draft version May 12, 2016
Preprint typeset using LaTeX style emulateapj v. 01/23/15

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

Jes Ford[1], Jake VanderPlas[1]

Draft version May 12, 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 can lead to a significant bias in the mass estimates if not taken into account. 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 often 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

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

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

sources, all of which affect the amplitude of the measured lensing profile. One approach to this is to use a so-called 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.

A popular model for the dark matter distribution in a gravitationally collapsed halo, such as a galaxy cluster, is the Navarro, Frenk, and White (NFW) model. This density profile (given in Equation 1 below) was determined from numerical simulations, including the dissipationless collapse of density fluctuations under gravity (Navarro et al. 1997). The simpler Singular Isothermal Sphere density model, which only has one free parameter in contrast to the two for NFW, does not tend to fit the inner profiles of halos well and is also unphysical in that the total mass diverges (Schneider 2006). Other models such as the generalized-NFW and the Einasto profile tend to better describe the full radial distribution of dark matter in halos, at the expense of adding a third parameter to characterize the inner slope of the density profiles (see *e*.g. discussion in Dutton & Macciò 2014). In the software package presented in this work we only include the standard 2-parameter NFW model, but future work should make alternative models available as well.

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).

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

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

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 `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:

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

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:

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

where $f(x) =$

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

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

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

where

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

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

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

where $g(x) =$

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

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])
```

[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

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:

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

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:

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

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

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 assumed to correspond to the M_{200} mass definition, in units of solar masses. 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

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

where

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

$$b = -0.101 + 0.206z. \tag{14}$$

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

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

where

$$\{A, B, C\} = \{5.71, -0.084, -0.47\}, \tag{16}$$

$$M_{\rm pivot} = 2 \times 10^{12}\ h^{-1} M_{\odot}. \tag{17}$$

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 $0 < z < 2$. $M_{\rm pivot}$ 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 $\sigma(M_{200}, x_{\rm p})$, the rms fluctuation of the density field. The Prada et al. (2012) halo concentration is given by[5]

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

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

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

The functions within Equation 18 are as follows:

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

$$y_{\rm p} \equiv \frac{10^{12} h^{-1} M_{\odot}}{M_{200}} \tag{21}$$

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

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

$$B_0(x_{\rm p}) = \frac{c_{\rm min}(x_{\rm p})}{c_{\rm min}(1.393)} \tag{23}$$

$$B_1(x_{\rm p}) = \frac{\sigma^{-1}_{\rm min}(x_{\rm p})}{\sigma^{-1}_{\rm min}(1.393)} \tag{24}$$

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

$$\sigma^{-1}_{\rm min}(x_{\rm p}) = 1.047 + 0.599 \left[\frac{1}{\pi} \arctan[7.386(x_{\rm p} - 0.526)] + \frac{1}{2} \right] \tag{26}$$

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:

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

Both input and output are dimensionless here. For example, to convert a concentration value of $c_{200} = 5$ to $\delta_{\rm c}$, 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:

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

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 *(update this to new cosmology usage when next code release is made)* 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

# update the mass-richness parameters
# and show the resulting 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: 3e+13 solMass
   slope: 1.5
```

The last 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 $\{r_\mathrm{s}, \rho_\mathrm{crit}, \delta_\mathrm{c}\}$ 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. The user must specify the desired radial bins `rbins` in Mpc.

```
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>
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>
```

Similar to the direct use of `SurfaceMassDensity()`, discussed in Section 2.1, the miscentered profiles can be calculated from the `calc_nfw()` method, by supplying the optional `offsets` keyword with an array-like object of length equal to the number of clusters, where each element is the width of the offset distribution in Mpc (σ_off in Equation 9).

```
c.calc_nfw(rbins=rbins, offsets=[0.3,0.3,0.3])
# the offset sigma profile is now:
c.sigma_nfw
<Quantity [[ 42.50844685, 39.74291121,
32.29894213, 18.50988719, 6.16284894,
1.89335218, 0.62609991, 0.20840423],
[ 68.10228964, 63.87901872, 52.56539317,
31.20890672, 11.17821854, 3.5884285,
1.20745376, 0.40574057], [ 95.16077234,
89.48298631, 74.29328561, 45.24074628,
17.06333763, 5.66481165, 1.93408383,
0.65518747]] solMass / pc2>
```

