

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

	Page
<b>ACKNOWLEDGMENTS</b> . . . . .	<b>i</b>
<b>LIST OF TABLES</b> . . . . .	<b>vii</b>
<b>LIST OF FIGURES</b> . . . . .	<b>viii</b>
<b>I Introduction</b> . . . . .	<b>1</b>
I.1 The Early Universe . . . . .	1
I.1.1 The CMB Epoch . . . . .	1
I.1.1.1 Recombination . . . . .	1
I.1.1.2 The Cosmic Microwave Background . . . . .	2
I.1.2 Dark Matter Halo Formation . . . . .	2
I.1.2.1 Collapse . . . . .	2
I.1.2.2 Accretion . . . . .	2
I.1.2.3 Mergers . . . . .	2
I.1.2.4 Large-scale Structure . . . . .	2
I.1.3 Halo Properties . . . . .	2
I.1.3.1 Mass . . . . .	2
I.1.3.2 Density and Concentration . . . . .	3
I.1.3.3 Substructure and Environment . . . . .	5
I.1.4 Baryonic Processes . . . . .	5
I.1.4.1 The First Stars . . . . .	5
I.1.4.2 Supermassive Black Holes . . . . .	5
I.1.4.3 Enrichment and the The Intergalactic Medium . . . . .	5
I.1.4.4 Reionization . . . . .	6
I.2 Computational Theory . . . . .	6
I.2.1 Simulation Initialization . . . . .	6
I.2.1.1 Initial Conditions and the Surface of Last Scattering . . . . .	6
I.2.1.2 The Zel'dovich Approximation . . . . .	6
I.2.1.3 Second-order Lagrangian Perturbation Theory . . . . .	6
I.2.2 Dark Matter Halos in N-body Simulations . . . . .	6
I.2.2.1 Spherical Overdensity . . . . .	6
I.2.2.2 Friends-of-Friends . . . . .	6
<b>II Numerical Methods</b> . . . . .	<b>9</b>
II.1 Initialization Code . . . . .	9

II.1.1	Sampling the Power Spectrum . . . . .	9
II.1.1.1	Cosmological Parameters . . . . .	9
II.1.1.2	Sampling . . . . .	9
II.1.2	Particle Displacement with ZA . . . . .	9
II.1.3	Particle Displacement with 2LPT . . . . .	9
II.2	Simulations with GADGET-2 . . . . .	9
II.2.1	GADGET-2 . . . . .	10
II.2.1.1	Force Calculation and Softening . . . . .	10
II.2.1.2	The Tree Code . . . . .	10
II.2.1.3	Parallelization . . . . .	11
II.2.1.4	Metrics . . . . .	11
II.2.2	Simulations . . . . .	11
II.3	Halo Finding with ROCKSTAR . . . . .	11
II.3.1	Halo Identification . . . . .	11
II.3.2	Halo Properties . . . . .	12
II.3.2.1	Mass . . . . .	12
II.3.2.2	Concentration . . . . .	12
II.3.2.3	Relaxation Parameters . . . . .	12
II.3.3	Halo Catalogs . . . . .	12
II.4	CROSSMATCH . . . . .	13
II.5	Analysis . . . . .	13
II.5.1	Halo Properties with ROCKSTAR . . . . .	14
II.5.1.1	Simulation Snapshots and ROCKSTAR Setup . . . . .	14
II.5.1.2	ROCKSTAR Output and Post-processing . . . . .	15
II.5.2	Density Profile Fitting . . . . .	17
II.5.2.1	Density Profiles . . . . .	17
II.5.2.2	Fitting . . . . .	17
II.5.2.3	Characterization of Uncertainty . . . . .	18
II.5.2.4	Mass Profiles . . . . .	19
II.5.2.5	Concentration Comparison to ROCKSTAR . . . . .	19
II.5.3	Cross-matched Halo Catalog . . . . .	19
II.5.3.1	Cross-matching . . . . .	20
II.5.3.2	Database Aggregation and Filtering . . . . .	20
II.5.4	Halo Comparison . . . . .	22
II.5.4.1	Match Verification . . . . .	22
II.5.4.2	Morphology . . . . .	22
II.5.4.3	Density Profiles . . . . .	22
II.5.5	Difference Distributions . . . . .	22
II.5.5.1	Histograms . . . . .	22
II.5.5.2	Fitting . . . . .	22
II.5.5.3	Mass Quartiles . . . . .	22
II.5.6	Redshift Trends . . . . .	22
II.5.6.1	Mean and Standard Deviation . . . . .	22
II.5.6.2	Skew . . . . .	23
II.5.6.3	Kurtosis . . . . .	23

II.5.7	2-D Histograms . . . . .	25
II.5.7.1	Binning and Plotting . . . . .	25
II.5.7.2	Fitting . . . . .	25
II.5.8	Alternate Difference Distributions . . . . .	25
II.5.8.1	Equivalent Displacement . . . . .	25
II.5.8.2	Redshift Trends . . . . .	25
II.6	Automation . . . . .	25
<b>III</b>	<b>Exploring Dark Matter Halo Populations in 2LPT and ZA Simulations . . .</b>	<b>31</b>
III.1	Introduction . . . . .	31
III.2	Numerical Methods . . . . .	36
III.3	Results . . . . .	41
III.3.1	Individual halo pairs . . . . .	41
III.3.2	Differences in ensemble halo properties . . . . .	41
III.3.3	Time evolution of mass and concentration . . . . .	43
III.3.4	Dependence of mass and concentration differences on halo mass .	45
III.3.5	Fractional differences in halo populations . . . . .	47
III.4	Discussion . . . . .	48
III.5	Conclusion . . . . .	50
<b>IV</b>	<b>Supermassive Black Holes and Their Hosts . . . . .</b>	<b>59</b>
IV.1	Introduction . . . . .	59
IV.1.1	Galaxy Properties . . . . .	59
IV.1.1.1	Color . . . . .	59
IV.1.1.2	Morphology . . . . .	60
IV.1.2	Supermassive Black Hole Properties . . . . .	61
IV.1.3	Correlations . . . . .	64
IV.1.3.1	The M-Sigma Relation . . . . .	64
IV.1.3.2	The Fundamental Plane . . . . .	66
IV.1.3.3	The Green Valley . . . . .	67
IV.2	Galaxy Evolution . . . . .	69
IV.2.1	Dark Matter Halos . . . . .	69
IV.2.2	Galaxy Mergers . . . . .	70
IV.3	Supermassive Black Hole Growth . . . . .	71
IV.3.1	Binary Mergers . . . . .	71
IV.3.1.1	Dynamical Friction and Inspiral . . . . .	71
IV.3.1.2	The Final Parsec Problem . . . . .	72
IV.3.1.3	Gravitational Waves and Recoil Kicks . . . . .	72
IV.3.2	Accretion . . . . .	76
IV.3.2.1	Bondi-Hoyle-Lyttleton Accretion . . . . .	76
IV.3.2.2	Disk Accretion and Active Galactic Nuclei . . . . .	78
IV.4	Conclusion . . . . .	79
IV.4.1	Correlations . . . . .	79

IV.4.2	Open Questions . . . . .	79
<b>V</b>	<b>Conclusion . . . . .</b>	<b>81</b>
<b>BIBLIOGRAPHY . . . . .</b>		<b>82</b>
<b>Appendices . . . . .</b>		<b>86</b>
<b>A</b>	<b>ROCKSTAR Configuration and Execution . . . . .</b>	<b>87</b>
A.1	Single Node Configuration File (Text) . . . . .	87
A.2	PBS Submission Script (Bash) . . . . .	87
A.3	Post-Processing Script (Bash) . . . . .	88
<b>B</b>	<b>CROSSMATCH Modifications and Configuration . . . . .</b>	<b>89</b>
B.1	2LPT First Configuration File (Text) . . . . .	89
B.2	ZA First Configuration File (Text) . . . . .	89
<b>C</b>	<b>BGC2 Import Code (Python) . . . . .</b>	<b>90</b>
<b>D</b>	<b>Density Profile Code (Python) . . . . .</b>	<b>92</b>
<b>E</b>	<b>CROSSMATCH Best Match Code . . . . .</b>	<b>100</b>
E.1	Best Match (Python) . . . . .	100
E.2	PBS Submission Script (Bash) . . . . .	100
<b>F</b>	<b>Database Generation Code . . . . .</b>	<b>102</b>
F.1	Halo Match (Python) . . . . .	102
F.2	PBS Submission Script (Bash) . . . . .	106
<b>G</b>	<b>Halo Comparison Code . . . . .</b>	<b>108</b>
G.1	Particle Comparison (Python) . . . . .	108
G.2	Density Comparison (Python) . . . . .	113
<b>H</b>	<b>Concentration Comparison Code (Python) . . . . .</b>	<b>121</b>

<b>I</b>	<b>Differential Histogram Code</b>	<b>123</b>
I.1	Histogram Generation and Fitting (Python)	123
I.2	PBS Submission Script (Bash)	131
I.3	PBS Submission Script - Individual Boxes (Bash)	131
I.4	Statistics Collection Script (Bash)	132
<b>J</b>	<b>Redshift Trends Code (Python)</b>	<b>133</b>
<b>K</b>	<b>Mass Trends Code</b>	<b>138</b>
K.1	Mass (Python)	138
K.2	Concentration (Python)	142
K.3	PBS Submission Script (Bash)	147
<b>L</b>	<b>Alternate Differential Distribution Redshift Trends Code (Python)</b>	<b>148</b>
<b>M</b>	<b>Miscellaneous Scripts</b>	<b>151</b>
M.1	Directory Structure Setup (Bash)	151
M.2	CROSSMATCH Setup (Bash)	151
M.3	Individual Snapshot ROCKSTAR Run Script (Bash)	152
M.4	All Snapshots ROCKSTAR 2LPT PBS Submission Script (Bash)	152
M.5	All Snapshots ROCKSTAR ZA PBS Submission Script (Bash)	153
M.6	All Snapshots ROCKSTAR Post-Process Script (Bash)	153
M.7	All Snapshots CROSSMATCH PBS Submission Script (Bash)	153
M.8	All Snapshots Density Profile PBS Submission Script (Bash)	154

## LIST OF TABLES

Table	Page
III.1      Coefficients for linear least squares fits from Figure III.3. . . . .	43
III.2      Coefficients for linear least squares fits from Figure III.5. . . . .	46

## LIST OF FIGURES

Figure	Page
I.1      Caption . . . . .	7
I.2      Caption . . . . .	8
II.1     Potential and force softening. . . . .	10
II.2     Barns-Hut oct-tree in two dimensions. . . . .	10
II.3     Domain decomposition. . . . .	11
II.4     Force parallelism. . . . .	12
II.5     Energy conservation for an initially isothermal gas sphere. . . . .	13
II.6     Density profiles for two large halos at $z = 14$ and $z = 6$ . . . . .	26
II.7     Mass profiles a large halo at $z = 6$ . . . . .	27
II.8     Example of halo particle matching at $z = 6$ . . . . .	28
II.9     Comparison of two large well-fit companion halos $z = 6$ . . . . .	29
II.10    Comparison of two large companion halos $z = 6$ with differing nuclear structure.	30
III.1    Comparison of matched 2LPT and ZA halos . . . . .	53
III.2    Histograms of $\Delta M_{\text{vir}}$ and $\Delta c$ . . . . .	54
III.3    Statistics as functions of redshift for generalized normal fits . . . . .	55
III.4 $\Delta M_{\text{vir}}$ and $\Delta c$ as a function of $M_{\text{vir,avg}}$ . . . . .	56
III.5    Slopes of the $\Delta q$ vs. $M_{\text{vir,avg}}$ fit functions. . . . .	57
III.6    Fractional error distribution statistics as functions of redshift . . . . .	58
IV.1    The Hubble tuning fork . . . . .	61

IV.2	Maser orbits fit to a warped disk for NGC4258 . . . . .	63
IV.3	The M- $\sigma$ relation for galaxies with dynamical measurements . . . . .	65
IV.4	The fundamental plane for elliptical galaxies . . . . .	67
IV.5	Distribution of the fraction of galaxies containing AGN . . . . .	68
IV.6	Rotation curves for 21 Sc galaxies . . . . .	69
IV.7	Gravitational waveform for a black hole binary merger . . . . .	74
IV.8	Gravitaional wave recoil velocity from black hole mergers . . . . .	76

# **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.

$$M_{\text{vir}} \equiv \frac{4\pi}{3} \Delta_{\text{vir}} \rho_m R_{\text{vir}}^3, \quad (\text{I.1})$$

where  $\rho_m = \Omega_M \rho_c$ .

$$V_{\text{circ}} = \sqrt{\frac{GM(< r)}{r}} \Big|_{\text{max}} \quad (\text{I.2})$$

### I.1.3.2 Density and Concentration

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

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

where  $\rho_0$  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  $c \equiv R_{\text{vir}}/R_s$ , where  $R_{\text{vir}}$  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., 2001). 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  $M_{\text{vir}}$  and maximum circular velocity  $V_{\text{circ}}$ . [citation needed] show that this relation can be found from

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

$$\hat{\rho} = \frac{M_{\text{vir}}}{R_{\text{vir}}^3} = \frac{4\pi}{3} \Delta_{\text{vir}} \rho_c \Omega_M, \quad (\text{I.5})$$

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

where  $x = r/R_s$  and  $x_{\text{max}} = 2.15$ .

At  $z = 0$ ,

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

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

for distinct halos and

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

for subhalos.

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

where  $c_0(z)$  and  $M_0(z)$  are free parameters.

The black lines of Figure I.2 are given by

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

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

### 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-Giguère, 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**

Text goes here.

#### **I.2.1 Simulation Initialization**

Text goes here.

##### **I.2.1.1 Initial Conditions and the Surface of Last Scattering**

Text goes here.

##### **I.2.1.2 The Zel'dovich Approximation**

Text goes here.

##### **I.2.1.3 Second-order Lagrangian Perturbation Theory**

Text goes here.

#### **I.2.2 Dark Matter Halos in N-body Simulations**

Text goes here.

##### **I.2.2.1 Spherical Overdensity**

Text goes here.

##### **I.2.2.2 Friends-of-Friends**

Text goes here.

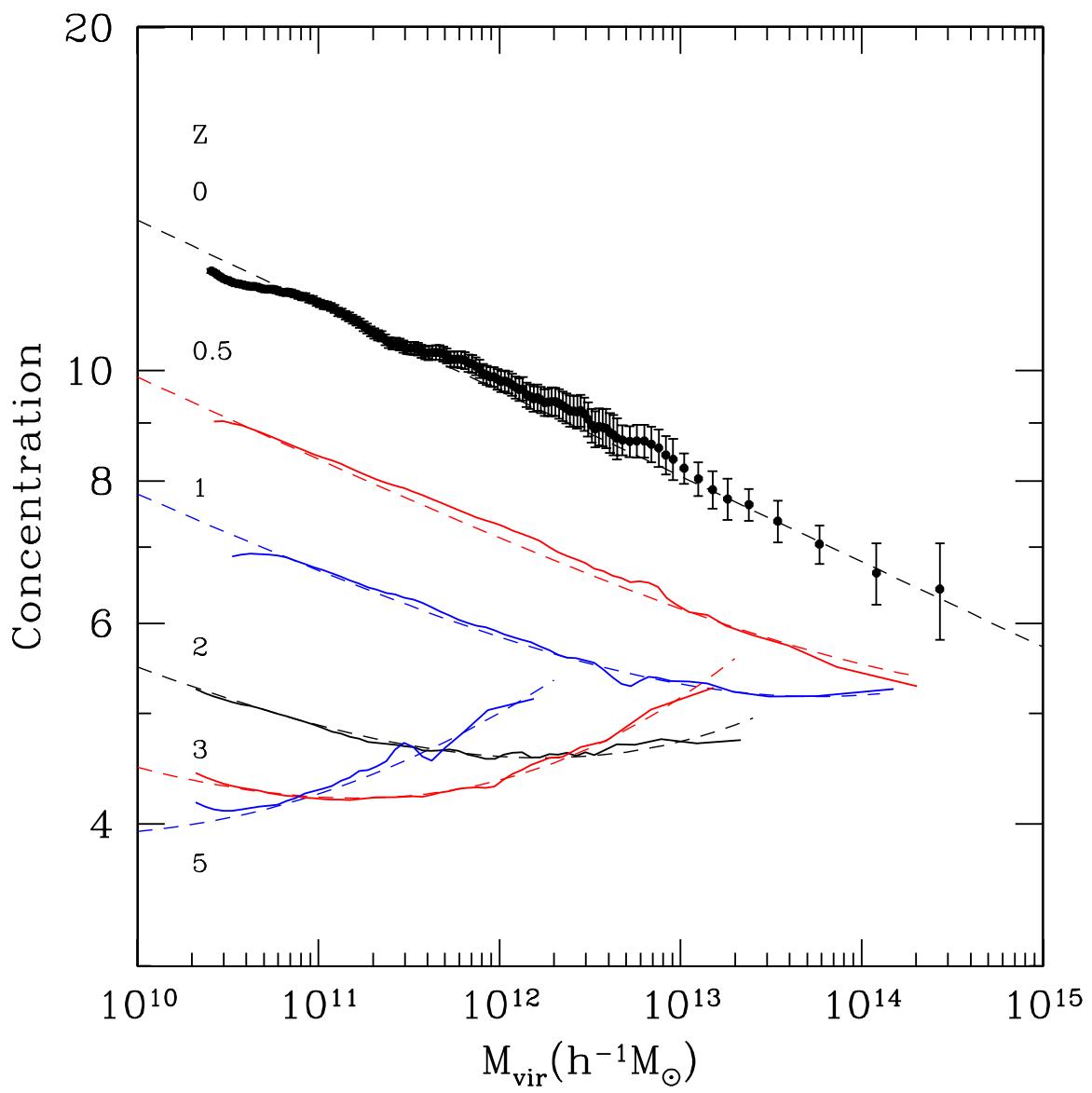


Figure I.1: Caption.

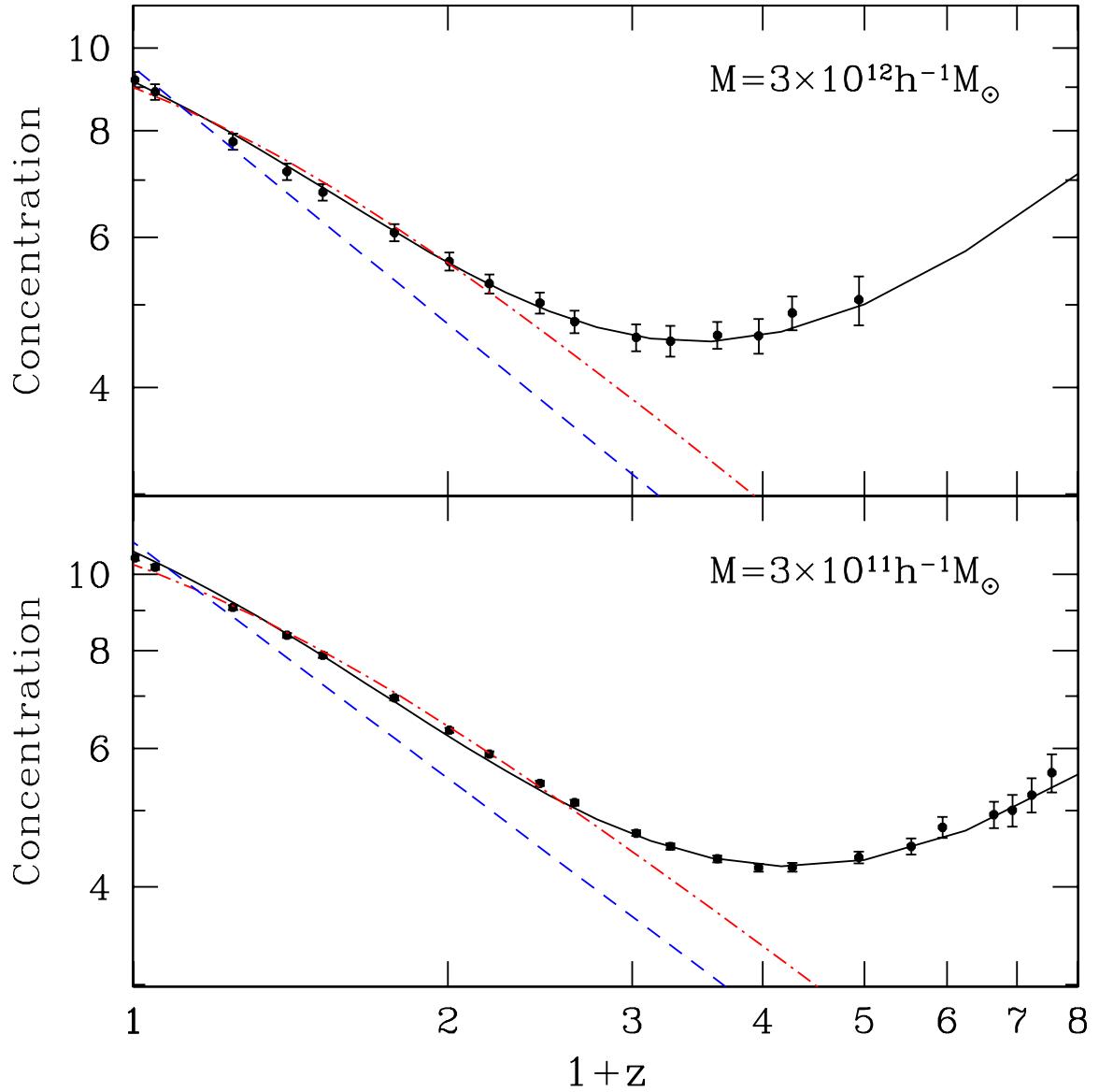


Figure I.2: Caption.

## 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 cosmological N-body simulation code GADGET-2 for the dark matter simulations presented in this work. In this section, we begin with

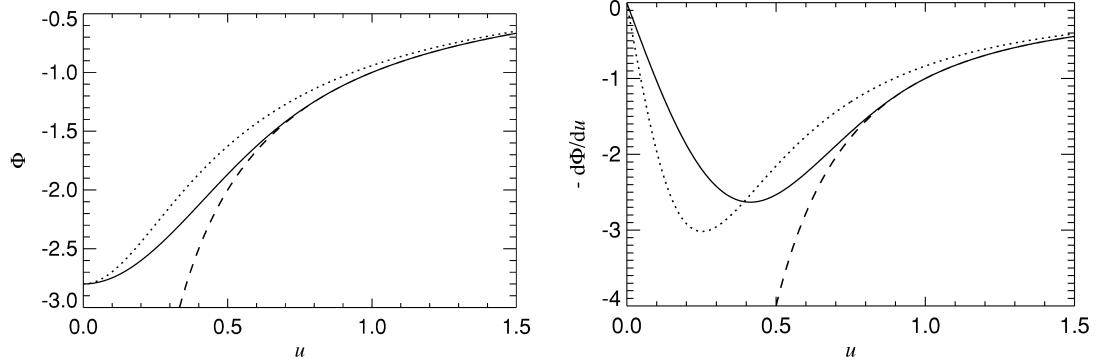


Figure II.1: Potential (*left*) and force (*right*) softening.

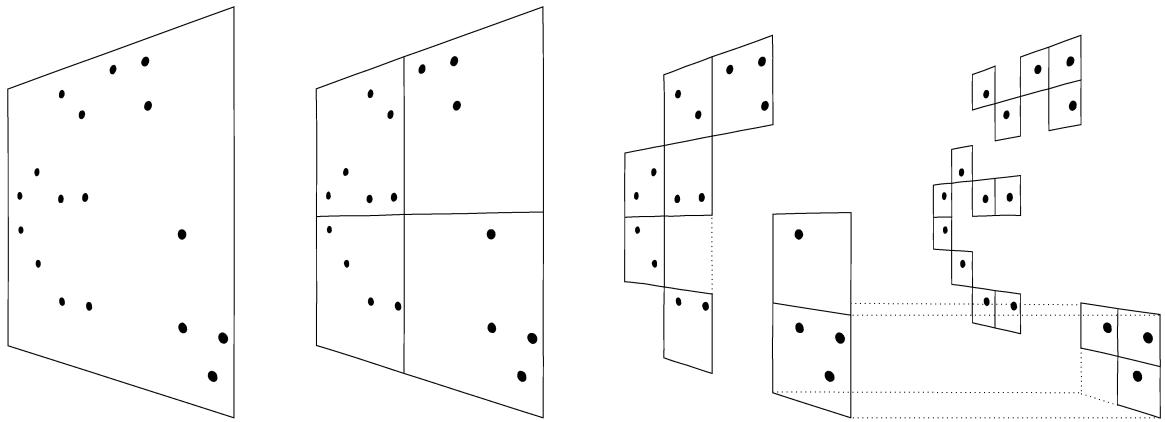


Figure II.2: Barnes-Hut oct-tree in two dimensions.

a discussion of the fundamental concepts presented in the original GADGET code, then proceed to the improvements made in the GADGET-2 code.

### II.2.1 GADGET-2

Text goes here.

#### II.2.1.1 Force Calculation and Softening

Text goes here.

#### II.2.1.2 The Tree Code

Text goes here.

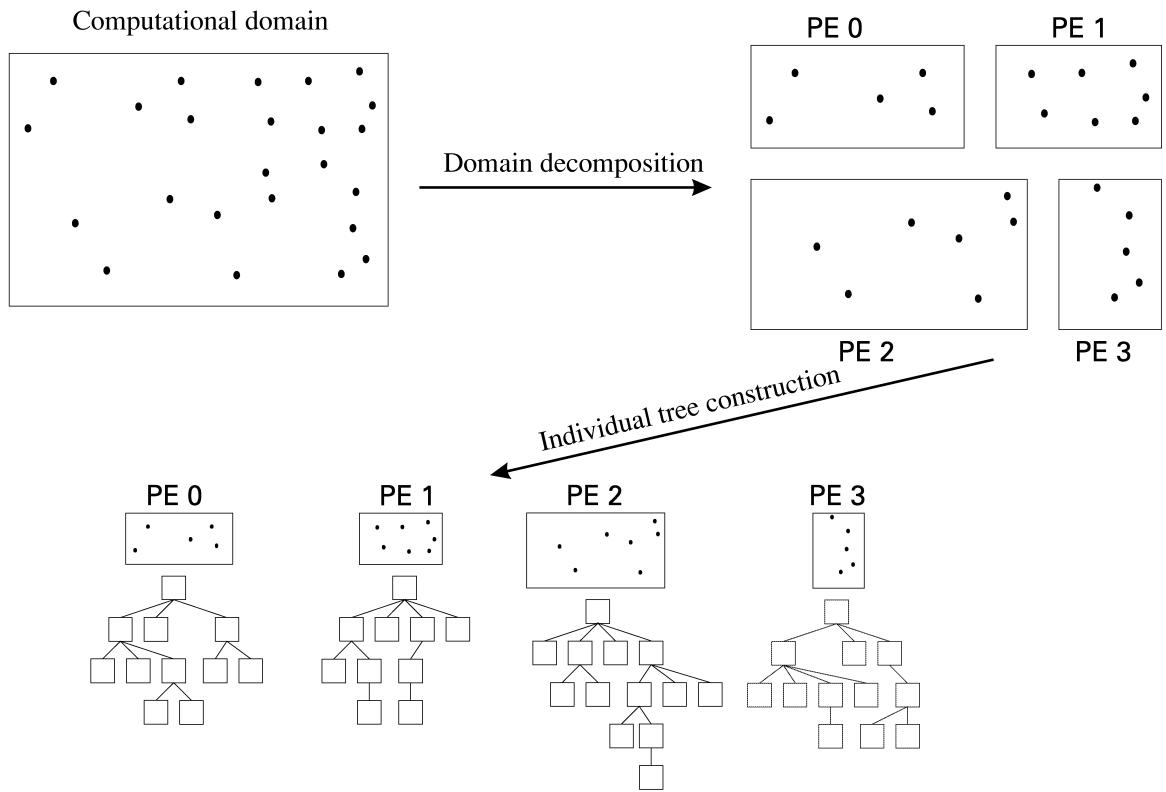


Figure II.3: Domain decomposition.

### II.2.1.3 Parallelization

Text goes here.

### II.2.1.4 Metrics

Text goes here.

## II.2.2 Simulations

Text goes here.

## II.3 Halo Finding with ROCKSTAR

Text goes here.

### II.3.1 Halo Identification

Text goes here.

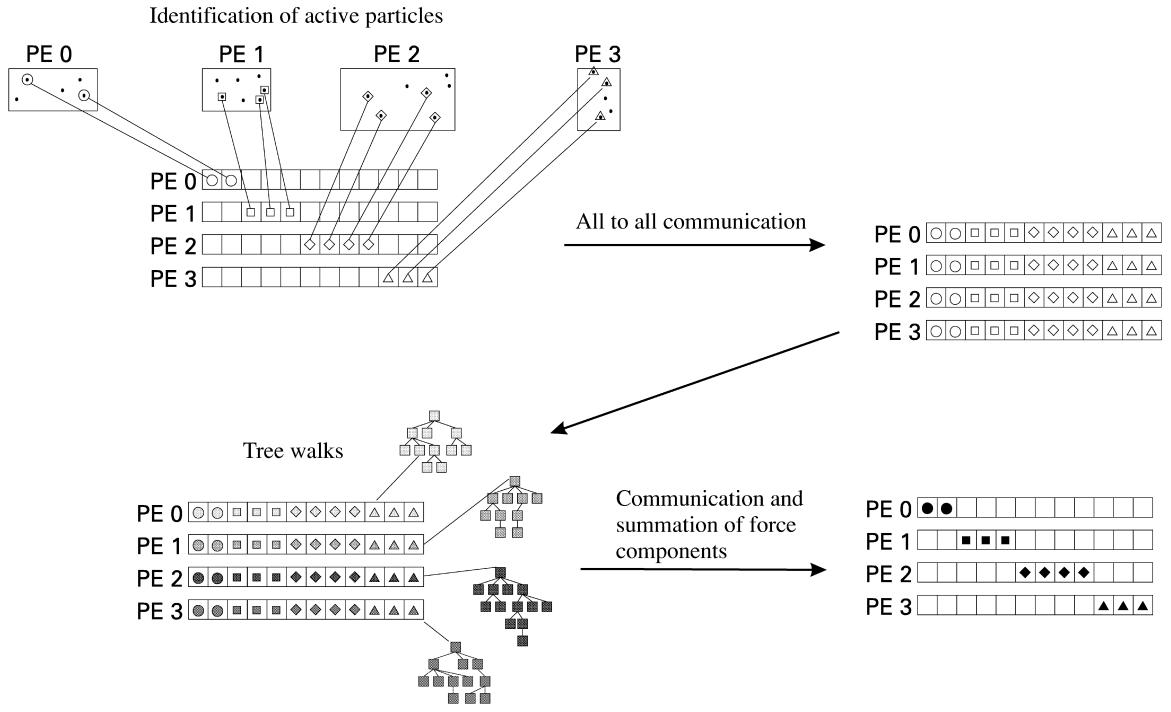


Figure II.4: Force parallelism.

### II.3.2 Halo Properties

Text goes here.

#### II.3.2.1 Mass

Text goes here.

#### II.3.2.2 Concentration

Text goes here.

#### II.3.2.3 Relaxation Parameters

Text goes here.

### II.3.3 Halo Catalogs

Text goes here.

