Early Growth in a Perturbed Universe: Dark Matter Halo Properties in 2LPT and ZA Simulations

By

Daniel J. Sissom

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

PHYSICS

August, 2014

Nashville, TN

Approved: Date:

_____ _____

Jocelyn K. Holley-Bockelmann, Ph.D.

_____ _____

Andreas A. Berlind, Ph.D.

_____ _____

David A. Weintraub, Ph.D.

_____ _____

Shane M. Hutson, Ph.D.

_____ _____

Robert J. Scherrer, Ph.D.

ACKNOWLEDGMENTS

This is where you thank the people that made your work possible: grant awarding agencies, advisers, your committee, mom and dad, whatever.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

Introduction

Text goes here. This is where we'll talk about the purpose of the project and the layout of this document.

The structure of this document is as follows: The remainder of this chapter, Chapter I, provides an introduction to the early universe and the processes that lead to galaxy-hosting dark matter halos, as well as the fundamentals of the computational theory for the numerical methods relevant to this discussion. Chapter II examines in more detail the specific numerical methods used for this work, with emphasis on the methodologies of the codes themselves, how they are implemented in the context of the overall simulation and analysis pipeline, and the results obtained at each step. Chapter III is a direct representation of the published paper which (more succinctly) presents an overview of the numerical methods and the main results in this work. Chapter IV is primarily the same material as previously submitted to fulfill the requirements of the Qualifying Exam, and is slightly edited to better suit the tone of this document. Chapter V concludes with a discussion of the results in this work and the greater implications to the overall field.

I.1 The Early Universe

Text goes here.

I.1.1 The CMB Epoch

Text goes here.

I.1.1.1 Recombination

Text goes here.

I.1.1.2 The Cosmic Microwave Background

Text goes here.

I.1.2 Dark Matter Halo Formation

Text goes here.

I.1.2.1 Collapse

Text goes here.

I.1.2.2 Accretion

Text goes here.

I.1.2.3 Mergers

Text goes here.

I.1.2.4 Large-scale Structure

Text goes here.

I.1.3 Halo Properties

Text goes here.

I.1.3.1 Mass

There are a number of ways to define a halo's mass. This becomes significant for mass-sensitive studies, such as the halo mass function (Press & Schechter, 1974; Reed et al., 2007; Heitmann et al., 2006; Lukić et al., 2007), the number density of halos as a function of mass and a key probe of cosmology. For a review, see, e.g., White (2001) and references therein. Additionally, see Voit (2005) and references therein for a more observation-focused discussion.

$$M_{\rm vir} \equiv \frac{4\pi}{3}\Delta_{\rm vir}\rho_m R_{\rm vir}^3, \tag{I.1}$$

where $\rho_m = \Omega_{\rm M}\rho_c$.

$$V_{\rm circ} = \sqrt{\frac{GM(<r)}{r}}\Bigg|_{\rm max} \tag{I.2}$$

I.1.3.2 Density and Concentration

The halo density profile is a measure of the spherically-averaged dark matter density as a function of radius. For numerical halos in N-body simulations, the density profile is typically computed by dividing the member particles into logarithmically-spaced bins from the virial radius inward towards the center, summing the mass of the particles in each bin, and dividing by the volume of the shell to find the density.

DM halos almost universally display a characteristic shape in their density profiles. This shape is most often parameterized with the Navarro-Frenk-White (NFW) profile (Navarro et al., 1996):

$$\rho(r) = \frac{\rho_0}{\frac{r}{R_s}\left(1+\frac{r}{R_s}\right)^2}, \tag{I.3}$$

where ρ_0 is the characteristic density and R_s is the scale radius where the inner $\sim r^{-1}$ profile transitions to the outer $\sim r^{-3}$ profile.

Halo concentration c provides a single-parameter quantization of the density profile. For the NFW profile, concentration is defined as $c \equiv R_{\rm vir}/R_s$, where $R_{\rm vir}$ is the halo virial radius. Generally, at low redshift, low mass halos are more dense than high mass halos (Navarro et al., 1997a), and concentration decreases with redshift and increases in dense environments (Bullock et al., 2001b). Neto et al. (2007) additionally find that concentration decreases with halo mass. Various additional studies have explored concentration's dependence on characteristics of the power spectrum (Eke et al., 2001), cosmological model (Macciò et al., 2008), redshift (Gao et al.,

2008; Muñoz-Cuartas et al., 2011), and halo merger and mass accretion histories (Wechsler et al., 2002; Zhao et al., 2003, 2009). For halos at high redshift, Klypin et al. (2011) find that concentration reverses and increases with mass for high mass halos, while Prada et al. (2012) find that concentration's dependence on mass and redshift is more complicated and is better described through $\sigma(M, z)$, the rms fluctuation amplitude of the linear density field.

Concentration may be estimated from a halo's virial mass $M_{\rm vir}$ and maximum circular velocity $V_{\rm circ}$. Following Klypin et al. (2011), we outline this relationship for $z = 0$ and as a function of redshift. The relation between the virial mass and maximum circular velocity may be given as (Klypin et al., 2001):

$$V_{\rm circ} = \left[G \frac{f(x_{\rm max})}{f(c)} \frac{c}{x_{\rm max}} \hat{\rho}^{1/3} \right]^{1/2} {M_{\rm vir}}^{1/3}, \tag{I.4}$$

$$\hat{\rho} = \frac{M_{\rm vir}}{{R_{\rm vir}}^3} = \frac{4\pi}{3} \Delta_{\rm vir} \rho_c \Omega_{\rm M}, \tag{I.5}$$

$$f(x) = \ln(1 + x) - \frac{x}{1 + x}, \tag{I.6}$$

where $x = r/R_s$, $x_{\rm max} = 2.15$, $\Delta_{\rm vir}$ is the overdensity limit that defines the virial radius, ρ_c is the critical density, and $\Omega_{\rm M}$ is the matter contribution to the average density of the universe. At $z = 0$, $\Delta_{\rm vir} = 360$ and $\Omega_{\rm M} = 0.27$, which yields

$$V_{\rm circ}(M_{\rm vir}) = \frac{6.72 \times 10^{-3} {M_{\rm vir}}^{1/3} \sqrt{c}}{\sqrt{\ln(1 + c) - c/(1 + c)}} \tag{I.7}$$

for $M_{\rm vir}$ in units of $h^{-1}{\rm M}_\odot$ and $V_{\rm circ}$ in units of km s^{-1}. Klypin et al. (2011) find that at $z = 0$, this yields the approximation

$$c(M_{\rm vir}) = 9.60 \left(\frac{M_{\rm vir}}{10^{12} h^{-1} {\rm M}_\odot} \right)^{-0.075} \tag{I.8}$$

for distinct halos and

$$c(M_{\rm sub}) = 12\left(\frac{M_{\rm sub}}{10^{12}h^{-1}{\rm M}_\odot}\right)^{-0.12} \tag{I.9}$$

for subhalos. Figure I.1 plots concentration as a function of virial mass from $z = 0$ to $z = 5$. The dotted lines are given by

$$c(M_{\rm vir}, z) = c_0(z)\left(\frac{M_{\rm vir}}{10^{12}h^{-1}{\rm M}_\odot}\right)^{-0.075} \times \left[1 + \left(\frac{M_{\rm vir}}{M_0(z)}\right)^{0.26}\right], \tag{I.10}$$

where $c_0(z)$ and $M_0(z)$ are free parameters for each z. Concentration displays a decreasing trend with mass at low redshift. At higher redshift, however, concentration flattens out and reverses its trend, increasing with mass for the most massive halos.

Figure I.2 plots concentration as a function of redshift for two representative halo masses. For a given fixed halo mass, concentration decreases with redshift for low redshift, then increases again with redshift at high redshift. The black curves are given by

$$c(M_{\rm vir}, z) = c(M_{\rm vir}, 0)[\delta^{4/3}(z) + \kappa(\delta^{-1}(z) - 1)], \tag{I.11}$$

where $\delta(z)$ is the linear growth factor of fluctuations normalized to $\delta(0) = 1$ and κ is a free parameter. For the masses shown in the figure, $\kappa = 0.084$ for $M = 3\times10^{11}h^{-1}{\rm M}_\odot$ and $\kappa = 0.135$ for $M = 3 \times 10^{12}h^{-1}{\rm M}_\odot$.

Using the same method of determining concentration from halo virial mass and maximum circular velocity, Prada et al. (2012) find that the complex mass and redshift dependence of concentration found by Klypin et al. (2011) may be simplified to a universal U-shaped profile when viewed as a function of the linear rms fluctuation of the density field $\sigma(M, z)$. Figure I.3 plots c as a function of $\log\sigma^{-1}$ for redshifts from $z = 0$ to $z = 6$ for halos from the Bolshoi (Klypin et al., 2011) and MultiDark

(Prada et al., 2012) simulations. If we define

$$x \equiv \left(\frac{\Omega_{M,0}}{\Omega_{\Lambda,0}}\right)^{1/3} a, \tag{I.12}$$

$$a \equiv (1+z)^{-1} \tag{I.13}$$

where $\Omega_{M,0}$ and $\Omega_{\Lambda,0}$ are the matter and cosmological constant contributions to the density of the universe at $z = 0$, then the overplotted curve is given by

$$c(M,z) = B_0(x)C(\sigma'), \tag{I.14}$$

$$\sigma' = B_1(x)\sigma(M,x), \tag{I.15}$$

$$C(\sigma') = A\left[\left(\frac{\sigma'}{b}\right)^c + 1\right]\exp\left(\frac{d}{\sigma'^2}\right), \tag{I.16}$$

where $A = 2.881$, $b = 1.257$, $c = 1.022$, and $d = 0.060$. The rms density fluctuation may be approximated as

$$\sigma(M,x) = D(x)\frac{16.9y^{0.41}}{1 + 1.102y^{0.20} + 6.22y^{0.333}}, \tag{I.17}$$

where

$$y \equiv \left[\frac{M}{10^{12}\ h^{-1}\ \mathrm{M}_\odot}\right]^{-1}, \tag{I.18}$$

$$D(x) = \frac{5}{2}\left(\frac{\Omega_{M,0}}{\Omega_{\Lambda,0}}\right)^{1/3}\frac{\sqrt{1+x^3}}{x^{3/2}}\int_0^x \frac{x^{3/2}\,\mathrm{d}x}{(1+x^3)^{3/2}}. \tag{I.19}$$

The functions $B_0(x)$ and $B_1(x)$ are defined such that they equal unity at $z = 0$ for WMAP5 parameters:

$$B_0(x) = \frac{c_{\min}(x)}{c_{\min}(1.393)}, \tag{I.20}$$

$$B_1(x) = \frac{\sigma^{-1}_{\min}(x)}{\sigma^{-1}_{\min}(1.393)}, \tag{I.21}$$

where

$$c_{\rm min}(x) = c_0 + (c_1 - c_0)\left[\frac{1}{\pi}\arctan[\alpha(x - x_0)] + \frac{1}{2}\right], \tag{I.22}$$

$$\sigma_{\rm min}^{-1}(x) = \sigma_0^{-1} + (\sigma_1^{-1} - \sigma_0^{-1})\left[\frac{1}{\pi}\arctan[\beta(x - x_1)] + \frac{1}{2}\right], \tag{I.23}$$

$$c_0 = 3.618,\ c_1 = 5.033,\ \alpha = 6.948,\ x_0 = 0.424, \tag{I.24}$$

$$\sigma_0^{-1} = 1.047,\ \sigma_1^{-1} = 1.646,\ \beta = 7.386,\ x_1 = 0.526. \tag{I.25}$$

The resulting curve closely follows the data at all redshifts from $z = 0$ to $z = 6$, with a minimum concentration of ~ 5 at a well-defined scale of $\sigma \sim 0.71$. The relation may also be seen as a function of mass without rescaling to $z = 0$ by plotting Equations I.14-I.16, as shown in Figure I.4.

I.1.3.3 Substructure and Environment

Text goes here.

I.1.4 Baryonic Processes

Early-forming dark matter halos provide an incubator for the baryonic processes that transform the surrounding space and allow galaxies to form. Initial gas accretion can lead to the formation of the first Pop-III stars (Couchman & Rees, 1986; Tegmark et al., 1997; Abel et al., 2000, 2002), which, upon their death, can collapse into the seeds for supermassive black holes (SMBHs) (Madau & Rees, 2001; Islam et al., 2003; Alvarez et al., 2009; Jeon et al., 2012) or enrich the surrounding medium with metals through supernovae (Heger & Woosley, 2002; Heger et al., 2003). The radiation from these early quasars (Shapiro & Giroux, 1987; Madau et al., 1999; Fan et al., 2001), Pop-III stars (Gnedin & Ostriker, 1997; Venkatesan et al., 2003; Alvarez et al., 2006), and proto-galaxy stellar populations (Bouwens et al., 2012; Kuhlen & Faucher-2012) all play a key role in contributing to the re-ionizing the universe by around $z = 6$ (Barkana & Loeb, 2001). Additionally, halo mergers can drastically increase

the temperature of halo gas through shock heating, increasing X-ray luminosity (Sinha & Holley-Bockelmann, 2009), and contribute to the unbinding of gas to form the warm-hot intergalactic medium (Bykov et al., 2008; Sinha & Holley-Bockelmann, 2010; Tanaka et al., 2012).

I.1.4.1 The First Stars

Text goes here.

I.1.4.2 Supermassive Black Holes

Text goes here.

I.1.4.3 Enrichment and the The Intergalactic Medium

Text goes here.

I.1.4.4 Reionization

Text goes here.

I.2 Computational Theory

In this section, we present a broad overview of the fundamental theory and driving equations of computational astrophysics that are relevant to this work. Specific code implementations, such as the N-body simulation code GADGET-2 and the halo finder ROCKSTAR, are discussed in Chapter II, so here we instead focus on the mathematical concepts that form the basis these codes rely on and have in common with varied other implementations. Specifically, in this section, we discuss collisionless dynamics in N-body simulations, simulation initialization with the Zel'dovich approximation (ZA) and second-order Lagrangian perturbation theory (2LPT), and numerical definitions of dark matter halos. As the simulations used in our study are of collisionless dark matter only, we forgo a discussion of collisional hydrodynamics.

I.2.1 Collisionless Dynamics and N-body Simulations

Astrophysical simulations of stars or dark matter, in essence, track a collisionless fluid, which is described in the continuum limit by the collisionless Boltzmann equation (CBE)

$$\frac{\mathrm{d}f(\mathbf{x},\mathbf{v},t)}{\mathrm{d}t} \equiv \frac{\partial f}{\partial t} + \mathbf{v}\cdot\frac{\partial f}{\partial \mathbf{x}} - \frac{\partial \Phi}{\partial \mathbf{x}}\cdot\frac{\partial f}{\partial \mathbf{v}} = 0 \tag{I.26}$$

coupled to the Poisson equation

$$\nabla^2\Phi(\mathbf{x},t) = 4\pi G\int f(\mathbf{x},\mathbf{v},t)\ \mathrm{d}\mathbf{v} \tag{I.27}$$

in an expanding background Universe, typically according to the Friedmann-Lemaître-Robertson-Walker metric. Here, Φ is the self consistent potential, and the distribution function $f(\mathbf{x},\mathbf{v},t)$ gives the mass density in phase space. The high-dimensionality of the problem, however, makes directly solving the coupled system of equations intractable. Instead, the N-body method, in which the phase-space density is sampled with a finite number N of tracer particles, is used to evolve the system in time. For the following discussion, we primarily follow the notation in Springel (2005).

The system of particles is described by the Hamiltonian

$$H(\mathbf{x}_1,\ldots,\mathbf{x}_N,\mathbf{p}_1,\ldots,\mathbf{p}_N,t) = \sum_i \frac{\mathbf{p}_i^2}{2m_i a(t)^2} + \frac{1}{2}\sum_{ij}\frac{m_i m_j \varphi(\mathbf{x}_i-\mathbf{x}_j)}{a(t)}, \tag{I.28}$$

where $\mathbf{x}_i$ are comoving coordinate vectors with corresponding canonical momenta $p_i = a^2 m_i \dot{\mathbf{x}}_i$ and $a(t)$ is the time evolution of the scale factor that introduces explicit time dependence to the Hamiltonian. For simulations with periodic boundary conditions, the interaction potential $\varphi(\mathbf{x})$ for a cube of size L^3 is the solution of

$$\nabla^2\varphi(x) = 4\pi G\left[-\frac{1}{L^3} + \sum_{\mathbf{n}}\tilde{\delta}(\mathbf{x}-\mathbf{n}L)\right], \tag{I.29}$$

where the sum over $\mathbf{n} = (n_1, n_2, n_3)$ iterates over all integer triplets. Here, the mean

density is subtracted, and the dynamics of the system follow

$$\nabla^2\phi(\mathbf{x}) = 4\pi G[\rho(\mathbf{x}) - \bar{\rho}] \tag{I.30}$$

with peculiar potential

$$\phi(x) = \sum_i m_i\varphi(\mathbf{x} - \mathbf{x}_i). \tag{I.31}$$

For non-periodic (vacuum) boundary conditions, the interaction potential for point masses simplifies to

$$\varphi(\mathbf{x}) = -\frac{G}{|\mathbf{x}|} \tag{I.32}$$

for large separations.

At small particle separations as $|\mathbf{x}_i - \mathbf{x}_j| \to 0$, particle accelerations computed via the standard force law

$$\mathbf{a}_i = -\sum_{j\neq i} \frac{Gm_j|\mathbf{x}_i - \mathbf{x}_j|}{|\mathbf{x}_i - \mathbf{x}_j|^3} \tag{I.33}$$

approach a numerical singularity that can introduce unphysical results for finite time-steps. To avoid this scenario, numerical simulations employ a softening parameter $\epsilon > 0$ in the force law so that it does not diverge for small particle separations. As a simple example, the softening parameter may be added to the particle displacement in the denominator of the Newtonian force law:

$$\mathbf{F}_i = -\sum_{j\neq i} \frac{Gm_im_j|\mathbf{x}_i - \mathbf{x}_j|}{(|\mathbf{x}_i - \mathbf{x}_j|^2 + \epsilon^2)^{3/2}}. \tag{I.34}$$

More generally, the single particle density distribution function $\tilde{\delta}(\mathrm{x})$ of Equation I.29 is the Dirac δ-function convolved with a gravitational softening kernel of comoving scale ϵ. The specific choice of softening is dependent on the type of simulation and the system of study. The softening parameter is typically on the order of the mean inter-particle separation.

Directly calculating forces for every particle from every other particle inherently requires a double sum, implying a computational cost of $\mathcal{O}(N^2)$ algorithm complexity scaling. For large N, this quickly becomes computationally expensive. While the accuracy afforded by direct summation is sometimes necessary, such as for collisional systems like high-density star clusters, most studies can tolerate random force errors up to $\sim 1\%$ (Hernquist et al., 1993), introducing the possibility of approximation methods. There are a number of implementations for force approximations, but a typical result is a reduction of algorithmic complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. The specific implementation employed by GADGET-2 is discussed in Section II.2.1.

I.2.2 Simulation Initialization

Text goes here.

I.2.2.1 Initial Conditions and the Surface of Last Scattering

Text goes here.

I.2.2.2 The Zel'dovich Approximation

Text goes here.

I.2.2.3 Second-order Lagrangian Perturbation Theory

Text goes here.

$$\mathbf{x} = \mathbf{q} + \mathbf{\Psi}(\mathbf{q}) \tag{I.35}$$

$$\frac{\mathrm{d}^x}{\mathrm{d}r^2} + \mathcal{H}(r)\frac{\mathrm{d}x}{\mathrm{d}r} = -\nabla\Phi \tag{I.36}$$

$$\mathbf{J}(\mathbf{q},\tau)\nabla \cdot \left[\frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}\tau^2} + \mathcal{H}(\tau)\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\tau}\right] = \frac{3}{2}\Omega\mathcal{H}^2(J-1) \tag{I.37}$$

$$\nabla_{\mathbf{q}} \cdot \mathbf{\Psi}^{(1)} = -D_1(\tau)\delta(\mathbf{q}) \tag{I.38}$$

$$\nabla_{\mathbf{q}} \cdot \mathbf{\Psi}^{(2)} = \frac{1}{2} D_2(\tau) \sum_{i \neq j} \left[\mathbf{\Psi}^{(1)}_{i,i} \mathbf{\Psi}^{(1)}_{j,j} - \mathbf{\Psi}^{(1)}_{i,j} \mathbf{\Psi}^{(1)}_{j,i} \right] \tag{I.39}$$

I.2.3 Dark Matter Halos in N-body Simulations

Text goes here.

I.2.3.1 Spherical Overdensity

Text goes here.

I.2.3.2 Friends-of-Friends

Text goes here.

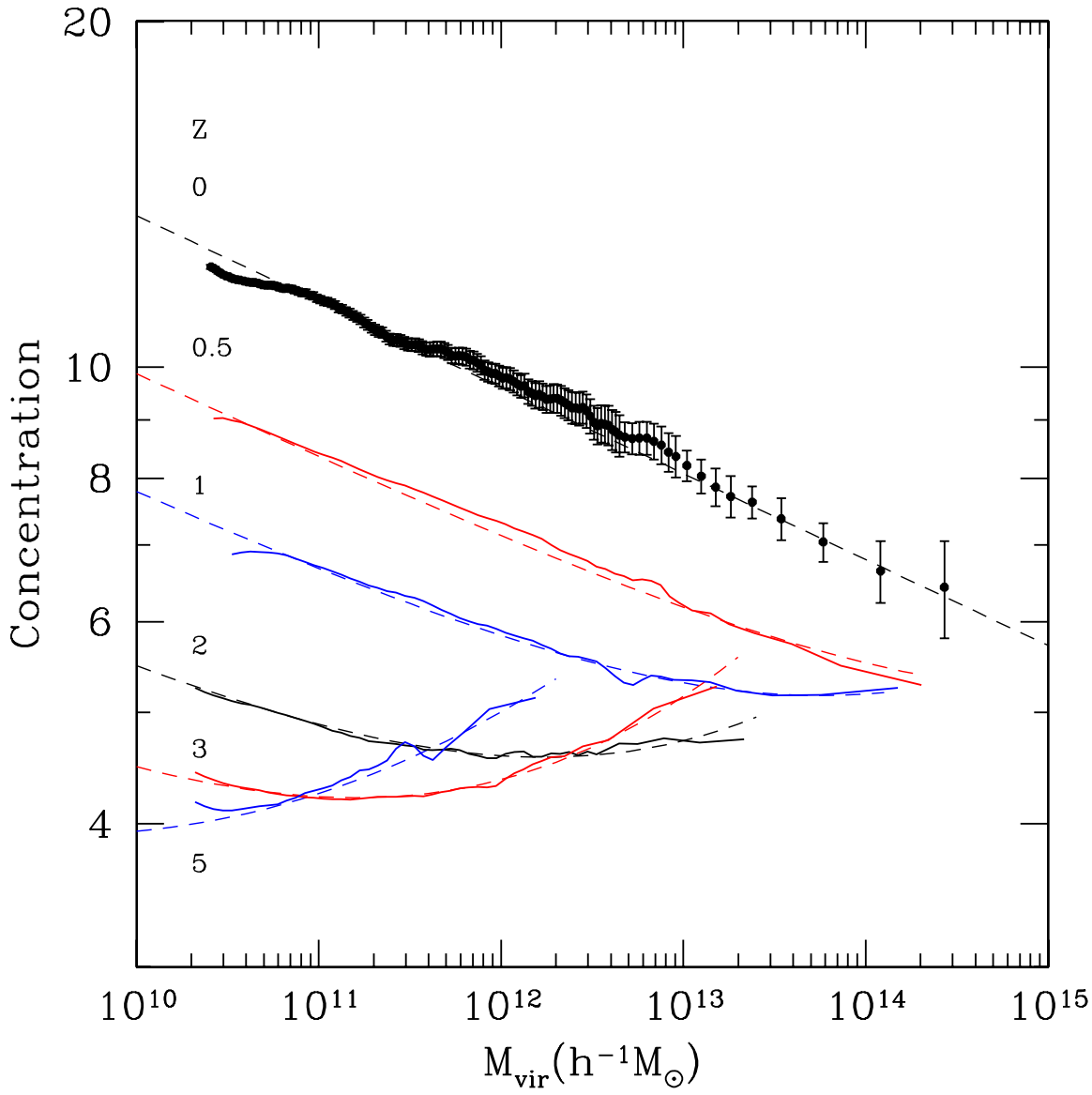


Figure I.1: Concentration as a function of virial mass for distinct halos from $z = 0$ to $z = 5$. Symbols and solid curves are numerical results, while the dashed curves are analytical fits (Equation I.10). Concentration decreases with increasing mass except for high-mass halos at high redshift, for which the concentration flattens and increases with mass. (Klypin et al., 2011)

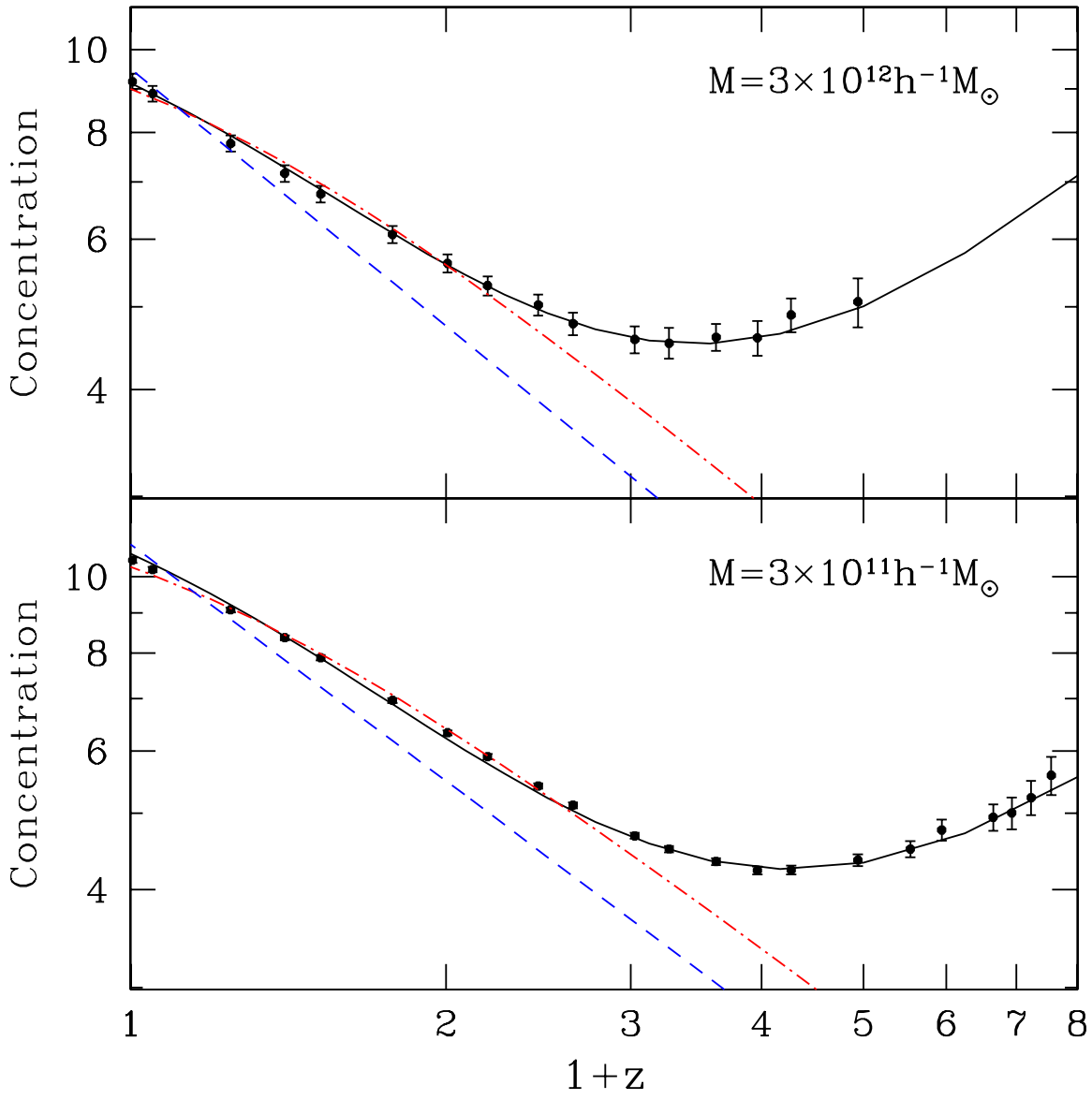


Figure I.2: Concentration as a function of redshift for two representative halo masses. Black dots are simulation results. The dashed blue curves show the power law $c \propto (1+z)^{-1}$ and the dot-dashed red curves are $c \propto \delta$. The solid black curves are given by Equation I.11. Concentration initially decreases with redshift, but reverses and increases with redshift for high redshift. Concentration for both masses reaches a minimum of $c_{\rm min} \approx 4 - 4.5$. (Klypin et al., 2011)

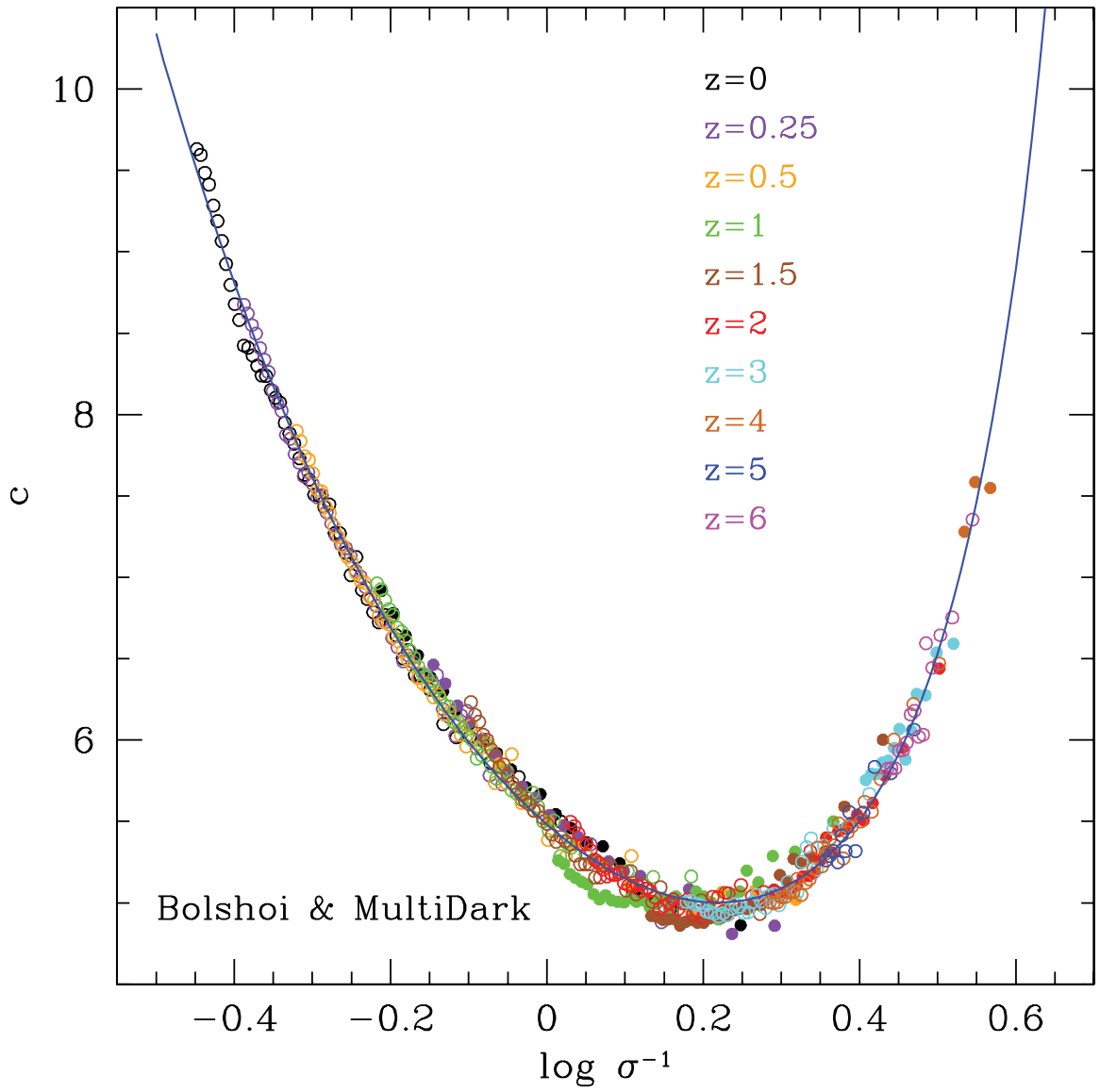


Figure I.3: Halo concentration c as a function of $\log \sigma^{-1}$ for halos in the Bolshoi and MultiDark simulations. The results are rescaled to $z = 0$. The solid curve $C(\sigma')$ is given by Equation I.16. A universal minimum concentration of ~ 5 is seen at $\sigma \sim 0.71$. (Prada et al., 2012)

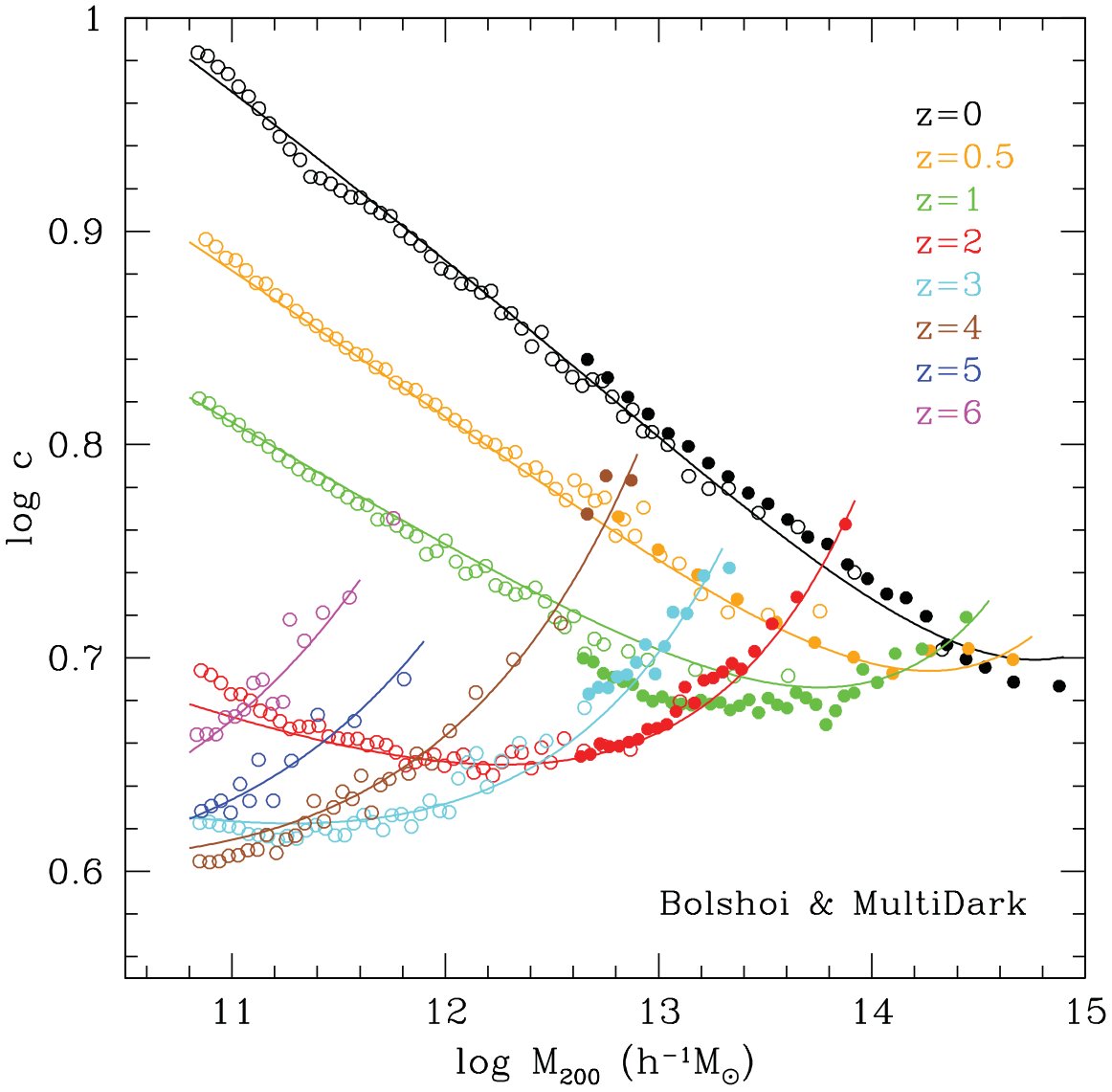


Figure I.4: Halo concentration c as a function of halo mass at various redshifts for halos in the Bolshoi (open circles) and MultiDark (filled circles) simulations. The overplotted curves are given by Equations I.14-I.16. The analytical approximations fit the data within a few percent. (Prada et al., 2012)

CHAPTER II

Numerical Methods

Text goes here. Here, we will discuss the computational tools used, their inner workings, and how they are implemented to accomplish their purpose in the pipeline.

II.1 Initialization Code

Text goes here.

II.1.1 Sampling the Power Spectrum

Text goes here.

II.1.1.1 Cosmological Parameters

Text goes here.

II.1.1.2 Sampling

Text goes here.

II.1.2 Particle Displacement with za

Text goes here.

II.1.3 Particle Displacement with 2lpt

Text goes here.

II.2 Simulations with Gadget-2

We use the massively parallel TreeSPH (Hernquist & Katz, 1989) cosmological N-body simulation code GADGET-2 (Springel et al., 2001; Springel, 2005) for the dark matter simulations presented in this work. In this section, we give an overview of the fundamentals of the GADGET-2 code, followed by details of our particular simulations.

II.2.1 Gadget-2

GADGET-2 is a massively parallel cosmological N-body simulation code which calculates gravitational forces via a hierarchical multipole expansion and ideal gas parameters via smoothed particle hydrodynamics (SPH; Gingold & Monaghan, 1977). This section will discuss the gravitational algorithms used to compute forces and the time integration method used to advance the simulation. As our simulations are collisionless only, we do not discuss the details of the implementation of gas dynamics in GADGET-2.

II.2.1.1 Gravitational Algorithms

Force computation suffers from a numerical singularity as the separation between two particles approaches zero, as discussed in Section I.2.1. A modification of the force law is therefore required at small separation scales. Force softening is accomplished in GADGET-2 using a spline kernel (Monaghan & Lattanzio, 1985) $W(|x|, h = 2.8\epsilon)$, where

$$W(r,h) = \frac{8}{\pi h^3} \begin{cases} 1 - 6\left(\frac{r}{h}\right)^2 + 6\left(\frac{r}{h}\right)^3, & 0 \leq \frac{r}{h} \leq \frac{1}{2} \\ 2\left(1 - \frac{r}{h}\right)^3, & \frac{1}{2} < \frac{r}{h} \leq 1, \\ 0, & \frac{r}{h} > 1. \end{cases} \tag{II.1}$$

An example of this softening is shown in Figure II.1 for the potential and force.

As discussed in Section I.2.1, direct summation N-body techniques are prohibitively slow for modern simulations. GADGET-2 therefore makes use of a hierarchical multipole expansion technique, often called a "tree" algorithm, using the Barnes-Hut octal tree (Barnes & Hut, 1986) algorithm. This method recursively divides the simulation volume into eight cells at each level of refinement, continuing the division until each cell contains only one particle. A visual description of this process in two dimensions is given in Figure II.2. Distant particles can then be grouped together for the force calculation, reducing the algorithm complexity to $\mathcal{O}(N \log N)$.

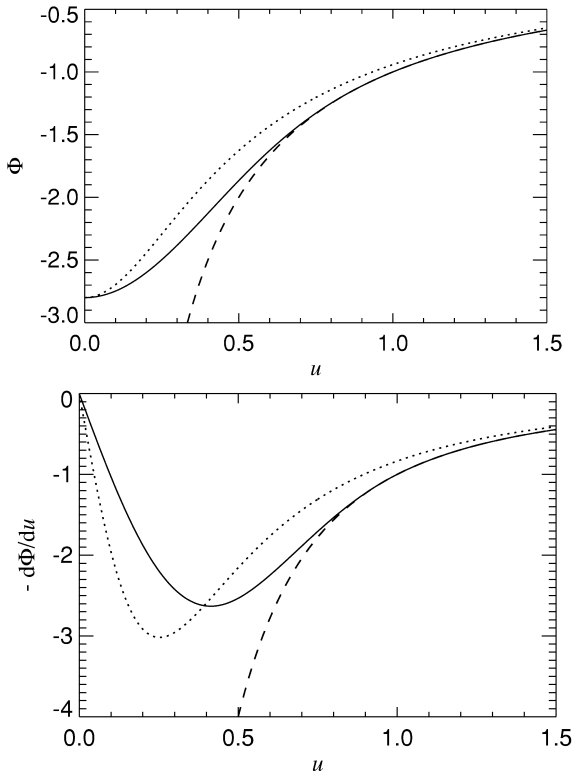


Figure II.1: Potential (*left*) and force (*right*) softening. The solid curves are the spline softening of Equation II.1. Curves for Plummer softening (dotted) and Newton's law (dashed) are provided for comparison. Here, $h = 1.0$ and $\epsilon = h/2.8$. Figure from Springel et al. (2001).

The Barnes-Hut octal tree algorithm begins with a cubic cell encompassing the entire simulation volume. The cell is then divided into eight daughter cells. If the cell contains no particles, it is ignored. If it contains one particle, the dividing process for that cell ends there. If it contains more than one particle, the process continues recursively, dividing daughter cells into eight octants each, until each cell contains either one or no particles. A multipole expansion of all daughter cells is then found for each node, or "leaf."

The accuracy of the force computation can be set by choosing how far to "walk" the tree. For each particle, the goal is to calculate the gravitational accelerations from all other particles accurately and quickly. There is a trade off, however, as increasing the accuracy of the tree code toward that of a direct summation approach also increases

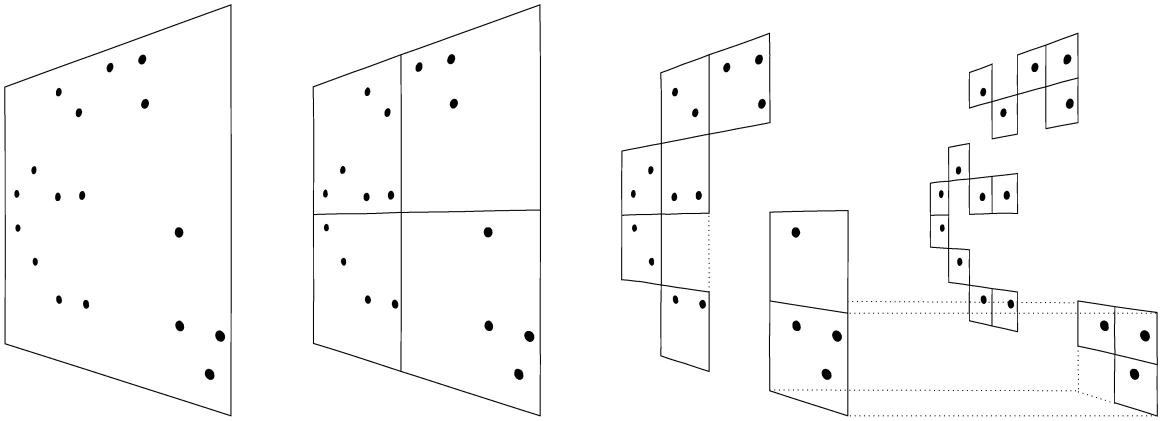


Figure II.2: Barns-Hut oct-tree in two dimensions. The simulation volume is recursively partitioned into cells until each contains only one particle each. Empty cells may be ignored. Figure from Springel et al. (2001).

the runtime complexity toward that of an $\mathcal{O}(N^2)$ algorithm. The balance between runtime and accuracy is controlled by the opening angle parameter α. A node of mass M and extension l will be considered for usage if

$$\frac{GM}{r^2}\left(\frac{l}{r}\right)^2 \leq \alpha|a|, \tag{II.2}$$

where r is the distance from the particle to the node and $|a|$ is the total acceleration from the previous time-step. Nodes that are massive, large, and near enough to fall outside this criterion are opened so that the daughter cells are recursively considered.

GADGET-2 can optionally make use of a hybrid approach for calculating forces, called the TreePM method (Xu, 1995; Bode et al., 2000; Bagla, 2002), where long range forces are computed using a particle-mesh algorithm instead of the Barnes-Hut octal tree. The GADGET-2 implementation of TreePM follows that of Bagla & Ray (2003).

II.2.1.2 Time Integration

The N-body Hamiltonian is separable such that $H = H_{\rm kin} + H_{\rm pot}$. Time evolution operators for each of $H_{\rm kin}$ and $H_{\rm pot}$ may be computed exactly, leading to "drift" and

"kick" operators (Quinn et al., 1997):

$$D_t(\Delta t) : \begin{cases} p_i & \mapsto & p_i, \\ x_i & \mapsto & x_i + \dfrac{p_i}{m_i} \displaystyle\int_t^{t+\Delta t} \dfrac{\mathrm{d}t}{a^2}, \end{cases} \tag{II.3}$$

$$K_t(\Delta t) : \begin{cases} x_i & \mapsto & x_i, \\ p_i & \mapsto & p_i + f_i \displaystyle\int_t^{t+\Delta t} \dfrac{\mathrm{d}t}{a}, \end{cases} \tag{II.4}$$

where

$$f_i = -\sum_j m_i m_j \frac{\partial \phi(x_{ij})}{\partial x_i} \tag{II.5}$$

is the force in particle i.

A time evolution operator $U(\Delta t)$ for an interval Δt may be approximated by combining the above two operators, where each fall a half time-step after the previous operation:

$$\tilde{U}(\Delta t) = D\left(\frac{\Delta t}{2}\right) K(\Delta t) D\left(\frac{\Delta t}{2}\right), \tag{II.6}$$

or

$$\tilde{U}(\Delta t) = K\left(\frac{\Delta t}{2}\right) D(\Delta t) K\left(\frac{\Delta t}{2}\right), \tag{II.7}$$

which gives us a leapfrog integrator constructed as a drift-kick-drift (DKD) or kick-drift-kick (KDK) operator. DKD and KDK are symplectic and time reversible, as both D_i and K_i are symplectic.

Cosmological simulations inherently contain a large dynamic range in time scales. Maintaining a constant time-step would be computationally prohibitive and wasteful, as high-density regions like the centers of galaxies require orders of magnitude smaller time-steps than low-density regions like the intergalactic medium. GADGET-2 therefore uses adaptive individual time-steps which are much more computationally

efficient. The time-step criterion for collisionless particles is

$$\Delta t_{\rm grav} = \min\left[\Delta t_{\rm max}, \left(\frac{2\eta\epsilon}{|a|}\right)^{1/2}\right], \tag{II.8}$$

where η is an accuracy parameter, ϵ is the gravitational softening, and a is the particle's acceleration. The maximum allowed timestep is $\Delta t_{\rm max}$, which is usually chosen to be a small fraction of the dynamical time of the system.

GADGET-2 allows particles to take on time-steps as a power of two subdivision of a global time-step. A particle is allowed to move to a smaller time-step at any time. However, moving to a larger time-step is only allowed on every second iteration and when this would lead to synchronization with the higher time-step hierarchy.

II.2.2 Simulations

We use GADGET-2 to evolve six dark matter–only cosmological volumes from $z_{start} = 300$ to $z = 6$ in a ΛCDM universe. Each simulation is initialized using WMAP-5 (Komatsu et al., 2009) parameters. For each of the three simulation pairs, we directly compare 2LPT and ZA by identically sampling the CMB transfer function and displacing the initial particle positions to the same starting redshift using 2LPT and ZA. The three sets of simulations differ only by the initial phase sampling random seed. Each volume contains 512^3 particles in a 10 h^{-1} Mpc box.

Following Heitmann et al. (2010), we choose conservative simulation parameters in order to ensure high accuracy in integrating the particle positions and velocities. We have force accuracy of 0.002, integration accuracy of 0.00125, and softening of 0.5 h^{-1} kpc, or 1/40 of the initial mean particle separation. We use a uniform particle mass of $5.3 \times 10^5 h^{-1}{\rm M}_\odot$. We select PMGRID, which defines the Fourier grid, to be 1024, SMTH, which defines the split between short- and long-range forces, to be 1.5 times the mesh cell size, and RCUT, which controls the maximum radius for short-range forces, to be 6.0 times the mesh cell size.

II.3 Halo Finding with Rockstar

ROCKSTAR (Robust Overdensity Calculation using K-Space Topoloogically Adaptive Refinement; Behroozi et al., 2013) is a halo finder based on the hierarchical refinement of friends-of-friends (FOF) groups in six phase space dimensions and, optionally, one time dimensions. It has been shown (Knebe et al., 2011) to be robust in recovering halo properties, determining substructure, and providing accurate particle member lists, even for notoriously difficult scenarios such as for low particle count halos and halos undergoing major merger events.

II.3.1 Halo Finding

Halo finding in ROCKSTAR is broken down into a number of steps, leading from the particle distribution of a simulation snapshot to the recovery of individual halo properties. FOF overdensity groups are distributed among the analysis processors which build hierarchies of FOF subgroups in phase space, determine particle membership for halos, compute host halo/subhalo relationships, remove unbounded particles, and compute halo properties. A summary of each of these steps is provided below.

II.3.1.1 FOF Groups

The 3D friends-of-friends algorithm groups particles together if they fall within a set linking length of each other. The linking length is often chosen as a fraction b of the mean interparticle distance, with typical values ranging from $b = 0.15$ to $b = 0.2$ (More et al., 2011). As ROCKSTAR only uses FOF groups for breaking up the simulation volume to be distributed to individual processors, it is able use a modified algorithm for calculating FOF groups that is an order of magnitude faster than the typical procedure of finding all particles within the linking length for every particle. For particles with more than 16 neighbor particles, the neighbor finding process is skipped for the neighboring particles. Instead, particles are linked to the same group if they are within two linking lengths of the original particle. This method runs much

faster than the standard FOF algorithm, and links together at minimum the same particles. With this approach, run time decreases instead of increases with increasing linking length. ROCKSTAR therefore uses a large linking length of $b = 0.28$. The FOF groups are distributed among the available processors according to individual processor load.

II.3.1.2 Phase-Space FOF Hierarchy

Within each FOF group, FOF subgroups are found hierarchically in phase-space. A phase-space linking length is adaptively chosen so that a constant fraction f of particles are linked together with at least one other particle. For two particles p_1 and p_2, the phase-space distance metric is defined as (Gottloeber, 1998)

$$d(p_1, p_2) = \left(\frac{|\vec{x_1} - \vec{x_2}|^2}{\sigma_x^2} + \frac{|\vec{v_1} - \vec{v_2}|^2}{\sigma_v^2} \right)^{1/2}, \tag{II.9}$$

where σ_x and σ_v are the particle position and velocity dispersions for the FOF group. The phase-space distance to the nearest neighbor is computed for each particle, the linking length is chosen such that $f = 0.7$, and a new FOF subgroup is determined. This process is repeated recursively on the new FOF subgroups until a minimum threshold of 10 particles is reached at the deepest level of the hierarchy.

II.3.1.3 Converting FOF Subgroups to Halos

Seed halos are created for each of the deepest level subgroups in the FOF hierarchy. Particles from successively higher levels of the hierarchy are then assigned to the seed halos until all particles in the original FOF group are accounted for. To suppress extraneous seed halo generation due to noise, seed halos are merged if their positions and velocities are within 10σ of Poisson uncertainties of each. Specifically, the halos are merged if

$$\sqrt{(x_1 - x_2)^2 \mu_x^{-2} + (v_1 - v_2)^2 \mu_v^{-2}} < 10\sqrt{2}, \tag{II.10}$$

with

$$\mu_x = \sigma_x/\sqrt{n}, \tag{II.11}$$

$$\mu_v = \sigma_v/\sqrt{n}, \tag{II.12}$$

where σ_x and σ_v are the position and velocity dispersions of the smaller seed halo, and n is the number of particles of the smaller seed halo.

If a parent FOF group contains multiple seed halos, particles are assigned to the closest seed halo in phase space. The distance between a halo h and a particle p is given by

$$d(h,p) = \left(\frac{|\vec{x_h} - \vec{x_p}|^2}{r_{\rm dyn,vir}^2} + \frac{|\vec{v_h} - \vec{v_p}|^2}{\sigma_v^2} \right)^{1/2}, \tag{II.13}$$

$$r_{\rm dyn,vir} = v_{\rm max} t_{\rm dyn,vir} = \frac{v_{\rm max}}{\sqrt{\frac{4}{3}\pi G \rho_{\rm vir}}}, \tag{II.14}$$

where the seed halo currently has velocity dispersion σ_v and maximum circular velocity $v_{\rm max}$. Here, "vir" refers to the virial overdensity as defined by Bryan & Norman (1998) for $\rho_{\rm vir}$, which is 360 times the background density at $z = 0$. ROCKSTAR does, however, allow other choices for density definitions.

II.3.1.4 Substructure

Satellite membership is assigned based on phase-space distances before calculating halo masses. Equation II.13 is used to find the distance to all other halos with a greater number of particles, treating each halo center as a particle. The halo is then assigned to be a subhalo of the closest larger halo within the same FOF group, if one exists. If data from an earlier time-step is available, then halo cores at the current time-step are linked to halos from the previous time-step based on the largest contribution to the current halo core's particle membership.

Halo masses are then determined so that particles assigned to the host are not counted in the mass of the subhalo, but particles in the subhalo are included in the

mass of the host. Subhalo membership is then recalculated such that subhalos are those that fall within r_Δ of more massive host halos.

II.3.2 Halo Properties

Halo positions based on maximum density peaks are more accurate than those found by averaging all FOF halo particles (Knebe et al., 2011). As ROCKSTAR has already determined the halo density distribution when calculating the FOF subgroup hierarchy, halo positions are readily calculated by taking the average position of the particles in the inner subgroup which best minimizes the Poisson error.

The velocity of the halo core can be substantial offset from that of the halo bulk (Behroozi et al., 2013). The velocity for the halo is calculated as the average velocity of the particles within the innermost 10% of the halo radius, as the galaxy hosted by the halo should be most associated with the halo core.

Halo masses are calculated using the spherical overdensity (SO) out to various density thresholds, including the virial threshold of Bryan & Norman (1998) and density thresholds relative to the background density and the critical density. Mass calculations include all particles from the substructure contained in the halo, and can optionally remove unbound particles. As subhalo particles can be isolated from those of the host halo, mass calculations for substructure can also be obtained with spherical overdensities using only the particles belonging to the subhalo.

The scale radius R_s is determined by dividing halo particles up into up to 50 radial equal-mass bins, with a minimum of 15 particles per bin, and fitting an NFW profile to the bins to find the maximum-likelihood fit. The Klypin scale radius (Klypin et al., 2011), which uses $v_{\rm max}$ and $M_{\rm vir}$ to calculate R_s, is also determined.

A number of other parameters are calculated, including the angular momentum, halo spin parameter (Peebles, 1969), Bullock spin parameter (Bullock et al., 2001a), central position offset (defined as the distance between the halo density peak and

the halo center of mass), central velocity offset (defined as the difference between the halo core velocity and the bulk velocity), ratio of kinetic to potential energy, and ellipsoidal shape parameters (Zemp et al., 2011).

II.4 CrossMatch

Having pairs of corresponding 2LPT and ZA simulations necessitates a method for reliably matching halos between the two if we wish to compare properties of companion halos. To accomplish this, we use the CROSSMATCH code initially developed by Manodeep Sinha. CROSSMATCH uses particle IDs to find matching halos based on the percentage of common constituent particles. The code was modified for this study to import and process the BGC2 files output by the ROCKSTAR halo finder.

As dynamical variations between 2LPT and ZA simulations can cause companion halos to diverge in their evolutionary history, we cannot rely on bulk halo properties such as mass or central position as a primary means of matching. CROSSMATCH therefore relies on ID-based particle matching to pair halos. Companion simulations are initialized with identical particle ID schemes, and CROSSMATCH can then use these particle IDs to find pairs that are most likely to be the "same" halo for a given simulation snapshot. At the most basic level, CROSSMATCH reads in halo and particle lists from a halo finder such as ROCKSTAR, iterates through the lists from one simulation, and finds the halo with the largest number of shared particles from the other simulation.

As CROSSMATCH needs to run on data from simulations with large numbers of particles, total runtime becomes a concern. A naive approach would be to iterate through the first particle list, and for every particle, linearly search through the entire second particle list to find which halo a particle belongs to. This would result in an $\mathcal{O}(N^2)$ runtime complexity. To decrease runtime to an acceptable level, the second particle list is first sorted by particle ID using a standard QuickSort algorithm, which

then enables the use of a more efficient binary search. This reduces runtime complexity to an $\mathcal{O}(N \log N)$ algorithm. Halos from the second simulation are then ranked by the percentage of particles in common with the halo from the first simulation, and the best match is selected.

II.5 Analysis

In this section, we discuss the details of the pipeline used for this work, including the analysis and plotting codes, databases, and automation scripts. We also present an overview of the results obtained at each step. A more in depth discussion of the observed trends and interpretations of results are presented in Sections III.3 and III.4.

