Density Profiles of Dark Matter Halos

Katie Robbins

Undergraduate Senior Honors Thesis
Spring 2010

Thesis Advisor: Andreas Berlind

Abstract:
Galaxies reside within dark matter halos; therefore, it is essential to study the internal properties of dark matter halos to better understand galaxy formation and evolution. We examine the density profiles of dark matter halos by analyzing data from the LasDamas (LArge Suite of DArk MAtter Simulations) project. We measure density profiles for over 3.3 million halos, and fit Navarro, Frenk & White (1996, 1997) profiles (NFW) to our measurements. With such a large dataset, we are able to study the full density profile distribution and its dependence on mass, even in the regime of rare cluster-sized halos. We also consider the effects of halo shape and goodness-of-fit. Finally, we investigate the sensitivity of our results to particle mass resolution and choice of binning. We find that the relation between halo mass and NFW concentration found by Bullock et al. (2001), $c \propto M^{-0.13}$, holds out to halo masses of $10^{15.5}$ $M_{\odot}$, a previously untested mass region. However, this relation does not hold in general for all subsets of halos. We also find that most of our halos are not statistically well fit by the NFW profile.

Contents

1 INTRODUCTION

Most of the matter in our universe is dark, interacting gravitationally but not electromagnetically. Dark matter was first theorized by Fritz Zwicky in the 1930s as a way of explaining the "missing mass" needed to support the orbital velocities of galaxies in the Coma cluster. Today, dark matter is a widely accepted, although not so well understood, major component of the universe. Indirect evidence for dark matter is provided by observations of the rotational speeds of galaxies, gravitational lensing of galaxy clusters, such as the Bullet Cluster, and other dynamical measures of galaxy masses.

Early in the history of the universe the distribution of matter was mostly uniform. However, as time progressed, the universe expanded and cooled. Small fluctuations in the distribution of matter led to gravitational collapse, forming clumps of dark matter called dark matter halos. Baryonic matter, because of gravitational forces, followed the dominant dark matter, and galaxies were formed inside these halos. Thus, dark matter halos provide the foundation for galaxy formation, and it is essential to study their internal properties to better understand galaxy evolution. In particular, properties of halos are used to model galaxy clustering, which can be compared to, and used to interpret clustering measurements from galaxy surveys.

One important internal property of a dark matter halo is its density profile. Knowledge of the distribution of matter within a halo is crucial to the understanding of the observed interactions of galaxies with their underlying dark matter halos. Also, experiments are in place to detect theoretical dark matter particles called WIMPs, Weakly Interacting Massive Particles. The expected rates for both direct detection and indirect detection, by finding neutrinos from WIMP annihilation, depend on the density profile. Navarro, Frenk and White (1996,1997) found a universal profile to describe the density profile of dark matter halos which has only two parameters, concentration and halo mass, where concentration is a measure of much how mass is packed toward the center of a halo. This form is widely used to model halos and is referred to as the NFW profile. A correlation between mass and concentration was found by Bullock et al. (2001) in which higher mass halos tend to have lower concentrations according to the relation: $c \propto M^{-0.13}$. This relation is used in modeling; however, their sample of ~5,000 halos was fairly small and did not include halos with masses above 10^{14} $M_\odot$.

One of our primary goals is to test the universality of the NFW profile and the relation found by Bullock et al. (2001). We seek to test relation for higher mass halos, and also for subsets of halos. With access to a much larger dataset, we are able to measure and fit the profiles of more than 3.3 million dark matter halos with masses up to $10^{15.5}$ $M_\odot$.

2 DATA

For this investigation of dark matter halos we consider data from cosmological N-body simulations. These simulations contain only particles which represent the total dark matter in a portion of the universe. The particles are collisionless and interact solely through Newtonian gravity. All the data considered is for halos at redshift z=0.

2.1 LasDamas

Our halos come from the LasDamas (LArge Suite of DArk MAtter Simulations) project. The LasDamas project simulates the evolution of dark matter in the universe with a suite of cosmological N-body realizations to enable statistical studies of galaxies and halos. The advantage of LasDamas is its large size; in our study of dark matter halos we consider 40 realizations with over 3.3 million halos. These halos range in mass from just under 10^{13} $M_\odot$ to $10^{15.5}$ $M_\odot$. With such a large dataset we are able to study the full density profile distribution and its dependence on mass, even in the regime of rare cluster-sized halos.

LasDamas will produce 200 simulations, 50 realizations in 4 different boxsizes. Each configuration is named, in order of increasing boxsize (and corresponding particle mass): Consuelo, Esmeralda, Carmen, and Oriana. Each realization has the same initial power spectrum but a different random seed. Details about the different boxes are listed in Table 1. For this investigation, data is taken from 40 Esmeralda boxes. We chose to use Esmeralda because it has the highest resolution, with the most completed realizations. Data is also used from one Carmen realization as a resolution check. Every simulation uses the same cosmological parameters, which are consistent with WMAP5 (Komatsu et al. 2009). Details about the cosmological model are listed in Table 2.

Name	Boxsize [Mpc/h]	Number of Particles	Particle Mass [$M_\odot$/h]	softening [kpc/h]
Oriana	2400	1280^3	45.73×10^{10}	53
Carmen	1000	1120^3	4.938×10^{10}	25
Esmeralda	640	1250^3	0.931×10^{10}	15
Consuelo	420	1400^3	0.187×10^{10}	8

Table 1. Simulation details for the LasDamas boxes.

Ω_m	Ω_L	Ω_b	$H_o/100$ (h)	σ_8	n_s
0.25	0.75	0.04	0.7	0.8	1.0

Table 2. Cosmological parameters used in the simulations. Ω_m is the dark matter density. Ω_L is the dark energy density. Ω_b is the baryon density. H_o is the Hubble constant. σ_8 is the fluctuation amplitude at 8 Mpc/h. n_s is the scalar spectral index.

2.2 Explanation of softening

It is important to understand the role of softening in the simulations and its effect on resolution. According to gravity, when two point masses get very close together the force between them goes to infinity. Thus, it is necessary to "soften" the potential so that forces do not become unrealistically large. Additionally, even though each box contains billions of particles, each individual particle is quite massive. These massive point particles actually represent masses which are spread over a certain volume. Therefore, simulations with higher mass particles require a larger softening length in order to be physically realistic. Our primary data comes from the Esmeralda boxes which have a softening length of 15 kpc/h. See Table 1 for the softening lengths of each simulation. As a consequence of softening, information on scales smaller than a few times the softening length is not necessarily reliable.

2.3 Friends-of-friends halos

Bound groups of dark matter particles were identified using a parallel friends-of-friends code. The implementation, ntropy-fofsv, was built with the Ntropy framework (Gardner, Connolly, & McBride 2007). The friends-of-friends algorithm (Davis et al. 1985) links together particles whose separation is less than a specified linking length. In this study we consider friends-of-friends halos with a linking length of 0.156 times the mean interparticle separation. For the cosmology in the simulations, it is expected that this value for the linking length will produce roughly virialized halos.

3 HALO DEFINITION

In order to establish a reasonable region for measuring the radial density profiles of the friends-of-friends halos, the following measurements are made: the position of the center, the radial extent, and the virial mass of the halos.

3.1 Defining halo center

Given a collection of points in space there are several ways of defining a center. We decided to use the location of the most bound particle (the particle with the lowest total energy) as the position of the halo center since it seemed to be the most physically relevant. Other choices were also considered, such as the position of the deepest potential particle (the particle with the lowest potential energy), and the location of the center of mass of the system. Fig. 1 illustrates the different types of centers. Generally, there was not a large difference in the positions of the most bound particle and the deepest potential particle. The deepest potential particle was within one softening length of the most bound particle in 49% of all halos, and within 10 softening lengths (150 kpc/h) in 99% of all halos.

3.2 Defining halo extent and halo mass

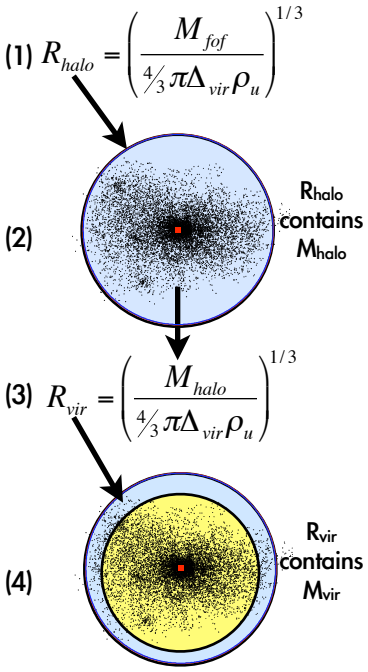
Once the center is defined, the extent of the halo can be determined by estimating the virial radius. Theoretically, the virial radius is defined to be the radius within which the virial theorem holds, that is, within the virial radius the halo is expected to be in virial equilibrium. We choose to find the virial radius in the following systematic way.

1. We define R_{halo} by assuming the total friends-of-friends halo mass is enclosed in a volume of a sphere with a radius R_{halo} and a density Δ_{vir} (377) times the mean density of the universe.

2. We then identify the mass that is actually enclosed in a sphere with radius R_{halo} as M_{halo}.

3. We iterate this process and define R_{vir} similarly by assuming that M_{halo} is enclosed in a volume of a sphere with a radius R_{vir} and a density Δ_{vir} times the mean density of the universe.

4. We define M_{vir} as the mass enclosed in a sphere with radius R_{vir}.



All further measurement and fitting is done by considering only the particles within R_{vir}. Thus, halos have mass M_{vir} and radius R_{vir}.

For the given cosmology of the simulations, an overdensity of Δ_{vir}=377 is expected to correspond to a virialized volume. This iterative procedure defines an acceptable region to measure the radial densities of the halo, and sometimes excludes irregular clumps of particles at the edges of a halo which have tagged along with the main halo. We found that, most of the time, the majority of the mass excluded in this procedure was excluded in the first radial cut. Thus, the second radial cut serves to refine the radius and mass of the halo we wish to fit. Also note that if none of the halo's mass is excluded in the first cut then R_{halo} and R_{vir} will be identical. The illustration to the right shows the steps in finding R_{vir} and M_{vir}. Fig. 1 shows the procedure applied to a halo from the Esmeralda simulation.

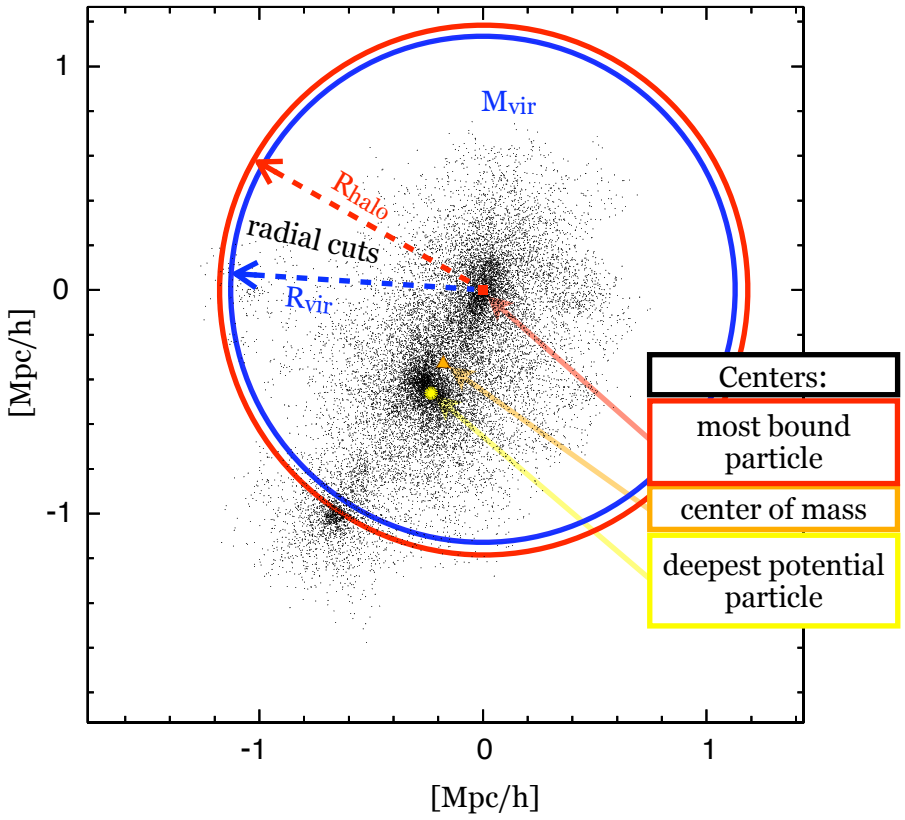


Figure 1. An example friends-of-friends halo illustrating different centers and the method for finding R_{vir} and M_{vir}. Black points show particles of the friends-of-friends halo. The red square, orange triangle, and yellow asterisk respectively indicate the positions of the most bound particle, the center of mass of the system, and the deepest potential particle. The red dashed arrow and outer circle show R_{halo}. The blue dashed arrow and inner circle show R_{vir}. The volume within R_{vir} holds the mass M_{vir}. Note that the large distance between the most bound particle and the deepest potential particle is not typical, but is convenient for illustrative purposes.

4 MEASURING PROFILES AND SHAPES

To measure the radial density profiles of the halos we partition each halo into concentric shells and calculate the average density for each shell. Considerations in measurement include: which halos to measure, the bounds on radius, and what type of binning scheme to apply across all halos. In addition, principle moments of inertia of each halo are determined in order to gain some indication of halo shape.

4.1 Choices in profile measurement

- Which halos to measure:
  Every halo with at least 1,000 particles inside R_{vir} is measured. We do not try to pick out "nice," spherical halos.

- Where to measure :

  We measure the density profile from the radius equal to 10 times the softening length, R_{core}, out to R_{vir} for every halo. Since the effects of gravity are most altered by softening on scales near the softening length we do not trust relative particle positions within R_{core}. However, we do trust that these particles are located within R_{core}, so we create an additional core bin that holds all of the particles between radius, r=0 and r=R_{core}. This bin is particularly important for small halos where R_{core} contains a significant fraction of the halo's mass.