Although `SurfaceMassDensity()` from the `nfw.py` module, and `ClusterEnsemble().calc_nfw()` from the `clusters.py` module, are both capable of computing the same $\Sigma(R)$ and $\Delta\Sigma(R)$ profiles, each require different forms of input which would make sense for different use cases. For the studies in Ford et al. (2015), Ford et al. (2014), and Ford et al. (2012), the authors wanted do the profile computations for many clusters at once, while varying the mass and the miscentering offset distribution during the process of fitting the model to the data. What was known were the redshifts and mass proxies (cluster richness in Ford et al. 2015 and Ford et al. 2014, and a previous mass estimate in Ford et al. 2012), and mass-concentration relations from the literature, so the `ClusterEnsemble()` framework made sense. However, if someone wanted to simply calculate the NFW profiles according to the Wright & Brainerd (2000) formulation, then they might prefer to use `SurfaceMassDensity()` as a tool to get profiles directly from the NFW and cosmological parameters r_s, δ_c, and $\rho_\mathrm{crit}(z)$.

3. EXAMPLE

As an example use case, we take the Canada-France-Hawaii Telescope Lensing Survey (CFHTLenS; Heymans et al. 2012; Erben et al. 2013) public galaxy cluster catalog, which is available on Zenodo at http://dx.doi.org/10.5281/zenodo.51291 (Ford 2014). This dataset was previously explored using a pre-release version of the `cluster-lensing` software in Ford et al. (2014, 2015). The W1 field of this survey contains 10,745 galaxy cluster candidates in the redshift range $0.2 \leq z \leq 0.9$:

```
import numpy as np
data = np.loadtxt('Clusters_W1.dat')
data.shape
(10745, 5)
data[0:4, :]  # print first 4 clusters
array([[ 34.8023, -7.01005, 0.3, 4.435, 10.],
       [ 34.9425, -7.38996, 0.5, 4.545, 21.],
       [ 34.8651, -6.69449, 0.5, 3.858, 6.],
       [ 34.6224, -7.32768, 0.5, 3.619, 8.]])
redshift = data[:, 2]
sig = data[:, 3]
richness = data[:, 4]
```

We select a subset of the lower redshift clusters that were detected at the highest significance. Then we import `clusterlensing` to create a dataframe of the cluster properties, of which we just print the first several, and calculate the NFW profiles.

```
# select a subset
here = (sig > 15) & (redshift < 0.5)
sig[here].shape
z = redshift[here]
n200 = richness[here]

import clusterlensing
c = clusterlensing.ClusterEnsemble(z)
c.n200 = n200
c.dataframe.head()
     z  n200          m200      r200
c200      delta_c        rs
0  0.4   181  5.897552e+14  1.531367  3.966101
5173.016417  0.386114
1  0.3   420  1.916332e+15  2.357815  3.658237
4332.615805  0.644522
2  0.4   176  5.670737e+14  1.511478  3.980218
5213.746469  0.379747
3  0.3   113  3.049521e+14  1.277703  4.341779
6324.420397  0.294281
4  0.4   162  5.049435e+14  1.454129  4.022285
5336.272412  0.361518

rbins = np.logspace(np.log10(0.1),
                    np.log10(10.0), num=20)
c.calc_nfw(rbins)
```

Next we import the `matplotlib` and `seaborn` libraries and configure some settings to make our plots more readable. The first plot we create with the commands below presents the $\Sigma(R)$ profile for every one of these 15 clusters, and is given in Figure 1.