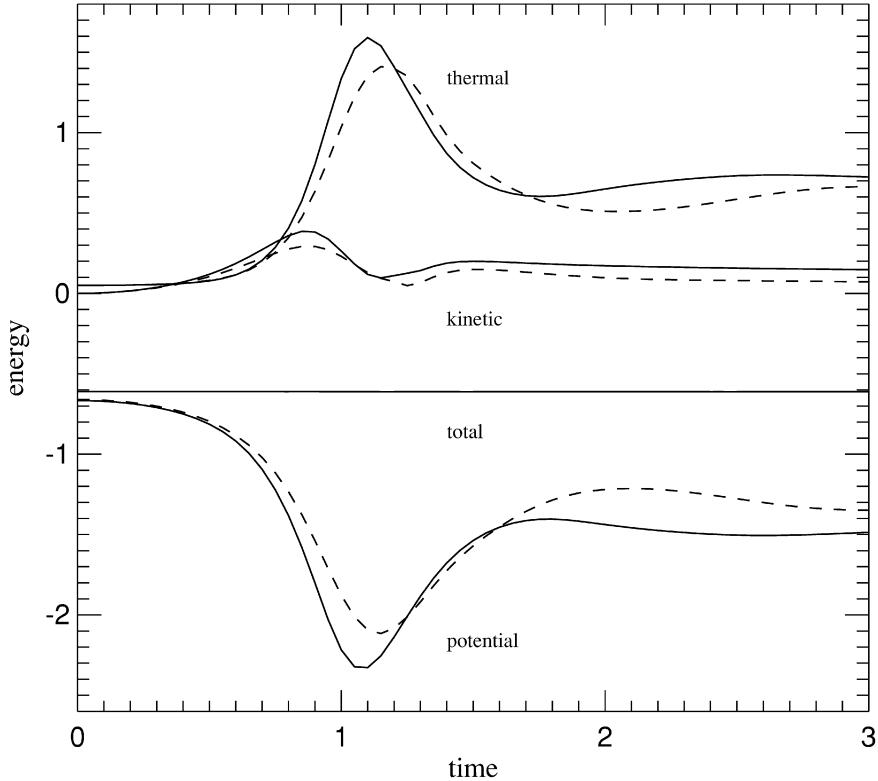


Figure II.5: Energy conservation for an initially isothermal gas sphere.

## II.4 CROSSMATCH

Text goes here.

## 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 its 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 in-

cluded 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  $\rho_0$ .

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 pa-

rameters 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  $R_s > R_{\text{vir}}$  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  $R_{\text{vir}}$ . 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.6, 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  $\rho$  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  $\chi^2$  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 Mass Profiles**

Text goes here. Mass profiles tested to avoid binning issues and the stats stuff Manodeep learned at the conference. Bad fitting results, so abandoned. Figure II.10.

#### **II.5.2.5 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  $\chi^2$  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.

One 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,  $\rho_0$  less than zero, or  $\chi^2$  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**

Text goes here.

### **II.5.4.1 Match Verification**

Text goes here.

### **II.5.4.2 Morphology**

Text goes here.

### **II.5.4.3 Density Profiles**

Text goes here.

## **II.5.5 Difference Distributions**

Text goes here.

### **II.5.5.1 Histograms**

Text goes here.

### **II.5.5.2 Fitting**

Text goes here.

### **II.5.5.3 Mass Quartiles**

Text goes here.

## **II.5.6 Redshift Trends**

Text goes here.

### **II.5.6.1 Mean and Standard Deviation**

Text goes here.

### II.5.6.2 Skew

Text goes here.

### II.5.6.3 Kurtosis

Text goes here.

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

$$s_f = \sqrt{\sum_x \left( \frac{\partial f}{\partial x} \right)^2 s_x^2} \quad (\text{II.1})$$

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

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

with mean  $\mu$ , scale parameter  $\alpha$ , and shape parameter  $\beta$ , has excess kurtosis

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

The gamma function

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \quad (\text{II.4})$$

has the first derivative

$$\Gamma'(x) = \Gamma(x)\psi_0(x) \quad (\text{II.5})$$

where the digamma function  $\psi_0$  is the derivative of the logarithm of the gamma function and is given by

$$\psi_0(x) = \int_0^\infty \left( \frac{e^{-t}}{t} - \frac{e^{-xt}}{1-e^{-t}} \right) dt \quad (\text{II.6})$$

if the real part of  $x$  is positive.

We now apply (II.1) to (III.13) to find the standard deviation of the excess kurtosis:

$$s_{\gamma_2} = \sqrt{\left(\frac{d\gamma_2}{d\beta}\right)^2 s_\beta^2} \quad (\text{II.7})$$

$$= s_\beta \frac{d\gamma_2}{d\beta} \quad (\text{II.8})$$

$$= s_\beta \frac{d}{d\beta} \left[ \frac{\Gamma(5/\beta)\Gamma(1/\beta)}{\Gamma(3/\beta)^2} - 3 \right]. \quad (\text{II.9})$$

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

$$s_{\gamma_2} = s_\beta \frac{d\gamma_2}{dx} \frac{dx}{d\beta} \quad (\text{II.10})$$

$$= s_\beta \left( -\frac{1}{\beta^2} \right) \frac{d}{dx} \left[ \frac{\Gamma(5x)\Gamma(x)}{\Gamma(3x)} - 3 \right] \quad (\text{II.11})$$

$$= -s_\beta x^2 \left\{ \frac{\Gamma(3x)^2 \frac{d}{dx} [\Gamma(5x)\Gamma(x)] - \Gamma(5x)\Gamma(x) \frac{d}{dx} [\Gamma(3x)^2]}{\Gamma(3x)^4} \right\} \quad (\text{II.12})$$

$$= -s_\beta \frac{x^2}{\Gamma(3x)^4} \{ \Gamma(3x)^2 [5\Gamma(5x)\psi_0(5x)\Gamma(x) + \Gamma(5x)\Gamma(x)\psi_0(x)] - \Gamma(5x)\Gamma(x)[6\Gamma(3x)^2\psi_0(3x)] \} \quad (\text{II.13})$$

$$= s_\beta \frac{x^2}{\Gamma(3x)^4} \{ 6\Gamma(5x)\Gamma(3x)^2\Gamma(x)\psi_0(3x) - \Gamma(5x)\Gamma(3x)^2\Gamma(x)[5\psi_0(5x) + \psi_0(x)] \} \quad (\text{II.14})$$

$$= s_\beta \frac{x^2}{\Gamma(3x)^4} \{ \Gamma(5x)\Gamma(3x)^2\Gamma(x)[6\psi_0(3x) - 5\psi_0(5x) - \psi_0(x)] \} \quad (\text{II.15})$$

$$= s_\beta x^2 \frac{\Gamma(5x)\Gamma(x)}{\Gamma(3x)^2} [6\psi_0(3x) - 5\psi_0(5x) - \psi_0(x)]. \quad (\text{II.16})$$

Substituting back in for  $x$  and recognizing an occurrence of  $\gamma_2$ , we have the result

$$s_{\gamma_2} = s_\beta \frac{1}{\beta^2} (\gamma_2 + 3) [6\psi_0(3/\beta) - 5\psi_0(5/\beta) - \psi_0(1/\beta)] \quad (\text{II.17})$$

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

## **II.5.7 2-D Histograms**

Text goes here.

### **II.5.7.1 Binning and Plotting**

Text goes here.

### **II.5.7.2 Fitting**

Text goes here.

## **II.5.8 Alternate Difference Distributions**

Text goes here.

### **II.5.8.1 Equivalent Displacement**

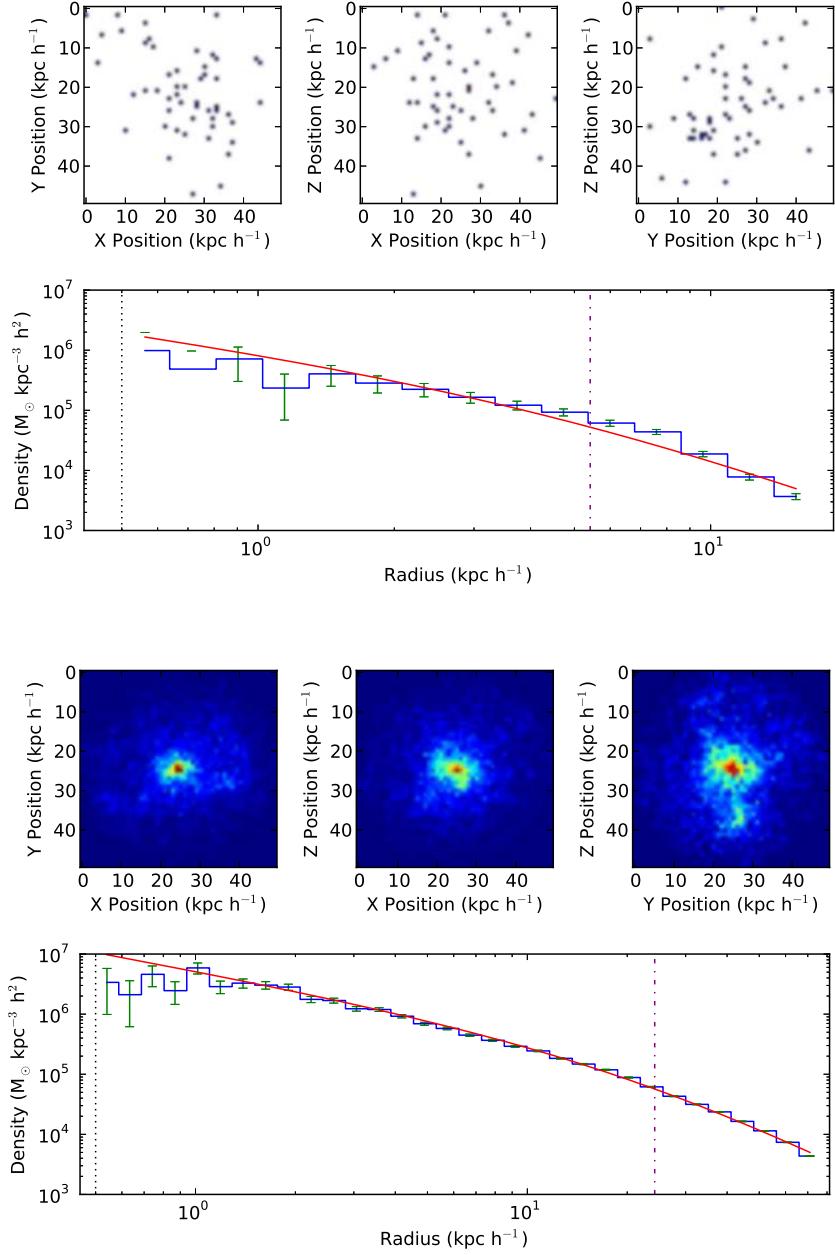
Text goes here.

### **II.5.8.2 Redshift Trends**

Text goes here.

## **II.6 Automation**

Text goes here.



**Figure II.6:** Density profiles for two large halos at  $z = 14$  and  $z = 6$ . Both halos are from the Box 1 2LPT simulation, and are the largest halos at their respective redshifts. ([Get rid of density projections.](#))

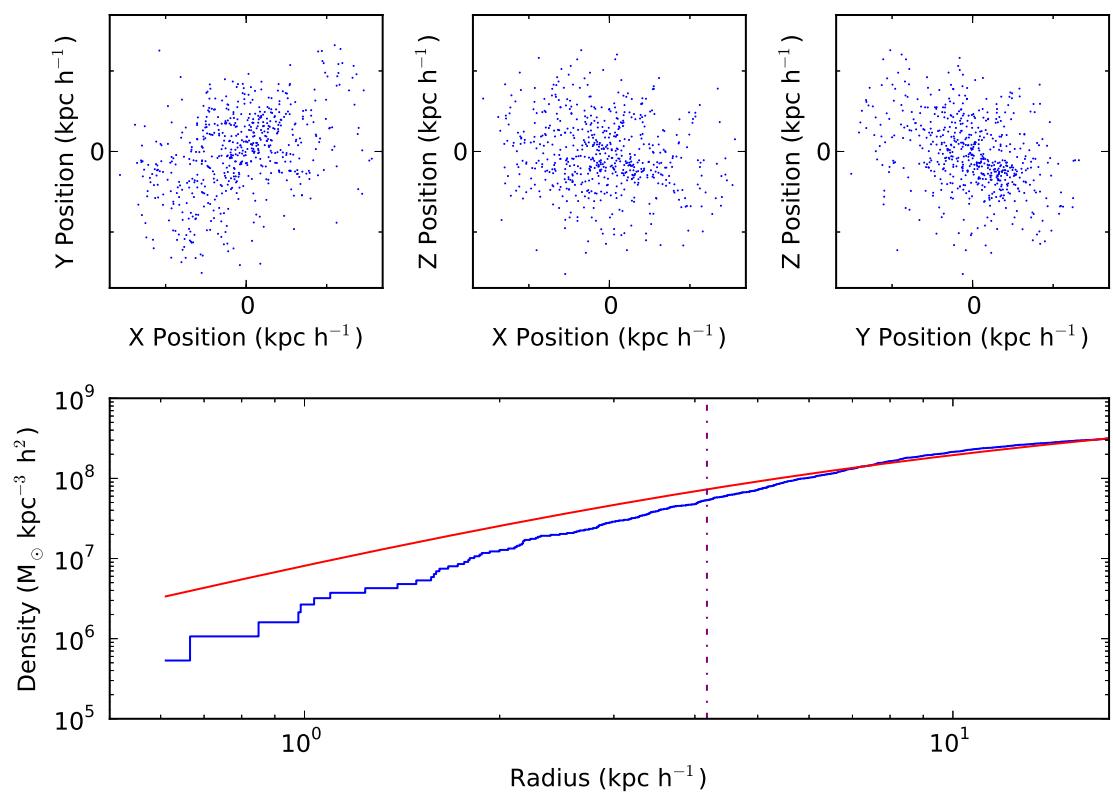
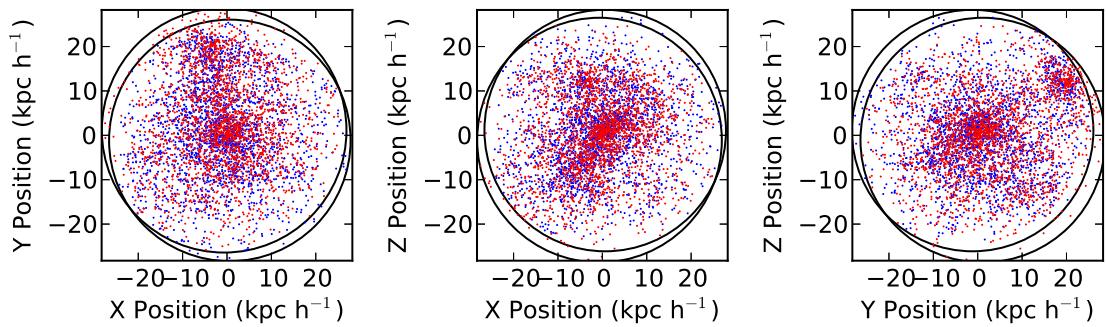


Figure II.7: Mass profiles for a large halo at  $z = 6$ . (Get rid of density projections.)



**Figure II.8:** 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.

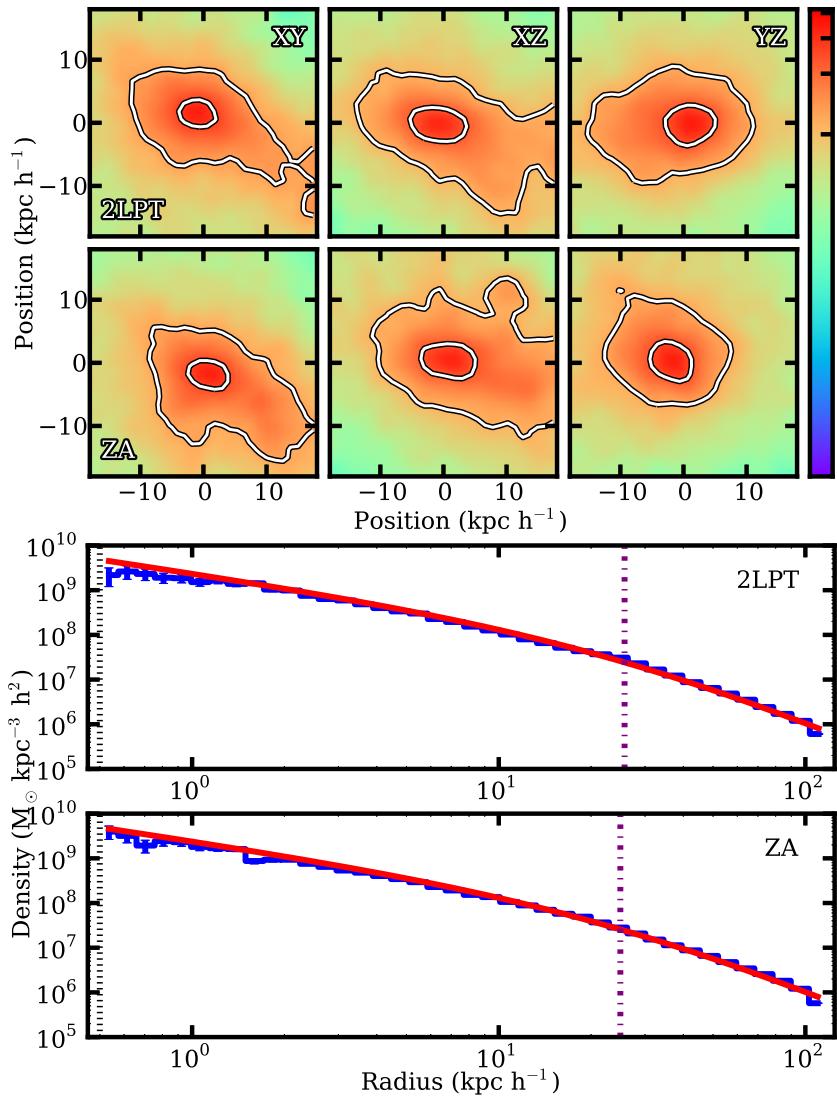


Figure II.9: Two large matched halos at  $z = 6$ .

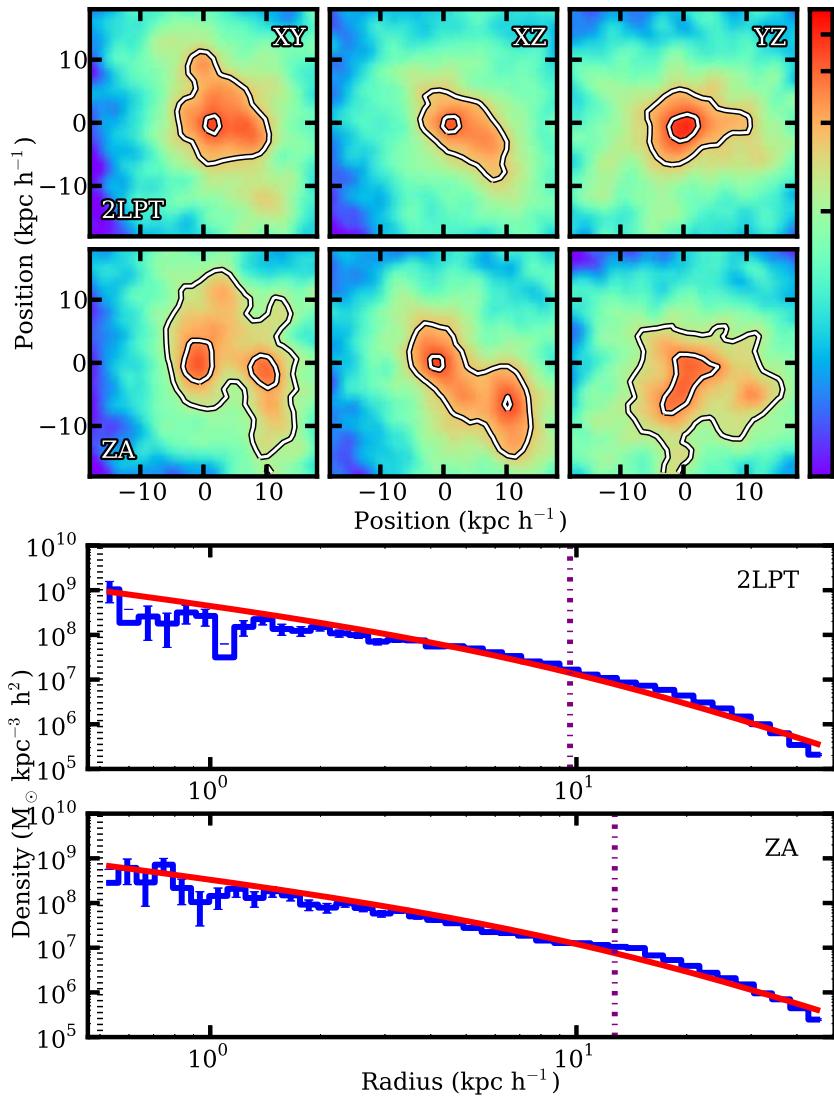


Figure II.10: Two large matched halos at  $z = 6$  with differing nuclear structure.

## 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  $\mu_{\Delta M_{\text{vir}}} = (7.88 \pm 0.17) \times 10^3 z - (3.07 \pm 0.14) \times 10^{-2}$ . 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 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-Giguère, 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  $\Omega_m$  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  $\Delta$  commonly ranges from  $\sim 100$  to  $\sim 500$ . 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

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

where  $\rho_0$  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  $c \equiv R_{\text{vir}}/R_s$ .  $R_{\text{vir}}$  is the halo virial radius, which is often defined as the radius at which the average interior density is some factor  $\Delta_c$  times the critical density of the universe  $\rho_c$ , where  $\Delta_c$  is typically  $\sim 200$ . 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., 2001). 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; Sirk, 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  $\Psi(q)$  is applied to the initial positions  $q$  to yield the Eulerian final comoving positions

$$x = q + \Psi. \quad (\text{III.2})$$

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

$$x = q - D_1 \nabla_q \phi^{(1)} + D_2 \nabla_q \phi^{(2)}, \quad (\text{III.3})$$

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

$$v = -D_1 f_1 H \nabla_q \phi^{(1)} + D_2 f_2 H \nabla_q \phi^{(2)}, \quad (\text{III.4})$$

with Hubble constant  $H$  and  $f_i = d \ln D_i / d \ln a$ , with expansion factor  $a$ . The relations

$f_1 \approx \Omega^{5/9}$  and  $f_2 \approx 2\Omega^{6/11}$ , with matter density  $\Omega$ , 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  $\Lambda$ CDM initial conditions, as  $\Omega$  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

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

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

$$\nabla_q^{(2)}(q) = \delta^{(2)}(q). \quad (\text{III.6})$$

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

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

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- $\sigma$  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 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 Numerical 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  $\Lambda$ CDM 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 sep-

aration must be considered. The initial RMS displacement  $\Delta_{\text{rms}}$  is given by

$$\Delta_{\text{rms}}^2 = \frac{4\pi}{3} \int_{k_f}^{k_{\text{Ny}}} P(k, z_{\text{start}}) dk, \quad (\text{III.8})$$

where  $k_f = 2\pi/L_{\text{box}}$  is the fundamental mode,  $L_{\text{box}}$  is the simulation box size,  $k_{\text{Ny}} = \frac{1}{2}Nk_f$  is the Nyquist frequency of an  $N^3$  simulation, and  $P(k, z_{\text{start}})$  is the power spectrum at starting redshift  $z_{\text{start}}$ . In order to avoid the “orbit crossings” that reduce the accuracy of the initial conditions,  $\Delta_{\text{rms}}$  must be some factor smaller than the mean particle separation  $\Delta_p = L_{\text{box}}/N$  (Holley-Bockelmann et al., 2012). For example, making orbit crossing a  $\sim 10\sigma$  event imposes  $\Delta_{\text{rms}}/\Delta_p = 0.1$ . However, for small-volume, high-resolution simulations, this quickly leads to impractical starting redshifts. Continuing our example, satisfying  $\Delta_{\text{rms}}/\Delta_p \sim 0.1$  for a  $10h^{-1}$  Mpc,  $512^3$  simulation suggests  $z_{\text{start}} \approx 799$ . 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  $\Delta_{\text{rms}}/\Delta_p = 0.25$  yields  $z_{\text{start}} = 300$ , 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 sim-

ulations, 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  $R_{\text{vir}}$ . Here,  $R_{\text{vir}}$  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  $\Delta q$ , the normalized difference in  $q$  between halos in the 2LPT and ZA simulations, defined as

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

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

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

where  $\mu$  is the mean,  $\alpha$  is the scale parameter,  $\beta$  is the shape parameter, and  $\Gamma$  is the gamma function

$$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x} dx. \quad (\text{III.11})$$

The shape parameter  $\beta$  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

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

and excess kurtosis

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

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  $\beta$  according to Equation III.13. Following the standard procedure for propagation of uncertainty, we calculate the standard deviation of the kurtosis as

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

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

The derivative of the gamma function is

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

where the digamma function  $\psi_0$  is the derivative of the logarithm of the gamma function

and is given by

$$\psi_0(x) = \int_0^\infty \left( \frac{e^{-t}}{t} - \frac{e^{-xt}}{1-e^{-t}} \right) dt \quad (\text{III.17})$$

if the real part of  $x$  is positive. Now, taking the derivative of  $\gamma_2$  and doing a bit of algebra yields

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

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

In addition to distributions of  $\Delta q$ , we also consider distributions of

$$\Delta'q = \frac{q_{\text{2LPT}} - q_{\text{ZA}}}{q_{\text{ZA}}} \quad (\text{III.19})$$

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

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

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

$$q_{\text{2LPT}} = xq_{\text{ZA}}. \quad (\text{III.21})$$

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

$$\Delta'q_2 = \frac{1}{\Delta'q_1 + 1} - 1, \quad (\text{III.22})$$

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  $\sim 10$  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  $M_{\text{vir}}$  and concentration  $c$ . We plot histograms of  $\Delta M_{\text{vir}}$  and  $\Delta c$  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  $\Delta M_{\text{vir}}$  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  $\mu_{\Delta M_{\text{vir}}} = (1.79 \pm 0.31) \times 10^{-2}$ .

The higher-order moments of the  $\Delta M_{\text{vir}}$  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  $\Delta M_{\text{vir}}$  histograms,  $\Delta c$  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  $\Delta M_{\text{vir}}$ , 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

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

	<i>A</i>	<i>B</i>
$\Delta M_{\text{vir}}$	$(7.88 \pm 0.17) \times 10^{-3}$	$(-3.07 \pm 0.14) \times 10^{-2}$
$\Delta c$	$(3.62 \pm 0.95) \times 10^{-3}$	$(-2.34 \pm 0.84) \times 10^{-2}$

very similar or very discrepant concentrations.

### 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  $\Delta M_{\text{vir}}$  and  $\Delta c$  as functions of redshift. Uncertainty in the mean is estimated directly by the least squares fitting routine.

The mean for  $\Delta M_{\text{vir}}$  is positive and highest at high redshift, trending toward zero by the end of the simulation. Distributions for  $\Delta c$  retain means close to and consistent with zero. Standard deviation decreases slightly for both  $\Delta M_{\text{vir}}$  and  $\Delta c$ . 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  $\Delta M_{\text{vir}}$  and from  $0.73 \pm 0.11$  to  $0.551 \pm 0.026$  for  $\Delta c$ .

We find least square linear fits for both mean  $\Delta M_{\text{vir}}$  vs  $z$  and mean  $\Delta c$  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  $\Delta M_{\text{vir}}$ , with a slope  $\sim 46\sigma$  from zero. Conversely, the slope for  $\Delta c$  is much smaller and, considering the larger spread of the underlying distributions, can be considered negligible. For  $\Delta M_{\text{vir}}$ , 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  $\Delta M_{\text{vir}}$  and  $\Delta c$ , 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  $\Delta M_{\text{vir}}$  and  $\Delta c$  starts highest at high redshift—0.19 for  $\Delta M_{\text{vir}}$  and 0.57 for  $\Delta c$  at  $z = 15$ —and steadily decreases throughout the simulation, reaching minimums of 0.11 for  $\Delta M_{\text{vir}}$  and 0.45 for  $\Delta c$  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  $\varepsilon = 10\%$  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  $17.5 \pm 2.4$  at redshift 10 for  $\Delta M_{\text{vir}}$  and  $15.4 \pm 1.0$  at the end of the simulation at redshift 6 for  $\Delta c$ . Skew is positive for much of the simulation for mass, but is much smaller for concentration. We find average skews of  $0.39 \pm 0.29$  for  $\Delta M_{\text{vir}}$  and  $0.045 \pm 0.028$  for  $\Delta c$ . 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  $\Delta M_{\text{vir}}$  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  $\Delta M_{\text{vir}}$  and  $\Delta c$  as a function of average halo mass  $M_{\text{vir,avg}} = (M_{\text{vir,2LPT}} + M_{\text{vir,ZA}})/2$  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  $\Delta M_{\text{vir}}$  tends to increase with increasing  $M_{\text{vir,avg}}$  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  $\Delta M_{\text{vir}} = -(0.5 \pm 1.5) \times 10^{-2} \log(M_{\text{vir,avg}}) + (0.15 \pm 0.12)$ . The slope of the fit lines then become positive and trends back towards zero as we progress in redshift, with a fit of  $\Delta M_{\text{vir}} = (3.49 \pm 0.99) \times 10^{-3} \log(M_{\text{vir,avg}}) - (6.8 \pm 8.3) \times 10^{-3}$  by  $z = 6$ .

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

	<i>A</i>	<i>B</i>
$\Delta M_{\text{vir}}$	$(9.4 \pm 2.4) \times 10^{-4}$	$(-1.8 \pm 1.8) \times 10^{-3}$
$\Delta c$	$(-7.3 \pm 1.9) \times 10^{-3}$	$(3.7 \pm 1.4) \times 10^{-2}$

We additionally find a trend for more massive halo pairs to be more concentrated in ZA. This trend is somewhat stronger than for  $\Delta M_{\text{vir}}$ , but again, high  $z$  snapshots differ from the trend. The fit equations for  $z = 15$  and  $z = 6$  are  $\Delta c = -(0.256 \pm 0.093) \log(M_{\text{vir,avg}}) + (2.07 \pm 0.76)$  and  $\Delta c = -(1.10 \pm 0.31) \times 10^{-2} \log(M_{\text{vir,avg}}) + (0.103 \pm 0.026)$ , 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  $\Delta M_{\text{vir}}$  by a factor of  $\sim 2$ . 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  $\Delta c$  distributions symmetric about zero.