- What type of binning:

  Each halo is divided into 20 radial bins, equally spaced in log space. There is also an additional core bin which holds all of the particles between radius, r=0 and r=R_{core}.

- What radius is assigned to each bin:

  When plotting the profiles we assign the radius for each bin to be the middle of the bin in log(r). The core bin is assigned the radius r=0 and so it is not visible on a log-log plot.

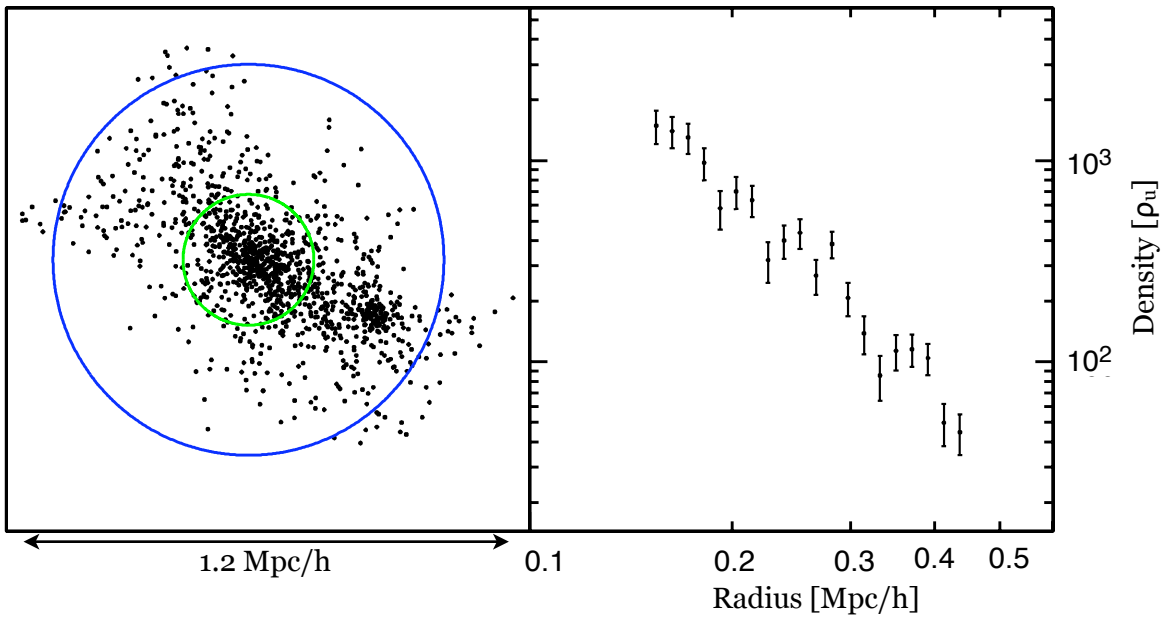


Figure 2. Measuring the density profile of a halo. The left-hand panel shows a two-dimensional projection of the halo to be measured. The blue circle shows R_{vir}, and the green circle shows R_{core}. There are 20 equally spaced logarithmic bins between R_{core} and R_{vir}. The density profile measured between these two radii is shown on the right along with corresponding Poisson error bars. There is also a core bin which measures the average density within R_{core}, but this point does not appear on the density plot.

4.2 Using principle moments of inertia as shape indicators

We find each halo's moment of inertia tensor and then diagonalize it to find the principle moments of inertia for that halo. Since the moment of inertia tensor is real and symmetric it can be diagonalized, meaning it is possible to reorient the Cartesian coordinate system such that the new inertia tensor is diagonal.

$$\hat{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix} \Rightarrow \begin{bmatrix} I_{x'} & 0 & 0 \\ 0 & I_{y'} & 0 \\ 0 & 0 & I_{z'} \end{bmatrix}$$

We can then define the largest principle moment of inertia to be I_{max}, the smallest to be I_{min}, and the intermediate to be I_{med}. If we assume that our halos are roughly ellipsoidal in shape and have a uniform density, then we can use the principle moments of inertia to calculate axis lengths:

$$a = \sqrt{\frac{5}{2}\left(\frac{I_{\max} + I_{med} - I_{\min}}{M}\right)}$$

$$b = \sqrt{\frac{5}{2}\left(\frac{I_{\max} + I_{\min} - I_{med}}{M}\right)}$$

$$c = \sqrt{\frac{5}{2}\left(\frac{I_{med} + I_{\min} - I_{\max}}{M}\right)}$$

Here, a, b, and c are the lengths of the axes of the an ellipsoid with the given principle moments of inertia. The major axis has length a, intermediate axis has length b, and smallest axis has length c. Using these estimated axis lengths, we can get a measure of symmetries, and oblateness/prolateness properties. We use an analytic formula (Smith, 1961) to find the eigenvalues of the moment of inertia tensor.

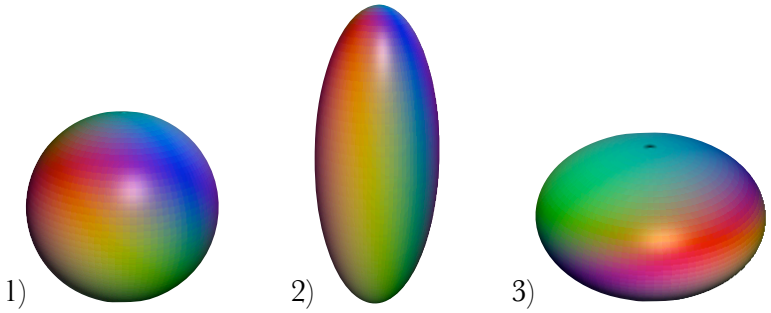


Figure 3. Several representative shapes. Shape (1) is a sphere with axis ratios 1:1:1. Shape (2) is a prolate ellipsoid, a "stretched sphere," with axis ratios 5:2:2. Shape (3) is a oblate ellipsoid, a "squashed sphere,"with axis ratios 5:5:3.

It should be noted that assuming halos have constant densities is a rather poor assumption. However, converting moments of inertia in to axis lengths for ellipsoids with non-constant density is mathematically non-trivial. Thus, our measures of shape are not the absolute halo dimensions, but rather a looser indication of halo shape. In the future, we may try using an isothermal profile instead. Additionally, while the actual values of the axis lengths may not be accurate, we have

reason to think that the axis ratios will still preserve information about halo shape, and will not be altered significantly by assuming constant density. In the end, we only consider the axis ratios and not the individual axis lengths.

5 FITTING PROFILES

We choose to fit the measured radial density profiles of all of our halos to the NFW two-parameter functional form (Navarro, Frenk & White 2006/2007), which is widely used to model halos. Fitting to the NFW form is a useful way of analyzing the profiles, but alternative forms could also have been used. Our choice of NFW was motivated by its widespread use and so that our results could be compared to those in Bullock et al. (2001). In this section we will introduce the NFW form, its parameters, and how these parameters affect the halo density profile.

5.1 Profile characteristics

In the NFW profile, the density of dark matter within a halo as a function of radius is given by:

$$\rho_{NFW}(r) = \frac{\rho_s}{(r/r_s)(1+r/r_s)^2} \quad (1)$$

Where r_s, the inner scale radius, and ρ_s, a corresponding inner density, vary from halo to halo.

The NFW form represents a smooth transition between two power law functions. The scale radius, r_s, is the radius at which the effective logarithmic slope of the profile is -2. For much smaller radii, $\rho_{NFW} \propto r^{-1}$, and for much greater radii, $\rho_{NFW} \propto r^{-3}$. The inner density is related to r_s by: $\rho_s = 4\,\rho_{NFW}(r_s)$. One can also define concentration, c_{vir}, which relates the inner parameters back to virial parameters such as the viral mass, and the virial radius. The concentration of a halo with an NFW profile is defined by: $c_{vir} = R_{vir}/r_s$. Since r_s indicates where the density function begins to drop off more steeply, for a given halo mass, a lower concentration indicates a more diffuse halo, and a higher concentration indicates that more mass is packed toward the center of the halo.

5.2 Changing parameters

One can re-write Eq. 1 in a form with parameters c_{vir} and M_{vir}. Although this form is not as elegant as Eq. 1, these parameters are more tangible and will give an intuitive understanding for the profile. Also, we have determined what M_{vir} is for our halos, but we do not know what r_s or ρ_s is for any particular halo, so this allows us to vary only one parameter instead of two when we fit to NFW profiles.

First, we have definitions for M_{vir} and c_{vir}:

$$M_{vir} \equiv \tfrac{4}{3}\pi R_{vir}^3 \Delta_{vir} \rho_u \quad (2)$$

$$c_{vir} \equiv R_{vir}/r_s \quad (3)$$

M_{vir} is also equal to the mass enclosed within R_{vir}, which can be found by integrating the density over the volume of the halo,

$$M_{vir} = \int_0^{R_{vir}} 4\pi r^2 \rho_{NFW}(r) dr = 4\pi \rho_s r_s^3 \left[\ln\left(\frac{r_s + R_{vir}}{r_s}\right) - \frac{R_{vir}}{r_s + R_{vir}} \right].$$

Now we can replace r_s by R_{vir}/c_{vir}:

$$M_{vir} = 4\pi \rho_s \left(\frac{R_{vir}}{c_{vir}}\right)^3 \left[\ln(1 + c_{vir}) - \frac{c_{vir}}{1 + c_{vir}} \right] \tag{4}$$

Combining Eq. 2 and Eq. 4 we can find ρ_s in terms of c_{vir}:

$$\rho_s = \frac{\Delta_{vir} \rho_u}{3} \left(\frac{c_{vir}^3}{\ln(1 + c_{vir}) - (c_{vir}/1 + c_{vir})} \right) \tag{5}$$

By combining Eq. 1 and Eq. 2 we can get r_s in terms of M_{vir} and c_{vir}:

$$r_s = \frac{1}{c_{vir}} \left(\frac{M_{vir}}{\frac{4}{3}\pi \Delta_{vir} \rho_u} \right)^{1/3} \tag{6}$$

Finally, substituting Eq. 3 and Eq. 4 back into Eq. 1, and after some simplification, we obtain:

$$\rho_{NFW}(r) = \frac{M_{vir} \left[\ln(1 + c_{vir}) - (c_{vir}/1 + c_{vir}) \right]^{-1}}{4\pi r \left[\frac{1}{c_{vir}} \left(\frac{M_{vir}}{\frac{4}{3}\pi \Delta_{vir} \rho_u} \right)^{1/3} + r \right]^2} \tag{7}$$

Thus, a halo with a given virial mass can have a range of NFW profiles for different values of concentration. Fig. 4 illustrates the how changing concentration affects the density profiles of a 10^{13} $M_\odot$ halo and 10^{14} $M_\odot$ halo.

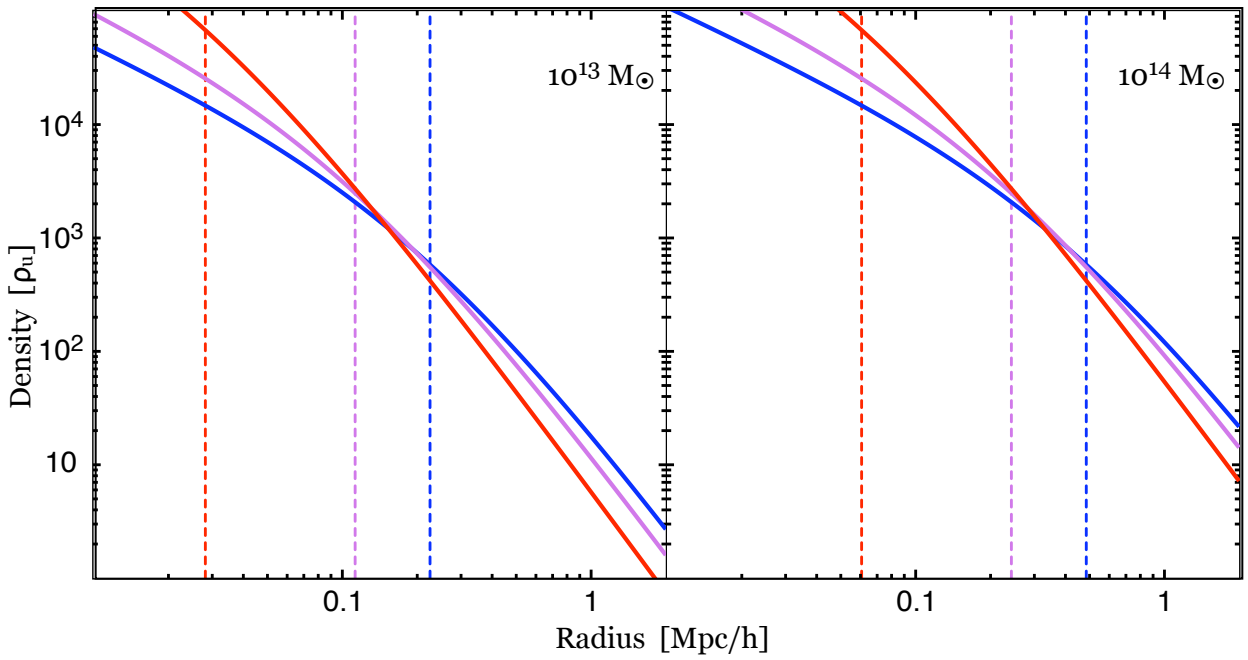


Figure 4. Effects of varying concentration for two ideal NFW halos. In both plots the x axis shows distance from the center of an NFW halo, and the y axis shows density in units of the mean density of the universe. The plot on the left is for a $M_{vir} = 10^{13}$ M⊙ halo, and the plot on the right is for a $M_{vir} = 10^{14}$ M⊙ halo. The solid curves show the halo's NFW profiles for several different concentrations. The curves in blue, purple, and red represent, respectively, the concentration values of 2, 4, and 16. The dashed lines show the location of the scale radius, r_s, for each of the values of concentration.