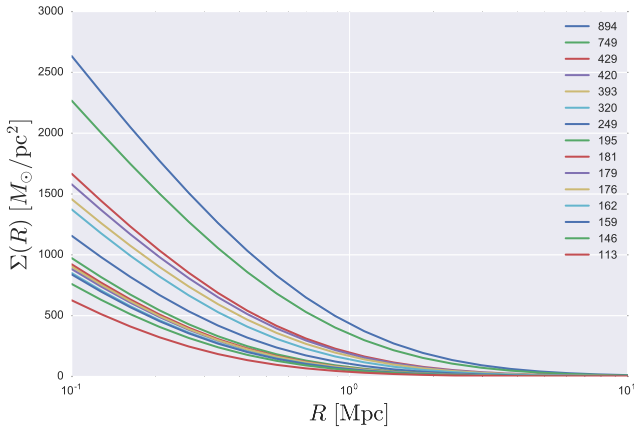

FIG. 1.— Surface mass density profiles $\Sigma(R)$ for all 15 clusters used in this example. These are the most significant (S/N > 15) clusters detected at low redshifts ($z < 0.5$) in the W1 field of CFHTLenS. See the text for links to download this public dataset. The legend gives the richness values estimated in Ford et al. (2015) corresponding to each of these clusters, which are assumed to scale with mass. They are listed from highest to lowest richness, in the same order as the curves.

```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import matplotlib
matplotlib.rcParams['axes.labelsize'] = 20
matplotlib.rcParams['legend.fontsize'] = 20

# strings for plots
raxis = '$R\ [\mathrm{Mpc}]$'
sgma = '$\Sigma(R)$'
sgmaoff = '$\Sigma^\mathrm{off}(R)$'
delta = '$\Delta$'
sgmaunits = ' $[M_{\odot} / \mathrm{pc}^2]$'

# order from high to low richness
order = c.n200.argsort()[::-1]

for s, n in zip(c.sigma_nfw[order], c.n200[order]):
    plt.plot(rbins, s, label=str(int(n)))
plt.xscale('log')
plt.legend(fontsize=10)
plt.ylabel(sgma+sgmaunits)
plt.xlabel(raxis)
plt.tight_layout()
plt.savefig('f1.eps')
plt.close()  # output is Figure 1
```

If we had made a stacked measurement of the shear or magnification profile of these clusters, then we will want to know what the average profile of the stack looks like. Since we already have the individual profiles, we just need to calculate the mean across the 0^{th} axis of the `sigma_nfw` and `deltasigma_nfw` arrays. The plot of these average profiles is given in Figure 2.

```
sigma = c.sigma_nfw.mean(axis=0)
dsigma = c.deltasigma_nfw.mean(axis=0)

plt.plot(rbins, sigma, label=sgma)
plt.plot(rbins, dsigma, '--', label=delta+sgma)
plt.legend()
plt.ylim([0., 1400.])
plt.xscale('log')
plt.xlabel(raxis)
plt.ylabel(sgmaunits)
plt.tight_layout()
plt.savefig('f2.eps')
plt.close()  # output is Figure 2
```

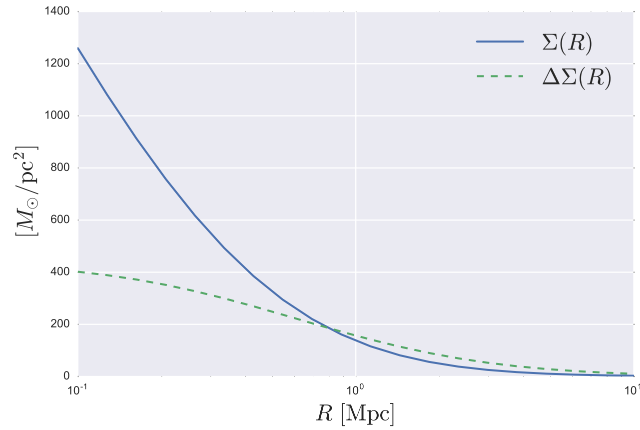

FIG. 2.— The average of all the surface mass density profiles $\Sigma(R)$ for each of the clusters shown in Figure 1 is given in blue. The green curve is the average of all the individual differential mass density profiles $\Delta\Sigma(R)$.