The slopes of the fits to the  $\Delta q$  vs.  $M_{\text{vir,avg}}$  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  $\Delta q$  dependence on  $M_{\text{vir,avg}}$  with increasing redshift, except for the highest  $z$  snapshots, which seem to reverse the trend. Coefficients  $A$  and  $B$  for the fit equation  $\text{Slope} = Az + B$  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  $\Delta M_{\text{vir}}$  and  $z \gtrsim 13$  for  $\Delta c$ , 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  $\Delta q$  rely on the average quantity  $q_{\text{avg}} = (q_{\text{2LPT}} + q_{\text{ZA}})/2$  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  $q_{\text{ZA}}$  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  $\Delta' M_{\text{vir}}$  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  $\Delta' M_{\text{vir}}$  of  $9 \times 10^{-2}$  at  $z = 15$  to  $3 \times 10^{-2}$  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  $\Delta M_{\text{vir}}$  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  $\Delta M_{\text{vir}}$  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 simu-

lations 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 sub-halo 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  $R_s$  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  $(10 \text{ Mpc})^3$ . 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:

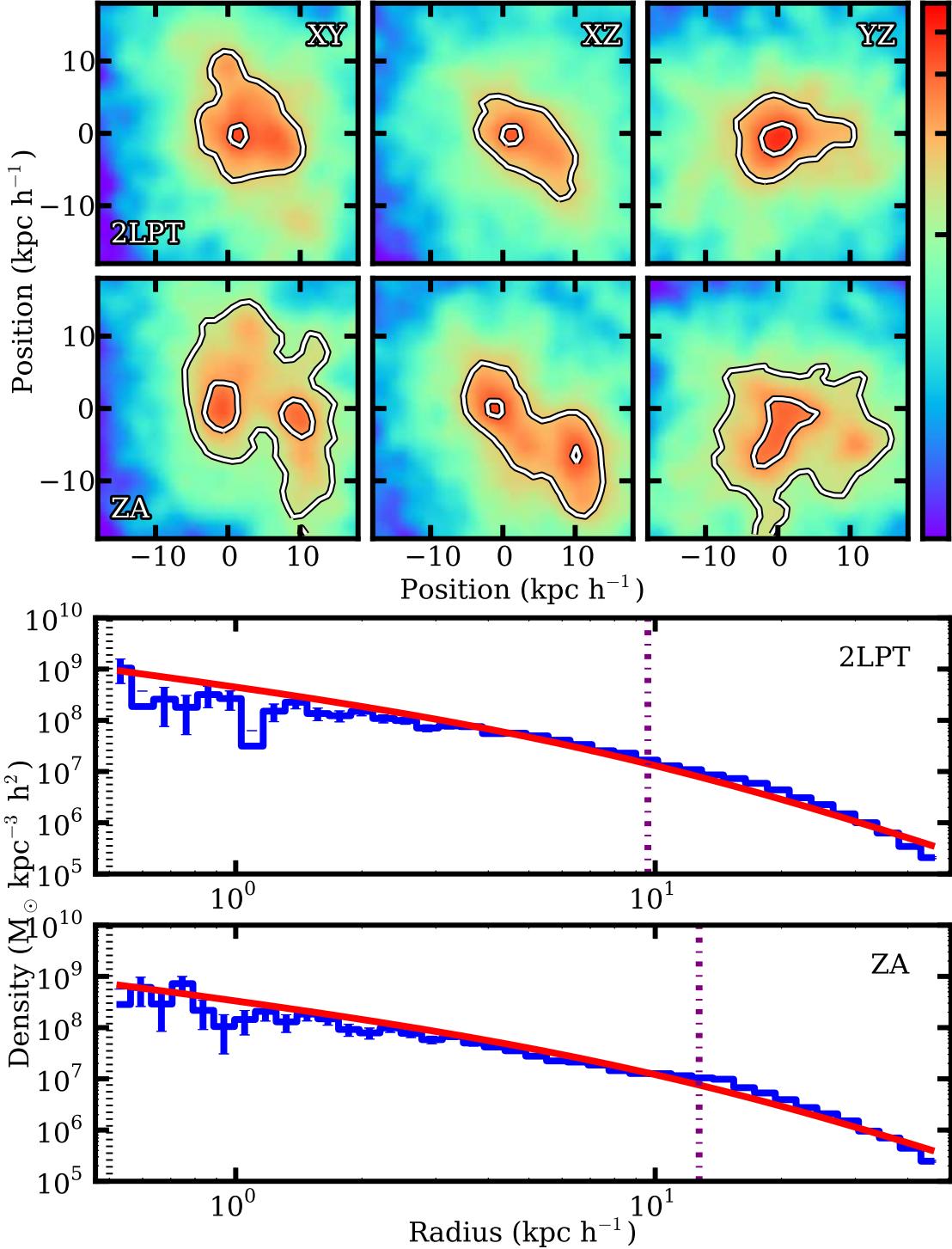
- 2LPT halos get a head start in the formation process and grow faster than their ZA counterparts. Companion halos in 2LPT and ZA simulations may have offset merger epochs and differing nuclear morphologies.
- 2LPT halos are, on average, more massive than ZA halos. At  $z = 15$ , the mean of the  $\Delta M_{\text{vir}}$  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  $\Delta M_{\text{vir}}$  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  $\Delta M_{\text{vir}} = (7.88 \pm 0.17) \times 10^{-3}z - (3.07 \pm 0.14) \times 10^{-2}$ .
- 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  $\Delta M_{\text{vir}} = (1.03 \pm 0.46) \times 10^{-2} \log(M_{\text{vir,avg}}) - (2.6 \pm 3.8) \times 10^{-2}$  for  $z = 10$ . By  $z = 6$ , this has flattened to  $\Delta M_{\text{vir}} = (3.49 \pm 0.99) \times 10^{-3} \log(M_{\text{vir,avg}}) - (6.8 \pm 8.3) \times 10^{-3}$ . As a function of redshift, the slopes of these equations are fit by  $\text{Slope} = (9.4 \pm 2.4) \times 10^{-4}z - (1.8 \pm 1.8) \times 10^{-3}$ .
- Halo concentration, on average, is similar for 2LPT and ZA halos. However, even by the end of the dark ages, the width of the  $\Delta c$  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  $\Delta c$ . 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  $\Delta c = -(0.256 \pm 0.093) \log(M_{\text{vir,avg}}) + (2.07 \pm 0.76)$  at  $z = 15$  and  $\Delta c = -(1.10 \pm 0.31) \times 10^{-2} \log(M_{\text{vir,avg}}) - (0.103 \pm 0.026)$  at  $z = 6$ . The slopes of these equations, as a function of redshift, are fit by  $\text{Slope} = -(7.3 \pm 1.9) \times 10^{-3}z + (3.7 \pm 1.4) \times 10^{-2}$ . This is not visible in the symmetrical  $\Delta c$  distributions, as the trends are roughly centered about zero and are washed away when integrated across the entire mass range.

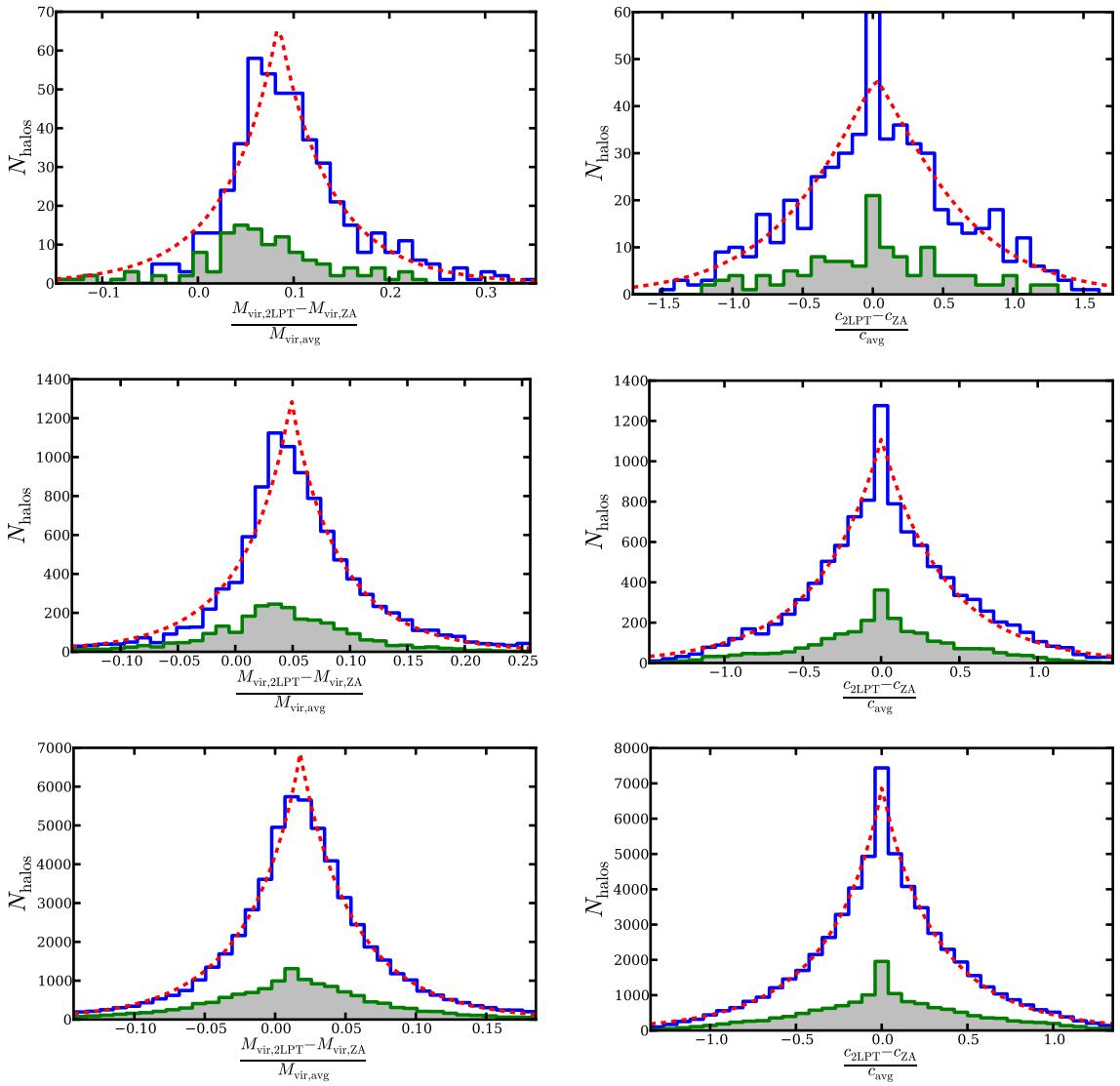
We have found that choice of initialization technique can play a significant role in the properties of halo populations during the pre-reionization dark ages. The early halo growth displayed 2LPT simulations, or conversely the delayed halo growth arising from the approximations made in ZA-initialized simulations, makes careful attention to simulation ini-

tialization 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.

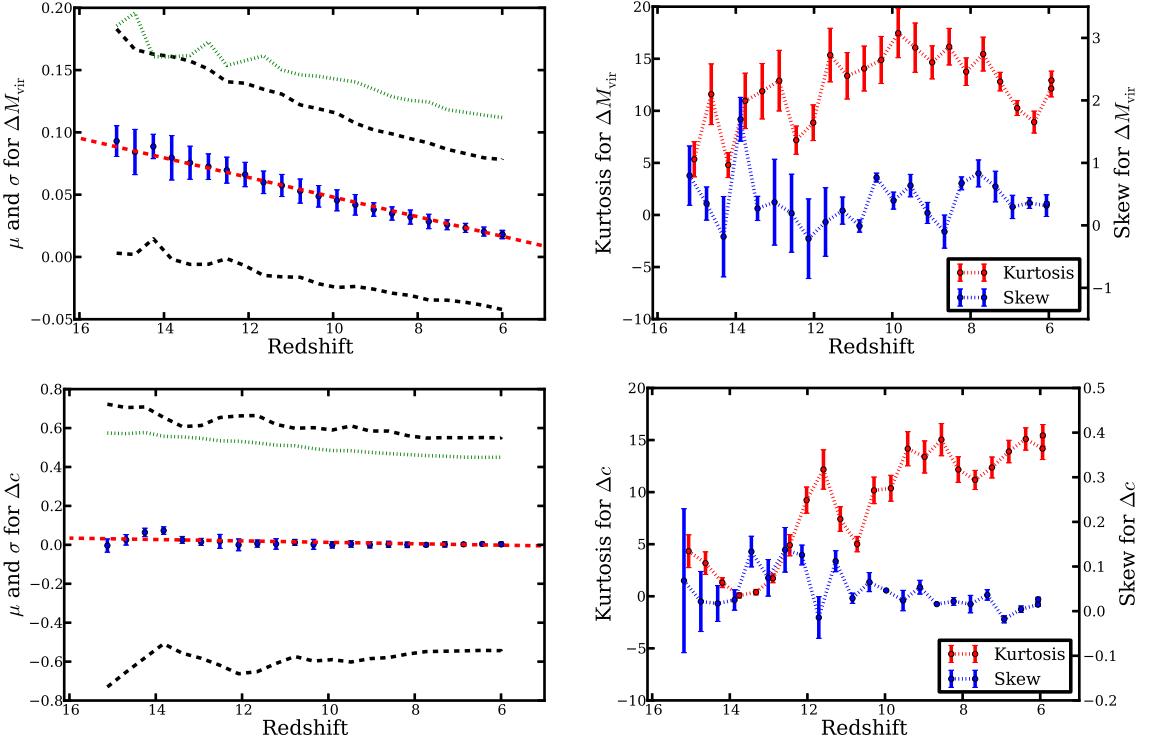
This work was conducted using the resources of the Advanced Computing Center for Research and Education (ACCRE) at Vanderbilt University, Nashville, TN. We also acknowledge the support of the NSF CAREER award AST-0847696. We would like to thank the referee for helpful comments, as well as the first author's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



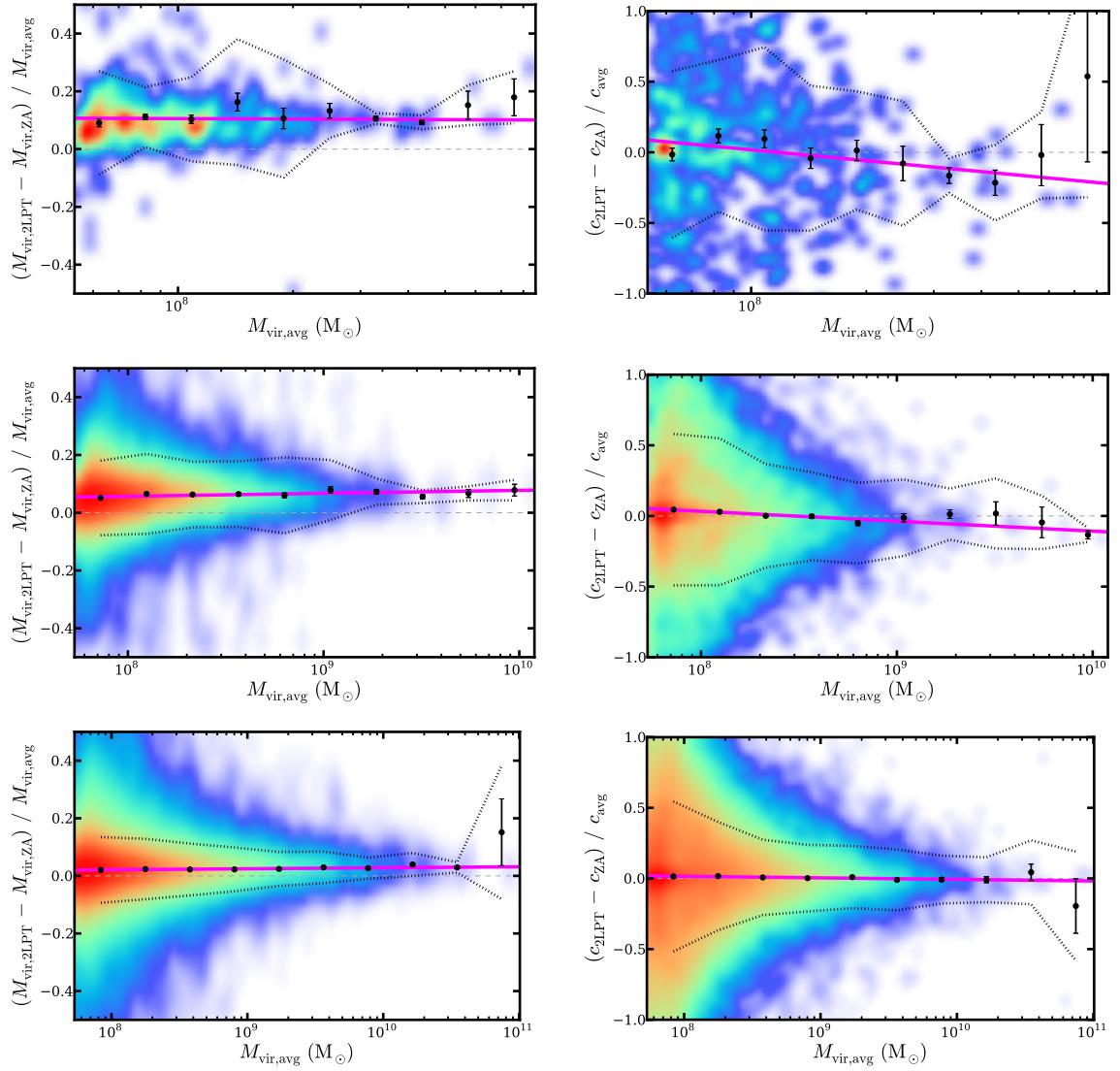
**Figure III.1:** Top two rows: Density projections for two matching halos at  $z = 6$ . The first and second row are 2LPT and ZA, respectively. The halos appear to be either undergoing or have recently undergone a major merger. The 2LPT halo appears to be more relaxed and further along in the merger 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.



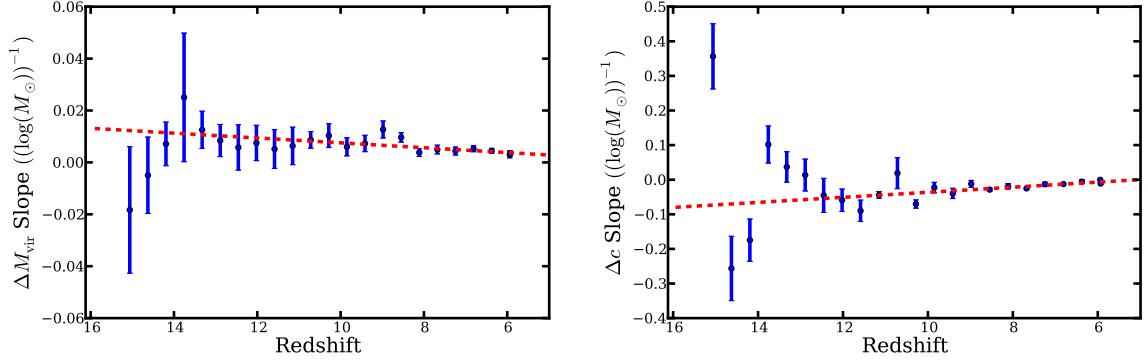
**Figure III.2:** Histograms of  $\Delta M_{\text{vir}}$  (left column) and  $\Delta c$  (right column) for snapshots at  $z = 14.7$ ,  $z = 10.3$ , and  $z = 6.0$  (top, middle, and bottom panels, respectively). The small gray-filled histograms count only the top 25% most massive halos. The main histograms are fit with a generalized normal distribution with parameters for mean, scale, and shape, overplotted as the red dashed line (see Equation III.10). The distributions for  $\Delta M_{\text{vir}}$  have positive means and heavier 2LPT halos, with the most pronounced difference at high redshift. The distributions shown here have means of  $(8.4 \pm 1.8) \times 10^{-2}$ ,  $(4.87 \pm 0.87) \times 10^{-2}$ , and  $(1.79 \pm 0.31) \times 10^{-2}$ , respectively. The skew of the distribution is also the most positive at high redshift, and shifts toward symmetry by  $z = 6$ . The  $\Delta c$  distributions remain symmetric about zero and have negligible skew. The means are consistent with zero, at  $(2.6 \pm 2.7) \times 10^{-2}$ ,  $(0.2 \pm 2.6) \times 10^{-2}$ , and  $(0.3 \pm 1.1) \times 10^{-2}$ , respectively. Both distributions have excess kurtosis consistently larger than that of a standard Gaussian distribution, with a sharp peak and heavy tails.



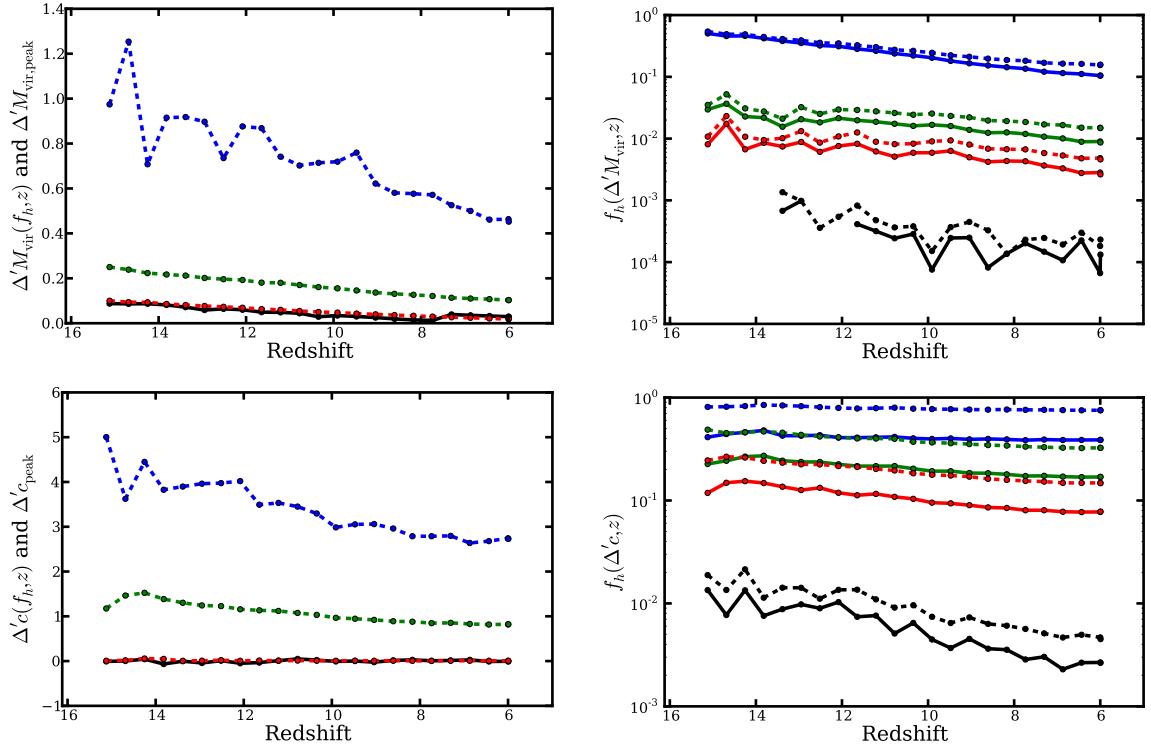
**Figure III.3:** Mean, standard deviation, and RMS (*left column*) and skew and excess kurtosis (*right column*) as functions of redshift for  $\Delta M_{\text{vir}}$  (*top row*) and  $\Delta c$  (*bottom row*). In the left column,  $\mu$  is plotted as blue points, and  $\mu \pm \sigma$  is plotted 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  $\mu$  for  $\Delta M_{\text{vir}}$  to be more positive at higher redshift and gradually shift toward zero as the simulation progresses, with a fit function of  $\mu_{\Delta M_{\text{vir}}} = (7.88 \pm 0.17) \times 10^{-3}z - (3.07 \pm 0.14) \times 10^{-2}$ . The mean for  $\Delta c$ , 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  $\Delta M_{\text{vir}}$  and  $\Delta c$  distributions narrow over time, with a slight decrease in  $\sigma$ . In the right column, we plot skew (blue line) and excess kurtosis (red line). Skew is positive for much of the simulation for  $\Delta M_{\text{vir}}$ , but is much smaller for  $\Delta c$ . Kurtosis is large (much more peaked than Gaussian) for both  $\Delta M_{\text{vir}}$  and  $\Delta c$  throughout much of the simulation, and especially at later redshift.



**Figure III.4:**  $\Delta M_{\text{vir}}$  (left column) and  $\Delta c$  (right column) as functions of  $M_{\text{vir,avg}}$ . 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 lines 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 three rows again correspond to snapshots at  $z = 14.7$ ,  $z = 10.3$ , and  $z = 6.0$ . We again see the overall offset for positive  $\Delta M_{\text{vir}}$  as before, and additionally find a small tendency for more massive halo pairs to be more likely to have even larger  $\Delta M_{\text{vir}}$ . Fit equations for the left column panels are  $\Delta M_{\text{vir}} = -(0.5 \pm 1.5) \times 10^{-2} \log(M_{\text{vir,avg}}) + (0.15 \pm 0.12)$ ,  $\Delta M_{\text{vir}} = (1.03 \pm 0.46) \times 10^{-2} \log(M_{\text{vir,avg}}) - (2.6 \pm 3.8) \times 10^{-2}$ , and  $\Delta M_{\text{vir}} = (3.49 \pm 0.99) \times 10^{-3} \log(M_{\text{vir,avg}}) - (6.8 \pm 8.3) \times 10^{-3}$ , respectively. Concentration show an opposite trend for more massive halos to be less concentrated in 2LPT than in ZA. The right column panels have fit equations  $\Delta c = -(0.256 \pm 0.093) \log(M_{\text{vir,avg}}) + (2.07 \pm 0.76)$ ,  $\Delta c = -(7.0 \pm 1.2) \times 10^{-2} \log(M_{\text{vir,avg}}) + (0.595 \pm 0.099)$ , and  $\Delta c = -(1.10 \pm 0.31) \times 10^{-2} \log(M_{\text{vir,avg}}) + (0.103 \pm 0.026)$ , respectively.



**Figure III.5:** Slopes of the  $\Delta q$  vs.  $M_{\text{vir,avg}}$  fit functions. The left and right panels correspond to the  $\Delta M_{\text{vir}}$  and  $\Delta c$  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  $\Delta M_{\text{vir}}$  and negative and decreasing slope with redshift for  $\Delta c$ . We find fit equations of  $\text{Slope} = (9.4 \pm 2.4) \times 10^{-4}z - (1.8 \pm 1.8) \times 10^{-3}$  for  $\Delta M_{\text{vir}}$  and  $\text{Slope} = -(7.3 \pm 1.9) \times 10^{-3}z + (3.7 \pm 1.4) \times 10^{-2}$  for  $\Delta c$ . Snapshots at very high redshift,  $z \gtrsim 14$  for  $\Delta M_{\text{vir}}$  and  $z \gtrsim 13$  for  $\Delta c$ , 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  $\Delta'M_{\text{vir}}$  (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  $\Delta M_{\text{vir}}$ ,  $\Delta'M_{\text{vir}}$  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  $\Delta'M_{\text{vir}}$ . *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 of 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.

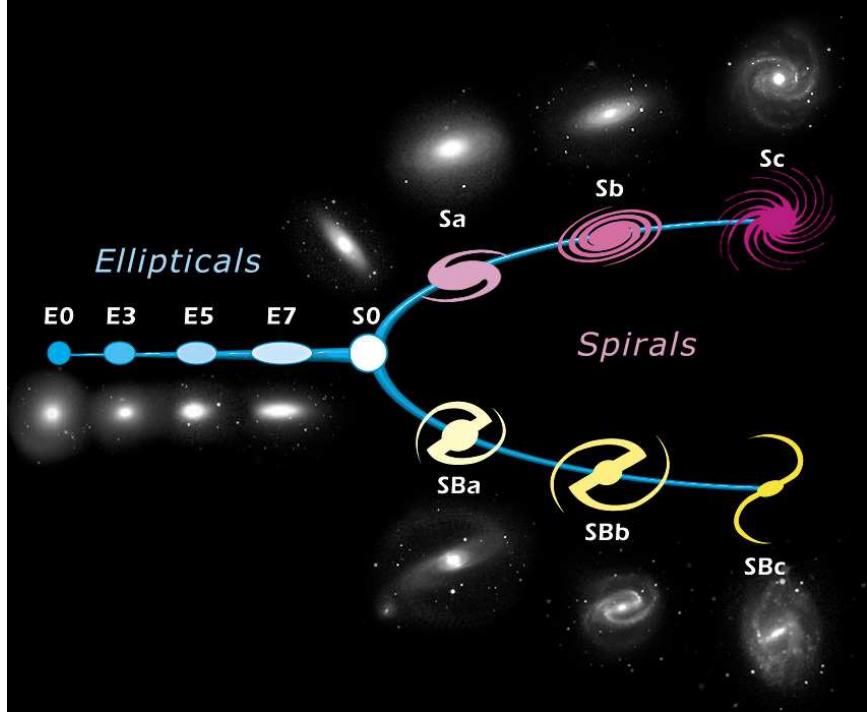


Figure IV.1: The Hubble tuning fork. On the left of the diagram are elliptical galaxies. E0 galaxies are the most spherical, while E7 are the most flattened or elongated. S0 are lenticular galaxies. The top branch on the right are spiral galaxies with no bar, while the bottom right branch are spiral 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.