5.3 Fitting procedure

The measured density profiles are fit to NFW profiles by performing a χ^2 minimization. We choose to vary the concentration parameter only, and assume that the measured virial mass of a halo is equal to the M_{vir} in Eq 7. The average density found for each radial bin described in Section 4.1 is compared with the average NFW density found by integrating Eq. 7 over the radial bin. For every halo we try 100 concentrations between 0.01 and 40.01 in steps of 0.4 and choose the concentration which gives the smallest value of χ^2 to be the concentration of the best-fit NFW profile.

We perform this fit for all 3,327,464 halos in the 40 Esmeralda realizations, and find that the NFW profile produces a good statistical fit in 12% of these halos. Here we designate a good fit to be one where $\chi^2 \leq 1.878$ which corresponds to a p value of 1%.

5.4 Representative fits

The next four pages show plots of representative dark matter halos along with their measured density profiles and their best-fit NFW profiles. Some interesting things to look for in theses plots include:

- substructure / subhalos, which can create bumps in the measured density profile
- non-spherical symmetry
- subhalos outside the determined virial radius, which have not been included as part of the fitted halo
- which parts of the measured density profile differ the most from the best-fit profile

Notice that, in most halos, we find that the our data is located outside r_s, the radius where the NFW profile changes slope. This may bring into question our ability to constrain a fit; however, this is not an issue because we are fixing the M_{vir} parameter for each halo. As can be seen in Fig. 4, at a fixed mass, the amplitude of the profile outside of r_s is determined uniquely by the concentration. This is because mass, or the density integrated over the halo's volume, is conserved. Moreover, the core bin, discussed in Section 4.1, provides additional constraint to the fit inside of r_s.

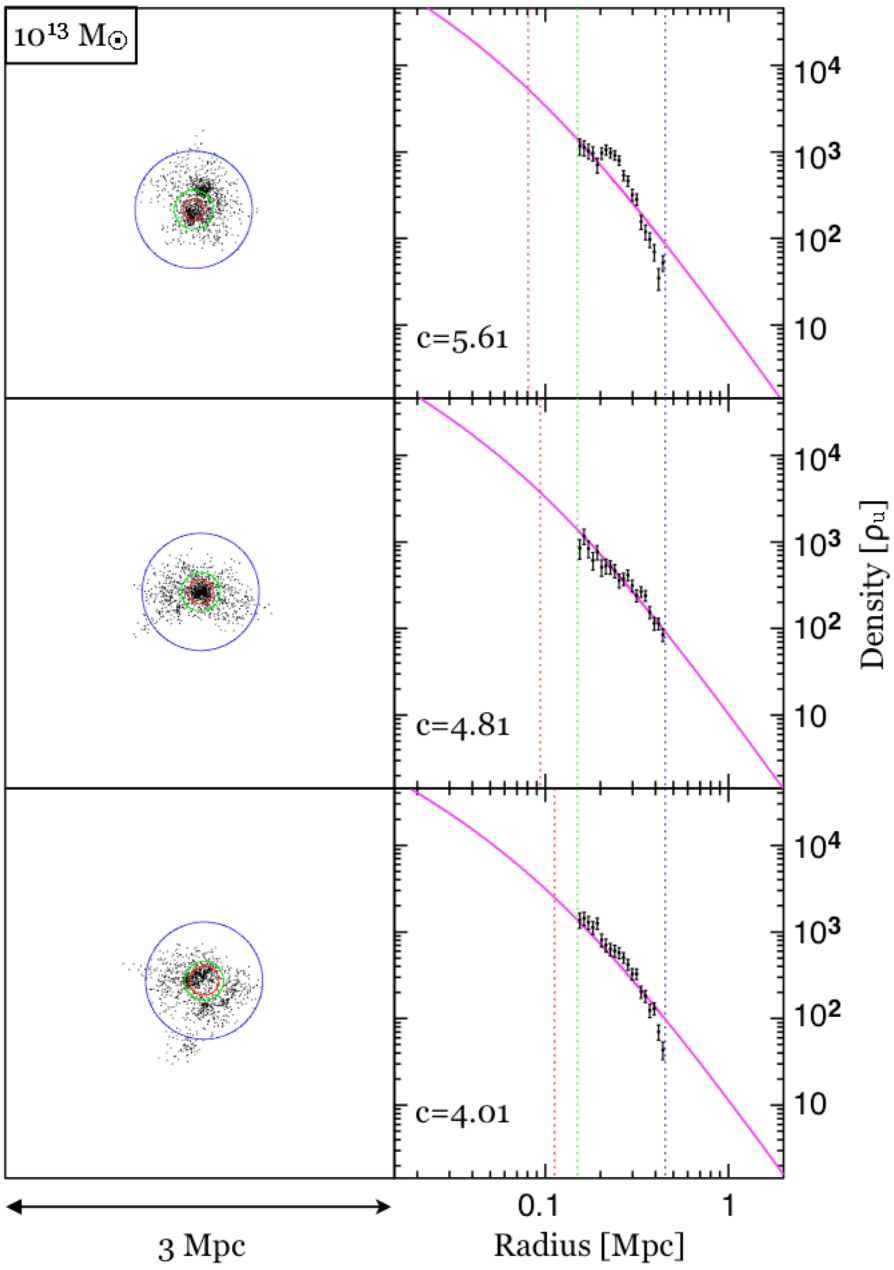


Figure 5a. Several 10^{13} M⊙ halos with their corresponding density profiles. The left-hand boxes show two dimensional projections of friends-of-friends halos with particles shown as black dots. The right-hand boxes show radial density profiles for the corresponding halo as black points. Each of these points represents the average density within each radial shell and has an associated Poisson errorbar. The green circle on the halo and the green dotted line on the profile indicate ten times the softening length, R_{core}, for the simulation. The blue circle on the halo and the blue dotted line on the profile indicate the virial radius, R_{vir}, of the halo. The profile is measured between these two radii. The solid magenta curve behind the profile is the best-fit NFW profile for the measured density profile. The red circle on the halo and the red dotted line on the profile indicate the scale radius of the halo, r_s, for the fit. The concentration value is quoted for each fit.

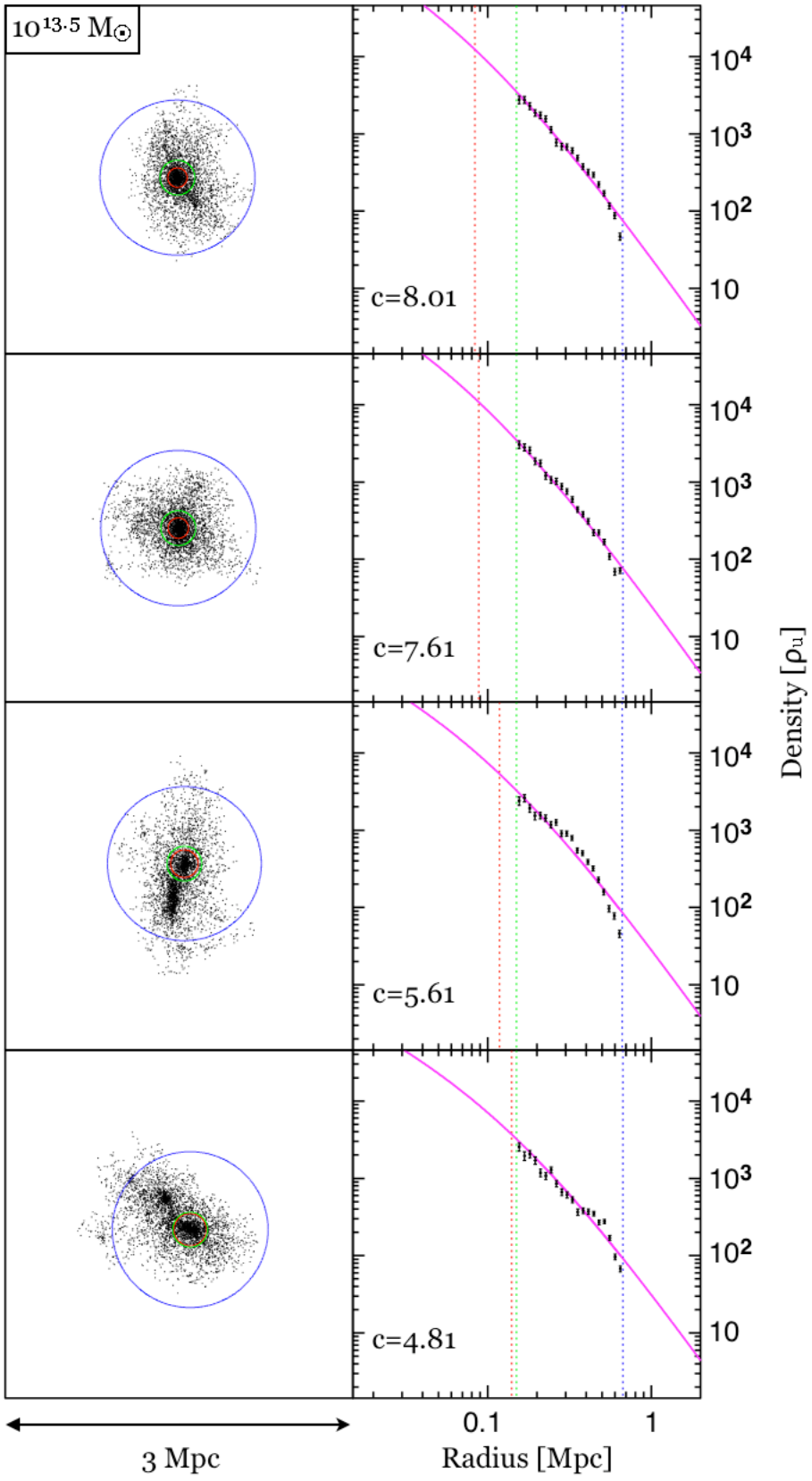


Figure 5b. Several $10^{13.5}$ M⊙ halos with their corresponding density profiles. All symbols are the same as Fig. 5a.

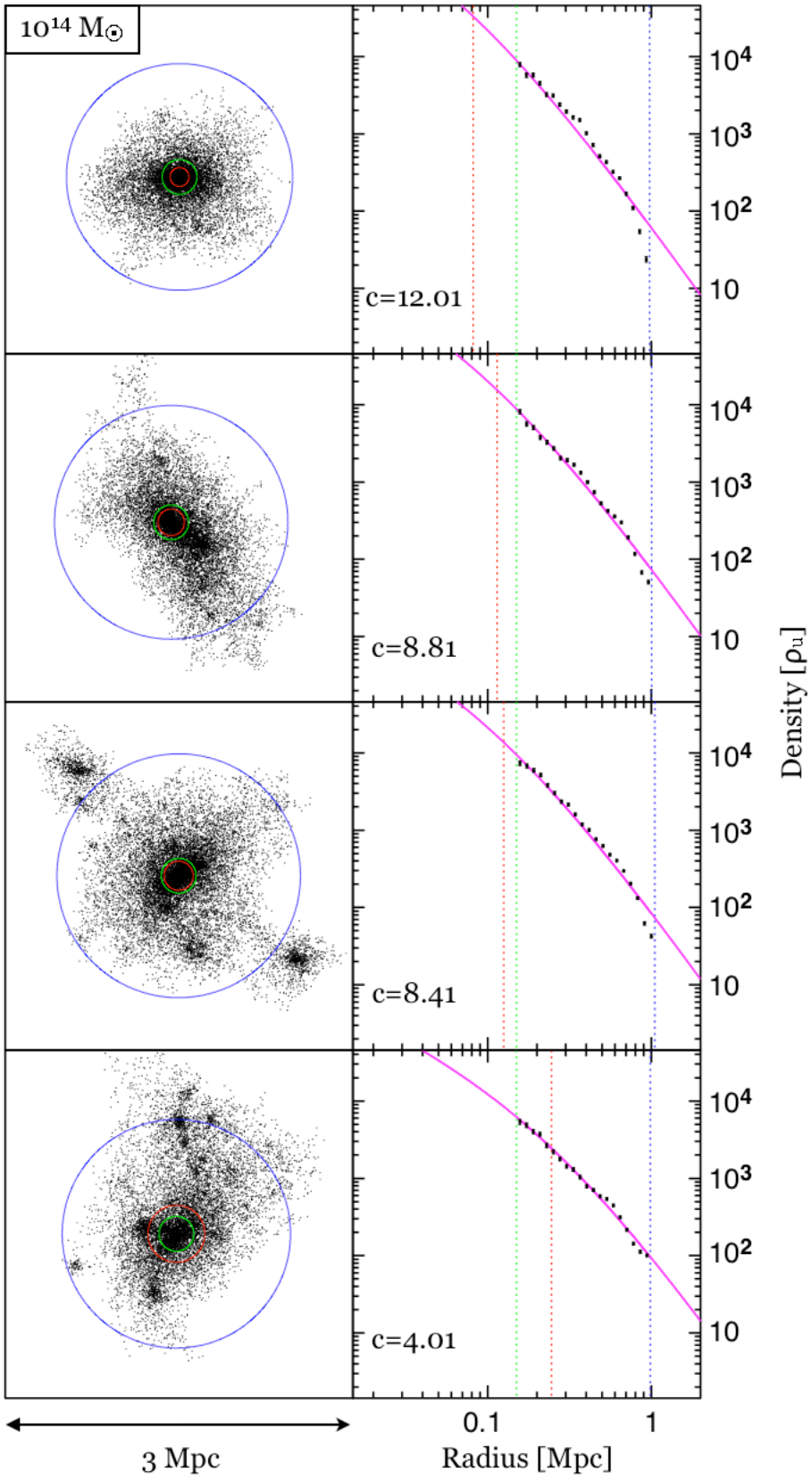


Figure 5c. Several 10^{14} M⊙ halos with their corresponding density profiles. All symbols are the same as Fig. 5a,b.