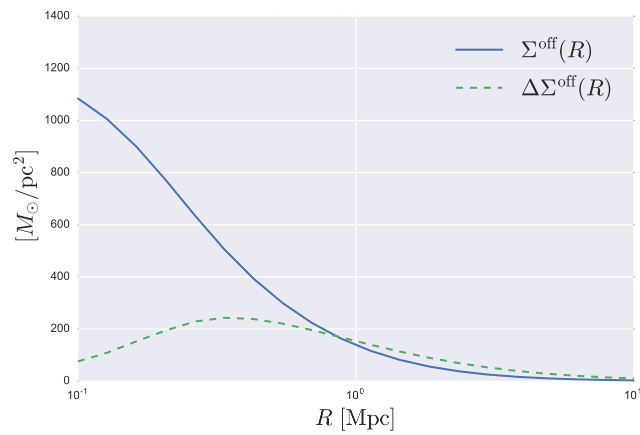

FIG. 3.— Same as Figure 2 except for miscentered profiles. Cluster centroid offsets are assumed to follow a Rayleigh probability distribution, which is convolved with the perfectly centered profiles to achieve this result.

Finally, we may want to investigate whether cluster miscentering has a significant effect on our sample. We calculate the miscentered profiles, given in Figure 3, which could be compared to the centered profiles in Figure 2 to see which is a better fit to our measurement.

```
offsets = np.ones(c.z.shape[0]) * 0.1
c.calc_nfw(rbins, offsets=offsets)
sigma_offset = c.sigma_nfw.mean(axis=0)
dsigma_offset = c.deltasigma_nfw.mean(axis=0)

plt.plot(rbins, sigma_offset, label=sgmaoff)
plt.plot(rbins, dsigma_offset, '--',
         label=delta+sgmaoff)
plt.legend()
plt.ylim([0., 1400.])
plt.xscale('log')
plt.xlabel(raxis)
plt.ylabel(sgmaunits)
plt.tight_layout()
plt.savefig('f3.eps')
plt.close()  # output is Figure 3
```

The above example shows a simple application of `cluster-lensing` to a real dataset – a subset of the CFHTLenS cluster catalog. For this example we kept the customizations to a minimum, but as shown in Sections 2.3, the user can alter the parameters in the power-law relation used to convert richness to mass, choose the form of the mass-concentration relation assumed for the NFW profiles, and specify a particular background cosmology. When fitting a model produced by `cluster-lensing` to a measurement, one could iterate through parameters in this space by setting various attributes of the `ClusterEnsemble()` object (as done, *e.g.* in Ford et al. 2014, 2015).

4. RELATION TO EXISTING CODE

The `cluster-lensing` project offers some unique capabilities over other publicly-available software, most notably the cluster miscentering calculations. Here we attempt to compare the software presented in this work with other open source tools that we are aware of, and show how `cluster-lensing` fits into the larger ecosystem of astronomical software.

`Colossus` is a Python package aimed at cosmology, halo, and large-scale structure calculations (Diemer 2015). It was used in work by Diemer & Kravtsov (2015) and is made available at http://www.benediktdiemer.com/code/ under the MIT license. Much of the functionality of `cluster-lensing` appears to overlap with `Colossus`, including mass-concentration relations (although `Colossus` has the advantage of containing many more relations from the literature) and NFW surface mass density profiles. However, `cluster-lensing` also provides the miscentered halo calculations, which are are lacking from `Colossus`.

While Diemer (2015) has chosen to implement basic cosmological calculations from scratch, `cluster-lensing` instead relies on external modules supplied by `astropy`. The only dependencies claimed by `Colossus` are `numpy` (Walt et al. 2011) and `scipy` (Jones et al. 2001–), whereas `cluster-lensing` additionally requires `astropy` (Astropy Collaboration et al. 2013) and `pandas` (McKinney 2010). Fewer dependencies might be seen as a positive feature of `Colossus`; on the other hand, `astropy` could be viewed as possibly a more robust source for standard astronomical and cosmological calculations, since it is maintained by a large community of developers.