As a high-level overview, we gather snapshots from previously run 2LPT and ZA simulations, find halos in each snapshot with ROCKSTAR, match halos between simulations with CROSSMATCH, and compare the differences in various properties between corresponding 2LPT and ZA halos, primarily as functions of redshift and halo mass. The specific codes developed for and used in our analysis are provided in the Appendices, and are referenced with the relevant discussions below.

II.5.1 Halo Properties with Rockstar

Halos are identified and measured with the ROCKSTAR halo finder, which is discussed in detail in Section II.3. Here, we discuss the setup necessary to run ROCKSTAR, as well as its output files, post-processing steps, and particle list extraction.

II.5.1.1 Simulation Snapshots and Rockstar Setup

We run ROCKSTAR on snapshots from each of our six simulation boxes. Each box has 62 snapshots, with 512^3 dark matter particles each. Due to the large size of the snapshot data and the per-user disk space quota of the ACCRE cluster, only one box is able to be processed at a time.

For each snapshot, a ROCKSTAR run directory is set up with a number of configuration files and scripts, including the ROCKSTAR configuration file (Appendix A.1), PBS submission script (Appendix A.2), a script to clean files from previous runs and begin a new run (Appendix M.3), and a script for post-processing generated output files (Appendix A.3). A directory for particle data contains a link to the actual simulation snapshot and a file containing a list of snapshot files, which in this case contains one item. A directory is also created for output halo data files. We discuss automation of run directory setup and simultaneous launching of multiple ROCKSTAR instances in Section II.6.

The parameter file controls various configuration options including simulation type, physical units, cosmological parameters, I/O options, halo definitions, and process setup. ROCKSTAR has native support for GADGET's snapshot format and can automatically import cosmological parameters and box size. Length and mass scales must be input to convert from simulation units. ROCKSTAR uses periodic boundary conditions based on the number of analysis processes. Periodic boundary conditions are assumed if using a multiple of eight analysis processes and are not assumed if using one analysis process. For ROCKSTAR to output BGC2 files (discusses below in Section II.5.1.2), the path of a file containing a list of snapshot filenames must be set as the BGC2 snapnames option. Halo virial radius and mass definitions may be set to either virial or a multiple of either the critical or background density. We select halos to be defined by the virial radius and mass. We are interested in defining halos as spherical overdensity halos rather than friends-of-friends halos, so we also choose to define halo properties based on all particles within the virial radius, whether or not they are energetically bound to the halo.

ROCKSTAR is run as a server-client setup. This is designed so that one processor acts as a director and output manager, one or more processors read in the input snapshots, and the remaining processors or compute nodes do the actual processing on

different segments of the simulation box. ROCKSTAR uses sockets for communication between the server process and the worker processes if running on multiple nodes. However, we were unable to configure ROCKSTAR in a way that it would run across multiple compute nodes, so we run each instance of ROCKSTAR on one node, with ten processor cores for the necessary functions. One processor acts as the server, one as the snapshot reader, and the remaining eight as halo finders.

II.5.1.2 Rockstar Output and Post-processing

ROCKSTAR outputs halo information in ASCII plaintext, binary, and BGC2 binary formats. As mentioned above, we run ROCKSTAR with eight worker processes per snapshot. Each worker process outputs it's own set of data files, with each file covering a separate octant of the simulation box plus a small overlap region. Halos with particles in the overlap region are saved based on the location of their centers. In addition to the per-processor output, a composite list of halos (and only halos) from all worker processors are created.

Through its various output files, ROCKSTAR provides a large number of measured halo properties. Whether or not full friends-of-friends particle lists are saved is controlled via the configuration file. Spherical overdensity particle lists are saved when utilizing BGC2 output. Particle data include particle ID, position, and velocity. Particle mass is not included as our simulations have uniform particle mass. Halo information consists of a large number of parameters, including halo ID, number of constituent particles, masses to various radii, position, velocity, angular momentum, spin, virial radius, scale radius, shape parameters, energy parameters, position and velocity offsets between the center of mass and the peak density, and parent halo ID.

As previously mentioned, we want halos defined based on spherical overdensity particle lists. These are only available from ROCKSTAR's BGC2 binary output format, with all other available particle lists consisting of friends-of-friends particles. The

BGC2 files consist of a 1024 byte header, halo data of 72 bytes per halo, and particle data with 32 bytes per particle. The header consists of an unsigned 8-byte integer, 16 8-byte signed integers, 19 8-byte double-precision floating point numbers, and extra padding out to 1024 bytes. The halo data consists of 2 8-byte signed integers, 2 8-byte unsigned integers, and 10 4-byte floating point numbers per halo. The particle data consists of 1 8-byte signed integer and 6 4-byte floating point numbers per particle. There is a 4-byte offset before the header, and 8-byte offsets between the header and halo data and between the halo data and particle data. The reader is referred to the bgc2.h header of the ROCKSTAR source code for further information on the contents of each structure. Our python code for reading in BGC2 files is presented in Appendix C. C code for reading in BGC2 files is bundled with the ROCKSTAR source code.

After ROCKSTAR is run, some post-processing of the output is needed. By default, ROCKSTAR does not provide information on membership information for substructure. Two scripts—one for the composite halo list and one for the BGC2 files—are provided with ROCKSTAR to cycle back through the halo lists and find the "parents," or the halo in which a given subhalo is contained. A script is also provided to convert halo information in the BGC2 files to ASCII plaintext. Our script for running these post-processing steps is presented in Appendix A.3.

II.5.2 Density Profile Fitting

While ROCKSTAR's output includes measurements for halo virial and scale radii, and thus concentration, we independently fit NFW density profiles to halos and measure concentration as a verification of ROCKSTAR's fitting. The full density profile python code is presented in Appendix D. This section is included for completeness only, as we find that only a small fraction of halos are well fit by our method, and we instead rely on concentration measurements directly from ROCKSTAR for subsequent analysis.

II.5.2.1 Density Profiles

For each halo, a list of constituent spherical overdensity particles is obtained from the post-processed BGC2 catalog from ROCKSTAR's output. For our purposes here, the relevant parameters are particle mass and position. We also use the values for each halo's center position and virial radius as found by ROCKSTAR.

Density profiles are then constructed by binning the particle positions in logarithmic radial bins from the resolution limit of the simulation to the halo virial radius and multiplying by particle mass. Before being passed to the fitting routine, density profiles are normalized to unity for both virial radius and maximum density.

II.5.2.2 Fitting

Halos are fit using the CurveFit routine from the SciPy Optimize library. It uses the Levenberg-Marquardt algorithm (Marquardt, 1963) for non-linear least squares fitting.

CurveFit is called by providing a model function, independent variable, measured dependent variable, and optionally weights for the dependent variable and initial guesses for fit coefficients. Here, our fit function is the NFW dark matter density profile (see Equation III.1). The free parameters to be fit are the scale radius R_s and the characteristic density ρ_0.

As the least squares algorithm is sensitive to local minima, care must be taken in choosing initial guesses for the fit coefficients. Additionally, large dynamic range in the fit parameters tended to produce poor results. We explored a number of solutions to improve solution stability, including fitting in logarithmic space and randomizing the initial guesses and picking the best solution. We found the best results were achieved by normalizing the data to unity for both radius and density, and choosing initial guesses within an order of magnitude for a typical halo, namely, normalized $R_s = 0.1$ and normalized $\rho_0 = 1.0$.

Some halos with irregular profiles presented the problem of the fitting algorithm choosing an unphysical scale radius larger than the virial radius of the halo. In order to heavily penalize this option from being chosen by the fitting algorithm, the dependent variable returned by the model function must differ from the input measured dependent variable as much as possible. However, we discovered that the transition must also be smooth, as a disjointed jump such as, say, returning a very large number for every value if $R_s > R_{\rm vir}$ would cause the algorithm to fail. We achieve this smooth transition penalty by adding the term $(R_s - 1)e^r$ to the density returned by the model function if the fitting algorithm tries to guess a value of R_s larger than $R_{\rm vir}$. However, while this did force halos to have definable concentrations, these halos often ended up with best fit scale radii equal to or just slightly less than the virial radii.

As we fit halos over a large range in redshift, we found low particle count halos to have noisy density profiles that were inherently more difficult to properly fit. Throughout our analysis, we use a lower bound of 100 particles to define a halo. At high redshift, even the largest halos are just beginning to cross this threshold. With such few particles spread across the number of bins necessary to properly define a density profile, we are left with only a handful of particles per bin. In Figure II.3, we compare one of the largest halos at $z = 14$ with one of the largest halos at the end of the simulation at $z = 6$.

II.5.2.3 Characterization of Uncertainty

An initial motivation for finding our own concentration parameters independent from ROCKSTAR is that ROCKSTAR does not provide information about the quality of its density profile fits. We assign Poisson errors to the density in each bin such that $\sigma_\rho = \rho\sqrt{N}/N$, where ρ is the density and N is the number of particles in each bin. These uncertainties are then provided as weights to the CurveFit routine. Upon

finding a best fit, the routine provides the fit parameters and an estimation of the uncertainty in those parameters via a covariance matrix, which we use to uncertainty in the concentration. Additionally, we find the χ^2 for the overall fit, which we use as an indicator of whether to accept or reject the fit for a given halo.

II.5.2.4 Concentration Comparison to Rockstar

Overall, we do not find good agreement with ROCKSTAR. Using a script (see Appendix H) to compare the concentrations derived from our fits with those from ROCKSTAR. At $z = 6$, we find that only 26% of halos fit by our method have concentrations within 20% of concentrations as measured by ROCKSTAR. We have slightly more agreement with high mass halos, with 37% agreement if we only consider the most massive 10% of halos. Additionally, we do not find good fits for every halo. If the distribution of particles would produce too few bins or the fitting routine exceeded a maximum number of iterations to find a stable solution, the halo is not fit. We also exclude halos with fits returned with very large χ^2 values. Because of the discrepancies in our results and the fact that we do not find acceptable fits for every halo, we use the more complete ROCKSTAR data for the final concentration measurements used in the remainder of our analysis.

II.5.3 Cross-matched Halo Catalog

With halo catalogs generated by ROCKSTAR for both 2LPT and ZA simulations, we need to be able to directly compare corresponding halos from the two suites of simulations. We match halos between simulations based on constituent particles with the CROSSMATCH code modified to import ROCKSTAR's BGC2 binary output files. Properties of the matched halos are then compiled into one large database per box for further filtering and analysis.

II.5.3.1 Cross-matching

Our simulations are initialized with identical particle ID schemes, and we are thus able to uniquely identify and track matching particles between simulations and match halos based on the largest number of shared particles. As the full implementation of the CrossMatch code is previously discussed in Section II.4, we only briefly summarize its place in our analysis pipeline here. The script in Appendix M.2 sets up the directory structure for the CrossMatch analysis and copies the CrossMatch parameter files (Appendices B.1 and B.2) to the appropriate run directories. CrossMatch is then run for each snapshot via the submission script in Appendix M.7, which is run for each simulation box.

Once caveat of the CrossMatch code is that matches are not necessarily unique. For each halo in the first simulation, only one best match halo will be selected from the second simulation. However, there may be other halos from the first simulation that also have the same halo from the second simulation selected as a best match. To counter this, we run CrossMatch in both directions—once matching za halos to 2lpt halos and once matching 2lpt halos to za halos—and choose best match halos as those that are matched in both directions. This assures a unique one-to-one matching between 2lpt and za halos. The code and submission script that select the best matches from the 2lpt-first and za-first cross-matched halo lists are presented in Appendix E.1.

II.5.3.2 Database Aggregation and Filtering

We now have raw halo data we need for further study, but are also left with a large number of disparate files that contain this information. For every snapshot, we have cross-simulation halo matching information from CrossMatch and the best match selection script, independent density profile and concentration measurement information from the density profile program, and original halo properties and host halo

membership information from ROCKSTAR spread across plaintext and BGC2 binary files for each processor on which ROCKSTAR was run, all for three simulation boxes each for both 2LPT and ZA.

We combine the information from all of these file into one centralized database per snapshot with the database generation program and submission script in Appendix F. The program reads in all of the source data files, finds companion halos from the output of CROSSMATCH, and outputs all available data for each halo pair aggregated together. The program is run for each of our 62 snapshots per simulation box, giving 186 total database files.

With the first version of our database generation code, total runtime became a significant factor. The halo matching code was initially implemented in a naive double loop search through all the data files to find collect halo pair properties. Pure python loop structures are exceedingly slow for larger data sets, and an initial estimate gave a runtime on the order of weeks or months. This was unacceptable, as there are many snapshots, and the aggregation may need to be performed multiple times if any of the previous steps in the analysis pipeline were to be modified. The code was therefore rewritten to take full advantage of the vectorization of the NumPy library, achieving a massive speedup to a runtime of order a few seconds.

In order to retain a centralized database of all available information for matched halos, we do not filter out halos at this step. Subsequent analysis, however, does remove halo pairs from consideration in certain circumstances. For early analysis involving our independent density profile fitting, we remove halos based on evidence of a poor fit, including halos that have measured concentrations greater than 100 or less than 1, ρ_0 less than zero, or χ^2 greater than 10. For all analysis, we remove halos with fewer than 100 particles and halos that exist as substructure in a larger host halo.

II.5.4 Halo Comparison

With a catalog of DM halos cross-matched between 2LPT and ZA simulations, we are able to directly compare properties on a halo-by-halo basis. At this stage, we are mostly concerned with a qualitative comparison between individual halos in order to judge the overall success of halo matching and the broad differences in halo evolution arising from differences in simulation initialization.

II.5.4.1 Match Verification

In order to compare halo evolution between 2LPT and ZA simulations, we first need to ensure that the halos being compared do actually represent the same halo in each simulation. The CROSSMATCH code as well as its implementation in our analysis pipeline are discussed above, so here we instead focus on the plots used as a visual sanity check on the resulting matches. The python code used to generate these plots is listed in Appendix G.1.

As we wish to compare halos that may have followed different evolutionary paths in their respective 2LPT or ZA simulations, we are unable to do a hard cut on a single parameter such as mass, radius, position or particle distribution. However, large variances in any of these properties can hint at a problem in the matching algorithm. We therefore perform a quick visual check on a number of halo pairs by plotting their relative positions, radii, and constituent particle distributions in order to verify that the CROSSMATCH code performed as expected.

An example of this comparison is shown in Figure II.4, where we plot two large matching halos at $z = 6$. Particles belonging to the halos are plotted as points, with 2LPT halo particles in blue and ZA halo particles in red. The virial radii of the two halos are represented by the black circles. The virial radii and particle distributions are very similar, and there is only a small offset in position. We consider this a successful match.

II.5.4.2 Morphology

The morphology of a dark matter halo can provide insight into its structural evolution and merger history. Features such as tidal features, irregular shapes, and offset nuclei hint at recent merger activity, while more symmetrical distributions suggest a quieter recent history. We compare DM particle distributions of matched halos by observing the projected density map along three axis vectors. The python code for plotting these, as well as the density profiles discussed below, is listed in Appendix G.2.

By comparing the projected density morphologies of companion 2LPT and ZA halos, we get a qualitative impression of the differences in their current evolutionary state. We found the inner nuclear region to often display the most discernible difference in structure between the two halos. For halo pairs where this difference is most apparent, such as one halo having a single central core with the other halo having two distinct density peaks, we believe the most likely cause to be an offset in merger epochs between the two simulations. In this case, the snapshot from one simulation would catch the merger in progress, with multiple unsettled density peaks still visible, while the other simulation snapshot would catch the halo after it has settled into a more virialized state.

As an example of this, we plot comparisons of two $z = 6$ halo pairs in Figures II.5 and II.6. The top two rows of panels of each show XY, XZ, and YZ projections of the dark matter density for the 2LPT and ZA halo on the first and second row, respectively. The density map is shown with a logarithmic color scale, and equal density contours are marked with white curves. Figure II.5 shows a pair of large halos that display similar central structure. These halos are unlikely to have largely differed in their evolution shortly prior to the snapshot. Figure II.6, however, shows a halo pair with differing nuclear structure. The ZA halo displays two distinct central density peaks, while the 2LPT halo shows only a single more relaxed core.

II.5.4.3 Density Profiles

The code listed in Appendix G.2, which produces the density projections discussed above, also plots comparisons of the halos' density profiles. We have addressed the creation of density profiles in Section II.5.2, and here the same method is used for each profile. In this case, we with to directly compare the profiles of the companion 2LPT and ZA halos, so they are plotted together, alongside the 2-D density projections discussed in the previous section.

We again consider the halo pairs compared in Figures II.5 and **??**, where the bottom two panels of each display the density profiles of the 2LPT and ZA halos, respectively. Halo particles are binned in logarithmically-spaced radial bins from the virial radius inward to the simulation resolution limit. The profiles are fit with the NFW profile model with free parameters for scale radius and characteristic density. The resulting fit is overplotted as red curves, and the scale radius is marked with the vertical purple dot-dash lines.

The halos in Figure II.5 display very similar central morphology and are both well-fit by the NFW profile. The more relaxed and spherically symmetrical halos such as these tend to be easier to fit well than more irregular halos. The measured scale radii for these halos are also very similar, and combined with the similar virial radii, produce similar concentration values. The halos in Figure II.6 display a more differing structure. While the 2LPT halo is relatively symmetrical, allowing it to be relatively well-fit by the NFW model, the ZA halo has two distinct central density peaks, causing the NFW fit to be a bit less accurate. Here, there is a marked difference in the resulting scale radii, with the 2LPT halo displaying a larger concentration than its ZA companion.

II.5.5 Difference Distributions

We now turn our focus to the ensemble halo population as a whole. Comparing individual companion halos can realistically only give a qualitative picture of differences arising between 2LPT and ZA simulations, as the large number of halos necessitates consideration of only a small percentage of the sample. We therefore need a consistent way of measuring the behavior of the entire population. In this section, we discuss how we measure these differences in halo populations using the codes listed in Appendix I. In particular, the analysis code itself is listed in Appendix I.1, the script to run the analysis on the combined halo population from all three simulation boxes is listed in Appendix I.2, the script to run the analysis on the simulation boxes independently is listed in Appendix I.3, and the script to collect the resulting statistics from all the individual snapshots into one database is listed in Appendix I.4.

II.5.5.1 Histograms

We wish to explore differences in a number of halo properties, so we construct a generic distribution so that any measured halo quantity q can be considered. The distribution should highlight the differences between 2LPT and ZA halo populations while remaining unbiased to the choice of simulation initialization. This leaves us with a distribution of the differences between 2LPT and ZA quantities, normalized by the average of the two:

$$\Delta q = \frac{q_{\rm 2LPT} - q_{\rm ZA}}{q_{\rm avg}}, \tag{II.15}$$

where $q_{\rm avg} = \frac{1}{2}(q_{\rm 2LPT} + q_{\rm ZA})$. Defined in this way, difference distributions of, e.g., virial mass $\Delta M_{\rm vir}$, concentration Δc, or the offset distance between the central density peak and the center of mass $\Delta X_{\rm off}$ can all be considered on equal footing. We create distribution histograms of Δq for various halo quantities both for the combined halo catalog from the stacked simulation boxes and for the individual simulation boxes separately.

II.5.5.2 Fitting

In order to extract a number of statistical quantities and to get a better high-level feel for the leading behavior of the distributions, we wish to fit a statistical model to the data histograms. While the data would seem to distributed according to a Gaussian distribution at first glance, we found the deviations from Gaussianity to be more significant than could be ignored. After significant trial and error, we found the Δq distributions to be best described by a generalized normal distribution (Nadarajah, 2005) with the probability density function

$$f(x) = \frac{\beta}{2\alpha\Gamma(1/\beta)} e^{(|x-\mu|/\alpha)^\beta}, \tag{II.16}$$

where μ is the mean, α is the scale parameter, β is the shape parameter, and Γ is the gamma function

$$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x}\, \mathrm{d}x. \tag{II.17}$$

The shape parameter β is restricted to $\beta \geq 1$. This allows the distribution to potentially vary from a Laplace distribution ($\beta = 1$) to a uniform distribution ($\beta = \infty$) and includes the normal distribution ($\beta = 2$). The distribution has variance

$$\sigma^2 = \frac{\alpha^2\Gamma(3/\beta)}{\Gamma(1/\beta)} \tag{II.18}$$

and excess kurtosis

$$\gamma_2 = \frac{\Gamma(5/\beta)\Gamma(1/\beta)}{\Gamma(3/\beta)^2} - 3. \tag{II.19}$$

The distribution is symmetric, and thus has no skewness by definition. As such, the values obtained for the skew of the distribution are measured directly from the data.

We use the CurveFit module from the SciPy library for all of our functional fitting. CurveFit is a non-linear least squares fitting routine that can fit an arbitrary input function to data with optional uncertainties. It can return estimates of the

free parameters of the model, as well as a covariance matrix used to determine the uncertainties in the fit coefficients.

We found our fitting routine to be fairly sensitive to differences in initial guess of fit coefficients. CurveFit is not guaranteed to find global minima, and can become stuck in local extrema. This ends up being most probable when trying to find multiple fit coefficients with large dynamic range. We found the best way to address this was to scale the data to unity in each dimension whenever possible. In the case of our difference histograms, the standard deviations of the distributions are typically around order unity, so it was only necessary to normalized the counts. We also found that we achieved better results when fitting in logarithmic space.

We explored a number of halo parameters, but found the most interesting distributions to be those for virial mass and concentration. In Figure II.7, we plot histograms of $\Delta M_{\rm vir}$ and Δc in the left and right columns, respectively, for three representative simulation snapshots at $z = 14.7$, $z = 10.3$, and $z = 6.0$. Data from the entire sample are plotted as blue histograms, data for the top 25% of halo pairs, sorted by 2LPT halo mass, are plotted as grey-filled green histograms, and the generalized normal distribution fits are overplotted as red dashed curves.

II.5.6 Redshift Trends

Up to this point, we have only considered one snapshot at a time. While we have observed variations with redshift, this has not been explicitly quantified. In this section, we consider the statistical quantities derived from the generalized normal distribution fits from the previous section as functions of redshift. The code used for this analysis is listed in Appendix J.

II.5.6.1 Mean and Standard Deviation

Representing the mean and standard deviation of the distributions is relatively straightforward. For the fit generalized normal distributions, we record values for the mean,

uncertainty in the mean, standard deviation, and uncertainty in the standard deviation. We also record the mean and standard deviation of the underlying distribution as directly measured from the data.

In Figure II.8, we plot the mean and standard deviation of the distributions for mass and concentration, as well as the rms value derived from the data, all as functions of redshift. The mean is plotted as blue points with error bars, the standard deviation is plotted as two black dashed lines that represent $\mu \pm \sigma$, and the rms is plotted as a dotted green line.

In this case, we wish to be conservative with the error bars on the mean. Since we have a measurement for the mean both from the fitting distribution and the underlying data, we can incorporate both of these into our result. The points plotted in Figure II.8 are the mean measured from the fit distribution, and the error bars are the uncertainty in the mean estimated from the least squares routine. However, if the mean measured directly from the data falls outside the error bars, the error bars are expanded to encompass that measurement. This is most often not a concern, as the means for most snapshots are very close together. However, when there is a slight discrepancy between the fit and data values, the error bars will reflect this.

II.5.6.2 Skew

The generalized normal distributions we use to fit our Δq histograms are symmetrical by definition and therefore have no inherent skew. This was a simplifying assumption necessary to use a well-defined distribution as well as reduce the number of free parameters during fitting. We do note, however, that the skew of our underlying data is often large enough to not be ignored.

Therefore, we need an alternate way to measure skew and its uncertainty. We use

the skew routine from the SciPy statistics library, which defines skew as

$$\gamma_1 = \frac{\mu_3}{\mu_2^{3/2}}, \tag{II.20}$$

where μ_m are central moments given by

$$\mu_m = E[(X-\mu)^m] = \sum_k (x_k - \mu)^m p(x_k) \tag{II.21}$$

$$= \sum_{k=0}^{m} (-1)^{m-k} \binom{m}{k} \mu^{m-k} \mu'_k, \tag{II.22}$$

with non-central moments μ'_m given by

$$\mu'_m = E[X^m] = \sum_k x_k^m p(x_k), \tag{II.23}$$

where $p(x_k)$ is the probability density function. The skew is then measured from the entire halo sample for the three combined simulation boxes. Uncertainty in skew is evaluated by taking the skew of the three boxes as independent measurements. The results for skew as a function of redshift are plotted as blue curves for the $\Delta M_{\rm vir}$ and Δc distributions in Figure II.9.

II.5.6.3 Kurtosis

Variable kurtosis is a fundamental part of the generalized normal distribution, so we may therefore derive the kurtosis directly from the fit distribution parameters. The generalized normal distribution is defined in terms of a shape parameter β, which does introduce some complexity in the conversion to kurtosis. The shape parameter is converted to excess kurtosis by way of Equation III.13. As this definition includes the Gamma function, a number of steps are required to convert the uncertainty in shape parameter to the uncertainty in kurtosis, which we outline below.

The standard deviation of a function $f(x_1, x_2, \ldots, x_n)$ is, in general, given by

$$s_f = \sqrt{\sum_x \left(\frac{\partial f}{\partial x}\right)^2 s_x^2} \tag{II.24}$$

with summation over all independent variables x. The generalized normal distribution

$$f(x) = \frac{\beta}{2\alpha\Gamma(1/\beta)} e^{-(|x-\mu|/\alpha)^\beta} \tag{II.25}$$

with mean μ, scale parameter α, and shape parameter β, has excess kurtosis

$$\gamma_2 = \frac{\Gamma(5/\beta)\Gamma(1/\beta)}{\Gamma(3/\beta)^2} - 3. \tag{II.26}$$

The gamma function

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, \mathrm{d}t \tag{II.27}$$

has the first derivative

$$\Gamma'(x) = \Gamma(x)\psi_0(x) \tag{II.28}$$

where the digamma function ψ_0 is the derivative of the logarithm of the gamma function and is given by

$$\psi_0(x) = \int_0^\infty \left(\frac{e^{-t}}{t} - \frac{e^{-xt}}{1-e^{-t}}\right)\, \mathrm{d}t \tag{II.29}$$

if the real part of x is positive.

We now apply (II.24) to (III.13) to find the standard deviation of the excess

kurtosis:

$$s_{\gamma_2} = \sqrt{\left(\frac{\mathrm{d}\gamma_2}{\mathrm{d}\beta}\right)^2 s_\beta^2} \tag{II.30}$$

$$= s_\beta \frac{\mathrm{d}\gamma_2}{\mathrm{d}\beta} \tag{II.31}$$

$$= s_\beta \frac{\mathrm{d}}{\mathrm{d}\beta}\left[\frac{\Gamma(5/\beta)\Gamma(1/\beta)}{\Gamma(3/\beta)^2} - 3\right]. \tag{II.32}$$

Making the substitution $x = 1/\beta$ and $\mathrm{d}x = -1/\beta^2\,\mathrm{d}\beta$, taking the derivative, and doing a bit of algebra, we have:

$$s_{\gamma_2} = s_\beta \frac{\mathrm{d}\gamma_2}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}\beta} \tag{II.33}$$

$$= s_\beta \left(-\frac{1}{\beta^2}\right)\frac{\mathrm{d}}{\mathrm{d}x}\left[\frac{\Gamma(5x)\Gamma(x)}{\Gamma(3x)} - 3\right] \tag{II.34}$$

$$= -s_\beta x^2 \left\{\frac{\Gamma(3x)^2 \frac{\mathrm{d}}{\mathrm{d}x}[\Gamma(5x)\Gamma(x)] - \Gamma(5x)\Gamma(x)\frac{\mathrm{d}}{\mathrm{d}x}[\Gamma(3x)^2]}{\Gamma(3x)^4}\right\} \tag{II.35}$$

$$= -s_\beta \frac{x^2}{\Gamma(3x)^4}\left\{\Gamma(3x)^2[5\Gamma(5x)\psi_0(5x)\Gamma(x) + \Gamma(5x)\Gamma(x)\psi_0(x)] - \Gamma(5x)\Gamma(x)[6\Gamma(3x)^2\psi_0(3x)]\right\} \tag{II.36}$$

$$= s_\beta \frac{x^2}{\Gamma(3x)^4}\left\{6\Gamma(5x)\Gamma(3x)^2\Gamma(x)\psi_0(3x) - \Gamma(5x)\Gamma(3x)^2\Gamma(x)[5\psi_0(5x) + \psi_0(x)]\right\} \tag{II.37}$$

$$= s_\beta \frac{x^2}{\Gamma(3x)^4}\left\{\Gamma(5x)\Gamma(3x)^2\Gamma(x)[6\psi_0(3x) - 5\psi_0(5x) - \psi_0(x)]\right\} \tag{II.38}$$

$$= s_\beta x^2 \frac{\Gamma(5x)\Gamma(x)}{\Gamma(3x)^2}[6\psi_0(3x) - 5\psi_0(5x) - \psi_0(x)]. \tag{II.39}$$

Substituting back in for x and recognizing an occurrence of γ_2, we have the result

$$s_{\gamma_2} = s_\beta \frac{1}{\beta^2}\left(\gamma_2 + 3\right)\left[6\psi_0(3/\beta) - 5\psi_0(5/\beta) - \psi_0(1/\beta)\right] \tag{II.40}$$

with which we can find the uncertainty in the kurtosis given the value and uncertainty of the shape parameter β.

With a method of determining the uncertainty in kurtosis established, we may now examine the results. In Figure II.9, we plot the kurtosis and associated uncertainties as a function of redshift as red curves for distributions of $\Delta M_{\rm vir}$ and Δc.

II.5.7 Mass Trends

So far, our analysis has mostly focused on the behavior of the entire halo sample as a single unit. However, there is also a wealth of information available when the statistics for our sample are viewed as functions of halo mass. In this section, we explore our halo ensemble more deeply by dividing into bins of mass and viewing the behavior of the resulting subsamples. In this way, we are able to explore differences in low- and high-mass halos, as well as quantify the explicit mass dependencies. The codes used for this analysis are listed in Appendix K.

II.5.7.1 Binning and Fitting

When representing the mass dependence of our various halo properties, we wished to do so in a way that was both straightforward to quantify and visually descriptive of the overall distribution of the data. We found the best way to accomplish this was to provide a dual representation, with the data both binned in mass for least-squares fitting and binned two dimensionally in mass and Δq, with a color scale representing bin density, for a human reader to more easily see the relative population of the parameter space.

First, the data is binned on a 2-D grid. We found this to be the most natural way to visually represent the distribution of the data, as some features like population sparseness at high redshift, asymmetry, and large differences in number between low- and high-mass halos would be more difficult to convey with only average mass bin means and standard deviations. The binned data are plotted with a logarithmic color scale and smoothed with a Gaussian kernel.

As a technical aside, we note that plotting bins with zero members with a log-

arithmic color scale naturally leads to poor results. We counter this by artificially counting one half halo for bins that are otherwise empty, and rescale the color representation to make anything less than one unit per bin display the minimum color value.

As an alternate representation, and mainly for the benefit of a more quantitative analysis, we bin the data along the average halo mass axis. For each bin, we measure the mean and standard deviation of the data. The uncertainty in the mean is then calculated as the standard deviation divided by the square root of the number of particles in the bin. We find a linear fit to the bin means using our standard least-squares approach, weighted by the mean uncertainties.

Example plots are provided in Figures II.10 and II.11 to demonstrate this approach. The 2-D binned data is plotted using a logarithmic color scale to represent the number density of halos in a given cell. The bin means and associated uncertainties are plotted as the black points with error bars. The standard deviation to either side of the mean is plotted as black dotted lines. The least-squares fit to the bin means is plotted as a solid magenta line.

II.5.7.2 Trends with Redshift

To better analyze the time evolution of the mass dependence, we need a more compact representation than simply looking at successive individual redshift snapshots. The most informative individual parameter from these plots is the slope of the linear fit line for Δq as a function of average halo mass. We therefore plot the slopes and associated uncertainties for each snapshot as a function of redshift, with the results for $\Delta M_{\rm vir}$ and Δc displayed in Figure II.12. The data are then fit with our linear least-squares routine, and the fit is overplotted as a red dashed line.

II.5.8 Alternate Difference Distributions

The distributions of Δq that have been discussed up to this point are an excellent measure of the overall behavior of the halo population differences between 2LPT and ZA simulations. However, as these distributions rely on the average quantity $q_{\rm avg} = (q_{\rm 2LPT} + q_{\rm ZA})/2$ for normalization, quantities like the fraction of halo pairs differing by a given amount between simulations are more difficult to extract. We therefore redefine out distribution quantity to instead use a normalization factor of $q_{\rm ZA}$:

$$\delta q = \frac{q_{\rm 2LPT} - q_{\rm ZA}}{q_{\rm ZA}}, \tag{II.41}$$

which allows for a more direct count of halo pairs. Statistics for these distributions are saved alongside the output for Δq distributions with the codes in Appendix I.

II.5.8.1 Equivalent Displacement

The question may be asked why these distributions have not been used all along, as they more readily offer more quantitative values for our halo populations. Our previous distributions of Δq are symmetrical between 2LPT and ZA quantities, which allows us to be completely unbiased as to which simulation initialization is correct. The distributions of δq lose this symmetry, and are only defined for $\delta q \geq -1$ for positive quantities like mass and concentration.

For this analysis, we therefore need a way to consider halo pairs that differ by a certain amount in either direction (e.g. pairs that differ in quantity q by 10%, whether q is larger in 2LPT or ZA). Rearranging Equation II.41 yields

$$q_{\rm 2LPT} = (\delta q + 1) q_{\rm ZA}, \tag{II.42}$$

and making the substitution $x = \delta q + 1$ gives us

$$q_{\text{2LPT}} = x q_{\text{ZA}}. \tag{II.43}$$

For a given x, we want to find x_{eq} such that $x_{\text{eq}} = 1/x$. Substituting now for x and x_{eq} and rearranging gives us

$$\delta q_{\text{eq}} = \frac{1}{\delta q + 1} - 1, \tag{II.44}$$

the value for which a halo pair with a larger q in ZA would differ by the same factor as a halo pair with a larger q in 2LPT.

II.5.8.2 Redshift Trends

In Figures II.13 and II.14, we plot statistics for our δq distributions as functions of redshift. In Figure II.13, we plot the δq of the peak of the distribution, as well as the δq values where 50%, 10%, and 1% of halo pairs fall at or above δq. In Figure II.14, we plot the fraction of halo pairs f_h that fall outside various δq values. The solid curves represent the fraction of halo pairs that have a 2LPT mass or concentration at least 1.1, 1.5, 2.0, or 5.0 times that of the corresponding ZA halo. Dashed curves represent the same values, regardless of whether the 2LPT or ZA mass or concentration is higher. This is the same as counting halos that fall above a given δq as well as below the corresponding δq_{eq}. The code for creating these plots is listed in Appendix L.

II.6 Automation

Dealing with the large number of data files, programs, and pipeline steps used in our analysis quickly becomes prohibitive in terms of time and complexity when each must be dealt with completely "by hand." In order to shorten the time needed for a full analysis of the data down to a reasonably human-scale level, a certain level of automation is required. A combination of shell scripting and basic parallelization was used to this effect. This has the added benefit of providing a self-documenting

reproducibility to the analysis that was invaluable for the inevitable times when an error was discovered and the entire pipeline had to be re-run from the beginning. In this section, we will give a very brief summary of the automation steps taken and the scripts written for these tasks. Scripts run locally or launched manually are written in Bash, while job scripts that are submitted to the ACCRE compute cluster use the PBS syntax for communication with the scheduler and Bash for the remaining logic.

The creation of the directory structure for analysis with ROCKSTAR and subsequent halo catalog generation steps was done using the script listed in Appendix M.1. The creation of the directory structure for CROSSMATCH was done using the script listed in M.2. Individual instances of ROCKSTAR may be run on individual snapshots with the script in Appendix M.3, while all snapshots may be run as a batch job using the scripts in Appendices M.4 and M.5 for 2LPT and ZA snapshots, respectively. The output from ROCKSTAR is run through a post-processing step that is automated using the script in Appendix M.6. The CROSSMATCH program is run with the script in Appendix M.7, and the python code to generate density profiles is launched with the script in Appendix M.8. A number of other Bash scripts, PBS submission scripts, and Python programs, which we have already discussed in the above sections, were used for automation of the remainder of the analysis pipeline.

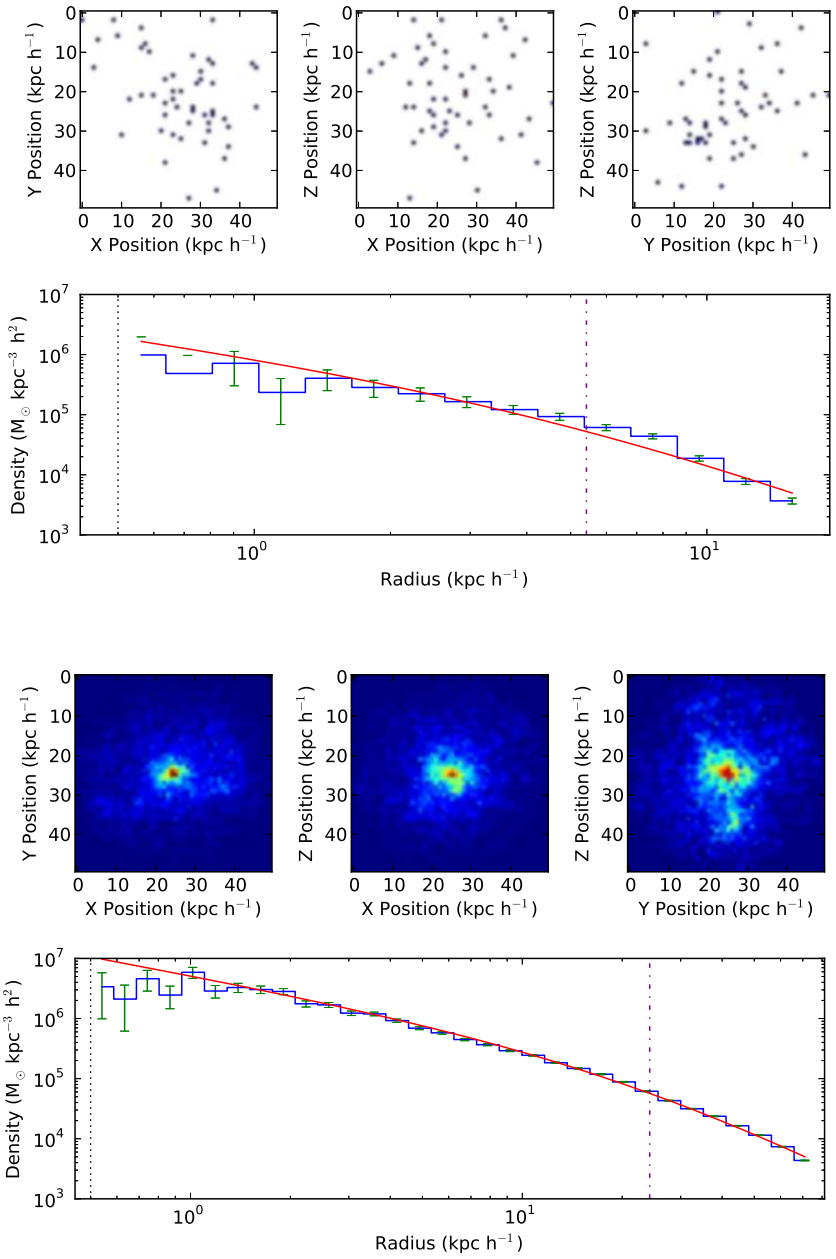


Figure II.3: Spacial projections and density profiles for two large halos at $z = 14$ (*top*) and $z = 6$ (*bottom*). Both halos are from the Box 1 2LPT simulation, and are the largest halos at their respective redshifts. The density profiles are fit with an NFW profile, and the resulting scale radius is plotted as a vertical dot-dash purple line.

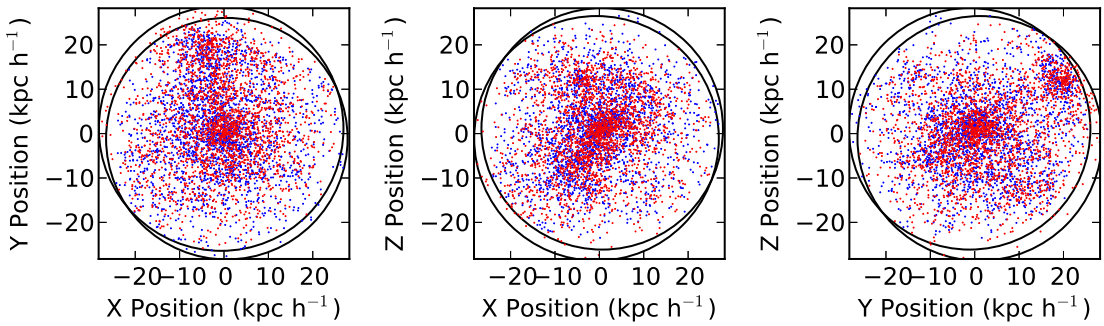


Figure II.4: Example of halo particle matching at $z = 6$. Blue dots are 2LPT halo particles, and red dots are ZA halo particles. Black circles are the virial radii of the halos. Good matches are achieved for halos, with only slight drift between simulations.

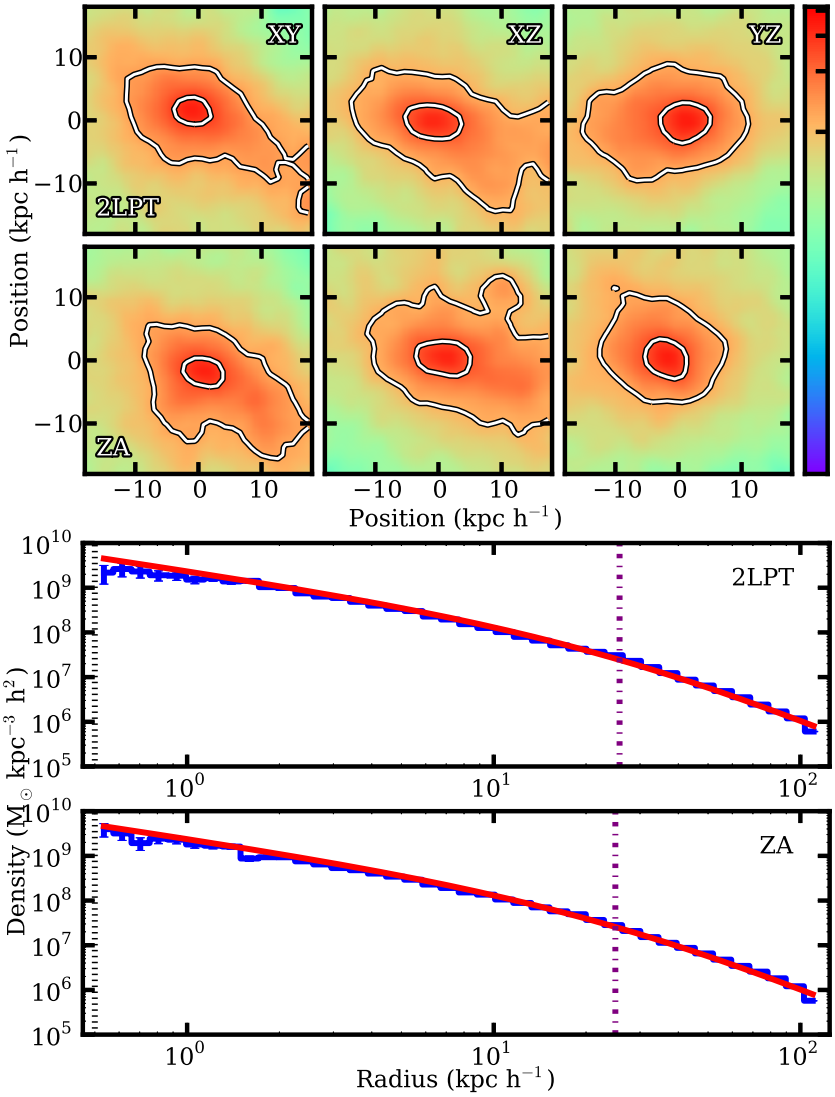


Figure II.5: Two large matched halos at $z = 6$ with similar nuclear structure. *Top two rows:* Projected density maps, with XY, XZ, and YZ views of the central nuclear region of the halos. Density is represented by a logarithmic color scale, and equal density contours are plotted as white curves. The first and second rows depict the 2LPT and ZA halo, respectively. *Bottom two rows:* Radially-binned halo density profiles fit with the NFW density profile model. The blue stepped profiles are the binned data, red curves are the fit NFW models, black dashed lines are the resolution limit of the simulation, and purple dot-dash lines are the measured scale radius.

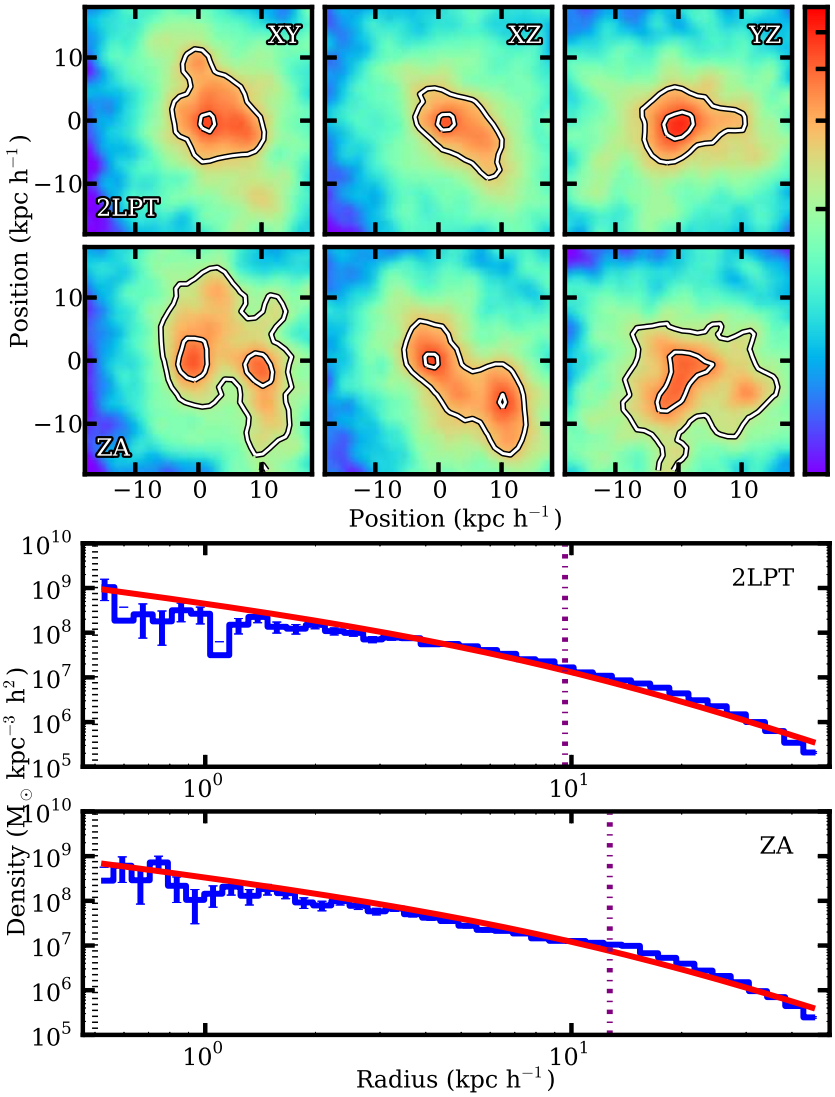
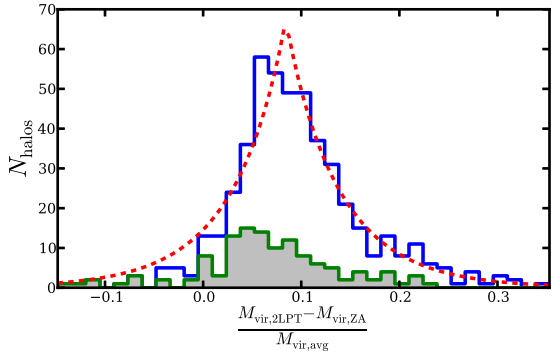
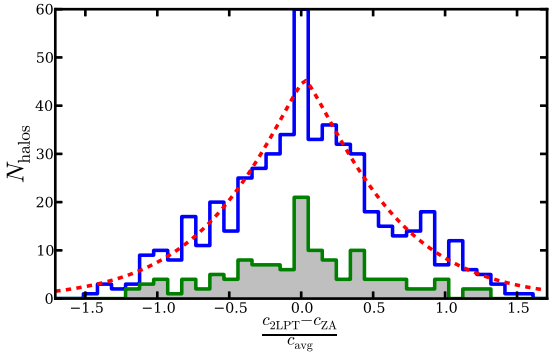
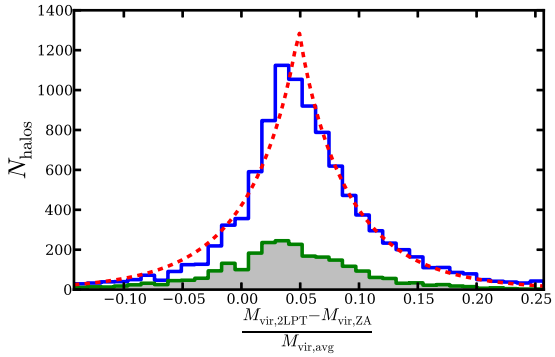
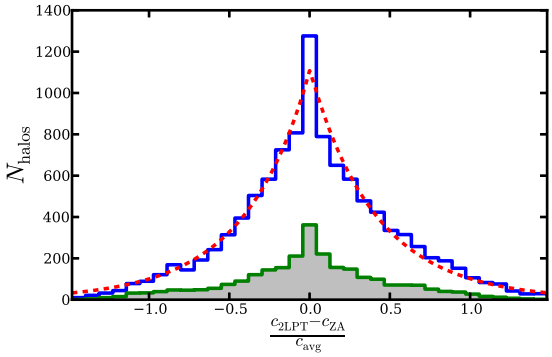


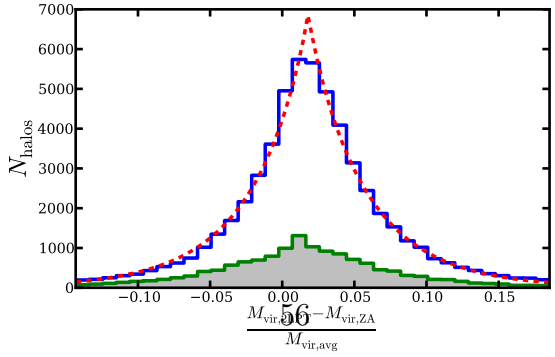
Figure II.6: Like Figure II.5, but for two large matched halos at $z = 6$ with differing nuclear structure.

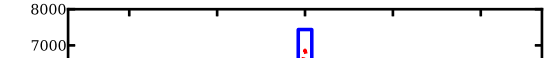

$N_{\rm halos}$
$\frac{M_{\rm vir,2LPT}-M_{\rm vir,ZA}}{M_{\rm vir,avg}}$


$N_{\rm halos}$
$\frac{c_{\rm 2LPT}-c_{\rm ZA}}{c_{\rm avg}}$


$N_{\rm halos}$
$\frac{M_{\rm vir,2LPT}-M_{\rm vir,ZA}}{M_{\rm vir,avg}}$


$N_{\rm halos}$
$\frac{c_{\rm 2LPT}-c_{\rm ZA}}{c_{\rm avg}}$


$N_{\rm halos}$
$\frac{M_{\rm vir,2LPT}-M_{\rm vir,ZA}}{M_{\rm vir,avg}}$


$N_{\rm halos}$

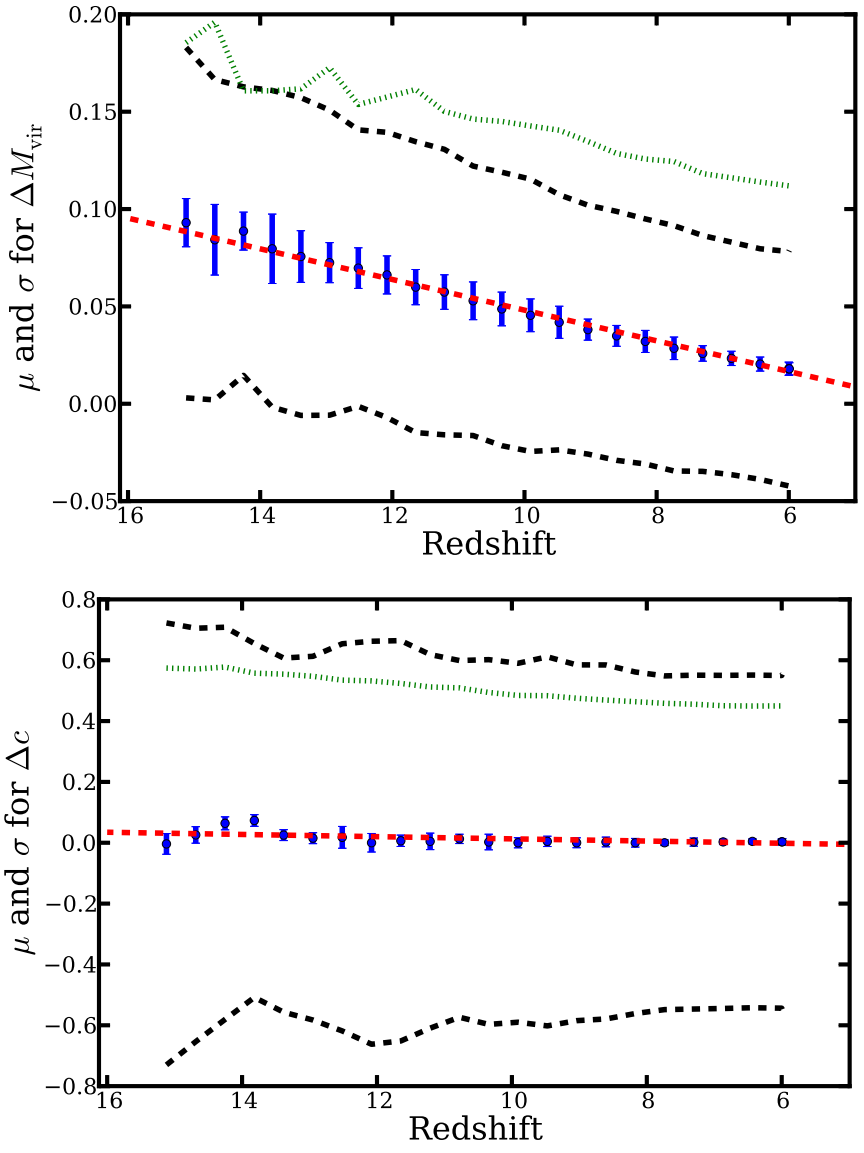


Figure II.8: Mean, standard deviation, and rms as functions of redshift for $\Delta M_{\rm vir}$ (*top*) and Δc (*bottom*). The mean is plotted as blue points, $\mu \pm \sigma$ is plotted as the black dashed curves, and rms values are plotted as a green dotted curve. The red dashed line is a linear fit to the mean.

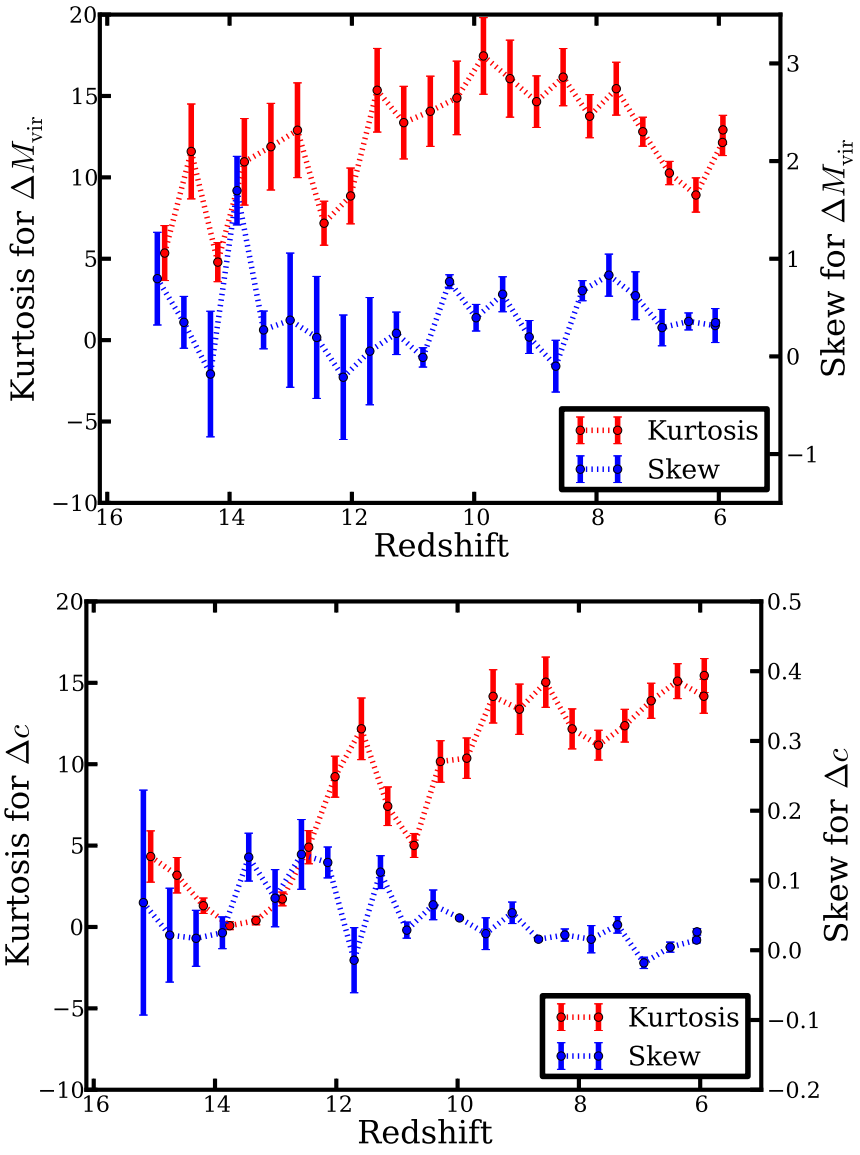


Figure II.9: Skew (blue curve) and excess kurtosis (red curve) from generalized normal distribution fits as functions of redshift for $\Delta M_{\rm vir}$ (*top*) and Δc (*bottom*). For both plots, the left axis is the scale for kurtosis and the right axis is the scale for skew.