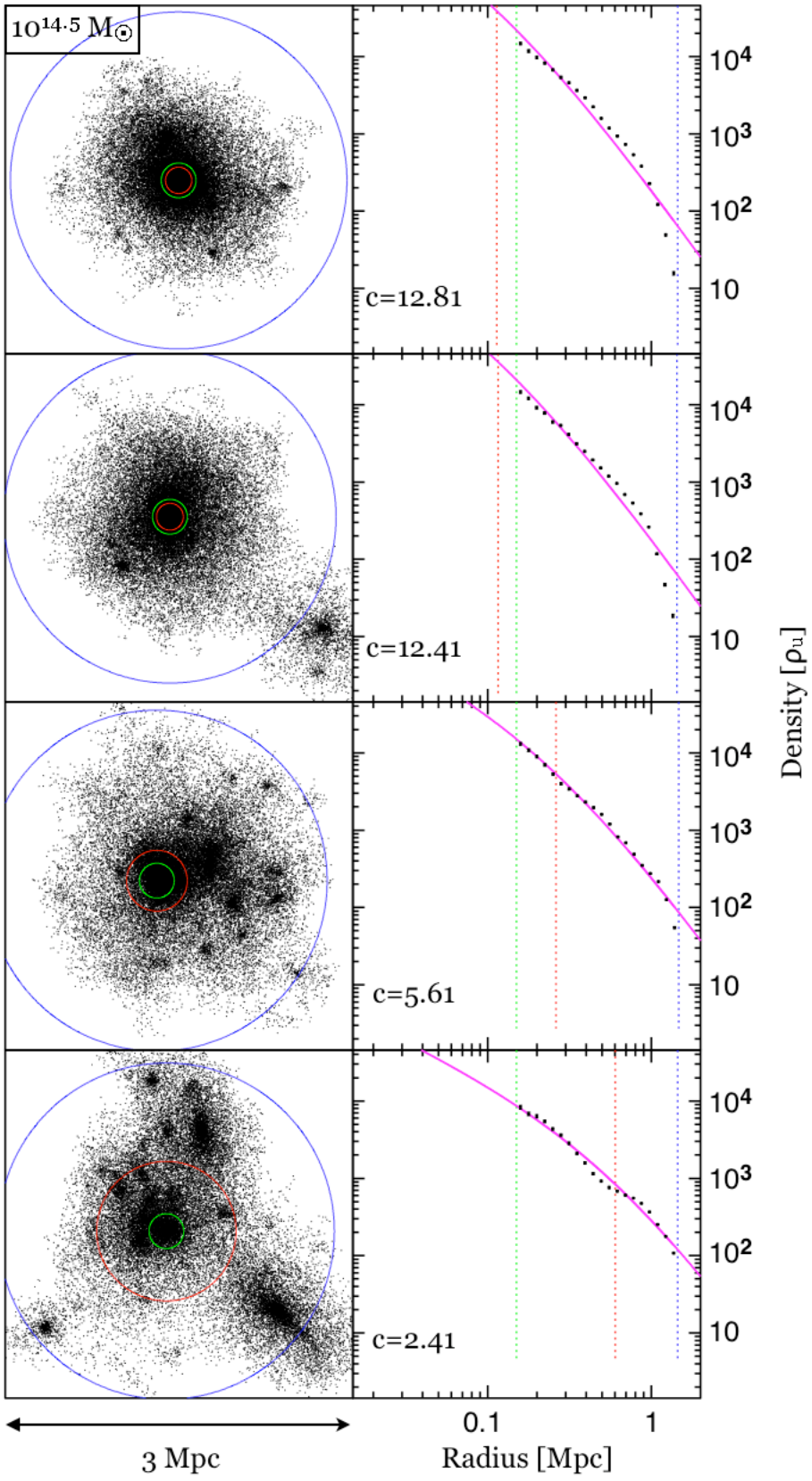


Figure 5d. Several $10^{14.5}$ M$\odot$ halos with their corresponding density profiles. All symbols are the same as Fig. 5a,b,c.

6 RESULTS

In this section we highlight our main result, the correlation between concentration and halo mass, along with results in our study of shapes. We also explore how the concentration-mass relationship behaves for subsets of halos. Particularly, we compare the subsets of worst-fit halos vs. best-fit halos, and prolate halos vs. oblate halos vs. spherical halos. Lastly, we consider the effects related to binning and resolution.

6.1 Concentration-mass relationship

To observe the concentration-mass relationship, our halos are binned according the mass. For each bin we consider the average value, the median value, and the 68 percentiles of c_{vir}. Each of the following concentration-mass plots contains the data from 40 Esmeralda realizations taken at redshift z=0.

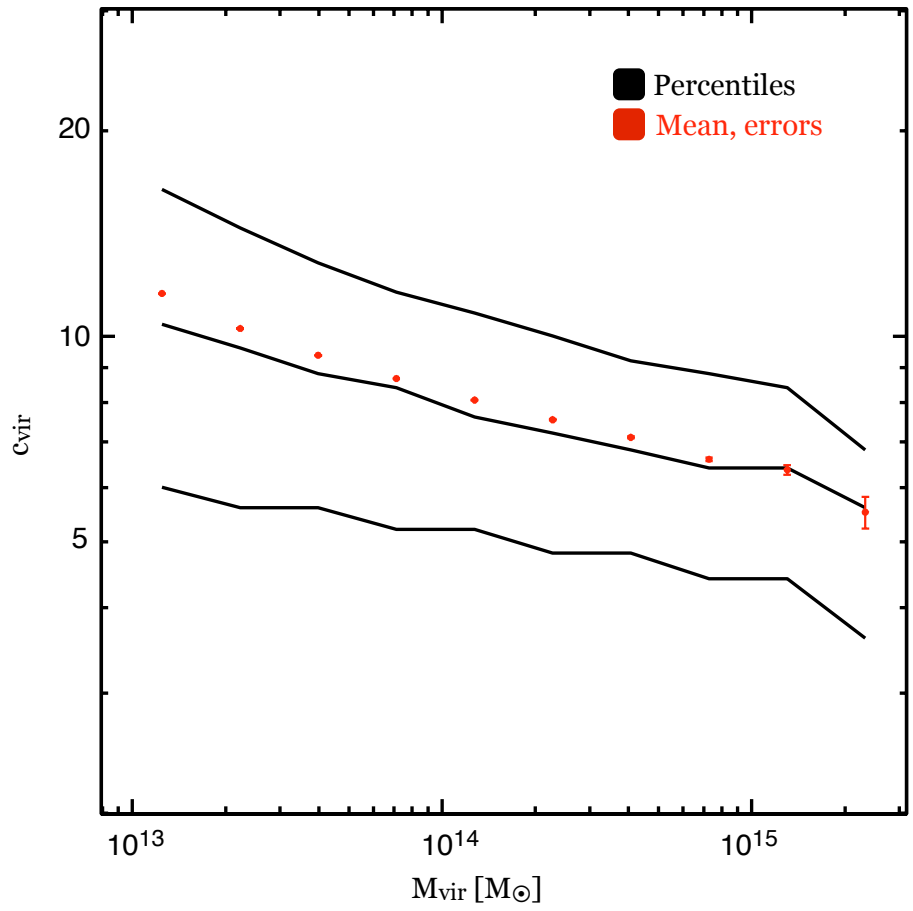


Figure 6. NFW concentration versus halo mass at z=0. The inner solid curve is the median at a given M_{vir}. The red points show the mean with error bars. The error bars, which are very small for most bins, represent the uncertainty in the mean calculated from the standard deviation of the concentration values in the bin. The outer solid curves encompass 68 percent of the concentration values. An unweighted power-law fit to the mean yields the relation: $c_{vir}(M_{vir})=11(M_{vir}/M_*)^{-0.13}$.

We find that there is a correlation between c_{vir} and M_{vir}, and that this relationship extends out to the high mass regime, even for cluster-sized dark matter halos. The average points follow the median curve for high masses, but then diverge above slightly for lower masses. It also appears that there is some curvature for the smallest masses.

Bullock et al. (2001) found the relation: $c_{vir}(M_{vir}) = 9(M_{vir}/M_*)^{-0.13}$. By applying an unweighted power-law fit to the mean, we obtain the relation: $c_{vir}(M_{vir}) = 11(M_{vir}/M_*)^{-0.13}$, which agrees with Bullock et al. (2001) in slope, but differs somewhat in intercept.

6.2 Halo shapes

In this section we consider halo shapes and categorize our halos as either spherical, prolate, oblate, or triaxial. We use ratios of moments of inertia, and axis ratios to see the distribution of halo shapes. Fig. 7 shows the distribution of halo shapes for one Esmeralda realization. Halos are called prolate if the ratio of their shortest axis to their intermediate length axis is close to 1 (> 0.9). Halos are called oblate if the ratio of their intermediate axis to their major axis is close to 1 (> 0.9). Halos which are classified as both prolate and oblate are called spherical as well. Fig. 8 shows the concentration-mass relation for 3 subsets of halo shapes: spherical, oblate, prolate.

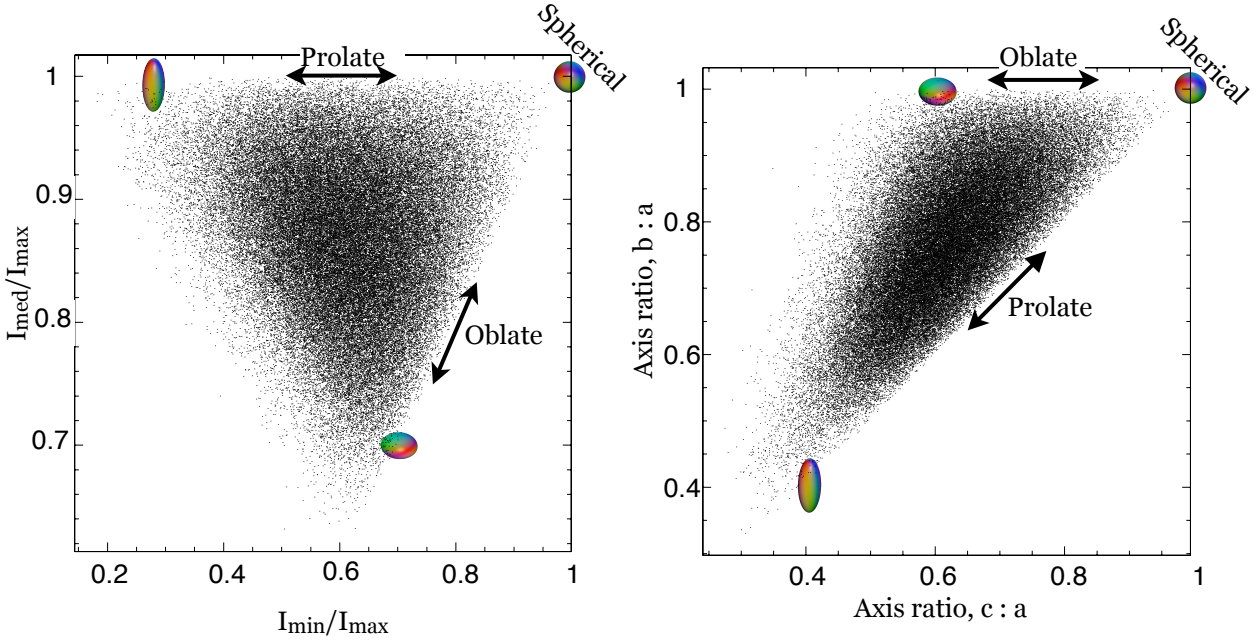


Figure 7. Distribution of halo shapes. In both plots the black points indicate the halos from one Esmeralda realization. On the left, shapes are inferred from ratios of moments of inertia. On the right, shapes are inferred from axis ratios (which are calculated from the moments of inertia). In both plots, spherical halos should be close to the point (1,1). Oblate halos lie along the lines indicated as oblate. Prolate halos lie along the lines indicated as prolate. Most halos occupy the interior regions and are triaxial.