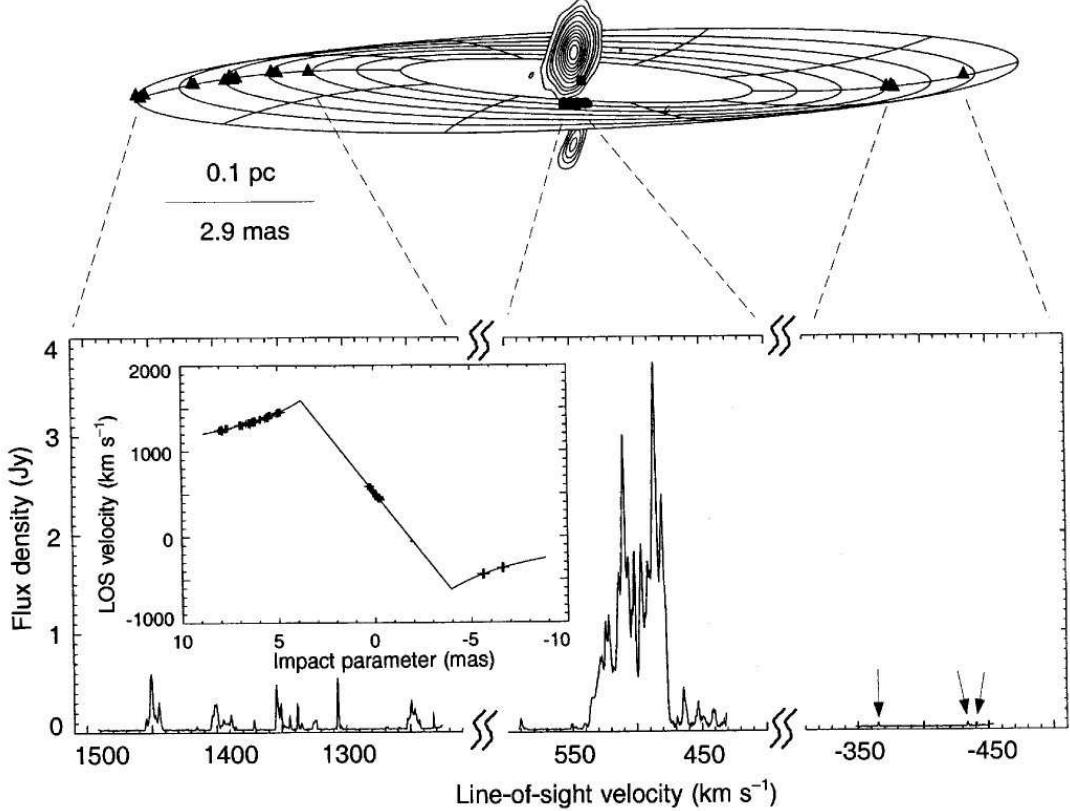


Figure IV.2: Maser orbits fit to a warped disk for NGC4258. Masers can also be useful for distance determinations. Here, the positions and velocities of water masers are able to be fit to a warped disk model surrounding a supermassive black hole. This allows the interpolation of physical radii away from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

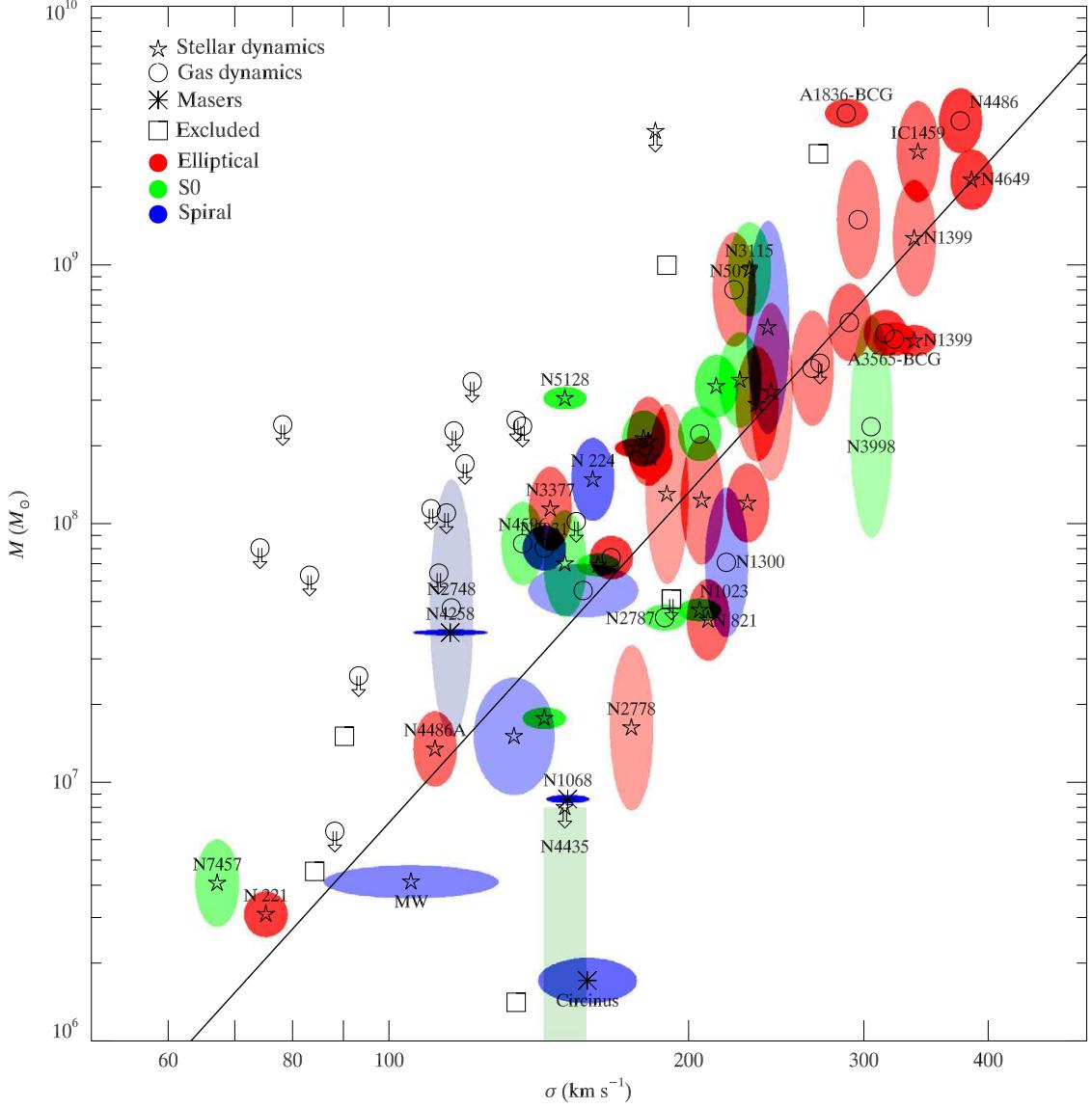
### IV.1.3 Correlations

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  $\sigma$  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  $(\alpha, \beta, \varepsilon_0) = (8.12 \pm 0.08 M_\odot, 4.24 \pm 0.41 M_\odot, 0.44 \pm 0.06 M_\odot)$  for all galaxies and  $(\alpha, \beta, \varepsilon_0) = (8.23 \pm 0.08 M_\odot, 3.96 \pm 0.42 M_\odot, 0.31 \pm$

$0.06M_{\odot}$ ) for ellipticals, where  $\varepsilon_0$  is the intrinsic scatter in the relation.



**Figure IV.3:** The M- $\sigma$  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 elliptical 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  $M_{BH} = 10^{8.12} M_{\odot}(\sigma/200 \text{ km s}^{-1})^{4.24}$ . 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  $\mu_B$ , and  $\sigma_0$  is the velocity dispersion in  $\text{km s}^{-1}$  (Binney & Merrifield, 1998).

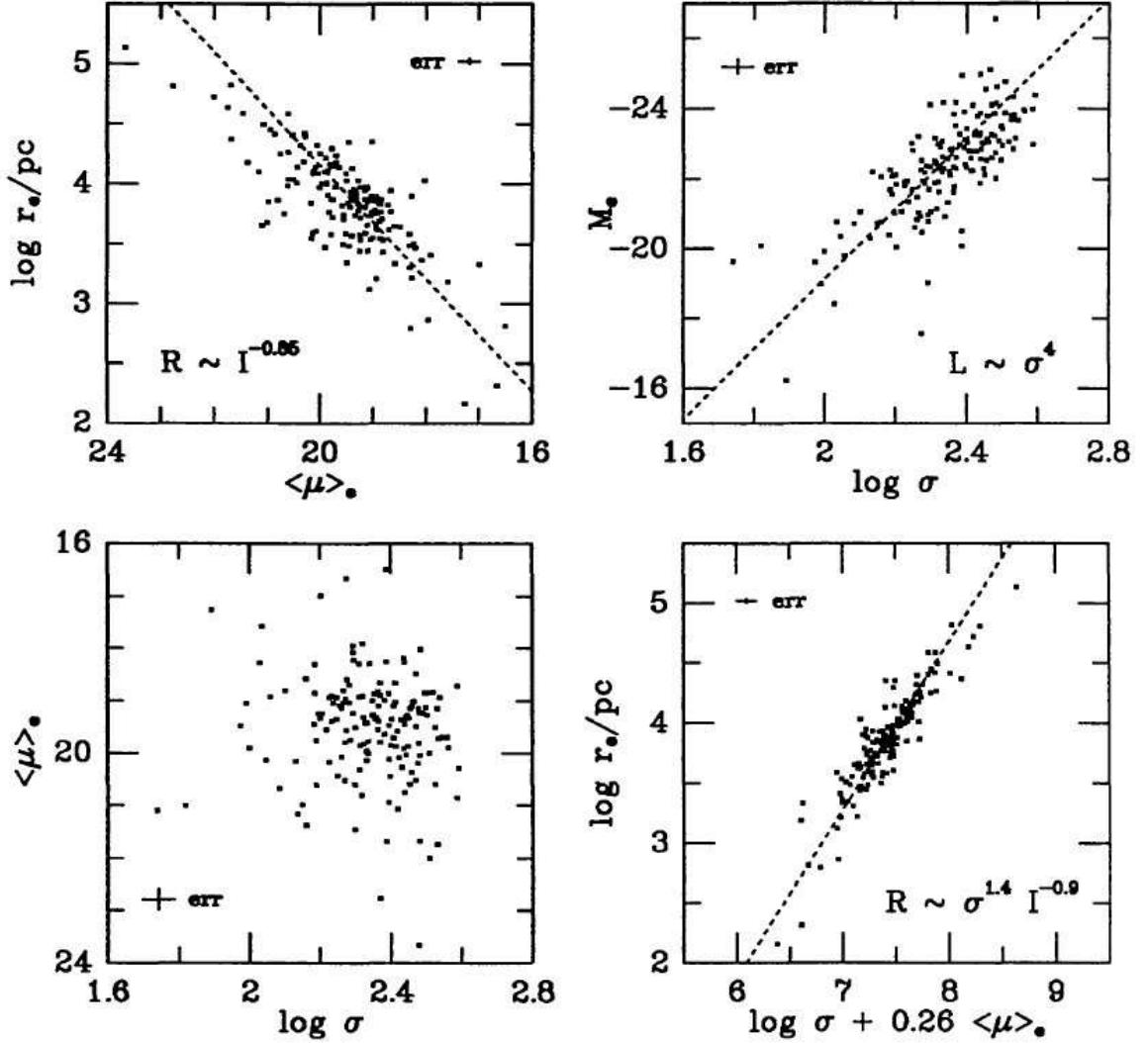


Figure IV.4: The fundamental plane for elliptical galaxies. *Top panels:* The top panels show the one-parameter scaling relations, with the relation between radius and mean surface brightness on the left and the relation between luminosity and velocity dispersion (the Faber-Jackson relation) on the right. *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.

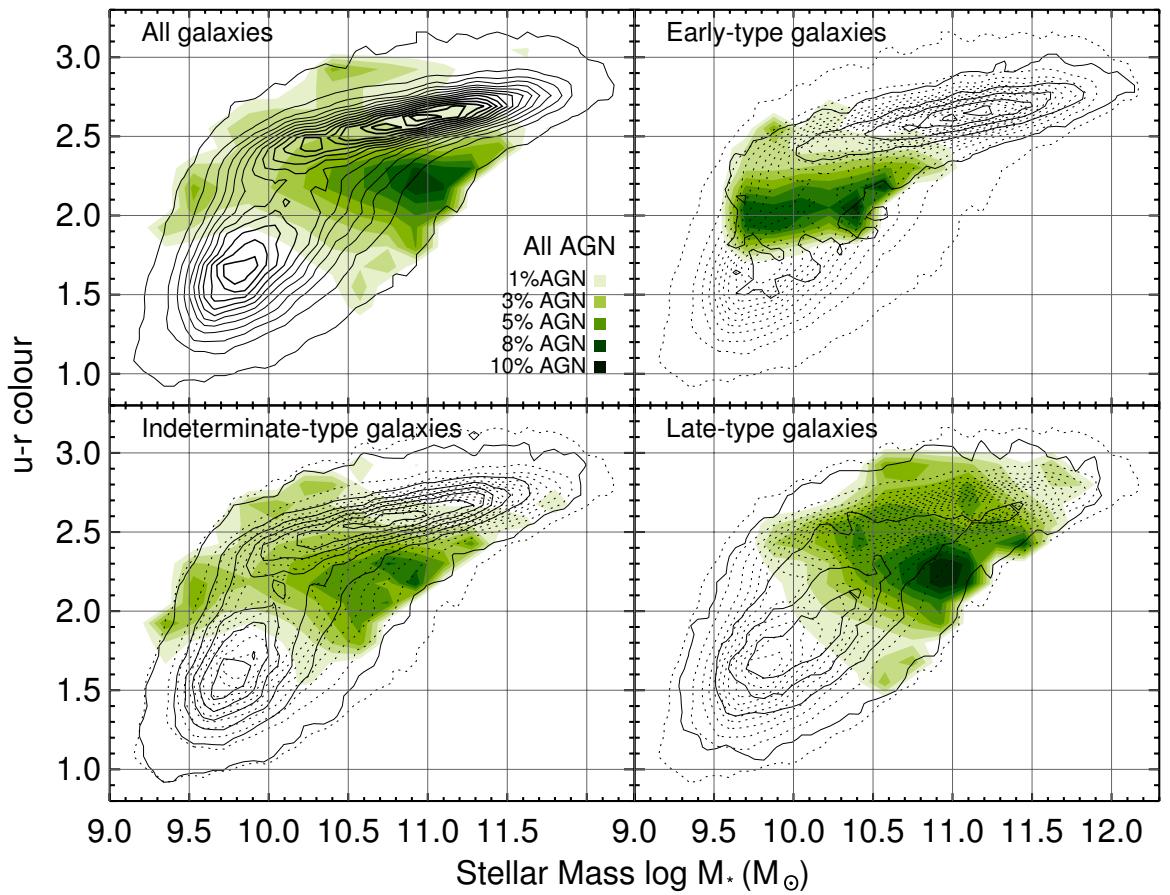


Figure IV.5: Distribution of the fraction of galaxies containing AGN. Galaxy color in  $u-r$  is plotted vs stellar mass. The contours are the galaxy population for all galaxies (top-left), early-type galaxies (top right), intermediate-type galaxies (bottom left), and late type galaxies (bottom right). 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 matter 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 gravitationally. 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.

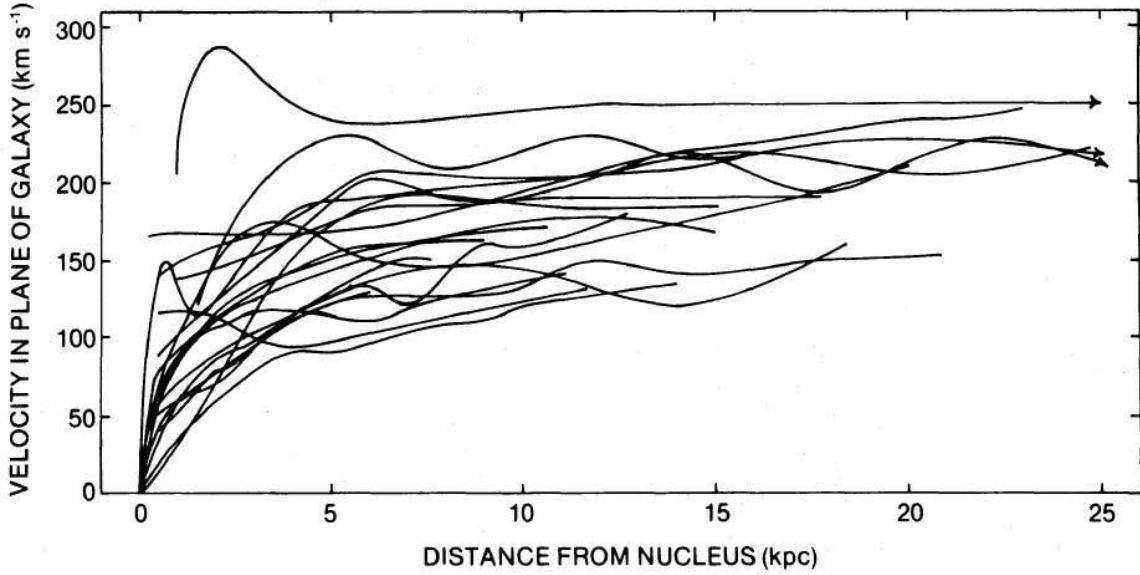


Figure IV.6: Rotation curves for 21 Sc galaxies. It is readily identifiable that the rotation curves do not fall off as would be expected for galaxies without a dark matter component. (Rubin et al., 1980)

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

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

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  $\sim 10^9\text{--}10^{10} M_\odot$  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

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

where  $v_M$  is the velocity of the black hole,  $M$  is its mass,  $\rho$  is the density of surrounding matter,  $\text{erf}$  is the error function,  $\ln \Lambda$  is the Coulomb logarithm, and  $X \equiv v_M/(\sqrt{2}\sigma)$  where  $\sigma$  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.

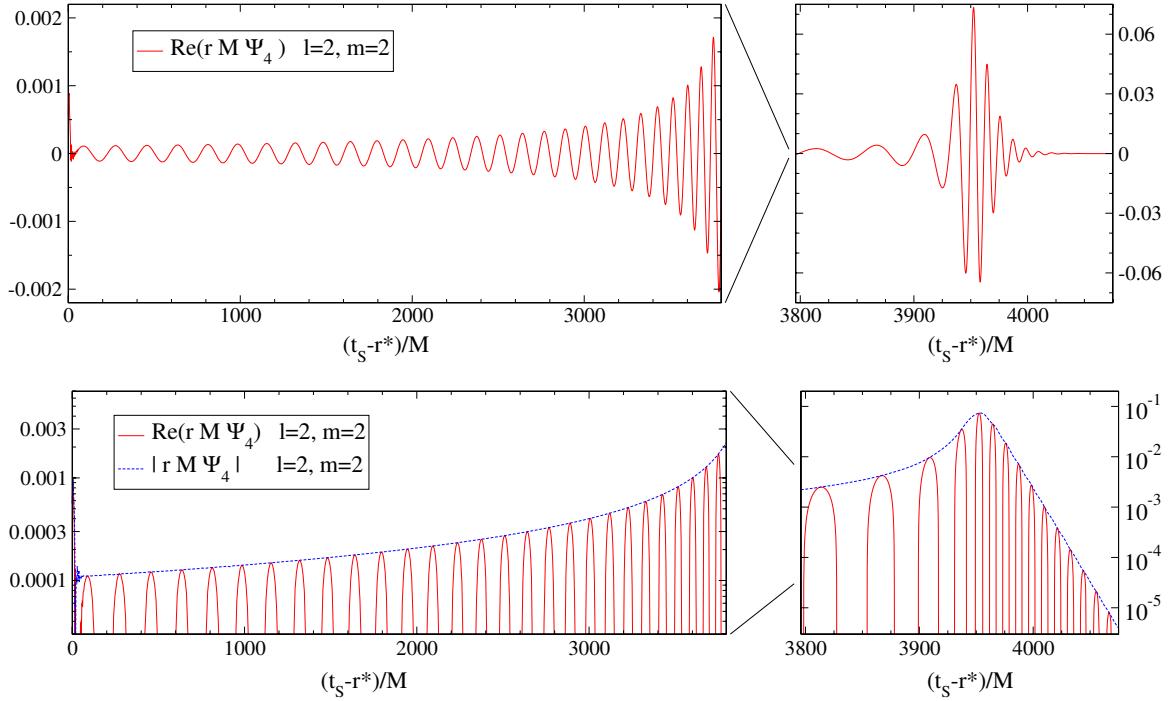
#### **IV.3.1.3 Gravitational Waves and Recoil Kicks**

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

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

where

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

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

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

Here, the fitting constants are  $A = 1.2 \times 10^4 \text{ km s}^{-1}$ ,  $B = -0.93$ ,  $H = (7.3 \pm 0.3) \times 10^3 \text{ km s}^{-1}$ , and  $K = (6.0 \pm 0.1) \times 10^4 \text{ km s}^{-1}$ . 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  $\Theta$ ,  $\Theta_0$ , and  $\xi$  (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  $\sim 4000 \text{ km s}^{-1}$ . 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.

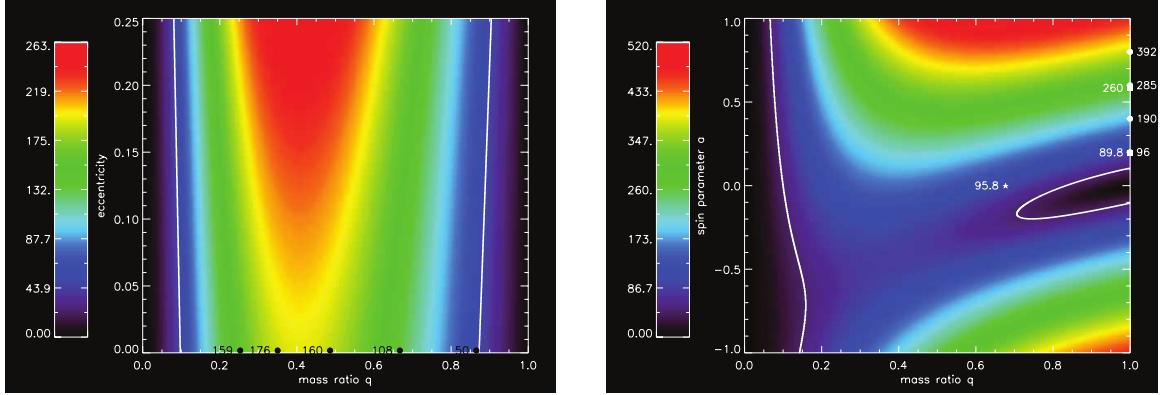


Figure IV.8: *Left:* Gravitaional wave recoil velocity from a merger of nonspinning black holes as a function of eccentricity and mass ratio. Data from numerical relativity simulations (González et al., 2007) are overlaid along the zero eccentricity line. The overlaid white contours are the escape velocity of a typical globular cluster,  $50 \text{ km s}^{-1}$ . *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  $50 \text{ km s}^{-1}$  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,

$$\sigma < \sigma_{HL} = \frac{2GM}{v_\infty^2}, \quad (\text{IV.7})$$

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

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

where  $v_\infty$  and  $\rho_\infty$  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

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

where  $\alpha$  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

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

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)

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

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 formula for the accretion rate,

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

#### 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

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

where  $c$  is the speed of light,  $m_H$  is the mass of hydrogen, and  $\sigma_T$  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  $M-\sigma$  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 over-abundance 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  $M-\sigma$  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
- Barkana, R., & Loeb, A. 2001, *Phys. Rep.*, 349, 125
- 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
- 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., Kolatt, T. S., Sigad, Y., et al. 2001, *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
- 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
- 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
- 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. 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
- 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
- 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
- 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
- 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
- Zel'dovich, Y. B. 1970, *Astron. Astrophys.*, 5, 84
- 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)

```
1 #Rockstar Halo Finder
2 #Parallel config file for multi-cpu, multi-snapshot halo finding
3 #Note that periodic boundary conditions are assumed for NUM_WRITERS > 1.
4 #See README for details.
5
6 #Once compiled ("make"), run Rockstar server as
7 # ./rockstar -c parallel.cfg
8 #Then launch the reading/analysis tasks with:
9 # ./rockstar -c auto-rockstar.cfg
10 #You will have to launch at least NUM_BLOCKS+NUM_WRITERS processes.
11
12 FILE_FORMAT = "GADGET2" # or "ART" or "ASCII"
13 PARTICLE_MASS = 0        # must specify (in Msun/h) for ART or ASCII
14
15 # You should specify cosmology parameters only for ASCII formats
16 # For GADGET2 and ART, these parameters will be replaced with values from the
17 # particle data file
18 SCALE_NOW = 1
19 h0 = 0.7
20 O1 = 0.73
21 Om = 0.27
22
23 # For GADGET2, you may need to specify conversion parameters.
24 # Rockstar's internal units are Mpc/h (lengths) and Msun/h (masses)
25 GADGET_LENGTH_CONVERSION = 1e-3
26 GADGET_MASS_CONVERSION = 1e+10
27
28 # This specifies the use of multiple processors:
29 PARALLEL_IO = 1
30
31 # Output full particle information as well as halos for N number of procs
32 FULL_PARTICLE_CHUNKS = 0
33
34 # This should be less than 1/5 of BOXSIZE
35 OVERLAP_LENGTH = 1.5
36
37 # This specifies how many CPUs you want to analyze the particles:
38 NUM_WRITERS = 8
39
40 # Calculate radii and other halo properties using unbound (0) or only bound (1) particles (default 1)
41 BOUND_PROPS = 0
42
43 # This sets the virial radius/mass definition ("vir", "XXXc", or "XXXb")
44 MASS_DEFINITION = "vir"
45
46 # This specifies the I/O filenames:
47 OUTBASE = "halos"
48 INBASE = "particles"
49 NUM_SNAPS = 1
50 NUM_BLOCKS = 1
51 #BGC2_SNAPNAMES = "snapnames.lst"
52 #FILENAME = "particles_<snap>.<block>.dat"
```

#### A.2 PBS Submission Script (Bash)

```
1#!/bin/sh
2#PBS -M djsissom@gmail.com
3#PBS -m bae
4#PBS -l nodes=1:ppn=10
5#PBS -l pmem=3000mb
6#PBS -l mem=30000mb
7#PBS -l walltime=0:30:00
8#PBS -o out.log
9#PBS -j oe
10
11# Change to working directory
12echo $PBS_NODEFILE
13cd $PBS_O_WORKDIR
14
15# Start the server
16rockstar -c onenode.cfg &> server.out &
17
18# Wait for auto-rockstar.cfg to be created
19perl -e 'sleep 1 while (!(-e "halos/auto-rockstar.cfg"))'
20mv halos/auto-rockstar.cfg .
21
22# Execute the reader processes
23mpieexec -verbose -n 1 rockstar -c auto-rockstar.cfg >> clients.out 2>&1 &
24sleep 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)

```
1 MIN_SNAPSHOT_NUM 0
2 MAX_SNAPSHOT_NUM 0
3
4 MAX_RANK_LOC 0
5
6 OUTBASE crossmatch_2lpt_first
```

#### B.2 ZA First Configuration File (Text)

```
1 MIN_SNAPSHOT_NUM 0
2 MAX_SNAPSHOT_NUM 0
3
4 MAX_RANK_LOC 0
5
6 OUTBASE crossmatch_za_first
```

## Appendix C

### BGC2 Import Code (Python)

```
1 #!/usr/bin/env python
2
3 import sys
4 import struct
5
6 def read_bgc2(filename):
7     offset = 4
8     groupoffset = 8
9     particleoffset = 8
10
11    headersize = 1024
12    groupsize = 4*8 + 10*4
13    particlesize = 1*8 + 6*4
14
15    headerformat = '=Q>16q>19d'
16    groupformat = '>2q>20>10f'
17    particleformat = '>q>6f'
18
19    print "Reading "+filename+"..."
20    fd = open(filename, 'rb')
21    bin_string = fd.read()
22    fd.close()
23    print "Finished reading file."
24    bin_string = bin_string[offset:]
25
26    # Header stuff
27    header_bin = bin_string[:headersize]
28    header_pad = headersize - 36*8
29    header = list(struct.unpack(headerformat, header_bin[:-header_pad]))
30
31    # Group stuff
32    ngroups = header[8]
33    print 'ngroups=%u', ngroups
34    groupstart = headersize + groupoffset
35    groupend = groupstart + ngroups*groupsize
36    group_bin = bin_string[groupstart:groupend]
37    group = []
38    for i in range(ngroups):
39        group.append(list(struct.unpack(groupformat, group_bin[i*groupsize:(i+1)*groupsize])))
40
41    # Particle stuff
42    particlestart = headersize + groupoffset + ngroups*groupsize + particleoffset
43    particle_bin = bin_string[particlestart:]
44    particle = []
45    p_start = 0
46    for i in range(ngroups):
47        npart = group[i][2]
48        particle.append([])
49        for j in range(npart):
50            particle[i].append(list(struct.unpack(particleformat, particle_bin[p_start:p_start+particlesize])))
51            p_start += particlesize
52        p_start += particleoffset
53
54    print "Finished parsing bgc2 file"
55    return header, group, particle
56
57
58 def main():
59     header, group, particle = read_bgc2(sys.argv[1])
60
61     print 'Header contents:'
62     for value in header:
63         print value
64     print
65
66     print 'Group[0] contents:'
67     for value in group[0]:
68         print value
69     print
70
71     print 'Particles in group[0]:'
72     for part in particle[0]:
73         print part
74     print
75
76     print 'Group[1] contents:'
77     for value in group[1]:
78         print value
79     print
80
81     print 'Particles in group[1]:'
82     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)

```
1 #!/usr/bin/env python
2
3 import sys
4 import bgc2
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from matplotlib.ticker import MultipleLocator
8 from scipy.optimize import curve_fit
9 from scipy.stats import chisquare
10
11 #read_mode = 'ascii2'
12 read_mode = 'bgc2'
13
14 if read_mode == 'bgc2':
15     use_bgc2 = True
16     use_all = False
17     individual_masses = False
18     halo_id = 146289
19     nbins = 50
20     nfit = 500
21     ooms = 3.0
22     mass_scale = 1.0
23     common_mass = 5.33423e5
24     dist_scale = 1.0e3
25     #res_limit = 0.488
26     #res_limit = 4.0
27     res_limit = 0.5
28     #res_limit = 10.0
29     draw_frac = 0.1
30     tick_base_major = 100.0
31     tick_base_minor = 10.0
32     find_com = False
33 elif read_mode == 'ascii':
34     use_bgc2 = False
35     use_all = True
36     individual_masses = True
37     halo_id = 0
38     nbins = 100
39     nfit = 500
40     ooms = 5.0
41     mass_scale = 1.0e12
42     dist_scale = 200.0
43     res_limit = 1.0e-2
44     draw_frac = 2.0e-4
45     tick_base_major = 80.0
46     tick_base_minor = 20.0
47     find_com = True
48 elif read_mode == 'ascii2':
49     use_bgc2 = False
50     use_all = True
51     individual_masses = True
52     halo_id = 0
53     nbins = 100
54     nfit = 500
55     ooms = 3.5
56     mass_scale = 1.0e10
57     dist_scale = 1.0
58     #res_limit = 3.0e-1
59     res_limit = 1.0
60     draw_frac = 1.0e-2
61     tick_base_major = 200.0
62     tick_base_minor = 40.0
63     find_com = True
64 else:
65     sys.exit(98712)
66
67 #outfile = 'asciitest_halo_properties.txt'
68 outfile = 'density_profile_halos.dat'
69 comfile = 'center_of_mass.txt'
70
71 make_plot = False
72 #make_plot = True
73 draw_density = True
74 #plot_base = 'asciitest_density_profile.fig.'
75 plot_base = 'figure_'
76 plot_ext = '.eps'
77 dist_units = 'kpc',
78 xlabel_proj = [r'X\u20d7Position\u20d7(%s\u20d7{-1})$', % (dist_units), r'X\u20d7Position\u20d7(%s\u20d7{-1})$', % (dist_units), r'Y\u20d7Position\u20d7(%s\u20d7{-1})$', % (dist_units)]
79 ylabel_proj = [r'Y\u20d7Position\u20d7(%s\u20d7{-1})$', % (dist_units), r'Z\u20d7Position\u20d7(%s\u20d7{-1})$', % (dist_units), r'Z\u20d7Position\u20d7(%s\u20d7{-1})$', % (dist_units)]
80 xlabel_prof = r'Radius\u20d7(%s\u20d7{-1})$', % (dist_units)
```

```

81 ylabel_prof = r'Density_(M_{\odot}\%{-3}h^2)', % (dist_units)
82 npixels = 50
83
84 #common_mass = 1.0e-7
85 #common_mass = 1.0e5
86 mass_col = 0
87 pos_cols = (1,2,3)
88 vel_cols = (4,5,6)
89 halo_id_col = 0
90
91 grav_const = 4.3e-6 # kpc M_sun^-1 (km/s)^2
92
93
94 def read_files(files):
95     data = 0
96     for file in files:
97         print 'Reading file %s...' % (file)
98         if data == 0:
99             data = np.genfromtxt(file, comments='#')
100        else:
101            data = np.append(data, np.genfromtxt(file, comments='#'), axis=0)
102    print 'Finished reading files.'
103    return data
104
105
106 def my_chisq(ydata, ymod, deg=2, sd=None):
107     """
108     Returns the reduced chi-square error statistic for an arbitrary model,
109     chisq/nu, where nu is the number of degrees of freedom. If individual
110     standard deviations (array_usd) are supplied, then the chi-square error
111     statistic is computed as the sum of squared errors divided by the standard
112     deviations. See http://en.wikipedia.org/wiki/Goodness\_of\_fit for reference.
113
114     ydata, ymod, sd assumed to be Numpy arrays. deg integer.
115
116     Usage:
117     >>> chisq=redchisq(ydata, ymod, n, sd)
118     where
119     ydata: data
120     ymod: model evaluated at the same x points as ydata
121     n: number of free parameters in the model
122     sd: uncertainties in ydata
123
124     Rodrigo Nemmen
125     http://goo.gl/8S10o
126     """
127     # Chi-square statistic
128     if sd==None:
129         chisq=np.sum((ydata-ymod)**2)
130     else:
131         chisq=np.sum( ((ydata-ymod)/sd)**2 )
132
133     # Number of degrees of freedom assuming 2 free parameters
134     nu=ydata.size-1-deg
135     return chisq/nu
136
137
138 def calc_m_enclosed(mass, pos):
139     r = np.sqrt(pos[:,0]**2 + pos[:,1]**2 + pos[:,2]**2)
140     r = np.sort(r)
141     first_good_bin = 0
142     for i in range(len(r)):
143         if r[i] > res_limit:
144             first_good_bin = i
145             break
146     print 'r1=%', r[first_good_bin-1]
147     print 'r2=%', r[first_good_bin]
148     print 'r3=%', r[first_good_bin+1]
149     m_extra = mass[0] * first_good_bin
150     r = r[first_good_bin:]
151     #m_enclosed = np.zeros(len(r))
152     #for i in range(len(r)):
153     #    m_enclosed[i] = mass[0] * (i + 1.0)
154     m_enclosed = (np.arange(len(r)) + 1.0) * mass[0] + m_extra
155     return r, m_enclosed
156
157
158 def calc_density_profile(mass, pos):
159     r = np.sqrt(pos[:,0]**2 + pos[:,1]**2 + pos[:,2]**2)
160     max_r = r.max()
161     #min_r = max_r / 10**oms
162     min_r = res_limit
163     log_range = np.log10(max_r) - np.log10(min_r)
164
165     #global nbins
166     local_nbins = float(nbins + 1)
167     #nbins = len(r) / 1000
168     while True:
169         bins = np.arange(local_nbins)
170         bins = max_r * 10.0**log_range * bins / (local_nbins-1.0) - log_range)
171         bin_mass, r_bins = np.histogram(r, bins, weights=mass)
172         if (bin_mass == 0.0).any():