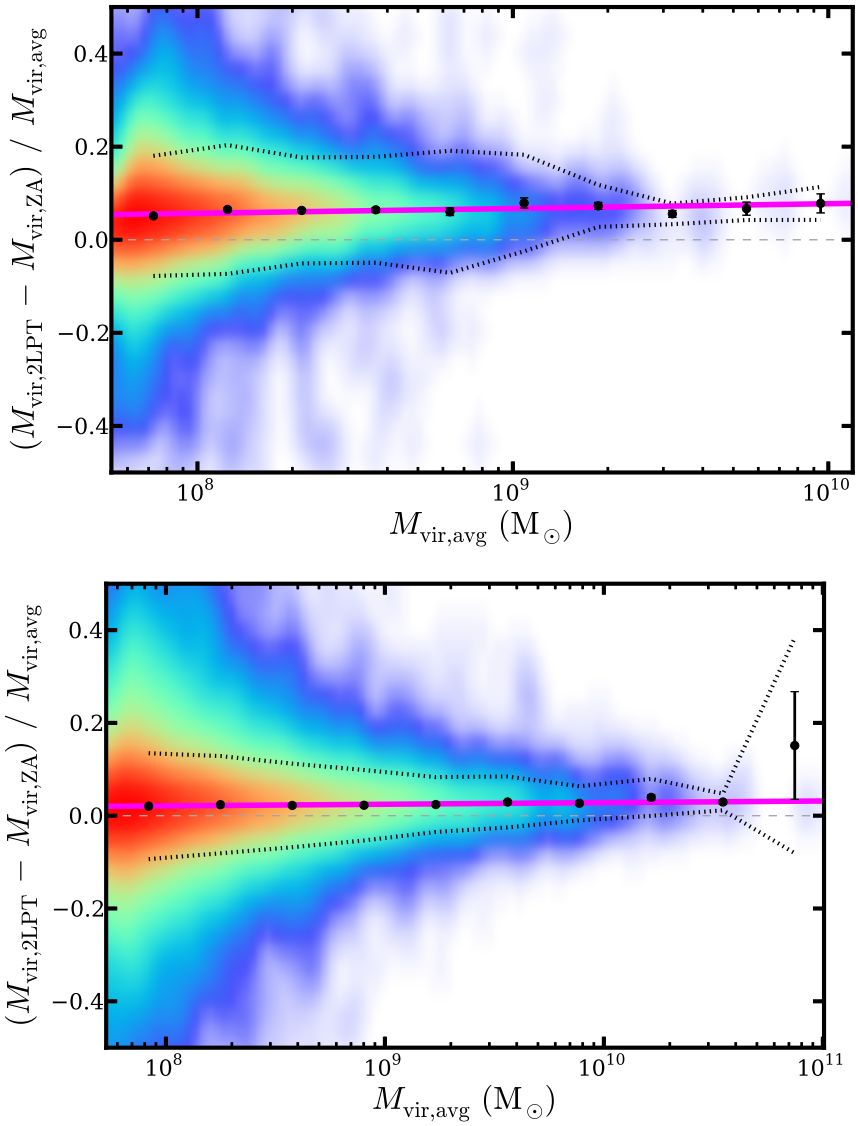


Figure II.10: $\Delta M_{\rm vir}$ as a function of $M_{\rm vir,avg}$. For the 2-D color histogram, halos are counted in rectangular bins and smoothed with a Gaussian kernel with a logarithmic color scale. The halos are also divided into logarithmically-spaced bins in average virial mass, and the mean for each bin is plotted as a black point. The black dotted curves are the standard deviation around the mean. The magenta line is the linear least-squares best fit to the bin means. The light grey dashed line at $\Delta q = 0$ is provided to guide the eye. The two panels correspond to snapshots at $z = 10.3$ and $z = 6.0$. These plots are provided as examples of the output at this stage of the analysis and are further discussed in Chapter III.

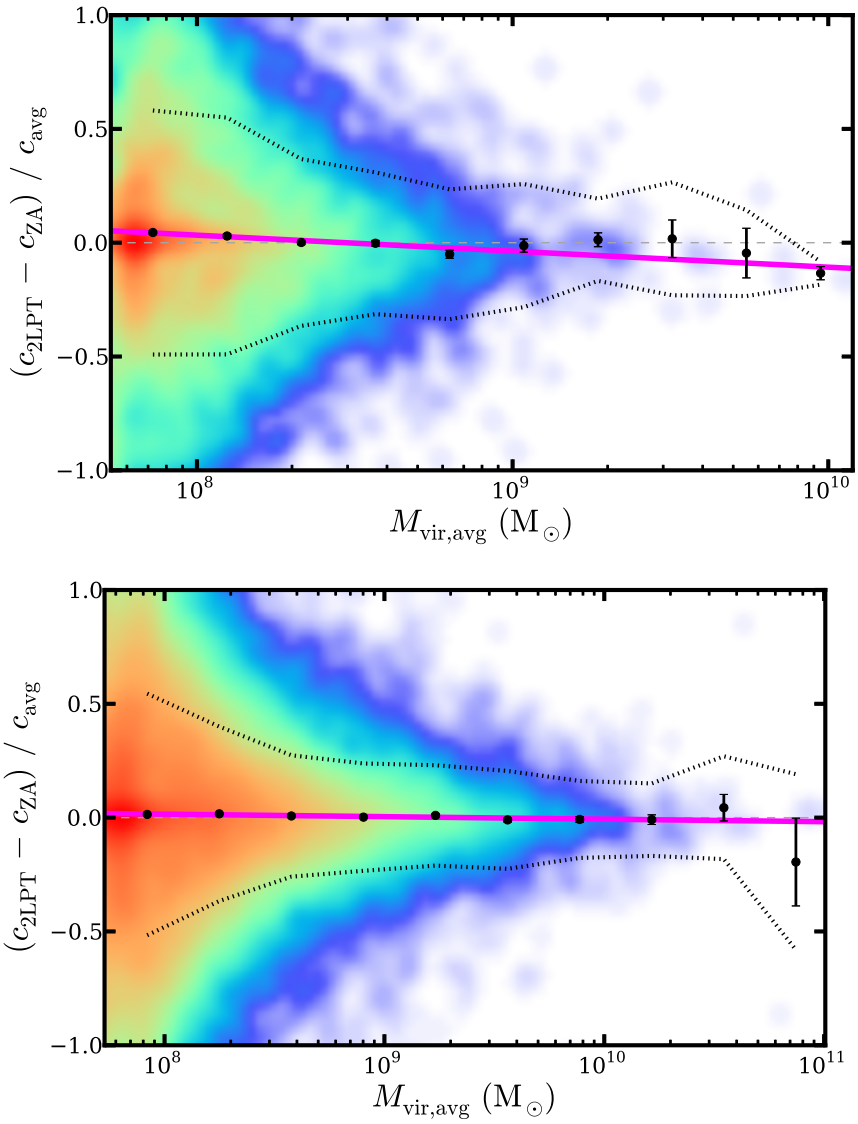


Figure II.11: Like Figure II.10, but for Δc instead of $\Delta M_{\rm vir}$ as a function of average halo mass.

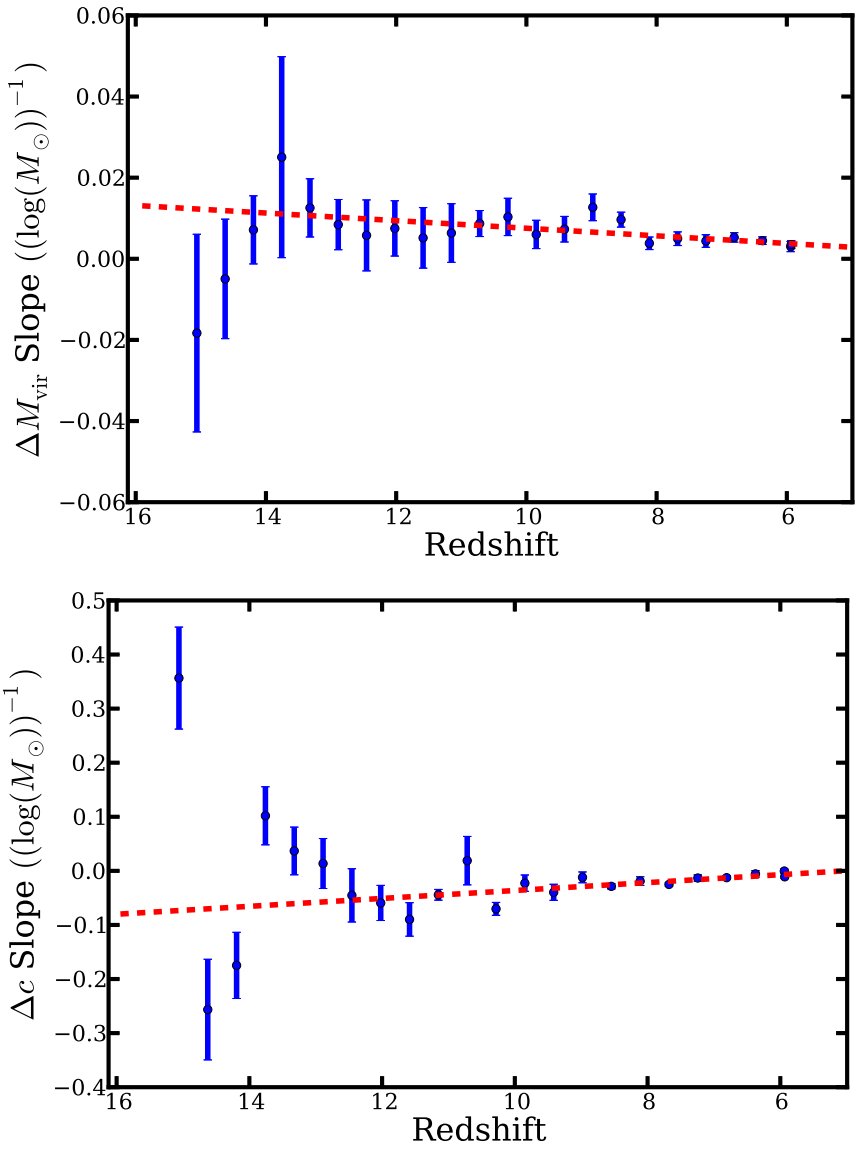


Figure II.12: Slopes of the Δq vs. $M_{\rm vir,avg}$ fit functions. The top and bottom panels correspond to the $\Delta M_{\rm vir}$ and Δc plots of Figures II.10 and II.11. Linear least-squares fits to the data are overplotted as red dashed lines. These plots are provided as examples of the output at this stage of the analysis and are further discussed in Chapter III.

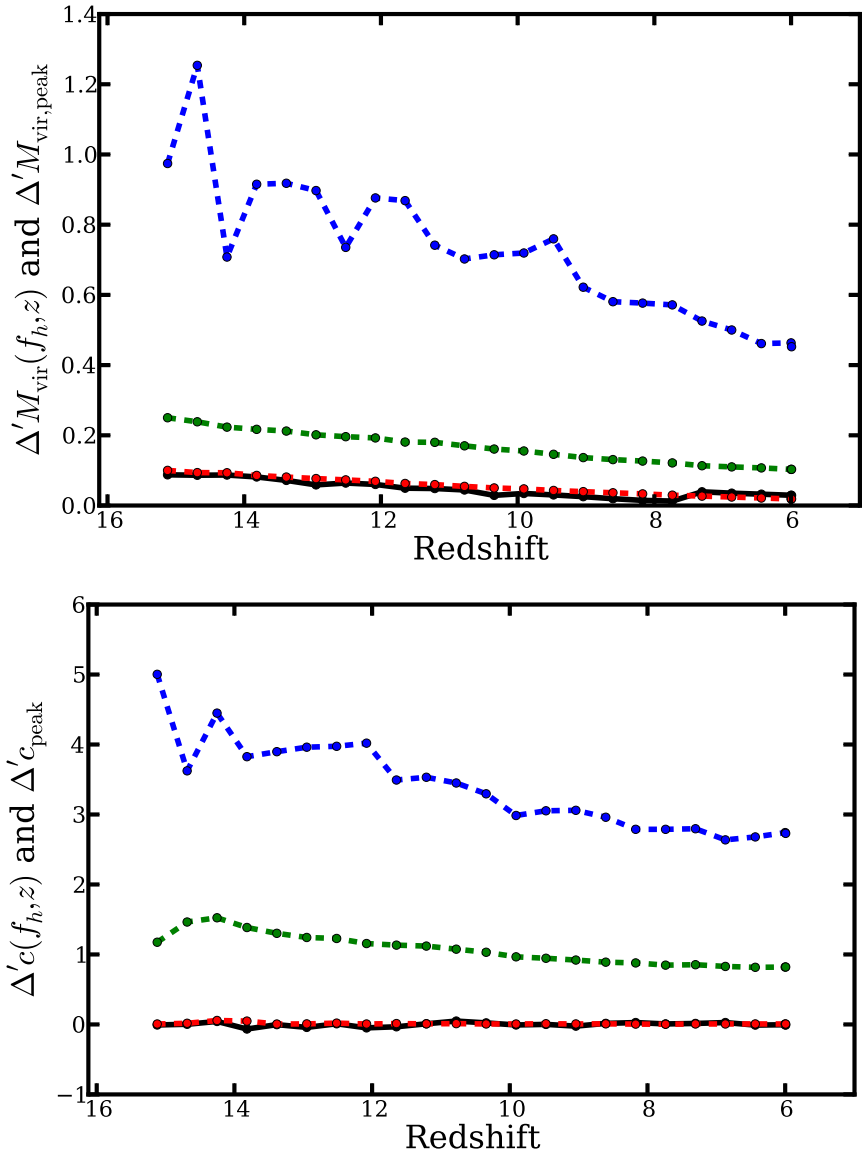


Figure II.13: Statistics for distributions of $\delta M_{\rm vir}$ (*top*) and δc (*bottom*) as functions of redshift. The δq of the peak of the distribution (black curve), and the δq where 50% (red dashed curve), 10% (green dashed curve), and 1% (blue dashed curve) of the halos fall at or above δq. These plots are provided as examples of the output at this stage of the analysis and are further discussed in Chapter III.

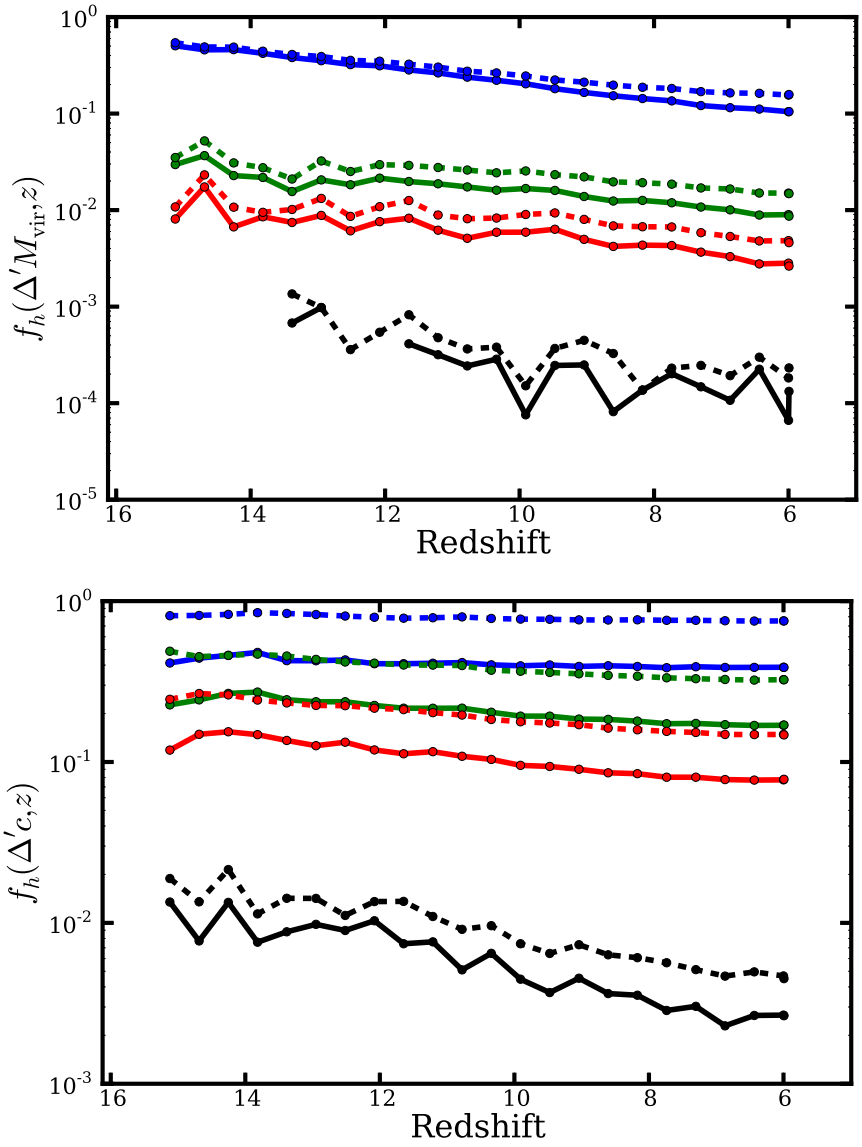


Figure II.14: Statistics for distributions of $\delta M_{\rm vir}$ (*top*) and δc (*bottom*) as functions of redshift. The fraction of halos with δq greater than 0.10 (solid blue curve), 0.50 (solid green curve), 1.00 (solid red curve), and 4.00 (solid black curve). The dashed curves additionally count halo pairs with δq lower than the corresponding equivalent displacements of -0.09, -0.33, -0.50, and -0.80, respectively (see Equation II.44). These plots are provided as examples of the output at this stage of the analysis and are further discussed in Chapter III.

CHAPTER III

Exploring Dark Matter Halo Populations in 2lpt and za Simulations

We study the structure and evolution of dark matter halos from $z = 300$ to $z = 6$ for two cosmological N-body simulation initialization techniques. While the second order Lagrangian perturbation theory (2LPT) and the Zel'dovich approximation (ZA) both produce accurate present day halo mass functions, earlier collapse of dense regions in 2LPT can result in larger mass halos at high redshift. We explore the differences in dark matter halo mass and concentration due to initialization method through three 2LPT and three ZA initialized cosmological simulations. We find that 2LPT induces more rapid halo growth, resulting in more massive halos compared to ZA. This effect is most pronounced for high mass halos and at high redshift, with a fit to the mean normalized difference between 2LPT and ZA halos as a function of redshift of $\mu_{\Delta M_{\rm vir}} = (7.88 \pm 0.17) \times 10^3 z - (3.07 \pm 0.14) \times 10^{-2}$. Halo concentration is, on average, largely similar between 2LPT and ZA, but retains differences when viewed as a function of halo mass. For both mass and concentration, the difference between typical individual halos can be very large, even for symmetrically distributed quantities, highlighting the shortcomings of ZA-initialized simulations for high-z halo population studies.

III.1 Introduction

The pre-reionization epoch is a time of significant evolution of early structure in the universe. Rare density peaks in the otherwise smooth dark matter (DM) sea lead to the collapse and formation of the first dark matter halos. For example, at $z = 20$, 10^7 $M_\odot$ halos are $\sim 4\sigma$ peaks, and 10^8 $M_\odot$ halos, candidates for hosting the first supermassive black hole seeds, are $\sim 5\sigma$ peaks.

These early-forming dark matter halos provide an incubator for the baryonic pro-

cesses that transform the surrounding space and allow galaxies to form. Initial gas accretion can lead to the formation of the first Pop-III stars (Couchman & Rees, 1986; Tegmark et al., 1997; Abel et al., 2000, 2002), which, upon their death, can collapse into the seeds for supermassive black holes (SMBHs) (Madau & Rees, 2001; Islam et al., 2003; Alvarez et al., 2009; Jeon et al., 2012) or enrich the surrounding medium with metals through supernovae (Heger & Woosley, 2002; Heger et al., 2003). The radiation from these early quasars (Shapiro & Giroux, 1987; Madau et al., 1999; Fan et al., 2001), Pop-III stars (Gnedin & Ostriker, 1997; Venkatesan et al., 2003; Alvarez et al., 2006), and proto-galaxy stellar populations (Bouwens et al., 2012; Kuhlen & Faucher-2012) all play a key role in contributing to the re-ionizing of the universe by around $z = 6$ (Barkana & Loeb, 2001). Additionally, halo mergers can drastically increase the temperature of halo gas through shock heating, increasing X-ray luminosity (Sinha & Holley-Bockelmann, 2009), and contribute to the unbinding of gas to form the warm-hot intergalactic medium (Bykov et al., 2008; Sinha & Holley-Bockelmann, 2010; Tanaka et al., 2012).

While a number of parameters are required to fully characterize a DM halo, a first-order description can be obtained from its mass and density profile. There are a number of ways to define a halo's mass, the subtleties of which becomes significant for mass-sensitive studies, such as the halo mass function (Press & Schechter, 1974; Reed et al., 2007; Heitmann et al., 2006; Lukić et al., 2007). For a review, see, e.g., White (2001) and references therein. Additionally, see Voit (2005) and references therein for a more observation-focused discussion.

From a simulation standpoint, the two most common ways to obtain halo mass are to define either spherical overdensity halos or friends-of-friends (FOF) halos. The spherical overdensity method identifies regions above a certain density threshold, either with respect to the critical density $\rho_c = 3H^2/8\pi G$ or the background density $\rho_b = \Omega_m \rho_c$, where Ω_m is the matter density of the universe. The mass is then the mass

enclosed in a sphere of some radius with mean density $\Delta\rho_c$, where Δ commonly ranges from ~ 100 to ~ 500. Alternatively, the FOF method finds particle neighbors and neighbors of neighbors defined to be within some separation distance (Einasto et al., 1984; Davis et al., 1985). Halo mass, then, is simply the sum of the masses of the constituent particles.

The density profile of a DM halo is determined by radially binning the constituent particles into spherical shells, and determining the average density per shell, giving a characteristic $\rho(r)$. The most widely used model for the DM halo density profile is the NFW (Navarro et al., 1996) profile

$$\rho(r) = \frac{\rho_0}{\frac{r}{R_s}\left(1+\frac{r}{R_s}\right)^2}, \tag{III.1}$$

where ρ_0 is the characteristic density, and the scale radius R_s is the break radius between the inner $\sim r^{-1}$ and outer $\sim r^{-3}$ density profiles.

The halo density profile is quantified by the halo concentration $c \equiv R_{\rm vir}/R_s$. $R_{\rm vir}$ is the halo virial radius, which is often defined as the radius at which the average interior density is some factor Δ_c times the critical density of the universe ρ_c, where Δ_c is typically ~ 200. Generally, at low redshift, low mass halos are more dense than high mass halos (Navarro et al., 1997a), and concentration decreases with redshift and increases in dense environments (Bullock et al., 2001b). Neto et al. (2007) additionally find that concentration decreases with halo mass. Various additional studies have explored concentration's dependence on characteristics of the power spectrum (Eke et al., 2001), cosmological model (Macciò et al., 2008), redshift (Gao et al., 2008; Muñoz-Cuartas et al., 2011), and halo merger and mass accretion histories (Wechsler et al., 2002; Zhao et al., 2003, 2009). For halos at high redshift, Klypin et al. (2011) find that concentration reverses and increases with mass for high mass halos, while Prada et al. (2012) find that concentration's dependence on mass

and redshift is more complicated and is better described through $\sigma(M, z)$, the RMS fluctuation amplitude of the linear density field.

The subtle $\mathcal{O}(10^{-5})$ density perturbations in place at the CMB epoch are vulnerable to numerical noise and intractable to simulate directly. Instead, a displacement field is applied to the particles to evolve them semi-analytically, nudging them from their initial positions to an approximation of where they should be at a more reasonable starting redshift for the numerical simulation. Starting at a later redshift saves computation time as well as avoiding interpolation systematics and round-off errors (Lukić et al., 2007).

The Zel'dovich approximation (Zel'dovich, 1970) and 2nd-order Lagrangian Perturbation Theory (Buchert, 1994; Buchert et al., 1994; Bouchet et al., 1995; Scoccimarro, 1998) are the two canonical frameworks for the initial particle displacement involved in generating simulation initial conditions. Zel'dovich approximation (ZA, hereafter) initial conditions (Klypin & Shandarin, 1983; Efstathiou et al., 1985) displace initial particle positions and velocities via a linear field, while 2nd-order Lagrangian Perturbation Theory (2LPT, hereafter) initial conditions (Scoccimarro, 1998; Sirko, 2005; Jenkins, 2010) add a second-order correction term to the expansion of the displacement field.

Following Jenkins (2010), we briefly outline the second-order Lagrangian perturbation theory and compare it to the Zel'dovich approximation. In 2LPT, a displacement field $\boldsymbol{\Psi}(\boldsymbol{q})$ is applied to the initial positions $\boldsymbol{q}$ to yield the Eulerian final comoving positions

$$\boldsymbol{x} = \boldsymbol{q} + \boldsymbol{\Psi}. \tag{III.2}$$

The displacement field is given in terms of two potentials $\phi^{(1)}$ and $\phi^{(2)}$ by

$$\boldsymbol{x} = \boldsymbol{q} - D_1 \boldsymbol{\nabla}_q \phi^{(1)} + D_2 \boldsymbol{\nabla}_q \phi^{(2)}, \tag{III.3}$$

with linear growth factor D_1 and second-order growth factor $D_2 \approx -3D_1^2/7$. The subscripts $\boldsymbol{q}$ refer to partial derivatives with respect to the Lagrangian coordinates $\boldsymbol{q}$. Likewise, the comoving velocities are given, to second order, by

$$\boldsymbol{v} = -D_1 f_1 H \boldsymbol{\nabla}_q \phi^{(1)} + D_2 f_2 H \boldsymbol{\nabla}_q \phi^{(2)}, \tag{III.4}$$

with Hubble constant H and $f_i = \mathrm{d} \ln D_i / \mathrm{d} \ln a$, with expansion factor a. The relations $f_1 \approx \Omega^{5/9}$ and $f_2 \approx 2\Omega^{6/11}$, with matter density Ω, apply for flat models with a non-zero cosmological constant (Bouchet et al., 1995). The f_1, f_2, and D_2 approximations here are very accurate for most actual ΛCDM initial conditions, as Ω is close to unity at high starting redshift (Jenkins, 2010). We may derive $\phi^{(1)}$ and $\phi^{(2)}$ by solving a pair of Poisson equations

$$\nabla_q^{(1)}(\boldsymbol{q}) = \delta^{(1)}(\boldsymbol{q}), \tag{III.5}$$

with linear overdensity $\delta^{(1)}(\boldsymbol{q})$, and

$$\nabla_q^{(2)}(\boldsymbol{q}) = \delta^{(2)}(\boldsymbol{q}). \tag{III.6}$$

The second order overdensity $\delta^{(2)}(\boldsymbol{q})$ is related to the linear overdensity field by

$$\delta^{(2)}(\boldsymbol{q}) = \sum_{i>j} \left\{ \phi_{,ii}^{(1)}(\boldsymbol{q}) \phi_{,jj}^{(1)}(\boldsymbol{q}) - \left[\phi_{,ij}^{(1)}(\boldsymbol{q}) \right]^2 \right\}, \tag{III.7}$$

where $\phi_{,ij} \equiv \partial^2 \phi / \partial q_i \partial q_j$. For initial conditions from the Zel'dovich approximation, or first-order Lagrangian initial conditions, the $\phi^{(2)}$ terms of Equations III.3 and III.4 are ignored.

Cosmological simulations that follow the initial collapse of dark matter density peaks into virialized halos often neglect to consider the nuances of initialization

method. Non-linear decaying modes, or transients, will be damped as $1/a$ in ZA. In 2LPT, however, transients are damped more quickly as $1/a^2$. It should be expected, then, that structure in 2LPT will be accurate after fewer e-folding times than in ZA (Scoccimarro, 1998; Crocce et al., 2006; Jenkins, 2010). The practical result is that high-σ DM density peaks at high redshift are suppressed in ZA compared with 2LPT for a given starting redshift (Crocce et al., 2006).

While differences in ensemble halo properties, such as the halo mass function, between simulation initialization methods are mostly washed away by $z = 0$ (Scoccimarro, 1998), trends at earlier redshifts are less studied (Lukić et al., 2007). In this paper, we explore the effects of ZA and 2LPT on the evolution of halo populations at high redshift. It is thought that 2LPT allows initial DM overdensities to get a "head start" compared with ZA, allowing earlier structure formation, more rapid evolution, and larger possible high-mass halos for a given redshift. We explore this possibility by evolving a suite of simulations from $z = 300$ to $z = 6$ and comparing the resulting differences in halo properties arising from initialization with ZA and 2LPT in these these otherwise identical simulations.

We discuss the simulations, halo finding, and analysis methods in Section III.2, results in Section III.3, implications, caveats, and future work in Section III.4, and a summary of our results and conclusions in Section III.5.

III.2 Numercial Methods

We use the Nbody tree/SPH code GADGET-2 (Springel et al., 2001; Springel, 2005) to evolve six dark matter–only cosmological volumes from $z_{start} = 300$ to $z = 6$ in a ΛCDM universe. Each simulation is initialized using WMAP-5 (Komatsu et al., 2009) parameters. For each of the three simulation pairs, we directly compare 2LPT and ZA by identically sampling the CMB transfer function and displacing the initial particle positions to the same starting redshift using 2LPT and ZA. The three sets

of simulations differ only by the initial phase sampling random seed. Each volume contains 512^3 particles in a 10 h^{-1} Mpc box. Full simulation details are discussed in Holley-Bockelmann et al. (2012).

One facet often overlooked when setting up an N-body simulation is an appropriate starting redshift, determined by box size and resolution (Lukić et al., 2007). As 2LPT more accurately displaces initial particle positions and velocities, initialization with 2LPT allows for a later starting redshift compared with an equivalent ZA-initialized simulation. However, many ZA simulations do not take this into account, starting from too late an initial redshift and not allowing enough e-foldings to adequately dampen away numerical transients. (Crocce et al., 2006; Jenkins, 2010). In order to characterize an appropriate starting redshift, the relation between the initial RMS particle displacement and mean particle separation must be considered. The initial RMS displacement $\Delta_{\rm rms}$ is given by

$$\Delta^2_{\rm rms} = \frac{4\pi}{3}\int_{k_f}^{k_{\rm Ny}} P(k, z_{\rm start})\ {\rm d}k, \tag{III.8}$$

where $k_f = 2\pi/L_{\rm box}$ is the fundamental mode, $L_{\rm box}$ is the simulation box size, $k_{\rm Ny} = \frac{1}{2}Nk_f$ is the Nyquist frequency of an N^3 simulation, and $P(k, z_{\rm start})$ is the power spectrum at starting redshift $z_{\rm start}$. In order to avoid the "orbit crossings" that reduce the accuracy of the initial conditions, $\Delta_{\rm rms}$ must be some factor smaller than the mean particle separation $\Delta_p = L_{\rm box}/N$ (Holley-Bockelmann et al., 2012). For example, making orbit crossing a $\sim 10\sigma$ event imposes $\Delta_{\rm rms}/\Delta_p = 0.1$. However, for small-volume, high-resolution simulations, this quickly leads to impractical starting redshifts. Continuing our example, satisfying $\Delta_{\rm rms}/\Delta_p \sim 0.1$ for a $10h^{-1}$ Mpc, 512^3 simulation suggests $z_{\rm start} \approx 799$. Starting at such a high redshift places such a simulation well into the regime of introducing errors from numerical noise caused by roundoff errors dominating the smooth potential. A more relaxed requirement of

$\Delta_{\rm rms}/\Delta_p = 0.25$ yields $z_{\rm start} = 300$, which we adopt for this work.

For each of our six simulations, we use the 6-D phase space halo finder code ROCKSTAR (Behroozi et al., 2013) to identify spherical overdensity halos at each timestep. ROCKSTAR follows an adaptive hierarchical refinement of friends-of-friends halos in 6-D phase space, allowing determination of halo properties such as halo mass, position, virial radius, internal energy, and number of subhalos. ROCKSTAR tracks halos down to a threshold of around 20 particles, but we use a more conservative 100 particle threshold for our analysis. We use all particles found within the virial radius to define our halos and their properties.

We identify matching halos based on the highest fraction of matching particles contained in each at any given timestep. We remove halo pairs where either one or both halos are considered subhalos (i.e. a halo must not be contained within another halo) and pairs with fewer than 100 particles in either 2LPT or ZA. We are left with approximately 60,000 total halo pairs for our three boxes at $z = 6$. With halo catalogs matched between simulations, we can compare properties of individual corresponding halos. To mitigate the effects of cosmic variance on our small volumes, we "stack" the three simulation boxes for each initialization method, and combine the halos from each into one larger sample in our analysis.

Halo concentration is derived from ROCKSTAR's output for R_s and $R_{\rm vir}$. Here, $R_{\rm vir}$ is the virial radius as defined by Bryan & Norman (1998). Figure III.1 makes evident the difficulty in fitting density profiles and obtaining concentration measurements for typical realistic halos. Large substructure, as displayed by the ZA halo, can disrupt the radial symmetry of the halo and cause significant deviations in the density profile. Centering can also be an issue in these cases. Due to these complications, there are a number of approaches for finding halo concentrations (Prada et al., 2012), but for consistency, we use the values derived from ROCKSTAR's fitting for our concentration measurements.

At each simulation snapshot, we measure and compare a number of parameters for halos in both 2LPT and ZA simulations. For each quantity q, we create histograms of Δq, the normalized difference in q between halos in the 2LPT and ZA simulations, defined as

$$\Delta q = \frac{q_{\text{2LPT}} - q_{\text{ZA}}}{q_{\text{avg}}}, \tag{III.9}$$

where $q_{\text{avg}} = \frac{1}{2}(q_{\text{2LPT}} + q_{\text{ZA}})$. For each of these, we fit the Δq histograms with a generalized normal distribution (Nadarajah, 2005) with the probability density function

$$f(x) = \frac{\beta}{2\alpha\Gamma(1/\beta)} e^{(|x-\mu|/\alpha)^\beta}, \tag{III.10}$$

where μ is the mean, α is the scale parameter, β is the shape parameter, and Γ is the gamma function

$$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x}\, \mathrm{d}x. \tag{III.11}$$

The shape parameter β is restricted to $\beta \geq 1$. This allows the distribution to potentially vary from a Laplace distribution ($\beta = 1$) to a uniform distribution ($\beta = \infty$) and includes the normal distribution ($\beta = 2$). The distribution has variance

$$\sigma^2 = \frac{\alpha^2\Gamma(3/\beta)}{\Gamma(1/\beta)} \tag{III.12}$$

and excess kurtosis

$$\gamma_2 = \frac{\Gamma(5/\beta)\Gamma(1/\beta)}{\Gamma(3/\beta)^2} - 3. \tag{III.13}$$

The distribution is symmetric, and thus has no skewness by definition. As such, the values for skew presented below are measured directly from the data.

As our fitting distributions are symmetrical and skew must therefore be measured directly from the data, in order to derive uncertainties for skew, we measure the skew of the distributions for each of our three simulation boxes individually as well as for

the single stacked data set. Uncertainty in skew is then simply the standard deviation of the mean of the skew of the three individual boxes.

Determining the uncertainty in the kurtosis is slightly more involved, as kurtosis is determined by a transformation of the generalized normal distribution's shape parameter β according to Equation III.13. Following the standard procedure for propagation of uncertainty, we calculate the standard deviation of the kurtosis as

$$s_{\gamma_2} = \sqrt{\left(\frac{\mathrm{d}\gamma_2}{\mathrm{d}\beta}\right)^2 s_\beta^2} \tag{III.14}$$

$$= s_\beta \frac{\mathrm{d}}{\mathrm{d}\beta}\left[\frac{\Gamma(5/\beta)\Gamma(1/\beta)}{\Gamma(3/\beta)^2} - 3\right]. \tag{III.15}$$

The derivative of the gamma function is

$$\Gamma'(x) = \Gamma(x)\psi_0(x), \tag{III.16}$$

where the digamma function ψ_0 is the derivative of the logarithm of the gamma function and is given by

$$\psi_0(x) = \int_0^\infty \left(\frac{e^{-t}}{t} - \frac{e^{-xt}}{1-e^{-t}}\right)\,\mathrm{d}t \tag{III.17}$$

if the real part of x is positive. Now, taking the derivative of γ_2 and doing a bit of algebra yields

$$s_{\gamma_2} = s_\beta \frac{1}{\beta^2}\left(\gamma_2 + 3\right)\left[6\psi_0(3/\beta) - 5\psi_0(5/\beta) - \psi_0(1/\beta)\right], \tag{III.18}$$

with which we can find the uncertainty in the kurtosis given the value and uncertainty of the shape parameter β estimated from the least squares fit routine.

In addition to distributions of Δq, we also consider distributions of

$$\Delta' q = \frac{q_{\rm 2LPT} - q_{\rm ZA}}{q_{\rm ZA}} \tag{III.19}$$

in order to better quantify the fraction of halos differing by a given amount between 2LPT and ZA simulations. This distribution is inherently non-symmetrical, and is only defined for $\Delta' q \geq -1$ for positive quantities like mass and concentration. In order to consider halo pairs that differ by a certain amount in either direction (e.g. pairs that differ by 10%, whether larger in 2LPT or ZA), a relation for equivalent displacement is required. Rearranging Equation III.19 yields

$$q_{\rm 2LPT} = (\Delta' q + 1) q_{\rm ZA}, \tag{III.20}$$

and making the substitution $x = \Delta' q + 1$ gives us

$$q_{\rm 2LPT} = x q_{\rm ZA}. \tag{III.21}$$

For a given x_1, we want an x_2 such that $x_2 = 1/x_1$. Substituting now for x_1 and x_2 and rearranging gives us

$$\Delta' q_2 = \frac{1}{\Delta' q_1 + 1} - 1, \tag{III.22}$$

the value for which a halo pair with a larger q in ZA would differ by the same factor as a halo pair with a larger q in 2LPT where $\Delta' q = \Delta' q_1$.

III.3 Results

With our catalog of matched dark matter halos, we directly compare differences in halo properties arising from initialization with 2LPT vs ZA. We consider halos on a pair–by–pair basis as well as the entire sample as a whole. Overall, we find 2LPT halos undergo more growth at a given redshift than their ZA counterparts.

III.3.1 Individual halo pairs

We compare large scale morphologies, density profiles, and various other halo properties for halo pairs on an individual halo–by–halo basis for several of the most massive halos. Morphologies appear similar for most halos, indicating good halo matches between simulations. However, many pairs display differences in central morphology, such as the number and separation of central density peaks. We interpret these cases to be examples of differences in merger epochs, in which case one halo may still be undergoing a major merger, while its companion is in a more relaxed post-merger state. We give an example of one such pair at $z = 6$ in Figure III.1. The top two rows show density projections of the nuclear regions for a large 2LPT and matching ZA halo (first and second rows, respectively). We find the ZA halo to contain two distinct density peaks with a separation of ~ 10 kpc, while the 2LPT halo displays only a single core. On the third and fourth rows, we plot the density profiles of the same two halos (2LPT and ZA, respectively). Here, with nearly identical virial radii, it is readily seen that the 2LPT halo is more concentrated than the ZA halo.

III.3.2 Differences in ensemble halo properties

For the halo population as a whole, we consider distributions of halo virial mass $M_{\rm vir}$ and concentration c. We plot histograms of $\Delta M_{\rm vir}$ and Δc in the left and right columns, respectively, of Figure III.2 for three representative timesteps at redshifts of $z = 14.7$, $z = 10.3$, and $z = 6.0$. For each panel, the blue histogram features the entire halo sample, and the smaller gray-filled green histogram displays only the top 25% most massive halos, ordered by 2LPT mass. Fits to the primary histograms are overplotted as red dashed curves.

Throughout the simulation, we find a tendency for 2LPT halos to be more massive. At $z = 15$, the mean of the $\Delta M_{\rm vir}$ distribution is $(9.3 \pm 1.2) \times 10^{-2}$. The mean is consistently positive (heavier 2LPT halos) and is most displaced from zero at high

redshift. The peak of the distribution gradually moves closer to zero as we progress in redshift. We find the least difference between paired halos for the final snapshot at $z = 6$, with $\mu_{\Delta M_{\rm vir}} = (1.79 \pm 0.31) \times 10^{-2}$.

The higher-order moments of the $\Delta M_{\rm vir}$ distribution are of interest as well, as we find significant deviation from a Gaussian distribution. As we use the symmetrical generalized normal distribution as our fit function, the skewness of the data is unable to be measured from the fit itself. However, a qualitative deviation from symmetry can be readily observed. By $z = 6$, we end up with a rather symmetrical distribution, with both sides of the histogram equally well described by our fit. However, at higher redshift, we note a marked increase in skewness and deviation from this symmetry. As redshift increases, we observe an increasing difference between the fit curve and the bins to the left of the histogram peak.

We find the distributions to be much closer to a Laplace distribution than a Gaussian, with shape parameter consistently sitting at or very close to $\beta = 1$. Compared to a Gaussian distribution, the larger excess kurtosis implies a narrower central peak and heavier outlying tails. Our fit constrains $\beta \geq 1$, so the kurtosis of the data itself could potentially be higher than the fit implies.

We find no overall preference for more concentrated 2LPT or ZA halos. In contrast to the $\Delta M_{\rm vir}$ histograms, Δc shows very little deviation from symmetry about zero. Throughout the simulation, we find the distributions to have a mean close to zero and negligible skew. The widths of the distributions are much larger than those for $\Delta M_{\rm vir}$, with an order of magnitude difference by $z = 6$. As with mass, concentration histograms are sharply peaked with heavy tails, implying a tendency for halo pairs to move towards the extremes of either very similar or very discrepant concentrations.

Table III.1: Coefficients for linear least squares fits from Figure III.3.

	A	B
$\Delta M_{\rm vir}$	$(7.88 \pm 0.17) \times 10^{-3}$	$(-3.07 \pm 0.14) \times 10^{-2}$
Δc	$(3.62 \pm 0.95) \times 10^{-3}$	$(-2.34 \pm 0.84) \times 10^{-2}$

III.3.3 Time evolution of mass and concentration

In Figure III.3, we more quantitatively assess the evolution of our various trends hinted at in Figure III.2. Here, we plot the mean, root mean square (RMS), standard deviation, skew, and kurtosis for $\Delta M_{\rm vir}$ and Δc as functions of redshift. Uncertainty in the mean is estimated directly by the least squares fitting routine.

The mean for $\Delta M_{\rm vir}$ is positive and highest at high redshift, trending toward zero by the end of the simulation. Distributions for Δc retain means close to and consistent with zero. Standard deviation decreases slightly for both $\Delta M_{\rm vir}$ and Δc. From $z = 15$ to $z = 6$, standard deviation falls from $(9.0 \pm 1.5) \times 10^{-2}$ to $(6.08 \pm 0.31) \times 10^{-2}$ for $\Delta M_{\rm vir}$ and from 0.73 ± 0.11 to 0.551 ± 0.026 for Δc.

We find least square linear fits for both mean $\Delta M_{\rm vir}$ vs z and mean Δc vs z. Coefficients for slope A and y-intercept B for the fit equation $\mu = Az + B$ are given in Table III.1 for both cases. We find a significant trend for $\Delta M_{\rm vir}$, with a slope $\sim 46\sigma$ from zero. Conversely, the slope for Δc is much smaller and, considering the larger spread of the underlying distributions, can be considered negligible. For $\Delta M_{\rm vir}$, the y-intercept coefficient B likely has little meaning in terms of the actual behavior at $z = 0$, as we expect the trend to level out at later redshift.

We do note, however, that the mean can be deceiving as an indicator of total difference between halo populations, especially when it is close to zero as with concentration. It should be noted that while the mean can indicate a lack of average difference between the whole sample of 2LPT and ZA halos, there can still be very large discrepancies between many individually paired halos. We visualize this by plotting the RMS of $\Delta M_{\rm vir}$ and Δc, which is plotted as a green dotted line. Unlike the mean,

standard deviation, and kurtosis, which are measured from fits to the histograms, RMS is measured directly from the data and is not dependent on fitting. The large RMS values are indicative of how much overall difference can arise between 2LPT and ZA halos, even though the differences may average to zero when considering the entire population. The RMS for both $\Delta M_{\rm vir}$ and Δc starts highest at high redshift—0.19 for $\Delta M_{\rm vir}$ and 0.57 for Δc at $z = 15$—and steadily decreases throughout the simulation, reaching minimums of 0.11 for $\Delta M_{\rm vir}$ and 0.45 for Δc by $z = 6$.

Additionally, it is of interest to consider the percentage of halo pairs that are "wrong" at some given time, regardless of whether the quantity is higher in 2LPT or ZA. For example, if we count halos outside a slit of $\epsilon = 10\%$ around $\Delta q = 0$, we find that by $z = 6$, 14.6% of halo pairs still have substantially mismatched masses, and 74.3% have mismatched concentrations. It is evident that a substantial percentage of halo pairs can have markedly different growth histories, even when there is little or no offset in the ensemble halo population average.

Kurtosis is consistently large for both mass and concentration, with a slight increasing trend throughout the simulation for concentration. It reaches maximum values of 17.5 ± 2.4 at redshift 10 for $\Delta M_{\rm vir}$ and 15.4 ± 1.0 at the end of the simulation at redshift 6 for Δc. Skew is positive for much of the simulation for mass, but is much smaller for concentration. We find average skews of 0.39 ± 0.29 for $\Delta M_{\rm vir}$ and 0.045 ± 0.028 for Δc. These higher moment deviations from Gaussianity hint at the non-linear dynamics at play in halo formation.

The narrow peak and heavy tails of the distribution may indicate a fair amount of sensitivity to initial differences in halo properties, in that halo pairs that start out within a certain range of the mean are more likely to move closer to the mean, while pairs that are initially discrepant will diverge even further in their characteristics. This is indicative of the non-linear gravitational influence present during halo evolution, and is further supported by a kurtosis that increases with time.

The skew at high redshift for $\Delta M_{\rm vir}$ may give another hint at the non-linear halo formation process. Runaway halo growth causes more massive halos to even favor faster mass accretion and growth. The positively skewed distributions show a picture of 2LPT halo growth in which initial differences in mass are amplified most readily in the earliest forming and most massive halos, again indicating the extra kick-start to halo growth provided by 2LPT initialization. While the slight decrease in skew with redshift may be counter-intuitive to this notion, it is likely that the large number of newly formed halos begin to mask the signal from the smaller number of large halos displaying this effect.

III.3.4 Dependence of mass and concentration differences on halo mass

We consider $\Delta M_{\rm vir}$ and Δc as a function of average halo mass $M_{\rm vir,avg} = (M_{\rm vir,2LPT} + M_{\rm vir,ZA})/2$ for three representative timesteps in Figure III.4. The data are binned in average virial mass, for which means and standard deviations are provided as the black points and black dotted lines, respectively. The error bars on the black points represent the uncertainty in the mean and are the standard deviation divided by the number of halos in that bin. We additionally bin the data in rectangular bins on a 2-D grid with a logarithmic color map in order to provide a more natural representation of the distribution of the data. Linear fits to the bin means are overplotted in magenta.

We find that $\Delta M_{\rm vir}$ tends to increase with increasing $M_{\rm vir,avg}$ for most snapshots. 2LPT halos are consistently more massive than their ZA counterparts, and, aside from the highest redshift snapshots, this difference increases with average halo mass. While less massive halo pairs have a larger spread in the difference in 2LPT and ZA mass, more massive halo pairs are consistently heavier in 2LPT than in ZA. At redshift 15, the transition snapshot between negative and positive slopes, the least squares fit to the data produces the fit equation $\Delta M_{\rm vir} = -(0.5 \pm 1.5) \times 10^{-2} \log(M_{\rm vir,avg}) + (0.15 \pm 0.12)$. The slope of the fit lines then become positive and trends back towards zero

Table III.2: Coefficients for linear least squares fits from Figure III.5.

	A	B
$\Delta M_{\rm vir}$	$(9.4 \pm 2.4) \times 10^{-4}$	$(-1.8 \pm 1.8) \times 10^{-3}$
Δc	$(-7.3 \pm 1.9) \times 10^{-3}$	$(3.7 \pm 1.4) \times 10^{-2}$

as we progress in redshift, with a fit of $\Delta M_{\rm vir} = (3.49 \pm 0.99) \times 10^{-3} \log(M_{\rm vir,avg}) - (6.8 \pm 8.3) \times 10^{-3}$ by $z = 6$.

We additionally find a trend for more massive halo pairs to be more concentrated in ZA. This trend is somewhat stronger than for $\Delta M_{\rm vir}$, but again, high z snapshots differ from the trend. The fit equations for $z = 15$ and $z = 6$ are $\Delta c = -(0.256 \pm 0.093) \log(M_{\rm vir,avg}) + (2.07 \pm 0.76)$ and $\Delta c = -(1.10 \pm 0.31) \times 10^{-2} \log(M_{\rm vir,avg}) + (0.103 \pm 0.026)$, respectively. The negative slope for most of the redshift range might be expected, as halo concentration is expected to decrease with increasing mass, at least at later redshift (Neto et al., 2007), and we find high mass halos to be more massive in 2LPT than in ZA for all but the highest redshift snapshots. However, the dependence of concentration on mass and redshift at high redshift is more complicated (Klypin et al., 2011; Prada et al., 2012). The data have a larger variance than $\Delta M_{\rm vir}$ by a factor of ~ 2. Again, mass dependence is smallest by $z = 6$. To reconcile these trends with the symmetrical concentration distributions of Figure III.2, we note that the trends in mass may be hidden by integration across the entire mass range and still result in overall Δc distributions symmetric about zero.

The slopes of the fits to the Δq vs.$M_{\rm vir,avg}$ data are plotted in Figure III.5. Linear least-squares fits are overplotted as red dashed lines. We find a trend for there to be more Δq dependence on $M_{\rm vir,avg}$ with increasing redshift, except for the highest z snapshots, which seem to reverse the trend. Coefficients A and B for the fit equation Slope $= Az + B$ are listed in Table III.2. The data are well-fit by the best fit line for most of the redshift range, except for $z \gtrsim 14$ for $\Delta M_{\rm vir}$ and $z \gtrsim 13$ for Δc, which begin to deviate from the trend. This may simply be due to the fluctuations inherent

when dealing with the low number of matched halos available in our sample at these very high redshifts.

III.3.5 Fractional differences in halo populations

As our distributions of Δq rely on the average quantity $q_{\rm avg} = (q_{\rm 2LPT} + q_{\rm ZA})/2$ for normalization, it can be difficult to extract certain statistics, such as the fraction of halo pairs differing by a certain amount between 2LPT and ZA simulations. To address this, for this section, we redefine our difference distributions to instead use $q_{\rm ZA}$ as the normalization factor (see Equation III.19). In Figure III.6, we plot, as functions of redshift, statistics derived from these alternate fractional difference distributions $\Delta' M_{\rm vir}$ and $\Delta' c$. In the left column, we plot the $\Delta' q$ of the peak of the distribution along with the $\Delta' q$ where various percentages of the halo pairs fall at or above $\Delta' q$.

As the $\Delta' q$ value of peak of the distribution is the location of the mode, it represents the most typical halo pair. While concentration differences remain close to zero throughout the simulation, mass difference peak moves from a $\Delta' M_{\rm vir}$ of 9×10^{-2} at $z = 15$ to 3×10^{-2} at $z = 6$. The 1% of halo pairs with the largest excess 2LPT mass have 2LPT mass at least twice ZA mass at $z = 15$ and 1.5 times ZA mass at $z = 6$. For concentration, the 1% most 2LPT concentrated halo pairs differ by at least a factor of 6 at $z = 15$ and 4 at $z = 6$.

In the right column of Figure III.6, we plot the fraction of halos f_h that fall outside various $\Delta' q$ values. The solid lines represent halo pairs that have $\Delta' q$ greater than or equal to the listed values, i.e., the fraction of halo pairs where the 2LPT halo has a virial mass or concentration that is at least 1.1, 1.5, 2.0, or 5.0 times that of its corresponding ZA halo. The dashed lines additionally count halos with $\Delta' q$ at or below the corresponding equivalent displacement (see Equation III.22) and represent the fraction of halo pairs where one halo has a virial mass or concentration at least 1.1, 1.5, 2.0, or 5.0 times that of its companion, regardless of whether the 2LPT or ZA

value is higher.

We find that half of halo pairs are at least 10% more massive in 2LPT at $z = 15$. By $z = 6$, this has fallen to 10%. Furthermore, 1% are at least twice as massive in 2LPT at $z = 15$, and by $z = 6$, this has only reduced to 0.3%. Halos in 2LPT are at least twice as concentrated as their ZA counterparts for at least 12% of the halo population at $z = 15$ and at least 8% by $z = 6$. Halo pairs that are at least 5 times as concentrated in 2LPT make up 1.3% of the sample at $z = 15$ and 0.3% at $z = 6$.

If we consider only the difference in properties between paired halos, regardless of whether the 2LPT or ZA halo has the higher mass or concentration, we include an even larger percentage of the population. This equates to additionally considering halo pairs that are less than or equal to the equivalent displacement as defined in Equation III.22. We find 54% of the halo pairs differ in mass by at least 10% at $z = 15$, with 16% differing by $z = 6$. Halos that are at least twice as massive in either 2LPT or ZA account for 1.1% at $z = 15$ and 0.5% at $z = 6$. Halos that are at least twice as concentrated in either 2LPT or ZA account for 25% at $z = 15$ and 15% at $z = 6$.

III.4 Discussion

As we evolve our DM halo population from our initial redshift to $z = 6$, we find that simulation initialization with 2LPT can have a significant effect on halo population compared to initialization with ZA. The second order displacement boost of 2LPT provides a head start on the initial collapse and formation of DM halos. This head start manifests itself further along in a halo's evolution as more rapid growth and earlier mergers. 2LPT halos are, on average, more massive than their ZA counterparts at a given redshift, with a maximum mean $\Delta M_{\rm vir}$ of $(9.3 \pm 1.2) \times 10^{-2}$ at $z = 15$. The larger mass for 2LPT halos is more pronounced for higher mass pairs, while 2LPT halo concentration is larger on the small mass end. Both mass and concentration

differences trend towards symmetry about zero as halos evolve in time, with the smallest difference observed at the end of the simulations at $z = 6$, with a mean $\Delta M_{\rm vir}$ of $(1.79 \pm 0.31) \times 10^{-2}$. Casual extrapolation of our observed trends with redshift to today would indicate that, barring structure like massive clusters that form at high redshift, 2LPT and ZA would produce very similar halo populations by $z = 0$. However, the larger differences at high redshift should not be ignored.

The earlier formation times and larger masses of halos seen in 2LPT-initialized simulations could have significant implications with respect to early halo life during the Dark Ages. Earlier forming, larger halos affect the formation of Pop-III stars, and cause SMBHs to grow more rapidly during their infancy (Holley-Bockelmann et al., 2012) and produce more powerful early AGN. The epoch of peak star formation may also be shifted earlier. This could additionally increase the contribution of SMBHs and early star populations to the re-ionization of the universe. Larger early halos may also increase clustering, speed up large scale structure formation, and influence studies of the high-z halo mass function, abundance matching, gas dynamics, and galaxy formation.

In these discussions, it is important to note that it is wrong to assume that the ZA halo properties are the "correct" halo properties, even in a statistical sense. While halo mass suggests the most obvious shortcoming of ZA simulations, even properties such as concentration—that show little difference on average between 2LPT and ZA—can have large discrepancies on an individual halo basis. Failure to consider uncertainties in halo properties for high z halos in ZA simulations can lead to catastrophic errors.

We note a few caveats with our simulations and analysis. We did not exclude substructure when determining the properties of a halo, and although this would not change the broad conclusions herein, care must be taken when comparing to works which remove subhalo particles in determining halo mass and concentration. Halo

matching is not perfect, as it is based on one snapshot at a time, and may miss-count halos due to merger activity and differences in merger epochs. However, we believe this effect to be minor. While we compared ROCKSTAR's output with our own fitting routines and found them to broadly agree, ROCKSTAR does not provide goodness of fit parameters for its NFW profile fitting and $R_{\rm s}$ measurements. It also may be debated whether it makes sense to even consider concentration of halos at high redshift which are not necessarily fully virialized.