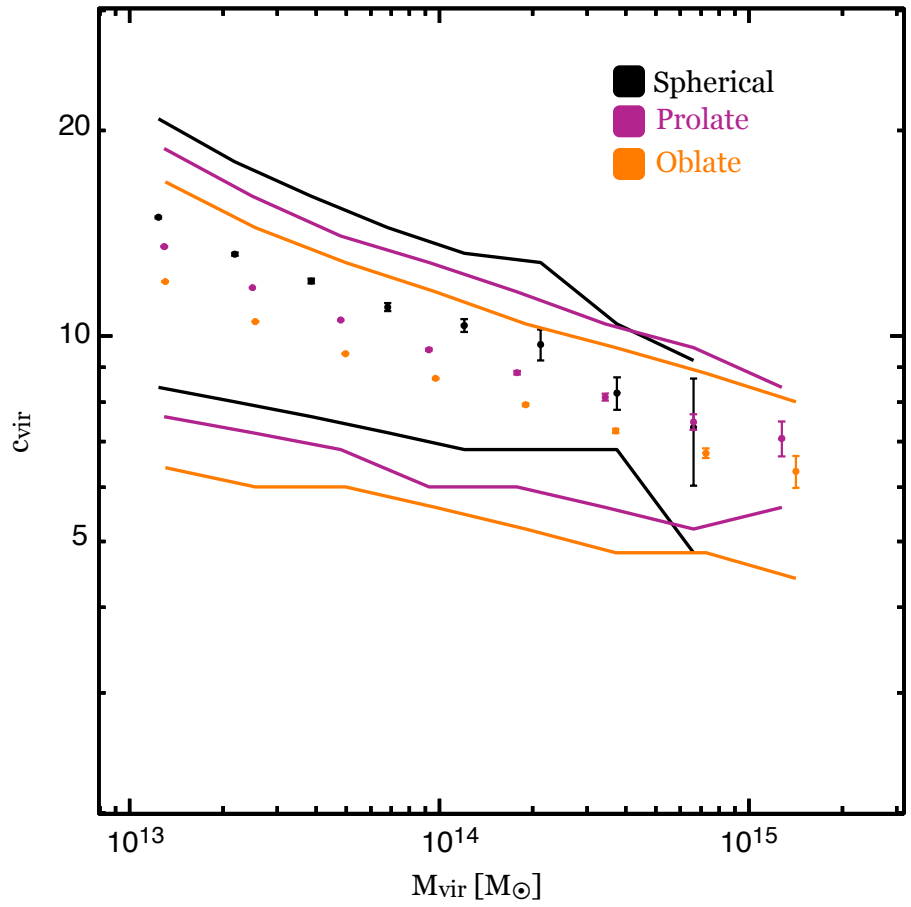
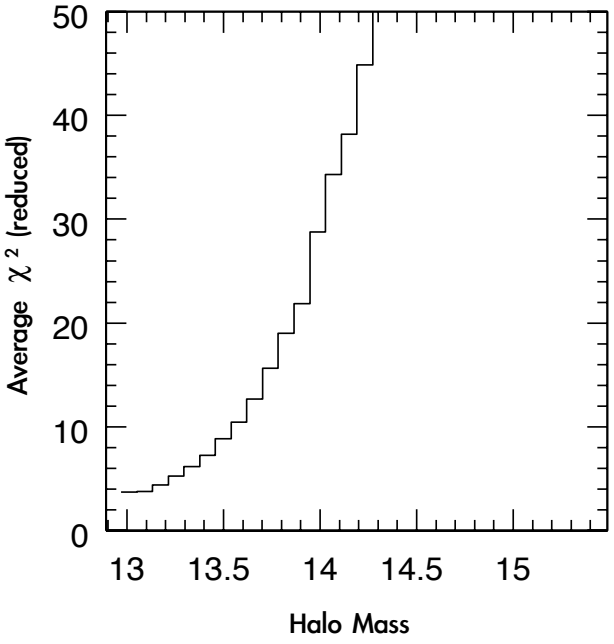


Figure 8. Comparison of NFW concentration versus halo mass for spherical, oblate, and prolate halos. Results for the halos indicated as being nearly spherical are shown in black (c/b and b/a > 0.9) . Results for the halos indicated as being nearly oblate are shown in orange (b/a > 0.9). Results for the halos indicated as being nearly prolate are shown in purple (c/b > 0.9). For each data set, the points show the mean with errorbars, and the outer curves encompass 68 percent of the concentration values.

Here, we see that the relationship for each of the subsets looks similar in slope but differs in amplitude. In general, at a given mass, halos categorized as spherical have higher concentration values than prolate halos, which have higher concentration values than oblate halos. We see that there are no spherical halos in the highest mass bins. Additionally, the main result, using all halos, most closely follows the oblate data. By applying an unweighted power-law fit to the mean, we obtain the relation: $c_{vir}(M_{vir}) = 14(M_{vir}/M_*)^{-0.17}$ for spherical halos, $c_{vir}(M_{vir}) = 12(M_{vir}/M_*)^{-0.13}$ for oblate halos, and $c_{vir}(M_{vir}) = 11(M_{vir}/M_*)^{-0.14}$ for prolate halos.

6.3 Goodness-of-fit

The majority of halos fitted to NFW profile did not find fits that were statistically "good." In particular, the average χ^2 value for the best fit profile increased exponentially with halo mass, as can be seen in the plot directly below. This is understandable since the large halos have many more points, and therefore have much smaller percent errors. For this reason we do not compare statistically good fits with statistically bad ones, but instead we divided our sample evenly according to χ^2. Halos with lower χ^2 values are put into the best fits subgroup, and halos with higher χ^2 values are put into the worst fits subgroup. The results of this are shown in Fig. 9.



we measure profiles
navarro gives us bad fits
c-m still holds up

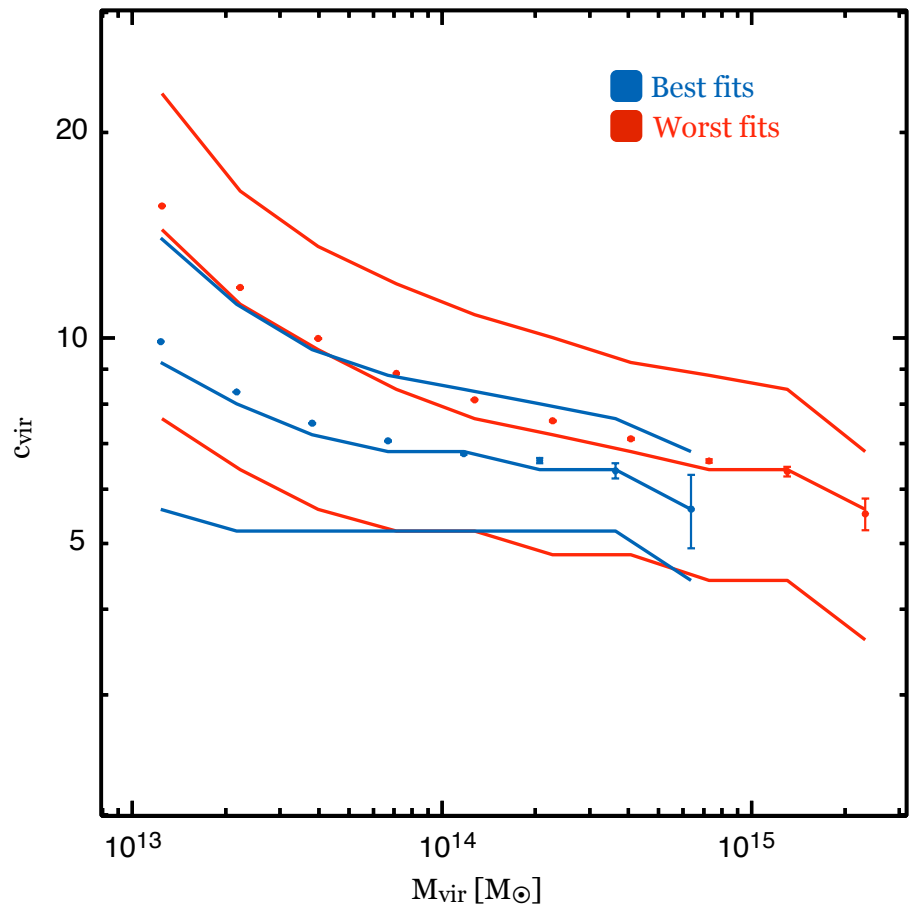


Figure 9. Comparison of NFW concentration versus halo mass for the best fit halos and the worst fit halos. Results for the best-fitting halos (with goodness-of-fit in the upper 50 percentile) are shown in blue. Results for the worst-fitting halos (with goodness-of-fit in the lower 50 percentile) are shown in red. For each data set, the inner curve is the median, the points show the mean with error bars, and the outer curves encompass 68 percent of the concentration values.

In this plot we see significant curvature for both the good fits subgroup, and for the bad fits subgroup. The halos with lower values of χ^2, the good fits, generally have lower values of concentration than the halos with higher values of χ^2, the bad fits. Also, the spread in concentrations is higher for the bad fits subgroup compared to the good fits subgroup. We find that there are no large mass halos with values of χ^2 in the good fits category, indicating that large halos are really poorly fit by the NFW profile.

6.4 Robustness: binning and resolution

Binning

As a test of robustness, we compare how changing the binning scheme when measuring the density profiles of our halos affects the concentration-mass relationship. Instead of a fixed number of bins for every halo, we chose our bins sizes such that every bin contained 50 particles.

Thus, a halo with 1,000 particles between R_{core} and R_{vir} would have 20 bins, and a halo with 10,000 particles between R_{core} and R_{vir} would have 200 bins. In both cases, the core bin contains all the particles within R_{core}.

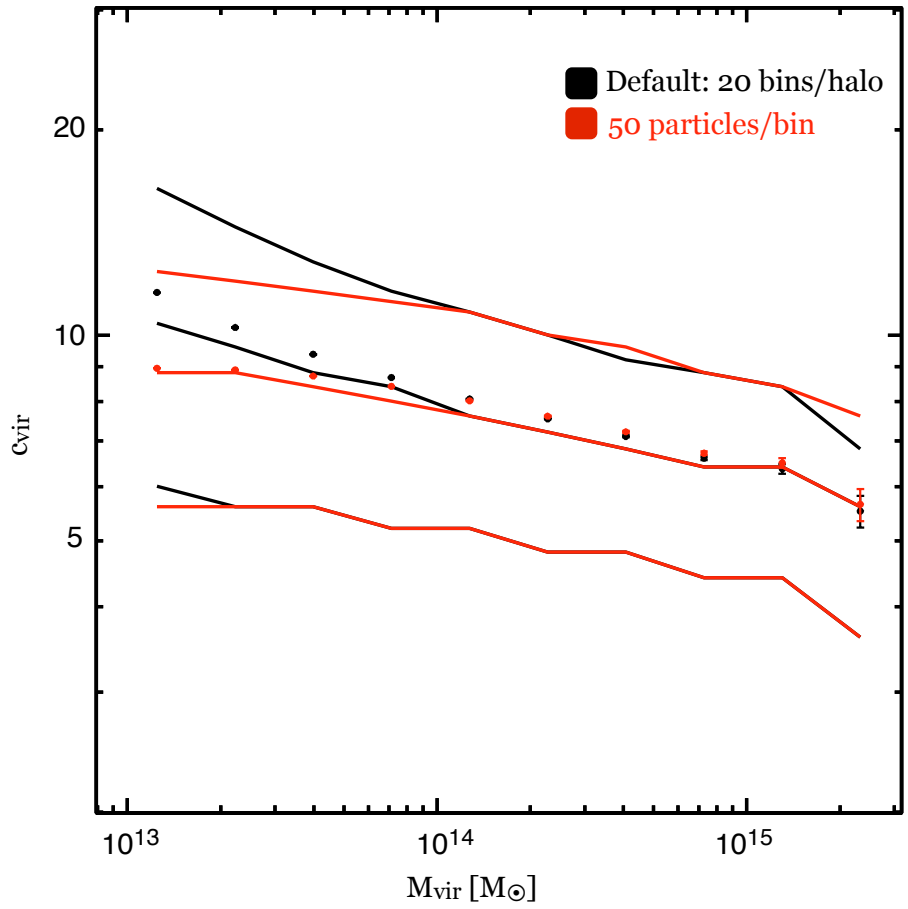


Figure 10. Comparison of NFW concentration versus halo mass for different binning schemes. Primary results from using 20 fixed-width, equally spaced, radial bins for each halo are shown in black. The results from an alternative binning scheme, in which each non-fixed-width bin holds 50 particles, is shown in red. For each data set, the inner curve is the median, the points show the mean with error bars, and the outer curves encompass 68 percent of the concentration values.

We see that binning had almost no effect over the majority of higher halo masses. However, the smaller masses are affected. The median, average, and upper percentile for concentration all decrease for smaller masses. The difference for smaller masses implies that the slight curvature seen for small mass halos is not necessarily robust.

Resolution

In order to test the possible effects of particle resolution, we compare the concentration-mass relationship for the halos in 40 Esmeralda realizations to the halos in one Carmen realization. The halos from both boxes were treated in the same way with regard to measuring the profiles and fitting these profiles to NFW. The Carmen box has a lower resolution and, thus, larger particles and a larger softening length.

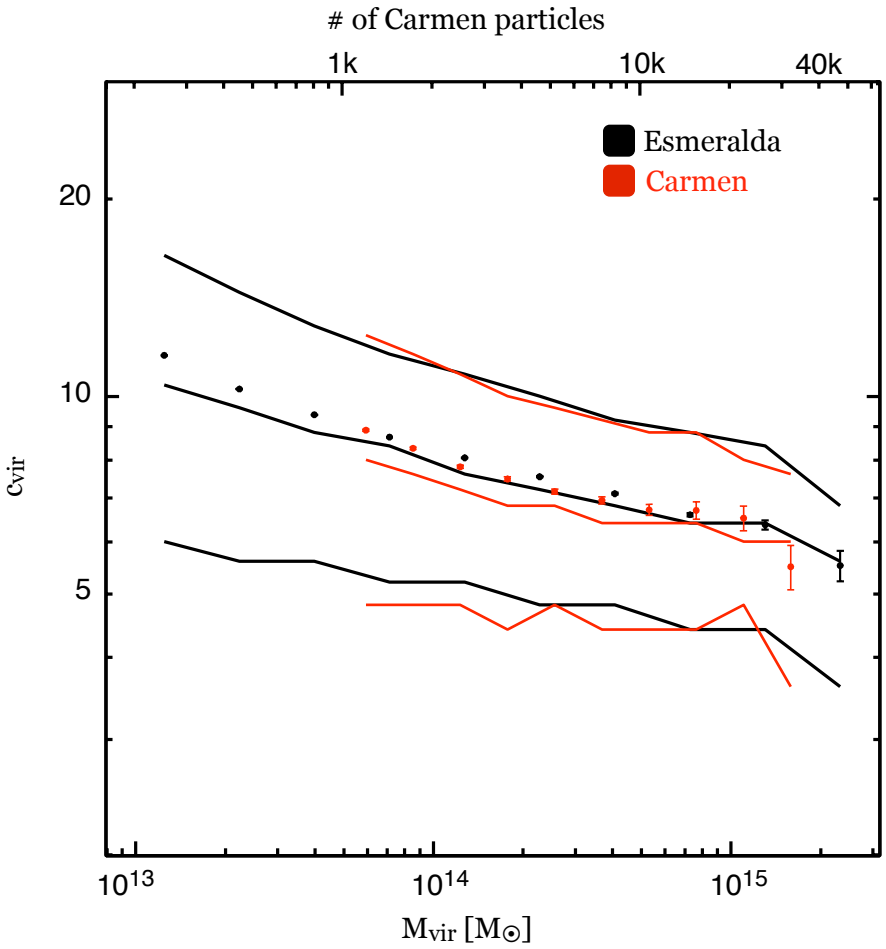


Figure 11. Comparison of NFW concentration versus halo mass for different resolutions. Primary results from the Esmeralda simulation are shown in black. Lower resolution results from the Carmen simulation are shown in red. One Carmen particle has nearly five times the mass of one Esmeralda particle. For each data set, the inner curve is the median, the points show the mean with error bars, and the outer curves encompass 68 percent of the concentration values. The top axis indicates the number of particles in a Carmen halo with the given mass. Because Carmen particles are more massive, a Carmen halo of a certain mass will have fewer particles than an Esmeralda halo of that same mass.