```

```

173     local_nbins -= 1
174     continue
175   else:
176     break
177
178 #print 'Binning particles using bin edges of \n', r_bins
179
180 rho = bin_mass / (sphere_vol(r_bins[1:]) - sphere_vol(r_bins[:-1]))
181
182 N_bin, blah = np.histogram(r, bins)
183 rho_err = poisson_error(N_bin) * rho
184
185 return r_bins, rho, rho_err
186
187
188 def logbin(pos):
189   r = np.sqrt(pos[:,0]**2 + pos[:,1]**2 + pos[:,2]**2)
190   max_r = r.max()
191   min_r = max_r / 10**ooms
192   log_range = np.log10(max_r) - np.log10(min_r)
193
194   global nbins
195   nbins = float(nbins + 1)
196   bins = np.arange(nbins)
197   bins = max_r * 10.0**(log_range * bins / (nbins-1.0) - log_range)
198
199 hist, bin_edges = np.histogram(r, bins)
200 #print 'Binning particles using bin edges of \n', bin_edges
201 return hist, bin_edges
202
203
204 def poisson_error(N):
205   err = np.sqrt(N) / N
206   return err
207
208
209 def sphere_vol(r):
210   volume = (4.0 / 3.0) * np.pi * r**3
211   return volume
212
213
214 def get_rho_0(R_s, R_vir):
215   H = 70.0e-3 # km s^-1 kpc^-1
216   G = 4.3e-6 # kpc M_sun^-1 (km/s)^2
217   rho_crit = 3.0 * H**2 / (8.0 * np.pi * G)
218
219 v = 178
220 c = R_vir / R_s
221 g = 1.0 / (np.log(1.0+c) - c/(1.0+c))
222 delta_char = v * c**3 * g / 3.0
223
224 return rho_crit * delta_char
225
226
227 def nfw_fit_rho0(r, R_s, rho_0):
228   if R_s >= 1.0:
229     return (R_s - 1.0) * np.exp(r) + rho_0 / ((r / R_s) * (1.0 + r / R_s)**2)
230   return rho_0 / ((r / R_s) * (1.0 + r / R_s)**2)
231
232
233 def nfw_fit_rho0_log(r, R_s, rho_0):
234   r = 10.0**r
235   R_s = 10.0**R_s
236   rho_0 = 10.0**rho_0
237   profile = rho_0 / ((r / R_s) * (1.0 + r / R_s)**2)
238   return np.log10(profile)
239
240
241 def nfw_def_rho0(R_vir):
242   def _nfw_def_rho0(r, R_s):
243     rho_0 = get_rho_0(R_s, R_vir)
244     return rho_0 / ((r / R_s) * (1.0 + r / R_s)**2)
245   return _nfw_def_rho0
246
247
248 def nfw_databin_rho0(rho_0):
249   def _nfw_databin_rho0(r, R_s):
250     return rho_0 / ((r / R_s) * (1.0 + r / R_s)**2)
251   return _nfw_databin_rho0
252
253
254 def dm_profile_fit_rho0_log(r, R_s, rho_0, alpha):
255   r = 10.0**r
256   R_s = 10.0**R_s
257   rho_0 = 10.0**rho_0
258   alpha = 10.0**alpha
259   profile = rho_0 / ((r / R_s) * (1.0 + r / R_s)**alpha)
260   return np.log10(profile)
261
262
263 def dm_profile_fit_rho0(r, R_s, rho_0, alpha):
264   return rho_0 / ((r / R_s) * (1.0 + r / R_s)**alpha)

```

```

265
266 def dm_profile_def_rho0(R_vir):
267     def _dm_profile_def_rho0(r, R_s, alpha):
268         rho_0 = get_rho_0(R_s, R_vir)
269         return rho_0 / ((r / R_s) * (1.0 + r / R_s)**alpha)
270     return _dm_profile_def_rho0
271
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
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_0before', 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_0after', 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[:,0] * vel[:,1] - pos[:,1] * vel[:,0]))
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 pos_x_pos_y_pos_z_pos_nbins_N_part\n')
491 #        fd.write('err_rho_0 R_vir R_s +- err_rho_0 +- err_nbins_N_part\n')
492 #        fd.write('err_alpha chi_square\n')
493
494 #    with open(comfile, 'w') as fd:
495 #        fd.write('#id mass dx dy dz\n')
496
497 #    if use_bgc2 == True:
498 #        header, halos, particles = bgc2.read_bgc2(sys.argv[1])
499 #        for i in range(len(halos)):
500 #            if halos[i][halo_id_col] == halo_id:
501 #                index = i
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        itteration = 0
532        #for index in range(len(halos)):
533        #for index in range(1):
534        #for index in indices[:10]:
535        for index in indices:
536            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] *
549         dist_scale])
550         halo_vel = np.array([halos[index][9], halos[index][10], halos[index][11]])
551     else:
552         r_vir = 241.48
553         halo_id = 0
554         #halo_mass = mass[0] * len(halo_particles)
555         halo_pos = np.array([0.0, 0.0, 0.0])
556         halo_vel = np.array([0.0, 0.0, 0.0])
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\n" % (halo_id, halo_mass, halo_pos[0] - com[0], halo_pos[1] - com[1],
581     halo_pos[2] - com[2]))
582
583     # Bin halo particles into logarithmic shells and compute density
584     r_bins, rho, rho_err = calc_density_profile(mass, pos)
585
586     if len(r_bins) < 5:
587         print 'Too few bins. Skipping this halo.'
588         with open(outfile, 'a') as fd:
589             fd.write("%8d%8d\n" % (halo_id, halo_mass, halo_pos[0], halo_pos[1], halo_pos[2], -9999, -9999, -9999,
590             -9999, -9999, -9999, len(halo_particles)))
591         continue
592
593     # hist, r_bins = logbin(pos)
594     # err = poisson_error(hist)
595     # rho = mass * hist / (sphere_vol(r_bins[1:]) - sphere_vol(r_bins[:-1]))
596     # rho_err = err * rho
597     mid_bins = 10.0**((0.5 * (np.log10(r_bins[1:]) + np.log10(r_bins[:-1]))))
598     print 'nbins =', len(mid_bins)
599
600     # Don't pass NaN's to fitting routine
601     rho_err_nonan = np.copy(rho_err)
602     nan_check = np.isnan(rho_err_nonan)
603     for i in range(len(rho_err_nonan)):
604         if (nan_check[i] == True):
605             # rho[i] = 1.0e-10
606             if (mid_bins[i] < res_limit) or (nan_check[i] == True):
607                 rho_err_nonan[i] = 1.0e10
608
609     # m_enclosed = calc_m_enclosed(mass, pos)
610
611     # Fit an NFW profile to the data
612     try:
613         nfw_r, nfw_fit, popt, pcov, chisq = fit_profile(mid_bins / r_vir, rho / rho.max(), err = rho_err_nonan /
614         rho.max(), R_vir = 1.0)
615         #nfw_r, nfw_fit, popt, pcov, chisq = fit_mass_profile(r / r_vir, m_enclosed / halo_mass)
616         nfw_r = nfw_r * r_vir
617         nfw_fit = nfw_fit * rho.max()
618         scale_radius = popt[0] * r_vir
619         scale_radius_err = pcov[0,0] * r_vir
620         rho_0 = popt[1] * rho.max()
621         rho_0_err = pcov[1,1] * rho.max()
622         concentration = r_vir / scale_radius
623         concentration_err = concentration * scale_radius_err / scale_radius
624
625     # Print parameters

```

```

625     print 'r_vir=%', r_vir
626     print "rho_0=%g+/-%g" % (rho_0, rho_0_err)
627     print "scale_radius=%g+/-%g" % (scale_radius, scale_radius_err)
628     print "concentration=%g+/-%g" % (concentration, concentration_err)
629
630 #put these back sometime#####
631 #     kin_energy = calc_kinetic_energy(mass, vel)
632 #     pot_energy = calc_potential_energy(mass, pos)
633 #     ang_mom = calc_angular_momentum(mass, pos, vel)
634 #
635 #     ttow = 2.0 * abs(kin_energy / pot_energy)
636 #     lambda_spin = ang_mom * np.sqrt(abs(kin_energy + pot_energy)) / (grav_const * (np.sum(mass))**2.5)
637     kin_energy = 0.0
638     pot_energy = 0.0
639     ang_mom = 0.0
640
641     ttow = 0.0
642     lambda_spin = 0.0
643 ##########
644
645     if isinstance(pcov, float):
646         print "info:covariance returned, skipping this halo..."
647         with open(outfile, 'a') as fd:
648             fd.write("%8d%16.12g%14.10g%14.10g%14d+-%14d%14d%14d+-%14d%14d%14d\n" % (halo_id, halo_mass, halo_pos[0], halo_pos[1], halo_pos[2], -9999, -9999, -9999, -9999,
649             -9999, -9999, -9999, len(halo_particles)))
650         continue
651
652     #Write parameters to file
653     with open(outfile, 'a') as fd:
654         #fd.write("%g %g %g %g +- %g %g +- %g %g\n" % (halo_mass, concentration, r_vir,
655         scale_radius, pcov[0,0], rho_0, pcov[1,1], alpha, pcov[2,2], chisq))
656         fd.write("%8d%16.12g%14.10g%14.10g%14.10g+-%14.6g%14.10g%14.10g%14.6g%14.10
657         g+-%14.6g%14.10g%8d%8d\n" % (halo_id, halo_mass, halo_pos[0], halo_pos[1], halo_pos[2],
658         concentration, concentration_err, r_vir, scale_radius, scale_radius_err, rho_0, rho_0_err, chisq, len(r_bins),
659         ), len(halo_particles)))
660
661
662     ##########
663     #debug
664     #blah_fit = nfw_fit_rho0(nfw_r, 20.0, 9.0e5)
665
666     # Plot density profile histogram
667     if (make_plot == True) and (iteration < 10):
668         # Find the maximum of x, y, or z to be limit of projection plots
669         plot_lim = pos.max()
670         # Pick only a certain percentage of particles for projection plots
671         if (draw_frac < 1.0):
672             np.random.shuffle(pos)
673             pos = pos[:int(draw_frac*pos.shape[0])]

674         fig = plt.figure()
675         if draw_density == True:
676             fig = draw_density_projection(fig, 0, plot_lim, pos[:,0], pos[:,1])
677             fig = draw_density_projection(fig, 1, plot_lim, pos[:,0], pos[:,2])
678             fig = draw_density_projection(fig, 2, plot_lim, pos[:,1], pos[:,2])
679         else:
680             fig = draw_projection(fig, 0, plot_lim, pos[:,0], pos[:,1])
681             fig = draw_projection(fig, 1, plot_lim, pos[:,0], pos[:,2])
682             fig = draw_projection(fig, 2, plot_lim, pos[:,1], pos[:,2])
683             fig, ax = draw_density_profile(fig, mid_bins, rho, err=rho_err) #put this back for binning
684             #fig, ax = draw_density_profile(fig, r, m_enclosed) #take this out for binning
685             fig = draw_nfw_profile(fig, ax, nfw_r, nfw_fit, R_s=scale_radius)
686             #fig = draw_nfw_profile(fig, ax, nfw_r, blah_fit, R_s=20.0)
687             fig.tight_layout()
688             plt.savefig(plot_base+str(iteration)+plot_ext)
689
690
691 if __name__ == '__main__':
692     main()

```

## Appendix E

### CROSSMATCH Best Match Code

#### E.1 Best Match (Python)

```
1 #!/usr/bin/env python
2
3 import sys
4 import getopt
5 import numpy as np
6
7
8 def main():
9     # read in files
10    print 'reading files...'
11    with open(sys.argv[1]) as f:
12        matches1 = f.readlines()
13    with open(sys.argv[2]) as f:
14        matches2 = f.readlines()
15    print 'done reading files'
16
17    header = matches1[2:6]
18    header.insert(0, '#Best matches for bi-directional crossmatch\n')
19    header.insert(1, '\n')
20
21    matches1 = matches1[7:]
22    matches2 = matches2[7:]
23
24    # convert to numpy arrays
25    print 'converting to numpy arrays...'
26    match_array1 = np.asarray([line.split() for line in matches1], dtype=int)
27    match_array2 = np.asarray([line.split() for line in matches2], dtype=int)
28    print 'done converting'
29
30    # find matches that exist in both lists
31    print 'finding matches...'
32    mask = np.zeros(len(match_array1), dtype=bool)
33    for i, line in enumerate(match_array1):
34        id1 = line[id1_col]
35        id2 = line[id2_col]
36        tmp = (match_array2[:,id1_col] == id2)
37        tmp = (match_array2[tmp,id2_col] == id1)
38        mask[i] = tmp.any()
39        if i % 1000 == 0:
40            print "Finished line", i
41
42    print 'done matching'
43
44    out_array = match_array1[mask]
45
46    # write results
47    print 'writing results...'
48    with open(sys.argv[3], 'w') as f:
49        f.writelines((("%s" % line for line in header)))
50        np.savetxt(f, out_array, fmt='%10d')
51
52    print 'Finished.'
53
54
55 id1_col      = 4
56 npart1_col   = 5
57 id2_col      = 1
58 npart2_col   = 2
59 ncommon_col  = 6
60 hnum1_col    = 3
61 hnum2_col    = 0
62
63
64 if __name__ == '__main__':
65     main()
```

#### E.2 PBS Submission Script (Bash)

```
1 #!/usr/bin/env bash
2 #PBS -M djsissom@gmail.com
3 #PBS -m bae
4 #PBS -l nodes=27:ppn=1
5 #PBS -l pmem=20000mb
6 #PBS -l mem=54000mb
7 #PBS -l walltime=0:30:00
8 #PBS -o out.log
9 #PBS -j oe
10
11 #nodes=186:ppn=1
```

```

12 #pmem=20000mb
13 #mem=372000mb
14
15 minsnap=53
16 maxsnap=61
17
18 minbox=1
19 maxbox=3
20
21 # Change to working directory
22 echo $PBS_NODEFILE
23 cd $PBS_O_WORKDIR
24
25 for ((i=$minbox; i<=$maxbox; i++)); do
26
27   for ((snap=$minsnap; snap<=$maxsnap; snap++)); do
28
29     if [ $snap -lt 10 ]; then
30       j=0$snap
31     elif [ $snap -lt 100 ]; then
32       j=0$snap
33     fi
34
35     base_dir=~/projects/simulations/rockstar/box${i}
36     crossmatch_dir=${base_dir}/crossmatch/snap${j}
37     first_file=${crossmatch_dir}/crossmatch_2lpt_first_000.txt
38     second_file=${crossmatch_dir}/crossmatch_za_first_000.txt
39     outfile=${crossmatch_dir}/crossmatch_000.txt
40     logfile=${crossmatch_dir}/best_crossmatch.log
41
42     echo "Starting box${i} snap${j}..."
43
44     {
45       mpiexec -verbose -n 1 ./best_crossmatch.py ${first_file} ${second_file} ${outfile} > ${logfile} 2>&1
46       echo "Finished box${i} snap${j}"
47     } &
48
49   done
50
51 done
52
53 wait
54
55 # - end of script

```

## Appendix F

### Database Generation Code

#### F.1 Halo Match (Python)

```
1 #!/usr/bin/env python
2
3 import sys
4 import getopt
5 import numpy as np
6
7
8 def main():
9     # read and parse command line arguments
10    opts, args = getopt.getopt(sys.argv[1:])
11    output_file, match_file, densprof_files, parents_files, ascii_files = parse_args(opts, args)
12
13    # read in headers as lists and data as numpy arrays
14    match_header, match_data = read_files(match_file, header_line = 3)
15    densprof_header1, densprof_data1 = read_files(densprof_files[0], header_line = 0)
16    densprof_header2, densprof_data2 = read_files(densprof_files[1], header_line = 0)
17    parents_header1, parents_data1 = read_files(parents_files[0], header_line = 0)
18    parents_header2, parents_data2 = read_files(parents_files[1], header_line = 0)
19    ascii_header1, ascii_data1 = read_files(ascii_files[:len(ascii_files)/2]), header_line = 0)
20    ascii_header2, ascii_data2 = read_files(ascii_files[len(ascii_files)/2:]), header_line = 0)
21    print 'Finished reading files.'
22
23    # filter matches, remove duplicate halo matches, and reorder match columns
24    print 'Filtering match data...'
25    match_data = filter_matches(match_data)
26    if filter_duplicate_matches:
27        match_data = filter_dups(match_data, unique_col = match_id1_col)
28        match_data = filter_dups(match_data, unique_col = match_id2_col)
29    if reorder_match_columns:
30        match_header, match_data = reorder_match_cols(match_header, match_data)
31
32    # calculate number of subhalos and add column to parents data and headers
33    print 'Finding number of subhalos...'
34    parents_header1.append('N_sub')
35    parents_header2.append('N_sub')
36    parents_data1 = count_subs(parents_data1)
37    parents_data2 = count_subs(parents_data2)
38
39    # create header
40    print 'Making header...'
41    header = make_header(match_header, densprof_header1, densprof_header2, \
42                          parents_header1, parents_header2, ascii_header1, ascii_header2)
43
44    # match halos
45    print 'Matching halos...'
46    halos = match_halos(match_data, [densprof_data1, densprof_data2, \
47                                      parents_data1, parents_data2, ascii_data1, ascii_data2])
48
49    # filter based on given criteria and sort
50    print 'Filtering halo data...'
51    if filter_halo_properties:
52        halos = filter_halos(halos)
53    if sort_col != None:
54        sort_mask = halos[:,sort_col].argsort()
55        sort_mask = sort_mask[::-1]
56        halos = halos[sort_mask]
57
58    # output matched table
59    print 'Writing results...'
60    write_results(output_file, header, halos)
61
62    print 'Finished.'
63
64
65 def get_args(arglist):
66     try:
67         opts, args = getopt.gnu_getopt(arglist, shortopts, longopts)
68     except getopt.GetoptError:
69         print "Invalid option(s)."
70         print help_string
71         sys.exit(2)
72     if opts == []:
73         print 'No options given.'
74         print help_string
75         sys.exit(2)
76     return opts, args
77
78
79 def parse_args(opts, args):
80     densprof_files = None
```

```

81 parentsfiles = None
82 asciiifiles = None
83 use_ascii = False
84 for opt in opts:
85     if (opt[0] == '-h') or (opt[0] == '--help') or (opts == None):
86         print help_string
87         sys.exit(0)
88     if (opt[0] == '-o') or (opt[0] == '--outfile'):
89         outfile = opt[1]
90     if (opt[0] == '-m') or (opt[0] == '--match'):
91         matchfile = opt[1]
92     if (opt[0] == '-d') or (opt[0] == '--density'):
93         densproffiles = create_append(densproffiles, opt[1])
94     if (opt[0] == '-p') or (opt[0] == '--parents'):
95         parentsfiles = create_append(parentsfiles, opt[1])
96     if (opt[0] == '-a'):
97         use_ascii = True
98 if use_ascii:
99     if len(args) % 2 != 0:
100        print 'Must have an even number of ascii files!'
101        sys.exit(3)
102    for arg in args:
103        asciiifiles = create_append(asciiifiles, arg)
104 return outfile, matchfile, densproffiles, parentsfiles, asciiifiles
105
106
107 def create_append(lst, value):
108     if lst == None:
109         lst = [value]
110     else:
111         lst.append(value)
112     return lst
113
114
115 def read_files(files, header_line = None, comment_char = '#'):
116     header = None
117     data = None
118     if type(files) == str:
119         files = [files]
120
121     if header_line != None:
122         with open(files[0], 'r') as fd:
123             for line in range(header_line):
124                 fd.readline()
125             header = fd.readline()
126             if header[0] != comment_char:
127                 print "Header must start with a %s" % comment_char
128                 sys.exit(4)
129             header = header[1:]
130             header = header.split()
131
132     for file in files:
133         print 'Reading file %s...' % (file)
134         if data == None:
135             data = np.genfromtxt(file, comments='#')
136         else:
137             data = np.append(data, np.genfromtxt(file, comments='#'), axis=0)
138
139     if header_line == None:
140         return data
141     else:
142         return header, data
143
144
145 def filter_matches(halos):
146     if filter_bad_matches:
147         halos = halos[:,match_id1_col] != -1]
148         halos = halos[:,match_id2_col] != -1]
149     if (min_npart != 0) and (min_npart != None):
150         halos = halos[:, match_npart1_col] >= min_npart]
151         halos = halos[:, match_npart2_col] >= min_npart]
152     if (minperc_ncommon != 0) and (minperc_ncommon != None):
153         halos = halos[:, match_ncommon_col] / halos[:, match_npart1_col] >= minperc_ncommon]
154         halos = halos[:, match_ncommon_col] / halos[:, match_npart2_col] >= minperc_ncommon]
155     return halos
156
157
158 def filter_dups(halos, unique_col = 0):
159     ncommon = halos[:, match_ncommon_col]
160     n1 = halos[:, match_npart1_col]
161     n2 = halos[:, match_npart2_col]
162     rank = ncommon**2 / (n1 * n2) - np.abs(n1 - n2) / (n1 + n2)
163
164     sort_mask = np.argsort(rank)
165     halos = halos[sort_mask]
166
167     unique, mask = np.unique(halos[:, unique_col], return_index=True)
168     halos = halos[mask]
169     return halos
170
171
172 def reorder_match_cols(match_header, match_data):

```

```

173     global match_id1_col
174     global match_id2_col
175     global match_hnum1_col
176     global match_hnum2_col
177     global match_npart1_col
178     global match_npart2_col
179     global match_ncommon_col
180
181     order = [match_id1_col, match_id2_col, \
182             match_hnum1_col, match_hnum2_col, \
183             match_npart1_col, match_npart2_col, \
184             match_ncommon_col]
185     match_header = [match_header[index] for index in order]
186     match_data = match_data[:, order]
187
188     match_id1_col = 0
189     match_id2_col = 1
190     match_hnum1_col = 2
191     match_hnum2_col = 3
192     match_npart1_col = 4
193     match_npart2_col = 5
194     match_ncommon_col = 6
195
196     return match_header, match_data
197
198
199 def count_subs(halos):
200     id = halos[:, id_col]
201     parents = halos[:, parents_col]
202     parents = parents[parents != -1]
203     nsubs = (id[:, np.newaxis] == parents).sum(axis = 1)
204     halos = np.column_stack((halos, nsubs))
205     return halos
206
207
208 def make_header(match, densprof1, densprof2, parents1, parents2, ascii1, ascii2):
209     # zeroeth line just lists column number
210     total_len = len(match + densprof1 + densprof2 + parents1 + parents2 + ascii1 + ascii2)
211     header_line0 = [str(i) for i in range(total_len)]
212     header_line0 = 'uu'.join(header_line0)
213     header_line0 = '#' + header_line0
214
215     # first line denotes which file columns are from
216     match_repeat = len(match) - 4
217     densprof_repeat = len(densprof1 + densprof2) - 4
218     parents_repeat = len(parents1 + parents2) - 4
219     ascii_repeat = len(ascii1 + ascii2) - 4
220
221     match_part = 'uu'.join(['|---', 'cross', 'match'] + ['|---'] * match_repeat + ['|---'])
222     densprof_part = 'uu'.join(['|---', 'density', 'profile'] + ['|---'] * densprof_repeat + ['|---'])
223     parents_part = 'uu'.join(['|---', 'rockstar', 'parents'] + ['|---'] * parents_repeat + ['|---'])
224     ascii_part = 'uu'.join(['|---', 'rockstar', 'ascii'] + ['|---'] * ascii_repeat + ['|---'])
225
226     header_line1 = 'uu'.join([match_part, densprof_part, parents_part, ascii_part])
227     header_line1 = '#' + header_line1
228
229     # second line labels 2lpt and za columns
230     tot_len = len(match + densprof1 + densprof2 + parents1 + parents2 + ascii1 + ascii2)
231     header_line2 = ['2lpt' if i % 2 == 0 else 'za' if i % 2 == 1 else 'blah' for i in range(tot_len - 1)]
232     header_line2.insert(len(match) - 1, 'matched')
233     header_line2 = 'uu'.join(header_line2)
234     header_line2 = '#' + header_line2
235
236     # third line pulls labels from original file headers
237     match_part = match
238     densprof_part = interweave(densprof1, densprof2)
239     parents_part = interweave(parents1, parents2)
240     ascii_part = interweave(ascii1, ascii2)
241
242     header_line3 = match_part + densprof_part + parents_part + ascii_part
243     header_line3 = 'uu'.join(header_line3)
244     header_line3 = '#' + header_line3
245
246     header = [header_line0, header_line1, header_line2, header_line3]
247     return header
248
249
250 def interweave(list1, list2):
251     newlist = list1 + list2
252     newlist[::2] = list1
253     newlist[1::2] = list2
254     return newlist
255
256
257 def interweave_np_2d(array1, array2):
258     newarray = np.empty((len(array1), len(array1[0]) + len(array2[0])))
259     newarray[:, ::2] = array1
260     newarray[:, 1::2] = array2
261     return newarray
262
263
264 def match_halos(matches, arrays):

```

```

265     halos = matches.copy()
266     for i, array in enumerate(arrays):
267         if array != None:
268             match_id_col = i % 2
269             halos = sort_stack(halos, array, match_id_col)
270
271     # interweave columns so that matching 2lpt/za columns are adjacent
272     tmp_halos = halos
273     halos = np.empty((len(tmp_halos), len(tmp_halos[0])))
274     halos[:,len(matches[0])] = matches
275     startcol = len(matches[0])
276     for i in range(0, len(arrays), 2):
277         colrange1 = len(arrays[i][0])
278         colrange2 = len(arrays[i+1][0])
279         endcol = startcol + colrange1 + colrange2
280
281         cols1 = tmp_halos[:,startcol:startcol+colrange1]
282         cols2 = tmp_halos[:,startcol+colrange1:startcol+colrange1+colrange2]
283
284         halos[:,startcol:endcol] = interweave_np_2d(cols1, cols2)
285         startcol = endcol
286
287     return halos
288
289 def sort_stack(halos, array, match_id_col):
290     # add empty columns to halos to later fill with halo data
291     rows = len(halos)
292     origcols = len(halos[0])
293     newcols = len(array[0])
294     empty = np.empty((rows, newcols))
295     empty[:] = np.nan
296     halos = np.column_stack((halos, empty))
297
298     # remove halos from array with no matches
299     match_id = halos[:, match_id_col]
300     array_id = array[:, id_col]
301     array_mask = np.in1d(array_id, match_id)
302     array = array[array_mask]
303
304     # create mask so we only add lines for halos in array
305     array_id = array[:, id_col]
306     halo_mask = np.in1d(match_id, array_id)
307     masked_halos = halos[halo_mask]
308
309     # create masks to sort by halo id
310     match_id_sort_mask = np.argsort(masked_halos[:, match_id_col])
311     sorted_masked_halos = masked_halos[match_id_sort_mask]
312
313     # sort array by halo id and copy to empty columns of view of halos
314     array_id_sort_mask = np.argsort(array[:,id_col])
315     sorted_masked_halos[:, origcols:] = array[array_id_sort_mask]
316
317     # 'unmask' - put data back in original halos
318     masked_halos[match_id_sort_mask] = sorted_masked_halos
319     halos[halo_mask] = masked_halos
320
321     return halos
322
323
324 def filter_halos(halos):
325     #todo
326     return halos
327
328
329 def write_results(output_file, header, halos):
330     format = get_format(halos[0])
331     with open(output_file, 'w') as fd:
332         for line in header:
333             fd.write(line + '\n')
334     np.savetxt(fd, halos, fmt=format)
335
336
337 def get_format(line):
338     format = ['%d' if col in int_cols else '%1.14g' for col in range(len(line))]
339     format = ''.join(format)
340     return format
341
342
343 help_string = '''
344 Available options are:
345     -h, --help
346     -v, --verbose
347     -o <outfile>, --outfile <outfile>
348     -m <matchlist>, --match <matchlist>
349     -d <densityprofile_file>, --density <densityprofile_file>
350     -p <parents_file>, --parents <parents_file>
351     -a <ascii_files>, --ascii <ascii_files> - must be last
352 '''
353 shortopts = "hvo:m:d:p:a"
354 longopts = ["help", "verbose", "outfile=", "matchfile=", "density=", "parents=", "ascii"]
355
356 int_cols = []

```

```

357 lt_vals = []
358
359 gt_cols = []
360 gt_vals = []
361
362 eq_cols = []
363 eq_vals = []
364
365 ne_cols = []
366 ne_vals = []
367
368 #int_cols = [0, 1, 2, 3, 4, 5, 6, 7, 8]
369 int_cols = []
370
371 match_id2_col      = 1
372 match_npart2_col   = 2
373 match_id1_col      = 4
374 match_npart1_col   = 5
375 match_ncommon_col  = 6
376 match_hnum1_col    = 3
377 match_hnum2_col    = 0
378
379 id_col             = 0      # col of each input file
380 sort_col            = 47     # col of final table - use None to turn off sorting
381 parents_col         = -1
382
383 filter_bad_matches = True
384 filter_duplicate_matches = False
385 reorder_match_columns = True
386 filter_halo_properties = False
387 min_npart          = 20     # use 0 or None to use all size halos
388 minperc_ncommon    = 0.05   # a fraction, use 0 or None to use any match percent
389
390
391 if __name__ == '__main__':
392     main()

```

## F.2 PBS Submission Script (Bash)

```

1#!/usr/bin/env bash
2#PBS -M djsissom@gmail.com
3#PBS -m bae
4#PBS -l nodes=1:ppn=1
5#PBS -l pmem=40000mb
6#PBS -l mem=4000mb
7#PBS -l walltime=1:00:00
8#PBS -o out.log
9#PBS -j oe
10
11 minsnap=0
12 maxsnap=61
13
14 minbox=1
15 maxbox=3
16
17 # Change to working directory
18 echo $PBS_NODEFILE
19 cd ${PBS_O_WORKDIR}
20
21 for ((i=$minbox; i<=$maxbox; i++)); do
22
23     for ((snap=$minsnap; snap<=$maxsnap; snap++)); do
24
25         if [ $snap -lt 10 ]; then
26             j=0$snap
27         elif [ $snap -lt 100 ]; then
28             j=0$snap
29         fi
30
31         base_dir=~/projects/simulations/rockstar/box${i}
32         crossmatch_dir=${base_dir}/crossmatch/snap${j}
33         snap_dir_2lpt=${base_dir}/2lpt/snap${j}
34         snap_dir_zap=${base_dir}/za/snap${j}
35         logfile=${crossmatch_dir}/match_halos.log
36
37         echo "Starting box${i} snap${j}..."
38
39         {
40             #mpixexec -verbose -n 1 \
41             ./match.py -o ${crossmatch_dir}/halos.dat \
42             -m ${crossmatch_dir}/crossmatch_000.txt \
43             -d ${snap_dir_2lpt}/halos/density_profile_halos.dat \
44             -d ${snap_dir_zap}/halos/density_profile_halos.dat \
45             -p ${snap_dir_2lpt}/halos/out_0.list.parents \
46             -p ${snap_dir_zap}/halos/out_0.list.parents \
47             -a \
48             ${snap_dir_2lpt}/halos/halos_0.*.ascii \
49             ${snap_dir_zap}/halos/halos_0.*.ascii \
50             > ${logfile} 2>&1
51
52         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)