As ROCKSTAR does not provide goodness-of-fit parameters for its internal density profile measurements used to derive concentration, error estimates for concentration values of individual halos are unknown. Additionally, proper density profile fitting is non-trivial, as the non-linear interactions of numerical simulations rarely result in simple spherical halos that can be well described using spherical bins. Halo centering issues may also come into play, although ROCKSTAR does claim to perform well in this regard.

We use a simulation box size of only $(10\ {\rm Mpc})^3$. This is too small to effectively capture very large outlier density peaks. We would, however, expect these large uncaptured peaks to be most affected by 2LPT initialization, so the effects presented here may even be dramatically underestimated. Additionally, a larger particle number would allow us to consider smaller mass halos than we were able to here, and to better resolve all existing structure. A higher starting redshift could probe the regime where 2LPT initialization contributes the most. It would also be of interest to evolve our halo population all the way to $z = 0$. The addition of baryons in a fully hydrodynamical simulation could also affect halo properties. These points may be addressed in future studies.

III.5 Conclusion

We analyzed three 2LPT and ZA simulation pairs and tracked the spherical overdensity dark matter halos therein with the 6-D phase space halo finder code ROCKSTAR to compare the effect of initialization technique on properties of particle–matched dark matter halos from $z = 300$ to $z = 6$. This approach allowed us to directly compare matching halos between simulations and isolate the effect of using 2LPT over ZA. In summary, we found the following:

- 2LPT halos get a head start in the formation process and grow faster than their ZA counterparts. Companion halos in 2LPT and ZA simulations may have offset merger epochs and differing nuclear morphologies.

- 2LPT halos are, on average, more massive than ZA halos. At $z = 15$, the mean of the $\Delta M_{\rm vir}$ distribution is $(9.3 \pm 1.2) \times 10^{-2}$, and 50% of 2LPT halos are at least 10% more massive than their ZA companions. By $z = 6$, the mean $\Delta M_{\rm vir}$ is $(1.79 \pm 0.31) \times 10^{-2}$, and 10% of 2LPT halos are at least 10% more massive.

- This preference for more massive 2LPT halos is dependent on redshift, with the effect most pronounced at high z. This trend is best fit by $\Delta M_{\rm vir} = (7.88 \pm 0.17) \times 10^{-3} z - (3.07 \pm 0.14) \times 10^{-2}$.

- Earlier collapse of the largest initial density peaks causes the tendency for more massive 2LPT halos to be most pronounced for the most massive halos, a trend that increases with redshift. We find a trend of $\Delta M_{\rm vir} = (1.03 \pm 0.46) \times 10^{-2} \log(M_{\rm vir,avg}) - (2.6 \pm 3.8) \times 10^{-2}$ for $z = 10$. By $z = 6$, this has flattened to $\Delta M_{\rm vir} = (3.49 \pm 0.99) \times 10^{-3} \log(M_{\rm vir,avg}) - (6.8 \pm 8.3) \times 10^{-3}$. As a function of redshift, the slopes of these equations are fit by Slope $= (9.4 \pm 2.4) \times 10^{-4} z - (1.8 \pm 1.8) \times 10^{-3}$.

- Halo concentration, on average, is similar for 2LPT and ZA halos. However, even by the end of the dark ages, the width of the Δc distribution—$\sigma_{\Delta c} = 0.551 \pm$

0.026 at $z = 6$—is large and indicative of a significant percentage of halos with drastically mismatched concentrations, despite the symmetrical distribution of Δc. At $z = 15$, 25% of halo pairs have at least a factor of 2 concentration difference, with this falling to 15% by $z = 6$.

- There is a trend for ZA halos to be more concentrated than 2LPT halos at high mass. We find $\Delta c = -(0.256 \pm 0.093)\log(M_{\rm vir,avg}) + (2.07 \pm 0.76)$ at $z = 15$ and $\Delta c = -(1.10 \pm 0.31) \times 10^{-2}\log(M_{\rm vir,avg}) - (0.103 \pm 0.026)$ at $z = 6$. The slopes of these equations, as a function of redshift, are fit by Slope $= -(7.3 \pm 1.9) \times 10^{-3} z + (3.7 \pm 1.4) \times 10^{-2}$. This is not visible in the symmetrical Δc distributions, as the trends are roughly centered about zero and are washed away when integrated across the entire mass range.

We have found that choice of initialization technique can play a significant role in the properties of halo populations during the pre-reionization dark ages. The early halo growth displayed 2LPT simulations, or conversely the delayed halo growth arising from the approximations made in ZA-initialized simulations, makes careful attention to simulation initialization imperative, especially for studies of halos at high redshift. It is recommended that future N-body simulations be initialized with 2LPT, and that previous high-z or high-mass halo studies involving ZA-initialized simulations be viewed with the potential offsets in halo mass and concentration in mind.

This work was conducted using the resources of the Advanced Computing Center for Research and Education (ACCRE) at Vanderbilt University, Nashville, TN. We also acknowledge the support of the NSF CAREER award AST-0847696. We would like to thank the referee for helpful comments, as well as the first author's graduate committee, who provided guidance throughout this work.

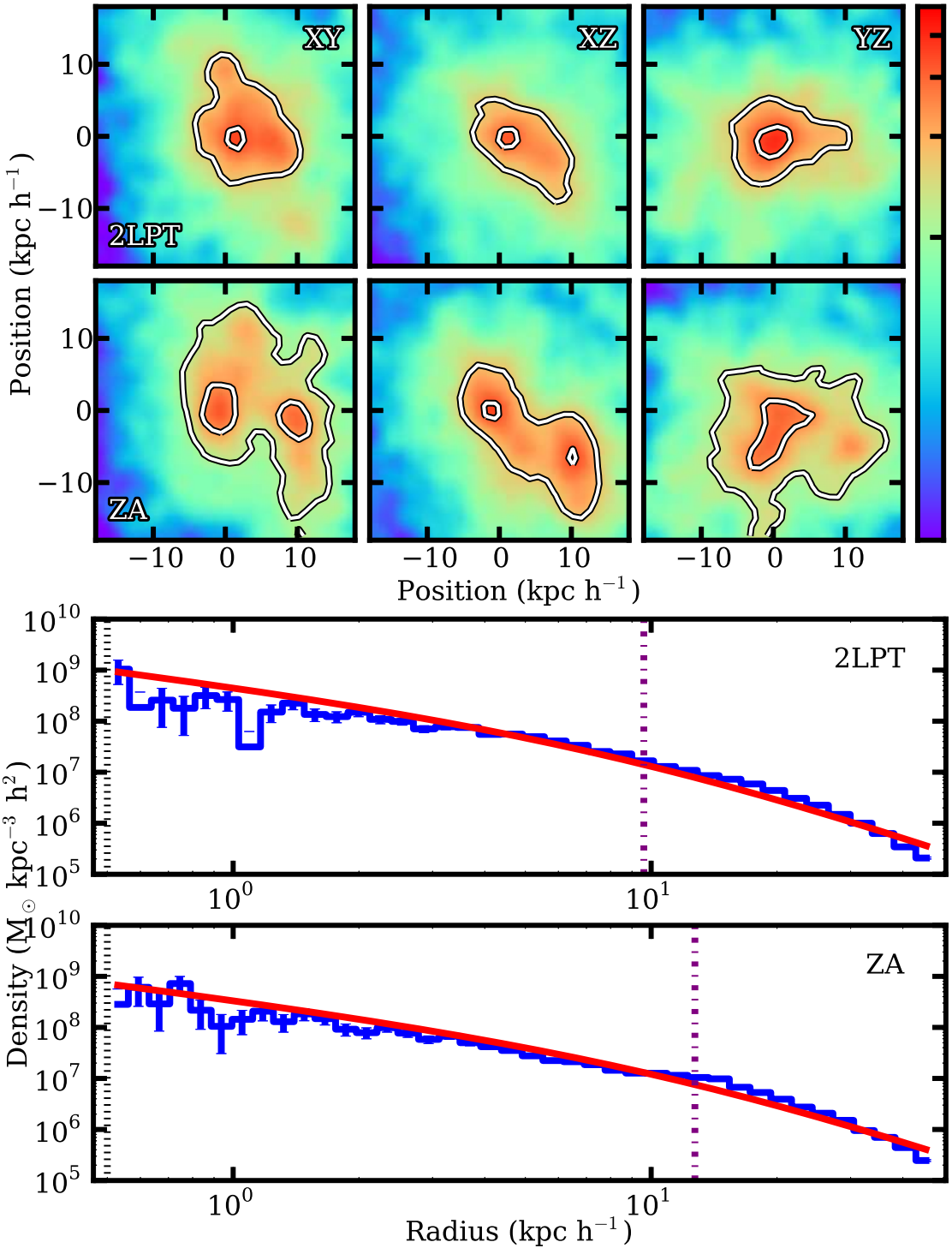
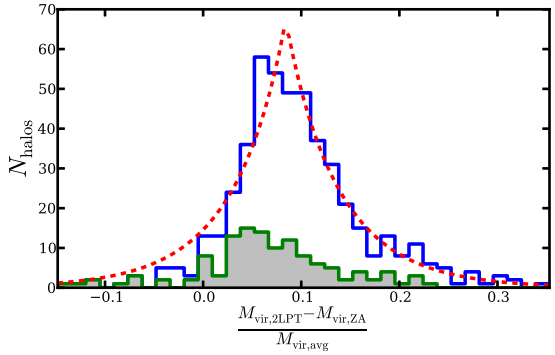
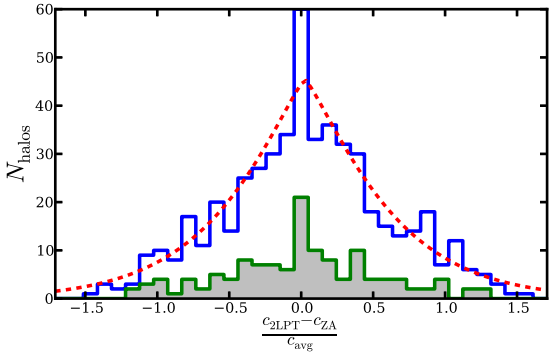
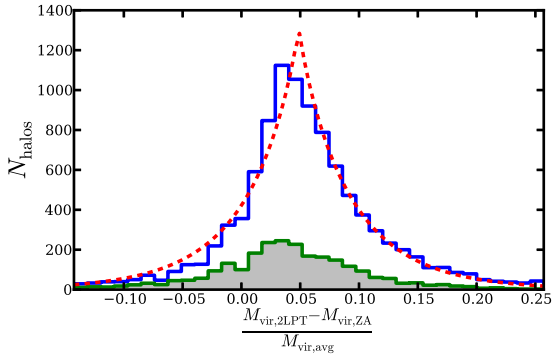
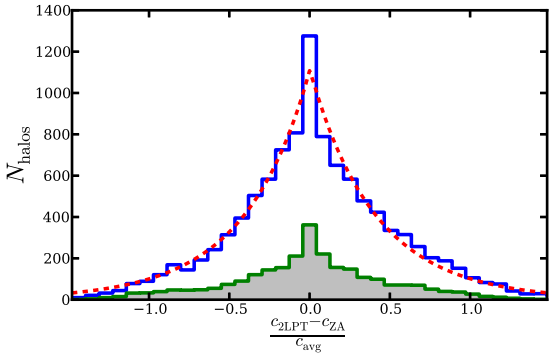


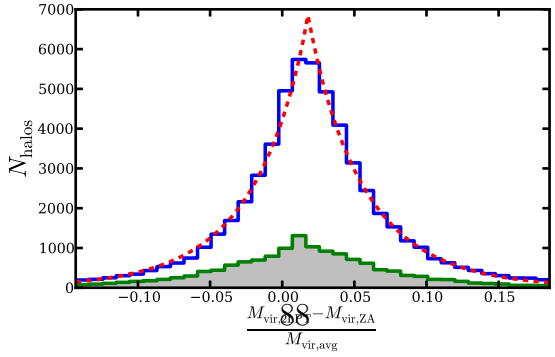
Figure III.1: *Top two rows:* Density projections for two matching halos at $z = 6$. The first and second row are 2LPT and ZA, respectively. The halos appear to be either undergoing or have recently undergone a major merger. The 2LPT halo appears to be more relaxed and further along in the merger process, while the ZA halo lags behind, still displaying two distinct cores. The halos have masses of $5.95 \times 10^9 M_\odot$ for 2LPT and $5.85 \times 10^9 M_\odot$ for ZA. *Bottom two rows:* Density profiles for the same two halos as above. NFW profiles are fit to logarithmic radial bins of particle position and are overplotted as red curves. The purple dot–dash lines mark the scale radii. The black dotted lines mark the resolution limit of the simulations.


$N_{\rm halos}$
$\frac{M_{\rm vir,2LPT}-M_{\rm vir,ZA}}{M_{\rm vir,avg}}$


$N_{\rm halos}$
$\frac{c_{\rm 2LPT}-c_{\rm ZA}}{c_{\rm avg}}$


$N_{\rm halos}$
$\frac{M_{\rm vir,2LPT}-M_{\rm vir,ZA}}{M_{\rm vir,avg}}$


$N_{\rm halos}$
$\frac{c_{\rm 2LPT}-c_{\rm ZA}}{c_{\rm avg}}$


$N_{\rm halos}$
$\frac{M_{\rm vir,2LPT}-M_{\rm vir,ZA}}{M_{\rm vir,avg}}$

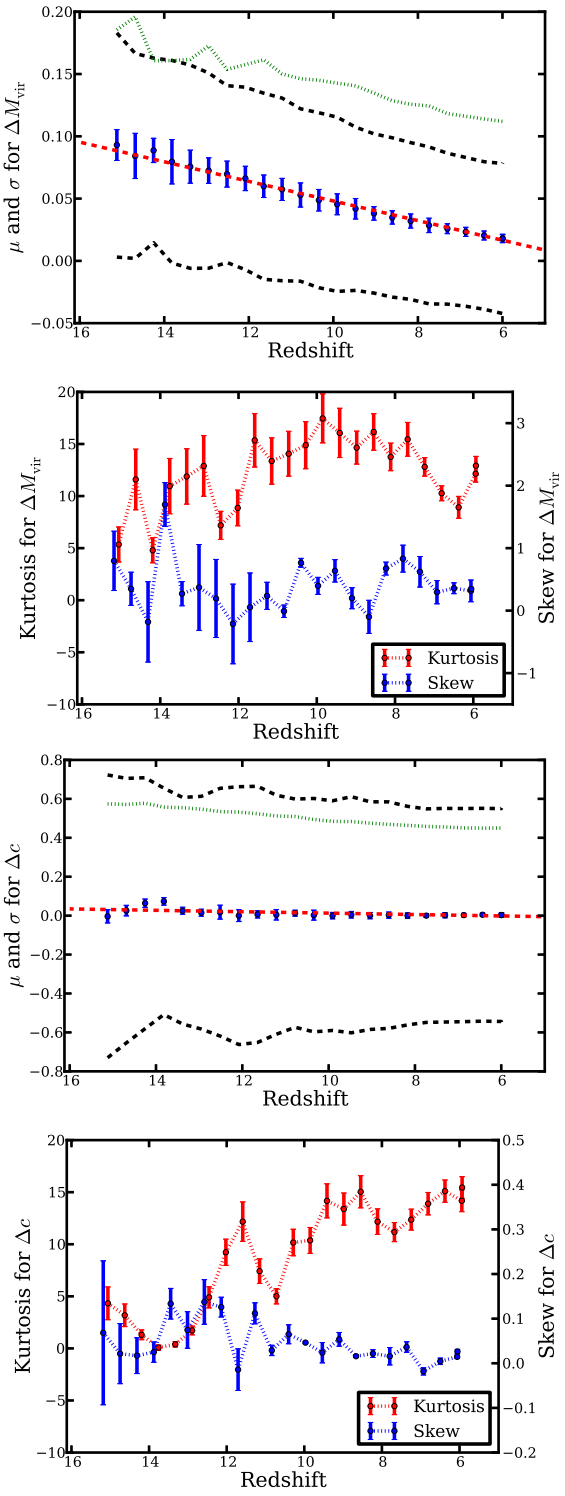
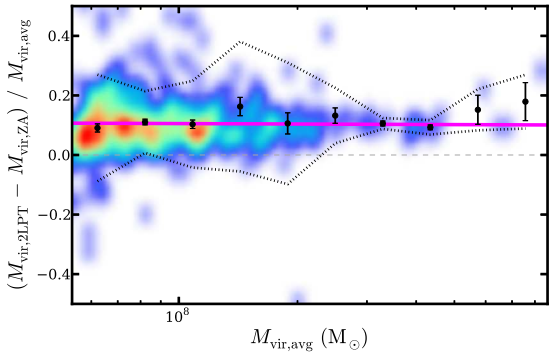
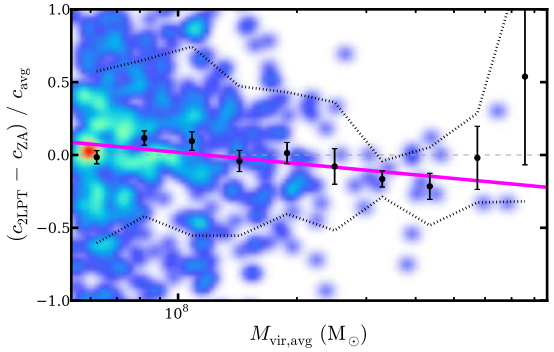
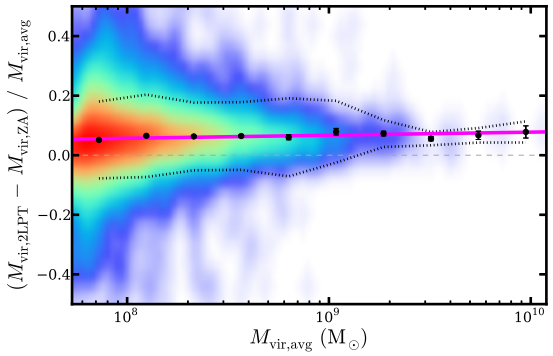
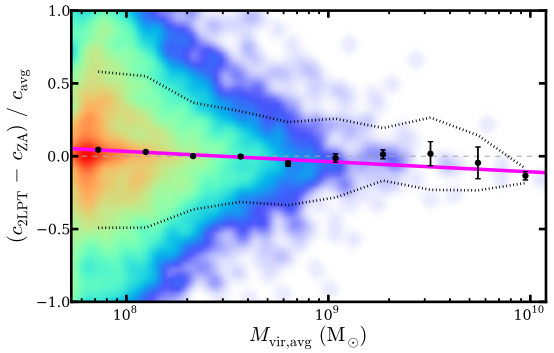


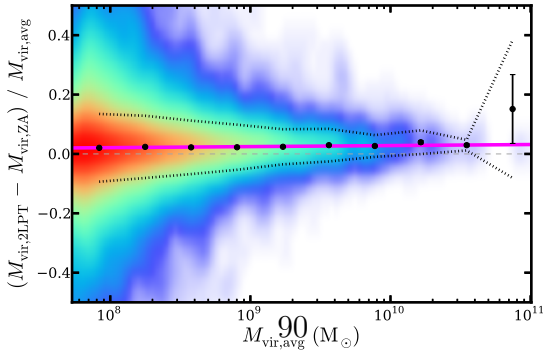
Figure III.3: Mean, standard deviation, and RMS (*left column*) and skew and excess kurtosis (*right column*) as functions of redshift for $\Delta M_{\rm vir}$ (*top row*) and Δc (*bottom row*). In the left column, μ is plotted as blue points, and $\mu \pm \sigma$ is plotted as the black dashed line, and RMS values are plotted as a green dotted line. The red dashed line is a linear fit to the mean. We find a significant trend for μ for $\Delta M_{\rm vir}$ to be more positive at higher redshift and gradually shift toward zero as the simulation progresses, with a fit function of $\mu_{\Delta M_{\rm vir}} = (7.88 \pm 0.17) \times 10^{-3} z - (3.07 \pm 0.14) \times 10^{-2}$. The mean for Δc, however, remains at or very near zero for most of the simulation and is fit by $\mu_{\Delta c} = (3.62 \pm 0.95) \times 10^{-3} z - (2.34 \pm 0.84) \times 10^{-2}$. The $\Delta M_{\rm vir}$ and Δc distributions narrow over time, with a slight decrease in σ. In the right column, we plot skew (blue line) and excess kurtosis (red line). Skew is positive for much of the simulation for $\Delta M_{\rm vir}$, but is much smaller for Δc. Kurtosis is large (much more peaked than Gaussian) for both $\Delta M_{\rm vir}$ and Δc throughout much of the simulation, and especially at later redshift.


$(M_{\rm vir,2LPT} - M_{\rm vir,ZA}) / M_{\rm vir,avg}$
$M_{\rm vir,avg}$ ($M_\odot$)


$(c_{\rm 2LPT} - c_{\rm ZA}) / c_{\rm avg}$
$M_{\rm vir,avg}$ ($M_\odot$)


$(M_{\rm vir,2LPT} - M_{\rm vir,ZA}) / M_{\rm vir,avg}$
$M_{\rm vir,avg}$ ($M_\odot$)


$(c_{\rm 2LPT} - c_{\rm ZA}) / c_{\rm avg}$
$M_{\rm vir,avg}$ ($M_\odot$)


$(M_{\rm vir,2LPT} - M_{\rm vir,ZA}) / M_{\rm vir,avg}$
$M_{\rm vir,avg}$ ($M_\odot$)

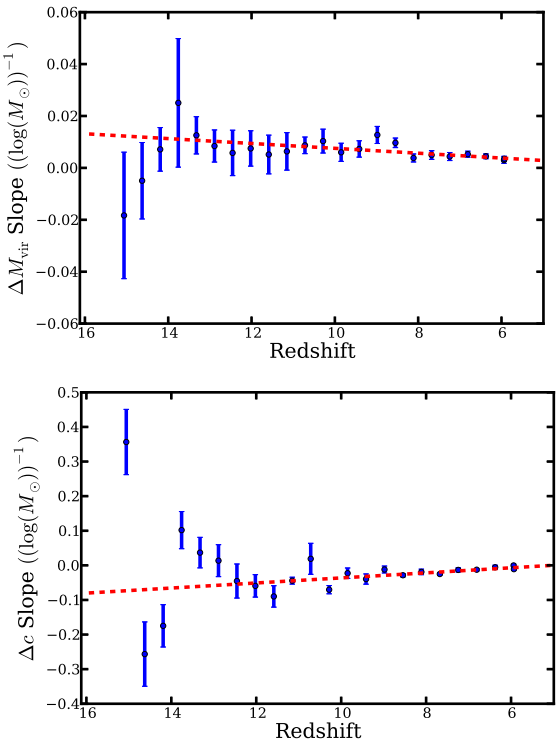


Figure III.5: Slopes of the Δq vs. $M_{\rm vir,avg}$ fit functions. The left and right panels correspond to the $\Delta M_{\rm vir}$ and Δc plots in the left and right columns, respectively, of Figure III.4. Linear least-squares fits to the data are overplotted as red dashed lines. Overall, we find a trend of positive and increasing slope with redshift for $\Delta M_{\rm vir}$ and negative and decreasing slope with redshift for Δc. We find fit equations of Slope $= (9.4 \pm 2.4) \times 10^{-4} z - (1.8 \pm 1.8) \times 10^{-3}$ for $\Delta M_{\rm vir}$ and Slope $= -(7.3 \pm 1.9) \times 10^{-3} z + (3.7 \pm 1.4) \times 10^{-2}$ for Δc. Snapshots at very high redshift, $z \gtrsim 14$ for $\Delta M_{\rm vir}$ and $z \gtrsim 13$ for Δc, begin to deviate from these trends. However, it is uncertain if this deviation is significant due to the low number statistics of our sample at such high z.

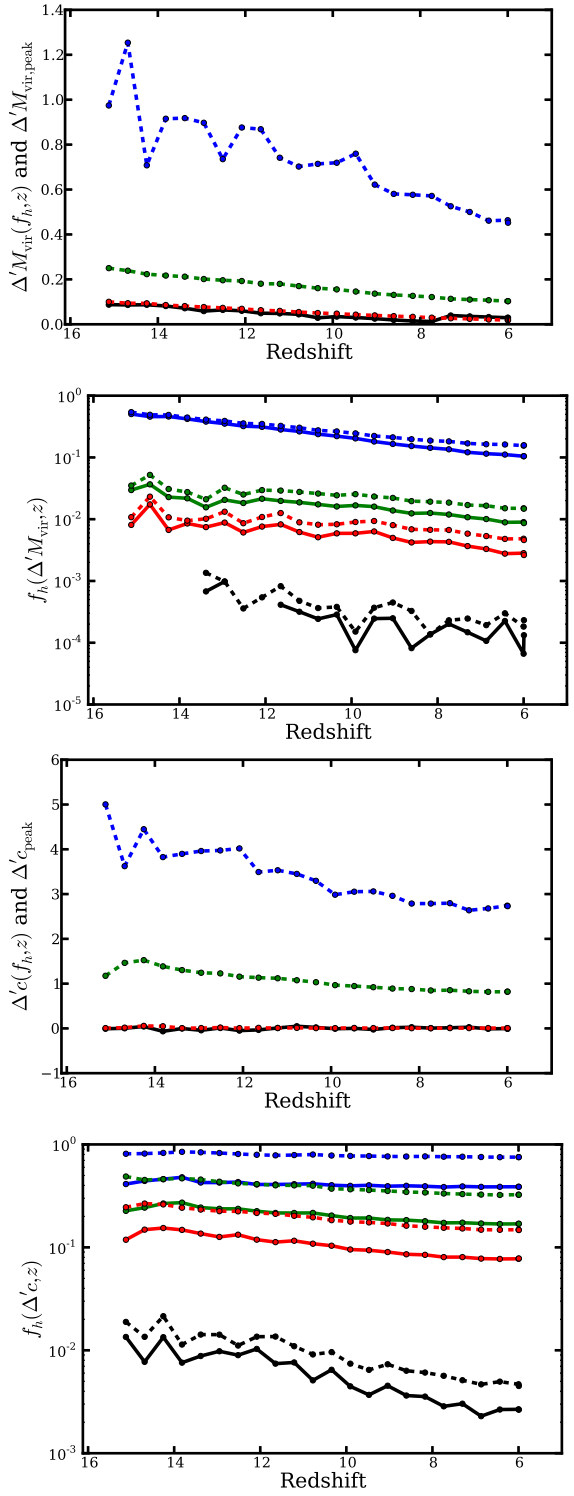


Figure III.6: Fractional error distributions statistics for $\Delta' M_{\rm vir}$ (*top row*) and $\Delta' c$ (*bottom row*) as functions of redshift. *Left column*: The $\Delta' q$ of the peak of the distribution (black line), and the $\Delta' q$ where 50% (red dashed line), 10% (green dashed line), and 1% (blue dashed line) of the halos fall at or above $\Delta' q$. As with distributions of $\Delta M_{\rm vir}$, $\Delta' M_{\rm vir}$ has the largest positive displacement at high redshift and steadily decreases throughout the simulation. Additionally, $\Delta' c$ maintains a peak near zero and has a spread much larger than that of $\Delta' M_{\rm vir}$. *Right column*: The fraction of halos with $\Delta' q$ greater than 0.10 (solid blue line), 0.50 (solid green line), 1.00 (solid red line), and 4.00 (solid black line). The dashed lines additionally count halo pairs with $\Delta' q$ lower than the corresponding equivalent displacements of -0.09, -0.33, -0.50, and -0.80, respectively (see Equation III.22). We find that 50% of 2LPT halos are at least 10% more massive than their ZA companions at $z = 15$, reducing to 10% by $z = 6$. Halos in 2LPT are at least twice as concentrated for 12% of halos at $z = 15$ and 7.8% of halos at $z = 6$.

CHAPTER IV

Supermassive Black Holes and Their Hosts

IV.1 Introduction

The study of the evolution of galaxies and the growth of the supermassive black holes at their cores go hand in hand. Although the typical length scales for the two can vary by many orders of magnitude, they seem inexorably linked. Observational correlations between galaxy and supermassive black hole properties hint at an underlying co-evolution driven by shared mechanisms.

IV.1.1 Galaxy Properties

How do we describe a galaxy? Being extended, resolvable objects, galaxies provide a unique wealth of observable characteristics not obtainable from point sources such as stars. While many characteristics can be deduced about point sources, the actual observations themselves come down to measuring position on the sky and measuring flux as a function of frequency and time. From this information, all that we know about stars and other point sources, such as temperature, age, size, and composition, can be inferred. However, for extended objects like galaxies, we are given more to work with.

IV.1.1.1 Color

A galaxy's color is determined by its stellar component. While a galaxy in itself may be resolvable, for all but the most nearby of galaxies, individual stars are not. What we see when looking at a particular small section of a galaxy is the averaged-together light from stars in that section.

Broadly, bluer late-type spirals have a $u-r$ color of around $1.3-2.0$, while redder early-type galaxies have a $u-r$ color of around $2.3-2.7$. The color of a galaxy can be a

good indicator for its age and evolutionary stage. Star formation processes generally tend to produce many smaller, cooler, redder stars and fewer larger, hotter, bluer stars. These small, cool stars are much longer-lived than their massive counterparts, while the large, warm stars are much brighter. After star formation turns off, the short-lived blue stars begin to die off, and the galaxy becomes redder, as more of the fraction of total light comes from the red end of the population.

IV.1.1.2 Morphology

The extended nature of galaxies allows us to observe their morphology. The classification scheme originally devised by Hubble (1926) places galaxies into the four broad categories: elliptical, spiral, lenticular, and irregular. Elliptical galaxies tend to be larger, redder, have less gas, and dominated by more radial orbits. Spiral galaxies tend to be smaller, bluer, have more gas, and have more of a disk component. Spirals can have a number or arms, a central bulge, and a central bar. Lenticular galaxies are middle-of-the-road galaxies, with both a strong central bulge like an elliptical, and an extended disk like a spiral, however without spiral arms. Irregular galaxies tend to defy this simple classification scheme, and can be found in any number of configurations.

Figure IV.1 is a cartoon of the classification scheme. To the left of the diagram are elliptical galaxies. The subcategories are an indication of the shape of the galaxy, with the most spherical on the left and progressing to more flattened shapes to the right. On the right of the diagram are spiral galaxies. These are broken into two branches, based on whether or not the galaxy contains a central bar. Moving from right to left, the spiral arms of the galaxies become more tightly wound, and the central bulges become more dominant. At the center of the diagram where the spiral fork meets the elliptical line, lie lenticular galaxies. Irregular galaxies are, as the name would imply, irregular and do not fall on the diagram.

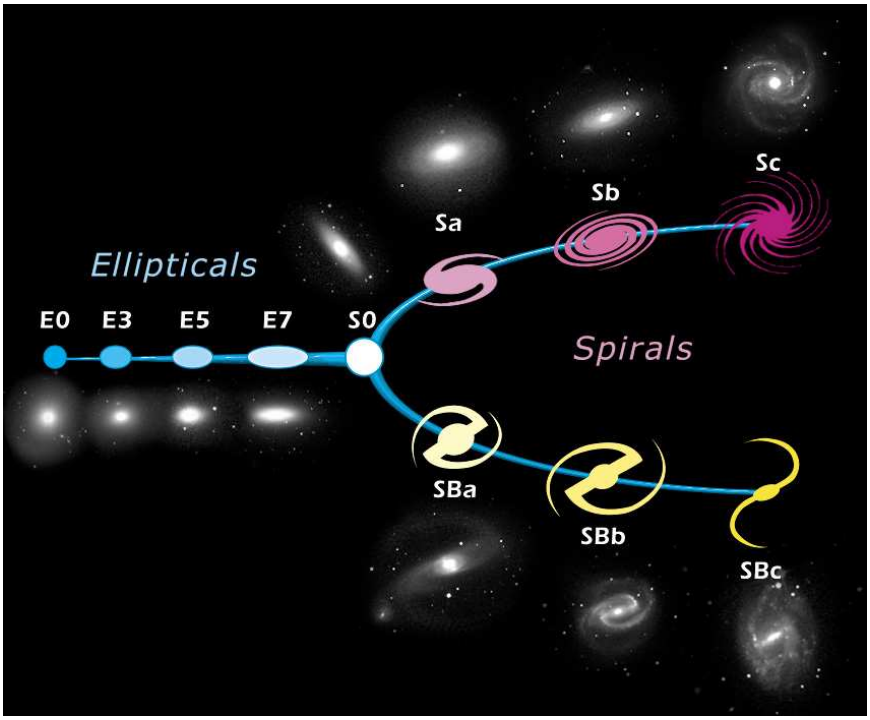


Figure IV.1: The Hubble tuning fork. On the left of the diagram are elliptical galaxies. E0 galaxies are the most spherical, while E7 are the most flattened or elongated. S0 are lenticular galaxies. The top branch on the right are spiral galaxies with no bar, while the bottom right branch are spiral galaxies with a bar. Both progress from tightly wound spiral arms and large bulges to loosely wound spiral arms and small to no bulges, going from Sa to Sc or SBa to SBc.

IV.1.2 Supermassive Black Hole Properties

A non-merging black hole, much like an elementary particle, can be described simply by its mass, charge, and spin. Its effect on its local spacetime, infalling matter, and surrounding environment all come back to these three parameters. However, determination of these parameters and the study of how black holes interact with their surroundings can be quite involved.

Black holes are, by their very nature, black, and difficult to observe. We cannot see light emitted directly from a black hole as we would a star, since a black hole is defined as an object massive and compact enough to not allow light within its event horizon to escape. We are forced, therefore, to employ other methods of measuring

black holes.

Thus far, the majority of progress in the measurement of black hole properties has been in measuring mass. There are a number of ways to measure the mass of a black hole. Here, we will briefly discuss masers, stellar dynamics, gas dynamics, and reverberation mapping as methods of measuring a supermassive black hole's mass.

Astrophysical masers are sources of stimulated spectral line emission in the microwave band formed in regions of high-density gas comprised of molecules such as hydroxyl, formaldehyde, and water (Lo, 2005). Since the emission frequencies of these sources are very well constrained, high-accuracy Doppler shifts can be determined. These Doppler shifts can then be used to determine velocities for the masers, and thus how much mass is enclosed by their orbits. If these masers lie very close to the supermassive black hole (SMBH) in the center of their galaxy, the enclosed mass can be constrained to be primarily that of the SMBH.

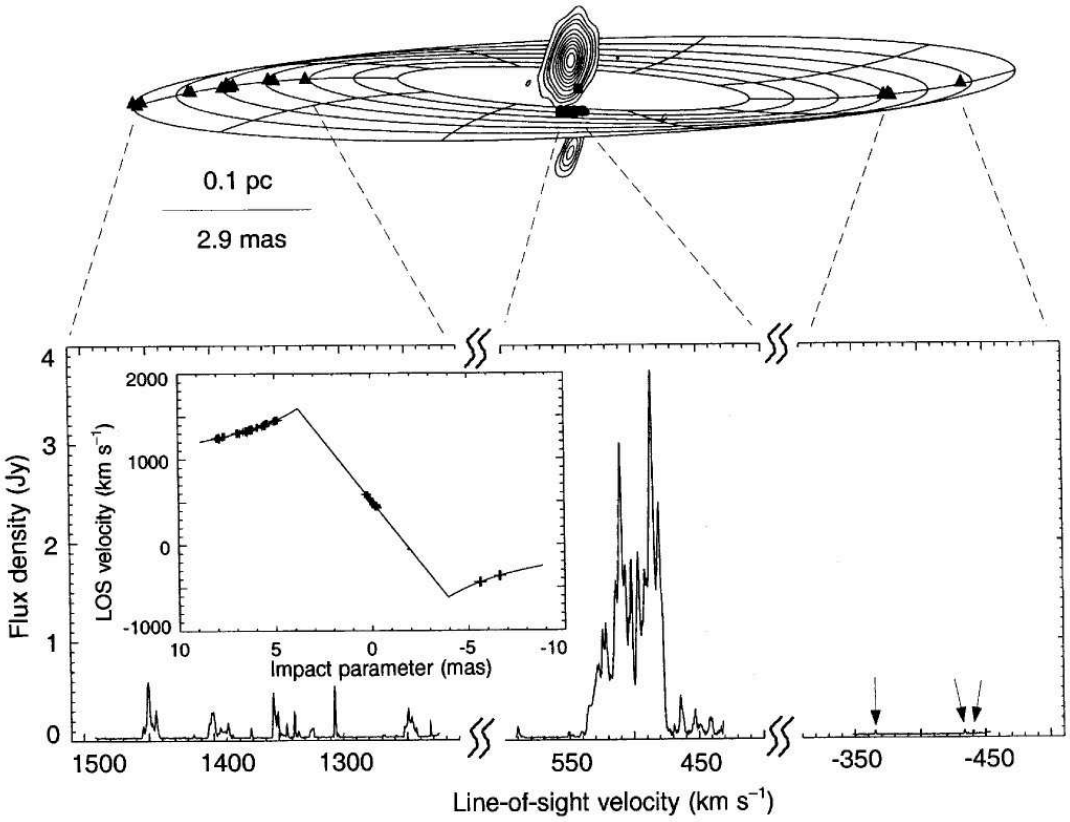


Figure IV.2: Maser orbits fit to a warped disk for NGC4258. Masers can also be useful for distance determinations. Here, the positions and velocities of water masers are able to be fit to a warped disk model surrounding a supermassive black hole. This allows the interpolation of physical radii away from the black hole, giving us both the black hole mass and an standard ruler to allow precise determination of the distance to NGC4258. (Herrnstein et al., 1999)

Stellar dynamics and gas dynamics both probe light coming from matter near the black hole. The width of broadened spectral lines from either the stars or gas can be used to determine a velocity dispersion for the matter local to the SMBH. This velocity dispersion, therefore, can then be used to determine the potential through which the matter is traveling, and thus the mass of the black hole.

A special case of stellar dynamics for which the orbits of the constituent stars can be resolved—namely, for the case of our own Milky Way—adds another dimension to our knowledge of the stellar orbits. Over time, we can observe the proper motion on the sky for these orbits. Combining these measurements with Doppler measurements

for radial velocity yields full orbital solutions. Then, it simply requires Kepler's laws to determine the mass of the SMBH.

Reverberation mapping can be thought of as "echo-mapping" the gas disk around a SMBH. Continuum emission very near the black hole travels outward and stimulates broad line emission in surrounding gas. Any changes in the continuum emission will take time to propagate to the broad line region, since the speed of light is finite. By measuring the timing difference in the change in continuum emission and change in stimulated broad line emission, the physical distance from the SMBH to the broad line region can be inferred. With this radius, and the velocity of the gas in the broad line region measured by the width of the broadened lines, a black hole mass can be determined (Blandford & McKee, 1982).

IV.1.3 Correlations

Correlations between varying properties of galaxies and black holes can provide much deeper insight into the dynamics that shape the evolution of both. Of particular interest here are the fundamental plane of elliptical galaxies, the $M - \sigma$ relation, and the green valley-AGN relation.

IV.1.3.1 The M-Sigma Relation

If we consider the all the observable properties of a galaxy and compare them to the mass of its SMBH, the tightest correlation can be found with the velocity dispersion σ of the galaxy's bulge. Such a tight correlation is surprising, as the sphere of influence of a typical SMBH does not extend much past order a few pc, while bulges exist on scales of a kpc or greater. In essence, the supermassive black hole and the outer edges of the bulge shouldn't "feel" each other. Nevertheless, the correlation is indeed there, suggesting some mechanism that influences—or is influenced by—both of them. Gültekin et al. (2009) use a sample of 49 M_{BH} measurements and 19 upper limits to measure this correlation, and find $\log(M_{BH}/M_{\odot}) = \alpha + \beta \log(\sigma/200 \text{ km s}^{-1})$

with $(\alpha, \beta, \epsilon_0) = (8.12 \pm 0.08 M_\odot, 4.24 \pm 0.41 M_\odot, 0.44 \pm 0.06 M_\odot)$ for all galaxies and $(\alpha, \beta, \epsilon_0) = (8.23 \pm 0.08 M_\odot, 3.96 \pm 0.42 M_\odot, 0.31 \pm 0.06 M_\odot)$ for ellipticals, where ϵ_0 is the intrinsic scatter in the relation.

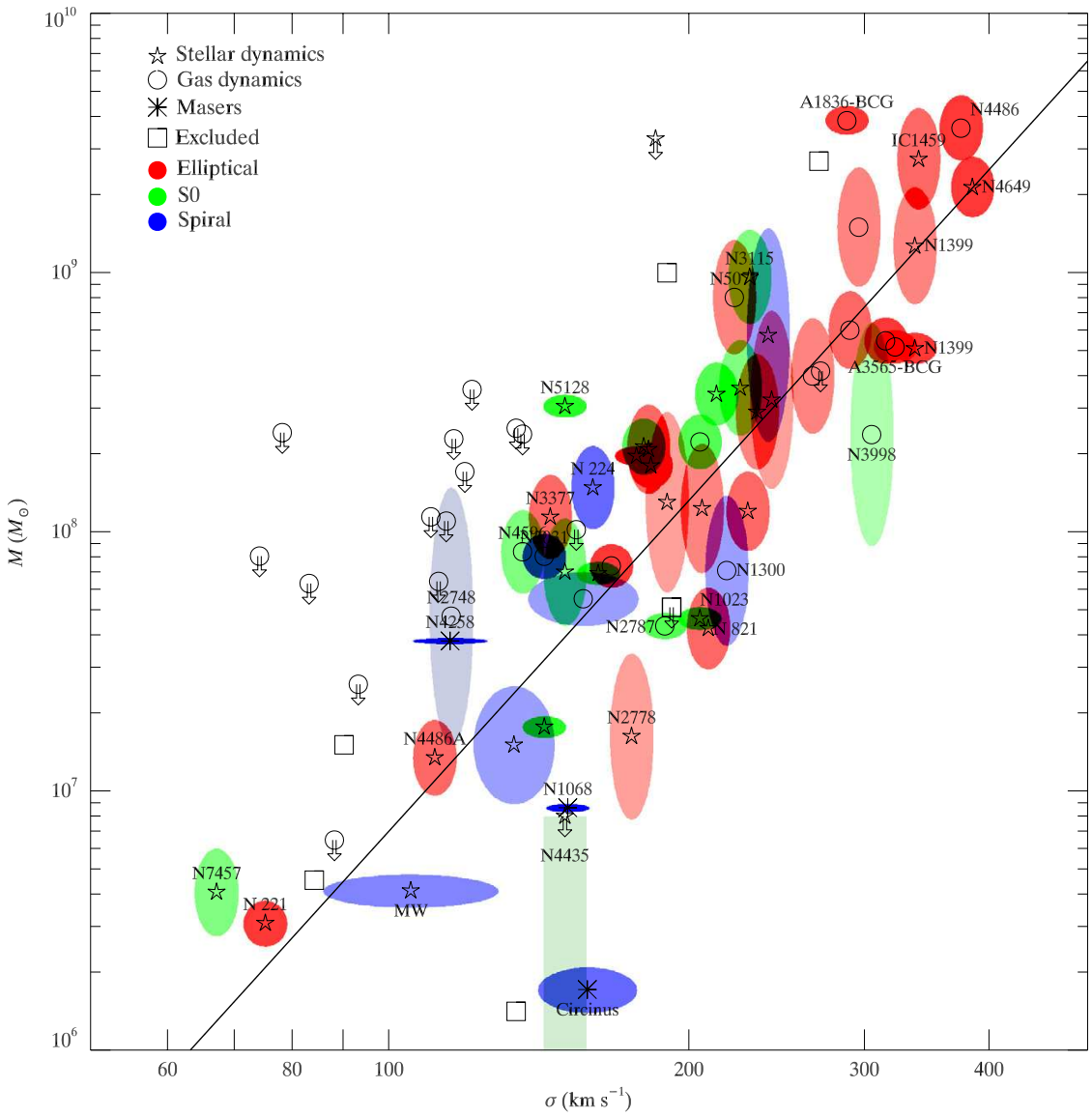


Figure IV.3: The M-σ relation for galaxies with dynamical measurements. Black hole mass is plotted vs velocity dispersion of its host spheroid. The symbols represent the method by which the black hole mass was measured: pentagrams for stellar dynamics, circles for gas dynamics, and asterisks for masers. Upper limits are given by arrows. Error ellipses are colored by galaxy type, with red for ellipticals galaxies, green for lenticular galaxies, and blue for spiral galaxies. The saturation of the color is inversely proportional to the area of the ellipse. For this sample, the best fit relation is $M_{BH} = 10^{8.12}\ \mathrm{M}_\odot(\sigma/200\ \mathrm{km\ s^{-1}})^{4.24}$. Galaxies not included in this fit are labeled as squares. (Gültekin et al., 2009)

IV.1.3.2 The Fundamental Plane

While not a direct correlation with the properties of supermassive black holes, the fundamental plane of elliptical galaxies offers insight into the characteristics of their hosts. The fundamental plane is a three-parameter correlation between properties of elliptical galaxies: velocity dispersion, effective radius, and surface brightness. This correlation (Figure IV.4) between these three parameters is tighter than the combination of any two alone (Djorgovski & Davis, 1987). The fit for this correlation can be given as $\log R_e = 0.36(\langle I \rangle_e / \mu_B) + 1.4 \log \sigma_0$, where R_e is the effective radius in kpc, $\langle I \rangle_e$ is the mean surface brightness interior to R_e in units of μ_B, and σ_0 is the velocity dispersion in km s^{-1} (Binney & Merrifield, 1998).

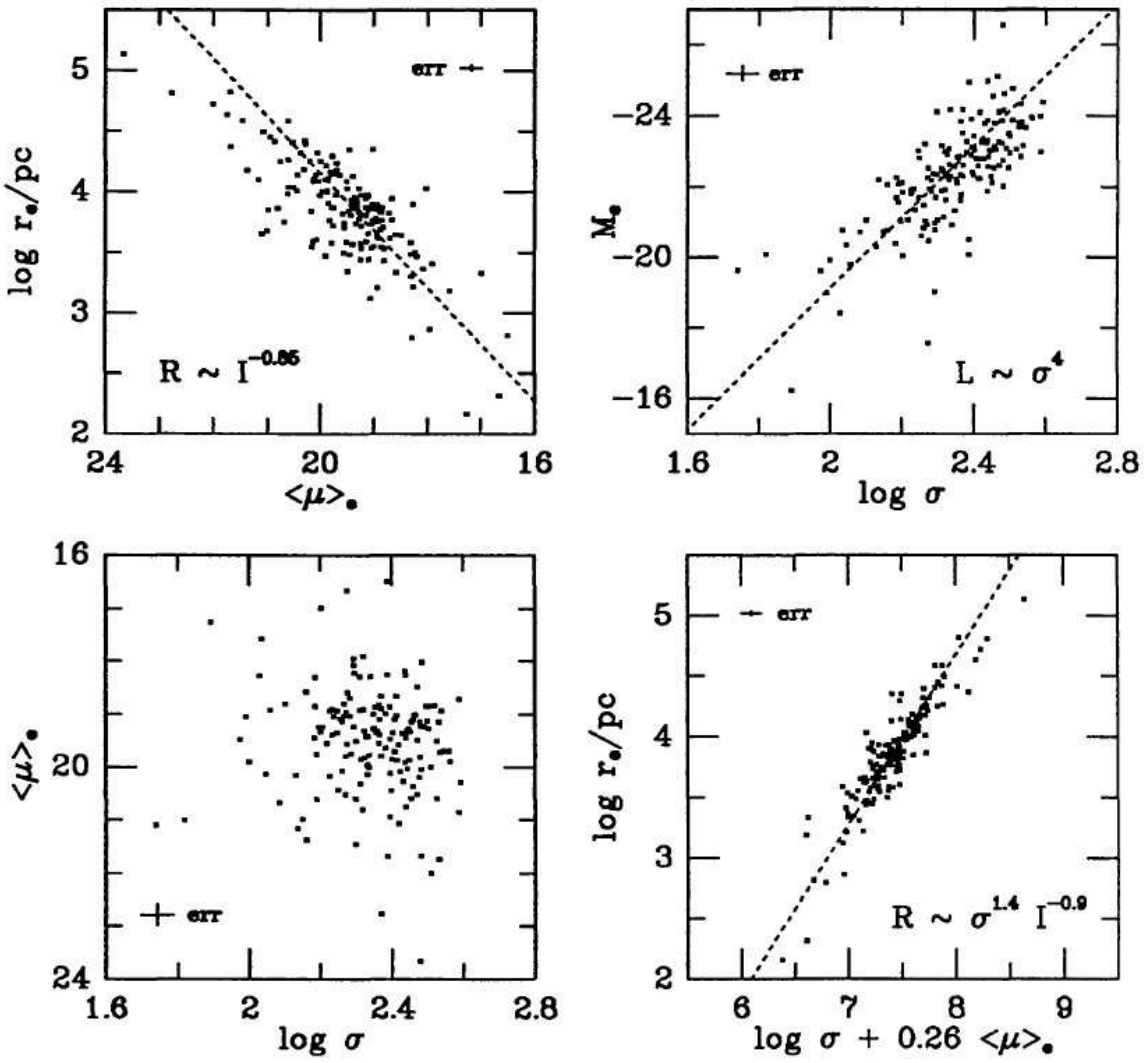


Figure IV.4: The fundamental plane for elliptical galaxies. *Top panels:* The top panels show the one-parameter scaling relations, with the relation between radius and mean surface brightness on the left and the relation between luminosity and velocity dispersion (the Faber-Jackson relation) on the right. *Bottom left:* The relation between the surface brightness and velocity dispersion. This is an almost face-on view of the fundamental plane. *Bottom right:* The relation between the effective radius and the combination of surface brightness and velocity dispersion. This is the edge-on view of the fundamental plane. (Kormendy & Djorgovski, 1989)

IV.1.3.3 The Green Valley

When considering both the color and stellar mass of a galaxies, a correlation emerges where many galaxies lie in either the "blue cloud" of bluer, lower mass galaxies, or

the "red sequence" of redder, generally higher mass galaxies. The area between these two is known as the "green valley" and, while not as populated as the blue cloud or red sequence, holds special interest when active galactic nuclei (AGN) are considered. AGN are very luminous regions at the centers of some galaxies. Schawinski et al. (2010) show that galaxies falling on the green valley are much more likely to host AGN than galaxies on the blue cloud or red sequence, hinting at an underlying link between the evolution of galaxies, and the activity at their centers.

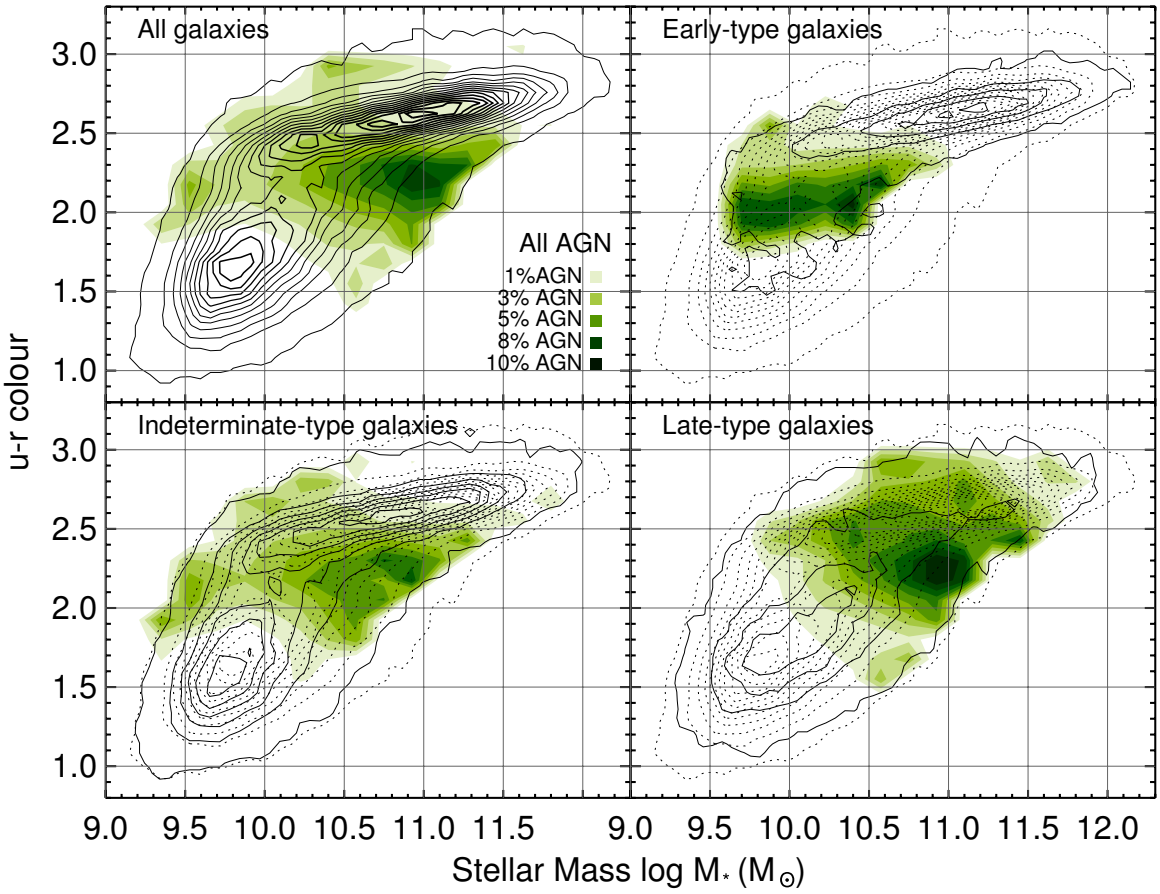


Figure IV.5: Distribution of the fraction of galaxies containing AGN. Galaxy color in u-r is plotted vs stellar mass. The contours are the galaxy population for all galaxies (top-left), early-type galaxies (top right), intermediate-type galaxies (bottom left), and late type galaxies (bottom right). For the three sub-samples, dotted contours represent the full sample for comparison. The green shaded contours represent the fraction of galaxies in that subsample that contain active galactic nuclei. It can be clearly seen that the AGN fraction is highest for galaxies falling within the green valley. (Schawinski et al., 2010)

IV.2 Galaxy Evolution

IV.2.1 Dark Matter Halos

Every galaxy resides inside a dark matter halo. Often about an order of magnitude larger in both radius and mass than the baryonic component, dark mater halos dominate the large-scale behavior of galaxies. Dark matter is matter that is thought to interact very weakly or not at all with light and ordinary matter, except gravita-

tionally. Evidence for dark matter comes from a number of sources, including the relatively flat rotational velocity curve of galaxies, the velocity dispersion of galaxies, gravitational lensing measurements, galaxy clustering, and the offset between the gas and dominant mass measured in the Bullet cluster. Here we will briefly discuss the evidence from flat rotation curves.

If there were no dark matter component and only the baryonic components (i.e. stars and gas) contributed to the galactic potential, we would expect the rotational velocity of galaxies to fall off with radius. However, observations show that the rotation curve remains relatively flat (Rubin et al., 1980). Figure IV.6 shows several observed rotation curves.

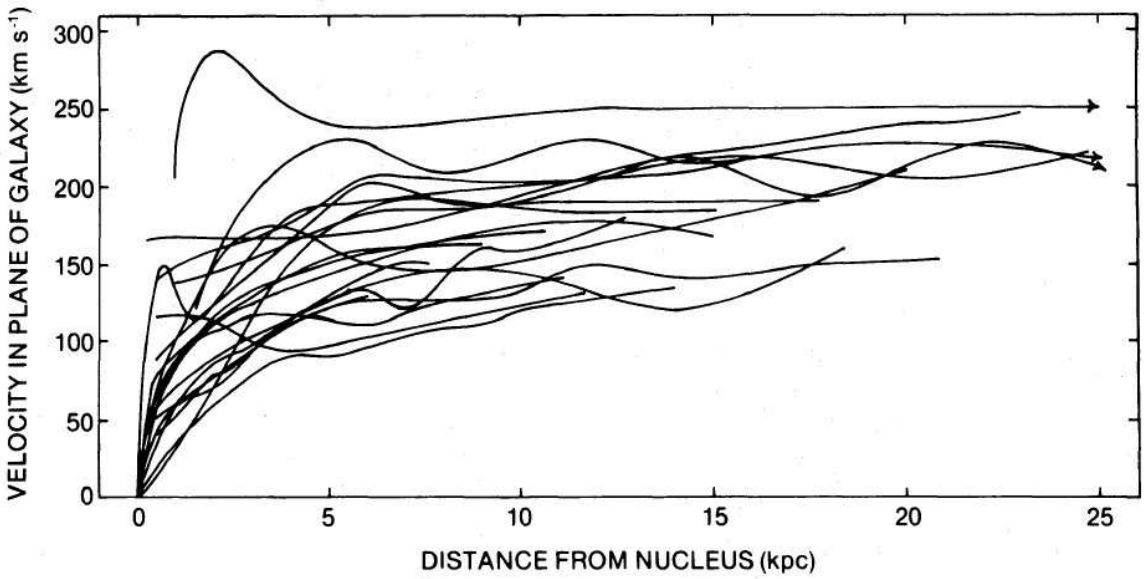


Figure IV.6: Rotation curves for 21 Sc galaxies. It is readily identifiable that the rotation curves do not fall off as would be expected for galaxies without a dark matter component. (Rubin et al., 1980)

Navarro et al. (1997b) found that dark matter halos generally follow the same density profile, regardless of mass. This universal dark matter density profile can be given as

$$\rho(r) \propto \frac{1}{(r/a)(1+r/a)^2}, \tag{IV.1}$$

where a is the radius where the profile transitions from an r^{-1} power law to an r^{-3} power law.