In general the Carmen data are a small amount lower than the Esmeralda data, but overall they are very similar. We can see a small upturn in the Carmen data as we approach the 1,000 particle lower limit which is similar to the upturn we see in the Esmeralda data. This implies that 1,000 particles may be too few obtain a realistic measure of the profile.

7 CONCLUSIONS

We find a concentration-mass relationship which agrees with the one found by Bullock et al. (2001) in slope, but not in amplitude. We also found that this power law-like relationship holds out to higher mass halos, in the previously untested mass regions. However, this relationship is dependent on which halos are considered; we found significant differences in amplitude, slope, and apparent curvature when considering various subgroups. Also, we find that most halos are not statistically we fit by the NFW profile, particularly the largest cluster-sized ones. Both binning and resolution, though to a lesser extent, have some effect on the results and indicate that 1,000 particles is too few to measure the profile accurately.

Future work

There is still work to be done on this project, and future plans may include:

- further resolution tests against Consuelo data, which will span the lower masses of Esmeralda
- further investigation of the effects of binning
- fitting profiles out to a radius less than R_{vir} since most fits appear to be failing in outer parts of the profile
- testing for substructure within halos
- measuring and fitting profiles at other redshifts
- determining shapes without assuming a constant density

ACKNOWLEDGMENTS

I thank Andreas Berlind, Cameron McBride, Jen Piscionere, Doug Watson, Qingqing Mao, and participants in Xgals for helpful guidance and discussions. I am also grateful for the funding received though the William and Nancy McMinn Scholarship which allowed me to participate in summer research in 2009, an experience which was greatly enhanced though the efforts of REU program director, Alyce Dobyns-Ladd. I also thank my friends and family for their constant love and support.

REFERENCES

- Bullock, J. S., Kolatt, T. S., Sigad, Y., Somerville, R. S., Kravtsov, A. V., Klypin, A. A., Primack, J. R. & Dekel, A. 2001, MNRAS, 321, 559, B01
- Davis, M., Efstathiou, G., Frenk, C. S., White, S. D. M., 1985, ApJ, 292, 371
- Gardner, J. P., Connolly, A., McBride, C., 2007, ASP, 376, 69
- Komatsu, E., et al 2009 ApJS 180 330
- Navarro, J. F., Frenk, C. S., White, S. D. M., 1996, ApJ, 462, 563
- Navarro, J. F., Frenk, C. S., White, S. D. M., 1997, ApJ, 490, 493
- Smith, O. K., 1961, *Commun*. of the ACM, 4, 4, 168.

APPENDIX

Code used in measuring density profiles and fitting to NFW. Written in C.

Program for determining centers:

```
// This program makes a data file containing all the centers of halos.
// Centers are positions of mbp (most bound particle) and dpp (deepest potential
  particle).
// Halos included have >= 1000 particles.
// updated 3/22 to include softening (redshift isn't quite included yet)

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include <string.h>

#define VERBOSE 0
#include "/home/robbinke/code/util/binary_output.h"
#include "/home/robbinke/code/util/bgc_read_utils.c"

FILE *fout;

// GLOBAL CONSTANTS AND VARIABLES
  char extension[] = ".centers.dat";// file name based on bgc file name
  int minNumberPart = 1000;              // halo skipped if number of particles, npart
  < minNumberPart
  double redshift = 0.0;                 // z, redshift, needed to properly compare
  potential and kinetic energies
  double a_scale = 1.0;                  // scale factor a=1/(1+z)
  double softening = 0.015;              // softening length, depends on simulation,
  Mpc/h

// FUNCTIONS
  void newFileName(char * file, char * longFile, char * extension);
  int process_halos(FILE *fp, const OUTPUT_HEADER hdr, char * bgc_file);
  int try_file( char * bgc_file);


int main(int argc, char ** argv)
{
    int i;

    if(argc < 2)
    {
        fprintf(stderr, "Usage: ");
        fprintf(stderr, "  %s redshift softening BGC_file[s] \n", argv[0]);
        exit(EXIT_FAILURE);
    }

    redshift = atof(argv[1]);
    a_scale = 1.0/(1.0 + redshift);
    softening = atof(argv[2]);

    /* loop over input files for processing */
    for(i=3; i<argc; i++)
    {
        char * bgc_file = argv[i];
        try_file( bgc_file );
    }
```

```
    return(EXIT_SUCCESS);
}

// FUNCTION process_halos
// Called by try_file, goes though all the halos in the current bgc file, calculates
  and outputs results into files
int process_halos(FILE *fp, const OUTPUT_HEADER hdr, char * bgc_file)
{
    int i,j,k,n;

    // Set up input and output files.
     char output[strlen(bgc_file) + strlen(extension)];
     newFileName(output, bgc_file, extension); //output filename is based on the bgc
  filename
     fout = fopen(output,"a");
     // Check that files have opened properly
     if(fout == NULL) { fprintf(stderr, "ERROR: problem opening file '%s'\n", output);
  assert(fout != NULL); }
     fprintf(stderr, "Output file: %s\n", output);

    int *nParticlesPerGroup;
    PARTICLE_DATA_PV  *pdata;

    /* allocate array of structures based on the biggest halo, which means only once
  per file */
    pdata = calloc( hdr.max_npart, bgc_sizeof_pdata(hdr.format) );
    assert(pdata != NULL);

    nParticlesPerGroup = bgc_read_grouplist(fp,hdr);

    // loops though each halo in the file
    for(i=0; i < hdr.ngroups; i++)
    {
     int gid   = i + hdr.first_group_id;
        int npart = nParticlesPerGroup[i];

        //int flag_periodic = 0;
        PARTICLE_DATA_PV pd1;
     PARTICLE_DATA_PV pd2;
     double xdpp[3], xmbp[3], E, U;
     E = U = 0.0;

     double mpart = 0.931*pow(10.,10.);
        double G = 4.3*pow(10.0,-9.0);

        bgc_read_part_into(fp, npart, hdr.format, pdata);


        // Conditions to skip haloes
     if(npart < minNumberPart)
          continue;


  /* Find center (deepest potential particle)*/
     for(k=0; k<3; k++)
          xdpp[k] = xmbp[k] = 0.0;

     for(n=0; n < npart; n++)
     {
          pd1 = pdata[n];
          double Ei = 0.0, Ki = 0.0, Ui = 0.0;

          for(j=0; j < npart; j++)
          {
```

```
                pd2 = pdata[j];
                double rsq = 0., d = 0., plength_check = hdr.BoxSize/5.0;
                for(k=0; k<3; k++)
                {
                     d = pd1.pos[k] - pd2.pos[k];

                     if(d > plength_check)
                          d = pd2.pos[k] + hdr.BoxSize - pd1.pos[k];

                     else if(d < -1.0 * plength_check)
                          d = pd1.pos[k] + hdr.BoxSize - pd2.pos[k];

                     rsq += d*d;
                }
                double r = sqrt(rsq);

                if(r != 0.0)
                     Ui += -1.0*G*mpart*mpart/(r+softening);
          }

          Ki = 0.5*mpart*(pow(pd1.vel[0], 2.0) + pow(pd1.vel[1], 2.0) + pow(pd1.vel
[2], 2.0));
          Ei = Ui + Ki;

          if(Ei < E || n==0)
          {
                E = Ei;
                for(k=0; k<3; k++)
                     xmbp[k] = pd1.pos[k];
          }

          if(Ui < U)
          {
                U = Ui;
                for(k=0; k<3; k++)
                     xdpp[k] = pd1.pos[k];
          }
     }

// Find the seperation of the two centers
     double rsq = 0., d = 0., plength_check = hdr.BoxSize/5.0;
                for(k=0; k<3; k++)
                {
                     d = xmbp[k] - xdpp[k];

                     if(d > plength_check)
                          d = xdpp[k] + hdr.BoxSize - xmbp[k];

                     else if(d < -1.0 * plength_check)
                          d = xmbp[k] + hdr.BoxSize - xdpp[k];

                     rsq += d*d;
                }
     double difference = sqrt(rsq);

// Print centers out to file
     fprintf(fout, "%08d\t%010d\t", gid, npart);
        for(k=0; k<3; k++)
            fprintf(fout, "%08.4f\t", xmbp[k]);
     for(k=0; k<3; k++)
          fprintf(fout, "%08.4f\t", xdpp[k]);
     fprintf(fout, "%08.4f\n", difference);

   }
```

```
    fclose(fout);
    free (pdata);
    return hdr.ngroups;
}

// FUNCTION try_file
// checks that bgc_file opens properly then calls function process_halos
int try_file( char * bgc_file )
{
    FILE * fp;
    OUTPUT_HEADER hdr;
    int ngroups_read = 0;

    fprintf(stderr, "Reading BGC file: %s\n", bgc_file);
//     fprintf(stdout, "# from BGC file: %s\n", bgc_file);
    fflush(stderr);
    fp = fopen(bgc_file, "r");
    if(fp == NULL)
    {
        fprintf(stderr, "ERROR: problem opening file '%s'\n", bgc_file);
        assert(fp != NULL);
    }

    bgc_read_header(fp,&hdr);

    if(PDATA_FORMAT_PV == hdr.format)
    {
//         fprintf(stdout, "# 11 columns: gid(0) npart(1) group_mass(2) pos_com(3,4,5)
  vel_com(6,7,8) vdisp(9) periodic_needed(10)\n");

        ngroups_read = process_halos(fp, hdr, bgc_file);
    }
    else if( PDATA_FORMAT_PVBE == hdr.format )
    {
//         ngroups_read = calc_stats_mbp(fp, hdr);
        fprintf(stderr, "USING BINDING ENERGY IS NOT YET IMPLEMENTED!\n");
        return(EXIT_FAILURE);
    }
    else
    {
        fprintf(stderr,"ERROR: skipping '%s' -- PDATA_FORMAT not compatible (%d)\n",
  bgc_file, hdr.format);
        return(EXIT_FAILURE);
    }

    return ngroups_read;
}

// FUNCTION newFileName
// 'longFile' is the filename we want to modify and is in the form [characters]/
  [middle characters].[more characters]
// [middle characters] cannot have any '/' or '.' in it.
// Changes 'file' to be a character array that is [middle_characters][extension]
// Be carefule to make sure that 'file' is long enough to hold all the characters and
  extension, or there will be problems...
// initializing file as: char * file[strlen(longFile) + strlen(extension)]; should
  guarantee good things
void newFileName(char * file, char * longFile, char * extension)
{
  int i, start = -1, end = strlen(longFile);
  for(i= 0; i < strlen(longFile); i++)
  {
     if(longFile[i] == '/')
          start = i;
```

```
  }
  for(i= strlen(longFile) - 1; i > start; i--)
  {
     if(longFile[i] == '.')
          end = i;
  }
  for(i= start+1; i < end; i++)
  {
     file[i-(start+1)] = longFile[i];
  }
  for(i= (end-start)-1; i < strlen(file); i++) //trims out any other characters that
  may have been left in 'file'
  {
     file[i] = '\0';
  }

  strcat(file, extension);
}
```