Another related set of code is provided by Jörg Dietrich's NFW routines, archived on Zenodo (Dietrich 2016), and available on GitHub at https://github.com/joergdietrich/NFW. These `Python` modules calculate NFW profiles for $\Sigma(R)$ and $\Delta\Sigma(R)$, as well as the 3-dimensional density profiles and total mass and projected mass inside a given radius. `cluster-lensing` goes beyond the functionality of Dietrich (2016) by supplying means for calculating cluster miscentering, and having a built-in framework for handling many halos at once. For Dietrich (2016), the user must provide the halo concentration (along with mass and redshift) to the `NFW()` class, but additional routines are available for converting mass to concentration, including Duffy et al. (2008) and another by Dolag et al. (2004) (a partial overlap with the mass-concentration relations provided by `cluster-lensing`). Dietrich (2016) depends on `astropy` for cosmological calculations and units, similar to `cluster-lensing`, as well as the `numpy` and `scipy` packages.

5. FUTURE DEVELOPMENT

Some of the future plans for `cluster-lensing` include adding support for different density profiles. Currently only the NFW model is provided, and alternative mass density models would make the package more complete and useful. The first priority will be inclusion of the Einasto profile (Einasto 1965), and later possibilities may include the generalized-NFW (Zhao 1996). The default cosmology is currently that of Planck Collaboration et al. (2014), but should be updated to Planck Collaboration et al. (2015), since this is now available as `astropy.cosmology.Planck15` (the user can currently specify this cosmology, it is just not the default).

When surface mass density profiles have to be calculated many times for many clusters, as is the case when iterating over parameters in the process of fitting a model, the processing time can become lengthy. This issue is most pronounced for calculation of miscentered profiles, which require the convolution laid out in Equations 10 and 11. One major improvement to `cluster-lensing` will be the option to use parallel-processing in these computations. The likely structure of this parallelism will be to divide the halos in a `ClusterEnsemble()` catalog object among the parallel threads, which will calculate the profiles for each of their assigned clusters.

All of these future developments are currently listed as issues at https://github.com/jesford/cluster-lensing/issues. This GitHub Issue tracker will continue to serve as the central place for listing future improvements and feature requests. Users and potential-users alike are encouraged to submit ideas and requests through that URL.

6. SUMMARY

In this work we presented `cluster-lensing`, a pure-`Python` package for calculating galaxy cluster profiles and properties. We described and gave worked examples of all the functionality currently available, including mass-concentration and mass-richness scaling relations, and the surface mass density profiles $\Sigma(R)$ and $\Delta\Sigma(R)$, which are relevant for gravitational lensing. The latter density profiles are not cluster-specific, but apply to any mass halo that can be approximated by the NFW prescription. The structure of `cluster-lensing` is ideal for calculating properties and profiles for many galaxy clusters at once. This "composite-halo" approach (see *e.g.* Ford et al. 2015), is especially useful for fitting models to a stacked sample of clusters that span a range of mass and/or redshift.

Compared to existing code, `cluster-lensing` stands out by seemingly being the only publicly-available software for calculating miscentered halo profiles. Miscentering is a problem of great relevance for stacked weak lensing studies of galaxy clusters, where halo centers are imperfectly estimated from observational data or simply not well defined (as is the case for individual non-spherical halos – for example in merging systems). The resulting offsets between the assumed and real centers change the shape of the measured shear or magnification profile and need to be accounted for in the modeling.

`cluster-lensing` is released under the MIT license, and can be downloaded from the Python Package

Index, by running `$ pip install cluster-lensing` on the command line, or by manually downloading the package from https://pypi.python.org/pypi/cluster-lensing. It being developed in a public repository on GitHub, http://github.com/jesford/cluster-lensing/. Much of the content of this paper is also available in the online documentation, available at http://jesford.github.io/cluster-lensing/. We welcome feedback, suggestions, and feature requests through GitHub issues (https://github.com/jesford/cluster-lensing/issues) or by emailing the author. Contributions to the code can be made by submitting a pull request to the repository (https://github.com/jesford/cluster-lensing/pulls). If `cluster-lensing` is used in a research project, the authors would appreciate citations to the code, *Zenodo citation*, and this paper.