IV.2.2 Galaxy Mergers

Galaxy mergers are the fundamental mechanism by which galaxies grow and evolve. Collisions between galaxies trigger processes that can alter nearly all the properties of the galaxies. Naturally, mergers increase the mass of galaxies. Starting from small perturbations in the early universe, gravity slowly pulls matter together to form larger and larger clumps. These clumps of gas and dark matter eventually form stars, beginning what we think of as typical galaxies, and over time, these galaxies merge together into larger and larger galaxies.

Mergers affect many other properties of galaxies as well. Mergers distort the shapes of galaxies, causing long tidal tails to form and the entire morphology to appear irregular. The disk structures of spiral galaxies that form from the settling of the rotational component are distorted and "puffed up" into components with ever increasing bulge-like properties.

Mergers can trigger wide-scale starburst events, where a large portion of gas goes into the formation of stars. Much of the gas component of the galaxy can subsequently be blown out by the winds from the supernovae of short-lived O and B stars. This shuts off star formation, and as the stellar population is no longer replenished with new high-mass stars, the galaxy becomes progressively redder as large stars die.

The general trend is for mergers to move galaxies from the right side of the Hubble tuning fork towards the left, turning blue, gas rich spirals into red, gas poor ellipticals. This process is aided by the AGN feedback also triggered during galaxy mergers, as we discuss in the following section.

IV.3 Supermassive Black Hole Growth

Supermassive black holes grow by two primary mechanisms, binary mergers and gas accretion. Through a combination of these, black holes can grow to as large as $\sim 10^9$–10^{10} $M_\odot$ by $z = 0$.

IV.3.1 Binary Mergers

When two galaxies merge, the supermassive black holes at their hearts begin a process that will eventually lead to their coalescence. There are generally thought to be three stages to this journey. First, the black holes sink towards the center of the merged galaxy through mass segregation and dynamical friction until they form a bound orbit with each other. Then, the black holes tighten their orbit through three-body scattering of nearby stars. Finally, as the black holes become close enough together for general relativistic effects to come into play, gravitational waves are emitted and radiate away the remaining orbital energy until the binary coalesces.

IV.3.1.1 Dynamical Friction and Inspiral

During the majority of the inspiral process, the black holes do not "feel" each other's gravitational pull. Instead, interactions with the galaxy itself push the holes together.

As it travels through a galaxy, a black hole—or any massive body—is slowed by the surrounding field of matter. Gravitational attraction pulls surrounding matter toward the black hole. However, as the black hole is moving with respect to the local medium, the attracted particles will tend to fall behind the black hole. This creates a wake of overdensity that gravitationally attracts the black hole from behind and slows its velocity. Chandrasekhar (1943) develops this notion of dynamical friction for the motion of a star through a sea of other stars. If the distribution of velocities of the surrounding particles is Maxwellian, the acceleration on the black hole can be

written as

$$\frac{d\mathbf{v}_M}{dt} = -\frac{4\pi G^2 M \rho \ln\Lambda}{v_M^3}\left[\mathrm{erf}(X) - \frac{2X}{\sqrt{\pi}}e^{-X^2}\right]\mathbf{v}_M, \tag{IV.2}$$

where $\mathbf{v}_M$ is the velocity of the black hole, M is it's mass, ρ is the density of surrounding matter, erf is the error function, $\ln\Lambda$ is the Coulomb logarithm, and $X \equiv v_M/(\sqrt{2}\sigma)$ where σ is the velocity dispersion of the surrounding medium (Binney & Tremaine, 1988). As the black hole is slowed by dynamical friction, it loses angular momentum and sinks towards the center of the galaxy's potential well.

IV.3.1.2 The Final Parsec Problem

Dynamical friction and mass segregation can only take us so far. Once the black holes are close enough together, they form a bound binary orbit. This generally occurs for separations of around a few to tens of parsecs. This presents a problem, however, since the orbit needs to shrink to around 10^{-2}–10^{-3} pc in order for gravitational wave emission to remove energy from the orbit in a significant amount. The orbit can be tightened with three-body scattering of stars that wander through the orbit of the binary, however, in the spherical galaxies where mergers often take place, there is a depletion of stars with orbits that intersect the binary. Khan et al. (2011), however, show that the non-spherical, triaxial potential typical of post-merger galaxy remnants can efficiently funnel stars through the orbit of the black hole binary with sufficient intensity to tighten the binary orbit to the gravitational wave regime.

IV.3.1.3 Gravitational Waves and Recoil Kicks

Once the black hole binary separation reaches the point where strong field general relativistic effects come into play, we no longer require external influences to nudge the black holes together. In the final plunge toward coalescence, the black hole binary sheds energy through emission of gravitational radiation. As energy is radiated away, the binary tightens its orbit until the two black holes merge into one. Following this

coalescence, the resultant black hole undergoes a "ringdown" phase, in which the distorted space time settles back down into a black hole that can again be simply described by mass, charge, and spin.

The emission of gravitational waves has two interesting consequences. First, the radiation from two merging supermassive black holes is extremely loud, and can potentially provide an observational signature of the process for gravitational wave observatories. Second, the gravitational waves carry linear momentum, leading to a recoil "kick" imparted to the black hole merger remnant.

Recent advances in numerical relativity simulations have provided a much deeper insight into the black hole binary merger process than has been previously available. Waveforms produced from these simulations (Figure IV.7) can be used to predict what gravitational wave observatories such as LIGO and LISA would expect to observe for signals originating from merging supermassive black hole binaries. Having these waveforms as templates for comparison to data can greatly increase the signal to noise ratio for these detectors, potentially allowing the gravitational wave events to be seen among the sea of noise. These waveforms produced from simulations of the last few orbits of inspiral through the merger and ringdown can be combined with waveforms suggested from post-Newtonian approximations for the longer duration inspiral to provide a complete extended signal to match against.

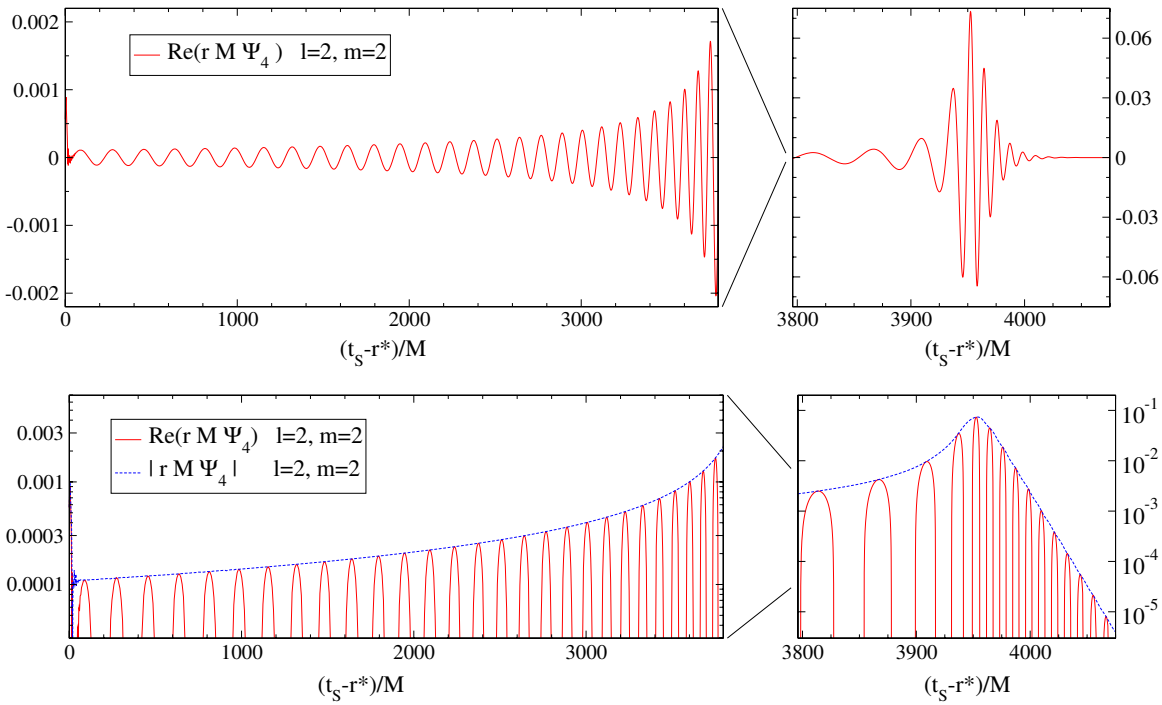


Figure IV.7: Gravitational waveform for an equal-mass, non-spinning black hole binary merger. This is the final waveform, extrapolated to infinity, from the numerical relativity simulation of Scheel et al. (2009). The waveform is shown on the top panel with a linear y-axis and on the bottom panel with a logarithmic y-axis. The left panels are the earlier stages of inspiral, and the right panels show the merger and ringdown stages.

For asymmetric mergers, gravitational radiation is emitted anisotropically. This causes a recoil kick, in which the gravitational waves impart a net velocity to the final black hole with respect to the original center of mass. The magnitude and direction of this kick are dependent on the mass ratio of the binary and the spins of the two black holes—in all, a 7-dimensional parameter space. This large parameter space has been largely explored with numerical relativistic simulations, and analytic equations can be fit to the data to predict the recoil from a given merger configuration. Holley-Bockelmann et al. (2008), give these equations as

$$\mathbf{v}_{kick} = (1+e)\left[\hat{\mathbf{x}}\left(v_m + v_\perp \cos\xi\right) + \hat{\mathbf{y}} v_\perp \sin\xi + \hat{\mathbf{z}} v_\parallel\right], \tag{IV.3}$$

where

$$v_m = A\frac{q^2(1-q)}{(1+q)^5}\left[1 + B\frac{q}{(1+q)^2}\right], \tag{IV.4}$$

$$v_\perp = H\frac{q^2}{(1+q)^5}\left(\alpha_2^\parallel - q\alpha_1^\parallel\right), \tag{IV.5}$$

$$v_\parallel = K\cos\left(\Theta - \Theta_0\right)\frac{q^2}{(1+q)^5}\left(\alpha_2^\perp - q\alpha_1^\perp\right). \tag{IV.6}$$

Here, the fitting constants are $A = 1.2\times10^4$ km s^{-1}, $B = -0.93$, $H = (7.3\pm0.3)\times10^3$ km s^{-1}, and $K = (6.0 \pm 0.1) \times 10^4$ km s^{-1}. The $\hat{z}$ unit vector is in the direction of the orbital angular momentum, and $\perp$ and $\parallel$ refer to components perpendicular and parallel to $\hat{z}$, respectively. The fitting parameters are the eccentricity e, the mass ratio $q \equiv M_2/M_1$, and the reduced spin parameters $\alpha_i \equiv S_i/M_i^2$ where S is the spin angular momentum. The orientation of the merger is given by the angles Θ, Θ_0, and ξ (Holley-Bockelmann et al., 2008).

Slices through this parameter space are shown in Figure IV.8. For certain configurations of the merger, the recoil velocity can be very high. Very asymmetric mergers can produce recoils as high as ~ 4000 km s^{-1}. These large recoils can be enough for the black hole to escape the potential well of its host galaxy and be ejected. Even less extreme recoil kicks can affect the evolution of black holes, as the kicked black hole can oscillate about its host's center, potentially changing its local gas environment and accretion rate.

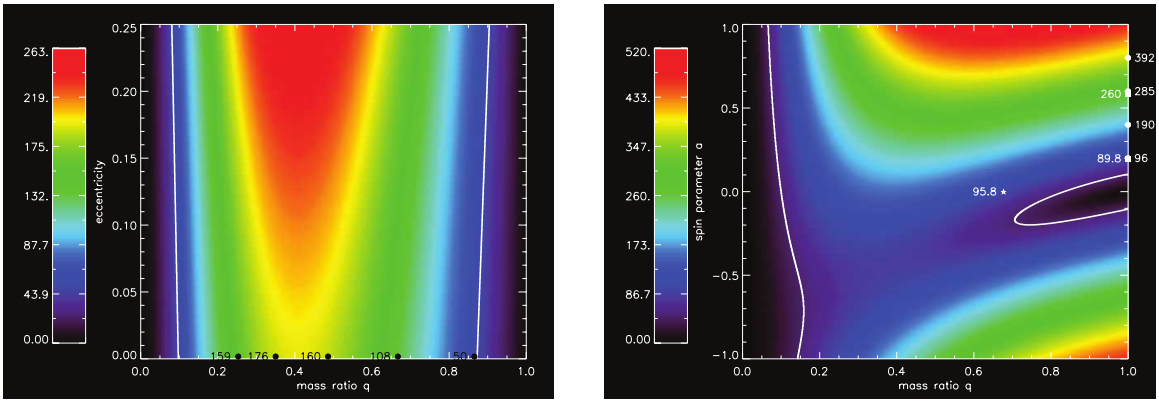


Figure IV.8: *Left:* Gravitaional wave recoil velocity from a merger of nonspinning black holes as a function of eccentricity and mass ratio. Data from numerical relativity simulations (González et al., 2007) are overlaid along the zero eccentricity line. The overlaid white contours are the escape velocity of a typical globular cluster, 50 km s^{-1}. *Right:* Gravitational wave recoil kick velocity as a function of spin parameter and mass ratio for a merger of spinning black holes on a circular orbit with spins perpendicular to the orbital plane of the binary and anti-aligned with each other. Again, the 50 km s^{-1} escape velocity of a globular cluster is overlaid as white contours. Results from numerical relativity simulations are over-plotted: squares for Koppitz et al. (2007), cirlces for Herrmann et al. (2007), and star for Brügmann et al. (2004). (Holley-Bockelmann et al., 2008)

IV.3.2 Accretion

Although mergers play an important role in the evolution of supermassive black holes, gas accretion can often dominate in terms of mass growth. Gas can fall into a black hole in a number of ways. Here, we will discuss accretion onto a moving black hole, spherical accretion onto a stationary black hole, and disk accretion onto a stationary black hole.

IV.3.2.1 Bondi-Hoyle-Lyttleton Accretion

Let us first consider a massive object, in this case our black hole, moving through a uniform density gas medium. Just as in the case of dynamical friction, particles close enough to the black hole will feel a gravitational attraction, causing them to move toward the black hole. As they move closer, the black hole is also moving

through the medium, causing the gas particles to focus behind the black hole. As the particle stream reaches the wake directly behind the black hole, it collides with opposing streams, causing the angular momentum to go to zero. If these particles are bound, they will proceed to fall onto the black hole. Hoyle & Lyttleton (1939) derive an impact parameter for which particles will be accreted,

$$\sigma < \sigma_{HL} = \frac{2GM}{v_\infty^2}, \tag{IV.7}$$

and a mass accretion from the wake column at a rate of

$$\dot{M}_{HL} = \pi \sigma_{HL}^2 v_\infty \rho_\infty = \frac{4\pi G^2 M^2 \rho_\infty}{v_\infty^3}, \tag{IV.8}$$

where v_∞ and ρ_∞ are the velocity and density far away from the black hole, respectively. Expanding upon this analysis, Bondi & Hoyle (1944) suggest that the accretion rate should rather be

$$\dot{M}_{BH} = \frac{2\alpha\pi G^2 M^2 \rho_\infty}{v_\infty^3}, \tag{IV.9}$$

where α is a constant between 1 and 2, with a typical value of around 1.25.

For an accretor at rest in an isotropic gas medium, one would expect accretion to be a spherical process. Bondi (1952) considers this configuration, and finds the accretion rate for this "temperature-limited" case to be

$$\dot{M}_{Bondi} = \frac{2\pi G^2 M^2 \rho_\infty}{c_{s,\infty}^3}, \tag{IV.10}$$

where $c_{s,\infty}$ is the speed of sound far away from the black hole.

Extrapolating between this result and the "velocity-limited" case of Equation IV.9

suggests (Bondi, 1952)

$$\dot{M}_{BH} = \frac{2\pi G^2 M^2 \rho_\infty}{\left(c_{s,\infty}^2 + v_\infty^2\right)^{3/2}} \tag{IV.11}$$

as an order of magnitude estimate of the more general case of accretion. Numerical simulations (Shima et al., 1985) suggest an additional factor of 2 is needed for better agreement with simulation results, giving us a generally applicable from for the accretion rate,

$$\dot{M}_{BH} = \frac{4\pi G^2 M^2 \rho_\infty}{\left(c_{s,\infty}^2 + v_\infty^2\right)^{3/2}}. \tag{IV.12}$$

IV.3.2.2 Disk Accretion and Active Galactic Nuclei

Active galactic nuclei play a fundamental role in the evolution of both supermassive black holes and their host galaxies. As gas falls in to a black hole in the center of a galaxy, its angular momentum forces it into an accretion disk. As matter moves towards the SMBH, it transfers its gravitational potential energy to thermal energy. For accretion disks around supermassive black holes, this can cause the disk to emit large amounts of electromagnetic radiation (Lin & Papaloizou, 1996).

This emitted radiation is important in a number of ways. Most critical to the SMBH itself is the radiation pressure exerted on infalling matter. This radiation pressure sets an upper limit on the rate of accretion, as there is a point where the force from emitted radiation balances the force of gravity for infalling gas (Rybicki & Lightman, 1979). This limit, known as the Eddington limit, is given by

$$L_{Edd} = 4\pi G M c m_H / \sigma_T = 1.25 \times 10^{38} \text{erg s}^{-1} (M/M_\odot), \tag{IV.13}$$

where c is the speed of light, m_H is the mass of hydrogen, and σ_T is the Thompson cross section.

The radiation given off by the accretion disk affects galactic properties as well. Powerful AGN can strip away gas from the center of the galaxy, halting star formation.

This can quickly change a galaxy from a blue, gaseous, star forming galaxy into one that is red, dry, and dead.

IV.4 Conclusion

We have seen that galaxies and the supermassive black holes at their centers both have their most dramatic periods of evolution around the same time. Galaxy mergers grow both the galaxy and the SMBH. Galaxies grow and become more elliptical as mergers bring in additional mass on orbits that can disrupt their gaseous disks. These mergers also bring in counterpart supermassive black holes that fall toward the center of the galaxy and merge with the central SMBH, while also triggering accretion events and AGN feedback that pump energy back into the galaxy, shutting off star formation.

IV.4.1 Correlations

In light of these shared growth mechanisms, the correlations mentioned in Section IV.1 begin to move from a purely observational coincidence to a natural result of co-evolution. The M–σ relation is a natural byproduct of the simultaneous growth of supermassive black holes and their galaxies during merger events. The mass of the SMBH increases due to the merging of binary companions and increased levels of accretion, while the host mass, and thus velocity dispersion, increases due to the infalling galaxy itself. Likewise, the overabundance of AGN in galaxies lying in the green valley is the consequence of simultaneous change. Mergers both trigger highly luminous AGN feedback and cause an inexorable shift from the blue cloud, through the green valley, to the red sequence. Even the increase in scatter of the M–σ relation at low masses can be explained by the galaxies having lower mass, and therefore being more likely to allow a gravitational wave recoil kicked black hole of a given velocity to escape.

IV.4.2 Open Questions

In the end, there remain a number of open questions. How can very large supermassive black holes form so early? What is dark matter actually made of? How do galaxies retain their black holes if merger recoils can kick them with velocities greater than the escape velocity of the galaxy? Over what range are our correlations truly valid? These are just some of the questions that are currently being investigated, and promise to provide a rich field of study for years to come.

CHAPTER V

Conclusion

BIBLIOGRAPHY

Abel, T., Bryan, G. L., & Norman, M. L. 2000, Astrophys. J., 540, 39

—. 2002, Science, 295, 93

Alvarez, M. A., Bromm, V., & Shapiro, P. R. 2006, Astrophys. J., 639, 621

Alvarez, M. A., Wise, J. H., & Abel, T. 2009, Astrophys. J., Lett., 701, L133

Bagla, J. S. 2002, Journal of Astrophysics and Astronomy, 23, 185

Bagla, J. S., & Ray, S. 2003, New Astronomy, 8, 665

Barkana, R., & Loeb, A. 2001, Phys. Rep., 349, 125

Barnes, J., & Hut, P. 1986, Nature, 324, 446

Behroozi, P. S., Wechsler, R. H., & Wu, H.-Y. 2013, Astrophys. J., 762, 109

Binney, J., & Merrifield, M. 1998, Galactic Astronomy, Princeton Series in Astrophysics (Princeton University Press)

Binney, J., & Tremaine, S. 1988, Galactic Dynamics, Princeton Series in Astrophysics (Princeton University Press)

Blandford, R. D., & McKee, C. F. 1982, Astrophys. J., 255, 419

Bode, P., Ostriker, J. P., & Xu, G. 2000, Astrophys. J., Suppl. Ser., 128, 561

Bondi, H. 1952, Mon. Not. R. Astron. Soc., 112, 195

Bondi, H., & Hoyle, F. 1944, Mon. Not. R. Astron. Soc., 104, 273

Bouchet, F. R., Colombi, S., Hivon, E., & Juszkiewicz, R. 1995, Astron. Astrophys., 296, 575

Bouwens, R. J., Illingworth, G. D., Oesch, P. A., et al. 2012, Astrophys. J., Lett., 752, L5

Brügmann, B., Tichy, W., & Jansen, N. 2004, Phys. Rev. Lett., 92, 211101

Bryan, G. L., & Norman, M. L. 1998, Astrophys. J., 495, 80

Buchert, T. 1994, Mon. Not. R. Astron. Soc., 267, 811

Buchert, T., Melott, A. L., & Weiss, A. G. 1994, Astron. Astrophys., 288, 349

Bullock, J. S., Dekel, A., Kolatt, T. S., et al. 2001a, Astrophys. J., 555, 240

Bullock, J. S., Kolatt, T. S., Sigad, Y., et al. 2001b, Mon. Not. R. Astron. Soc., 321, 559

Bykov, A. M., Paerels, F. B. S., & Petrosian, V. 2008, Space Sci. Rev., 134, 141

Chandrasekhar, S. 1943, Astrophys. J., 97, 255

Couchman, H. M. P., & Rees, M. J. 1986, Mon. Not. R. Astron. Soc., 221, 53

Crocce, M., Pueblas, S., & Scoccimarro, R. 2006, Mon. Not. R. Astron. Soc., 373, 369

Davis, M., Efstathiou, G., Frenk, C. S., & White, S. D. M. 1985, Astrophys. J., 292, 371

Djorgovski, S., & Davis, M. 1987, Astrophys. J., 313, 59

Efstathiou, G., Davis, M., White, S. D. M., & Frenk, C. S. 1985, Astrophys. J., Suppl. Ser., 57, 241

Einasto, J., Klypin, A. A., Saar, E., & Shandarin, S. F. 1984, Mon. Not. R. Astron. Soc., 206, 529

Eke, V. R., Navarro, J. F., & Steinmetz, M. 2001, Astrophys. J., 554, 114

Fan, X., Narayanan, V. K., Lupton, R. H., et al. 2001, Astron. J., 122, 2833

Gao, L., Navarro, J. F., Cole, S., et al. 2008, Mon. Not. R. Astron. Soc., 387, 536

Gingold, R. A., & Monaghan, J. J. 1977, Mon. Not. R. Astron. Soc., 181, 375

Gnedin, N. Y., & Ostriker, J. P. 1997, Astrophys. J., 486, 581

González, J. A., Sperhake, U., Brügmann, B., Hannam, M., & Husa, S. 2007, Phys. Rev. Lett., 98, 091101

Gottloeber, S. 1998, in Large Scale Structure: Tracks and Traces, ed. V. Mueller, S. Gottloeber, J. P. Muecket, & J. Wambsganss, 43–46

Gültekin, K., Richstone, D. O., Gebhardt, K., et al. 2009, The Astrophysical Journal, 698, 198

Heger, A., Fryer, C. L., Woosley, S. E., Langer, N., & Hartmann, D. H. 2003, Astrophys. J., 591, 288

Heger, A., & Woosley, S. E. 2002, Astrophys. J., 567, 532

Heitmann, K., Lukić, Z., Habib, S., & Ricker, P. M. 2006, Astrophys. J., Lett., 642, L85

Heitmann, K., White, M., Wagner, C., Habib, S., & Higdon, D. 2010, Astrophys. J., 715, 104

Hernquist, L., Hut, P., & Makino, J. 1993, Astrophys. J., Lett., 402, L85

Hernquist, L., & Katz, N. 1989, Astrophys. J., Suppl. Ser., 70, 419

Herrmann, F., Hinder, I., Shoemaker, D., Laguna, P., & Matzner, R. A. 2007, Astrophys. J., 661, 430

Herrnstein, J. R., Moran, J. M., Greenhill, L. J., et al. 1999, Nature, 400, 539

Holley-Bockelmann, K., Gültekin, K., Shoemaker, D., & Yunes, N. 2008, Astrophys. J., 686, 829

Holley-Bockelmann, K., Wise, J. H., & Sinha, M. 2012, Astrophys. J., Lett., 761, L8

Hoyle, F., & Lyttleton, R. A. 1939, Proceedings of the Cambridge Philosophical Society, 35, 405

Hubble, E. P. 1926, Astrophys. J., 64, 321

Islam, R. R., Taylor, J. E., & Silk, J. 2003, Mon. Not. R. Astron. Soc., 340, 647

Jenkins, A. 2010, Mon. Not. R. Astron. Soc., 403, 1859

Jeon, M., Pawlik, A. H., Greif, T. H., et al. 2012, Astrophys. J., 754, 34

Khan, F. M., Just, A., & Merritt, D. 2011, Astrophys. J., 732, 89

Klypin, A., Kravtsov, A. V., Bullock, J. S., & Primack, J. R. 2001, Astrophys. J., 554, 903

Klypin, A. A., & Shandarin, S. F. 1983, Mon. Not. R. Astron. Soc., 204, 891

Klypin, A. A., Trujillo-Gomez, S., & Primack, J. 2011, Astrophys. J., 740, 102

Knebe, A., Knollmann, S. R., Muldrew, S. I., et al. 2011, Mon. Not. R. Astron. Soc., 415, 2293

Komatsu, E., Dunkley, J., Nolta, M. R., et al. 2009, Astrophys. J., Suppl. Ser., 180, 330

Koppitz, M., Pollney, D., Reisswig, C., et al. 2007, Phys. Rev. Lett., 99, 041102

Kormendy, J., & Djorgovski, S. 1989, Ann. Rev. Astron. Astrophys., 27, 235

Kuhlen, M., & Faucher-Giguère, C.-A. 2012, Mon. Not. R. Astron. Soc., 423, 862

Lin, D., & Papaloizou, J. 1996, Annual Review of Astronomy and Astrophysics, 34, 703

Lo, K. Y. 2005, Ann. Rev. Astron. Astrophys., 43, 625

Lukić, Z., Heitmann, K., Habib, S., Bashinsky, S., & Ricker, P. M. 2007, Astrophys. J., 671, 1160

Macciò, A. V., Dutton, A. A., & van den Bosch, F. C. 2008, Mon. Not. R. Astron. Soc., 391, 1940

Madau, P., Haardt, F., & Rees, M. J. 1999, Astrophys. J., 514, 648

Madau, P., & Rees, M. J. 2001, Astrophys. J., Lett., 551, L27

Marquardt, D. W. 1963, Journal of the Society for Industrial and Applied Mathematics, 11(2), 431

Monaghan, J. J., & Lattanzio, J. C. 1985, Astron. Astrophys., 149, 135

More, S., Kravtsov, A. V., Dalal, N., & Gottlöber, S. 2011, Astrophys. J., Suppl. Ser., 195, 4

Muñoz-Cuartas, J. C., Macciò, A. V., Gottlöber, S., & Dutton, A. A. 2011, Mon. Not. R. Astron. Soc., 411, 584

Nadarajah, S. 2005, Journal of Applied Statistics, 32, 685

Navarro, J. F., Frenk, C. S., & White, S. D. M. 1996, Astrophys. J., 462, 563

—. 1997a, Astrophys. J., 490, 493

—. 1997b, Astrophys. J., 490, 493

Neto, A. F., Gao, L., Bett, P., et al. 2007, Mon. Not. R. Astron. Soc., 381, 1450

Peebles, P. J. E. 1969, Astrophys. J., 155, 393

Prada, F., Klypin, A. A., Cuesta, A. J., Betancort-Rijo, J. E., & Primack, J. 2012, Mon. Not. R. Astron. Soc., 423, 3018

Press, W. H., & Schechter, P. 1974, Astrophys. J., 187, 425

Quinn, T., Katz, N., Stadel, J., & Lake, G. 1997, ArXiv Astrophysics e-prints, astro-ph/9710043

Reed, D. S., Bower, R., Frenk, C. S., Jenkins, A., & Theuns, T. 2007, Mon. Not. R. Astron. Soc., 374, 2

Rubin, V. C., Ford, W. K. J., & . Thonnard, N. 1980, Astrophys. J., 238, 471

Rybicki, G. B., & Lightman, A. P. 1979, Radiative processes in astrophysics / George B. Rybicki, Alan P. Lightman (Wiley, New York :), xv, 382 p. :

Schawinski, K., Urry, C. M., Virani, S., et al. 2010, The Astrophysical Journal, 711, 284

Scheel, M. A., Boyle, M., Chu, T., et al. 2009, Phys. Rev. D, 79, 024003

Scoccimarro, R. 1998, Mon. Not. R. Astron. Soc., 299, 1097

Shapiro, P. R., & Giroux, M. L. 1987, Astrophys. J., Lett., 321, L107

Shima, E., Matsuda, T., Takeda, H., & Sawada, K. 1985, Mon. Not. R. Astron. Soc., 217, 367

Sinha, M., & Holley-Bockelmann, K. 2009, Mon. Not. R. Astron. Soc., 397, 190

—. 2010, Mon. Not. R. Astron. Soc., 405, L31

Sirko, E. 2005, Astrophys. J., 634, 728

Springel, V. 2005, Mon. Not. R. Astron. Soc., 364, 1105

Springel, V., Yoshida, N., & White, S. D. M. 2001, New Astron, 6, 79

Tanaka, T., Perna, R., & Haiman, Z. 2012, Mon. Not. R. Astron. Soc., 425, 2974

Tegmark, M., Silk, J., Rees, M. J., et al. 1997, Astrophys. J., 474, 1

Venkatesan, A., Tumlinson, J., & Shull, J. M. 2003, Astrophys. J., 584, 621

Voit, G. M. 2005, Reviews of Modern Physics, 77, 207

Wechsler, R. H., Bullock, J. S., Primack, J. R., Kravtsov, A. V., & Dekel, A. 2002, Astrophys. J., 568, 52

White, M. 2001, Astron. Astrophys., 367, 27

Xu, G. 1995, Astrophys. J., Suppl. Ser., 98, 355

Zel'dovich, Y. B. 1970, Astron. Astrophys., 5, 84

Zemp, M., Gnedin, O. Y., Gnedin, N. Y., & Kravtsov, A. V. 2011, Astrophys. J., Suppl. Ser., 197, 30

Zhao, D. H., Jing, Y. P., Mo, H. J., & Börner, G. 2009, Astrophys. J., 707, 354

Zhao, D. H., Mo, H. J., Jing, Y. P., & Börner, G. 2003, Mon. Not. R. Astron. Soc., 339, 12

Appendices

Appendix A

Rockstar Configuration and Execution

A.1 Single Node Configuration File (Text)

```
#Rockstar Halo Finder
#Parallel config file for multi-cpu, multi-snapshot halo finding
#Note that periodic boundary conditions are assumed for NUM_WRITERS > 1.
#See README for details.

#Once compiled ("make"), run Rockstar server as
# ./rockstar -c parallel.cfg
#Then launch the reading/analysis tasks with:
# ./rockstar -c auto-rockstar.cfg
#You will have to launch at least NUM_BLOCKS+NUM_WRITERS processes.

FILE_FORMAT = "GADGET2" # or "ART" or "ASCII"
PARTICLE_MASS = 0       # must specify (in Msun/h) for ART or ASCII

# You should specify cosmology parameters only for ASCII formats
# For GADGET2 and ART, these parameters will be replaced with values from the
# particle data file
SCALE_NOW = 1
h0 = 0.7
Ol = 0.73
Om = 0.27

# For GADGET2, you may need to specify conversion parameters.
# Rockstar's internal units are Mpc/h (lengths) and Msun/h (masses)
GADGET_LENGTH_CONVERSION = 1e-3
GADGET_MASS_CONVERSION = 1e+10

# This specifies the use of multiple processors:
PARALLEL_IO = 1

# Output full particle information as well as halos for N number of procs
FULL_PARTICLE_CHUNKS = 0

# This should be less than 1/5 of BOXSIZE
OVERLAP_LENGTH = 1.5

# This specifies how many CPUs you want to analyze the particles:
NUM_WRITERS = 8

# Calculate radii and other halo properties using unbound (0) or only bound (1) particles (default 1)
BOUND_PROPS = 0

# This sets the virial radius/mass definition ("vir", "XXXc", or "XXXb")
MASS_DEFINITION = "vir"

# This specifies the I/O filenames:
OUTBASE = "halos"
INBASE = "particles"
NUM_SNAPS = 1
NUM_BLOCKS = 1
#BGC2_SNAPNAMES = "snapnames.lst"
#FILENAME = "particles_<snap>.<block>.dat"
```

A.2 PBS Submission Script (Bash)

```
#!/bin/sh
#PBS -M djsissom@gmail.com
#PBS -m bae
#PBS -l nodes=1:ppn=10
#PBS -l pmem=3000mb
#PBS -l mem=30000mb
#PBS -l walltime=0:30:00
#PBS -o out.log
#PBS -j oe

# Change to working directory
echo $PBS_NODEFILE
cd $PBS_O_WORKDIR

# Start the server
rockstar -c onenode.cfg &> server.out &

# Wait for auto-rockstar.cfg to be created
perl -e 'sleep 1 while (!(-e "halos/auto-rockstar.cfg"))'
mv halos/auto-rockstar.cfg .

# Execute the reader processes
mpiexec -verbose -n 1 rockstar -c auto-rockstar.cfg >> clients.out 2>&1 &
sleep 20
```

```
25
26 # Execute the analysis processes
27 mpiexec -verbose -n 8 rockstar -c auto-rockstar.cfg >> clients.out 2>&1
28
29 # - end of script
```

A.3 Post-Processing Script (Bash)

```
1  #!/bin/bash
2
3  echo 'running finish_bgc2...'
4  ~/projects/programs/nbody/rockstar/Rockstar-0.99.9/util/finish_bgc2 -c onenode.cfg -s 0
5
6  echo 'running bgc2_to_ascii...'
7  ~/projects/programs/nbody/rockstar/Rockstar-0.99.9/util/bgc2_to_ascii -c onenode.cfg -s 0 > halos/all_halos.bgc2.
       ascii
8
9  echo 'running find_parents...'
10 ~/projects/programs/nbody/rockstar/Rockstar-0.99.9/util/find_parents halos/out_0.list 10.0 > halos/out_0.list.
       parents
11
12 echo 'finished'
```

Appendix B

CrossMatch Modifications and Configuration

B.1 2lpt First Configuration File (Text)

```
MIN_SNAPSHOT_NUM  0
MAX_SNAPSHOT_NUM  0

MAX_RANK_LOC      0

OUTBASE           crossmatch_2lpt_first
```

B.2 za First Configuration File (Text)

```
MIN_SNAPSHOT_NUM  0
MAX_SNAPSHOT_NUM  0

MAX_RANK_LOC      0

OUTBASE           crossmatch_za_first
```

Appendix C

BGC2 Import Code (Python)

```
#!/usr/bin/env python

import sys
import struct

def read_bgc2(filename):
  offset = 4
  groupoffset = 8
  particleoffset = 8

  headersize = 1024
  groupsize = 4*8 + 10*4
  particlesize = 1*8 + 6*4

  headerformat = '=Q 16q 19d'
  groupformat = '=2q 2Q 10f'
  particleformat = '=q 6f'

  print "Reading "+filename+"..."
  fd = open(filename, 'rb')
  bin_string = fd.read()
  fd.close()
  print "Finished reading file."
  bin_string = bin_string[offset:]

  # Header stuff
  header_bin = bin_string[:headersize]
  header_pad = headersize - 36*8
  header = list(struct.unpack(headerformat, header_bin[:-header_pad]))

  # Group stuff
  ngroups = header[8]
  print 'ngroups = ', ngroups
  groupstart = headersize + groupoffset
  groupend = groupstart + ngroups*groupsize
  group_bin = bin_string[groupstart:groupend]
  group = []
  for i in range(ngroups):
    group.append(list(struct.unpack(groupformat, group_bin[i*groupsize:(i+1)*groupsize])))

  # Particle stuff
  particlestart = headersize + groupoffset + ngroups*groupsize + particleoffset
  particle_bin = bin_string[particlestart:]
  particle = []
  p_start = 0
  for i in range(ngroups):
    npart = group[i][2]
    particle.append([])
    for j in range(npart):
      particle[i].append(list(struct.unpack(particleformat, particle_bin[p_start:p_start+particlesize])))
      p_start += particlesize
    p_start += particleoffset

  print "Finished parsing bgc2 file"
  return header, group, particle


def main():
  header, group, particle = read_bgc2(sys.argv[1])

  print 'Header contents:'
  for value in header:
    print value
  print

  print 'Group[0] contents:'
  for value in group[0]:
    print value
  print

  print 'Particles in group[0]:'
  for part in particle[0]:
    print part
  print

  print 'Group[1] contents:'
  for value in group[1]:
    print value
  print

  print 'Particles in group[1]:'
  for part in particle[1]:
```

```
83      print part
84
85
86
87
88
89  if __name__ == '__main__':
90    main()
```

Appendix D

Density Profile Code (Python)

```
#!/usr/bin/env python

import sys
import bgc2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator
from scipy.optimize import curve_fit
from scipy.stats import chisquare

#read_mode = 'ascii2'
read_mode = 'bgc2'

if read_mode == 'bgc2':
  use_bgc2 = True
  use_all = False
  individual_masses = False
  halo_id = 146289
  nbins = 50
  nfit = 500
  ooms = 3.0
  mass_scale = 1.0
  common_mass = 5.33423e5
  dist_scale = 1.0e3
  #res_limit = 0.488
  #res_limit = 4.0
  res_limit = 0.5
  #res_limit = 10.0
  draw_frac = 0.1
  tick_base_major = 100.0
  tick_base_minor = 10.0
  find_com = False
elif read_mode == 'ascii':
  use_bgc2 = False
  use_all = True
  individual_masses = True
  halo_id = 0
  nbins = 100
  nfit = 500
  ooms = 5.0
  mass_scale = 1.0e12
  dist_scale = 200.0
  res_limit = 1.0e-2
  draw_frac = 2.0e-4
  tick_base_major = 80.0
  tick_base_minor = 20.0
  find_com = True
elif read_mode == 'ascii2':
  use_bgc2 = False
  use_all = True
  individual_masses = True
  halo_id = 0
  nbins = 100
  nfit = 500
  ooms = 3.5
  mass_scale = 1.0e10
  dist_scale = 1.0
  #res_limit = 3.0e-1
  res_limit = 1.0
  draw_frac = 1.0e-2
  tick_base_major = 200.0
  tick_base_minor = 40.0
  find_com = True
else:
  sys.exit(98712)

#outfile = 'asciitest_halo_properties.txt'
outfile = 'density_profile_halos.dat'
comfile = 'center_of_mass.txt'

make_plot = False
#make_plot = True
draw_density = True
#plot_base = 'asciitest_density_profile.fig.'
plot_base = 'figure_'
plot_ext = '.eps'
dist_units = 'kpc'
xlabel_proj = [r'X Position (%s h$^{-1}$)' % (dist_units), r'X Position (%s h$^{-1}$)' % (dist_units), r'Y
    Position (%s h$^{-1}$)' % (dist_units)]
ylabel_proj = [r'Y Position (%s h$^{-1}$)' % (dist_units), r'Z Position (%s h$^{-1}$)' % (dist_units), r'Z
    Position (%s h$^{-1}$)' % (dist_units)]
xlabel_prof = r'Radius (%s h$^{-1}$)' % (dist_units)
```

```
ylabel_prof = r'Density␣(M$_{\odot}$␣%s$^{-3}$␣h$^{2}$)' % (dist_units)
npixels = 50

#common_mass = 1.0e-7
#common_mass = 1.0e5
mass_col = 0
pos_cols = (1,2,3)
vel_cols = (4,5,6)
halo_id_col = 0

grav_const = 4.3e-6 # kpc M_sol^-1 (km/s)^2


def read_files(files):
  data = 0
  for file in files:
    print 'Reading␣file␣%s...' % (file)
    if data == 0:
      data = np.genfromtxt(file, comments='#')
    else:
      data = np.append(data, np.genfromtxt(file, comments='#'), axis=0)
  print 'Finished␣reading␣files.'
  return data


def my_chisq(ydata,ymod,deg=2,sd=None):
  """
Returns␣the␣reduced␣chi-square␣error␣statistic␣for␣an␣arbitrary␣model,
chisq/nu,␣where␣nu␣is␣the␣number␣of␣degrees␣of␣freedom.␣If␣individual
standard␣deviations␣(array␣sd)␣are␣supplied,␣then␣the␣chi-square␣error
statistic␣is␣computed␣as␣the␣sum␣of␣squared␣errors␣divided␣by␣the␣standard
deviations.␣See␣http://en.wikipedia.org/wiki/Goodness_of_fit␣for␣reference.

ydata,ymod,sd␣assumed␣to␣be␣Numpy␣arrays.␣deg␣integer.

Usage:
>>>␣chisq=redchisqg(ydata,ymod,n,sd)
where
ydata␣:␣data
ymod␣:␣model␣evaluated␣at␣the␣same␣x␣points␣as␣ydata
n␣:␣number␣of␣free␣parameters␣in␣the␣model
sd␣:␣uncertainties␣in␣ydata

Rodrigo␣Nemmen
http://goo.gl/8S1Oo
␣␣␣"""
  # Chi-square statistic
  if sd==None:
    chisq=np.sum((ydata-ymod)**2)
  else:
    chisq=np.sum( ((ydata-ymod)/sd)**2 )

  # Number of degrees of freedom assuming 2 free parameters
  nu=ydata.size-1-deg
  return chisq/nu


def calc_m_enclosed(mass, pos):
  r = np.sqrt(pos[:,0]**2 + pos[:,1]**2 + pos[:,2]**2)
  r = np.sort(r)
  first_good_bin = 0
  for i in range(len(r)):
    if r[i] > res_limit:
      first_good_bin = i
      break
  print 'r1␣=', r[first_good_bin-1]
  print 'r2␣=', r[first_good_bin]
  print 'r3␣=', r[first_good_bin+1]
  m_extra = mass[0] * first_good_bin
  r = r[first_good_bin:]
  #m_enclosed = np.zeros(len(r))
  #for i in range(len(r)):
  #  m_enclosed[i] = mass[0] * (i + 1.0)
  m_enclosed = (np.arange(len(r)) + 1.0) * mass[0] + m_extra
  return r, m_enclosed


def calc_density_profile(mass, pos):
  r = np.sqrt(pos[:,0]**2 + pos[:,1]**2 + pos[:,2]**2)
  max_r = r.max()
  #min_r = max_r / 10**ooms
  min_r = res_limit
  log_range = np.log10(max_r) - np.log10(min_r)

  #global nbins
  local_nbins = float(nbins + 1)
  #nbins = len(r) / 1000
  while True:
    bins = np.arange(local_nbins)
    bins = max_r * 10.0**(log_range * bins / (local_nbins-1.0) - log_range)
    bin_mass, r_bins = np.histogram(r, bins, weights=mass)
    if (bin_mass == 0.0).any():
```

```
      local_nbins -= 1
      continue
    else:
      break

  #print 'Binning particles using bin edges of \n', r_bins

  rho = bin_mass / (sphere_vol(r_bins[1:]) - sphere_vol(r_bins[:-1]))

  N_bin, blah = np.histogram(r, bins)
  rho_err = poisson_error(N_bin) * rho

  return r_bins, rho, rho_err


def logbin(pos):
  r = np.sqrt(pos[:,0]**2 + pos[:,1]**2 + pos[:,2]**2)
  max_r = r.max()
  min_r = max_r / 10**ooms
  log_range = np.log10(max_r) - np.log10(min_r)

  global nbins
  nbins = float(nbins + 1)
  bins = np.arange(nbins)
  bins = max_r * 10.0**(log_range * bins / (nbins-1.0) - log_range)

  hist, bin_edges = np.histogram(r, bins)
  #print 'Binning particles using bin edges of \n', bin_edges
  return hist, bin_edges


def poisson_error(N):
  err = np.sqrt(N) / N
  return err


def sphere_vol(r):
  volume = (4.0 / 3.0) * np.pi * r**3
  return volume


def get_rho_0(R_s, R_vir):
  H = 70.0e-3 # km s^-1 kpc^-1
  G = 4.3e-6 # kpc M_sol^-1 (km/s)^2
  rho_crit = 3.0 * H**2 / (8.0 * np.pi * G)

  v = 178
  c = R_vir / R_s
  g = 1.0 / (np.log(1.0+c) - c/(1.0+c))
  delta_char = v * c**3 * g / 3.0

  return rho_crit * delta_char


def nfw_fit_rho0(r, R_s, rho_0):
  if R_s >= 1.0:
    return (R_s - 1.0) * np.exp(r) + rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**2)
  return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**2)


def nfw_fit_rho0_log(r, R_s, rho_0):
  r = 10.0**r
  R_s = 10.0**R_s
  rho_0 = 10.0**rho_0
  profile =  rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**2)
  return np.log10(profile)


def nfw_def_rho0(R_vir):
  def _nfw_def_rho0(r, R_s):
    rho_0 = get_rho_0(R_s, R_vir)
    return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**2)
  return _nfw_def_rho0


def nfw_databin_rho0(rho_0):
  def _nfw_databin_rho0(r, R_s):
    return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**2)
  return _nfw_databin_rho0


def dm_profile_fit_rho0_log(r, R_s, rho_0, alpha):
  r = 10.0**r
  R_s = 10.0**R_s
  rho_0 = 10.0**rho_0
  alpha = 10.0**alpha
  profile = rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**alpha)
  return np.log10(profile)


def dm_profile_fit_rho0(r, R_s, rho_0, alpha):
  return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**alpha)
```

```
265
266
267 def dm_profile_def_rho0(R_vir):
268   def _dm_profile_def_rho0(r, R_s, alpha):
269     rho_0 = get_rho_0(R_s, R_vir)
270     return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**alpha)
271   return _dm_profile_def_rho0
272
273
274 def dm_profile_databin_rho0(rho_0):
275   def _dm_profile_databin_rho0(r, R_s, alpha):
276     return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**alpha)
277   return _dm_profile_databin_rho0
278
279
280 def nfw_cdf(r, R_s, rho_0):
281   r = 10.0**r
282   R_s = 10.0**R_s
283   rho_0 = 10.0**rho_0
284   profile = rho_0 * R_s * (np.log(1.0 + r / R_s) - 1.0 / (1.0 + r / R_s))
285   return np.log10(profile)
286
287
288 def nfw_cdf_nolog(r, R_s, rho_0):
289   profile = rho_0 * R_s * (np.log(1.0 + r / R_s) - 1.0 / (1.0 + r / R_s))
290   return profile
291
292
293 def mass_profile(s, c):
294   g = 1.0 / (np.log(1.0 + c) - c / (1.0 + c))
295   return g * (np.log(1.0 + c * s) - c * s / (1.0 + c * s))
296
297
298 def fit_mass_profile(s, m_enclosed, err=None, R_vir=None):
299   #for i in range(len(s)):
300   #  if s[i] > res_limit:
301   #    first_good_bin = i
302   #    break
303   first_good_bin = 0
304
305   #popt, pcov = curve_fit(nfw_cdf, np.log10(r), np.log10(m_outside), sigma=np.log10(err))
306 #  popt, pcov = curve_fit(nfw_cdf, np.log10(r), np.log10(m_outside))
307 #  popt = 10.0**popt
308 #  pcov = 10.0**pcov
309   popt, pcov = curve_fit(mass_profile, s, m_enclosed)
310
311   print 'fit_params␣=', popt
312   print 'covariance␣=', pcov
313   nfw_r = np.linspace(s[0], s[-1], nfit)
314   nfw_fit = mass_profile(nfw_r, popt[0])
315   chi2_fit = mass_profile(s, popt[0])
316
317   chi2 = chisquare(np.log10(m_enclosed[first_good_bin:]), np.log10(chi2_fit[first_good_bin:]))
318   chi2_nolog = chisquare(m_enclosed[first_good_bin:], chi2_fit[first_good_bin:])
319   print 'chi_square␣=', chi2
320   print 'chi_square_nolog␣=', chi2_nolog
321   return nfw_r, nfw_fit, popt, pcov, chi2[0]
322
323
324 def fit_profile(r, rho, err=None, R_vir=None):
325   first_good_bin = 0
326 #  for i in range(len(r)):
327 #    if r[i] > res_limit:
328 #      rho_0_databin = rho[i]
329 #      first_good_bin = i
330 #      break
331 #  print 'first_good_bin =', first_good_bin
332
333   #--------- choose one fitting type ---------#
334   #popt, pcov = curve_fit(nfw_fit_rho0, r, rho, sigma=err)
335   #popt, pcov = curve_fit(nfw_def_rho0(R_vir), r, rho, p0=[10.0], sigma=err)
336   #popt, pcov = curve_fit(nfw_databin_rho0(rho_0_databin), r, rho, sigma=err)
337   blah = 3
338   if blah == 0:
339     for i in range(100):
340       a = 2.0 * np.random.random() * 0.1 * r.max()
341       b = 2.0 * np.random.random() * 10.0
342       c = 2.0 * np.random.random() * 2.0
343       try:
344         popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, p0=[a,b,c], sigma=err)
345       except RuntimeError:
346         continue
347       if (popt[0] < r.max()) and (popt[2] >= 0.0):
348         break
349       elif i >= 99:
350         print 'no␣good␣fit␣found␣for␣this␣halo...'
351 #        return None, None, None, None, None
352   elif blah == 1:
353     #a = r.max() / 100.0
354     a = 0.001
355     b = rho[first_good_bin]
356     c = 0.001
```

```
357       #popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, sigma=err)
358       print '-------------------------------------'
359       print 'rho_0 before =', b
360       #try:
361         #popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, p0=[a,b,c], sigma=err, maxfev=1, xtol=100.0)
362       popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, p0=[a,b,c], sigma=err, xtol=1.0e-1)
363       #except RuntimeError:
364       #  print 'just checking for now...'
365       print 'rho_0 after =', popt[1]
366       #sys.exit()
367     elif blah == 2:
368       #popt, pcov = curve_fit(dm_profile_fit_rho0_log, np.log10(r), np.log10(rho), sigma=np.log10(err))
369       popt, pcov = curve_fit(nfw_fit_rho0_log, np.log10(r), np.log10(rho), sigma=np.log10(err))
370       popt = 10.0**popt
371       pcov = 10.0**pcov
372     elif blah == 3:
373       popt, pcov = curve_fit(nfw_fit_rho0, r, rho, sigma=err, p0 = [0.1, 1.0])
374
375     #popt, pcov = curve_fit(dm_profile_def_rho0(R_vir), r, rho, sigma=err)
376     #popt, pcov = curve_fit(dm_profile_databin_rho0(rho_0_databin), r, rho, sigma=err)
377     #-----------------------------------------#
378
379     print 'fit_params =', popt
380     print 'covariance =', pcov
381
382     nfw_r = np.linspace(r[0], r[-1], nfit)
383     #--------- choose one fitting type ---------#
384     nfw_fit = nfw_fit_rho0(nfw_r, popt[0], popt[1])
385     #nfw_fit = nfw_def_rho0(R_vir)(nfw_r, popt[0])
386     #nfw_fit = nfw_databin_rho0(rho_0_databin)(nfw_r, popt[0])
387     #nfw_fit = dm_profile_fit_rho0(nfw_r, popt[0], popt[1], popt[2])
388     #nfw_fit = dm_profile_def_rho0(R_vir)(nfw_r, popt[0], popt[1])
389     #nfw_fit = dm_profile_databin_rho0(rho_0_databin)(nfw_r, popt[0], popt[1])
390     #-----------------------------------------#
391     #--------- choose one fitting type ---------#
392     chi2_fit = nfw_fit_rho0(r, popt[0], popt[1])
393     #chi2_fit = nfw_def_rho0(R_vir)(r, popt[0])
394     #chi2_fit = nfw_databin_rho0(rho_0_databin)(r, popt[0])
395     #chi2_fit = dm_profile_fit_rho0(r, popt[0], popt[1], popt[2])
396     #chi2_fit = dm_profile_def_rho0(R_vir)(r, popt[0], popt[1])
397     #chi2_fit = dm_profile_databin_rho0(rho_0_databin)(r, popt[0], popt[1])
398     #-----------------------------------------#
399
400     #chi2 = my_chisq(rho, chi2_fit, 2, err)
401     chi2 = chisquare(rho, chi2_fit)
402     print 'chi_square =', chi2
403     chi2 = chi2[0]
404
405     return nfw_r, nfw_fit, popt, pcov, chi2
406
407
408 def draw_projection(fig, place, plot_lim, x, y):
409     ax = plt.subplot(2,3,place+1, aspect='equal')
410     im = ax.plot(x, y, linestyle='', marker='.', markersize=1, markeredgecolor='blue')
411     ax.set_xlabel(xlabel_proj[place])
412     ax.set_ylabel(ylabel_proj[place])
413     ax.set_xlim(-plot_lim, plot_lim)
414     ax.set_ylim(-plot_lim, plot_lim)
415     ax.xaxis.set_major_locator(MultipleLocator(tick_base_major))
416     ax.xaxis.set_minor_locator(MultipleLocator(tick_base_minor))
417     ax.yaxis.set_major_locator(MultipleLocator(tick_base_major))
418     ax.yaxis.set_minor_locator(MultipleLocator(tick_base_minor))
419     return fig
420
421
422 def draw_density_projection(fig, place, plot_lim, x, y):
423     limits = [[-plot_lim, plot_lim], [-plot_lim, plot_lim]]
424     ax = plt.subplot(2,3,place+1, aspect='equal')
425     #ax.set_xlim(-plot_lim, plot_lim)
426     #ax.set_ylim(-plot_lim, plot_lim)
427     #im = ax.plot(x, y, linestyle='', marker='.', markersize=1, markeredgecolor='blue')
428     z, xedges, yedges = np.histogram2d(x, y, bins = npixels, range = limits)
429     #z = np.log10(z)
430     im = ax.imshow(z.T, extent=(-plot_lim, plot_lim, -plot_lim, plot_lim), interpolation='gaussian', origin='lower'
        )
431     ax.locator_params(nbins=6)
432     ax.set_xlabel(xlabel_proj[place])
433     ax.set_ylabel(ylabel_proj[place])
434 #   ax.xaxis.set_major_locator(MultipleLocator(tick_base_major))
435 #   ax.xaxis.set_minor_locator(MultipleLocator(tick_base_minor))
436 #   ax.yaxis.set_major_locator(MultipleLocator(tick_base_major))
437 #   ax.yaxis.set_minor_locator(MultipleLocator(tick_base_minor))
438     return fig
439
440
441 def draw_density_profile(fig, r, rho, err=None):
442     ax = plt.subplot(2,1,2)
443     im = ax.loglog(r, rho, linestyle='steps-mid-')
444     line1 = ax.axvline(res_limit, color='black', linestyle=':')
445     #ax.set_xlim(r_bins[0], r_bins[-1])
446     ax.set_xlim(r[0] - (r[1]-r[0]), r[-1] + (r[-1]-r[-2]))
447     ax.set_xlabel(xlabel_prof)
```

```
448   ax.set_ylabel(ylabel_prof)
449   if err != None:
450     err_bars = ax.errorbar(r, rho, yerr=err,linestyle='None')
451   return fig, ax
452 
453 
454 def draw_nfw_profile(fig, ax, r, rho, R_s=None):
455   ax.loglog(r, rho, linestyle='-', color='red')
456   if R_s != None:
457     line = ax.axvline(R_s, color='purple', linestyle='-.')
458   return fig
459 
460 
461 def calc_kinetic_energy(mass, vel):
462   vsq = vel[:,0]**2 + vel[:,1]**2 + vel[:,2]**2
463   energy = 0.5 * np.sum(mass*vsq)
464   return energy
465 
466 
467 def calc_potential_energy(mass, pos):
468   local_sqrt = np.sqrt
469   partial_sum = 0.0
470   for i in range(len(mass)):
471     for j in range(len(mass)):
472       if j != i:
473         r_diff = local_sqrt((pos[i,0] - pos[j,0])**2 + (pos[i,1] - pos[j,1])**2 + (pos[i,2] - pos[j,2])**2)
474         partial_sum = partial_sum - mass[i]*mass[j]/r_diff
475   energy = partial_sum * grav_const / 2.0
476   return energy
477 
478 
479 def calc_angular_momentum(mass, pos, vel):
480   ang_mom_x = np.sum(mass * (pos[:,1] * vel[:,2] - pos[:,2] * vel[:,1]))
481   ang_mom_y = np.sum(mass * (pos[:,2] * vel[:,0] - pos[:,0] * vel[:,2]))
482   ang_mom_z = np.sum(mass * (pos[:,1] * vel[:,2] - pos[:,2] * vel[:,1]))
483   ang_mom = np.sqrt(ang_mom_x**2 + ang_mom_y**2 + ang_mom_z**2)
484   return ang_mom
485 
486 
487 def main():
488   with open(outfile, 'w') as fd:
489     #fd.write('#halo_mass  concentration  R_vir  R_s +- err  rho_0 +- err  alpha +- err  chi_square\n')
490     fd.write('#halo_id␣␣␣␣␣␣␣␣halo_mass␣␣␣␣␣␣␣␣␣␣␣x_pos␣␣␣␣␣␣␣␣␣␣␣y_pos␣␣␣␣␣␣␣␣␣␣␣z_pos␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣c␣+-␣␣␣␣␣␣␣
      ␣␣␣␣␣␣err␣␣␣␣␣␣␣␣␣␣␣R_vir␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣R_s␣+-␣␣␣␣␣␣␣␣␣␣␣␣␣err␣␣␣␣␣␣␣␣␣␣␣rho_0␣+-␣␣␣␣␣␣␣␣␣␣␣␣␣err␣␣␣␣␣␣␣chi_square
      ␣␣␣␣␣␣nbins␣␣␣␣␣N_part\n')
491 #  with open(comfile, 'w') as fd:
492 #    fd.write('#id mass dx dy dz\n')
493 
494 #  if use_bgc2 == True:
495 #    header, halos, particles = bgc2.read_bgc2(sys.argv[1])
496 #    for i in range(len(halos)):
497 #      if halos[i][halo_id_col] == halo_id:
498 #        index = i
499 #    halo_particles = np.asarray(particles[index])
500 #    pos = halo_particles[:,pos_cols[0]:pos_cols[0]+3] * dist_scale
501 #    r_vir = halos[index][4] * dist_scale
502 #  else:
503 #    # Read in particle files
504 #    data = read_files(sys.argv[1:])
505 #    # Select particles with a given halo ID and convert positions from Mpc to kpc
506 #    if use_all == False:
507 #      halo_particles = data[np.where(data[:,halo_id_col] == halo_id)]
508 #    if use_all == True:
509 #      halo_particles = data
510 #    del data
511 #    pos = halo_particles[:,pos_cols[0]:pos_cols[0]+3] * dist_scale
512 #    r_vir = 241.48
513 #    #r_vir = pos.max()
514 
515   for input_file in sys.argv[1:]:
516     if use_bgc2 == True:
517       #header, halos, particles = bgc2.read_bgc2(sys.argv[1])
518       header, halos, particles = bgc2.read_bgc2(input_file)
519       halos = np.asarray(halos)
520       indices = np.argsort(halos[:,2])      # sort by number of particles
521       indices = indices[::-1]               # start with the biggest
522     else:
523       data = read_files([input_file])
524       # Select particles with a given halo ID and convert positions from Mpc to kpc
525       if use_all == False:
526         particles = [data[np.where(data[:,halo_id_col] == halo_id)]]
527       if use_all == True:
528         particles = [data]
529       del data
530 
531 
532     itteration = 0
533     #for index in range(len(halos)):
534     #for index in range(1):
535     #for index in indices[:10]:
536     for index in indices:
537       if ((len(particles[index]) >= 100) and (halos[index][1] == -1)):
```

```
538         print '-------------------------------------------------------'
539
540         halo_particles = np.asarray(particles[index])
541         pos = halo_particles[:,pos_cols[0]:pos_cols[0]+3] * dist_scale
542         vel = halo_particles[:,vel_cols[0]:vel_cols[0]+3]
543
544         if use_bgc2 == True:
545           halo_id = halos[index][0]
546           r_vir = halos[index][4] * dist_scale
547           halo_mass = halos[index][5]
548           halo_pos = np.array([halos[index][6] * dist_scale, halos[index][7] * dist_scale, halos[index][8] *
    dist_scale])
549           halo_vel = np.array([halos[index][9], halos[index][10], halos[index][11]])
550         else:
551           r_vir = 241.48
552           halo_id = 0
553           #halo_mass = mass[0] * len(halo_particles)
554           halo_pos = np.array([0.0, 0.0, 0.0])
555           halo_vel = np.array([0.0, 0.0, 0.0])
556
557
558         if individual_masses == True:
559           mass = halo_particles[:,mass_col] * mass_scale
560         else:
561           mass = np.ones(halo_particles.shape[0]) * common_mass * mass_scale
562
563         if use_bgc2 == False:
564           halo_mass = mass[0] * len(halo_particles)  #fix placement of this for ascii test
565
566         print 'Using␣%d␣particles␣in␣halo␣%d.' % (halo_particles.shape[0], halo_id)
567
568         # Find center of mass
569         if find_com == True:
570           mass_tot = mass.sum()
571           m_pos = mass.reshape(mass.shape[0],1) * pos
572           com = m_pos.sum(axis=0) / mass_tot
573           pos = pos - com
574           print 'Center␣of␣mass␣=␣(%g␣|␣%g␣|␣%g)' % (com[0], com[1], com[2])
575         else:
576           pos = pos - halo_pos
577           vel = vel - halo_vel
578
579         #with open(comfile, 'a') as fd:
580         #  fd.write("%d %g %g %g %g\n" % (halo_id, halo_mass, halo_pos[0] - com[0], halo_pos[1] - com[1],
    halo_pos[2] - com[2]))
581
582         # Bin halo particles into logarithmic shells and compute density
583         r_bins, rho, rho_err = calc_density_profile(mass, pos)
584
585         if len(r_bins) < 5:
586           print 'Too␣few␣bins.␣␣Skipping␣this␣halo.'
587           with open(outfile, 'a') as fd:
588             fd.write("%8d␣%16.12g␣␣%14.10g␣␣%14.10g␣␣%14.10g␣␣%14d␣+-␣%14d␣␣%14d␣␣%14d␣+-␣%14d␣␣%14d␣+-␣%14d␣␣%14
    d␣␣%8d␣␣%8d\n" % (halo_id, halo_mass, halo_pos[0], halo_pos[1], halo_pos[2], -9999, -9999, -9999, -9999,
    -9999, -9999, -9999, -9999, -9999, len(halo_particles)))
589           continue
590
591       #  hist, r_bins = logbin(pos)
592       #  err = poisson_error(hist)
593       #  rho = mass * hist / (sphere_vol(r_bins[1:]) - sphere_vol(r_bins[:-1]))
594       #  rho_err = err * rho
595         mid_bins = 10.0**(0.5 * (np.log10(r_bins[1:]) + np.log10(r_bins[:-1])))
596         print 'nbins␣=␣', len(mid_bins)
597
598         # Don't pass NaN's to fitting routine
599         rho_err_nonan = np.copy(rho_err)
600         nan_check = np.isnan(rho_err_nonan)
601         for i in range(len(rho_err_nonan)):
602           #if (nan_check[i] == True):
603           #  rho[i] = 1.0e-10
604           if (mid_bins[i] < res_limit) or (nan_check[i] == True):
605             rho_err_nonan[i] = 1.0e10
606
607
608 #       r, m_enclosed = calc_m_enclosed(mass, pos)
609
610         # Fit an NFW profile to the data
611 #       try:
612         nfw_r, nfw_fit, popt, pcov, chisq = fit_profile(mid_bins / r_vir, rho / rho.max(), err = rho_err_nonan /
    rho.max(), R_vir = 1.0)
613         #nfw_r, nfw_fit, popt, pcov, chisq = fit_mass_profile(r / r_vir, m_enclosed / halo_mass)
614         nfw_r = nfw_r * r_vir
615         nfw_fit = nfw_fit * rho.max()
616         #nfw_fit = nfw_fit * halo_mass
617         scale_radius = popt[0] * r_vir
618         scale_radius_err = pcov[0,0] * r_vir
619         rho_0 = popt[1] * rho.max()
620         rho_0_err = pcov[1,1] * rho.max()
621         concentration = r_vir / scale_radius
622         concentration_err = concentration * scale_radius_err / scale_radius
623
624         # Print parameters
```

```
        print 'r_vir = ', r_vir
        print "rho_0 = %g +/- %g" % (rho_0, rho_0_err)
        print "scale radius = %g +/- %g" % (scale_radius, scale_radius_err)
        print "concentration = %g +/- %g" % (concentration, concentration_err)

#put these back sometime#############################################################################
#        kin_energy = calc_kinetic_energy(mass, vel)
#        pot_energy = calc_potential_energy(mass, pos)
#        ang_mom = calc_angular_momentum(mass, pos, vel)
#
#        ttow = 2.0 * abs(kin_energy / pot_energy)
#        lambda_spin = ang_mom * np.sqrt(abs(kin_energy + pot_energy)) / (grav_const * (np.sum(mass))**2.5)
        kin_energy = 0.0
        pot_energy = 0.0
        ang_mom = 0.0

        ttow = 0.0
        lambda_spin = 0.0
#####################################################################################################


        if isinstance(pcov, float):
          print "inf covariance returned, skipping this halo..."
          with open(outfile, 'a') as fd:
            fd.write("%8d %16.12g %14.10g %14.10g %14.10g %14d + - %14d %14d %14d + - %14d %14d + - %14d %14d %8d %8d\n" % (halo_id, halo_mass, halo_pos[0], halo_pos[1], halo_pos[2], -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, -9999, len(halo_particles)))
          continue

        #Write parameters to file
        with open(outfile, 'a') as fd:
          #fd.write("%g  %g  %g  %g +- %g  %g +- %g  %g +- %g  %g\n" % (halo_mass, concentration, r_vir, scale_radius, pcov[0,0], rho_0, pcov[1,1], alpha, pcov[2,2], chisq))
          fd.write("%8d %16.12g %14.10g %14.10g %14.10g %14.10g + - %14.6g %14.10g %14.10g + - %14.6g %14.10g + - %14.6g %14.10g %8d %8d\n" % (halo_id, halo_mass, halo_pos[0], halo_pos[1], halo_pos[2], concentration, concentration_err, r_vir, scale_radius, scale_radius_err, rho_0, rho_0_err, chisq, len(r_bins), len(halo_particles)))


        ##################################################debug
        #blah_fit = nfw_fit_rho0(nfw_r, 20.0, 9.0e5)

        # Plot density profile histogram
        if (make_plot == True) and (itteration < 10):
          # Find the maximum of x, y, or z to be limit of projection plots
          plot_lim = pos.max()
          # Pick only a certain perentage of particles for projection plots
          if (draw_frac < 1.0):
            np.random.shuffle(pos)
            pos = pos[:(draw_frac*pos.shape[0])]

          fig = plt.figure()
          if draw_density == True:
            fig = draw_density_projection(fig, 0, plot_lim, pos[:,0], pos[:,1])
            fig = draw_density_projection(fig, 1, plot_lim, pos[:,0], pos[:,2])
            fig = draw_density_projection(fig, 2, plot_lim, pos[:,1], pos[:,2])
          else:
            fig = draw_projection(fig, 0, plot_lim, pos[:,0], pos[:,1])
            fig = draw_projection(fig, 1, plot_lim, pos[:,0], pos[:,2])
            fig = draw_projection(fig, 2, plot_lim, pos[:,1], pos[:,2])
          fig, ax = draw_density_profile(fig, mid_bins, rho, err=rho_err)  #put this back for binning
          #fig, ax = draw_density_profile(fig, r, m_enclosed)            #take this out for binning
          fig = draw_nfw_profile(fig, ax, nfw_r, nfw_fit, R_s=scale_radius)
          #fig = draw_nfw_profile(fig, ax, nfw_r, blah_fit, R_s=20.0)
          fig.tight_layout()
          plt.savefig(plot_base+str(itteration)+plot_ext)

          #sys.exit()

        itteration += 1


if __name__ == '__main__':
  main()
```