Program for measuring profiles and fitting to NFW:

```
/*
  fittingNFW.c

  Created 2/03/2010 by Katie Robbins

  **What it does**
  Fits dark matter halos with > minNumPart particles to an NFW profile by using either
  (1) a fixed number of equal width bins in log(radius) or (2) a specified number of
  particles per bin, such that the total number of bins across halos varies. Does one
  realization at a time.

  **Usage**
  Usage: ./fittingNFW [bins] [deltavir] [softing] [centers_file] [BGC_file(s)]
       - where bins is the number of bins when using a fixed number of bins, if bins
  is <= 0 the binning will be a fixed number of particles per bin.
       - deltavir specifies the value of deltavir (normally 377, corresponds to an
  overdensity and is used to find Rvir)
       - softening specifies the softening length in Mpc/h, differs according to
  simulation type
       - centers_file is the file containing centers data for the halos, created by
  the centers program, there is 1 centers file for each realization.
       - BGC_file(s) are the files containing friends-of-friends halo data. There are
  several BGC files for each realization.

  **Outputs**
       Ouputs a data file (with a name based on the input BGC files) containing a
  bunch of stuff about each halo. One line per halo.
       Also outputs a geometry file also containing a bunch of stuff about each halo,
  one line per halo.

       Ouputs: identification #, # particles, chi squared (reduced), Mass2(final
  trimmed mass), best fit concentration, Radius2(encloses M2), Mass friend-of-friends,
  Mass1(partially trimmed mass), Radius1(encloses M1), chi squared, degrees of
  freedom, maximum/intermediate/minimum principle moment of inertia, A/B/C axis
  lengths, center positions x,y,z for most bound particle/ center of mass/ deepest
  potential particle, seperation between center of mass and most bound particle,
  difference (distance between) deepest potential particle and the most bound
  particle.

       Data output file includes: id(1) #particles(2) chi^2red.(3) M2 (4) cfit(5) R2
  (6) Mfof(7) M1(8) R1(9) chi^2(10) dof(11)
```

```
      Geo output file includes: id(1) #particles(2) Imax(3) Imin(4) Imed(5) Aaxis(6)
  Baxis(7) Caxis(8) mbp-x,y,z(9,10,11) com-x,y,z(12,13,14) dpp-x,y,z(15,16,17)
  seperation(18) difference(19)

  Note on scaling errors:
  Gives all halos similar percent errors.
  Turning on scaleErrors will change sigma from sigma = sqrt(N), where N is the number
  of particles in a bin to sigma = sqrt(N)*sqrt(Ntot/minNumPart), where Ntot is the
  total number of particles in that halo, and minNumPart is the least possible number
  of particles considered for fitting.
  Scaling the errors like this allows us to compare how well halos of different masses
  will fit an NFW profile when only considering the shape of the profile. This still
  preserves the relative size of the errors across bins.
*/


#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <math.h>
#include <string.h>

#define VERBOSE 0
#include "/home/robbinke/code/util/binary_output.h"
#include "/home/robbinke/code/util/bgc_read_utils.c"
FILE *fout;
FILE *fout2;
FILE *fin;

#define avg_density (0.25*2.775*pow(10.0,11.0))

// GLOBAL CONSTANTS AND VARIABLES
  int minNumPart = 1000;      // only halos with >= minNumPart will be fit
  int ppbin = 50;             // number of particles per bin, percent errors are sqrt
  (Nbin)/Nbin= 1/sqrt(Nbin),
  double scaleRmin = 10.;     // Rmin = softening*scaleRmin, higher is more
  conservative
  double scaleRmax = 1.0;     // Rmax = R2*scaleRmax, should be <= 1
  int const N_C = 100;        // number of concentrations to try
  double cRangeMin = 0.01; // lower limit for concentration, can't be 0!
  double cRangeMax = 40.01;   // upper limit for concentration
  double const PLENGTH = 5.0; // when checking for periodic wrapping to determine
  distances, wrapping occurs when the distance between two points is greater than
  BOXLENGTH/PLENGTH.
  double const GOODCHI = 1.8; // p = 1%
  int printedHalo = 0;        // keeps track of whether a single halo has been printed
  out when singleHalo is >= 0

  //options
  int scaleErrors = 0;  // errors will be scaled (see note at top) if scaleErrors is
  set to 1
  int printgeo = 1;     // prints a geometry file if set to 1
  int printfit = 1;     // prints a fit file if set to 1

  //set in main()
  char * extension;     // extension for the fitting file
  char * extension2;    // extension for the geometry file
  int BINS;             // number of radial bins to put particles into, if zero then
  equal number of particles for each bin
  int singleHalo;       // if >= 0, only print out the fit information for 1 halo with
  the ID#=singleHalo
  double delta_vir;     // used to find Rvir
  double softening;     // softening for given simulation in Mpc/h
```

```
// FUNCTIONS
  void diagonalize(double S[3][3], double * Ixx, double * Iyy, double * Izz);
  double get_r(double pos1[], double pos2[], double BoxSize);
  double getNFWDensity(double logR1, double logR2, double c, double logM);
  void newFileName(char * file, char * longFile, char * extension);
  int process_halos(FILE *fp, const OUTPUT_HEADER hdr, char * centers_file);
  int try_file( char * bgc_file, char * centers_file );
  int compareDoubles (const void *A, const void *B);
  void findAxisRatios(double Ieig[3], double mass, double * major, double * medium,
  double * minor);


int main(int argc, char ** argv)
{
    int i;

    if(argc <= 6)
    {
        fprintf(stderr, "Usage: ");
        fprintf(stderr, "  %s [bins] [deltavir] [softing] [centers_file] [BGC_file(s)]
  \n", argv[0]);
        exit(EXIT_FAILURE);
    }

  singleHalo = atoi(argv[1]);
  BINS = atoi(argv[2]);
  delta_vir = atof(argv[3]);
  softening = atof(argv[4]);
  extension = argv[5];
  extension2 = argv[6];
  char * centers_file  = argv[7];

    /* loop over input files for processing */
    for(i=8; i<argc; i++)
    {
        char * bgc_file = argv[i];

        // Check to make sure that bgc_file matches centers_file.
          char ext[] = "";
          char cen[strlen(centers_file) + strlen(ext)];
          char bgc[strlen(bgc_file) + strlen(ext)];
          newFileName(cen, centers_file, ext);
          newFileName(bgc, bgc_file, ext);
          int same = strcmp(cen, bgc);

       if(same == 0)
          try_file( bgc_file, centers_file );
    }

    return(EXIT_SUCCESS);
}

// FUNCTION process_halos
// Called by try_file, goes though all the halos in the current bgc file, calculates
  and outputs results into files
int process_halos(FILE *fp, const OUTPUT_HEADER hdr, char * centers_file)
{
    int i,j,k,n, haloIndex;

    // Set up input and output files.
       char output[strlen(centers_file) + strlen(extension)];
       char output2[strlen(centers_file) + strlen(extension2)];
       newFileName(output, centers_file, extension); //output filename is based on the
  centers filename
```

```
     newFileName(output2, centers_file, extension2);
     fin = fopen(centers_file,"r");
     fout = fopen(output,"a");
     fout2 = fopen(output2,"a");
// Check that files have opened properly
     if(fin == NULL) { fprintf(stderr, "ERROR: problem opening file '%s'\n",
centers_file); assert(fin != NULL); }
     if(fout == NULL) { fprintf(stderr, "ERROR: problem opening file '%s'\n",
output); assert(fout != NULL); }
     if(fout2 == NULL) { fprintf(stderr, "ERROR: problem opening file '%s'\n",
output2); assert(fout2 != NULL); }

  int *nParticlesPerGroup;
  PARTICLE_DATA_PV  *pdata;
  double plength_check = hdr.BoxSize / PLENGTH;

  /* Allocate array of structures based on the biggest halo, which means only once
per file */
  pdata = calloc( hdr.max_npart, bgc_sizeof_pdata(hdr.format) );
  assert(pdata != NULL);

  // Allocate arrays for binning
     int *number = malloc (sizeof (int) * BINS);
     double *radii = malloc (sizeof (double) * hdr.max_npart);

     if(BINS > 0) // fixed width bins in log(R)
        n = BINS;
     else
        n = hdr.max_npart/ppbin;

     double *density = malloc (sizeof (double) * n);
     double *sig_d = malloc (sizeof (double) * n);

  nParticlesPerGroup = bgc_read_grouplist(fp,hdr);

  // Loops though each halo in the file
  for(haloIndex=0; haloIndex < hdr.ngroups && printedHalo == 0; haloIndex++)
  {
     int gid   = haloIndex + hdr.first_group_id;
      int npart = nParticlesPerGroup[haloIndex];
      PARTICLE_DATA_PV pd1;
      double mpart = hdr.part_mass * pow(10.0,10.0);

      bgc_read_part_into(fp, npart, hdr.format, pdata);

 // Check if we only want to print out the info for 1 halo with id=singleHalo
     if(singleHalo >= 0)
     {
        if(singleHalo != gid) // skip it
           continue;
        else
           printedHalo = 1;
     }

// Conditions to skip haloes
     if(npart < minNumPart)
        continue;

     int gid_r, npart_r;
     double xmbp[3], xdpp[3], diff;
     int halofound = 0;

// Look for matching gid and npart, will scan to the end of input then stop looking
     while(fscanf(fin, "%d %d %lf %lf %lf %lf %lf %lf %lf", &gid_r, &npart_r, &xmbp
```

```
[0], &xmbp[1], &xmbp[2], &xdpp[0], &xdpp[1], &xdpp[2], &diff) == 9)
     {
            if(gid == gid_r && npart == npart_r)
            {
               halofound = 1;
               break;
            }
     }

// Skip over halo if center isn't found, and rewind file input stream
     if(halofound == 0)
     {
        fprintf(stderr, "ERROR: center isn't found for ID=%d in '%s'\n", gid,
centers_file);
        rewind(fin);
        continue;
     }

/* Trimming points.
// Mfof is total mass of halo
// Radius R1 contains mass M1
// Radius R2 contains mass M2
*/
        double Mfof = mpart *(double)npart;
        double R1 = pow(3.0/4.0/M_PI*Mfof/delta_vir/avg_density, 1.0/3.0);
        double M1 = 0.0;
        double R2;
        double M2 = 0.0;

     //First trim
        for(n=0; n < npart; n++)
        {
           pd1 = pdata[n];
           double pos[3] = {pd1.pos[0], pd1.pos[1], pd1.pos[2]};
           double r = get_r(pos, xmbp, hdr.BoxSize);
           if(r < R1)
              M1 += mpart;
        }

     //Second trim, Binning, determine center of mass position and principle moments
of inertia
     //  (done simultaneously in order to only loop though all particles once)
        R2 = pow(3.0/4.0/M_PI*M1/delta_vir/avg_density, 1.0/3.0);
        int numberCore = 0;           // number of particles in core bin
        int numberShells = 0;         // number of particles in fitting shells
        double com[3] = {0.,0.,0.};   // the center of mass position
        double I[3][3];               // moment of inertia tensor
           for(i=0; i<3; i++)
              for(j=0; j<3; j++)
                 I[i][j] = 0.0;

        double logrRangeMax = log10(R2*scaleRmax);           // Maximum radius for
fitting.
        double logrRangeMin = log10(softening*scaleRmin);        // Minimum radius
for fitting.
        double logrWidth = logrRangeMax - logrRangeMin;          // Radial bin
width. Only used for equal width bins.
        if(BINS > 0)
        {
           logrWidth = (logrRangeMax - logrRangeMin)/((double)BINS);
           for(k=0; k<BINS; k++)
              number[k] = 0;
        }
```

```
         for(n=0; n < npart; n++)
         {
            int isInsideR2 = 0;
            pd1 = pdata[n];
            double pos[3] = {pd1.pos[0], pd1.pos[1], pd1.pos[2]};
            double logr = log10( get_r(pos, xmbp, hdr.BoxSize) );

            if(logr <= logrRangeMax && logr >= logrRangeMin) //particle is in the
shell region
            {
               if(BINS > 0)
                  number[(int)((logr-logrRangeMin)/logrWidth)]++;
               else
                  radii[numberShells] = pow(10.0,logr);

               numberShells++;
               isInsideR2 = 1;
            }

            else if(logr < logrRangeMin) // particle is in the core region
            {
               numberCore++;
               isInsideR2 = 1;
            }

         /* Particles inside logrRangeMax contribute to M2, center of mass position,
and the moment of inertia tensor */
            if(isInsideR2 == 1)
            {
               M2 += mpart;

               // Find center of mass, and moment of inertia tensor for trimmed halo
                  double d[3];  // position of particle (with the mbp at origin)
                  for(k=0; k<3; k++)
                  {
                     d[k] = pd1.pos[k] - xmbp[k];

                     if(d[k] > plength_check)
                        d[k] = xmbp[k] + hdr.BoxSize - pd1.pos[k];

                     else if(d[k] < -1.0 * plength_check)
                        d[k] = pd1.pos[k] + hdr.BoxSize - xmbp[k];

                     com[k] += xmbp[k] + d[k];
                  }

                  I[0][0] += mpart*(d[1]*d[1] + d[2]*d[2]);
                  I[1][1] += mpart*(d[0]*d[0] + d[2]*d[2]);
                  I[2][2] += mpart*(d[0]*d[0] + d[1]*d[1]);
                  I[0][1] -= mpart*d[0]*d[1];
                  I[0][2] -= mpart*d[0]*d[2];
                  I[1][2] -= mpart*d[1]*d[2];
                  I[1][0] = I[0][1];
                  I[2][0] = I[0][2];
                  I[2][1] = I[1][2];
            }
         }

// Condition to skip haloes after trimming
     if((numberShells + numberCore) < minNumPart)
         continue;

     for(k=0; k<3; k++)
         com[k] = com[k]/(double)(numberShells + numberCore);
```

```
/*Find the eigenvalues for moment of inertia tensor*/
     double Ieig[3] = {0.,0.,0.};         // the eigenvalues of the moment of inertia
tensor
     double axisRatios[3] = {0.,0.,0.};   // the axis ratios
     diagonalize(I, &Ieig[0], &Ieig[1], &Ieig[2]);
     findAxisRatios(Ieig, (double)(numberShells + numberCore)*mpart, &axisRatios[0],
&axisRatios[1], &axisRatios[2]);

     if(BINS > 0)
     {