```
1 #!/usr/bin/env python
2
3 import sys
4 import bgc2
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from matplotlib.patches import Circle
8 from matplotlib.ticker import MultipleLocator
9 from scipy.optimize import curve_fit
10 from scipy.stats import chisquare
11
12 #id1, id2 = 727, 4420 # 2lpt first
13 #id1, id2 = 4416, 727 # za first
14
15 #id1, id2 = 4416, 4420 # both za
16 #id1, id2 = 4416, 4416 # both za
17
18 #id1, id2 = 653, 4355
19 #id1, id2 = 38, 3803
20 #id1, id2 = 155099, 80362
21 #id1, id2 = 98722, 14357
22 id1, id2 = 84289, 143514
23
24
25 #read_mode = 'ascii'
26 read_mode = 'bgc2'
27
28 if read_mode == 'bgc2':
29     use_bgc2 = True
30     use_all = False
31     multiple_halos = True
32     individual_masses = False
33     halo_id = 146289
34     nbins = 50
35     nfit = 500
36     ooms = 3.0
37     mass_scale = 1.0
38     common_mass = 5.33423e5
39     dist_scale = 1.0e3
40     #res_limit = 0.488
41     res_limit = 4.0
42     #res_limit = 10.0
43     #draw_frac = 1.0e-2
44     draw_frac = 1.0
45     tick_base_major = 10.0
46     tick_base_minor = 1.0
47 elif read_mode == 'ascii':
48     use_bgc2 = False
49     use_all = True
50     individual_masses = True
51     halo_id = 0
52     nbins = 100
53     nfit = 500
54     ooms = 5.0
55     mass_scale = 1.0e12
56     dist_scale = 200.0
57     res_limit = 1.0e-2
58     draw_frac = 2.0e-4
59     tick_base_major = 80.0
60     tick_base_minor = 20.0
61 elif read_mode == 'ascii2':
62     use_bgc2 = False
63     use_all = True
64     individual_masses = True
65     halo_id = 0
66     nbins = 100
67     nfit = 500
68     ooms = 3.5
69     mass_scale = 1.0e10
70     dist_scale = 1.0
71     res_limit = 3.0e-1
72     draw_frac = 1.0e-2
73     tick_base_major = 200.0
74     tick_base_minor = 40.0
75 else:
76     sys.exit(98712)
77
78 outfile = 'halo_properties.txt'
79 comfile = 'center_of_mass.txt'
```

```

81 make_plot = True
82 plot_base = 'density_profile.fig.'
83 plot_ext = '.eps'
84 dist_units = 'kpc'
85 xlabel_proj = [r'X\u20d7Position\u20d7(%s\u20d7h\u20d7^{\{-1\}})', % (dist_units), r'X\u20d7Position\u20d7(%s\u20d7h\u20d7^{\{-1\}})', % (dist_units), r'Y\u20d7
Position\u20d7(%s\u20d7h\u20d7^{\{-1\}})', % (dist_units)]
86 ylabel_proj = [r'Y\u20d7Position\u20d7(%s\u20d7h\u20d7^{\{-1\}})', % (dist_units), r'Z\u20d7Position\u20d7(%s\u20d7h\u20d7^{\{-1\}})', % (dist_units), r'Z\u20d7
Position\u20d7(%s\u20d7h\u20d7^{\{-1\}})', % (dist_units)]
87 xlabel_prof = r'Radius\u20d7(%s\u20d7h\u20d7^{\{-1\}})', % (dist_units)
88 ylabel_prof = r'Density\u20d7(M_{\u20d7\odot}\u20d7%s\u20d7^{\{-3\}}\u20d7h\u20d7^{\{2\}})', % (dist_units)
89
90 #common_mass = 1.0e-7
91 #common_mass = 1.0e5
92 mass_col = 0
93 pos_cols = (1,2,3)
94 vel_cols = (4,5,6)
95 halo_id_col = 0
96
97 grav_const = 4.3e-6 # kpc M_sol^-1 (km/s)^2
98
99 profile_type = 0 # 0 -> nfw, fit rho_0
100           # 1 -> nfw, calculate rho_0
101           # 2 -> nfw, rho_0 middle of leftmost bin above resolution
102           # 3 -> fit outer slope, fit rho_0
103           # 4 -> fit outer slope, calculate rho_0
104           # 5 -> fit outer slope, rho_0 middle of leftmost bin above resolution
105
106 def read_files(files):
107     data = 0
108     for file in files:
109         print 'Reading\u20d7file\u20d7%s...', % (file)
110         if data == 0:
111             data = np.genfromtxt(file, comments='#')
112         else:
113             data = np.append(data, np.genfromtxt(file, comments='#'), axis=0)
114     print 'Finished\u20d7reading\u20d7files.'
115     return data
116
117
118 def calc_density_profile(mass, pos):
119     r = np.sqrt(pos[:,0]**2 + pos[:,1]**2 + pos[:,2]**2)
120     max_r = r.max()
121     #min_r = max_r / 10**ooms
122     min_r = res_limit
123     log_range = np.log10(max_r) - np.log10(min_r)
124
125     #global nbins
126     local_nbins = float(nbins + 1)
127     #nbins = len(r) / 1000
128     while True:
129         bins = np.arange(local_nbins)
130         bins = max_r * 10.0**(log_range * bins / (local_nbins-1.0) - log_range)
131         bin_mass, r_bins = np.histogram(r, bins, weights=mass)
132         if (bin_mass == 0.0).any():
133             local_nbins -= 1
134             continue
135         else:
136             break
137
138     #print 'Binning particles using bin edges of \n', r_bins
139
140     rho = bin_mass / (sphere_vol(r_bins[1:]) - sphere_vol(r_bins[:-1]))
141
142     N_bin, blah = np.histogram(r, bins)
143     rho_err = poisson_error(N_bin) * rho
144
145     return r_bins, rho, rho_err
146
147
148 def logbin(pos):
149     r = np.sqrt(pos[:,0]**2 + pos[:,1]**2 + pos[:,2]**2)
150     max_r = r.max()
151     min_r = max_r / 10**ooms
152     log_range = np.log10(max_r) - np.log10(min_r)
153
154     global nbins
155     nbins = float(nbins + 1)
156     bins = np.arange(nbins)
157     bins = max_r * 10.0**(log_range * bins / (nbins-1.0) - log_range)
158
159     hist, bin_edges = np.histogram(r, bins)
160     #print 'Binning particles using bin edges of \n', bin_edges
161     return hist, bin_edges
162
163
164 def poisson_error(N):
165     err = np.sqrt(N) / N
166     return err
167
168
169 def sphere_vol(r):
170     volume = (4.0 / 3.0) * np.pi * r**3

```

```

171     return volume
172
173
174 def get_rho_0(R_s, R_vir):
175     H = 70.0e-3 # km s^-1 kpc^-1
176     G = 4.3e-6 # kpc M_sun^-1 (km/s)^2
177     rho_crit = 3.0 * H**2 / (8.0 * np.pi * G)
178
179     v = 178
180     c = R_vir / R_s
181     g = 1.0 / (np.log(1.0+c) - c/(1.0+c))
182     delta_char = v * c**3 * g / 3.0
183
184     return rho_crit * delta_char
185
186
187 def nfw_fit_rho0(r, R_s, rho_0):
188     return rho_0 / ((r / R_s) * (1.0 + r / R_s)**2)
189
190
191 def nfw_fit_rho0_log(r, R_s, rho_0):
192     r = 10.0**r
193     R_s = 10.0**R_s
194     rho_0 = 10.0**rho_0
195     profile = rho_0 / ((r / R_s) * (1.0 + r / R_s)**2)
196     return np.log10(profile)
197
198
199 def nfw_def_rho0(R_vir):
200     def _nfw_def_rho0(r, R_s):
201         rho_0 = get_rho_0(R_s, R_vir)
202         return rho_0 / ((r / R_s) * (1.0 + r / R_s)**2)
203     return _nfw_def_rho0
204
205
206 def nfw_databin_rho0(rho_0):
207     def _nfw_databin_rho0(r, R_s):
208         return rho_0 / ((r / R_s) * (1.0 + r / R_s)**2)
209     return _nfw_databin_rho0
210
211
212 def dm_profile_fit_rho0_log(r, R_s, rho_0, alpha):
213     r = 10.0**r
214     R_s = 10.0**R_s
215     rho_0 = 10.0**rho_0
216     alpha = 10.0**alpha
217     profile = rho_0 / ((r / R_s) * (1.0 + r / R_s)**alpha)
218     return np.log10(profile)
219
220
221 def dm_profile_fit_rho0(r, R_s, rho_0, alpha):
222     return rho_0 / ((r / R_s) * (1.0 + r / R_s)**alpha)
223
224
225 def dm_profile_def_rho0(R_vir):
226     def _dm_profile_def_rho0(r, R_s, alpha):
227         rho_0 = get_rho_0(R_s, R_vir)
228         return rho_0 / ((r / R_s) * (1.0 + r / R_s)**alpha)
229     return _dm_profile_def_rho0
230
231
232 def dm_profile_databin_rho0(rho_0):
233     def _dm_profile_databin_rho0(r, R_s, alpha):
234         return rho_0 / ((r / R_s) * (1.0 + r / R_s)**alpha)
235     return _dm_profile_databin_rho0
236
237
238 def fit_profile(r, rho, err=None, R_vir=None):
239     for i in range(len(r)):
240         if r[i] > res.limit:
241             rho_0_databin = rho[i]
242             first_good_bin = i
243             break
244     #----- choose one fitting type -----
245     #popt, pcov = curve_fit(nfw_fit_rho0, r, rho, sigma=err)
246     #popt, pcov = curve_fit(nfw_def_rho0(R_vir), r, rho, p0=[10.0], sigma=err)
247     #popt, pcov = curve_fit(nfw_databin_rho0(rho_0_databin), r, rho, sigma=err)
248     blah = 2
249     if blah == 0:
250         for i in range(100):
251             a = 2.0 * np.random.random() * 0.1 * r.max()
252             b = 2.0 * np.random.random() * 10.0
253             c = 2.0 * np.random.random() * 2.0
254             try:
255                 popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, p0=[a,b,c], sigma=err)
256             except RuntimeError:
257                 continue
258             if (popt[0] < r.max()) and (popt[2] >= 0.0):
259                 break
260             elif i >= 99:
261                 print 'no good fit found for this halo...'
262     #     return None, None, None, None

```

```

263 elif blah == 1:
264     #a = r.max() / 100.0
265     a = 0.001
266     b = rho[first_good_bin]
267     c = 0.001
268     #popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, sigma=err)
269     print '-----',
270     print 'rho_0before=', b
271     #try:
272     #    popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, p0=[a,b,c], sigma=err, maxfev=1, xtol=100.0)
273     popt, pcov = curve_fit(dm_profile_fit_rho0, r, rho, p0=[a,b,c], sigma=err, xtol=1.0e-1)
274     #except RuntimeError:
275     #    print 'just checking for now...'
276     print 'rho_0after=', popt[1]
277     #sys.exit()
278 elif blah == 2:
279     #popt, pcov = curve_fit(dm_profile_fit_rho0_log, np.log10(r), np.log10(rho), sigma=np.log10(err))
280     popt, pcov = curve_fit(nfw_fit_rho0_log, np.log10(r), np.log10(rho), sigma=np.log10(err))
281     popt = 10.0**popt
282     pcov = 10.0**pcov
283 elif blah == 3:
284     popt, pcov = curve_fit(nfw_fit_rho0, r, rho, sigma=err)
285
286 #popt, pcov = curve_fit(dm_profile_def_rho0(R_vir), r, rho, sigma=err)
287 #popt, pcov = curve_fit(dm_profile_databin_rho0(rho_0_databin), r, rho, sigma=err)
288 #-----
289 print 'fit_params=', popt
290 print 'covariance=', pcov
291 nfw_r = np.linspace(r[0], r[-1], nfit)
292 #----- choose one fitting type -----#
293 nfw_fit = nfw_fit_rho0(nfw_r, popt[0], popt[1])
294 #nfw_fit = nfw_def_rho0(R_vir)(nfw_r, popt[0])
295 #nfw_fit = nfw_databin_rho0(rho_0_databin)(nfw_r, popt[0])
296 #nfw_fit = dm_profile_fit_rho0(nfw_r, popt[0], popt[1], popt[2])
297 #nfw_fit = dm_profile_def_rho0(R_vir)(nfw_r, popt[0], popt[1])
298 #nfw_fit = dm_profile_databin_rho0(rho_0_databin)(nfw_r, popt[0], popt[1])
299 #-----#
300 #----- choose one fitting type -----#
301 chi2_fit = nfw_fit_rho0(r, popt[0], popt[1])
302 #chi2_fit = nfw_def_rho0(R_vir)(r, popt[0])
303 #chi2_fit = nfw_databin_rho0(rho_0_databin)(r, popt[0])
304 #chi2_fit = dm_profile_fit_rho0(r, popt[0], popt[1], popt[2])
305 #chi2_fit = dm_profile_def_rho0(R_vir)(r, popt[0], popt[1])
306 #chi2_fit = dm_profile_databin_rho0(rho_0_databin)(r, popt[0], popt[1])
307 #-----#
308
309 chi2 = chisquare(np.log10(rho[first_good_bin:]), np.log10(chi2_fit[first_good_bin:]))
310 chi2_nolog = chisquare(rho[first_good_bin:], chi2_fit[first_good_bin:])
311 print 'chi_square=', chi2
312 print 'chi_square_nolog=', chi2_nolog
313 return nfw_r, nfw_fit, popt, pcov, chi2[0]
314
315
316 def draw_projection(fig, place, plot_lim, x, y):
317     ax = plt.subplot(1,3,place+1, aspect='equal')
318     im = ax.plot(x, y, linestyle=':', marker='.', markersize=1, markeredgecolor='blue')
319     ax.set_xlabel(xlabel_proj[place])
320     ax.set_ylabel(ylabel_proj[place])
321     ax.set_xlim(-plot_lim, plot_lim)
322     ax.set_ylim(-plot_lim, plot_lim)
323     # ax.xaxis.set_major_locator(MultipleLocator(tick_base_major))
324     # ax.xaxis.set_minor_locator(MultipleLocator(tick_base_minor))
325     # ax.yaxis.set_major_locator(MultipleLocator(tick_base_major))
326     # ax.yaxis.set_minor_locator(MultipleLocator(tick_base_minor))
327     return fig, ax
328
329
330 def draw_projection_again(fig, ax, x, y):
331     im = ax.plot(x, y, linestyle=':', marker='.', markersize=1, markeredgecolor='red')
332     return fig
333
334
335 def draw_density_profile(fig, r, rho, err=None):
336     ax = plt.subplot(2,1,2)
337     im = ax.loglog(r, rho, linestyle='steps-mid')
338     line1 = ax.axvline(res_limit, color='black', linestyle=':')
339     #ax.set_xlim(r_bins[0], r_bins[-1])
340     ax.set_xlim(r[0] - (r[1]-r[0]), r[-1] + (r[-1]-r[-2]))
341     ax.set_xlabel(xlabel_prof)
342     ax.set_ylabel(ylabel_prof)
343     if err != None:
344         err_bars = ax.errorbar(r, rho, yerr=err, linestyle='None')
345     return fig, ax
346
347
348 def draw_nfw_profile(fig, ax, r, rho, R_s=None):
349     ax.loglog(r, rho, linestyle='-', color='red')
350     if R_s != None:
351         line = ax.axvline(R_s, color='purple', linestyle='-.')
352     return fig
353
354
```

```

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[:,0] * vel[:,1] - pos[:,1] * vel[:,0]))
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
396         if i == 1:
397             halos1, particles1 = temp_halos1, temp_particles1
398             halos2, particles2 = temp_halos2, temp_particles2
399         else:
400             halos1 = np.append(halos1, temp_halos1, axis=0)
401             halos2 = np.append(halos2, temp_halos2, axis=0)
402             particles1 = np.append(particles1, temp_particles1, axis=0)
403             particles2 = np.append(particles2, temp_particles2, axis=0)
404
405             halos1 = np.asarray(halos1)
406             halos2 = np.asarray(halos2)
407             #indices = np.argsort(halos[:,2])           # sort by number of particles
408             #indices = indices[::-1]                   # start with the biggest
409
410             iteration = 0
411             #for index in indices[:1000]:
412             #for index in indices:
413                 for index in range(halos1.shape[0]):
414                     halo_id = halos1[index,0]
415                     if (halo_id == id1):
416                         print '-----',
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 percentage of particles for projection plots
433                         if (draw_frac < 1.0):
434                             np.random.shuffle(pos1)
435                             pos1 = pos1[:int(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]

```

```

446     r_vir2 = halos2[index][4] * dist_scale
447     halo_mass2 = halos2[index][5]
448     halo_pos2 = np.array([halos2[index][6] * dist_scale, halos2[index][7] * dist_scale, halos2[index][8] *
449     dist_scale])
450     #halo_vel2 = np.array([halos2[index][9], halos2[index][10], halos2[index][11]])
451
452     print 'Using %d particles in halo %d.' % (halo_particles2.shape[0], halo_id)
453
454     # Find center of mass
455     #pos = pos - halo_pos
456     #vel = vel - halo_vel
457
458     # Pick only a certain percentage of particles for projection plots
459     if (draw_frac < 1.0):
460         np.random.shuffle(pos2)
461         pos2 = pos2[: (draw_frac * pos2.shape[0])]
462
463     # Find the maximum of x, y, or z to be limit of projection plots
464     center = (halo_pos1 + halo_pos2) / 2.0
465     pos1 = pos1 - center
466     pos2 = pos2 - center
467     halo_pos1 = halo_pos1 - center
468     halo_pos2 = halo_pos2 - center
469     plot_lim = np.append(pos1, pos2).max()
470
471     # Plot density profile histogram
472     if (make_plot == True):
473         fig = plt.figure()
474
475         fig, ax = draw_projection(fig, 0, plot_lim, pos1[:, 0], pos1[:, 1])
476         fig = draw_projection_again(fig, ax, pos2[:, 0], pos2[:, 1])
477         ax.add_patch(Circle((halo_pos1[0], halo_pos1[1]), r_vir1, fc="None", ec="black", lw=1))
478         ax.add_patch(Circle((halo_pos2[0], halo_pos2[1]), r_vir2, fc="None", ec="black", lw=1))
479
480         fig, ax = draw_projection(fig, 1, plot_lim, pos1[:, 0], pos1[:, 2])
481         fig = draw_projection_again(fig, ax, pos2[:, 0], pos2[:, 2])
482         ax.add_patch(Circle((halo_pos1[0], halo_pos1[2]), r_vir1, fc="None", ec="black", lw=1))
483         ax.add_patch(Circle((halo_pos2[0], halo_pos2[2]), r_vir2, fc="None", ec="black", lw=1))
484
485         fig, ax = draw_projection(fig, 2, plot_lim, pos1[:, 1], pos1[:, 2])
486         fig = draw_projection_again(fig, ax, pos2[:, 1], pos2[:, 2])
487         ax.add_patch(Circle((halo_pos1[1], halo_pos1[2]), r_vir1, fc="None", ec="black", lw=1))
488         ax.add_patch(Circle((halo_pos2[1], halo_pos2[2]), r_vir2, fc="None", ec="black", lw=1))
489
490         #fig, ax = draw_density_profile(fig, mid_bins, rho, err=rho_err)
491         #fig = draw_nfw_profile(fig, ax, nfw_r, nfw_fit, R_s=scale_radius)
492         fig.tight_layout()
493         #plt.savefig(plot_base+str(itteration)+plot_ext)
494         plt.savefig('test.eps')
495
496     if __name__ == '__main__':
497         main()

```

## G.2 Density Comparison (Python)

```

1 #!/usr/bin/env python
2
3 import sys
4 import bgc2
5 import numpy as np
6 import matplotlib as mpl
7 mpl.use('Agg')
8 import matplotlib.pyplot as plt
9 from matplotlib.patches import Circle
10 from matplotlib import patheffects
11 from mpl_toolkits.axes_grid1 import ImageGrid
12 from scipy.stats import ks_2samp
13 from scipy.stats import chisquare
14 from scipy.optimize import curve_fit
15 from scipy.ndimage.filters import gaussian_filter
16 from ipdb import set_trace
17
18
19 ##### Note: only run one box pair at a time.
20 ##### ex: ./compare.py /crossmatch_dir/halos.dat /2lpt_dir/halos_0.*.bgc2 /za_dir/halos_0.*.bgc2
21
22 def main():
23     crossmatched_halo_file, bgc2_2lpt_files, bgc2_za_files = parse_args(sys.argv[1:])
24
25     header, halos = read_files(crossmatched_halo_file, header_line = 3)
26
27     bgc2_2lpt_header, bgc2_2lpt_halos, bgc2_2lpt_particles = get_bgc2_data(bgc2_2lpt_files)
28     bgc2_za_header, bgc2_za_halos, bgc2_za_particles = get_bgc2_data(bgc2_za_files)
29
30     header = np.asarray(header)
31     bgc2_2lpt_halos, bgc2_za_halos = map(np.asarray, (bgc2_2lpt_halos, bgc2_za_halos))
32
33     if sort_col != None:
34         halos = sort_by_column(halos, sort_col)
35     if remove_nonfit_halos:

```

```

36         halos = remove_nans(halos)
37     if global_filter_halos:
38         halos = filter_halos(halos)
39     if (nhalos != None) or (nhalos != 0):
40         #halos = halos[:nhalos]
41         halos = halos[[0,70]]           ##### hard coded for the moment
42         #halos = halos[10000:10050]
43
44     header, halos = add_c_columns(header, halos)
45     header = reduce_header(header)
46
47     for i, halo_pair in enumerate(halos):
48         make_plot(i, header, halo_pair, bgc2_2lpt_halos, bgc2_za_halos, \
49                    bgc2_2lpt_particles, bgc2_za_particles)
50
51 print 'Finished all plots.'
52
53
54 def parse_args(args):
55     crossmatched_halo_file = args[0]
56     if len(args[1:]) % 2 != 0:
57         print 'Must call with even number of bgc2 files... exiting.'
58         sys.exit(-1)
59     bgc2_files = args[1:]
60     bgc2_2lpt_files = bgc2_files[:len(bgc2_files)/2]
61     bgc2_za_files = bgc2_files[len(bgc2_files)/2:]
62     return crossmatched_halo_file, bgc2_2lpt_files, bgc2_za_files
63
64
65 def read_files(files, header_line = None, comment_char = '#'):
66     header = None
67     data = None
68     if type(files) == str:
69         files = [files]
70
71     if header_line != None:
72         with open(files[0], 'r') as fd:
73             for line in range(header_line):
74                 fd.readline()
75             header = fd.readline()
76             if header[0] != comment_char:
77                 print 'Header must start with a %s' % comment_char
78                 sys.exit(4)
79             header = header[1:]
80             header = header.split()
81
82     for file in files:
83         print 'Reading file %s...' % (file)
84         if data == None:
85             data = np.genfromtxt(file, comments=comment_char)
86         else:
87             data = np.append(data, np.genfromtxt(file, comments=comment_char), axis=0)
88
89 print 'Finished reading files.'
90 if header_line == None:
91     return data
92 else:
93     return header, data
94
95
96 def get_bgc2_data(bgc2_files):
97     header = None
98     halos = None
99     particles = None
100    for bgc2_file in bgc2_files:
101        print 'Reading file %s...' % (bgc2_file)
102        tmp_header, tmp_halos, tmp_particles = bgc2.read_bgc2(bgc2_file)
103        if header == None:
104            header = tmp_header
105            halos = tmp_halos
106            particles = tmp_particles
107        else:
108            halos = np.append(halos, tmp_halos, axis=0)
109            particles = np.append(particles, tmp_particles, axis=0)
110    print 'Finished reading bgc2 files.'
111    return header, halos, particles
112
113
114 def sort_by_column(halos, col):
115     print 'Sorting halos...'
116     mask = np.argsort(halos[:, col])
117     mask = mask[::-1]
118     halos = halos[mask]
119     return halos
120
121
122 def remove_nans(halos):
123     print 'Removing NaNs...'
124     halos = halos[halos[:, c_2lpt_col] != -9999]
125     halos = halos[np.isfinite(halos[:, c_2lpt_col])]
126     halos = halos[np.isfinite(halos[:, c_za_col])]
127     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
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
150     return header, halos
151
152
153 def reduce_header(header):
154     header_2lpt = header[print_cols_2lpt]
155     header_zा = header[print_cols_zा]
156     if (header_2lpt == header_zा).all():
157         header = header_2lpt
158     else:
159         print 'column mismatch... exiting'
160         set_trace()
161         sys.exit(123)
162
163     return header
164
165 def make_plot(itteration, header, halo_pair, bgc2_halos_2lpt, bgc2_halos_zा, \
166               bgc2_particles_2lpt, bgc2_particles_zा):
167     id_2lpt = halo_pair[id_col_2lpt]
168     id_zा = halo_pair[id_col_zा]
169     properties_2lpt = halo_pair[print_cols_2lpt]
170     properties_zा = halo_pair[print_cols_zा]
171
172     # find 2lpt and zा halo from id
173     halo_index_2lpt = np.where(bgc2_halos_2lpt[:, halo_id_col] == id_2lpt)[0][0]
174     halo_index_zा = np.where(bgc2_halos_zा[:, halo_id_col] == id_zा)[0][0]
175
176     bgc2_halos_2lpt = bgc2_halos_2lpt[halo_index_2lpt]
177     bgc2_halos_zा = bgc2_halos_zा[halo_index_zा]
178
179     # convert particles to numpy arrays
180     bgc2_particles_2lpt = np.asarray(bgc2_particles_2lpt[halo_index_2lpt])
181     bgc2_particles_zा = np.asarray(bgc2_particles_zा[halo_index_zा])
182
183     # make density profiles
184     r_2lpt, rho_2lpt, rho_err_2lpt, r_vir_2lpt = density.profile(bgc2_halos_2lpt, bgc2_particles_2lpt)
185     r_zा, rho_zा, rho_err_zा, r_vir_zा = density.profile(bgc2_halos_zा, bgc2_particles_zा)
186
187     # fit density profiles
188     nfw_r_2lpt, nfw_rho_2lpt, r_s_2lpt = fit_profile(r_2lpt / r_vir_2lpt, rho_2lpt / rho_2lpt.max(), err =
189             rho_err_2lpt / rho_2lpt.max())
190     nfw_r_zा, nfw_rho_zा, r_s_zा = fit_profile(r_zा / r_vir_zा, rho_zा / rho_zा.max(), err =
191             rho_err_zा / rho_zा.max())
192
193     # de-normalize values
194     nfw_r_2lpt = nfw_r_2lpt * r_vir_2lpt
195     nfw_r_zा = nfw_r_zा * r_vir_zा
196     nfw_rho_2lpt = nfw_rho_2lpt * rho_2lpt.max()
197     nfw_rho_zा = nfw_rho_zा * rho_zा.max()
198     r_s_2lpt = r_s_2lpt * r_vir_2lpt
199     r_s_zा = r_s_zा * r_vir_zा
200
201     # find center of halos and plot limit
202     halo_pos_2lpt = bgc2_halos_2lpt[:, halo_pos_cols] * dist_scale
203     halo_pos_zा = bgc2_halos_zा[:, halo_pos_cols] * dist_scale
204     particle_pos_2lpt = bgc2_particles_2lpt[:, particle_pos_cols] * dist_scale
205     particle_pos_zा = bgc2_particles_zा[:, particle_pos_cols] * dist_scale
206
207     if wrap_box:
208         for i in range(3):
209             if abs(halo_pos_2lpt[i] - halo_pos_zा[i]) > box_size / 2.0:
210                 print "#####wrapping#####wrappinghalo#####wrapping#####wrapping#####wrapping#####wrapping#####"
211                 if (halo_pos_2lpt[i] > halo_pos_zा[i]):
212                     halo_pos_zा[i] += box_size
213                     particle_pos_zा[:, i] += box_size
214                 if (halo_pos_2lpt[i] < halo_pos_zा[i]):
215                     halo_pos_zा[i] += box_size
216                     particle_pos_2lpt[:, i] += box_size
217             else:
218                 print "errorinwrapping"
219                 sys.exit()

```

```

218     center = (halo_pos_2lpt + halo_pos_z) / 2.0
219     halo_pos_2lpt = halo_pos_2lpt - center
220     halo_pos_z = halo_pos_z - center
221     particle_pos_2lpt = particle_pos_2lpt - center
222     particle_pos_z = particle_pos_z - center
223
224     if zoom_projections:
225         plot_lim = zoom_scale
226     else:
227         plot_lim = np.append(particle_pos_2lpt, particle_pos_z).max()
228
229
230     r_vir_2lpt = bgc2_halos_2lpt[halo_r_col] * dist_scale
231     r_vir_z = bgc2_halos_z[halo_r_col] * dist_scale
232
233     if make_stats:
234         print 'generating plot...'
235         fig = plt.figure(figsize = (9.0, 6.0))
236         fig = make_projections(fig, 221, halo_pos_2lpt, halo_pos_z, particle_pos_2lpt, particle_pos_z, \
237                                r_vir_2lpt, r_vir_z, plot_lim)
238         ax = fig.add_subplot(223)
239         ax = draw_density_profile(ax, r_2lpt, rho_2lpt, err=rho_err_2lpt, color='blue', label='2lpt')
240         ax = draw_density_profile(ax, r_z, rho_z, err=rho_err_z, color='red', label='z')
241
242         ax = fig.add_subplot(122)
243         ax = draw_parameters(ax, header, properties_2lpt, properties_z)
244
245         fig.tight_layout()
246         plot_name = "%s%0.3d_(%d,%d)%s" % (plot_base, itteration, id_2lpt, id_z, plot_ext)
247         plt.savefig(plot_name, bbox_inches='tight')
248         print 'finished plot' + plot_name
249
250     if make_projection:
251         print 'generating density projection plot...'
252         fig = plt.figure(figsize = (9.0, 6.0))
253
254         if label_projection:
255             ax = fig.add_subplot(111, aspect=2.0/3.2)
256             ax = hide_axes(ax)
257             ax.set_xlabel(proj_xlabel)
258             ax.set_ylabel(proj_ylabel)
259
260             fig = make_projections(fig, 111, halo_pos_2lpt, halo_pos_z, particle_pos_2lpt, particle_pos_z, \
261                                    r_vir_2lpt, r_vir_z, plot_lim)
262             fig.tight_layout()
263             plot_name = "%s%0.3d_(%d,%d)%s" % (plot_base, itteration, id_2lpt, id_z, proj_name, plot_ext)
264             plt.savefig(plot_name, bbox_inches='tight')
265             print 'finished density projection plot' + plot_name
266
267     if make_density_profile:
268         print 'generating density profile plot...'
269         fig = plt.figure(figsize = (9.0, 12.0))
270
271         if label_projection:
272             ax = fig.add_subplot(211, aspect=2.0/3.2)
273             ax = hide_axes(ax)
274             ax.set_xlabel(proj_xlabel)
275             ax.set_ylabel(proj_ylabel)
276
277             fig = make_projections(fig, 211, halo_pos_2lpt, halo_pos_z, particle_pos_2lpt, particle_pos_z, \
278                                    r_vir_2lpt, r_vir_z, plot_lim)
279
280             ax = fig.add_subplot(212)
281             ax = hide_axes(ax)
282             ax.set_xlabel(prof_xlabel)
283             ax.set_ylabel(prof_ylabel)
284
285             #grid = ImageGrid(fig, 212, nrows_ncols=(2,1), axes_pad=0.24)
286
287             ax1 = fig.add_subplot(413)
288             ax1 = draw_density_profile(ax1, r_2lpt, rho_2lpt, err=rho_err_2lpt, color='blue')
289             ax1 = draw_nfw_profile(ax1, nfw_r_2lpt, nfw_rho_2lpt, R_s=r_s_2lpt, color='red')
290
291             ax2 = fig.add_subplot(414)
292             ax2 = draw_density_profile(ax2, r_z, rho_z, err=rho_err_z, color='blue')
293             ax2 = draw_nfw_profile(ax2, nfw_r_z, nfw_rho_z, R_s=r_s_z, color='red')
294
295         if equal_profile_axes:
296             ymin = min(ax1.get_ylim()[0], ax2.get_ylim()[0])
297             ymax = max(ax1.get_ylim()[1], ax2.get_ylim()[1])
298             ax1.set_ylim(ymin, ymax)
299             ax2.set_ylim(ymin, ymax)
300
301             xmin = min(ax1.get_xlim()[0], ax2.get_xlim()[0])
302             xmax = max(ax1.get_xlim()[1], ax2.get_xlim()[1])
303             ax1.set_xlim(xmin, xmax)
304             ax2.set_xlim(xmin, xmax)
305
306         if print_text:
307             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',
309         transform=ax2.transAxes)
310
311     #fig.tight_layout()
312     plot_name = "%s%0.3d_(%d,%d)%s%s" % (plot_base, iteration, 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]

```