Appendix E

CrossMatch Best Match Code

E.1 Best Match (Python)

```
#!/usr/bin/env python

import sys
import getopt
import numpy as np


def main():
    # read in files
    print 'reading files...'
    with open(sys.argv[1]) as f:
        matches1 = f.readlines()
    with open(sys.argv[2]) as f:
        matches2 = f.readlines()
    print 'done reading files'

    header = matches1[2:6]
    header.insert(0, '# Best matches for bi-directional crossmatch\n')
    header.insert(1, '#\n')

    matches1 = matches1[7:]
    matches2 = matches2[7:]

    # convert to numpy arrays
    print 'converting to numpy arrays...'
    match_array1 = np.asarray([line.split() for line in matches1], dtype=int)
    match_array2 = np.asarray([line.split() for line in matches2], dtype=int)
    print 'done converting'

    # find matches that exist in both lists
    print 'finding matches...'
    mask = np.zeros(len(match_array1), dtype=bool)
    for i, line in enumerate(match_array1):
        id1 = line[id1_col]
        id2 = line[id2_col]
        tmp = (match_array2[:,id1_col] == id2)
        tmp = (match_array2[tmp,id2_col] == id1)
        mask[i] = tmp.any()
        if i % 1000 == 0:
            print "Finished line ", i

    print 'done matching'

    out_array = match_array1[mask]

    # write results
    print 'writing results...'
    with open(sys.argv[3], 'w') as f:
        f.writelines(("%s" % line for line in header))
        np.savetxt(f, out_array, fmt='%10d')

    print 'Finished.'


id1_col             =  4
npart1_col          =  5
id2_col             =  1
npart2_col          =  2
ncommon_col         =  6
hnum1_col           =  3
hnum2_col           =  0


if __name__ == '__main__':
  main()
```

E.2 PBS Submission Script (Bash)

```
#!/usr/bin/env bash
#PBS -M djsissom@gmail.com
#PBS -m bae
#PBS -l nodes=27:ppn=1
#PBS -l pmem=20000mb
#PBS -l mem=54000mb
#PBS -l walltime=0:30:00
#PBS -o out.log
#PBS -j oe

#nodes=186:ppn=1
```

```
#pmem=20000mb
#mem=372000mb

minsnap=53
maxsnap=61

minbox=1
maxbox=3

# Change to working directory
echo $PBS_NODEFILE
cd $PBS_O_WORKDIR

for ((i=$minbox; i<=$maxbox; i++)); do

  for ((snap=$minsnap; snap<=$maxsnap; snap++)); do

    if [ $snap -lt 10 ]; then
      j=00$snap
    elif [ $snap -lt 100 ]; then
      j=0$snap
    fi

    base_dir=~/projects/simulations/rockstar/box${i}
    crossmatch_dir=${base_dir}/crossmatch/snap${j}
    first_file=${crossmatch_dir}/crossmatch_2lpt_first_000.txt
    second_file=${crossmatch_dir}/crossmatch_za_first_000.txt
    outfile=${crossmatch_dir}/crossmatch_000.txt
    logfile=${crossmatch_dir}/best_crossmatch.log

    echo "Starting box${i} snap${j}..."

    {
      mpiexec -verbose -n 1 ./best_crossmatch.py ${first_file} ${second_file} ${outfile} > ${logfile} 2>&1
      echo "Finished box${i} snap${j}"
    } &

  done

done

wait

# - end of script
```

Appendix F

Database Generation Code

F.1 Halo Match (Python)

```
#!/usr/bin/env python

import sys
import getopt
import numpy as np


def main():
  # read and parse command line arguments
  opts, args = get_args(sys.argv[1:])
  output_file, match_file, densprof_files, parents_files, ascii_files = parse_args(opts, args)

  # read in headers as lists and data as numpy arrays
  match_header,     match_data     = read_files(match_file, header_line = 3)
  densprof_header1, densprof_data1 = read_files(densprof_files[0], header_line = 0)
  densprof_header2, densprof_data2 = read_files(densprof_files[1], header_line = 0)
  parents_header1,  parents_data1  = read_files(parents_files[0], header_line = 0)
  parents_header2,  parents_data2  = read_files(parents_files[1], header_line = 0)
  ascii_header1,    ascii_data1    = read_files(ascii_files[:(len(ascii_files)/2)], header_line = 0)
  ascii_header2,    ascii_data2    = read_files(ascii_files[(len(ascii_files)/2):], header_line = 0)
  print 'Finished reading files.'

  # filter matches, remove duplicate halo matches, and reorder match columns
  print 'Flitering match data...'
  match_data = filter_matches(match_data)
  if filter_duplicate_matches:
    match_data = filter_dups(match_data, unique_col = match_id1_col)
    match_data = filter_dups(match_data, unique_col = match_id2_col)
  if reorder_match_columns:
    match_header, match_data = reorder_match_cols(match_header, match_data)

  # calculate number of subhalos and add column to parents data and headers
  print 'Finding number of subhalos...'
  parents_header1.append('N_subs')
  parents_header2.append('N_subs')
  parents_data1 = count_subs(parents_data1)
  parents_data2 = count_subs(parents_data2)

  # create header
  print 'Making header...'
  header = make_header(match_header, densprof_header1, densprof_header2, \
                       parents_header1, parents_header2, ascii_header1, ascii_header2)

  # match halos
  print 'Matching halos...'
  halos = match_halos(match_data, [densprof_data1, densprof_data2, \
                      parents_data1, parents_data2, ascii_data1, ascii_data2])

  # filter based on given criteria and sort
  print 'Filtering halo data...'
  if filter_halo_properties:
    halos = filter_halos(halos)
  if sort_col != None:
    sort_mask = halos[:,sort_col].argsort()
    sort_mask = sort_mask[::-1]
    halos = halos[sort_mask]

  # output matched table
  print 'Writing resluts...'
  write_results(output_file, header, halos)

  print 'Finished.'


def get_args(arglist):
  try:
    opts, args = getopt.gnu_getopt(arglist, shortopts, longopts)
  except getopt.GetoptError:
    print "Invalid option(s)."
    print help_string
    sys.exit(2)
  if opts == []:
    print 'No options given.'
    print help_string
    sys.exit(2)
  return opts, args


def parse_args(opts, args):
  densproffiles = None
```

```
  parentsfiles = None
  asciifiles = None
  use_ascii = False
  for opt in opts:
    if (opt[0] == '-h') or (opt[0] == '--help') or (opts == None):
      print help_string
      sys.exit(0)
    if (opt[0] == '-o') or (opt[0] == '--outfile'):
      outfile = opt[1]
    if (opt[0] == '-m') or (opt[0] == '--match'):
      matchfile = opt[1]
    if (opt[0] == '-d') or (opt[0] == '--density'):
      densproffiles = create_append(densproffiles, opt[1])
    if (opt[0] == '-p') or (opt[0] == '--parents'):
      parentsfiles = create_append(parentsfiles, opt[1])
    if (opt[0] == '-a'):
      use_ascii = True
  if use_ascii:
    if len(args) % 2 != 0:
      print 'Must have an even number of ascii files!'
      sys.exit(3)
    for arg in args:
      asciifiles = create_append(asciifiles, arg)
  return outfile, matchfile, densproffiles, parentsfiles, asciifiles


def create_append(lst, value):
  if lst == None:
    lst = [value]
  else:
    lst.append(value)
  return lst


def read_files(files, header_line = None, comment_char = '#'):
  header = None
  data = None
  if type(files) == str:
    files = [files]

  if header_line != None:
    with open(files[0], 'r') as fd:
      for line in range(header_line):
        fd.readline()
      header = fd.readline()
    if header[0] != comment_char:
      print "Header must start with a '%s'" % comment_char
      sys.exit(4)
    header = header[1:]
    header = header.split()

  for file in files:
    print 'Reading file %s...' % (file)
    if data == None:
      data = np.genfromtxt(file, comments='#')
    else:
      data = np.append(data, np.genfromtxt(file, comments='#'), axis=0)

  if header_line == None:
    return data
  else:
    return header, data


def filter_matches(halos):
  if filter_bad_matches:
    halos = halos[halos[:,match_id1_col] != -1]
    halos = halos[halos[:,match_id2_col] != -1]
  if (min_npart != 0) and (min_npart != None):
    halos = halos[halos[:, match_npart1_col] >= min_npart]
    halos = halos[halos[:, match_npart2_col] >= min_npart]
  if (minperc_ncommon != 0) and (minperc_ncommon != None):
    halos = halos[halos[:, match_ncommon_col] / halos[:, match_npart1_col] >= minperc_ncommon]
    halos = halos[halos[:, match_ncommon_col] / halos[:, match_npart2_col] >= minperc_ncommon]
  return halos


def filter_dups(halos, unique_col = 0):
  ncommon   = halos[:, match_ncommon_col]
  n1        = halos[:, match_npart1_col]
  n2        = halos[:, match_npart2_col]
  rank      = ncommon**2 / (n1 * n2)  -  np.abs(n1 - n2) / (n1 + n2)

  sort_mask = np.argsort(rank)
  halos     = halos[sort_mask]

  unique, mask = np.unique(halos[:, unique_col], return_index=True)
  halos = halos[mask]
  return halos


def reorder_match_cols(match_header, match_data):
```

```
  global match_id1_col
  global match_id2_col
  global match_hnum1_col
  global match_hnum2_col
  global match_npart1_col
  global match_npart2_col
  global match_ncommon_col

  order = [match_id1_col, match_id2_col, \
           match_hnum1_col, match_hnum2_col, \
           match_npart1_col, match_npart2_col, \
           match_ncommon_col]
  match_header = [match_header[index] for index in order]
  match_data   = match_data[:, order]

  match_id1_col     = 0
  match_id2_col     = 1
  match_hnum1_col   = 2
  match_hnum2_col   = 3
  match_npart1_col  = 4
  match_npart2_col  = 5
  match_ncommon_col = 6

  return match_header, match_data


def count_subs(halos):
  id      = halos[:, id_col]
  parents = halos[:, parents_col]
  parents = parents[parents != -1]
  nsubs   = (id[:, np.newaxis] == parents).sum(axis = 1)
  halos   = np.column_stack((halos, nsubs))
  return halos


def make_header(match, densprof1, densprof2, parents1, parents2, ascii1, ascii2):
  # zeroeth line just lists column number
  total_len = len(match + densprof1 + densprof2 + parents1 + parents2 + ascii1 + ascii2)
  header_line0 = [str(i) for i in range(total_len)]
  header_line0 = '␣␣'.join(header_line0)
  header_line0 = '#' + header_line0

  # first line denotes which file columns are from
  match_repeat = len(match) - 4
  densprof_repeat = len(densprof1 + densprof2) - 4
  parents_repeat = len(parents1 + parents2) - 4
  ascii_repeat = len(ascii1 + ascii2) - 4

  match_part    = '␣␣'.join(['|---', 'cross', 'match'] + ['----'] * match_repeat + ['---|'])
  densprof_part = '␣␣'.join(['|---', 'density', 'profile'] + ['----'] * densprof_repeat + ['---|'])
  parents_part  = '␣␣'.join(['|---', 'rockstar', 'parents'] + ['----'] * parents_repeat + ['---|'])
  ascii_part    = '␣␣'.join(['|---', 'rockstar', 'ascii'] + ['----'] * ascii_repeat + ['---|'])

  header_line1 = '␣␣'.join([match_part, densprof_part, parents_part, ascii_part])
  header_line1 = '#' + header_line1

  # second line labels 2lpt and za columns
  tot_len = len(match + densprof1 + densprof2 + parents1 + parents2 + ascii1 + ascii2)
  header_line2 = ['2lpt' if i % 2 == 0 else 'za' if i % 2 == 1 else 'blah' for i in range(tot_len - 1)]
  header_line2.insert(len(match) - 1, 'matched')
  header_line2 = '␣␣'.join(header_line2)
  header_line2 = '#' + header_line2

  # third line pulls labels from original file headers
  match_part    = match
  densprof_part = interweave(densprof1, densprof2)
  parents_part  = interweave(parents1, parents2)
  ascii_part    = interweave(ascii1, ascii2)

  header_line3 = match_part + densprof_part + parents_part + ascii_part
  header_line3 = '␣␣'.join(header_line3)
  header_line3 = '#' + header_line3

  header = [header_line0, header_line1, header_line2, header_line3]
  return header


def interweave(list1, list2):
  newlist = list1 + list2
  newlist[::2]  = list1
  newlist[1::2] = list2
  return newlist


def interweave_np_2d(array1, array2):
  newarray = np.empty((len(array1), len(array1[0]) + len(array2[0])))
  newarray[:,::2]  = array1
  newarray[:,1::2] = array2
  return newarray


def match_halos(matches, arrays):
```

```
  halos = matches.copy()
  for i, array in enumerate(arrays):
    if array != None:
      match_id_col = i % 2
      halos = sort_stack(halos, array, match_id_col)

  # interweave columns so that matching 2lpt/za columns are adjacent
  tmp_halos = halos
  halos = np.empty((len(tmp_halos), len(tmp_halos[0])))
  halos[:,:len(matches[0])] = matches
  startcol = len(matches[0])
  for i in range(0, len(arrays), 2):
    colrange1 = len(arrays[i][0])
    colrange2 = len(arrays[i+1][0])
    endcol = startcol + colrange1 + colrange2

    cols1 = tmp_halos[:,startcol:startcol+colrange1]
    cols2 = tmp_halos[:,startcol+colrange1:startcol+colrange1+colrange2]

    halos[:,startcol:endcol] = interweave_np_2d(cols1, cols2)
    startcol = endcol
  return halos


def sort_stack(halos, array, match_id_col):
  # add empty columns to halos to later fill with halo data
  rows = len(halos)
  origcols = len(halos[0])
  newcols = len(array[0])
  empty = np.empty((rows, newcols))
  empty[:] = np.nan
  halos = np.column_stack((halos, empty))

  # remove halos from array with no matches
  match_id = halos[:, match_id_col]
  array_id = array[:, id_col]
  array_mask = np.in1d(array_id, match_id)
  array = array[array_mask]

  # create mask so we only add lines for halos in array
  array_id = array[:, id_col]
  halo_mask = np.in1d(match_id, array_id)
  masked_halos = halos[halo_mask]

  # create masks to sort by halo id
  match_id_sort_mask = np.argsort(masked_halos[:, match_id_col])
  sorted_masked_halos = masked_halos[match_id_sort_mask]

  # sort array by halo id and copy to empty columns of view of halos
  array_id_sort_mask = np.argsort(array[:,id_col])
  sorted_masked_halos[:, origcols:] = array[array_id_sort_mask]

  # 'unmask' - put data back in original halos
  masked_halos[match_id_sort_mask] = sorted_masked_halos
  halos[halo_mask] = masked_halos

  return halos


def filter_halos(halos):
  #todo
  return halos


def write_results(output_file, header, halos):
  format = get_format(halos[0])
  with open(output_file, 'w') as fd:
    for line in header:
      fd.write(line + '\n')
    np.savetxt(fd, halos, fmt=format)


def get_format(line):
  format = ['%d' if col in int_cols else '%1.14g' for col in range(len(line))]
  format = ' '.join(format)
  return format


help_string = '''
Available options are:
    -h, --help
    -v, --verbose
    -o <outfile>, --outfile <outfile>
    -m <matchlist>, --match <matchlist>
    -d <densityprofile_file>, --density <densityprofile_file>
    -p <parents_file>, --parents <parents_file>
    -a <ascii_files>, --ascii <ascii_files> - must be last)
'''
shortopts = "hvo:m:d:p:a"
longopts  = ["help", "verbose", "outfile=", "matchfile=", "density=", "parents=", "ascii"]

lt_cols = []
```

```
lt_vals = []

gt_cols = []
gt_vals = []

eq_cols = []
eq_vals = []

ne_cols = []
ne_vals = []

#int_cols = [0, 1, 2, 3, 4, 5, 6, 7, 8]
int_cols = []

match_id2_col             =  1
match_npart2_col          =  2
match_id1_col             =  4
match_npart1_col          =  5
match_ncommon_col         =  6
match_hnum1_col           =  3
match_hnum2_col           =  0

id_col                    =  0      #  col of each input file
sort_col                  = 47      #  col of final table - use None to turn off sorting
parents_col               = -1

filter_bad_matches        =  True
filter_duplicate_matches  =  False
reorder_match_columns     =  True
filter_halo_properties    =  False
min_npart                 =  20     #  use 0 or None to use all size halos
minperc_ncommon           =  0.05   #  a fraction, use 0 or None to use any match percent


if __name__ == '__main__':
  main()
```

F.2 PBS Submission Script (Bash)

```
#!/usr/bin/env bash
#PBS -M djsissom@gmail.com
#PBS -m bae
#PBS -l nodes=1:ppn=1
#PBS -l pmem=40000mb
#PBS -l mem=4000mb
#PBS -l walltime=1:00:00
#PBS -o out.log
#PBS -j oe

minsnap=0
maxsnap=61

minbox=1
maxbox=3

# Change to working directory
echo $PBS_NODEFILE
cd $PBS_O_WORKDIR

for ((i=$minbox; i<=$maxbox; i++)); do

  for ((snap=$minsnap; snap<=$maxsnap; snap++)); do

    if [ $snap -lt 10 ]; then
      j=00$snap
    elif [ $snap -lt 100 ]; then
      j=0$snap
    fi

    base_dir=~/projects/simulations/rockstar/box${i}
    crossmatch_dir=${base_dir}/crossmatch/snap${j}
    snap_dir_2lpt=${base_dir}/2lpt/snap${j}
    snap_dir_za=${base_dir}/za/snap${j}
    logfile=${crossmatch_dir}/match_halos.log

    echo "Starting␣box${i}␣snap${j}..."

    {
      #mpiexec -verbose -n 1 \
      ./match.py -o ${crossmatch_dir}/halos.dat \
                 -m ${crossmatch_dir}/crossmatch_000.txt \
                 -d ${snap_dir_2lpt}/halos/density_profile_halos.dat \
                 -d ${snap_dir_za}/halos/density_profile_halos.dat \
                 -p ${snap_dir_2lpt}/halos/out_0.list.parents \
                 -p ${snap_dir_za}/halos/out_0.list.parents \
                 -a \
                 ${snap_dir_2lpt}/halos/halos_0.*.ascii \
                 ${snap_dir_za}/halos/halos_0.*.ascii \
                 > ${logfile} 2>&1

      echo 'Aligning columns...' >> ${logfile} 2>&1
```

```
53        column -t ${crossmatch_dir}/halos.dat > ${crossmatch_dir}/tmp156546.dat 2>> ${logfile}
54        mv ${crossmatch_dir}/tmp156546.dat ${crossmatch_dir}/halos.dat 2>> ${logfile}
55        echo 'Finished.' >> ${logfile} 2>&1
56        echo "Finished box${i} snap${j}"
57      }
58      #} &
59
60    done
61
62  done
63
64  wait
65
66  # - end of script
```

Appendix G

Halo Comparison Code

G.1 Particle Comparison (Python)

```
#!/usr/bin/env python

import sys
import bgc2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from matplotlib.ticker import MultipleLocator
from scipy.optimize import curve_fit
from scipy.stats import chisquare

#id1, id2 = 727, 4420 # 2lpt first
#id1, id2 = 4416, 727 # za first

#id1, id2 = 4416, 4420 # both za
#id1, id2 = 4416, 4416 # both za

#id1, id2 = 653, 4355
#id1, id2 = 38, 3803
#id1, id2 = 155099, 80362
#id1, id2 = 98722, 14357
id1, id2 = 84289, 143514


#read_mode = 'ascii2'
read_mode = 'bgc2'

if read_mode == 'bgc2':
  use_bgc2 = True
  use_all = False
  multiple_halos = True
  individual_masses = False
  halo_id = 146289
  nbins = 50
  nfit = 500
  ooms = 3.0
  mass_scale = 1.0
  common_mass = 5.33423e5
  dist_scale = 1.0e3
  #res_limit = 0.488
  res_limit = 4.0
  #res_limit = 10.0
  #draw_frac = 1.0e-2
  draw_frac = 1.0
  tick_base_major = 10.0
  tick_base_minor = 1.0
elif read_mode == 'ascii':
  use_bgc2 = False
  use_all = True
  individual_masses = True
  halo_id = 0
  nbins = 100
  nfit = 500
  ooms = 5.0
  mass_scale = 1.0e12
  dist_scale = 200.0
  res_limit = 1.0e-2
  draw_frac = 2.0e-4
  tick_base_major = 80.0
  tick_base_minor = 20.0
elif read_mode == 'ascii2':
  use_bgc2 = False
  use_all = True
  individual_masses = True
  halo_id = 0
  nbins = 100
  nfit = 500
  ooms = 3.5
  mass_scale = 1.0e10
  dist_scale = 1.0
  res_limit = 3.0e-1
  draw_frac = 1.0e-2
  tick_base_major = 200.0
  tick_base_minor = 40.0
else:
  sys.exit(98712)

outfile = 'halo_properties.txt'
comfile = 'center_of_mass.txt'

```

```
make_plot = True
plot_base = 'density_profile.fig.'
plot_ext = '.eps'
dist_units = 'kpc'
xlabel_proj = [r'X Position (%s h$^{-1}$)' % (dist_units), r'X Position (%s h$^{-1}$)' % (dist_units), r'Y Position (%s h$^{-1}$)' % (dist_units)]
ylabel_proj = [r'Y Position (%s h$^{-1}$)' % (dist_units), r'Z Position (%s h$^{-1}$)' % (dist_units), r'Z Position (%s h$^{-1}$)' % (dist_units)]
xlabel_prof = r'Radius (%s h$^{-1}$)' % (dist_units)
ylabel_prof = r'Density (M$_{\odot}$ %s$^{-3}$ h$^{2}$)' % (dist_units)

#common_mass = 1.0e-7
#common_mass = 1.0e5
mass_col = 0
pos_cols = (1,2,3)
vel_cols = (4,5,6)
halo_id_col = 0

grav_const = 4.3e-6 # kpc M_sol^-1 (km/s)^2

profile_type = 0  # 0 -> nfw, fit rho_0
                  # 1 -> nfw, calculate rho_0
                  # 2 -> nfw, rho_0 middle of leftmost bin above resolution
                  # 3 -> fit outer slope, fit rho_0
                  # 4 -> fit outer slope, calculate rho_0
                  # 5 -> fit outer slope, rho_0 middle of leftmost bin above resolution

def read_files(files):
  data = 0
  for file in files:
    print 'Reading file %s...' % (file)
    if data == 0:
      data = np.genfromtxt(file, comments='#')
    else:
      data = np.append(data, np.genfromtxt(file, comments='#'), axis=0)
  print 'Finished reading files.'
  return data


def calc_density_profile(mass, pos):
  r = np.sqrt(pos[:,0]**2 + pos[:,1]**2 + pos[:,2]**2)
  max_r = r.max()
  #min_r = max_r / 10**ooms
  min_r = res_limit
  log_range = np.log10(max_r) - np.log10(min_r)

  #global nbins
  local_nbins = float(nbins + 1)
  #nbins = len(r) / 1000
  while True:
    bins = np.arange(local_nbins)
    bins = max_r * 10.0**(log_range * bins / (local_nbins-1.0) - log_range)
    bin_mass, r_bins = np.histogram(r, bins, weights=mass)
    if (bin_mass == 0.0).any():
      local_nbins -= 1
      continue
    else:
      break

  #print 'Binning particles using bin edges of \n', r_bins

  rho = bin_mass / (sphere_vol(r_bins[1:]) - sphere_vol(r_bins[:-1]))

  N_bin, blah = np.histogram(r, bins)
  rho_err = poisson_error(N_bin) * rho

  return r_bins, rho, rho_err


def logbin(pos):
  r = np.sqrt(pos[:,0]**2 + pos[:,1]**2 + pos[:,2]**2)
  max_r = r.max()
  min_r = max_r / 10**ooms
  log_range = np.log10(max_r) - np.log10(min_r)

  global nbins
  nbins = float(nbins + 1)
  bins = np.arange(nbins)
  bins = max_r * 10.0**(log_range * bins / (nbins-1.0) - log_range)

  hist, bin_edges = np.histogram(r, bins)
  #print 'Binning particles using bin edges of \n', bin_edges
  return hist, bin_edges


def poisson_error(N):
  err = np.sqrt(N) / N
  return err


def sphere_vol(r):
  volume = (4.0 / 3.0) * np.pi * r**3
```

```
  return volume


def get_rho_0(R_s, R_vir):
  H = 70.0e-3 # km s^-1 kpc^-1
  G = 4.3e-6 # kpc M_sol^-1 (km/s)^2
  rho_crit = 3.0 * H**2 / (8.0 * np.pi * G)

  v = 178
  c = R_vir / R_s
  g = 1.0 / (np.log(1.0+c) - c/(1.0+c))
  delta_char = v * c**3 * g / 3.0

  return rho_crit * delta_char


def nfw_fit_rho0(r, R_s, rho_0):
  return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**2)


def nfw_fit_rho0_log(r, R_s, rho_0):
  r = 10.0**r
  R_s = 10.0**R_s
  rho_0 = 10.0**rho_0
  profile =  rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**2)
  return np.log10(profile)


def nfw_def_rho0(R_vir):
  def _nfw_def_rho0(r, R_s):
    rho_0 = get_rho_0(R_s, R_vir)
    return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**2)
  return _nfw_def_rho0


def nfw_databin_rho0(rho_0):
  def _nfw_databin_rho0(r, R_s):
    return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**2)
  return _nfw_databin_rho0


def dm_profile_fit_rho0_log(r, R_s, rho_0, alpha):
  r = 10.0**r
  R_s = 10.0**R_s
  rho_0 = 10.0**rho_0
  alpha = 10.0**alpha
  profile = rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**alpha)
  return np.log10(profile)


def dm_profile_fit_rho0(r, R_s, rho_0, alpha):
  return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**alpha)


def dm_profile_def_rho0(R_vir):
  def _dm_profile_def_rho0(r, R_s, alpha):
    rho_0 = get_rho_0(R_s, R_vir)
    return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**alpha)
  return _dm_profile_def_rho0


def dm_profile_databin_rho0(rho_0):
  def _dm_profile_databin_rho0(r, R_s, alpha):
    return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**alpha)
  return _dm_profile_databin_rho0


def fit_profile(r, rho, err=None, R_vir=None):
  for i in range(len(r)):
    if r[i] > res_limit:
      rho_0_databin = rho[i]
      first_good_bin = i
      break
  #--------- choose one fitting type ---------#
  #popt, pcov = curve_fit(nfw_fit_rho0, r, rho, sigma=err)
  #popt, pcov = curve_fit(nfw_def_rho0(R_vir), r, rho, p0=[10.0], sigma=err)
  #popt, pcov = curve_fit(nfw_databin_rho0(rho_0_databin), r, rho, sigma=err)
  blah = 2
  if blah == 0:
    for i in range(100):
      a = 2.0 * np.random.random() * 0.1 * r.max()
      b = 2.0 * np.random.random() * 10.0
      c = 2.0 * np.random.random() * 2.0
      try:
        popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, p0=[a,b,c], sigma=err)
      except RuntimeError:
        continue
      if (popt[0] < r.max()) and (popt[2] >= 0.0):
        break
      elif i >= 99:
        print 'no good fit found for this halo...'
#        return None, None, None, None, None
```

```
  elif blah == 1:
    #a = r.max() / 100.0
    a = 0.001
    b = rho[first_good_bin]
    c = 0.001
    #popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, sigma=err)
    print '-------------------------------------'
    print 'rho_0 before =', b
    #try:
      #popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, p0=[a,b,c], sigma=err, maxfev=1, xtol=100.0)
    popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, p0=[a,b,c], sigma=err, xtol=1.0e-1)
    #except RuntimeError:
    #  print 'just checking for now...'
    print 'rho_0 after =', popt[1]
    #sys.exit()
  elif blah == 2:
    #popt, pcov = curve_fit(dm_profile_fit_rho0_log, np.log10(r), np.log10(rho), sigma=np.log10(err))
    popt, pcov = curve_fit(nfw_fit_rho0_log, np.log10(r), np.log10(rho), sigma=np.log10(err))
    popt = 10.0**popt
    pcov = 10.0**pcov
  elif blah == 3:
    popt, pcov = curve_fit(nfw_fit_rho0, r, rho, sigma=err)

  #popt, pcov = curve_fit(dm_profile_def_rho0(R_vir), r, rho, sigma=err)
  #popt, pcov = curve_fit(dm_profile_databin_rho0(rho_0_databin), r, rho, sigma=err)
  #------------------------------------------#
  print 'fit_params =', popt
  print 'covariance =', pcov
  nfw_r = np.linspace(r[0], r[-1], nfit)
  #--------- choose one fitting type ---------#
  nfw_fit = nfw_fit_rho0(nfw_r, popt[0], popt[1])
  #nfw_fit = nfw_def_rho0(R_vir)(nfw_r, popt[0])
  #nfw_fit = nfw_databin_rho0(rho_0_databin)(nfw_r, popt[0])
  #nfw_fit = dm_profile_fit_rho0(nfw_r, popt[0], popt[1], popt[2])
  #nfw_fit = dm_profile_def_rho0(R_vir)(nfw_r, popt[0], popt[1])
  #nfw_fit = dm_profile_databin_rho0(rho_0_databin)(nfw_r, popt[0], popt[1])
  #------------------------------------------#
  #--------- choose one fitting type ---------#
  chi2_fit = nfw_fit_rho0(r, popt[0], popt[1])
  #chi2_fit = nfw_def_rho0(R_vir)(r, popt[0])
  #chi2_fit = nfw_databin_rho0(rho_0_databin)(r, popt[0])
  #chi2_fit = dm_profile_fit_rho0(r, popt[0], popt[1], popt[2])
  #chi2_fit = dm_profile_def_rho0(R_vir)(r, popt[0], popt[1])
  #chi2_fit = dm_profile_databin_rho0(rho_0_databin)(r, popt[0], popt[1])
  #------------------------------------------#

  chi2 = chisquare(np.log10(rho[first_good_bin:]), np.log10(chi2_fit[first_good_bin:]))
  chi2_nolog = chisquare(rho[first_good_bin:], chi2_fit[first_good_bin:])
  print 'chi_square =', chi2
  print 'chi_square_nolog =', chi2_nolog
  return nfw_r, nfw_fit, popt, pcov, chi2[0]


def draw_projection(fig, place, plot_lim, x, y):
  ax = plt.subplot(1,3,place+1, aspect='equal')
  im = ax.plot(x, y, linestyle='', marker='.', markersize=1, markeredgecolor='blue')
  ax.set_xlabel(xlabel_proj[place])
  ax.set_ylabel(ylabel_proj[place])
  ax.set_xlim(-plot_lim, plot_lim)
  ax.set_ylim(-plot_lim, plot_lim)
#  ax.xaxis.set_major_locator(MultipleLocator(tick_base_major))
#  ax.xaxis.set_minor_locator(MultipleLocator(tick_base_minor))
#  ax.yaxis.set_major_locator(MultipleLocator(tick_base_major))
#  ax.yaxis.set_minor_locator(MultipleLocator(tick_base_minor))
  return fig, ax


def draw_projection_again(fig, ax, x, y):
  im = ax.plot(x, y, linestyle='', marker='.', markersize=1, markeredgecolor='red')
  return fig


def draw_density_profile(fig, r, rho, err=None):
  ax = plt.subplot(2,1,2)
  im = ax.loglog(r, rho, linestyle='steps-mid-')
  line1 = ax.axvline(res_limit, color='black', linestyle=':')
  #ax.set_xlim(r_bins[0], r_bins[-1])
  ax.set_xlim(r[0] - (r[1]-r[0]), r[-1] + (r[-1]-r[-2]))
  ax.set_xlabel(xlabel_prof)
  ax.set_ylabel(ylabel_prof)
  if err != None:
    err_bars = ax.errorbar(r, rho, yerr=err,linestyle='None')
  return fig, ax


def draw_nfw_profile(fig, ax, r, rho, R_s=None):
  ax.loglog(r, rho, linestyle='-', color='red')
  if R_s != None:
    line = ax.axvline(R_s, color='purple', linestyle='-.')
  return fig
```

```
355 def calc_kinetic_energy(mass, vel):
356   vsq = vel[:,0]**2 + vel[:,1]**2 + vel[:,2]**2
357   energy = 0.5 * np.sum(mass*vsq)
358   return energy
359
360
361 def calc_potential_energy(mass, pos):
362   local_sqrt = np.sqrt
363   partial_sum = 0.0
364   for i in range(len(mass)):
365     for j in range(len(mass)):
366       if j != i:
367         r_diff = local_sqrt((pos[i,0] - pos[j,0])**2 + (pos[i,1] - pos[j,1])**2 + (pos[i,2] - pos[j,2])**2)
368         partial_sum = partial_sum - mass[i]*mass[j]/r_diff
369   energy = partial_sum * grav_const / 2.0
370   return energy
371
372
373 def calc_angular_momentum(mass, pos, vel):
374   ang_mom_x = np.sum(mass * (pos[:,1] * vel[:,2] - pos[:,2] * vel[:,1]))
375   ang_mom_y = np.sum(mass * (pos[:,2] * vel[:,0] - pos[:,0] * vel[:,2]))
376   ang_mom_z = np.sum(mass * (pos[:,1] * vel[:,2] - pos[:,2] * vel[:,1]))
377   ang_mom = np.sqrt(ang_mom_x**2 + ang_mom_y**2 + ang_mom_z**2)
378   return ang_mom
379
380
381 def main():
382   #for input_file in sys.argv[1:]:
383   #header1, halos1, particles1 = bgc2.read_bgc2(sys.argv[1])
384   #header2, halos2, particles2 = bgc2.read_bgc2(sys.argv[2])
385
386   nargs = len(sys.argv) - 1
387   if (float(nargs) % 2.0) != 0.0:
388     print 'number␣of␣arguments␣must␣be␣even'
389     sys.exit()
390
391   for i in range(nargs / 2):
392     i += 1
393     temp_header1, temp_halos1, temp_particles1 = bgc2.read_bgc2(sys.argv[i])
394     temp_header2, temp_halos2, temp_particles2 = bgc2.read_bgc2(sys.argv[(nargs / 2) + i])
395     if i == 1:
396       halos1, particles1 = temp_halos1, temp_particles1
397       halos2, particles2 = temp_halos2, temp_particles2
398     else:
399       halos1 = np.append(halos1, temp_halos1, axis=0)
400       halos2 = np.append(halos2, temp_halos2, axis=0)
401       particles1 = np.append(particles1, temp_particles1, axis=0)
402       particles2 = np.append(particles2, temp_particles2, axis=0)
403
404   halos1 = np.asarray(halos1)
405   halos2 = np.asarray(halos2)
406   #indices = np.argsort(halos[:,2])      # sort by number of particles
407   #indices = indices[::-1]               # start with the biggest
408
409   itteration = 0
410   #for index in indices[:1000]:
411   #for index in indices:
412   for index in range(halos1.shape[0]):
413     halo_id = halos1[index,0]
414     if (halo_id == id1):
415       print '------------------------------------------------------------'
416
417       halo_particles1 = np.asarray(particles1[index])
418       pos1 = halo_particles1[:,pos_cols[0]:pos_cols[0]+3] * dist_scale
419       #vel1 = halo_particles1[:,vel_cols[0]:vel_cols[0]+3]
420
421       r_vir1 = halos1[index][4] * dist_scale
422       halo_mass1 = halos1[index][5]
423       halo_pos1 = np.array([halos1[index][6] * dist_scale, halos1[index][7] * dist_scale, halos1[index][8] *
      dist_scale])
424       #halo_vel1 = np.array([halos1[index][9], halos1[index][10], halos1[index][11]])
425
426       print 'Using␣%d␣particles␣in␣halo␣%d.' % (halo_particles1.shape[0], halo_id)
427
428       # Find center of mass
429       #pos = pos - halo_pos
430       #vel = vel - halo_vel
431
432       # Pick only a certain perentage of particles for projection plots
433       if (draw_frac < 1.0):
434         np.random.shuffle(pos1)
435         pos1 = pos1[:(draw_frac*pos1.shape[0])]
436
437   for index in range(halos2.shape[0]):
438     halo_id = halos2[index,0]
439     if (halo_id == id2):
440       print '------------------------------------------------------------'
441
442       halo_particles2 = np.asarray(particles2[index])
443       pos2 = halo_particles2[:,pos_cols[0]:pos_cols[0]+3] * dist_scale
444       #vel2 = halo_particles2[:,vel_cols[0]:vel_cols[0]+3]
445
```

```
      r_vir2 = halos2[index][4] * dist_scale
      halo_mass2 = halos2[index][5]
      halo_pos2 = np.array([halos2[index][6] * dist_scale, halos2[index][7] * dist_scale, halos2[index][8] *
     dist_scale])
      #halo_vel2 = np.array([halos2[index][9], halos2[index][10], halos2[index][11]])

      print 'Using␣%d␣particles␣in␣halo␣%d.' % (halo_particles2.shape[0], halo_id)

      # Find center of mass
      #pos = pos - halo_pos
      #vel = vel - halo_vel

      # Pick only a certain perentage of particles for projection plots
      if (draw_frac < 1.0):
        np.random.shuffle(pos2)
        pos2 = pos2[:(draw_frac*pos2.shape[0])]

  # Find the maximum of x, y, or z to be limit of projection plots
  center = (halo_pos1 + halo_pos2) / 2.0
  pos1 = pos1 - center
  pos2 = pos2 - center
  halo_pos1 = halo_pos1 - center
  halo_pos2 = halo_pos2 - center
  plot_lim = np.append(pos1, pos2).max()

  # Plot density profile histogram
  if (make_plot == True):
    fig = plt.figure()

    fig, ax = draw_projection(fig, 0, plot_lim, pos1[:,0], pos1[:,1])
    fig = draw_projection_again(fig, ax, pos2[:,0], pos2[:,1])
    ax.add_patch(Circle((halo_pos1[0], halo_pos1[1]), r_vir1, fc="None", ec="black", lw=1))
    ax.add_patch(Circle((halo_pos2[0], halo_pos2[1]), r_vir2, fc="None", ec="black", lw=1))

    fig, ax = draw_projection(fig, 1, plot_lim, pos1[:,0], pos1[:,2])
    fig = draw_projection_again(fig, ax, pos2[:,0], pos2[:,2])
    ax.add_patch(Circle((halo_pos1[0], halo_pos1[2]), r_vir1, fc="None", ec="black", lw=1))
    ax.add_patch(Circle((halo_pos2[0], halo_pos2[2]), r_vir2, fc="None", ec="black", lw=1))

    fig, ax = draw_projection(fig, 2, plot_lim, pos1[:,1], pos1[:,2])
    fig = draw_projection_again(fig, ax, pos2[:,1], pos2[:,2])
    ax.add_patch(Circle((halo_pos1[1], halo_pos1[2]), r_vir1, fc="None", ec="black", lw=1))
    ax.add_patch(Circle((halo_pos2[1], halo_pos2[2]), r_vir2, fc="None", ec="black", lw=1))

    #fig, ax = draw_density_profile(fig, mid_bins, rho, err=rho_err)
    #fig = draw_nfw_profile(fig, ax, nfw_r, nfw_fit, R_s=scale_radius)
    fig.tight_layout()
    #plt.savefig(plot_base+str(itteration)+plot_ext)
    plt.savefig('test.eps')


if __name__ == '__main__':
  main()
```