     /* Make array of densities in each shell, binning by logR the same as above.
Also, get sigma (poisson errors).  */
        // double density[BINS] holds the density of each bin
        // double sig_d[BINS] holds the error for each bin
        for(n=0; n < BINS; n++)
        {
           double Rout = pow(10.0, logrRangeMin + logrWidth*(double)(n+1));
           double Rin = pow(10.0, logrRangeMin + logrWidth*(double)n);
           double Vshell = 4./3.*M_PI*(Rout*Rout*Rout - Rin*Rin*Rin);
           density[n] = number[n]*mpart/Vshell;
           sig_d[n] = sqrt(number[n])*mpart/Vshell;
           if(scaleErrors == 1)
           {
              sig_d[n] *= sqrt((double)(numberShells + numberCore)/(double)
minNumPart);
           }
        }
     }

     else // fixed number of particles per bin
     {
        qsort(radii, numberShells, sizeof(double *), compareDoubles);
        logrRangeMax = log10(radii[numberShells-1]);
        logrRangeMin = log10(radii[0]);
        for(n=0; n < numberShells/ppbin; n++)
        {
           int lastBin;
           if(n == numberShells/ppbin - 1)
              lastBin = 1;
           else
              lastBin = 0;

           double Rin = radii[n*ppbin];
           double Rout = radii[(n+1)*ppbin - 1*lastBin];
           double Vshell = 4./3.*M_PI*(Rout*Rout*Rout - Rin*Rin*Rin);
           density[n] = ppbin*mpart/Vshell;
           sig_d[n] = sqrt(ppbin)*mpart/Vshell;
           if(scaleErrors == 1)
           {
              sig_d[n] *= sqrt((double)(numberShells + numberCore)/(double)
minNumPart);
           }
        }
     }

     double Vcore = 4./3.*M_PI*pow(pow(10.0, logrRangeMin), 3.0);
     double densityCore =  (double)numberCore*mpart/Vcore;
     double sig_dCore = sqrt((double)numberCore)*mpart/Vcore;


/* Make a 1d array to hold values of chi^2. chisquared[c] */
     double chisquared[N_C];
     int goodfit[N_C];
```

```
      double cwidth = (cRangeMax-cRangeMin)/(double)N_C;

      double cmax = cRangeMin-1.0;  // dummy values
      double cmin = cRangeMax+1.0;
      double chimin = 12345678.0, cFIT = 12345678.0;
      int good = 0;
      int dof;
      int freeParam = 1;

      for(i=0; i < N_C; i++)
      {
             double c = cRangeMin + cwidth*i;
             chisquared[i] = 0.0;
             goodfit[i] = 0;
             dof = 0;

             if(BINS > 0)
             {
                for(n=0; n < BINS; n++)
                {
                    if(number[n] != 0) // skip bins that have no particles
                    {
                        double Rin = pow(10., logrRangeMin + logrWidth*((double)n));
                        double Rout = pow(10., logrRangeMin + logrWidth*((double)n +
1.));
                        chisquared[i] += pow(density[n] - getNFWDensity(Rin, Rout, c,
log10(M2)), 2.0)/pow(sig_d[n], 2.0);
                        dof++;
                    }
                }
             }

             else //fixed number of particles per bin
             {
                for(n=0; n < numberShells/ppbin; n++)
                {
                    int lastBin;
                    if(n == numberShells/ppbin - 1)
                        lastBin = 1;
                    else
                        lastBin = 0;

                    double Rin = radii[n*ppbin];
                    double Rout = radii[(n+1)*ppbin - 1*lastBin];
                    chisquared[i] += pow(density[n] - getNFWDensity(Rin, Rout, c,
log10(M2)), 2.0)/pow(sig_d[n], 2.0);
                    dof++;
                }
             }

             //contribution to chi^2 from inner average density
             chisquared[i] += pow(densityCore - getNFWDensity(0.0, pow(10.,
logrRangeMin), c, log10(M2)), 2.0)/pow(sig_dCore, 2.0);
             dof++;

             // mark as good fit, look for Mmax, cmax, Mmin, cmin
             if(chisquared[i]/(double)dof <= GOODCHI)
             {
                goodfit[i] = 1;

                if(c > cmax)
                    cmax = c;
                else if(c < cmin)
                    cmin = c;
```

```
            }

            // where is chisquared min? (even if there is no "good fit")
            if(chisquared[i] < chimin || i == 0)
            {
               chimin = chisquared[i];
               good = goodfit[i];
               cFIT = c;
            }
     } // chisquared minimization

     dof = dof - freeParam;

/* Print out results */
     if(singleHalo < 0) // print out for all halos
     {
        if(printfit == 1)
        {
           fprintf(fout, "%8d\t", gid);
           fprintf(fout, "%8d\t", npart);
           fprintf(fout, "%8.3f\t", chimin/(double)dof);
           fprintf(fout, "%7.4f\t", log10(M2));
           fprintf(fout, "%6.3f\t", cFIT);
           fprintf(fout, "%5.4f\t", R2);
           fprintf(fout, "%7.4f\t", log10(Mfof));
           fprintf(fout, "%7.4f\t", log10(M1));
           fprintf(fout, "%5.4f\t", R1);
           fprintf(fout, "%8.3f\t", chimin);
           fprintf(fout, "%8d\n", dof);
        }

        if(printgeo == 1)
        {
           fprintf(fout2, "%8d\t", gid);
           fprintf(fout2, "%8d\t", npart);
           for(k=0; k<3; k++)
              fprintf(fout2, "%9.4f\t", Ieig[k]/mpart);
           for(k=0; k<3; k++)
              fprintf(fout2, "%9.4f\t", axisRatios[k]);
           for(k=0; k<3; k++)
              fprintf(fout2, "%9.4f\t", xmbp[k]);
           for(k=0; k<3; k++)
              fprintf(fout2, "%9.4f\t", com[k]);
           for(k=0; k<3; k++)
              fprintf(fout2, "%9.4f\t", xdpp[k]);
           fprintf(fout2, "%5.4f\t", get_r(com, xmbp, hdr.BoxSize));
           fprintf(fout2, "%5.4f\n", get_r(xdpp, xmbp, hdr.BoxSize));
        }

        // This is the ratio of densities for the core bin
        //fprintf(fout, "%8.4f\n", densityCore/getNFWDensity(0.0, pow
(10.,logrRangeMin), cFIT, log10(M2)));
     }

     else // print out for only one halo
     {
        for(n=0; n < npart; n++)
        {
           pd1 = pdata[n];
           for(k=0; k<3; k++)  //printout particle positions
              fprintf(fout, "%8f\t", pd1.pos[k]);

           if(BINS > 0 && n < BINS)   //print out density information
           {
```

```
                // density profile
                fprintf(fout, "%8f\t", (logrRangeMin + logrWidth*((double)n +.5)));
                fprintf(fout, "%8f\t", log10(density[n]));
                fprintf(fout, "%8f\t", log10(sig_d[n]));
                double Rin = pow(10., logrRangeMin + logrWidth*((double)n));
                double Rout = pow(10., logrRangeMin + logrWidth*((double)n + 1.));
                fprintf(fout, "%8f\t", log10(getNFWDensity(Rin, Rout, cFIT, log10
  (M2))));
            }
            else if(BINS <= 0 && n < numberShells/ppbin)
            {
                  int lastBin;
                  if(n == numberShells/ppbin - 1)
                     lastBin = 1;
                  else
                     lastBin = 0;

                  double Rin = radii[n*ppbin];
                  double Rout = radii[(n+1)*ppbin - 1*lastBin];
                  fprintf(fout, "%8f\t", (log10(Rin) + log10(Rout))/2.0);
                  fprintf(fout, "%8f\t", log10(density[n]));
                  fprintf(fout, "%8f\t", log10(sig_d[n]));
                  fprintf(fout, "%8f\t", log10(getNFWDensity(Rin, Rout, cFIT, log10
  (M2))));
            }

            fprintf(fout, "\n");
         }

         fprintf(fout2, "%6.3f\t", cFIT);
         fprintf(fout2, "%7.4f\t", log10(M2));
         for(k=0; k<3; k++)
               fprintf(fout2, "%9.4f\t", xmbp[k]);
         fprintf(fout2, "%5.4f\t", R1);
         fprintf(fout2, "%5.4f\t", R2);
         fprintf(fout2, "%8.3f\t", chimin/(double)dof);
         fprintf(fout2, "%7.4f\t", log10(Mfof));
         fprintf(fout2, "%7.4f\n", log10(M1));
      }

      // This is the ratio of densities for the core bin
      //fprintf(fout, "%8.4f\n", densityCore/getNFWDensity(0.0, pow
  (10.,logrRangeMin), cFIT, log10(M2)));


    } // loops over all halos

    // free up malloc memory
  free (number);
  free (radii);
  free (density);
  free (sig_d);

    fclose(fin);
    fclose(fout);
    fclose(fout2);
    free (pdata);
    return hdr.ngroups;
}

// FUNCTION try_file
// checks that bgc_file opens properly then calls function process_halos
int try_file( char * bgc_file, char * centers_file )
{
```

```
    FILE * fp;
    OUTPUT_HEADER hdr;
    int ngroups_read = 0;

  fprintf(stderr, "Reading Center file: %s\n", centers_file);
    fprintf(stderr, "Reading BGC file: %s\n", bgc_file);
//     fprintf(stdout, "# from BGC file: %s\n", bgc_file);
    fflush(stderr);
    fp = fopen(bgc_file, "r");
    if(fp == NULL)
    {
        fprintf(stderr, "ERROR: problem opening file '%s'\n", bgc_file);
        assert(fp != NULL);
    }

    bgc_read_header(fp,&hdr);

    if(PDATA_FORMAT_PV == hdr.format)
    {
//         fprintf(stdout, "# 11 columns: gid(0) npart(1) group_mass(2) pos_com(3,4,5)
  vel_com(6,7,8) vdisp(9) periodic_needed(10)\n");

        ngroups_read = process_halos(fp, hdr, centers_file);
    }
    else if( PDATA_FORMAT_PVBE == hdr.format )
    {
//         ngroups_read = calc_stats_mbp(fp, hdr);
        fprintf(stderr, "USING BINDING ENERGY IS NOT YET IMPLEMENTED!\n");
        return(EXIT_FAILURE);
    }
    else
    {
        fprintf(stderr,"ERROR: skipping '%s' -- PDATA_FORMAT not compatible (%d)\n",
  bgc_file, hdr.format);
        return(EXIT_FAILURE);
    }

    return ngroups_read;
}

// FUNCTION: diagonalize
// Takes a 3x3 symmetric matrix S and diagonalizes it. Ixx, Iyy, Izz are the diagonal
  elements.
// From wiki Eigenvalues of a Symmetric 3x3 matrix: http://en.wikipedia.org/wiki/
  Eigenvalue_algorithm#Eigenvalues_of_a_Symmetric_3x3_Matrix
// Originally from Oliver K. Smith: Eigenvalues of a symmetric 3 × 3 matrix. Commun.
  ACM 4(4): 168 (1961)
void diagonalize(double S[3][3], double * Imax, double * Imin, double * Imed)
{
  int i,j;
  double m,q,p,detK,phi;
  double K[3][3];

  m = (S[0][0] + S[1][1] + S[2][2])/3.0;

  for(i=0; i<3; i++) {
       for(j=0; j<3; j++) {
          if (i == j)
             K[i][j] = S[i][j] - m;
          else
             K[i][j] = S[i][j];
       } }

  detK = ( K[0][0]*(K[1][1]*K[2][2] - K[1][2]*K[2][1]) - K[0][1]*(K[1][0]*K[2][2] - K
```

```
  [1][2]*K[2][0]) + K[0][2]*(K[1][0]*K[2][1] - K[1][1]*K[2][0]) );

  q = detK/2.0;

  p = 0.0;
  for(i=0; i<3; i++) {
       for(j=0; j<3; j++) {
       p += K[i][j]*K[i][j];
  } }
  p = p/6.0;

  phi = 1./3.*atan(sqrt(p*p*p - q*q)/q);

  if(phi < 0.)
       phi=phi+M_PI/3.0;   // this is equivalent to phi = 1/3*(atan(-#) + pi)

  *Imax = m + 2.0*sqrt(p)*cos(phi);
  *Imin = m - sqrt(p)*(cos(phi) + sqrt(3.)*sin(phi));
  *Imed = m - sqrt(p)*(cos(phi) - sqrt(3.)*sin(phi));
}

// FUNCTION get_r
// Returns the distance between two points (pos1(x,y,z) and pos2(x,y,z)), checks for
  periodic wrap
double get_r(double pos1[], double pos2[], double BoxSize)
{
  int k;
  double rsq = 0., d = 0., plength_check = BoxSize/PLENGTH;
       for(k=0; k<3; k++)
       {
          d = pos1[k] - pos2[k];

          if(d > plength_check)
             d = pos2[k] + BoxSize - pos1[k];

          else if(d < -1.0 * plength_check)
             d = pos1[k] + BoxSize - pos2[k];

          rsq += d*d;
       }
  return sqrt(rsq);
}

// FUNCTION: getNFWDensity
// Returns average density between R1 and R2 for an NFW with mass logM and
  concentration c.
double getNFWDensity(double R1, double R2, double c, double logM)
{
  // NFW
  // average density = 4*pi/Vbin*integral_R1^R2[denistyNFW*r^2dr]
  // = 4*pi/Vbin*B/A^3*[ln(AR+1/c) + 1/(cAR + 1)]|R1^R2

  double Vbin = 4./3.*M_PI*(R2*R2*R2 - R1*R1*R1);

  double A = pow(pow(10.,logM)/(4./3.*M_PI*delta_vir*avg_density),-1.0/3.0);
  double B = delta_vir*avg_density/3./(-c/(1.+c)+log(1.+c));
  double x = A*c*R2 + 1.;
  double y = A*c*R1 + 1.;

  return 4.*M_PI*B/(Vbin*A*A*A)*(log(x/y) + 1./x - 1./y); //log gives ln
}

// FUNCTION: newFileName
// 'longFile' is the filename we want to modify and is in the form [characters]/
```

```
  [middle characters].[more characters]
// [middle characters] cannot have any '/' or '.' in it.
// Changes 'file' to be a character array that is [middle_characters][extension]
// Be carefule to make sure that 'file' is long enough to hold all the characters and
  extension, or there will be problems...
// initializing file as: char * file[strlen(longFile) + strlen(extension)]; should
  guarantee good things
void newFileName(char * file, char * longFile, char * extension)
{
  int i, start = -1, end = strlen(longFile);
  for(i= 0; i < strlen(longFile); i++)
  {
       if(longFile[i] == '/')
          start = i;
  }
  for(i= strlen(longFile) - 1; i > start; i--)
  {
       if(longFile[i] == '.')
          end = i;
  }
  for(i= start+1; i < end; i++)
  {
       file[i-(start+1)] = longFile[i];
  }
  for(i= (end-start)-1; i < strlen(file); i++) //trims out any other characters that
  may have been left in 'file'
  {
       file[i] = '\0';
  }

  strcat(file, extension);
}

// FUNCTION: compareDoubles
// used in qsort to compare doubles
int compareDoubles (const void *A, const void *B)
{
   double a = *((double *)A);
   double b = *((double *)B);

   if (a < b)
       return -1;
   else if (a > b)
       return 1;
   else
       return 0;
}

// FUNCTION: findAxisRatios
// Takes moment of inertia eigenvalues (principle moments of inertia) and assumes the
  object is an ellipsiod to calculate axis ratios.
// For now, this also assumes that the density of the ellipsiod is constant (probably
  not the best assumption).
void findAxisRatios(double Ieig[3], double mass, double * major, double * medium,
  double * minor)
{
  *major  = sqrt(5./2.*(Ieig[0] + Ieig[2] - Ieig[1])/mass);
  *medium = sqrt(5./2.*(Ieig[0] + Ieig[1] - Ieig[2])/mass);
  *minor  = sqrt(5./2.*(Ieig[1] + Ieig[2] - Ieig[0])/mass);
}
```