```

399     return x
400
401
402 def draw_hist(fig, ax, x, x_min=None, x_max=None, use_log=False, color=None, label=None):
403     if use_log:
404         xbins = np.logspace(np.log10(x_min), np.log10(x_max), num=nbins+1)
405         ax.set_xscale('log')
406     else:
407         xbins = np.linspace(x_min, x_max, num=nbins+1)
408
409     n, bins, patches = ax.hist(x, bins=xbins, histtype='step', log=ylog, color=color, label=label)
410
411     return fig, ax, n, bins, patches
412
413 def add_text(fig, ax, textstr):
414     props = dict(boxstyle='round', facecolor='white', alpha=0.7)
415     ax.text(0.02, 0.08, textstr, transform=ax.transAxes, fontsize=14,
416             verticalalignment='top', bbox=props)
417
418     return fig, ax
419
420 def make_projections(fig, position, halo_pos1, halo_pos2, pos1, pos2, r_vir1, r_vir2, plot_lim):
421     #grid = ImageGrid(fig, position, nrows_ncols=(2,3), axes_pad=0.05, cbar_mode='single')
422     grid = ImageGrid(fig, position, nrows_ncols=(2,3), axes_pad=0.12, cbar_mode='single')
423     for i, (x, y, hx, hy, r) in enumerate(zip(
424         (pos1[:,0], pos1[:,0], pos1[:,1], pos2[:,0], pos2[:,0], pos2[:,1]), \
425         (pos1[:,1], pos1[:,2], pos1[:,2], pos2[:,1], pos2[:,2], pos2[:,2]), \
426         (halo_pos1[0], halo_pos1[0], halo_pos1[1], halo_pos2[0], halo_pos2[0], halo_pos2[1]), \
427         (halo_pos1[1], halo_pos1[2], halo_pos1[2], halo_pos2[1], halo_pos2[2], halo_pos2[2]), \
428         (r_vir1, r_vir1, r_vir1, r_vir2, r_vir2, r_vir2))):
429         ax = grid[i]
430         draw_projection(ax, x, y, hx, hy, r, plot_lim)
431         if print_text:
432             if i == 0:
433                 ax.text(0.05, 0.12, '2LPT', color='white', horizontalalignment='left', verticalalignment='center',
434                         transform=ax.transAxes, path_effects=[patheffects.withStroke(linewidth=3, foreground='black')])
435             if i == 3:
436                 ax.text(0.05, 0.12, 'ZA', color='white', horizontalalignment='left', verticalalignment='center',
437                         transform=ax.transAxes, path_effects=[patheffects.withStroke(linewidth=3, foreground='black')])
438             if i == 0:
439                 ax.text(0.95, 0.88, 'XY', color='white', horizontalalignment='right', verticalalignment='center',
440                         transform=ax.transAxes, path_effects=[patheffects.withStroke(linewidth=3, foreground='black')])
441             if i == 1:
442                 ax.text(0.95, 0.88, 'XZ', color='white', horizontalalignment='right', verticalalignment='center',
443                         transform=ax.transAxes, path_effects=[patheffects.withStroke(linewidth=3, foreground='black')])
444             if i == 2:
445                 ax.text(0.95, 0.88, 'YZ', color='white', horizontalalignment='right', verticalalignment='center',
446                         transform=ax.transAxes, path_effects=[patheffects.withStroke(linewidth=3, foreground='black')])
447
448     return fig
449
450
451 def draw_projection(ax, x, y, hx, hy, r, plot_lim):
452     limits = [[-plot_lim, plot_lim], [-plot_lim, plot_lim]]
453     z, xedges, yedges = np.histogram2d(x, y, bins=npixels, range=limits)
454     if log_scale_projections:
455         z[z<1.0] = 0.5
456         #z = np.log10(z)
457         #z = np.log10(z)
458         #z[np.isinf(z)] = -0.1
459         plot_norm = mpl.colors.LogNorm(vmin = 1, vmax = z.max(), clip=True)
460         #plot_norm = None
461     else:
462         plot_norm = None
463     if extra_smoothing:
464         z = gaussian_filter(z, smoothing_radius)
465     im = ax.imshow(z.T, extent=(-plot_lim, plot_lim, -plot_lim, plot_lim), \
466                     interpolation='gaussian', origin='lower', cmap=cmap, norm=plot_norm)
467     #interpolation='gaussian', origin='lower', cmap=cmap)
468     ax.locator_params(nbins=6)
469     if draw_circle:
470         ax.add_patch(Circle((hx, hy), r, fc="None", ec="black", lw=1))
471     if draw_contours:
472         x_midpoints = (xedges[:-1] + xedges[1:]) / 2.0
473         y_midpoints = (yedges[:-1] + yedges[1:]) / 2.0
474         X, Y = np.meshgrid(x_midpoints, y_midpoints)
475         ax.contour(X, Y, z.T, 2, colors='black', linewidths=4)
476         ax.contour(X, Y, z.T, 2, colors='white', linewidths=2)
477     if label_colorbar:
478         if log_scale_projections:
479             log_format = mpl.ticker.LogFormatterMathtext(10, labelOnlyBase=False)
480             ax.cax.colorbar(im, format=log_format)
481         else:
482             ax.cax.colorbar(im)
483     else:
484         bar = ax.cax.colorbar(im, ticks[])
485         bar.ax.set_yticklabels([])
486         #plt.setp(bar.ax.get_yticklabels(), visible=False)
487
488 def draw_density_profile(ax, r, rho, err=None, color='black', label=None):
489     im = ax.loglog(r, rho, linestyle='steps-mid', color=color, label=label)

```

```

486     line1 = ax.axvline(res_limit, color='black', linestyle=':')
487     ax.set_xlim(r[0] - (r[1]-r[0]), r[-1] + (r[-1]-r[-2]))
488     #ax.set_xlabel(xlabel_prof)
489     #ax.set_ylabel(ylabel_prof)
490     if err != None:
491         err_bars = ax.errorbar(r, rho, yerr=err, linestyle='None', color=color)
492     if label != None:
493         ax.legend(fontsize='x-small')
494     return ax
495
496
497 def draw_nfw_profile(ax, r, rho, R_s=None, color='black'):
498     ax.loglog(r, rho, linestyle='--', color=color)
499     if R_s != None:
500         line = ax.axvline(R_s, color='purple', linestyle='-.')
501     return ax
502
503
504 def draw_parameters(ax, header, params1, params2):
505     strlen = 12
506     header = [str(item)[:strlen] for item in header]
507     params1 = [str(item)[:strlen] for item in params1]
508     params2 = [str(item)[:strlen] for item in params2]
509     header.insert(0, 'simulation')
510     params1.insert(0, '--_2lpt_--')
511     params2.insert(0, '---_za_---')
512     header = '\n'.join(header)
513     params1 = '\n'.join(params1)
514     params2 = '\n'.join(params2)
515     ax.text(0.05, 0.5, header, horizontalalignment="left", verticalalignment="center", transform=ax.transAxes)
516     ax.text(0.40, 0.5, params1, horizontalalignment="left", verticalalignment="center", transform=ax.transAxes)
517     ax.text(0.75, 0.5, params2, horizontalalignment="left", verticalalignment="center", transform=ax.transAxes)
518     ax.axis('off')
519     return ax
520
521
522 def hide_axes(ax):
523     ax.spines['top'].set_color('none')
524     ax.spines['bottom'].set_color('none')
525     ax.spines['left'].set_color('none')
526     ax.spines['right'].set_color('none')
527     ax.tick_params(labelcolor='w', top='off', bottom='off', left='off', right='off')
528     return ax
529
530
531
532
533 nhalos = 1
534 sort_col = 9 # density_profile 2lpt halo mass
535 #sort_col = 47 # rockstar 2lpt halo mass (M200c)
536
537 nbins = 40
538 nfit = 100
539 npixels = 30
540 #npixels = 100
541 smoothing_radius = 0.9
542 remove_nonfit_halos = True
543 global_filter_halos = True
544 column_filter_halos = True
545 log_scale_projections = True
546 wrap_box = False
547 label_colorbar = False
548 label_projection = True
549 zoom_projections = True
550 zoom_scale = 18.0 # kpc
551 draw_circle = False
552 draw_contours = True
553 extra_smoothing = True
554 label_proj = True
555 label_2lpt_z = True
556 equal_profile_axes = True
557 print_text = True
558
559 box_size = 10000.0 # kpc
560
561 id_col_2lpt = 0
562 id_col_z = 1
563
564 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]
565 print_cols_z = [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]
566
567 Rv1_col = 53
568 Rv2_col = 54
569 Rs1_col = 55
570 Rs2_col = 56
571
572 c_2lpt_col = 17
573 c_z_col = 18
574
575 # c_2lpt, c_z, chi2_2lpt, chi2_z

```

```

576 lt_cols = [17, 18, 37, 38]
577 lt_vals = [100.0, 100.0, 10.0, 10.0]
578
579 # c_2lpt, c_za, rho_0_2lpt, rho_0_za, chi2_2lpt, chi2_za
580 gt_cols = [17, 18, 31, 32, 37, 38]
581 gt_vals = [1.0, 1.0, 0.0, 0.0, 0.0, 0.0]
582
583 eq_cols = []
584 eq_vals = []
585
586 ne_cols = []
587 ne_vals = []
588
589 # bgc2 halo array columns
590 halo_id_col = 0
591 halo_r_col = 4
592 halo_mass_col = 5
593 halo_pos_cols = [6,7,8]
594
595 # bgc2 particle array columns
596 particle_mass_col = 0
597 particle_pos_cols = [1,2,3]
598 particle_vel_cols = [4,5,6]
599
600 mass_scale = 1.0
601 common_mass = 5.33423e5
602 dist_scale = 1.0e3
603 res_limit = 0.5 #changed from 4.0 to 0.5 to match density_profile.py <-- maybe check why it was 4.0?
604 nfit = 500
605
606 dist_units = 'kpc'
607 xlabel_proj = r'X Position (%s h$^{\{-1\}})' % (dist_units), r'X Position (%s h$^{\{-1\}})' % (dist_units), r'Y
   Position (%s h$^{\{-1\}})' % (dist_units)
608 ylabel_proj = r'Y Position (%s h$^{\{-1\}})' % (dist_units), r'Z Position (%s h$^{\{-1\}})' % (dist_units), r'Z
   Position (%s h$^{\{-1\}})' % (dist_units)
609 proj_xlabel = r'Position_(kpc_h$^{\{-1\}})'
610 proj_ylabel = r'Position_(kpc_h$^{\{-1\}})'
611 prof_xlabel = r'Radius_(%suh$^{\{-1\}})' % (dist_units)
612 prof_ylabel = r'Density_(M$_{\odot}$%s$^{\{-3\}}$uh$^{\{2\}}$)' % (dist_units)
613
614 colormap = 'ocean_r'
615 colormap = 'rainbow'
616 plot_base = 'plots/halo_pair_'
617 proj_name = '_proj'
618 dens_name = '_dens'
619 plot_ext = '.eps'
620
621 make_stats = False
622 make_projection = False
623 make_density_profile = True
624
625 plot_dest_type = 'paper'
626 if plot_dest_type == 'paper':
627     mpl.rcParams['font.family'] = 'serif'
628     mpl.rcParams['font.size'] = 16
629     mpl.rcParams['axes.linewidth'] = 3
630     mpl.rcParams['lines.linewidth'] = 4
631     mpl.rcParams['patch.linewidth'] = 4
632     mpl.rcParams['xtick.major.width'] = 3
633     mpl.rcParams['ytick.major.width'] = 3
634     mpl.rcParams['xtick.major.size'] = 8
635     mpl.rcParams['ytick.major.size'] = 8
636
637
638 if __name__ == '__main__':
639     main()

```

## Appendix H

### Concentration Comparison Code (Python)

```

1 #!/usr/bin/env python
2
3 import sys
4 import numpy as np
5 from ipdb import set_trace
6
7 def main():
8     # Read in particle files
9     header, halos = read_files(sys.argv[1:], header_line = 3)
10
11    if remove_nonfit_halos:
12        print 'Removing NaNs...'
13        halos = halos[np.isfinite(halos[:,c_lpt_col])]
14        halos = halos[np.isfinite(halos[:,c_za_col])]
15
16    if global_filter_halos:
17        print 'Filtering data...'
18        for col, val in zip(glob_lt_cols, glob_lt_vals):
19            halos = halos[halos[:, col] <= val]
20        for col, val in zip(glob_gt_cols, glob_gt_vals):
21            halos = halos[halos[:, col] >= val]
22        for col, val in zip(glob_eq_cols, glob_eq_vals):
23            halos = halos[halos[:, col] == val]
24        for col, val in zip(glob_ne_cols, glob_ne_vals):
25            halos = halos[halos[:, col] != val]
26
27
28    if sort_col != None:
29        halos = sort_by_column(halos, sort_col)
30    if (nhalos != None) or (nhalos != 0):
31        halos = halos[:nhalos]
32    #if (nhalos == 'perc25'):
33    #    halos = halos[:len(halos)/10]
34    if bad_halo_pairs != None:
35        mask = np.arange(len(halos))
36        mask = np.in1d(mask, bad_halo_pairs)
37        mask = np.invert(mask)
38        halos = halos[mask]
39
40    c_rockstar_2lpt = halos[:, Rv1_col] / halos[:, Rs1_col]
41    c_rockstar_za = halos[:, Rv2_col] / halos[:, Rs2_col]
42
43    if use_klypin:
44        mask = (halos[:,4] < 100)
45        print "changed %d halos" % (mask.sum())
46        print "c_2lpt_before", c_rockstar_2lpt[mask][0]
47        c_rockstar_2lpt[mask] = halos[mask, Rv1_col] / halos[mask, 79]
48        print "c_2lpt_klypin", c_rockstar_2lpt[mask][0]
49        mask = (halos[:,5] < 100)
50        print "changed %d halos" % (mask.sum())
51        print "c_za_before", c_rockstar_za[mask][0]
52        c_rockstar_za[mask] = halos[mask, Rv2_col] / halos[mask, 80]
53        print "c_za_klypin", c_rockstar_za[mask][0]
54    c_diff_2lpt = 2.0 * (c_rockstar_2lpt - halos[:, c_lpt_col]) / (c_rockstar_2lpt + halos[:, c_lpt_col])
55    c_diff_za = 2.0 * (c_rockstar_za - halos[:, c_za_col]) / (c_rockstar_za + halos[:, c_za_col])
56    #halos = np.column_stack((halos, c_rockstar_2lpt, c_rockstar_za, c_diff_2lpt, c_diff_za))
57    #header.append('c_rockstar')
58    #header.append('c_rockstar')
59    #header.append('c_diff')
60    #header.append('c_diff')
61
62    c_diff_2lpt = c_diff_2lpt[np.isfinite(c_diff_2lpt)]
63    c_diff_za = c_diff_za[np.isfinite(c_diff_za)]
64    c_diff_tot = np.append(c_diff_2lpt, c_diff_za)
65
66    c_diff_2lpt_frac = (np.abs(c_diff_za) <= cutoff_diff_frac).sum() / float(len(c_diff_2lpt))
67    c_diff_za_frac = (np.abs(c_diff_za) <= cutoff_diff_frac).sum() / float(len(c_diff_za))
68    c_diff_tot_frac = (np.abs(c_diff_tot) <= cutoff_diff_frac).sum() / float(len(c_diff_tot))
69
70    with open(c_diff_file, 'w') as fd:
71        fd.write("%g %g %g\n" % (c_diff_tot_frac, c_diff_za_frac, c_diff_2lpt_frac))
72
73    print 'Finished snapshot.'
74
75 def read_files(files, header_line = None, comment_char = '#'):
76     header = None
77     data = None
78     if type(files) == str:
79         files = [files]
80
81     if header_line != None:
82         with open(files[0], 'r') as fd:

```

```

83         for line in range(header_line):
84             fd.readline()
85             header = fd.readline()
86             if header[0] != comment_char:
87                 print "Header must start with a '%s'" % comment_char
88                 sys.exit(4)
89             header = header[1:]
90             header = header.split()
91
92     for file in files:
93         print 'Reading file %s...' % (file)
94         if data == None:
95             data = np.genfromtxt(file, comments=comment_char)
96         else:
97             data = np.append(data, np.genfromtxt(file, comments=comment_char), axis=0)
98
99     print 'Finished reading files.'
100    if header_line == None:
101        return data
102    else:
103        return header, data
104
105
106 def sort_by_column(halos, col):
107     print 'Sorting halos...'
108     mask = np.argsort(halos[:, col])
109     mask = mask[::-1]
110     halos = halos[mask]
111     return halos
112
113
114
115 remove_nonfit_halos = False
116 global.filter_halos = True
117 use_klypin = False
118
119 nhalos = 100
120 #nhalos = 'perc25'
121 #sort_col = None
122 sort_col = 9
123
124 cutoff_diff_frac = 0.2
125
126
127 Rv1_col = 53
128 Rv2_col = 54
129 Rs1_col = 55
130 Rs2_col = 56
131
132 c_lpt_col = 17
133 c_za_col = 18
134
135
136 lt_cols = [17, 18]
137 lt_vals = [100.0, 100.0]
138
139 gt_cols = [17, 18, 31, 32]
140 gt_vals = [1.0, 1.0, 0.0, 0.0]
141
142 eq_cols = []
143 eq_vals = []
144
145 ne_cols = []
146 ne_vals = []
147
148
149 # global filters
150 glob_lt_cols = []
151 glob_lt_vals = []
152
153 glob_gt_cols = [4, 5]
154 glob_gt_vals = [100, 100]
155
156 glob_eq_cols = [109, 110]
157 glob_eq_vals = [-1, -1]
158
159 glob_ne_cols = []
160 glob_ne_vals = []
161
162 bad_halo_pairs = None
163
164 c_diff_file = 'stats/c_diff.dat'
165
166
167
168 if __name__ == '__main__':
169     main()

```

## Appendix I

### Differential Histogram Code

#### I.1 Histogram Generation and Fitting (Python)

```
1 #!/usr/bin/env python
2
3 import sys
4 import numpy as np
5 import matplotlib as mpl
6 mpl.use('Agg')
7 import matplotlib.pyplot as plt
8 import matplotlib.gridspec as gridspec
9 from scipy import stats
10 from scipy.special import gamma as gamma_func
11 from scipy.optimize import curve_fit
12 import statsmodels.sandbox.distributions.extras as extrastats
13 from ipdb import set_trace
14
15 def main():
16     # Read in particle files
17     header, halos = read_files(sys.argv[1:], header_line = 3)
18
19     if remove_nonfit_halos:
20         print 'Removing NaNs...'
21         halos = halos[np.isfinite(halos[:,c_lpt_col])]
22         halos = halos[np.isfinite(halos[:,c_zs_col])]
23
24     if global_filter_halos:
25         print 'Filtering data...'
26         for col, val in zip(glob_lt_cols, glob_lt_vals):
27             halos = halos[halos[:, col] <= val]
28         for col, val in zip(glob_gt_cols, glob_gt_vals):
29             halos = halos[halos[:, col] >= val]
30         for col, val in zip(glob_eq_cols, glob_eq_vals):
31             halos = halos[halos[:, col] == val]
32         for col, val in zip(glob_ne_cols, glob_ne_vals):
33             halos = halos[halos[:, col] != val]
34
35
36     if sort_col != None:
37         halos = sort_by_column(halos, sort_col)
38     if (nhalos != None) or (nhalos != 0):
39         halos = halos[:nhalos]
40     if bad_halo_pairs != None:
41         mask = np.arange(len(halos))
42         mask = np.in1d(mask, bad_halo_pairs)
43         mask = np.invert(mask)
44         halos = halos[mask]
45
46     c_rockstar_2lpt = halos[:, Rv1_col] / halos[:, Rs1_col]
47     c_rockstar_zs = halos[:, Rv2_col] / halos[:, Rs2_col]
48     if use_klypin:
49         mask = (halos[:,4] < 100)
50         print "changed %d halos" % (mask.sum())
51         print "c_2lpt_before", c_rockstar_2lpt[mask][0]
52         c_rockstar_2lpt[mask] = halos[mask, Rv1_col] / halos[mask, 79]
53         print "c_2lpt_klypin", c_rockstar_2lpt[mask][0]
54         mask = (halos[:,5] < 100)
55         print "changed %d halos" % (mask.sum())
56         print "c_zs_before", c_rockstar_zs[mask][0]
57         c_rockstar_zs[mask] = halos[mask, Rv2_col] / halos[mask, 80]
58         print "c_zs_klypin", c_rockstar_zs[mask][0]
59         c_diff_2lpt = 2.0 * (c_rockstar_2lpt - halos[:, c_lpt_col]) / (c_rockstar_2lpt + halos[:, c_lpt_col])
60         c_diff_zs = 2.0 * (c_rockstar_zs - halos[:, c_zs_col]) / (c_rockstar_zs + halos[:, c_zs_col])
61         halos = np.column_stack((halos, c_rockstar_2lpt, c_rockstar_zs, c_diff_2lpt, c_diff_zs))
62         header.append('c_rockstar')
63         header.append('c_rockstar')
64         header.append('c_diff')
65         header.append('c_diff')
66
67     if mass_quartiles and len(halos) > 50:
68         start_fracs = [0.0, 0.25, 0.50, 0.75, 0.0]
69         end_fracs = [0.25, 0.50, 0.75, 1.0, 1.0]
70     else:
71         start_fracs = [0.0]
72         end_fracs = [1.0]
73
74     for start_frac, end_frac in zip(start_fracs, end_fracs):
75         halos_to_pass = halos[start_frac * len(halos) : end_frac * len(halos)]
76         if use_alt_frac and (start_frac == 0.0) and (end_frac == 1.0):
77             alt_halos_to_pass = halos[alt_start_frac * len(halos) : alt_end_frac * len(halos)]
78         else:
79             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,
84             header, use_log=True)
85         for (lpt_col, za_col, fancy_x_label) in zip(lpt_cols, za_cols, fancy_x_labels):
86             make_plot(halos_to_pass, alt_halos_to_pass, lpt_col, za_col, start_frac, end_frac, fancy_x_label,
87             header, use_log=False)
88
89     print 'Finished all plots.'
90
91 def read_files(files, header_line = None, comment_char = '#'):
92     header = None
93     data = None
94     if type(files) == str:
95         files = [files]
96
97     if header_line != None:
98         with open(files[0], 'r') as fd:
99             for line in range(header_line):
100                 fd.readline()
101             header = fd.readline()
102             if header[0] != comment_char:
103                 print 'Header must start with %s' % comment_char
104                 sys.exit(4)
105             header = header[1:]
106             header = header.split()
107
108     for file in files:
109         print 'Reading file %s...' % (file)
110         if data == None:
111             data = np.genfromtxt(file, comments=comment_char)
112         else:
113             data = np.append(data, np.genfromtxt(file, comments=comment_char), axis=0)
114
115     print 'Finished reading files.'
116     if header_line == None:
117         return data
118     else:
119         return header, data
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 def make_plot(halos, alt_halos, lpt_col, za_col, start_frac, end_frac, fancy_x_label, header=None, use_log=False):
129     :
130     print 'start=%s, start_frac'
131     print 'end=%s, 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,
162                          stats_ext)
163         perc_diff_stats(x_perc_diff, perc_diff_file, use_log=use_log)
164         print 'done.'
165
166     x = 2.0 * (x_lpt - x_za) / (x_lpt + x_za)
167     x[np.logical_and(x_lpt == 0, x_za == 0)] = 0
168
169     if alt_halos != None:

```

```

169     alt_x = 2.0 * (alt_x_lpt - alt_x_za) / (alt_x_lpt + alt_x_za)
170     alt_x[np.logical_and(alt_x_lpt == 0, alt_x_za == 0)] = 0
171
172 #    set_trace()
173
174 if x_lim == None:
175     #x_max = max(abs(x.max()), abs(x.min()))
176     if lpt_col == 47:
177         x_max = x.mean() + x.std() * 1.5
178         x_min = x.mean() - x.std() * 1.5
179     else:
180         x_max = np.std(x) * 3.0
181         x_min = -x_max
182     else:
183         x_max = x_lim
184         x_min = -x_lim
185
186 #    get stats
187 data_mean = x.mean()
188 data_stdev = x.std()**2
189 data_skew = stats.skew(x)
190 data_kurt = stats.kurtosis(x)
191 data_rms = np.sqrt(np.mean(x**2))
192 data_gt_epsilon = float(len(x[np.abs(x) >= 0.1])) / float(len(x))
193
194 # Generate plot
195 print 'generating', xlabel, 'plot...'
196 fig = plt.figure(figsize=(9.0, 6.0))
197 if add_residuals_panel:
198     grid = gridspec.GridSpec(2, 1, height_ratios=[1,4])
199     ax = fig.add_subplot(grid[1])
200 else:
201     ax = fig.add_subplot(111)
202 ax, n, bins, patches = draw_hist(ax, x, x_min=x_min, x_max=x_max, \
203                                     use_log=use_log, color='blue', fill=None)
204
205 p0 = [1.0, data_mean, data_stdev, 2.0]
206 ax, fit_height, fit_mean, fit_stdev, fit_skew, fit_kurt, fit_height_err, fit_mean_err, fit_stdev_err,
207     fit_skew_err, fit_kurt_err, chi2, pval = draw_fit(ax, n, bins, p0)
208
209 if draw_data_fit:
210     ax = draw_data_gaussian(ax, x, n, bins)
211
212 if alt_halos != None:
213     ax, n_alt, bins_alt, patches_alt = draw_hist(ax, alt_x, x_min=x_min, x_max=x_max, \
214                                                 use_log=use_log, color='green', fill="0.75")
215     #ax = draw_fit(ax, n, bins)
216
217 #ax.grid(color='gray', linestyle='dashed')
218 ax.set_xlim([x_min, x_max])
219 #ax.set_xlabel('(' + xlabel + '_lpt - ' + xlabel + '_za) / ' + xlabel + '_avg')
220 #ax.set_ylabel(ylabel)
221 if label_axes:
222     ax.set_xlabel(fancy_x_label, fontsize="xx-large")
223     ax.set_ylabel(fancy_y_label, fontsize="xx-large")
224 #ax.legend()
225
226 if add_residuals_panel:
227     ax = fig.add_subplot(grid[0])
228     ax = draw_residuals(ax, n, bins, fit_height, fit_mean, fit_stdev, fit_kurt)
229     ax.tick_params(axis='x', labelbottom='off')
230
231 fig.tight_layout()
232 plot_name = "%s%0.3d%s%0.3d%s(%s-%s)%s" % \
233     (plot_base, '(', lpt_col, ',', za_col, ')', xlabel, start_frac, end_frac, plot_ext)
234 fig.savefig(plot_name, bbox_inches='tight')
235
236 if save_stats:
237     statsfile = "%s%0.3d%s%0.3d%s(%s-%s)%s" % \
238         (stats_base, '(', lpt_col, ',', za_col, ')', xlabel, start_frac, end_frac, stats_ext)
239     with open(statsfile, 'w') as fd:
240         if bin_test:
241             for ntestbins in range(nbins_min, nbins_max+1, 5):
242                 fit_mean, fit_stdev = rebin_stats(ntestbins, x, x_min=x_min, x_max=x_max, use_log=use_log)
243                 fd.write("%d%g%g%g%g%g%g%g%g%g%g%g%g%g%g%g%g%g%g\n" % \
244                     (ntestbins, data_mean, data_stdev, data_skew, data_kurt, \
245                      fit_height, fit_height_err, fit_mean, fit_mean_err, fit_stdev, fit_stdev_err, fit_skew,
246                      fit_skew_err, fit_kurt_err, \
247                      data_rms, data_gt_epsilon, chi2, pval))
248
249 print 'finished uplot' + plot_name
250 return
251
252
253 def perc_diff_stats(x, filename, use_log=False):
254     data_mean = x.mean()
255     data_stdev = x.std()**2
256     data_skew = stats.skew(x)
257     data_kurt = stats.kurtosis(x)
258     data_rms = np.sqrt(np.mean(x**2))

```