ACKNOWLEDGEMENTS

The authors are grateful for funding from the Washington Research Foundation Fund for Innovation in Data-Intensive Discovery and the Moore/Sloan Data Science Environments Project at the University of Washington. This project makes use of Astropy, a community-developed core Python package for Astronomy (Astropy Collaboration, 2013), http://www.astropy.org.

REFERENCES

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33 [2, 4]
Bullock, J. S., Kolatt, T. S., Sigad, Y., et al. 2001, MNRAS, 321, 559 [2.2]
Diemer, B. 2015, Colossus: COsmology, haLO, and large-Scale StrUcture toolS, Astrophysics Source Code Library, ascl:1501.016 [4]
Diemer, B., & Kravtsov, A. V. 2015, ApJ, 799, 108 [4]
Dietrich, J. 2016, NFW: NFW v0.1, Zenodo, doi:10.5281/zenodo.50664 [4]
Dolag, K., Bartelmann, M., Perrotta, F., et al. 2004, A&A, 416, 853 [4]
Duffy, A. R., Schaye, J., Kay, S. T., & Dalla Vecchia, C. 2008, MNRAS, 390, L64 [2.2, 2.2, 4]
Dutton, A. A., & Macciò, A. V. 2014, MNRAS, 441, 3359 [1, 2.2, 2.2]
Einasto, J. 1965, Trudy Astrofizicheskogo Instituta Alma-Ata, 5, 87 [5]
Erben, T., Hildebrandt, H., Miller, L., et al. 2013, MNRAS, 433, 2545 [3]
Ford, J. 2014, CFHTLenS 3D-MF Galaxy Cluster Catalog, Zenodo, This catalog was made public as part of work by the CFHTLenS collaboration., doi:10.5281/zenodo.51291 [3]
Ford, J., Hildebrandt, H., Van Waerbeke, L., et al. 2014, MNRAS, 439, 3755 [1, 2.1, 2.3, 3, 3]
—. 2012, ApJ, 754, 143 [1, 2.3]
Ford, J., Van Waerbeke, L., Milkeraitis, M., et al. 2015, MNRAS, 447, 1304 [1, 2.1, 2.3, 3, 1, 3, 6]
George, M. R., Leauthaud, A., Bundy, K., et al. 2012, ApJ, 757, 2 [1, 2.1]
Heymans, C., Van Waerbeke, L., Miller, L., et al. 2012, MNRAS, 427, 146 [3]
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]
Jones, E., Oliphant, T., Peterson, P., et al. 2001–, SciPy: Open source scientific tools for Python, http://www.scipy.org/, [Online; accessed 2016-04-27] [4]
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]
McKinney, W. 2010, in Proceedings of the 9th Python in Science Conference, ed. S. van der Walt & J. Millman, 51 – 56 [4]
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, 5]
—. 2015, ArXiv e-prints, arXiv:1502.01589 [5]
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]
Schneider, P. 2006, in Saas-Fee Advanced Course 33: Gravitational Lensing: Strong, Weak and Micro, ed. G. Meylan, P. Jetzer, P. North, P. Schneider, C. S. Kochanek, & J. Wambsganss, 269–451 [1]
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]
Walt, S. v. d., Colbert, S. C., & Varoquaux, G. 2011, Computing in Science and Engg., 13, 22 [4]
Wright, C. O., & Brainerd, T. G. 2000, ApJ, 534, 34 [2.1, 2.1, 2.3]
Yang, X., Mo, H. J., van den Bosch, F. C., et al. 2006, MNRAS, 373, 1159 [2.1, 2.1]
Zhao, H. 1996, MNRAS, 278, 488 [5]