G.2 Density Comparison (Python)

```
#!/usr/bin/env python

import sys
import bgc2
import numpy as np
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from matplotlib import patheffects
from mpl_toolkits.axes_grid1 import ImageGrid
from scipy.stats import ks_2samp
from scipy.stats import chisquare
from scipy.optimize import curve_fit
from scipy.ndimage.filters import gaussian_filter
from ipdb import set_trace


#### Note:  only run one box pair at a time.
#### ex:    ./compare.py /crossmatch_dir/halos.dat /2lpt_dir/halos_0.*.bgc2 /za_dir/halos_0.*.bgc2

def main():
    crossmatched_halo_file, bgc2_2lpt_files, bgc2_za_files = parse_args(sys.argv[1:])

    header, halos = read_files(crossmatched_halo_file, header_line = 3)

    bgc2_2lpt_header, bgc2_2lpt_halos, bgc2_2lpt_particles = get_bgc2_data(bgc2_2lpt_files)
    bgc2_za_header,   bgc2_za_halos,   bgc2_za_particles   = get_bgc2_data(bgc2_za_files)

    header = np.asarray(header)
    bgc2_2lpt_halos, bgc2_za_halos = map(np.asarray, (bgc2_2lpt_halos, bgc2_za_halos))

    if sort_col != None:
        halos = sort_by_column(halos, sort_col)
    if remove_nonfit_halos:
```

```
        halos = remove_nans(halos)
    if global_filter_halos:
        halos = filter_halos(halos)
    if (nhalos != None) or (nhalos != 0):
        #halos = halos[:nhalos]
        halos = halos[[0,70]]        ############################## hard coded for the moment
        #halos = halos[10000:10050]

    header, halos = add_c_columns(header, halos)
    header = reduce_header(header)

    for i, halo_pair in enumerate(halos):
        make_plot(i, header, halo_pair, bgc2_2lpt_halos, bgc2_za_halos, \
                  bgc2_2lpt_particles, bgc2_za_particles)

    print 'Finished all plots.'


def parse_args(args):
    crossmatched_halo_file = args[0]
    if len(args[1:]) % 2 != 0.0:
        print 'Must call with even number of bgc2 files...exiting.'
        sys.exit(-1)
    bgc2_files = args[1:]
    bgc2_2lpt_files = bgc2_files[:len(bgc2_files)/2]
    bgc2_za_files = bgc2_files[len(bgc2_files)/2:]
    return crossmatched_halo_file, bgc2_2lpt_files, bgc2_za_files


def read_files(files, header_line = None, comment_char = '#'):
    header = None
    data = None
    if type(files) == str:
        files = [files]

    if header_line != None:
        with open(files[0], 'r') as fd:
            for line in range(header_line):
                fd.readline()
            header = fd.readline()
        if header[0] != comment_char:
            print "Header must start with a '%s'" % comment_char
            sys.exit(4)
        header = header[1:]
        header = header.split()

    for file in files:
        print 'Reading file %s...' % (file)
        if data == None:
            data = np.genfromtxt(file, comments=comment_char)
        else:
            data = np.append(data, np.genfromtxt(file, comments=comment_char), axis=0)

    print 'Finished reading files.'
    if header_line == None:
        return data
    else:
        return header, data


def get_bgc2_data(bgc2_files):
    header    = None
    halos     = None
    particles = None
    for bgc2_file in bgc2_files:
        print 'Reading file %s...' % (bgc2_file)
        tmp_header, tmp_halos, tmp_particles = bgc2.read_bgc2(bgc2_file)
        if header == None:
            header = tmp_header
            halos = tmp_halos
            particles = tmp_particles
        else:
            halos = np.append(halos, tmp_halos, axis=0)
            particles = np.append(particles, tmp_particles, axis=0)
    print 'Finished reading bgc2 files.'
    return header, halos, particles


def sort_by_column(halos, col):
    print 'Sorting halos...'
    mask = np.argsort(halos[:, col])
    mask = mask[::-1]
    halos = halos[mask]
    return halos


def remove_nans(halos):
    print 'Removing NaNs...'
    halos = halos[halos[:,c_2lpt_col] != -9999]
    halos = halos[np.isfinite(halos[:,c_2lpt_col])]
    halos = halos[np.isfinite(halos[:,c_za_col])]
    return halos
```

```
128
129
130 def filter_halos(halos):
131     print 'Filtering data...'
132     for col, val in zip(lt_cols, lt_vals):
133         halos = halos[halos[:, col] <= val]
134     for col, val in zip(gt_cols, gt_vals):
135         halos = halos[halos[:, col] >= val]
136     for col, val in zip(eq_cols, eq_vals):
137         halos = halos[halos[:, col] == val]
138     for col, val in zip(ne_cols, ne_vals):
139         halos = halos[halos[:, col] != val]
140     return halos
141
142
143 def add_c_columns(header, halos):
144     c1_rockstar = halos[:, Rv1_col] / halos[:, Rs1_col]
145     c2_rockstar = halos[:, Rv2_col] / halos[:, Rs2_col]
146     halos = np.column_stack((halos, c1_rockstar, c2_rockstar))
147     header = np.append(header, 'c_rockstar')
148     header = np.append(header, 'c_rockstar')
149     return header, halos
150
151
152 def reduce_header(header):
153     header_2lpt = header[print_cols_2lpt]
154     header_za   = header[print_cols_za]
155     if (header_2lpt == header_za).all():
156         header = header_2lpt
157     else:
158         print 'column mismatch... exiting'
159         set_trace()
160         sys.exit(123)
161     return header
162
163
164 def make_plot(itteration, header, halo_pair, bgc2_halos_2lpt, bgc2_halos_za, \
165               bgc2_particles_2lpt, bgc2_particles_za):
166     id_2lpt = halo_pair[id_col_2lpt]
167     id_za   = halo_pair[id_col_za]
168     properties_2lpt = halo_pair[print_cols_2lpt]
169     properties_za   = halo_pair[print_cols_za]
170
171     # find 2lpt and za halo from id
172     halo_index_2lpt = np.where(bgc2_halos_2lpt[:, halo_id_col] == id_2lpt)[0][0]
173     halo_index_za   = np.where(bgc2_halos_za[:, halo_id_col]   == id_za)[0][0]
174
175     bgc2_halos_2lpt = bgc2_halos_2lpt[halo_index_2lpt]
176     bgc2_halos_za   = bgc2_halos_za[halo_index_za]
177
178     # convert particles to numpy arrays
179     bgc2_particles_2lpt = np.asarray(bgc2_particles_2lpt[halo_index_2lpt])
180     bgc2_particles_za   = np.asarray(bgc2_particles_za[halo_index_za])
181
182     # make density profiles
183     r_2lpt, rho_2lpt, rho_err_2lpt, r_vir_2lpt = density_profile(bgc2_halos_2lpt, bgc2_particles_2lpt)
184     r_za,   rho_za,   rho_err_za,   r_vir_za   = density_profile(bgc2_halos_za, bgc2_particles_za)
185
186     # fit density profiles
187     nfw_r_2lpt, nfw_rho_2lpt, r_s_2lpt = fit_profile( r_2lpt / r_vir_2lpt, rho_2lpt / rho_2lpt.max(), err =
        rho_err_2lpt / rho_2lpt.max() )
188     nfw_r_za  , nfw_rho_za  , r_s_za   = fit_profile( r_za   / r_vir_za,   rho_za   / rho_za.max(),   err =
        rho_err_za   / rho_za.max()   )
189
190     # de-normalize values
191     nfw_r_2lpt = nfw_r_2lpt * r_vir_2lpt
192     nfw_r_za   = nfw_r_za   * r_vir_za
193     nfw_rho_2lpt = nfw_rho_2lpt * rho_2lpt.max()
194     nfw_rho_za   = nfw_rho_za   * rho_za.max()
195     r_s_2lpt = r_s_2lpt * r_vir_2lpt
196     r_s_za   = r_s_za   * r_vir_za
197
198     # find center of halos and plot limit
199     halo_pos_2lpt = bgc2_halos_2lpt[:,halo_pos_cols] * dist_scale
200     halo_pos_za   = bgc2_halos_za[:,halo_pos_cols] * dist_scale
201     particle_pos_2lpt = bgc2_particles_2lpt[:,particle_pos_cols] * dist_scale
202     particle_pos_za   = bgc2_particles_za[:,particle_pos_cols] * dist_scale
203
204     if wrap_box:
205         for i in range(3):
206             if abs(halo_pos_2lpt[i] - halo_pos_za[i]) > box_size / 2.0:
207                 print "################################## wrapping halo ##################################"
208                 if (halo_pos_2lpt[i] > halo_pos_za[i]):
209                     halo_pos_za[i] += box_size
210                     particle_pos_za[:,i] += box_size
211                 if (halo_pos_2lpt[i] < halo_pos_za[i]):
212                     halo_pos_2lpt[i] += box_size
213                     particle_pos_2lpt[:,i] += box_size
214                 else:
215                     print "error in wrapping"
216                     sys.exit()
217
```

```
    center = (halo_pos_2lpt + halo_pos_za) / 2.0
    halo_pos_2lpt = halo_pos_2lpt - center
    halo_pos_za   = halo_pos_za   - center
    particle_pos_2lpt = particle_pos_2lpt - center
    particle_pos_za   = particle_pos_za   - center

    if zoom_projections:
        plot_lim = zoom_scale
    else:
        plot_lim = np.append(particle_pos_2lpt, particle_pos_za).max()


    r_vir_2lpt = bgc2_halos_2lpt[halo_r_col] * dist_scale
    r_vir_za   = bgc2_halos_za[halo_r_col] * dist_scale

    if make_stats:
        print 'generating plot...'
        fig = plt.figure(figsize = (9.0, 6.0))
        fig = make_projections(fig, 221, halo_pos_2lpt, halo_pos_za, particle_pos_2lpt, particle_pos_za, \
                               r_vir_2lpt, r_vir_za, plot_lim)
        ax = fig.add_subplot(223)
        ax = draw_density_profile(ax, r_2lpt, rho_2lpt, err=rho_err_2lpt, color='blue', label='2lpt')
        ax = draw_density_profile(ax, r_za,   rho_za,   err=rho_err_za,   color='red',  label='za')

        ax = fig.add_subplot(122)
        ax = draw_parameters(ax, header, properties_2lpt, properties_za)

        fig.tight_layout()
        plot_name = "%s%0.3d_(%d,%d)%s" % (plot_base, itteration, id_2lpt, id_za, plot_ext)
        plt.savefig(plot_name, bbox_inches='tight')
        print 'finished plot ' + plot_name

    if make_projection:
        print 'generating density projection plot...'
        fig = plt.figure(figsize = (9.0, 6.0))

        if label_projection:
            ax = fig.add_subplot(111, aspect=2.0/3.2)
            ax = hide_axes(ax)
            ax.set_xlabel(proj_xlabel)
            ax.set_ylabel(proj_ylabel)

        fig = make_projections(fig, 111, halo_pos_2lpt, halo_pos_za, particle_pos_2lpt, particle_pos_za, \
                               r_vir_2lpt, r_vir_za, plot_lim)
        fig.tight_layout()
        plot_name = "%s%0.3d_(%d,%d)%s%s" % (plot_base, itteration, id_2lpt, id_za, proj_name, plot_ext)
        plt.savefig(plot_name, bbox_inches='tight')
        print 'finished density projection plot ' + plot_name

    if make_density_profile:
        print 'generating density profile plot...'
        fig = plt.figure(figsize = (9.0, 12.0))

        if label_projection:
            ax = fig.add_subplot(211, aspect=2.0/3.2)
            ax = hide_axes(ax)
            ax.set_xlabel(proj_xlabel)
            ax.set_ylabel(proj_ylabel)

        fig = make_projections(fig, 211, halo_pos_2lpt, halo_pos_za, particle_pos_2lpt, particle_pos_za, \
                               r_vir_2lpt, r_vir_za, plot_lim)

        ax = fig.add_subplot(212)
        ax = hide_axes(ax)
        ax.set_xlabel(prof_xlabel)
        ax.set_ylabel(prof_ylabel)

        #grid = ImageGrid(fig, 212, nrows_ncols=(2,1), axes_pad=0.24)

        ax1 = fig.add_subplot(413)
        ax1 = draw_density_profile(ax1, r_2lpt, rho_2lpt, err=rho_err_2lpt, color='blue')
        ax1 = draw_nfw_profile(ax1, nfw_r_2lpt, nfw_rho_2lpt, R_s=r_s_2lpt, color='red')

        ax2 = fig.add_subplot(414)
        ax2 = draw_density_profile(ax2, r_za, rho_za, err=rho_err_za, color='blue')
        ax2 = draw_nfw_profile(ax2, nfw_r_za, nfw_rho_za, R_s=r_s_za, color='red')

        if equal_profile_axes:
            ymin = min(ax1.get_ylim()[0], ax2.get_ylim()[0])
            ymax = max(ax1.get_ylim()[1], ax2.get_ylim()[1])
            ax1.set_ylim(ymin, ymax)
            ax2.set_ylim(ymin, ymax)

            xmin = min(ax1.get_xlim()[0], ax2.get_xlim()[0])
            xmax = max(ax1.get_xlim()[1], ax2.get_xlim()[1])
            ax1.set_xlim(xmin, xmax)
            ax2.set_xlim(xmin, xmax)

        if print_text:
            ax1.text(0.95, 0.85, '2LPT', color='black', horizontalalignment='right', verticalalignment='center',
     transform=ax1.transAxes)
```

```
308             ax2.text(0.95, 0.85, 'ZA',   color='black', horizontalalignment='right', verticalalignment='center',
      transform=ax2.transAxes)
309
310
311         #fig.tight_layout()
312         plot_name = "%s%0.3d_(%d,%d)%s%s" % (plot_base, itteration, id_2lpt, id_za, dens_name, plot_ext)
313         plt.savefig(plot_name, bbox_inches='tight')
314         print 'finished density profile plot ' + plot_name
315
316
317
318 def density_profile(halo, particles):
319     r_vir = halo[halo_r_col] * dist_scale
320     halo_pos = halo[halo_pos_cols] * dist_scale
321     #mass = np.ones(particles.shape[0]) * common_mass * mass_scale
322     mass = particles[:,particle_mass_col] * mass_scale
323     pos = particles[:,particle_pos_cols] * dist_scale
324     pos = pos - halo_pos
325
326     r_bins, rho, rho_err = calc_density_profile(mass, pos)
327     mid_bins = 10.0**(0.5 * (np.log10(r_bins[1:]) + np.log10(r_bins[:-1])))
328
329     # Don't pass NaN's to fitting routine
330     rho_err_nonan = np.copy(rho_err)
331     nan_check = np.isnan(rho_err_nonan)
332     for i in range(len(rho_err_nonan)):
333         if (mid_bins[i] < res_limit) or (nan_check[i] == True):
334             rho_err_nonan[i] = 1.0e10
335
336     return mid_bins, rho, rho_err, r_vir
337
338
339 def calc_density_profile(mass, pos):
340     r = np.sqrt(pos[:,0]**2 + pos[:,1]**2 + pos[:,2]**2)
341     max_r = r.max()
342     min_r = res_limit
343     log_range = np.log10(max_r) - np.log10(min_r)
344     local_nbins = float(nbins + 1)
345     while True:
346         bins = np.arange(local_nbins)
347         bins = max_r * 10.0**(log_range * bins / (local_nbins-1.0) - log_range)
348         bin_mass, r_bins = np.histogram(r, bins, weights=mass)
349         if (bin_mass == 0.0).any():
350             local_nbins -= 1
351             continue
352         else:
353             break
354     rho = bin_mass / (sphere_vol(r_bins[1:]) - sphere_vol(r_bins[:-1]))
355     N_bin, blah = np.histogram(r, bins)
356     rho_err = poisson_error(N_bin) * rho
357     return r_bins, rho, rho_err
358
359
360 def sphere_vol(r):
361     volume = (4.0 / 3.0) * np.pi * r**3
362     return volume
363
364
365 def poisson_error(N):
366     err = np.sqrt(N) / N
367     return err
368
369
370 def fit_profile(r, rho, err=None, R_vir=None):
371     popt, pcov = curve_fit(nfw_profile, r, rho, sigma=err, p0=[0.1, 1.0])
372     R_s, rho_0 = popt[0], popt[1]
373     nfw_r = np.linspace(r[0], r[-1], nfit)
374     nfw_rho = nfw_profile(nfw_r, R_s, rho_0)
375     return nfw_r, nfw_rho, R_s
376
377
378 def nfw_profile(r, R_s, rho_0):
379     if R_s >= 1.0:
380         return (R_s - 1.0) * np.exp(r) + rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**2)
381     return rho_0 / (( r / R_s ) * ( 1.0 + r / R_s )**2)
382
383
384 def filter_column(x, x_col):
385     print 'Filtering data...'
386     x = x[x != -9999]
387     if x_col in lt_cols:
388         val = lt_vals[lt_cols.index(x_col)]
389         x = x[x <= val]
390     if x_col in gt_cols:
391         val = gt_vals[gt_cols.index(x_col)]
392         x = x[x >= val]
393     if x_col in eq_cols:
394         val = eq_vals[eq_cols.index(x_col)]
395         x = x[x == val]
396     if x_col in ne_cols:
397         val = ne_vals[ne_cols.index(x_col)]
398         x = x[x != val]
```

```
    return x


def draw_hist(fig, ax, x, x_min=None, x_max=None, use_log=False, color=None, label=None):
    if use_log:
        xbins = np.logspace(np.log10(x_min), np.log10(x_max), num=nbins+1)
        ax.set_xscale('log')
    else:
        xbins = np.linspace(x_min, x_max, num=nbins+1)

    n, bins, patches = ax.hist(x, bins=xbins, histtype='step', log=ylog, color=color, label=label)
    return fig, ax, n, bins, patches


def add_text(fig, ax, textstr):
    props = dict(boxstyle='round', facecolor='white', alpha=0.7)
    ax.text(0.02, 0.08, textstr, transform=ax.transAxes, fontsize=14, \
            verticalalignment='top', bbox=props)
    return fig, ax


def make_projections(fig, position, halo_pos1, halo_pos2, pos1, pos2, r_vir1, r_vir2, plot_lim):
    #grid = ImageGrid(fig, position, nrows_ncols=(2,3), axes_pad=0.05, cbar_mode='single')
    grid = ImageGrid(fig, position, nrows_ncols=(2,3), axes_pad=0.12, cbar_mode='single')
    for i, (x, y, hx, hy, r) in enumerate(zip( \
            (pos1[:,0], pos1[:,0], pos1[:,1], pos2[:,0], pos2[:,0], pos2[:,1]), \
            (pos1[:,1], pos1[:,2], pos1[:,2], pos2[:,1], pos2[:,2], pos2[:,2]), \
            (halo_pos1[0], halo_pos1[0], halo_pos1[1], halo_pos2[0], halo_pos2[0], halo_pos2[1]), \
            (halo_pos1[1], halo_pos1[2], halo_pos1[2], halo_pos2[1], halo_pos2[2], halo_pos2[2]), \
            (r_vir1, r_vir1, r_vir1, r_vir2, r_vir2, r_vir2))):
        ax = grid[i]
        draw_projection(ax, x, y, hx, hy, r, plot_lim)
        if print_text:
            if i == 0:
                ax.text(0.05, 0.12, '2LPT', color='white', horizontalalignment='left', verticalalignment='center'
 , transform=ax.transAxes, path_effects=[patheffects.withStroke(linewidth=3, foreground='black')])
            if i == 3:
                ax.text(0.05, 0.12, 'ZA',   color='white', horizontalalignment='left', verticalalignment='center'
 , transform=ax.transAxes, path_effects=[patheffects.withStroke(linewidth=3, foreground='black')])

            if i == 0:
                ax.text(0.95, 0.88, 'XY', color='white', horizontalalignment='right', verticalalignment='center',
  transform=ax.transAxes, path_effects=[patheffects.withStroke(linewidth=3, foreground='black')])
            if i == 1:
                ax.text(0.95, 0.88, 'XZ', color='white', horizontalalignment='right', verticalalignment='center',
  transform=ax.transAxes, path_effects=[patheffects.withStroke(linewidth=3, foreground='black')])
            if i == 2:
                ax.text(0.95, 0.88, 'YZ', color='white', horizontalalignment='right', verticalalignment='center',
  transform=ax.transAxes, path_effects=[patheffects.withStroke(linewidth=3, foreground='black')])
    return fig


def draw_projection(ax, x, y, hx, hy, r, plot_lim):
    limits = [[-plot_lim, plot_lim], [-plot_lim, plot_lim]]
    z, xedges, yedges = np.histogram2d(x, y, bins=npixels, range=limits)
    if log_scale_projections:
        z[z<1.0] = 0.5
        #z = np.log10(z)
        #z = np.log10(z)
        #z[np.isinf(z)] = -0.1
        plot_norm = mpl.colors.LogNorm(vmin = 1, vmax = z.max(), clip=True)
        #plot_norm = None
    else:
        plot_norm = None
    if extra_smoothing:
        z = gaussian_filter(z, smoothing_radius)
    im = ax.imshow(z.T, extent=(-plot_lim, plot_lim, -plot_lim, plot_lim), \
            interpolation='gaussian', origin='lower', cmap=colormap, norm=plot_norm)
            #interpolation='gaussian', origin='lower', cmap=colormap)
    ax.locator_params(nbins=6)
    if draw_circle:
        ax.add_patch(Circle((hx, hy), r, fc="None", ec="black", lw=1))
    if draw_contours:
        x_midpoints = (xedges[:-1] + xedges[1:]) / 2.0
        y_midpoints = (yedges[:-1] + yedges[1:]) / 2.0
        X, Y = np.meshgrid(x_midpoints, y_midpoints)
        ax.contour(X, Y, z.T, 2, colors='black', linewidths=4)
        ax.contour(X, Y, z.T, 2, colors='white', linewidths=2)
    if label_colorbar:
        if log_scale_projections:
            log_format = mpl.ticker.LogFormatterMathtext(10, labelOnlyBase=False)
            ax.cax.colorbar(im, format=log_format)
        else:
            ax.cax.colorbar(im)
    else:
        bar = ax.cax.colorbar(im, ticks=[])
        bar.ax.set_yticklabels([])
        #plt.setp(bar.ax.get_yticklabels(), visible=False)


def draw_density_profile(ax, r, rho, err=None, color='black', label=None):
    im = ax.loglog(r, rho, linestyle='steps-mid-', color=color, label=label)
```

```
    line1 = ax.axvline(res_limit, color='black', linestyle=':')
    ax.set_xlim(r[0] - (r[1]-r[0]), r[-1] + (r[-1]-r[-2]))
    #ax.set_xlabel(xlabel_prof)
    #ax.set_ylabel(ylabel_prof)
    if err != None:
        err_bars = ax.errorbar(r, rho, yerr=err,linestyle='None', color=color)
    if label != None:
        ax.legend(fontsize='x-small')
    return ax


def draw_nfw_profile(ax, r, rho, R_s=None, color='black'):
    ax.loglog(r, rho, linestyle='-', color=color)
    if R_s != None:
        line = ax.axvline(R_s, color='purple', linestyle='-.')
    return ax


def draw_parameters(ax, header, params1, params2):
    strlen = 12
    header  = [str(item)[:strlen] for item in header]
    params1 = [str(item)[:strlen] for item in params1]
    params2 = [str(item)[:strlen] for item in params2]
    header.insert(0, 'simulation')
    params1.insert(0, '--␣2lpt␣--')
    params2.insert(0, '---␣za␣---')
    header = '\n'.join(header)
    params1 = '\n'.join(params1)
    params2 = '\n'.join(params2)
    ax.text(0.05, 0.5, header,  horizontalalignment="left", verticalalignment="center", transform=ax.transAxes)
    ax.text(0.40, 0.5, params1, horizontalalignment="left", verticalalignment="center", transform=ax.transAxes)
    ax.text(0.75, 0.5, params2, horizontalalignment="left", verticalalignment="center", transform=ax.transAxes)
    ax.axis('off')
    return ax


def hide_axes(ax):
    ax.spines['top'].set_color('none')
    ax.spines['bottom'].set_color('none')
    ax.spines['left'].set_color('none')
    ax.spines['right'].set_color('none')
    ax.tick_params(labelcolor='w', top='off', bottom='off', left='off', right='off')
    return ax




nhalos = 1
sort_col = 9  # density_profile 2lpt halo mass
#sort_col = 47  # rockstar 2lpt halo mass (M200c)

nbins = 40
nfit = 100
npixels = 30
#npixels = 100
smoothing_radius = 0.9
remove_nonfit_halos = True
global_filter_halos = True
column_filter_halos = True
log_scale_projections = True
wrap_box = False
label_colorbar = False
label_projection = True
zoom_projections = True
zoom_scale = 18.0 # kpc
draw_circle = False
draw_contours = True
extra_smoothing = True
label_proj = True
label_2lpt_za = True
equal_profile_axes = True
print_text = True

box_size = 10000.0 # kpc

id_col_2lpt = 0
id_col_za   = 1

print_cols_2lpt  = [43, 57, 6,  9, 17, 23, 31, 47, 51, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 91, 93, 97,  99,
     101, 103, 105, 107, 111, 163, 201, -2]
print_cols_za    = [44, 58, 6, 10, 18, 24, 32, 48, 52, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 92, 94, 98, 100,
     102, 104, 106, 108, 112, 164, 202, -1]

Rv1_col = 53
Rv2_col = 54
Rs1_col = 55
Rs2_col = 56

c_2lpt_col = 17
c_za_col  = 18

# c_2lpt, c_za, chi2_2lpt, chi2_za
```

```
lt_cols = [17, 18, 37, 38]
lt_vals = [100.0, 100.0, 10.0, 10.0]

# c_2lpt, c_za, rho_0_2lpt, rho_0_za, chi2_2lpt, chi2_za
gt_cols = [17, 18, 31, 32, 37, 38]
gt_vals = [1.0, 1.0, 0.0, 0.0, 0.0, 0.0]

eq_cols = []
eq_vals = []

ne_cols = []
ne_vals = []

# bgc2 halo array columns
halo_id_col   = 0
halo_r_col    = 4
halo_mass_col = 5
halo_pos_cols = [6,7,8]

# bgc2 particle array columns
particle_mass_col = 0
particle_pos_cols = [1,2,3]
particle_vel_cols = [4,5,6]

mass_scale = 1.0
common_mass = 5.33423e5
dist_scale = 1.0e3
res_limit = 0.5  #changed from 4.0 to 0.5 to match density_profile.py <-- maybe check why it was 4.0?
nfit = 500

dist_units = 'kpc'
#xlabel_proj = [r'X Position (%s h$^{-1}$)' % (dist_units), r'X Position (%s h$^{-1}$)' % (dist_units), r'Y Position (%s h$^{-1}$)' % (dist_units)]
#ylabel_proj = [r'Y Position (%s h$^{-1}$)' % (dist_units), r'Z Position (%s h$^{-1}$)' % (dist_units), r'Z Position (%s h$^{-1}$)' % (dist_units)]
proj_xlabel = r'Position␣(kpc␣h$^{-1}$)'
proj_ylabel = r'Position␣(kpc␣h$^{-1}$)'
prof_xlabel = r'Radius␣(%s␣h$^{-1}$)' % (dist_units)
prof_ylabel = r'Density␣(M$_{\odot}$␣%s$^{-3}$␣h$^{2}$)' % (dist_units)

#colormap = 'ocean_r'
colormap = 'rainbow'
plot_base = 'plots/halo_pair_'
proj_name = '_proj'
dens_name = '_dens'
plot_ext  = '.eps'

make_stats = False
make_projection = False
make_density_profile = True

plot_dest_type = 'paper'
if plot_dest_type == 'paper':
    mpl.rcParams['font.family'] = 'serif'
    mpl.rcParams['font.size'] = 16
    mpl.rcParams['axes.linewidth'] = 3
    mpl.rcParams['lines.linewidth'] = 4
    mpl.rcParams['patch.linewidth'] = 4
    mpl.rcParams['xtick.major.width'] = 3
    mpl.rcParams['ytick.major.width'] = 3
    mpl.rcParams['xtick.major.size'] = 8
    mpl.rcParams['ytick.major.size'] = 8


if __name__ == '__main__':
    main()
```

Appendix H

Concentration Comparison Code (Python)

```
#!/usr/bin/env python

import sys
import numpy as np
from ipdb import set_trace

def main():
    # Read in particle files
    header, halos = read_files(sys.argv[1:], header_line = 3)

    if remove_nonfit_halos:
        print 'Removing␣NaNs...'
        halos = halos[np.isfinite(halos[:,c_lpt_col])]
        halos = halos[np.isfinite(halos[:,c_za_col])]

    if global_filter_halos:
        print 'Filtering␣data...'
        for col, val in zip(glob_lt_cols, glob_lt_vals):
            halos = halos[halos[:, col] <= val]
        for col, val in zip(glob_gt_cols, glob_gt_vals):
            halos = halos[halos[:, col] >= val]
        for col, val in zip(glob_eq_cols, glob_eq_vals):
            halos = halos[halos[:, col] == val]
        for col, val in zip(glob_ne_cols, glob_ne_vals):
            halos = halos[halos[:, col] != val]


    if sort_col != None:
        halos = sort_by_column(halos, sort_col)
    if (nhalos != None) or (nhalos != 0):
        halos = halos[:nhalos]
    #if (nhalos == 'perc25'):
    #    halos = halos[:len(halos)/10]
    if bad_halo_pairs != None:
        mask = np.arange(len(halos))
        mask = np.in1d(mask, bad_halo_pairs)
        mask = np.invert(mask)
        halos = halos[mask]

    c_rockstar_2lpt = halos[:, Rv1_col] / halos[:, Rs1_col]
    c_rockstar_za   = halos[:, Rv2_col] / halos[:, Rs2_col]
    if use_klypin:
        mask = (halos[:,4] < 100)
        print "changed␣%d␣halos" % (mask.sum())
        print "c_2lpt␣before␣", c_rockstar_2lpt[mask][0]
        c_rockstar_2lpt[mask] = halos[mask, Rv1_col] / halos[mask, 79]
        print "c_2lpt␣klypin␣", c_rockstar_2lpt[mask][0]
        mask = (halos[:,5] < 100)
        print "changed␣%d␣halos" % (mask.sum())
        print "c_za␣before␣", c_rockstar_za[mask][0]
        c_rockstar_za[mask] = halos[mask, Rv2_col] / halos[mask, 80]
        print "c_za␣klypin␣", c_rockstar_za[mask][0]
    c_diff_2lpt = 2.0 * (c_rockstar_2lpt - halos[:, c_lpt_col]) / (c_rockstar_2lpt + halos[:, c_lpt_col])
    c_diff_za   = 2.0 * (c_rockstar_za   - halos[:, c_za_col])  / (c_rockstar_za   + halos[:, c_za_col])
    #halos = np.column_stack((halos, c_rockstar_2lpt, c_rockstar_za, c_diff_2lpt, c_diff_za))
    #header.append('c_rockstar')
    #header.append('c_rockstar')
    #header.append('c_diff')
    #header.append('c_diff')

    c_diff_2lpt = c_diff_2lpt[np.isfinite(c_diff_2lpt)]
    c_diff_za = c_diff_za[np.isfinite(c_diff_za)]
    c_diff_tot = np.append(c_diff_2lpt, c_diff_za)

    c_diff_2lpt_frac = (np.abs(c_diff_za) <= cutoff_diff_frac).sum() / float(len(c_diff_2lpt))
    c_diff_za_frac = (np.abs(c_diff_za) <= cutoff_diff_frac).sum() / float(len(c_diff_za))
    c_diff_tot_frac = (np.abs(c_diff_tot) <= cutoff_diff_frac).sum() / float(len(c_diff_tot))

    with open(c_diff_file, 'w') as fd:
        fd.write("%g␣␣%g␣␣%g\n" % (c_diff_tot_frac, c_diff_za_frac, c_diff_2lpt_frac))

    print 'Finished␣snapshot.'


def read_files(files, header_line = None, comment_char = '#'):
    header = None
    data = None
    if type(files) == str:
        files = [files]

    if header_line != None:
        with open(files[0], 'r') as fd:
```

```
            for line in range(header_line):
                fd.readline()
            header = fd.readline()
        if header[0] != comment_char:
            print "Header must start with a '%s'" % comment_char
            sys.exit(4)
        header = header[1:]
        header = header.split()

    for file in files:
        print 'Reading file %s...' % (file)
        if data == None:
            data = np.genfromtxt(file, comments=comment_char)
        else:
            data = np.append(data, np.genfromtxt(file, comments=comment_char), axis=0)

    print 'Finished reading files.'
    if header_line == None:
        return data
    else:
        return header, data


def sort_by_column(halos, col):
    print 'Sorting halos...'
    mask = np.argsort(halos[:, col])
    mask = mask[::-1]
    halos = halos[mask]
    return halos



remove_nonfit_halos = False
global_filter_halos = True
use_klypin = False

nhalos = 100
#nhalos = 'perc25'
#sort_col = None
sort_col = 9

cutoff_diff_frac = 0.2


Rv1_col = 53
Rv2_col = 54
Rs1_col = 55
Rs2_col = 56

c_lpt_col = 17
c_za_col  = 18


lt_cols = [17, 18]
lt_vals = [100.0, 100.0]

gt_cols = [17, 18, 31, 32]
gt_vals = [1.0, 1.0, 0.0, 0.0]

eq_cols = []
eq_vals = []

ne_cols = []
ne_vals = []


# global filters
glob_lt_cols = []
glob_lt_vals = []

glob_gt_cols = [4, 5]
glob_gt_vals = [100, 100]

glob_eq_cols = [109, 110]
glob_eq_vals = [-1, -1]

glob_ne_cols = []
glob_ne_vals = []

bad_halo_pairs = None

c_diff_file = 'stats/c_diff.dat'



if __name__ == '__main__':
    main()
```

Appendix I

Differential Histogram Code

I.1 Histogram Generation and Fitting (Python)

```
#!/usr/bin/env python

import sys
import numpy as np
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from scipy import stats
from scipy.special import gamma as gamma_func
from scipy.optimize import curve_fit
import statsmodels.sandbox.distributions.extras as extrastats
from ipdb import set_trace

def main():
    # Read in particle files
    header, halos = read_files(sys.argv[1:], header_line = 3)

    if remove_nonfit_halos:
        print 'Removing NaNs...'
        halos = halos[np.isfinite(halos[:,c_lpt_col])]
        halos = halos[np.isfinite(halos[:,c_za_col])]

    if global_filter_halos:
        print 'Filtering data...'
        for col, val in zip(glob_lt_cols, glob_lt_vals):
            halos = halos[halos[:, col] <= val]
        for col, val in zip(glob_gt_cols, glob_gt_vals):
            halos = halos[halos[:, col] >= val]
        for col, val in zip(glob_eq_cols, glob_eq_vals):
            halos = halos[halos[:, col] == val]
        for col, val in zip(glob_ne_cols, glob_ne_vals):
            halos = halos[halos[:, col] != val]


    if sort_col != None:
        halos = sort_by_column(halos, sort_col)
    if (nhalos != None) or (nhalos != 0):
        halos = halos[:nhalos]
    if bad_halo_pairs != None:
        mask = np.arange(len(halos))
        mask = np.in1d(mask, bad_halo_pairs)
        mask = np.invert(mask)
        halos = halos[mask]

    c_rockstar_2lpt = halos[:, Rv1_col] / halos[:, Rs1_col]
    c_rockstar_za   = halos[:, Rv2_col] / halos[:, Rs2_col]
    if use_klypin:
        mask = (halos[:,4] < 100)
        print "changed %d halos" % (mask.sum())
        print "c_2lpt before ", c_rockstar_2lpt[mask][0]
        c_rockstar_2lpt[mask] = halos[mask, Rv1_col] / halos[mask, 79]
        print "c_2lpt klypin ", c_rockstar_2lpt[mask][0]
        mask = (halos[:,5] < 100)
        print "changed %d halos" % (mask.sum())
        print "c_za before ", c_rockstar_za[mask][0]
        c_rockstar_za[mask] = halos[mask, Rv2_col] / halos[mask, 80]
        print "c_za klypin ", c_rockstar_za[mask][0]
    c_diff_2lpt = 2.0 * (c_rockstar_2lpt - halos[:, c_lpt_col]) / (c_rockstar_2lpt + halos[:, c_lpt_col])
    c_diff_za   = 2.0 * (c_rockstar_za   - halos[:, c_za_col])  / (c_rockstar_za   + halos[:, c_za_col])
    halos = np.column_stack((halos, c_rockstar_2lpt, c_rockstar_za, c_diff_2lpt, c_diff_za))
    header.append('c_rockstar')
    header.append('c_rockstar')
    header.append('c_diff')
    header.append('c_diff')

    if mass_quartiles and len(halos) > 50:
        start_fracs = [0.0, 0.25, 0.50, 0.75, 0.0]
        end_fracs   = [0.25, 0.50, 0.75, 1.0, 1.0]
    else:
        start_fracs = [0.0]
        end_fracs   = [1.0]


    for start_frac, end_frac in zip(start_fracs, end_fracs):
        halos_to_pass = halos[start_frac * len(halos) : end_frac * len(halos)]
        if use_alt_frac and (start_frac == 0.0) and (end_frac == 1.0):
            alt_halos_to_pass = halos[alt_start_frac * len(halos) : alt_end_frac * len(halos)]
        else:
            alt_halos_to_pass = None
```

```
81          if len(halos_to_pass) > 0:
82              for (lpt_col, za_col, fancy_x_label) in zip(lpt_log_cols, za_log_cols, fancy_log_x_labels):
83                  make_plot(halos_to_pass, alt_halos_to_pass, lpt_col, za_col, start_frac, end_frac, fancy_x_label,
        header, use_log=True)
84              for (lpt_col, za_col, fancy_x_label) in zip(lpt_cols, za_cols, fancy_x_labels):
85                  make_plot(halos_to_pass, alt_halos_to_pass, lpt_col, za_col, start_frac, end_frac, fancy_x_label,
        header, use_log=False)
86
87      print 'Finished all plots.'
88
89
90  def read_files(files, header_line = None, comment_char = '#'):
91      header = None
92      data = None
93      if type(files) == str:
94          files = [files]
95
96      if header_line != None:
97          with open(files[0], 'r') as fd:
98              for line in range(header_line):
99                  fd.readline()
100             header = fd.readline()
101         if header[0] != comment_char:
102             print "Header must start with a '%s'" % comment_char
103             sys.exit(4)
104         header = header[1:]
105         header = header.split()
106
107     for file in files:
108         print 'Reading file %s...' % (file)
109         if data == None:
110             data = np.genfromtxt(file, comments=comment_char)
111         else:
112             data = np.append(data, np.genfromtxt(file, comments=comment_char), axis=0)
113
114     print 'Finished reading files.'
115     if header_line == None:
116         return data
117     else:
118         return header, data
119
120
121 def sort_by_column(halos, col):
122     print 'Sorting halos...'
123     mask = np.argsort(halos[:, col])
124     mask = mask[::-1]
125     halos = halos[mask]
126     return halos
127
128
129 def make_plot(halos, alt_halos, lpt_col, za_col, start_frac, end_frac, fancy_x_label, header=None, use_log=False)
        :
130     print 'start =', start_frac
131     print 'end =', end_frac
132     x_lpt = halos[:, lpt_col]
133     x_za  = halos[:, za_col]
134     x_lpt, x_za = filter(x_lpt, x_za, lpt_col, za_col)
135
136     if alt_halos != None:
137         alt_x_lpt = alt_halos[:, lpt_col]
138         alt_x_za  = alt_halos[:, za_col]
139         alt_x_lpt, alt_x_za = filter(alt_x_lpt, alt_x_za, lpt_col, za_col)
140
141     if header != None:
142         header_lpt = header[lpt_col]
143         header_za  = header[za_col]
144         if header_lpt == header_za:
145             xlabel = header_lpt
146             xlabel = xlabel.replace('/', '_over_')
147         else:
148             print 'column mismatch... exiting'
149             set_trace()
150             sys.exit(123)
151
152     if len(x_lpt) == 0 or len(x_za) == 0:
153         print "Skipping range %f - %f for %s plot.  No halos found." % (start_frac, end_frac, xlabel)
154         return
155         #set_trace()
156
157     if perc_diff:
158         print 'Finding percent difference stats...'
159         x_perc_diff = (x_lpt - x_za) / x_za
160         perc_diff_file = "%s%s%0.3d%s%0.3d%s%s_(%s-%s)%s" % \
161                          (perc_diff_base, '(', lpt_col, ',', za_col, ')_', xlabel, start_frac, end_frac,
       stats_ext)
162         perc_diff_stats(x_perc_diff, perc_diff_file, use_log=use_log)
163         print 'done.'
164
165     x = 2.0 * (x_lpt - x_za) / (x_lpt + x_za)
166     x[np.logical_and(x_lpt == 0, x_za == 0)] = 0
167
168     if alt_halos != None:
```

```
        alt_x = 2.0 * (alt_x_lpt - alt_x_za) / (alt_x_lpt + alt_x_za)
        alt_x[np.logical_and(alt_x_lpt == 0, alt_x_za == 0)] = 0

#   set_trace()

    if x_lim == None:
        #x_max = max(abs(x.max()), abs(x.min()))
        if lpt_col == 47:
            x_max = x.mean() + x.std() * 1.5
            x_min = x.mean() - x.std() * 1.5
        else:
            x_max = np.std(x) * 3.0
            x_min = -x_max
    else:
        x_max = x_lim
        x_min = -x_lim

    # get stats
    data_mean = x.mean()
    data_stdev = x.std()**2
    data_skew = stats.skew(x)
    data_kurt = stats.kurtosis(x)
    data_rms = np.sqrt(np.mean(x**2))
    data_gt_epsilon = float(len(x[np.abs(x) >= 0.1])) / float(len(x))

    # Generate plot
    print 'generating', xlabel, 'plot...'
    fig = plt.figure(figsize=(9.0, 6.0))
    if add_residuals_panel:
        grid = gridspec.GridSpec(2, 1, height_ratios=[1,4])
        ax = fig.add_subplot(grid[1])
    else:
        ax = fig.add_subplot(111)
    ax, n, bins, patches = draw_hist(ax, x, x_min=x_min, x_max=x_max, \
                                     use_log=use_log, color='blue', fill=None)

    p0 = [1.0, data_mean, data_stdev, 2.0]
    ax, fit_height, fit_mean, fit_stdev, fit_skew, fit_kurt, fit_height_err, fit_mean_err, fit_stdev_err,
     fit_skew_err, fit_kurt_err, chi2, pval = draw_fit(ax, n, bins, p0)

    if draw_data_fit:
        ax = draw_data_gaussian(ax, x, n, bins)

    if alt_halos != None:
        ax, n_alt, bins_alt, patches_alt = draw_hist(ax, alt_x, x_min=x_min, x_max=x_max, \
                                                     use_log=use_log, color='green', fill="0.75")
        #ax = draw_fit(ax, n, bins)

    #ax.grid(color='gray', linestyle='dashed')
    ax.set_xlim([x_min, x_max])
    #ax.set_xlabel('(' + xlabel + '_2lpt - ' + xlabel + '_za) / ' + xlabel + '_avg')
    #ax.set_ylabel(ylabel)
    if label_axes:
        ax.set_xlabel(fancy_x_label, fontsize="xx-large")
        ax.set_ylabel(fancy_y_label, fontsize="xx-large")
    #ax.legend()

    if add_residuals_panel:
        ax = fig.add_subplot(grid[0])
        ax = draw_residuals(ax, n, bins, fit_height, fit_mean, fit_stdev, fit_kurt)
        ax.tick_params(axis='x', labelbottom='off')

    fig.tight_layout()
    plot_name = "%s%s%0.3d%s%0.3d%s%s_(%s-%s)%s" % \
                (plot_base, '(', lpt_col, ',', za_col, ')_', xlabel, start_frac, end_frac, plot_ext)
    fig.savefig(plot_name, bbox_inches='tight')

    if save_stats:
        statsfile = "%s%s%0.3d%s%0.3d%s%s_(%s-%s)%s" % \
                    (stats_base, '(', lpt_col, ',', za_col, ')_', xlabel, start_frac, end_frac, stats_ext)
        with open(statsfile, 'w') as fd:
            if bin_test:
                for ntestbins in range(nbins_min, nbins_max+1, 5):
                    fit_mean, fit_stdev = rebin_stats(ntestbins, x, x_min=x_min, x_max=x_max, use_log=use_log)
                    fd.write("%d␣%g␣%g␣%g␣%g\n" % (ntestbins, data_mean, data_stdev, fit_mean, fit_stdev))
            else:
                fd.write("%d␣␣␣%g␣%g␣%g␣%g␣␣␣%g␣%g␣%g␣%g␣%g␣%g␣%g␣%g␣%g␣%g␣␣␣%g␣%g␣␣␣%g␣%g\n" % \
                         (nbins, data_mean, data_stdev, data_skew, data_kurt, \
                          fit_height, fit_height_err, fit_mean, fit_mean_err, fit_stdev, fit_stdev_err, fit_skew,
      fit_skew_err, fit_kurt, fit_kurt_err, \
                          data_rms, data_gt_epsilon, chi2, pval))

    print 'finished␣plot␣' + plot_name
    return


def perc_diff_stats(x, filename, use_log=False):
    data_mean = x.mean()
    data_stdev = x.std()**2
    data_skew = stats.skew(x)
    data_kurt = stats.kurtosis(x)
    data_rms = np.sqrt(np.mean(x**2))
```

```
    data_gt_epsilon = float(len(x[np.abs(x) >= 0.1])) / float(len(x))

    if x_lim == None:
        x_max = min((x.mean() + x.std() * 3.0), x.max())
        x_min = max((x.mean() - x.std() * 3.0), x.min())
    else:
        x_max = x_lim
        x_min = -x_max

    global nbins
    if nbins <= 0:
        nbins = np.sqrt(len(x))
        if nbins % 2 == 0:
            nbins = nbins - 1
    if nbins < nbins_min:
        nbins = nbins_min
    elif nbins > nbins_max:
        nbins = nbins_max

    if use_log:
        xbins = np.logspace(np.log10(x_min), np.log10(x_max), num=nbins+1)
        mid_bins = 10.0**(0.5 * (np.log10(xbins[1:]) + np.log10(xbins[:-1])))
    else:
        xbins = np.linspace(x_min, x_max, num=nbins+1)
        mid_bins = 0.5 * (xbins[1:] + xbins[:-1])

    hist, bin_edges = np.histogram(x, bins=xbins)
    x_peak = mid_bins[hist == hist.max()][0]

    x_sorted = np.sort(x)
    n_halos = len(x_sorted)

    x_vals = []
    for frac in fractions:
        x_vals.append(x_sorted[len(x_sorted)*frac])
    x_vals = np.array(x_vals)

    sum_frac_halos = []
    for diff_val in diff_vals:
        n_gt_val = (x_sorted >= diff_val).sum()
        sum_frac_halos.append(float(n_gt_val) / float(n_halos))
    sum_frac_halos = np.array(sum_frac_halos)

    doublesum_frac_halos = []
    for right_diff_val in diff_vals:
        left_diff_val = (1.0 / (right_diff_val + 1.0)) - 1.0
        n_gt_val = (x_sorted >= right_diff_val).sum() + (x_sorted <= left_diff_val).sum()
        doublesum_frac_halos.append(float(n_gt_val) / float(n_halos))
    doublesum_frac_halos = np.array(doublesum_frac_halos)

    with open(filename, 'w') as fd:
        fd.write("%d␣␣␣%g␣␣␣%s␣␣␣%s␣␣␣%g␣%g␣%g␣%g␣␣␣%g␣%g\n" % \
                 (nbins, x_peak, \
                  '␣'.join("%g" % x for x in x_vals), \
                  '␣'.join("%g" % x for x in sum_frac_halos), \
                  '␣'.join("%g" % x for x in doublesum_frac_halos), \
                  data_mean, data_stdev, data_skew, data_kurt, \
                  data_rms, data_gt_epsilon))

    return


def find_frac_bounds(hist, start_bin, frac):
    n_tot = hist.sum()
    n_sum = hist[start_bin]

    left_tot = hist[:start_bin].sum() + hist[start_bin]/2.0
    right_tot = hist[start_bin+1:].sum() + hist[start_bin]/2.0

    if float(left_tot) / float(n_tot) <= frac / 2.0:
        right_only = True
    if float(right_tot) / float(n_tot) <= frac / 2.0:
        left_only = True

    left_bound = start_bin
    right_bound = start_bin
    while(float(n_sum) / float(n_tot) < frac):

        pass

    return left_bound, right_bound


def filter(x_lpt, x_za, lpt_col, za_col):
    mask = np.isfinite(x_lpt)
    x_lpt = x_lpt[mask]
    x_za  = x_za[mask]
    mask = np.isfinite(x_za)
    x_lpt = x_lpt[mask]
    x_za  = x_za[mask]

    if column_filter_halos:
```

```
351         x_lpt, x_za = filter_columns(lpt_col, x_lpt, x_za)
352         x_za, x_lpt = filter_columns(za_col, x_za, x_lpt)
353 
354     return x_lpt, x_za
355 
356 
357 def filter_columns(x_col, x1, x2):
358     print 'Filtering␣data...'
359 
360     mask = np.isfinite(x1)
361     x1 = x1[mask]
362     x2 = x2[mask]
363 
364     mask = (x1 != -9999)
365     x1 = x1[mask]
366     x2 = x2[mask]
367 
368     if x_col in lt_cols:
369         val = lt_vals[lt_cols.index(x_col)]
370         mask = (x1 <= val)
371         x1 = x1[mask]
372         x2 = x2[mask]
373     if x_col in gt_cols:
374         val = gt_vals[gt_cols.index(x_col)]
375         mask = (x1 >= val)
376         x1 = x1[mask]
377         x2 = x2[mask]
378     if x_col in eq_cols:
379         val = eq_vals[eq_cols.index(x_col)]
380         mask = (x1 == val)
381         x1 = x1[mask]
382         x2 = x2[mask]
383     if x_col in ne_cols:
384         val = ne_vals[ne_cols.index(x_col)]
385         mask = (x1 != val)
386         x1 = x1[mask]
387         x2 = x2[mask]
388     return x1, x2
389 
390 
391 def draw_hist(ax, x, x_min=None, x_max=None, use_log=False, color=None, fill=None, label=None):
392     global nbins
393     if nbins <= 0:
394         nbins = np.sqrt(len(x))
395         if nbins % 2 == 0:
396             nbins = nbins - 1
397     if nbins < nbins_min:
398         nbins = nbins_min
399     elif nbins > nbins_max:
400         nbins = nbins_max
401 
402     if use_log:
403         xbins = np.logspace(np.log10(x_min), np.log10(x_max), num=nbins+1)
404         ax.set_xscale('log')
405     else:
406         xbins = np.linspace(x_min, x_max, num=nbins+1)
407 
408     if fill == None:
409         type='step'
410     else:
411         type='stepfilled'
412 
413     n, bins, patches = ax.hist(x, bins=xbins, histtype=type, facecolor=fill, normed=hist_normed, cumulative=
        hist_cumulative, log=ylog, edgecolor=color, label=label)
414     return ax, n, bins, patches
415 
416 
417 def draw_fit(ax, hist, bin_edges, p0):
418     bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2.0
419 
420     if ignore_central_bin:
421         mask = (np.abs(bin_centers) > 0.000001)
422         bin_centers = bin_centers[mask]
423         hist = hist[mask]
424 
425     hist[hist==0] = 1  #fix devide by zero error
426 
427     try:
428         if poisson_weight:
429             sigma=np.sqrt(hist)/hist
430             sigma = sigma / float(hist.max())
431         else:
432             sigma=None
433 
434         if fit_in_log:
435             #if sigma != None:
436             #   sigma = np.log10(sigma)
437 
438             coeffs, var_matrix = curve_fit(log_generalized_normal, bin_centers, np.log10(hist/float(hist.max())),
        p0=p0, sigma=sigma)
439 
440             coeffs[0] = coeffs[0]**2
```

```
441             var_matrix[0,0] = var_matrix[0,0]**2
442         else:
443             coeffs, var_matrix = curve_fit(generalized_normal, bin_centers, hist/float(hist.max()), p0=p0, sigma=
        sigma)
444 
445         if prevent_small_shape_param and coeffs[3] < 1.0:
446             coeffs[3] = 1.0 / coeffs[3]
447         print 'coeffs␣=', coeffs
448 
449     except RuntimeError:
450         print '*******curve_fit␣failed!'
451         return ax, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan
452 
453     height, mean, stdv, skew, kurt = coeffs[0] * hist.max(), coeffs[1], coeffs[2], 0.0, coeffs[3]
454     height_err, mean_err, stdv_err, skew_err, kurt_err = np.sqrt(var_matrix[0,0]*hist.max()), np.sqrt(var_matrix
        [1,1]), np.sqrt(var_matrix[2,2]), 0.0, np.sqrt(var_matrix[3,3])
455 
456     fit_x = np.linspace(bin_edges[0], bin_edges[-1], nfitpoints+1)
457     hist_fit = generalized_normal(fit_x, height, mean, stdv, kurt)
458     ax.plot(fit_x, hist_fit, color='red', linestyle='--')
459 
460     chi2_fit = generalized_normal(bin_centers, height, mean, stdv, kurt)
461     chi2, pval = stats.chisquare(hist / hist.max(), chi2_fit / hist.max())
462 
463     return ax, height, mean, stdv, skew, kurt, height_err, mean_err, stdv_err, skew_err, kurt_err, chi2, pval
464 
465 
466 def draw_residuals(ax, hist, bin_edges, fit_height, fit_mean, fit_stdev, fit_kurt):
467     bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2.0
468     fit = generalized_normal(bin_centers, fit_height, fit_mean, fit_stdev, fit_kurt)
469     ratio = (hist - fit) / hist.max()
470     #ax.plot(bin_centers, ratio, linestyle='steps-mid-')
471     ax.plot(bin_centers, ratio, linestyle='steps-mid-')
472     return ax
473 
474 
475 def draw_data_gaussian(ax, x, hist, bins):
476     bin_centers = (bins[:-1] + bins[1:]) / 2.0
477     x_min = bins[0]
478     x_max = bins[-1]
479 
480     mean = np.mean(x)
481     stdv = np.std(x)**2
482     skew = stats.skew(x)
483     kurt = stats.kurtosis(x)
484 
485     print "data␣stats:␣␣mean␣=␣%g␣␣stdv␣=␣%g␣␣skew␣=␣%g␣␣kurt␣=␣%g" % (mean, stdv, skew, kurt)
486 
487     coeffs, var_matrix = curve_fit(gaussian_height(mean, stdv, skew, kurt), bin_centers, hist, p0=[hist.max()])
488     height = coeffs[0]
489 
490     fit_x = np.linspace(x_min, x_max, nfitpoints+1)
491     hist_fit = gaussian(fit_x, height, mean, stdv, skew, kurt)
492     ax.plot(fit_x, hist_fit, color='0.25', linestyle='-.')
493     return ax
494 
495 
496 #def gaussian(x, A, mu, sigma, skew, kurtosis):
497 #    pdf_function = extrastats.pdf_mvsk([mu, sigma, skew, kurtosis])
498 #    return A * pdf_function(x)
499 
500 
501 def double_gaussian(x, A, mu, sigma, skew, kurtosis, A2, mu2, sigma2, skew2, kurtosis2):
502     return gaussian(x, A, mu, sigma, skew, kurtosis) + gaussian(x, A2, mu2, sigma2, skew2, kurtosis2)
503 
504 
505 def gaussian_height(mu, sigma, skew, kurtosis):
506     def func(x, A):
507         pdf_function = extrastats.pdf_mvsk([mu, sigma, skew, kurtosis])
508         return A * pdf_function(x)
509     return func
510 
511 
512 #def log_gaussian(x, A, mu, sigma, skew=0.0, kurtosis=0.0):
513 def log_gaussian(x, A, mu, sigma):
514     A = A**2  # remember to also square fit value for A
515     y = gaussian(x, A, mu, sigma)
516     #y = gaussian(x, A, mu, sigma, skew, kurtosis)
517     if (y <= 0.0).any():
518         #y[y<=0] = -y[y<=0] + 1
519         y[y<=0] = (y[y<=0] + 0.0001)**2
520     return np.log10(y)
521 
522 
523 #def log_double_gaussian(x, A1, mu1, sigma1, skew1, kurtosis1, A2, sigma2, skew2, kurtosis2):    # for common
        mean
524 #def log_double_gaussian(x, A1, mu1, sigma1, skew1, kurtosis1, A2, mu2, sigma2, skew2, kurtosis2):
525 def log_double_gaussian(x, A1, mu1, sigma1, A2, mu2, sigma2):
526     #mu2 = mu1  # for common mean
527     A1 = A1**2  # remember to also square fit value for A
528     A2 = A2**2
529     skew1 = 0.0
```

```
    skew2 = 0.0
    kurtosis1 = 0.0
    kurtosis2 = 0.0
    y = double_gaussian(x, A1, mu1, sigma1, skew1, kurtosis1, A2, mu2, sigma2, skew2, kurtosis2)
    if (y <= 0.0).any():
        #y[y<=0] = -y[y<=0] + 1
        y[y<=0] = (y[y<=0] + 0.0001)**2
    return np.log10(y)


def gaussian(x, A, mu, sigma):
    return A * np.exp(-(x - mu)**2 / (2.0 * sigma**2))


def generalized_normal(x, A, mu, alpha, beta):
    if prevent_small_shape_param and beta < 1.0:
        beta = 1.0 / beta
    return A * ( beta / (2.0 * alpha * gamma_func(1.0 / beta)) ) * np.exp(-(np.abs(x - mu)/alpha)**beta)


def log_generalized_normal(x, A, mu, alpha, beta):
    A = A**2
    y = generalized_normal(x, A, mu, alpha, beta)
    if (y <= 0.0).any():
        #y[y<=0] = -y[y<=0] + 1.0
        y[y<=0] = (y[y<=0] + 0.0001)**2
    return np.log10(y)


def add_text(fig, ax, textstr):
    #props = dict(boxstyle='round', facecolor='white', alpha=0.25)
    props = dict(edgecolor='none', facecolor='none')
    ax.text(0.02, 0.16, textstr, transform=ax.transAxes, fontsize=14, \
            verticalalignment='top', bbox=props)
    return fig, ax


def rebin_stats(ntestbins, x, x_min=None, x_max=None, use_log=False):
    if use_log:
        xbins = np.logspace(np.log10(x_min), np.log10(x_max), num=ntestbins+1)
    else:
        xbins = np.linspace(x_min, x_max, num=ntestbins+1)

    hist, bin_edges = np.histogram(x, bins=xbins)

    bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2.0
    if ignore_central_bin:
        mask = (np.abs(bin_centers) > 0.000001)
        bin_centers = bin_centers[mask]
        hist = hist[mask]
    #p0 = [hist.max(), 0.0, 0.2]
    p0 = [hist.max(), hist.mean(), hist.std(), stats.skew(hist), stats.kurtosis(hist)]
    hist[hist==0] = 1  #fix devide by zero error
    try:
        if poisson_weight:
            coeffs, var_matrix = curve_fit(gaussian, bin_centers, hist, p0=p0, sigma=(np.sqrt(hist)/hist))
        else:
            coeffs, var_matrix = curve_fit(gaussian, bin_centers, hist, p0=p0)
    except RuntimeError:
        print '*******curve_fit failed!'
        return np.nan, np.nan

    mean, stdev = coeffs[1], coeffs[2]
    return mean, stdev


nbins = 35
#nbins = 25
#nbins = -1
nbins_min = 15
nbins_max = 200
#nbins_max = 200
nfitpoints = 100
remove_nonfit_halos = False
global_filter_halos = True
column_filter_halos = True
use_klypin = False
label_axes = True
ignore_central_bin = False
save_stats = True
bin_test = False
poisson_weight = True
fit_in_log = True
draw_data_fit = False
mass_quartiles = False
prevent_small_shape_param = False
add_residuals_panel = False
perc_diff = True

hist_normed = False
hist_cumulative = False
ylog = False
```

```
ylabel= 'Number of Halos'

#                                      v           v           v
fractions = [0.01, 0.05, 0.10, 0.25, 0.50, 0.75, 0.90, 0.95, 0.99]
diff_vals = [0.01, 0.05, 0.10, 0.25, 0.50, 0.75, 1.00, 2.00, 4.00]
#                           ^           ^           ^           ^

#nhalos = 100
nhalos = None
sort_col = 9

#lpt_log_cols  = [ 9, 23, 31, 47,  51, 57]
#za_log_cols   = [10, 24, 32, 48,  52, 58]
#lpt_cols      = [17, 77, 91, 93, 97,  99, 107, 111, -4, -2]
#za_cols       = [18, 78, 92, 94, 98, 100, 108, 112, -3, -1]

lpt_log_cols  = []
za_log_cols   = []
#lpt_cols      = [-4, 47, 91, 107, 111]
#za_cols       = [-3, 48, 92, 108, 112]
#lpt_cols      = [-4, 31, 47, 91, 107, 111]
#za_cols       = [-3, 32, 48, 92, 108, 112]
#lpt_cols      = [-4, 31, 47, 91, 111]
#za_cols       = [-3, 32, 48, 92, 112]
lpt_cols      = [-4, 47, 91, 93, 107]
za_cols       = [-3, 48, 92, 94, 108]
# conentration, mass, x_off, v_off, T/|U|

fancy_log_x_labels = []
#fancy_x_labels = [r"$\mathrm{\frac{c_{2LPT} - c_{ZA}}{c_{avg}}}$",
#                  r"$\mathrm{\frac{\rho_{0, 2LPT} - \rho_{0, ZA}}{\rho_{0, avg}}}$",
#                  r"$\mathrm{\frac{M_{vir, 2LPT} - M_{vir, ZA}}{M_{vir, avg}}}$",
#                  r"$\mathrm{\frac{X_{off, 2LPT} - X_{off, ZA}}{X_{off, avg}}}$",
#                  r"$\mathrm{\frac{N_{subs, 2LPT} - N_{subs, ZA}}{N_{subs, avg}}}$"]

fancy_x_labels = [r"$\mathrm{\frac{c_{2LPT} - c_{ZA}}{c_{avg}}}$",
                  r"$\mathrm{\frac{M_{vir, 2LPT} - M_{vir, ZA}}{M_{vir, avg}}}$",
                  r"$\mathrm{\frac{X_{off, 2LPT} - X_{off, ZA}}{X_{off, avg}}}$",
                  r"$\mathrm{\frac{V_{off, 2LPT} - V_{off, ZA}}{V_{off, avg}}}$",
                  r"$\mathrm{\frac{(T/|U|)_{2LPT} - (T/|U|)_{ZA}}{(T/|U|)_{avg}}}$"]

fancy_y_label  = r"$\mathrm{N_{halos}}$"

Rv1_col = 53
Rv2_col = 54
Rs1_col = 55
Rs2_col = 56

c_lpt_col = 17
c_za_col  = 18


# c_2lpt, c_za, chi2_2lpt, chi2_za
#lt_cols = [17, 18, 37, 38]
#lt_vals = [100.0, 100.0, 10.0, 10.0]
lt_cols = [17, 18]
lt_vals = [100.0, 100.0]

# c_2lpt, c_za, rho_0_2lpt, rho_0_za, chi2_2lpt, chi2_za
#gt_cols = [17, 18, 31, 32, 37, 38]
#gt_vals = [1.0, 1.0, 0.0, 0.0, 0.0, 0.0]
gt_cols = [17, 18, 31, 32]
gt_vals = [1.0, 1.0, 0.0, 0.0]

eq_cols = []
eq_vals = []

ne_cols = []
ne_vals = []


# global filters
glob_lt_cols = []
glob_lt_vals = []

glob_gt_cols = [4, 5]
glob_gt_vals = [100, 100]

glob_eq_cols = [109, 110]
glob_eq_vals = [-1, -1]

glob_ne_cols = []
glob_ne_vals = []



use_alt_frac = True
alt_start_frac = 0.75
alt_end_frac = 1.0

#x_lim = 0.5
#x_lim = 1.0
```

```
x_lim = None

bad_halo_pairs = None
#bad_halo_pairs = [9, 28, 39, 51, 59, 95]

perc_diff_base = 'plots/perc_diff_'
#statsfile = 'plots/stats.txt'
stats_base = 'plots/stats_'
stats_ext  = '.txt'
plot_base = 'plots/hist_'
plot_ext  = '.eps'

plot_dest_type = 'paper'
if plot_dest_type == 'paper':
    mpl.rcParams['font.family'] = 'serif'
    mpl.rcParams['font.size'] = 16
    mpl.rcParams['axes.linewidth'] = 3
    mpl.rcParams['lines.linewidth'] = 4
    mpl.rcParams['patch.linewidth'] = 4
    mpl.rcParams['xtick.major.width'] = 3
    mpl.rcParams['ytick.major.width'] = 3
    mpl.rcParams['xtick.major.size'] = 8
    mpl.rcParams['ytick.major.size'] = 8


if __name__ == '__main__':
    main()
```

I.2 PBS Submission Script (Bash)

```
#!/usr/bin/env bash
#PBS -M djsissom@gmail.com
#PBS -m bae
#PBS -l nodes=1:ppn=1
#PBS -l pmem=40000mb
#PBS -l mem=4000mb
#PBS -l walltime=1:00:00
#PBS -o out.log
#PBS -j oe

minsnap=0
maxsnap=61

# Change to working directory
echo $PBS_NODEFILE
cd $PBS_O_WORKDIR

for ((snap=$minsnap; snap<=$maxsnap; snap++)); do

    if [ $snap -lt 10 ]; then
        j=00$snap
    elif [ $snap -lt 100 ]; then
        j=0$snap
    fi

    new_plot_dir=snap${j}_plots

    if [ ! -e plots_all_snaps/${new_plot_dir} ]; then
        mkdir plots_all_snaps/${new_plot_dir}
    fi

    echo "Starting box${i} snap${j}..."
    ./hist.py ~/projects/simulations/rockstar/box{1,2,3}/crossmatch/snap${j}/halos.dat > plots/out.log 2>&1
    mv plots/* plots_all_snaps/${new_plot_dir}/.
    echo "Finished snap${j}"

done

wait

# - end of script
```

I.3 PBS Submission Script - Individual Boxes (Bash)

```
#!/usr/bin/env bash
#PBS -M djsissom@gmail.com
#PBS -m bae
#PBS -l nodes=1:ppn=1
#PBS -l pmem=40000mb
#PBS -l mem=4000mb
#PBS -l walltime=2:00:00
#PBS -o out.log
#PBS -j oe

minsnap=0
maxsnap=61

minbox=1
maxbox=3

```

```
# Change to working directory
echo $PBS_NODEFILE
cd $PBS_O_WORKDIR


for ((box=$minbox; box<=$maxbox; box++)); do

    new_box_dir=plots_all_snaps_box${box}
    if [ ! -e ${new_box_dir} ]; then
        mkdir ${new_box_dir}
    fi

    for ((snap=$minsnap; snap<=$maxsnap; snap++)); do

        if [ $snap -lt 10 ]; then
            j=00$snap
        elif [ $snap -lt 100 ]; then
            j=0$snap
        fi

        new_plot_dir=snap${j}_plots

        if [ ! -e ${new_box_dir}/${new_plot_dir} ]; then
            mkdir ${new_box_dir}/${new_plot_dir}
        fi

        echo -n "Starting box${box} snap${j}... "
        ./hist.py ~/projects/simulations/rockstar/box${box}/crossmatch/snap${j}/halos.dat > plots/out.log 2>&1
        mv plots/* ${new_box_dir}/${new_plot_dir}/.
        echo "Finished snap${j}"

    done
done

# - end of script
```

I.4 Statistics Collection Script (Bash)

```
#!/usr/bin/env bash

if [ "$#" -ne 1 ]; then
    echo "Please provide a directory as an argument."
    exit -1
fi

parent_dir=$1

for stats_path in $parent_dir/snap061_plots/{stats_*,perc_diff_*}; do
    stats_file=$(basename "$stats_path")
    out_file=${stats_file/_\(/_allsnaps_\(}
    echo "Merging stats for $stats_file..."

    for snap_dir in $parent_dir/snap*_plots; do
        if [ -e $snap_dir/$stats_file ]; then
            snap_num=$(basename "$snap_dir")
            echo -n "${snap_num:5:2}  "
            cat $snap_dir/$stats_file | cut -d' ' -f 2-
        fi
    done | column -t > $parent_dir/$out_file

    echo "Stats written to $out_file..."

done
```

Appendix J

Redshift Trends Code (Python)

```
#!/usr/bin/env python

import sys
import os
import numpy as np
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
from scipy.special import gamma as Gamma
from scipy.special import psi as digamma
from ipdb import set_trace


def main():
    #for filenum, file in enumerate(sys.argv[1:]):
    if (len(sys.argv[1:]) == 4):
        data1 = read_files(sys.argv[1], header_line = None)
        data2 = read_files(sys.argv[2], header_line = None)
        data3 = read_files(sys.argv[3], header_line = None)
        rsnap_data = read_files(sys.argv[4], header_line = None)
    else:
        print 'need 4 files'
        sys.exit(15)

    if fit_mean_trend:
        with open(statsfile, 'w') as fd:
            fd.write("#plot slope slope_err intercept intercept_err\n")

    if skew_err_boxes:
        skew_err1 = get_skew_err(sys.argv[1])
        skew_err2 = get_skew_err(sys.argv[2])
        skew_err3 = get_skew_err(sys.argv[3])

    if minsnap > 0:
        #for data in data1, data2, data3:
        #   data = data[data[:,0] >= minsnap]
        data1 = data1[data1[:,0] >= minsnap]
        data2 = data2[data2[:,0] >= minsnap]
        data3 = data3[data3[:,0] >= minsnap]
        if skew_err_boxes:
            skew_err1 = skew_err1[-len(data1):]
            skew_err2 = skew_err2[-len(data2):]
            skew_err3 = skew_err3[-len(data3):]

    if skew_err_col == -2:
        data1 = np.column_stack((data1, skew_err1))
        data2 = np.column_stack((data2, skew_err2))
        data3 = np.column_stack((data3, skew_err3))

    #if (mean_err_col == -2) or (var_err_col == -2) or (skew_err_col == -2) or (kurt_err_col == -2):
    #   fake_err = np.zeros(len(data1))
    #   data1 = np.column_stack((data1, fake_err))
    #   data2 = np.column_stack((data2, fake_err))
    #   data3 = np.column_stack((data3, fake_err))

    z = 1.0 / rsnap_data[:,1] - 1.0
    if (len(data1) == len(data2)) and (len(data1) == len(data3)):
        z = z[-len(data1):]
    else:
        sys.exit(16)

    data1 = np.column_stack((data1, z))
    data2 = np.column_stack((data2, z))
    data3 = np.column_stack((data3, z))

    #data1[:,-1] = data1[:,-1] - 0.12
    #data2[:,-1] = data2[:,-1] + 0.12

    for data in [data1, data2, data3]:
        if expand_error:
            mask = (np.abs(data[:,data_mean_col] - data[:,mean_col]) > data[:,mean_err_col])
            data[mask,mean_err_col] = np.abs(data[mask,data_mean_col] - data[mask,mean_col])
        if transform_variance:
            data[:,var_col] = data[:,var_col]**2 * Gamma(3.0 / data[:,beta_col]) / Gamma(1.0 / data[:,beta_col])
            data[:,var_err_col] = data[:,var_err_col]**2 * Gamma(3.0 / data[:,beta_col]) / Gamma(1.0 / data[:,
 beta_col])
        if transform_kurtosis:
            #data[:,kurt_col] = ( Gamma(5.0 / data[:,kurt_col]) * Gamma(1.0 / data[:,kurt_col]) / Gamma(3.0 /
 data[:,kurt_col]) ) - 3.0
            beta = data[:,beta_col]
            beta_err = data[:,beta_err_col]
            kurtosis = ( Gamma(5.0 / beta) * Gamma(1.0 / beta) / Gamma(3.0 / beta) ) - 3.0
```

```
            kurtosis_err = beta_err * (1.0 / beta**2) * (kurtosis + 3) * (6.0 * digamma(3.0/beta) - 5.0 * digamma(5.0/beta) - digamma(1.0/beta))

            data[:,kurt_col] = kurtosis[:]
            data[:,kurt_err_col] = kurtosis_err[:]

        data[:,var_col] = np.sqrt(data[:,var_col])       # var to stdev
        data[:,var_err_col] = np.sqrt(data[:,var_err_col])       # var to stdev


    if save_transformed_data:
        for data, path in zip([data1, data2, data3], sys.argv[1:4]):
            fname = transform_file_base + os.path.basename(path)
            with open(fname, 'w') as fd:
                fd.write(transformed_data_header)
                np.savetxt(fd, np.column_stack((z, data)), fmt='%g')



    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
    # make mean and stdv plots                                                   #
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#

    for (data, ylabel, color, label, name) in zip([data1, data2, data3], ylabels1, colors, labels1, names):
        print "Making %s plot..." % (name)
        fig = plt.figure(figsize=(9.0, 6.0))
        ax = fig.add_subplot(111)

        ax = make_plot(ax, data[:,z_col], data[:,mean_col], err = data[:,mean_err_col], color = 'blue', marker='o', label=label)
        ax = make_plot(ax, data[:,z_col], data[:,mean_col] + data[:,var_col], color = 'black', linestyle='--')
        ax = make_plot(ax, data[:,z_col], data[:,mean_col] - data[:,var_col], color = 'black', linestyle='--')

        if add_rms_line:
            ax = make_plot(ax, data[:,z_col], data[:,data_rms_col], color = 'green', linestyle=':')

        if fit_mean_trend:
            ax, slope, slope_err, intercept, intercept_err = add_fit(ax, data[:,z_col], data[:,mean_col], err=data[:,mean_err_col], color='red')
            save_fits(statsfile, name, slope, np.sqrt(slope_err), intercept, np.sqrt(intercept_err))

        #ax.legend(loc='lower right')
        ax.set_xlim(z[0] + 1.0, z[-1] - 1.0)
        #ax.invert_xaxis()

        ax.set_xlabel(xlabel, fontsize='x-large')
        ax.set_ylabel(ylabel, fontsize='x-large')

        fig.tight_layout()
        fig.savefig(plot_base + 'mean_stdev_' + name + plot_ext, bbox_inches='tight')

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
    # make skew and kurtosis plots                                               #
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#

    #data1[:,-1] = data1[:,-1] + 0.12
    #data2[:,-1] = data2[:,-1] - 0.12


    for (data, ylabel_kurt, ylabel_skew, color, name, ylim_low1, ylim_high1, ylim_low2, ylim_high2) in zip([data1, data2, data3], ylabels2_kurt, ylabels2_skew, colors, names, [-10.0, -10.0, -1.0], [20.0, 20.0, 1.5], [-0.2, -1.5, -0.4], [0.5, 3.5, 0.1]):
        print "Making %s plot..." % (name)
        fig = plt.figure(figsize=(9.0, 6.0))
        ax = fig.add_subplot(111)

        #ax = make_plot(ax, data[:,z_col] - offset, data[:,kurt_col], err = data[:,kurt_err_col], color = 'red', marker='o', linestyle='-', label='Kurtosis')
        #ax = make_plot(ax, data[:,z_col] + offset, data[:,skew_col], err = data[:,skew_err_col], color = 'blue', marker='o', linestyle='-', label='Skew')
        ax = make_plot(ax, data[:,z_col] - offset, data[:,kurt_col], err = data[:,kurt_err_col], color = 'red', marker='o', linestyle=':', label='Kurtosis')
        legend_lines1, legend_labels1 = ax.get_legend_handles_labels()

        ax.set_xlabel(xlabel, fontsize='x-large')
        ax.set_ylabel(ylabel_kurt, fontsize='x-large')
        ax.set_ylim(ylim_low1, ylim_high1)

        if separate_skew_axes:
            ax = ax.twinx()
        ax = make_plot(ax, data[:,z_col] + offset, data[:,skew_col], err = data[:,skew_err_col], color = 'blue', marker='o', linestyle=':', label='Skew')
        legend_lines2, legend_labels2 = ax.get_legend_handles_labels()

        ax.set_ylabel(ylabel_skew, fontsize='x-large')
        ax.legend(legend_lines1 + legend_lines2, legend_labels1 + legend_labels2, loc='lower right')
        ax.set_xlim(z[0] + 1.0, z[-1] - 1.0)
        ax.set_ylim(ylim_low2, ylim_high2)
        #ax.invert_xaxis()

        fig.tight_layout()
        fig.savefig(plot_base + 'skew_kurtosis_' + name + plot_ext, bbox_inches='tight')
```

```
164
165     #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
166
167     print 'Finished all plots.'
168
169
170 def make_plot(ax, x, y, err=None, color='black', marker='None', linestyle='None', label=None):
171     if err == None:
172         if label == None:
173             ax.plot(x, y, color=color, marker=marker, linestyle=linestyle)
174         else:
175             ax.plot(x, y, color=color, marker=marker, linestyle=linestyle, label=label)
176     else:
177         if label == None:
178             ax.errorbar(x, y, yerr=err, color=color, marker=marker, linestyle=linestyle)
179         else:
180             ax.errorbar(x, y, yerr=err, color=color, marker=marker, linestyle=linestyle, label=label)
181     return ax
182
183
184
185 def add_fit(ax, x, y, err=None, color='red'):
186     from scipy.optimize import curve_fit
187     p0 = [0.0, 0.0]
188     try:
189         coeffs, pcov = curve_fit(linear, x, y, sigma=err, p0=p0)
190     except RuntimeError:
191         print '*********  Curve fit failed  *********'
192         return np.nan, np.nan
193     xmin, xmax = ax.get_xlim()
194     x_fit = np.linspace(xmin, xmax, 20)
195     y_fit = linear(x_fit, coeffs[0], coeffs[1])
196     ax.plot(x_fit, y_fit, color=color, linestyle='--')
197     return ax, coeffs[0], pcov[0,0], coeffs[1], pcov[1,1]
198
199
200 def linear(x, slope, intercept):
201     return slope * x + intercept
202
203
204 def read_files(files, header_line = None, comment_char = '#'):
205     header = None
206     data = None
207     if type(files) == str:
208         files = [files]
209
210     if header_line != None:
211         with open(files[0], 'r') as fd:
212             for line in range(header_line):
213                 fd.readline()
214             header = fd.readline()
215         if header[0] != comment_char:
216             print "Header must start with a '%s'" % comment_char
217             sys.exit(4)
218         header = header[1:]
219         header = header.split()
220
221     for file in files:
222         print 'Reading file %s...' % (file)
223         if data == None:
224             data = np.genfromtxt(file, comments=comment_char)
225         else:
226             data = np.append(data, np.genfromtxt(file, comments=comment_char), axis=0)
227
228     print 'Finished reading files.'
229     if header_line == None:
230         return data
231     else:
232         return header, data
233
234
235 def get_skew_err(filebase):
236     z = None
237     skew = None
238     for i in range(3):
239         filename = filebase.replace('plots_all_snaps', 'plots_all_snaps_box'+str(i+1))
240         data = read_files(filename, header_line = None)
241
242         if i == 0:
243             min_length = len(data)
244         elif len(data) < min_length:
245             min_length = len(data)
246
247         if z == None:
248             z = data[-min_length:,snap_col]
249         else:
250             z = np.column_stack((z[-min_length:], data[-min_length:,snap_col]))
251
252         if skew == None:
253             skew = data[-min_length:,skew_col]
254         else:
255             skew = np.column_stack((skew[-min_length:], data[-min_length:,skew_col]))
```

```
256
257     if (z[:,0] != z[:,1]).all() or (z[:,0] != z[:,2]).all():
258         print 'Need␣matching␣snapshots␣for␣skew␣error␣from␣individual␣boxes.'
259         print z
260         sys.exit(-1)
261
262     skew_err = np.std(skew, axis=1) / np.sqrt(3.0)
263     return skew_err
264
265
266 def save_fits(file, name, slope, slope_err, intercept, intercept_err):
267     with open(file, 'a') as fd:
268         fd.write("%s␣%g␣%g␣%g␣%g\n" % (name, slope, slope_err, intercept, intercept_err))
269
270
271 plot_dest_type = 'paper'
272 if plot_dest_type == 'paper':
273     mpl.rcParams['font.family'] = 'serif'
274     mpl.rcParams['font.size'] = 16
275     mpl.rcParams['axes.linewidth'] = 3
276     mpl.rcParams['lines.linewidth'] = 4
277     mpl.rcParams['patch.linewidth'] = 4
278     mpl.rcParams['xtick.major.width'] = 3
279     mpl.rcParams['ytick.major.width'] = 3
280     mpl.rcParams['xtick.major.size'] = 8
281     mpl.rcParams['ytick.major.size'] = 8
282
283 #colors = ['red', 'green', 'blue']
284 colors = ['black', 'black', 'black']
285 labels1 = [r'$c$', r'$M_{\mathrm{vir}}$', r'$X_{\mathrm{off}}$']
286 names = ['c_rockstar', 'Mvir', 'Xoff']
287 xlabel = 'Redshift'
288 ylabels1 = [r'$\mu$␣and␣$\sigma$␣for␣$\Delta␣c$', r'$\mu$␣and␣$\sigma$␣for␣$\Delta␣M_{\mathrm{vir}}$', r'$\mu$␣
        and␣$\sigma$␣for␣$\Delta␣X_{\mathrm{off}}$']
289 ylabels2_kurt = [r'Kurtosis␣for␣$\Delta␣c$', r'Kurtosis␣for␣$\Delta␣M_{\mathrm{vir}}$', r'Kurtosis␣for␣$\Delta␣X_
        {\mathrm{off}}$']
290 ylabels2_skew = [r'Skew␣for␣$\Delta␣c$', r'Skew␣for␣$\Delta␣M_{\mathrm{vir}}$', r'Skew␣for␣$\Delta␣X_{\mathrm{off
        }}$']
291 plot_base = 'plots/'
292 plot_ext  = '.eps'
293
294 statsfile = 'plots/stats.dat'
295 transform_file_base = 'plots/'
296 transformed_data_header = '#z␣␣snap␣␣data_mean␣␣data_stdev␣␣data_skew␣␣data_kurt␣␣fit_height␣␣+/-err␣␣fit_mean␣␣
        +/-err␣␣fit_stdev␣␣+/-err␣␣fit_skew␣␣+/-err␣␣fit_kurt␣␣+/-err␣␣data_rms␣␣data_gt_epsilon␣␣chi2␣␣pval␣␣
        skew_err␣␣z\n'
297
298 z_col        = -1
299 snap_col     = 0
300 mean_col     = 7
301 mean_err_col = 8
302 var_col      = 9
303 var_err_col  = 10
304 skew_col     = 3
305 skew_err_col = -2
306 #skew_col     = 7
307 #skew_err_col = 8
308 #kurt_col     = 4
309 #kurt_err_col = -2
310 kurt_col     = 13
311 kurt_err_col = 14
312 beta_col     = 13
313 beta_err_col = 14
314
315 data_mean_col = 1
316 data_rms_col  = 15
317
318 #z_col        = -1
319 #snap_col     = 0
320 #mean_col     = 1
321 #mean_err_col = -2
322 #var_col      = 2
323 #var_err_col  = -2
324 #skew_col     = 3
325 #skew_err_col = -2
326 #kurt_col     = 4
327 #kurt_err_col = -2
328
329 offset = 0.06
330 #offset = 0.0
331
332 minsnap = 39
333 #minsnap = None
334
335 transform_variance = True
336 transform_kurtosis = True
337 expand_error       = True
338 fit_mean_trend     = True
339 separate_skew_axes = True
340 skew_err_boxes     = True
341 add_rms_line       = True
342 save_transformed_data = True
```

```
343
344
345 if __name__ == '__main__':
346     main()
```

Appendix K

Mass Trends Code

K.1 Mass and Concentration vs. Mass (Python)

```
#!/usr/bin/env python

import sys
import numpy as np
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import cm
from scipy import interpolate
from scipy.ndimage.filters import gaussian_filter
from scipy.optimize import curve_fit
#from ipdb import set_trace



def main():
    # Read in particle files
    header, halos = read_files(sys.argv[1:], header_line = 3)

    if c_source == 'density_profile':
        print 'len(halos) = ', len(halos)
        halos = halos[np.isfinite(halos[:,c_2lpt_col])]
        halos = halos[np.isfinite(halos[:,c_za_col])]
        print 'len(halos) = ', len(halos)

    print 'Filtering data...'
    for col, val in zip(lt_cols, lt_vals):
        halos = halos[halos[:, col] <= val]
    for col, val in zip(gt_cols, gt_vals):
        halos = halos[halos[:, col] >= val]
    for col, val in zip(eq_cols, eq_vals):
        halos = halos[halos[:, col] == val]
    for col, val in zip(ne_cols, ne_vals):
        halos = halos[halos[:, col] != val]

    m_avg = (halos[:,47] + halos[:,48])/2.0
    halos = np.column_stack((halos, m_avg))
    header = np.append(header, 'M_avg')

    if x_min_lim > 0:
        print 'nhalos =', len(halos)
        mask = (m_avg >= x_min_lim)
        halos = halos[mask]
        print 'nhalos =', len(halos)

    if c_source == 'rockstar':
        c1 = halos[:, Rv1_col] / halos[:, Rs1_col]
        c2 = halos[:, Rv2_col] / halos[:, Rs2_col]
        if use_klypin:
            mask = (halos[:,4] < 100)
            c1[mask] = halos[mask, Rv1_col] / halos[mask, 79]
            mask = (halos[:,5] < 100)
            c1[mask] = halos[mask, Rv2_col] / halos[mask, 80]
    if c_source == 'density_profile':
        c1 = halos[:, c_2lpt_col]
        c2 = halos[:, c_za_col]

    dc = 2.0 * (c1 - c2) / (c1 + c2)
    #dc = c1 - c2

    m1 = halos[:,47]
    m2 = halos[:,48]
    dm = 2.0 * (m1 - m2) / (m1 + m2)

    for x_col, xlabel in zip(x_cols, xlabels):
        make_plot(halos[:, x_col], dm, x_col, header[x_col], xlabel, ylabel_m, plot_base_m, stats_file_m, y_lim_m
      , use_log=False)
        make_plot(halos[:, x_col], dc, x_col, header[x_col], xlabel, ylabel_c, plot_base_c, stats_file_c, y_lim_c
      , use_log=False)
    for x_col, xlabel in zip(x_log_cols, xlabels_log):
        make_plot(halos[:, x_col], dm, x_col, header[x_col], xlabel, ylabel_m, plot_base_m, stats_file_m, y_lim_m
      , use_log=True)
        make_plot(halos[:, x_col], dc, x_col, header[x_col], xlabel, ylabel_c, plot_base_c, stats_file_c, y_lim_c
      , use_log=True)

    print 'Finished all plots.'


def read_files(files, header_line = None, comment_char = '#'):
    header = None
```

```
    data = None
    if type(files) == str:
        files = [files]

    if header_line != None:
        with open(files[0], 'r') as fd:
            for line in range(header_line):
                fd.readline()
            header = fd.readline()
        if header[0] != comment_char:
            print "Header must start with a '%s'" % comment_char
            sys.exit(4)
        header = header[1:]
        header = header.split()

    for file in files:
        print 'Reading file %s...' % (file)
        if data == None:
            data = np.genfromtxt(file, comments=comment_char)
        else:
            data = np.append(data, np.genfromtxt(file, comments=comment_char), axis=0)

    print 'Finished reading files.'
    if header_line == None:
        return data
    else:
        return header, data


def make_plot(x, y, x_col, header, xlabel, ylabel, plot_base, stats_file, y_lim, use_log):
    print 'generating plot...'
    fig = plt.figure(figsize=(9.0,6.0))
    ax = fig.add_subplot(1,1,1)
    ax = draw_hist2d(ax, x, y, y_lim)
    if fit_to_data:
        ax = draw_data_fit(ax, x, y, x.min(), x.max(), use_log=use_log)
    if fit_to_binned_data:
        mid_bins, mean, stdev, n = get_bin_avgs(x, y, use_log=use_log)
        ax = draw_bin_fit(ax, mid_bins, mean, stdev/np.sqrt(n), x.min(), x.max(), stats_file, use_log=use_log)
        ax = draw_bin_avgs(ax, mid_bins, mean, stdev, n, use_log=use_log)

    ax.set_xlim([x.min(), x.max()])
    #ax.set_yscale("log")
    ax.set_xlabel(xlabel, fontsize="x-large")
    ax.set_ylabel(ylabel, fontsize="x-large")

    fig.tight_layout()
    header = header.replace("/", "over")
    plot_name = "%s%s%0.3d%s%s%s" % (plot_base, '(', x_col, ')_', header, plot_ext)
    plt.savefig(plot_name, bbox_inches='tight')
    print 'finished plot ' + plot_name


def draw_hist2d(ax, x, y, y_lim):
    if use_log:
        xbins = np.logspace(np.log10(x.min()), np.log10(x.max()), num=nbins+1)
    else:
        xbins = np.linspace(x.min(), x.max(), num=nbins+1)

    ybins = np.linspace(y.min(), y.max(), num=nbins+1)

    if use_log:
        ax.set_xscale("log")
        im = my_hist2d(ax, x, y, bins=[xbins, ybins], zorder=-50)
    else:
        im = ax.hist2d(x, y, bins=[xbins, ybins], cmap=colormap, zorder=-50)

    if y_lim > 0.0:
        ax.set_ylim([-y_lim, y_lim])

    line = ax.plot([x.min(), x.max()], [0.0, 0.0], color='0.65', linestyle='--', linewidth=1, zorder=-20)
    return ax


def my_hist2d(ax, x, y, bins=10, range=None, normed=False, weights=None,
                        cmin=None, cmax=None, **kwargs):
    import matplotlib as mpl

    bin_range = range
    range = mpl.axes.__builtins__["range"]
    h, xedges, yedges = np.histogram2d(x, y, bins=bins, range=bin_range,
                                       normed=normed, weights=weights)

    if cmin is not None:
        h[h < cmin] = None
    if cmax is not None:
        h[h > cmax] = None

    if z_log:
        h[h<1.0] = 0.5
        h = np.log10(h)

```

```
    h = gaussian_filter(h, len(h) / 75.0)

    pc = ax.imshow(h[:,::-1].T, cmap=colormap, extent=[x.min(), x.max(), y.min(), y.max()], interpolation='
     gaussian', **kwargs)
    ax.set_xlim(xedges[0], xedges[-1])
    ax.set_ylim(yedges[0], yedges[-1])
    return h, xedges, yedges, pc


def get_bin_avgs(x, y, use_log):
    if use_log:
        fit_bins = np.logspace(np.log10(x.min()), np.log10(x.max()), num=nfit_bins+1)
    else:
        fit_bins = np.linspace(x.min(), x.max(), num=nfit_bins+1)

    mid_bins = (fit_bins[:-1] + fit_bins[1:]) / 2.0

    mean = np.array([])
    stdev = np.array([])
    n = np.array([])
    for xmin, xmax in zip(fit_bins[:-1], fit_bins[1:]):
        mask = np.logical_and(x > xmin, x <= xmax)
        if mask.sum() > 0:
            mean_el = y[mask].mean()
            #stdev_el = y[mask].std() / np.sqrt(len(y))
            stdev_el = y[mask].std()
            #stdev_el = stdev / np.sqrt(len(y[mask]))
            n_el = len(y[mask])
        else:
            mean_el = 0.0
            stdev_el = -1.0
            n_el = 0
        mean = np.append(mean, mean_el)
        stdev = np.append(stdev, stdev_el)
        n = np.append(n, n_el)

    mask = (n > 0)
    mean = mean[mask]
    stdev = stdev[mask]
    n = n[mask]
    mid_bins = mid_bins[mask]

    return mid_bins, mean, stdev, n


def draw_bin_avgs(ax, mid_bins, mean, stdev, n, use_log):
    ax.errorbar(mid_bins, mean, yerr=stdev/np.sqrt(n), fmt='o', color='black', linewidth=2)

    if draw_stdev_lines:
        ax.plot(mid_bins, mean + stdev, color='black', linestyle=':', linewidth=3, zorder=-15)
        ax.plot(mid_bins, mean - stdev, color='black', linestyle=':', linewidth=3, zorder=-15)
    return ax


def draw_bin_fit(ax, mid_bins, mean, stdev, x_min, x_max, stats_file, use_log):
    stdev[stdev == 0.0] = 0.1
    #fit data
    if use_log:
        #coefs, res, rank, singvals, rcond = np.polyfit(np.log10(mid_bins), mean, 1, full=True)
        coefs, pcov = curve_fit(linear, np.log10(mid_bins), mean, sigma=stdev, p0=[0.0, 0.0])
    else:
        #coefs, stats = np.polynomial.polynomial.polyfit(mid_bins, mean, 1, full=True)
        coefs, pcov = curve_fit(linear, mid_bins, mean, sigma=stdev, p0=[0.0, 0.0])
    print 'coefs = ', coefs


    m = coefs[0]
    b = coefs[1]
    m_err = pcov[0,0]
    b_err = pcov[1,1]

    if use_log:
        x = np.logspace(np.log10(x_min), np.log10(x_max), 100)
        y = m * np.log10(x) + b
    else:
        x = np.linspace(x_min, x_max, 100)
        y = m * x + b
    #y = x**m + b
    #line = ax.plot(x, y, color='white', linewidth=8)  # to avoid blending with colormap background
    line = ax.plot(x, y, color='magenta', zorder=-10)

    if print_fit_params:
        if use_log:
            textstr = '$y = m \log x + b$\n$m = %g$\n$b = %g$' % (m, b)
        else:
            textstr = '$y = m x + b$\n$m = %g$\n$b = %g$' % (m, b)
        props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
        ax.text(0.75, 0.95, textstr, transform=ax.transAxes, fontsize=14,
                verticalalignment='top', bbox=props)

    if save_fit_params:
        with open(stats_file, "a") as fd:
```

```
            fd.write("%g␣␣%g␣␣%g␣␣%g\n" % (m, m_err, b, b_err))

    return ax


def linear(x, slope, intercept):
    return slope * x + intercept


def draw_data_fit(ax, x, y, x_min, x_max, use_log):
    if remove_zero_strip:
        mask = (np.abs(y) >= y_epsilon)
        x = x[mask]
        y = y[mask]

    #fit data
    if use_log:
        coefs, residual, rank, singular_values, rcond = np.polyfit(np.log10(x), y, 1, full=True)
    #   coefs, stats = np.polynomial.polynomial.polyfit(np.log10(mid_bins), mean, 1, w=1.0/stdev, full=True)
    #   coefs, res, rank, singvals, rcond = np.polyfit(np.log10(mid_bins), mean, 1, full=True)
    else:
        coefs, residual, rank, singular_values, rcond = np.polyfit(x, y, 1, full=True)
    #   coefs, stats = np.polynomial.polynomial.polyfit(mid_bins, mean, 1, w=1.0/stdev, full=True)
    #   coefs, stats = np.polynomial.polynomial.polyfit(mid_bins, mean, 1, full=True)
    print 'coefs␣=', coefs, '+/-', residual


    m = coefs[0]
    b = coefs[1]
    if use_log:
        x = np.logspace(np.log10(x_min), np.log10(x_max), 100)
        y = m * np.log10(x) + b
    else:
        x = np.linspace(x_min, x_max, 100)
        y = m * x + b
    #y = x**m + b
    line = ax.plot(x, y, color='red')

    if print_fit_params:
        if use_log:
            textstr = '$y␣=␣m␣\log␣x␣+␣b$\n$m␣=␣%g$\n$b␣=␣%g$' % (m, b)
        else:
            textstr = '$y␣=␣m␣x␣+␣b$\n$m␣=␣%g$\n$b␣=␣%g$' % (m, b)
        props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
        ax.text(0.75, 0.95, textstr, transform=ax.transAxes, fontsize=14,
                verticalalignment='top', bbox=props)

    if save_fit_params:
        with open("fits_to_data.dat", "a") as fd:
            fd.write("%g␣%g␣%g\n" % (m, b, residual))

    return ax


use_log = True
#use_log = False
z_log = True

#fit_bins = True
#fit_data = True

print_fit_params = False
save_fit_params = True

use_klypin = True

remove_zero_strip = False
y_epsilon = 0.01

y_lim_m = 0.5
y_lim_c = 1.0
x_min_lim = 5.33e5 * 100

#if use_log:
#  x_cols = [4, 5, 6, 9, 10, 23, 24, 31, 32, 47, 48, 51, 52, 57, 58] # log10 columns
#else:
#  x_cols = [17, 18, 77, 78, 91, 92, 93, 94, 97, 98, 99, 100, 107, 108, 111, 112] # nolog columns

x_cols     = []
x_log_cols = [-1]
#x_log_cols = [47, 48, -1]

xlabels = []
xlabels_log = [r"$M_{\mathrm{vir,avg}}␣\,␣\mathrm{(M_{\odot})}$"]
#xlabels_log = [r"$\mathrm{M_{2LPT} (M_{\odot})}$",
#               r"$\mathrm{M_{ZA} (M_{\odot})}$",
#              r"$\mathrm{M_{avg} (M_{\odot})}$"]

#ylabel = r"$\mathrm{(M_{2LPT} - M_{ZA}) / M_{avg}}$"
ylabel_m = r"$(M_{\mathrm{vir,2LPT}}␣\,␣-␣\,␣M_{\mathrm{vir,ZA}})␣\,␣/␣\,␣M_{\mathrm{vir,avg}}$"
ylabel_c = r"$(c_{\mathrm{2LPT}}␣\,␣-␣\,␣c_{\mathrm{ZA}})␣\,␣/␣\,␣c_{\mathrm{avg}}$"

```

```
#c_source = 'density_profile'
c_source = 'rockstar'

plot_base_m = 'plots/diff_M_-_vs_-_'
plot_base_c = 'plots/diff_c_-_vs_-_'
plot_ext  = '.eps'

stats_file_m = 'fits_to_bins_m.dat'
stats_file_c = 'fits_to_bins_c.dat'

#plot_name = 'test.eps'
#plot_name = 'c_v_M200c_2lpt.eps'
fit_to_binned_data = True
fit_to_data = False
draw_stdev_lines = True

Rv1_col = 53
Rv2_col = 54
Rs1_col = 55
Rs2_col = 56

c_2lpt_col = 17
c_za_col   = 18

nbins = 100
nfit_bins = 10

## c_2lpt, c_za, chi2_2lpt, chi2_za
#lt_cols = [17, 18, 37, 38]
#lt_vals = [100.0, 100.0, 10.0, 10.0]
#
## c_2lpt, c_za, chi2_2lpt, chi2_za
#gt_cols = [17, 18, 37, 38]
#gt_vals = [1.0, 1.0, 0.0, 0.0]

lt_cols = []
lt_vals = []

gt_cols = [4, 5]
gt_vals = [100, 100]

eq_cols = [109, 110]
eq_vals = [-1, -1]

ne_cols = []
ne_vals = []

#colormap = cm.PuBuGn
#colormap = cm.cubehelix_r
#colormap = cm.ocean_r
#colormap = cm.rainbow
#colormap = cm.gnuplot2_r
#colormap = cm.CMRmap_r

def add_white(orig_map, num):
    temp_cmap = cm.get_cmap(orig_map, num)
    vals = temp_cmap(np.arange(num))
    nfade = num / 7
    vals[:nfade,0] = np.linspace(1., vals[nfade-1,0], nfade)
    vals[:nfade,1] = np.linspace(1., vals[nfade-1,1], nfade)
    vals[:nfade,2] = np.linspace(1., vals[nfade-1,2], nfade)
    #vals[:nfade,3] = np.linspace(0., vals[nfade-1,3], nfade)
    #vals[0] = [1.0, 1.0, 1.0, 1.0]
    #vals[1] = (vals[1] + [1.0, 1.0, 1.0, 1.0]) / 2.0
    newcmap = mpl.colors.LinearSegmentedColormap.from_list("custom_1", vals)
    return newcmap

colormap = add_white('rainbow', 30)

plot_dest_type = 'paper'
if plot_dest_type == 'paper':
    mpl.rcParams['font.family'] = 'serif'
    mpl.rcParams['font.size'] = 16
    mpl.rcParams['axes.linewidth'] = 3
    mpl.rcParams['lines.linewidth'] = 4
    #mpl.rcParams['lines.linewidth'] = 3
    mpl.rcParams['patch.linewidth'] = 4
    #mpl.rcParams['patch.linewidth'] = 3
    mpl.rcParams['xtick.major.width'] = 3
    mpl.rcParams['ytick.major.width'] = 3
    mpl.rcParams['xtick.major.size'] = 8
    mpl.rcParams['ytick.major.size'] = 8
    mpl.rcParams['xtick.minor.width'] = 2
    mpl.rcParams['ytick.minor.width'] = 2
    mpl.rcParams['xtick.minor.size'] = 4
    mpl.rcParams['ytick.minor.size'] = 4
    #mpl.rcParams['lines.antialiased'] = True


if __name__ == '__main__':
    main()
```

K.2 PBS Submission Script (Bash)

```
#!/usr/bin/env bash
#PBS -M djsissom@gmail.com
#PBS -m bae
#PBS -l nodes=1:ppn=1
#PBS -l pmem=40000mb
#PBS -l mem=4000mb
#PBS -l walltime=1:00:00
#PBS -o out.log
#PBS -j oe

minsnap=0
maxsnap=61

# Change to working directory
echo $PBS_NODEFILE
cd $PBS_O_WORKDIR

[ -e fits_to_bins_m.dat ] && rm -v fits_to_bins_m.dat
[ -e fits_to_bins_c.dat ] && rm -v fits_to_bins_c.dat
#rm -v fits_to_data.dat

echo "#snap␣␣slope␣␣slope_err␣␣intercept␣␣intercept_err" > fits_to_bins_m.dat
echo "#snap␣␣slope␣␣slope_err␣␣intercept␣␣intercept_err" > fits_to_bins_c.dat

for ((snap=$minsnap; snap<=$maxsnap; snap++)); do

    if [ $snap -lt 10 ]; then
        j=00$snap
    elif [ $snap -lt 100 ]; then
        j=0$snap
    fi

    new_plot_dir=snap${j}_plots

    if [ ! -e plots_all_snaps/${new_plot_dir} ]; then
        mkdir plots_all_snaps/${new_plot_dir}
    fi

    {
        echo "Starting␣snap${j}..."
        echo -n "${j}␣␣␣␣" >> fits_to_bins_m.dat
        echo -n "${j}␣␣␣␣" >> fits_to_bins_c.dat
        ./mass_plot.py ~/projects/simulations/rockstar/box{1,2,3}/crossmatch/snap${j}/halos.dat > plots/out.log
     2>&1
        mv plots/* plots_all_snaps/${new_plot_dir}/.
        echo "Finished␣snap${j}"
    }

done

# - end of script
```

Appendix L

Alternate Differential Distribution Redshift Trends Code (Python)

```
#!/usr/bin/env python

import sys
import numpy as np
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
from scipy.special import gamma as Gamma
from scipy.special import psi as digamma
from ipdb import set_trace


def main():
    if (len(sys.argv[1:]) == 4):
        data1 = read_files(sys.argv[1], header_line = None)
        data2 = read_files(sys.argv[2], header_line = None)
        data3 = read_files(sys.argv[3], header_line = None)
        rsnap_data = read_files(sys.argv[4], header_line = None)
    else:
        print 'need 4 files'
        sys.exit(15)

    if fit_mean_trend:
        with open(statsfile, 'w') as fd:
            fd.write("#plot slope slope_err intercept intercept_err\n")

    if minsnap > 0:
        data1 = data1[data1[:,0] >= minsnap]
        data2 = data2[data2[:,0] >= minsnap]
        data3 = data3[data3[:,0] >= minsnap]

    z = 1.0 / rsnap_data[:,1] - 1.0
    if (len(data1) == len(data2)) and (len(data1) == len(data3)):
        z = z[-len(data1):]
    else:
        sys.exit(16)

    data1 = np.column_stack((data1, z))
    data2 = np.column_stack((data2, z))
    data3 = np.column_stack((data3, z))


    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
    # make mean and stdv plots                                                     #
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#

    for (data, ylabel, label, name) in zip([data1, data2, data3], ylabels1, labels1, names):
        print "Making %s plot..." % (name + ' xvals')
        fig = plt.figure(figsize=(9.0, 6.0))
        ax = fig.add_subplot(111)

        ax = make_plot(ax, data[:,z_col], data[:,peak_col], err = None, color = 'black', marker='o', linestyle='-
    ', label=None)

        for (x_val_col, color) in zip(x_val_cols, colors1):
            ax = make_plot(ax, data[:,z_col], data[:,x_val_col], err = None, color = color, marker='o', linestyle
    ='--', label=None)

        #if add_rms_line:
        #    ax = make_plot(ax, data[:,z_col], data[:,data_rms_col], color = 'green', linestyle=':')

        #if fit_mean_trend:
        #    ax, slope, slope_err, intercept, intercept_err = add_fit(ax, data[:,z_col], data[:,mean_col], err=
    data[:,mean_err_col], color='red')
        #    save_fits(statsfile, name, slope, np.sqrt(slope_err), intercept, np.sqrt(intercept_err))

        #ax.legend(loc='lower right')
        ax.set_xlim(z[0] + 1.0, z[-1] - 1.0)
        #ax.invert_xaxis()

        ax.set_xlabel(xlabel, fontsize='x-large')
        ax.set_ylabel(ylabel, fontsize='x-large')

        fig.tight_layout()
        fig.savefig(plot_base + name + '_xvals' + plot_ext, bbox_inches='tight')

    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#

    for (data, ylabel, label, name) in zip([data1, data2, data3], ylabels2, labels1, names):
        print "Making %s plot..." % (name + ' sumfrac')
        fig = plt.figure(figsize=(9.0, 6.0))
        ax = fig.add_subplot(111)
```

```
80
81          for (sum_frac_col, color) in zip(sum_frac_cols, colors2):
82              ax = make_plot(ax, data[:,z_col], data[:,sum_frac_col], err = None, color = color, marker='o',
        linestyle='-', label=None)
83          for (doublesum_frac_col, color) in zip(doublesum_frac_cols, colors2):
84              ax = make_plot(ax, data[:,z_col], data[:,doublesum_frac_col], err = None, color = color, marker='o',
        linestyle='--', label=None)
85
86          ax.set_xlabel(xlabel, fontsize='x-large')
87          ax.set_ylabel(ylabel, fontsize='x-large')
88          ax.set_xlim(z[0] + 1.0, z[-1] - 1.0)
89          ax.set_yscale('log')
90
91          fig.tight_layout()
92          fig.savefig(plot_base + name + '_sumfrac' + plot_ext, bbox_inches='tight')
93
94
95      print 'Finished all plots.'
96
97
98  def make_plot(ax, x, y, err=None, color='black', marker='None', linestyle='None', label=None):
99      if err == None:
100         if label == None:
101             ax.plot(x, y, color=color, marker=marker, linestyle=linestyle)
102         else:
103             ax.plot(x, y, color=color, marker=marker, linestyle=linestyle, label=label)
104     else:
105         if label == None:
106             ax.errorbar(x, y, yerr=err, color=color, marker=marker, linestyle=linestyle)
107         else:
108             ax.errorbar(x, y, yerr=err, color=color, marker=marker, linestyle=linestyle, label=label)
109     return ax
110
111
112
113 def add_fit(ax, x, y, err=None, color='red'):
114     from scipy.optimize import curve_fit
115     p0 = [0.0, 0.0]
116     try:
117         coeffs, pcov = curve_fit(linear, x, y, sigma=err, p0=p0)
118     except RuntimeError:
119         print '********* Curve fit failed *********'
120         return np.nan, np.nan
121     xmin, xmax = ax.get_xlim()
122     x_fit = np.linspace(xmin, xmax, 20)
123     y_fit = linear(x_fit, coeffs[0], coeffs[1])
124     ax.plot(x_fit, y_fit, color=color, linestyle='--')
125     return ax, coeffs[0], pcov[0,0], coeffs[1], pcov[1,1]
126
127
128 def linear(x, slope, intercept):
129     return slope * x + intercept
130
131
132 def read_files(files, header_line = None, comment_char = '#'):
133     header = None
134     data = None
135     if type(files) == str:
136         files = [files]
137
138     if header_line != None:
139         with open(files[0], 'r') as fd:
140             for line in range(header_line):
141                 fd.readline()
142             header = fd.readline()
143         if header[0] != comment_char:
144             print "Header must start with a '%s'" % comment_char
145             sys.exit(4)
146         header = header[1:]
147         header = header.split()
148
149     for file in files:
150         print 'Reading file %s...' % (file)
151         if data == None:
152             data = np.genfromtxt(file, comments=comment_char)
153         else:
154             data = np.append(data, np.genfromtxt(file, comments=comment_char), axis=0)
155
156     print 'Finished reading files.'
157     if header_line == None:
158         return data
159     else:
160         return header, data
161
162
163 def save_fits(file, name, slope, slope_err, intercept, intercept_err):
164     with open(file, 'a') as fd:
165         fd.write("%s %g %g %g %g\n" % (name, slope, slope_err, intercept, intercept_err))
166
167
168 plot_dest_type = 'paper'
169 if plot_dest_type == 'paper':
```

```
    mpl.rcParams['font.family'] = 'serif'
    mpl.rcParams['font.size'] = 16
    mpl.rcParams['axes.linewidth'] = 3
    mpl.rcParams['lines.linewidth'] = 4
    mpl.rcParams['patch.linewidth'] = 4
    mpl.rcParams['xtick.major.width'] = 3
    mpl.rcParams['ytick.major.width'] = 3
    mpl.rcParams['xtick.major.size'] = 8
    mpl.rcParams['ytick.major.size'] = 8

#colors = ['red', 'green', 'blue']
colors = ['black', 'black', 'black']
labels1 = [r'$c$', r'$M_{\mathrm{vir}}$', r'$X_{\mathrm{off}}$']
names = ['c_rockstar', 'Mvir', 'Xoff']
xlabel = 'Redshift'
ylabels1 = [r"$\Delta␣c(f_{h},z)$␣and␣$\Delta␣c_{\mathrm{peak}}$", r"$\Delta␣M_{\mathrm{vir}}(f_{h},z)$␣and␣$\
    Delta␣M_{\mathrm{vir,␣peak}}$", r"$\Delta␣X_{\mathrm{off}}(f_{h},z)$␣and␣$\Delta␣X_{\mathrm{off,␣peak}}$"
    ]
ylabels2 = [r"$f_{h}(\Delta␣c,z)$", r"$f_{h}(\Delta␣M_{\mathrm{vir}},z)$", r"$f_{h}(\Delta␣X_{\mathrm{off}},z)
    $"]
plot_base = 'plots/'
plot_ext  = '.eps'

statsfile = 'plots/stats.dat'

z_col        = -1
snap_col     = 0
mean_col     = 7
mean_err_col = 8
var_col      = 9
var_err_col  = 10
skew_col     = 3
skew_err_col = -2
#skew_col     = 7
#skew_err_col = 8
#kurt_col     = 4
#kurt_err_col = -2
kurt_col     = 13
kurt_err_col = 14
beta_col     = 13
beta_err_col = 14

data_mean_col = 1
data_rms_col  = 15



peak_col  = 1
x_val_cols = np.array([4, 6, 8]) + 2
sum_frac_cols = np.array([2, 4, 6, 8]) + 2 + 9
doublesum_frac_cols = sum_frac_cols + 9

colors1 = ['red', 'green', 'blue']
colors2 = ['blue', 'green', 'red', 'black']

offset = 0.06
#offset = 0.0

minsnap = 39
#minsnap = None

fit_mean_trend     = False
add_rms_line       = False


if __name__ == '__main__':
    main()
```

Appendix M

Miscellaneous Scripts

M.1 Directory Structure Setup (Bash)

```
#!/usr/bin/env bash

minsnap=0
maxsnap=61

minbox=1
maxbox=3

for ((i=$minbox; i<=$maxbox; i++)); do
  if [ ! -e ../box$i ]; then
    mkdir -v ../box$i
  fi
  if [ ! -e ../box$i/2lpt ]; then
    mkdir -v ../box$i/2lpt
  fi
  if [ ! -e ../box$i/za ]; then
    mkdir -v ../box$i/za
  fi
  if [ ! -e ../box$i/crossmatch ]; then
    mkdir -v ../box$i/crossmatch
  fi

  cp -v run_*.pbs ../box$i/.
  cp -v postprocess.sh ../box$i/.

  for ((snap=$minsnap; snap<=$maxsnap; snap++)); do
    if [ $snap -lt 10 ]; then
      j=00$snap
    elif [ $snap -lt 100 ]; then
      j=0$snap
    fi

    if [ ! -e ../box$i/2lpt/snap$j ]; then
      mkdir -v ../box$i/2lpt/snap$j
    fi
    if [ ! -e ../box$i/za/snap$j ]; then
      mkdir -v ../box$i/za/snap$j
    fi

    cp -v -r proto/* ../box$i/2lpt/snap$j/.
    cp -v -r proto/* ../box$i/za/snap$j/.

    ln -v -s ~/projects/data/2lpt/box$i/2lpt_512_z300_PM_$j ../box$i/2lpt/snap$j/particles/2lpt_512_z300_PM_$j
    ln -v -s ~/projects/data/za/box$i/za_512_z300_PM_$j ../box$i/za/snap$j/particles/za_512_z300_PM_$j

    echo /home/sissomdj/projects/simulations/rockstar/box$i/2lpt/snap$j/particles/2lpt_512_z300_PM_$j > ../box$i
     /2lpt/snap$j/particles/snapnames.lst
    echo /home/sissomdj/projects/simulations/rockstar/box$i/za/snap$j/particles/za_512_z300_PM_$j > ../box$i/za/
     snap$j/particles/snapnames.lst

    echo "BGC2_SNAPNAMES = \"/home/sissomdj/projects/simulations/rockstar/box$i/2lpt/snap$j/particles/snapnames.
     lst\"">> ../box$i/2lpt/snap$j/onenode.cfg
    echo "BGC2_SNAPNAMES = \"/home/sissomdj/projects/simulations/rockstar/box$i/za/snap$j/particles/snapnames.lst
     \"">> ../box$i/za/snap$j/onenode.cfg

    echo "FILENAME = \"2lpt_512_z300_PM_$j\"" >> ../box$i/2lpt/snap$j/onenode.cfg
    echo "FILENAME = \"za_512_z300_PM_$j\"" >> ../box$i/za/snap$j/onenode.cfg
  done

done
```

M.2 CrossMatch Setup (Bash)

```
#!/usr/bin/env bash

minsnap=0
maxsnap=61

minbox=1
maxbox=3

for ((i=$minbox; i<=$maxbox; i++)); do
  if [ ! -e ../box$i/crossmatch ]; then
    mkdir -v ../box$i/crossmatch
  fi

  cp -v run_crossmatch.pbs ../box$i/.

  for ((snap=$minsnap; snap<=$maxsnap; snap++)); do
```

```
    if [ $snap -lt 10 ]; then
      j=00$snap
    elif [ $snap -lt 100 ]; then
      j=0$snap
    fi

    if [ ! -e ../box$i/crossmatch/snap$j ]; then
      mkdir -v ../box$i/crossmatch/snap$j
    fi

    cp -v -r crossmatch_proto/* ../box$i/crossmatch/snap$j/.

    echo "OUTPUT_DIR␣␣␣␣␣␣␣␣/home/sissomdj/projects/simulations/rockstar/box$i/crossmatch/snap$j" >> ../box$i/
     crossmatch/snap$j/rockstar_2lpt.param
    echo "FIRST_GROUPDIR␣␣␣␣/home/sissomdj/projects/simulations/rockstar/box$i/2lpt/snap$j/halos" >> ../box$i/
     crossmatch/snap$j/rockstar_2lpt.param
    echo "SECOND_GROUPDIR␣␣␣/home/sissomdj/projects/simulations/rockstar/box$i/za/snap$j/halos" >> ../box$i/
     crossmatch/snap$j/rockstar_2lpt.param

    echo "OUTPUT_DIR␣␣␣␣␣␣␣␣/home/sissomdj/projects/simulations/rockstar/box$i/crossmatch/snap$j" >> ../box$i/
     crossmatch/snap$j/rockstar_za.param
    echo "FIRST_GROUPDIR␣␣␣␣/home/sissomdj/projects/simulations/rockstar/box$i/za/snap$j/halos" >> ../box$i/
     crossmatch/snap$j/rockstar_za.param
    echo "SECOND_GROUPDIR␣␣␣/home/sissomdj/projects/simulations/rockstar/box$i/2lpt/snap$j/halos" >> ../box$i/
     crossmatch/snap$j/rockstar_za.param

  done

done
```

M.3 Individual Snapshot Rockstar Run Script (Bash)

```
#!/bin/bash

echo "Cleaning␣old␣files..."
if [ -e out.log ]; then
  mv -v out.log out.log.bak
fi
if [ -e server.out ]; then
  mv -v server.out server.out.bak
fi
if [ -e clients.out ]; then
  mv -v clients.out clients.out.bak
fi
if [ -e auto-rockstar.cfg ]; then
  rm -v auto-rockstar.cfg
fi
if [ $(ls halos/* 2> /dev/null | wc -l) != "0" ]; then
  rm -rv halos/*
fi

echo "Submitting␣run␣script..."
echo "qsub␣run_rockstar.pbs"
qsub run_rockstar.pbs
```

M.4 All Snapshots Rockstar 2lpt PBS Submission Script (Bash)

```
#!/usr/bin/env bash

#PBS -M djsissom@gmail.com
#PBS -m bae
#PBS -l nodes=1:ppn=10
#PBS -l pmem=3000mb
#PBS -l mem=30000mb
#PBS -l walltime=6:00:00
#PBS -o out_2lpt.log
#PBS -j oe

echo $PBS_NODEFILE
cd $PBS_O_WORKDIR

for snapdir in 2lpt/*; do
  # Change to working directory
  echo Working on $snapdir...
  cd $PBS_O_WORKDIR/$snapdir

  # Start the server
  rockstar -c onenode.cfg &> server.out &

  # Wait for auto-rockstar.cfg to be created
  perl -e 'sleep 1 while (!(-e "halos/auto-rockstar.cfg"))'
  mv halos/auto-rockstar.cfg .

  # Execute the reader processes
  mpiexec -verbose -n 1 rockstar -c auto-rockstar.cfg >> clients.out 2>&1 &
  sleep 20

  # Execute the analysis processes
  mpiexec -verbose -n 8 rockstar -c auto-rockstar.cfg >> clients.out 2>&1

```

```
  # - end of script
done
```

M.5 All Snapshots Rockstar za PBS Submission Script (Bash)

```
#!/usr/bin/env bash

#PBS -M djsissom@gmail.com
#PBS -m bae
#PBS -l nodes=1:ppn=10
#PBS -l pmem=3000mb
#PBS -l mem=30000mb
#PBS -l walltime=6:00:00
#PBS -o out_za.log
#PBS -j oe

echo $PBS_NODEFILE
cd $PBS_O_WORKDIR

for snapdir in za/*; do
  # Change to working directory
  echo Working on $snapdir...
  cd $PBS_O_WORKDIR/$snapdir

  # Start the server
  rockstar -c onenode.cfg &> server.out &

  # Wait for auto-rockstar.cfg to be created
  perl -e 'sleep 1 while (!(-e "halos/auto-rockstar.cfg"))'
  mv halos/auto-rockstar.cfg .

  # Execute the reader processes
  mpiexec -verbose -n 1 rockstar -c auto-rockstar.cfg >> clients.out 2>&1 &
  sleep 20

  # Execute the analysis processes
  mpiexec -verbose -n 8 rockstar -c auto-rockstar.cfg >> clients.out 2>&1

  # - end of script
done
```

M.6 All Snapshots Rockstar Post-Process Script (Bash)

```
#!/usr/bin/env bash

startdir=`pwd`

for snapdir in {2lpt,za}/*; do
  echo Working on $snapdir...
  cd $startdir/$snapdir

  ./postprocess

done

# - end of script
```

M.7 All Snapshots CrossMatch PBS Submission Script (Bash)

```
#!/usr/bin/env bash

#PBS -M djsissom@gmail.com
#PBS -m bae
#PBS -l nodes=62:ppn=1
#PBS -l pmem=3000mb
#PBS -l mem=186000mb
#PBS -l walltime=1:00:00
#PBS -o out_crossmatch.log
#PBS -j oe

echo $PBS_NODEFILE
cd $PBS_O_WORKDIR

for snapdir in crossmatch/*; do
  # Change to working directory
  echo Working on $snapdir...
  cd $PBS_O_WORKDIR/$snapdir

  {
      mpiexec -verbose -n 1 crossmatch rockstar_2lpt.param > out_2lpt_first.log 2>&1
      mpiexec -verbose -n 1 crossmatch rockstar_za.param   > out.za_first.log   2>&1
      echo "Finished␣$snapdir"
  } &

done

wait
# - end of script
```

M.8 All Snapshots Density Profile PBS Submission Script (Bash)

```bash
#!/usr/bin/env bash

#PBS -M djsissom@gmail.com
#PBS -m bae
#PBS -l nodes=124:ppn=1
#PBS -l pmem=4000mb
#PBS -l mem=496000mb
#PBS -l walltime=1:00:00
#PBS -o out_density_profile.log
#PBS -j oe

echo $PBS_NODEFILE
cd $PBS_O_WORKDIR

for snapdir in {2lpt,za}/snap*/halos; do
  # Change to working directory
  echo Working on $snapdir...
  cd $PBS_O_WORKDIR/$snapdir

  {
    mpiexec -verbose -n 1 density_profile halos_0.*.bgc2 > density_profile_out.log 2>&1
    echo "Finished $snapdir"
  } &

done

wait
# - end of script
```