```

259     data_gt_epsilon = float(len(x[np.abs(x) >= 0.1])) / float(len(x))
260
261     if x_lim == None:
262         x_max = min((x.mean() + x.std() * 3.0), x.max())
263         x_min = max((x.mean() - x.std() * 3.0), x.min())
264     else:
265         x_max = x_lim
266         x_min = -x_max
267
268     global nbins
269     if nbins <= 0:
270         nbins = np.sqrt(len(x))
271         if nbins % 2 == 0:
272             nbins = nbins - 1
273     if nbins < nbins_min:
274         nbins = nbins_min
275     elif nbins > nbins_max:
276         nbins = nbins_max
277
278     if use_log:
279         xbins = np.logspace(np.log10(x_min), np.log10(x_max), num=nbins+1)
280         mid_bins = 10.0**((0.5 * (np.log10(xbins[1:]) + np.log10(xbins[:-1]))))
281     else:
282         xbins = np.linspace(x_min, x_max, num=nbins+1)
283         mid_bins = 0.5 * (xbins[1:] + xbins[:-1])
284
285     hist, bin_edges = np.histogram(x, bins=xbins)
286     x_peak = mid_bins[hist == hist.max()][0]
287
288     x_sorted = np.sort(x)
289     n_halos = len(x_sorted)
290
291     x_vals = []
292     for frac in fractions:
293         x_vals.append(x_sorted[len(x_sorted)*frac])
294     x_vals = np.array(x_vals)
295
296     sum_frac_halos = []
297     for diff_val in diff_vals:
298         n_gt_val = (x_sorted >= diff_val).sum()
299         sum_frac_halos.append(float(n_gt_val) / float(n_halos))
300     sum_frac_halos = np.array(sum_frac_halos)
301
302     doublesum_frac_halos = []
303     for right_diff_val in diff_vals:
304         left_diff_val = (1.0 / (right_diff_val + 1.0)) - 1.0
305         n_gt_val = (x_sorted >= right_diff_val).sum() + (x_sorted <= left_diff_val).sum()
306         doublesum_frac_halos.append(float(n_gt_val) / float(n_halos))
307     doublesum_frac_halos = np.array(doublesum_frac_halos)
308
309     with open(filename, 'w') as fd:
310         fd.write("%d\n" % nbins, x_peak, \
311                 nbins, x_peak, \
312                 '\n'.join(["%g" % x for x in x_vals]), \
313                 '\n'.join(["%g" % x for x in sum_frac_halos]), \
314                 '\n'.join(["%g" % x for x in doublesum_frac_halos]), \
315                 data_mean, data_stdev, data_skew, data_kurt, \
316                 data_rms, data_gt_epsilon)
317
318     return
319
320
321 def find_frac_bounds(hist, start_bin, frac):
322     n_tot = hist.sum()
323     n_sum = hist[start_bin]
324
325     left_tot = hist[:start_bin].sum() + hist[start_bin]/2.0
326     right_tot = hist[start_bin+1:].sum() + hist[start_bin]/2.0
327
328     if float(left_tot) / float(n_tot) <= frac / 2.0:
329         right_only = True
330     if float(right_tot) / float(n_tot) <= frac / 2.0:
331         left_only = True
332
333     left_bound = start_bin
334     right_bound = start_bin
335     while(float(n_sum) / float(n_tot) < frac):
336
337         pass
338
339     return left_bound, right_bound
340
341
342 def filter(x_lpt, x_za, lpt_col, za_col):
343     mask = np.isfinite(x_lpt)
344     x_lpt = x_lpt[mask]
345     x_za = x_za[mask]
346     mask = np.isfinite(x_za)
347     x_lpt = x_lpt[mask]
348     x_za = x_za[mask]
349
350     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
389     return x1, x2
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=
414         hist_cumulative, log=ylog, edgecolor=color, label=label)
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())),
439             p0=p0, sigma=sigma)
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=
444             sigma)
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
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
455 [1,1]), np.sqrt(var_matrix[2,2]), 0.0, np.sqrt(var_matrix[3,3])
456
457 fit_x = np.linspace(bin_edges[0], bin_edges[-1], nfitpoints+1)
458 hist_fit = generalized_normal(fit_x, height, mean, stdv, kurt)
459 ax.plot(fit_x, hist_fit, color='red', linestyle='--')
460
461 chi2_fit = generalized_normal(bin_centers, height, mean, stdv, kurt)
462 chi2, pval = stats.chisquare(hist / hist.max(), chi2_fit / hist.max())
463
464 return ax, height, mean, stdv, skew, kurt, height_err, mean_err, stdv_err, skew_err, kurt_err, chi2, pval
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).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
524     mean
525 #def log_double_gaussian(x, A1, mu1, sigma1, skew1, kurtosis1, A2, mu2, sigma2, skew2, kurtosis2):
526 def log_double_gaussian(x, A1, mu1, sigma1, skew1, kurtosis1, A2, mu2, sigma2, skew2, kurtosis2):
527     #mu2 = mu1 # for common mean
528     A1 = A1**2 # remember to also square fit value for A
529     A2 = A2**2
530     skew1 = 0.0

```

```

530     skew2 = 0.0
531     kurtosis1 = 0.0
532     kurtosis2 = 0.0
533     y = double_gaussian(x, A1, mu1, sigma1, skew1, kurtosis1, A2, mu2, sigma2, skew2, kurtosis2)
534     if (y <= 0.0).any():
535         #y[y<=0] = -y[y<=0] + 1
536         y[y<=0] = (y[y<=0] + 0.0001)**2
537     return np.log10(y)
538
539
540 def gaussian(x, A, mu, sigma):
541     return A * np.exp(-(x - mu)**2 / (2.0 * sigma**2))
542
543
544 def generalized_normal(x, A, mu, alpha, beta):
545     if prevent_small_shape_param and beta < 1.0:
546         beta = 1.0 / beta
547     return A * (beta / (2.0 * alpha * gamma_func(1.0 / beta))) * np.exp(-(np.abs(x - mu)/alpha)**beta)
548
549
550 def log_generalized_normal(x, A, mu, alpha, beta):
551     A = A**2
552     y = generalized_normal(x, A, mu, alpha, beta)
553     if (y <= 0.0).any():
554         #y[y<=0] = -y[y<=0] + 1.0
555         y[y<=0] = (y[y<=0] + 0.0001)**2
556     return np.log10(y)
557
558
559 def add_text(fig, ax, textstr):
560     #props = dict(boxstyle='round', facecolor='white', alpha=0.25)
561     props = dict(edgecolor='none', facecolor='none')
562     ax.text(0.02, 0.16, textstr, transform=ax.transAxes, fontsize=14,
563             verticalalignment='top', bbox=props)
564     return fig, ax
565
566
567 def rebin_stats(ntestbins, x, x_min=None, x_max=None, use_log=False):
568     if use_log:
569         xbins = np.logspace(np.log10(x_min), np.log10(x_max), num=ntestbins+1)
570     else:
571         xbins = np.linspace(x_min, x_max, num=ntestbins+1)
572
573     hist, bin_edges = np.histogram(x, bins=xbins)
574
575     bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2.0
576     if ignore_central_bin:
577         mask = (np.abs(bin_centers) > 0.000001)
578         bin_centers = bin_centers[mask]
579         hist = hist[mask]
580     p0 = [hist.max(), 0.0, 0.2]
581     p0 = [hist.max(), hist.mean(), hist.std(), stats.skew(hist), stats.kurtosis(hist)]
582     hist[hist==0] = 1 #fix devide by zero error
583     try:
584         if poisson_weight:
585             coeffs, var_matrix = curve_fit(gaussian, bin_centers, hist, p0=p0, sigma=(np.sqrt(hist)/hist))
586         else:
587             coeffs, var_matrix = curve_fit(gaussian, bin_centers, hist, p0=p0)
588     except RuntimeError:
589         print '*****curve_fit failed!'
590     return np.nan, np.nan
591
592     mean, stdev = coeffs[1], coeffs[2]
593     return mean, stdev
594
595
596 nbins = 35
597 #nbins = 25
598 #nbins = -1
599 nbins_min = 15
600 nbins_max = 200
601 #nbins_max = 200
602 nfitpoints = 100
603 remove_nofit_halos = False
604 global_filter_halos = True
605 column_filter_halos = True
606 use_klypin = False
607 label_axes = True
608 ignore_central_bin = False
609 save_stats = True
610 bin_test = False
611 poisson_weight = True
612 fit_in_log = True
613 draw_data_fit = False
614 mass_quartiles = False
615 prevent_small_shape_param = False
616 add_residuals_panel = False
617 perc_diff = True
618
619 hist_normed = False
620 hist_cumulative = False
621 ylog = False

```

```

622 ylabel= 'Number\u00d7of\u00d7Halos'
623
624 #
625 fractions = [0.01, 0.05, 0.10, 0.25, 0.50, 0.75, 0.90, 0.95, 0.99]
626 diff_vals = [0.01, 0.05, 0.10, 0.25, 0.50, 0.75, 1.00, 2.00, 4.00]
627 #
628
629 #nhalos = 100
630 nhalos = None
631 sort_col = 9
632
633 #lpt_log_cols = [ 9, 23, 31, 47, 51, 57]
634 #za_log_cols = [10, 24, 32, 48, 52, 58]
635 #lpt_cols = [17, 77, 91, 93, 97, 99, 107, 111, -4, -2]
636 #za_cols = [18, 78, 92, 94, 98, 100, 108, 112, -3, -1]
637
638 lpt_log_cols = []
639 za_log_cols = []
640 #lpt_cols = [-4, 47, 91, 107, 111]
641 #za_cols = [-3, 48, 92, 108, 112]
642 #lpt_cols = [-4, 31, 47, 91, 107, 111]
643 #za_cols = [-3, 32, 48, 92, 108, 112]
644 #lpt_cols = [-4, 31, 47, 91, 111]
645 #za_cols = [-3, 32, 48, 92, 112]
646 lpt_cols = [-4, 47, 91, 93, 107]
647 za_cols = [-3, 48, 92, 94, 108]
648 # concentration, mass, x_off, v_off, T/|U|
649
650 fancy_log_x_labels = []
651 #fancy_x_labels = [r"\frac{c_{2LPT} - c_{ZA}}{c_{avg}}",
652 #                   r"\frac{\rho_0, 2LPT} - \rho_0, ZA}{\rho_0, avg}}",
653 #                   r"\frac{M_{vir}, 2LPT} - M_{vir, ZA}}{M_{vir, avg}}",
654 #                   r"\frac{X_{off}, 2LPT} - X_{off, ZA}}{X_{off, avg}}",
655 #                   r"\frac{N_{subs}, 2LPT} - N_{subs, ZA}}{N_{subs, avg}}"]
656
657 fancy_x_labels = [r"\frac{c_{2LPT} - c_{ZA}}{c_{avg}}",
658                   r"\frac{M_{vir}, 2LPT} - M_{vir, ZA}}{M_{vir, avg}}",
659                   r"\frac{X_{off}, 2LPT} - X_{off, ZA}}{X_{off, avg}}",
660                   r"\frac{V_{off}, 2LPT} - V_{off, ZA}}{V_{off, avg}}",
661                   r"\frac{(T/|U|)_{2LPT} - (T/|U|)_{ZA}}{(T/|U|)_{avg}}"]
662
663 fancy_y_label = r"N_{halos}"
664
665 Rv1_col = 53
666 Rv2_col = 54
667 Rs1_col = 55
668 Rs2_col = 56
669
670 c_lpt_col = 17
671 c_za_col = 18
672
673
674 # c_2lpt, c_za, chi2_2lpt, chi2_za
675 #lt_cols = [17, 18, 37, 38]
676 #lt_vals = [100.0, 100.0, 10.0, 10.0]
677 lt_cols = [17, 18]
678 lt_vals = [100.0, 100.0]
679
680 # c_2lpt, c_za, rho_0_2lpt, rho_0_za, chi2_2lpt, chi2_za
681 #gt_cols = [17, 18, 31, 32, 37, 38]
682 #gt_vals = [1.0, 1.0, 0.0, 0.0, 0.0, 0.0]
683 gt_cols = [17, 18, 31, 32]
684 gt_vals = [1.0, 1.0, 0.0, 0.0]
685
686 eq_cols = []
687 eq_vals = []
688
689 ne_cols = []
690 ne_vals = []
691
692
693 # global filters
694 glob_lt_cols = []
695 glob_lt_vals = []
696
697 glob_gt_cols = [4, 5]
698 glob_gt_vals = [100, 100]
699
700 glob_eq_cols = [109, 110]
701 glob_eq_vals = [-1, -1]
702
703 glob_ne_cols = []
704 glob_ne_vals = []
705
706
707
708 use_alt_frac = True
709 alt_start_frac = 0.75
710 alt_end_frac = 1.0
711
712 #x_lim = 0.5
713 #x_lim = 1.0

```

```

714 x_lim = None
715
716 bad_halo_pairs = None
717 #bad_halo_pairs = [9, 28, 39, 51, 59, 95]
718
719 perc_diff_base = 'plots/perc_diff_'
720 statsfile = 'plots/stats.txt'
721 stats_base = 'plots/stats_'
722 stats_ext = '.txt'
723 plot_base = 'plots/hist_'
724 plot_ext = '.eps'
725
726 plot_dest_type = 'paper'
727 if plot_dest_type == 'paper':
728     mpl.rcParams['font.family'] = 'serif'
729     mpl.rcParams['font.size'] = 16
730     mpl.rcParams['axes.linewidth'] = 3
731     mpl.rcParams['lines.linewidth'] = 4
732     mpl.rcParams['patch.linewidth'] = 4
733     mpl.rcParams['xtick.major.width'] = 3
734     mpl.rcParams['ytick.major.width'] = 3
735     mpl.rcParams['xtick.major.size'] = 8
736     mpl.rcParams['ytick.major.size'] = 8
737
738 if __name__ == '__main__':
739     main()

```

## I.2 PBS Submission Script (Bash)

```

1#!/usr/bin/env bash
2#PBS -M djsissom@gmail.com
3#PBS -m bae
4#PBS -l nodes=1:ppn=1
5#PBS -l pmem=4000mb
6#PBS -l mem=4000mb
7#PBS -l walltime=1:00:00
8#PBS -o out.log
9#PBS -j oe
10
11 minsnap=0
12 maxsnap=61
13
14 # Change to working directory
15 echo $PBS_NODEFILE
16 cd $PBS_O_WORKDIR
17
18 for ((snap=$minsnap; snap<=$maxsnap; snap++)); do
19
20     if [ $snap -lt 10 ]; then
21         j=0$snaps
22     elif [ $snap -lt 100 ]; then
23         j=0$snaps
24     fi
25
26     new_plot_dir=snap{j}_plots
27
28     if [ ! -e plots_all_snaps/${new_plot_dir} ]; then
29         mkdir plots_all_snaps/${new_plot_dir}
30     fi
31
32     echo "Starting_box{i}_snap{j}..."
33     ./hist.py ~/projects/simulations/rockstar/box{1,2,3}/crossmatch/snap{j}/halos.dat > plots/out.log 2>&1
34     mv plots/* plots_all_snaps/${new_plot_dir}/.
35     echo "Finished_snap{j}"
36
37 done
38
39 wait
40
41 # - end of script

```

## I.3 PBS Submission Script - Individual Boxes (Bash)

```

1#!/usr/bin/env bash
2#PBS -M djsissom@gmail.com
3#PBS -m bae
4#PBS -l nodes=1:ppn=1
5#PBS -l pmem=4000mb
6#PBS -l mem=4000mb
7#PBS -l walltime=2:00:00
8#PBS -o out.log
9#PBS -j oe
10
11 minsnap=0
12 maxsnap=61
13
14 minbox=1
15 maxbox=3
16

```

```

17 # Change to working directory
18 echo $PBS_NODEFILE
19 cd $PBS_O_WORKDIR
20
21
22 for ((box=$minbox; box<=$maxbox; box++)); do
23
24     new_box_dir=plots_all_snaps_box${box}
25     if [ ! -e ${new_box_dir} ]; then
26         mkdir ${new_box_dir}
27     fi
28
29     for ((snap=$minsnap; snap<=$maxsnap; snap++)); do
30
31         if [ $snap -lt 10 ]; then
32             j=0${snap}
33         elif [ $snap -lt 100 ]; then
34             j=0${snap}
35         fi
36
37         new_plot_dir=snap${j}_plots
38
39         if [ ! -e ${new_box_dir}/${new_plot_dir} ]; then
40             mkdir ${new_box_dir}/${new_plot_dir}
41         fi
42
43         echo -n "Starting box${box} snap${j}... "
44         ./hist.py ~/projects/simulations/rockstar/box${box}/crossmatch/snap${j}/halos.dat > plots/out.log 2>&1
45         mv plots/* ${new_box_dir}/${new_plot_dir}/.
46         echo "Finished snap${j}"
47
48     done
49 done
50
51 # - end of script

```

## I.4 Statistics Collection Script (Bash)

```

1#!/usr/bin/env bash
2
3 if [ "$#" -ne 1 ]; then
4     echo "Please provide a directory as an argument."
5     exit -1
6 fi
7
8 parent_dir=$1
9
10 for stats_path in $parent_dir/snap061_plots/{stats_*,perc_diff_*}; do
11     stats_file=$(basename "$stats_path")
12     out_file=${stats_file/_allsnaps_}()
13     echo "Merging stats for $stats_file..."
14
15     for snap_dir in $parent_dir/snap*_plots; do
16         if [ -e $snap_dir/$stats_file ]; then
17             snap_num=$(basename "$snap_dir")
18             echo -n "${snap_num:5:2} "
19             cat $snap_dir/$stats_file | cut -d' ' -f 2-
20         fi
21     done | column -t > $parent_dir/$out_file
22
23     echo "Stats written to $out_file..."
24
25 done

```

## Appendix J

### Redshift Trends Code (Python)

```
1  #!/usr/bin/env python
2
3  import sys
4  import os
5  import numpy as np
6  import matplotlib as mpl
7  mpl.use('Agg')
8  import matplotlib.pyplot as plt
9  from scipy.special import gamma as Gamma
10 from scipy.special import psi as digamma
11 from ipdb import set_trace
12
13
14 def main():
15     #for filenum, file in enumerate(sys.argv[1:]):
16     if (len(sys.argv[1:]) == 4):
17         data1 = read_files(sys.argv[1], header_line = None)
18         data2 = read_files(sys.argv[2], header_line = None)
19         data3 = read_files(sys.argv[3], header_line = None)
20         rsnap_data = read_files(sys.argv[4], header_line = None)
21     else:
22         print 'need 4 files'
23         sys.exit(15)
24
25     if fit_mean_trend:
26         with open(statsfile, 'w') as fd:
27             fd.write("#plot slope_slope_err_intercept_intercept_err\n")
28
29     if skew_err_boxes:
30         skew_err1 = get_skew_err(sys.argv[1])
31         skew_err2 = get_skew_err(sys.argv[2])
32         skew_err3 = get_skew_err(sys.argv[3])
33
34     if minsnap > 0:
35         #for data in data1, data2, data3:
36         #    data = data[data[:,0] >= minsnap]
37         data1 = data1[data1[:,0] >= minsnap]
38         data2 = data2[data2[:,0] >= minsnap]
39         data3 = data3[data3[:,0] >= minsnap]
40         if skew_err_boxes:
41             skew_err1 = skew_err1[-len(data1):]
42             skew_err2 = skew_err2[-len(data2):]
43             skew_err3 = skew_err3[-len(data3):]
44
45     if skew_err_col == -2:
46         data1 = np.column_stack((data1, skew_err1))
47         data2 = np.column_stack((data2, skew_err2))
48         data3 = np.column_stack((data3, skew_err3))
49
50     #if (mean_err_col == -2) or (var_err_col == -2) or (skew_err_col == -2) or (kurt_err_col == -2):
51     #    fake_err = np.zeros(len(data1))
52     #    data1 = np.column_stack((data1, fake_err))
53     #    data2 = np.column_stack((data2, fake_err))
54     #    data3 = np.column_stack((data3, fake_err))
55
56     z = 1.0 / rsnap_data[:,1] - 1.0
57     if (len(data1) == len(data2)) and (len(data1) == len(data3)):
58         z = z[-len(data1):]
59     else:
60         sys.exit(16)
61
62     data1 = np.column_stack((data1, z))
63     data2 = np.column_stack((data2, z))
64     data3 = np.column_stack((data3, z))
65
66     #data1[:, -1] = data1[:, -1] - 0.12
67     #data2[:, -1] = data2[:, -1] + 0.12
68
69     for data in [data1, data2, data3]:
70         if expand_error:
71             mask = (np.abs(data[:, data_mean_col] - data[:, mean_col]) > data[:, mean_err_col])
72             data[mask, mean_err_col] = np.abs(data[mask, data_mean_col] - data[mask, mean_col])
73
74         if transform_variance:
75             data[:, var_col] = data[:, var_col]**2 * Gamma(3.0 / data[:, beta_col]) / Gamma(1.0 / data[:, beta_col])
76             data[:, var_err_col] = data[:, var_err_col]**2 * Gamma(3.0 / data[:, beta_col]) / Gamma(1.0 / data[:, beta_col])
77
78             if transform_kurtosis:
79                 #data[:, kurt_col] = (Gamma(5.0 / data[:, kurt_col]) * Gamma(1.0 / data[:, kurt_col])) / Gamma(3.0 /
80                 data[:, kurt_col]) - 3.0
81                 beta = data[:, beta_col]
82                 beta_err = data[:, beta_err_col]
83                 kurtosis = (Gamma(5.0 / beta) * Gamma(1.0 / beta) / Gamma(3.0 / beta)) - 3.0
84
85
```

```

81         kurtosis_err = beta_err * (1.0 / beta**2) * (kurtosis + 3) * (6.0 * digamma(3.0/beta) - 5.0 * digamma
82             (5.0/beta) - digamma(1.0/beta))
83
84         data[:,kurt_col] = kurtosis[:]
85         data[:,kurt_err_col] = kurtosis_err[:]
86
87         data[:,var_col] = np.sqrt(data[:,var_col])           # var to stdev
88         data[:,var_err_col] = np.sqrt(data[:,var_err_col])    # var to stdev
89
90     if save_transformed_data:
91         for data, path in zip([data1, data2, data3], sys.argv[1:4]):
92             fname = transform_file_base + os.path.basename(path)
93             with open(fname, 'w') as fd:
94                 fd.write(transformed_data_header)
95                 np.savetxt(fd, np.column_stack((z, data)), fmt='%g')
96
97
98
99     ######
100    # make mean and stdv plots
101    ######
102
103   for (data, ylabel, color, label, name) in zip([data1, data2, data3], ylabels1, colors, labels1, names):
104       print "Making %s plot..." % (name)
105       fig = plt.figure(figsize=(9.0, 6.0))
106       ax = fig.add_subplot(111)
107
108       ax = make_plot(ax, data[:,z_col], data[:,mean_col], err = data[:,mean_err_col], color = 'blue', marker='o',
109                      , label=label)
110       ax = make_plot(ax, data[:,z_col], data[:,mean_col] + data[:,var_col], color = 'black', linestyle='--')
111       ax = make_plot(ax, data[:,z_col], data[:,mean_col] - data[:,var_col], color = 'black', linestyle='--')
112
113       if add_rms_line:
114           ax = make_plot(ax, data[:,z_col], data[:,data_rms_col], color = 'green', linestyle=':')
115
116       if fit_mean_trend:
117           ax, slope, slope_err, intercept, intercept_err = add_fit(ax, data[:,z_col], data[:,mean_col], err =
118 data[:,mean_err_col], color='red')
119           save_fits(statsfile, name, slope, np.sqrt(slope_err), intercept, np.sqrt(intercept_err))
120
121       #ax.legend(loc='lower right')
122       ax.set_xlim(z[0] + 1.0, z[-1] - 1.0)
123       #ax.invert_xaxis()
124
125       ax.set_xlabel(xlabel, fontsize='x-large')
126       ax.set_ylabel(ylabel, fontsize='x-large')
127
128       fig.tight_layout()
129       fig.savefig(plot_base + 'mean_stddev_' + name + plot_ext, bbox_inches='tight')
130
131     ######
132    # make skew and kurtosis plots
133    ######
134
135
136
137   for (data, ylabel_kurt, ylabel_skew, color, name, ylim_low1, ylim_high1, ylim_low2, ylim_high2) in zip([data1
138       , data2, data3], ylabels2_kurt, ylabels2_skew, colors, names, [-10.0, -10.0, -1.0], [20.0, 20.0, 1.5],
139       [-0.2, -1.5, -0.4], [0.5, 3.5, 0.1]):
140       print "Making %s plot..." % (name)
141       fig = plt.figure(figsize=(9.0, 6.0))
142       ax = fig.add_subplot(111)
143
144       #ax = make_plot(ax, data[:,z_col] - offset, data[:,kurt_col], err = data[:,kurt_err_col], color = 'red',
145       marker='o', linestyle='-', label='Kurtosis')
146       #ax = make_plot(ax, data[:,z_col] + offset, data[:,skew_col], err = data[:,skew_err_col], color = 'blue',
147       marker='o', linestyle='--', label='Skew')
148       ax = make_plot(ax, data[:,z_col] - offset, data[:,kurt_col], err = data[:,kurt_err_col], color = 'red',
149       marker='o', linestyle=':', label='Kurtosis')
150       legend_lines1, legend_labels1 = ax.get_legend_handles_labels()
151
152       ax.set_xlabel(xlabel, fontsize='x-large')
153       ax.set_ylabel(ylabel_kurt, fontsize='x-large')
154       ax.set_ylim(ylim_low1, ylim_high1)
155
156       if separate_skew_axes:
157           ax = ax.twinx()
158           ax = make_plot(ax, data[:,z_col] + offset, data[:,skew_col], err = data[:,skew_err_col], color = 'blue',
159           marker='o', linestyle=':', label='Skew')
160           legend_lines2, legend_labels2 = ax.get_legend_handles_labels()
161
162           ax.set_ylabel(ylabel_skew, fontsize='x-large')
163           ax.legend(legend_lines1 + legend_lines2, legend_labels1 + legend_labels2, loc='lower right')
164           ax.set_xlim(z[0] + 1.0, z[-1] - 1.0)
165           ax.set_ylim(ylim_low2, ylim_high2)
166           #ax.invert_xaxis()
167
168       fig.tight_layout()
169       fig.savefig(plot_base + 'skew_kurtosis_' + name + plot_ext, bbox_inches='tight')

```

```

164     #-----#
165     print 'Finished all plots.'
166
167 def make_plot(ax, x, y, err=None, color='black', marker='None', linestyle='None', label=None):
168     if err == None:
169         if label == None:
170             ax.plot(x, y, color=color, marker=marker, linestyle=linestyle)
171         else:
172             ax.plot(x, y, color=color, marker=marker, linestyle=linestyle, label=label)
173     else:
174         if label == None:
175             ax.errorbar(x, y, yerr=err, color=color, marker=marker, linestyle=linestyle)
176         else:
177             ax.errorbar(x, y, yerr=err, color=color, marker=marker, linestyle=linestyle, label=label)
178     return ax
179
180
181
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 %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     if (z[:,0] != z[:,1]).all() or (z[:,0] != z[:,2]).all():
257         print 'Need matching snapshots for skew error from individual boxes.'
258         print z
259         sys.exit(-1)
260
261     skew_err = np.std(skew, axis=1) / np.sqrt(3.0)
262     return skew_err
263
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\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 ylabel1 = [r'$\mu_{\Delta c}$', r'$\mu_{\Delta M_{\mathrm{vir}}}$', r'$\mu_{\Delta X_{\mathrm{off}}}$']
289 ylabel2_kurt = ['Kurtosis for $\Delta c$', 'Kurtosis for $M_{\mathrm{vir}}$', 'Kurtosis for $X_{\mathrm{off}}$']
290 ylabel2_skew = ['Skew for $\Delta c$', 'Skew for $M_{\mathrm{vir}}$', 'Skew for $X_{\mathrm{off}}$']
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 +/-errufit_mean +/-errufit_stdev +/-errufit_skew +/-errufit_kurt +/-errufit_rms data_gt_epsilon_chi2_pval skew_erruz\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 (Python)

```
1 #!/usr/bin/env python
2
3 import sys
4 import numpy as np
5 import matplotlib as mpl
6 mpl.use('Agg')
7 import matplotlib.pyplot as plt
8 from matplotlib import cm
9 from scipy import interpolate
10 from scipy.ndimage.filters import gaussian_filter
11 #from ipdb import set_trace
12
13
14
15 def main():
16     # Read in particle files
17     header, halos = read_files(sys.argv[1:], header_line = 3)
18
19     if c_source == 'density_profile':
20         print 'len(halos)=', len(halos)
21         halos = halos[np.isfinite(halos[:,c_2lpt_col])]
22         halos = halos[np.isfinite(halos[:,c_zs_col])]
23         print 'len(halos)=', len(halos)
24
25     print 'Filtering data...'
26     for col, val in zip(lt_cols, lt_vals):
27         halos = halos[halos[:, col] <= val]
28     for col, val in zip(gt_cols, gt_vals):
29         halos = halos[halos[:, col] >= val]
30     for col, val in zip(eq_cols, eq_vals):
31         halos = halos[halos[:, col] == val]
32     for col, val in zip(ne_cols, ne_vals):
33         halos = halos[halos[:, col] != val]
34
35     m_avg = (halos[:,47] + halos[:,48])/2.0
36     halos = np.column_stack((halos, m_avg))
37     header = np.append(header, 'M_avg')
38
39     if x_min_lim > 0:
40         print 'nhalos=', len(halos)
41         mask = (m_avg >= x_min_lim)
42         halos = halos[mask]
43         print 'nhalos=', len(halos)
44
45     if c_source == 'rockstar':
46         c1 = halos[:, Rv1_col] / halos[:, Rs1_col]
47         c2 = halos[:, Rv2_col] / halos[:, Rs2_col]
48         if use_klypin:
49             mask = (halos[:,4] < 100)
50             c1[mask] = halos[mask, Rv1_col] / halos[mask, 79]
51             mask = (halos[:,5] < 100)
52             c1[mask] = halos[mask, Rv2_col] / halos[mask, 80]
53     if c_source == 'density_profile':
54         c1 = halos[:, c_2lpt_col]
55         c2 = halos[:, c_zs_col]
56
57     dc = 2.0 * (c1 - c2) / (c1 + c2)
58     #dc = c1 - c2
59
60     m1 = halos[:,47]
61     m2 = halos[:,48]
62     dm = 2.0 * (m1 - m2) / (m1 + m2)
63
64     for x_col, xlabel in zip(x_cols, xlabels):
65         make_plot(dm, x_col, halos, header, xlabel, use_log=False)
66     for x_col, xlabel in zip(x_log_cols, xlabels_log):
67         make_plot(dm, x_col, halos, header, xlabel, use_log=True)
68
69     print 'Finished all plots.'
70
71 def read_files(files, header_line = None, comment_char = '#'):
72     header = None
73     data = None
74     if type(files) == str:
75         files = [files]
76
77     if header_line != None:
78         with open(files[0], 'r') as fd:
79             for line in range(header_line):
```

```

81         fd.readline()
82     header = fd.readline()
83     if header[0] != comment_char:
84         print "Header must start with a '%s'" % comment_char
85         sys.exit(4)
86     header = header[1:]
87     header = header.split()
88
89     for file in files:
90         print 'Reading file %s...' % (file)
91         if data == None:
92             data = np.genfromtxt(file, comments=comment_char)
93         else:
94             data = np.append(data, np.genfromtxt(file, comments=comment_char), axis=0)
95
96     print 'Finished reading files.'
97     if header_line == None:
98         return data
99     else:
100        return header, data
101
102
103 def make_plot(y, x_col, halos, header, xlabel, use_log):
104     x = halos[:, x_col]
105     col_header = header[x_col]
106
107     print 'generating plot...'
108     fig = plt.figure(figsize=(9.0,6.0))
109     ax = fig.add_subplot(1,1,1)
110     ax = draw_hist2d(ax, x, y)
111     ax = draw_data_fit(ax, x, y, x.min(), x.max(), use_log=use_log)
112     if fit_to_binned_data:
113         ax, mid_bins, mean, stdev = draw_bin_avgs(ax, x, y, use_log=use_log)
114         ax = draw_bin_fit(ax, mid_bins, mean, stdev, x.min(), x.max(), use_log=use_log)
115
116     ax.set_xlim([x.min(), x.max()])
117     #ax.set_yscale("log")
118     ax.set_xlabel(xlabel, fontsize="xx-large")
119     ax.set_ylabel(ylabel, fontsize="xx-large")
120
121     fig.tight_layout()
122     col_header = col_header.replace("/", "over")
123     plot_name = "%s%s%0.3d%s%s" % (plot_base, '(', x_col, ')', col_header, plot_ext)
124     plt.savefig(plot_name, bbox_inches='tight')
125     print 'finished plot' + plot_name
126
127
128 def draw_hist2d(ax, x, y):
129     if use_log:
130         xbins = np.logspace(np.log10(x.min()), np.log10(x.max()), num=nbins+1)
131     else:
132         xbins = np.linspace(x.min(), x.max(), num=nbins+1)
133
134     ybins = np.linspace(y.min(), y.max(), num=nbins+1)
135
136     if use_log:
137         ax.set_xscale("log")
138         im = my_hist2d(ax, x, y, bins=[xbins, ybins])
139     else:
140         im = ax.hist2d(x, y, bins=[xbins, ybins], cmap=cmap)
141
142     if y_lim > 0.0:
143         ax.set_ylim([-y_lim, y_lim])
144
145     line = ax.plot([x.min(), x.max()], [0.0, 0.0], 'b--')
146     return ax
147
148
149 def my_hist2d(ax, x, y, bins=10, range=None, normed=False, weights=None,
150               cmin=None, cmax=None, **kwargs):
151     import matplotlib as mpl
152
153     bin_range = range
154     range = mpl.axes._builtins__["range"]
155     h, xedges, yedges = np.histogram2d(x, y, bins=bins, range=bin_range,
156                                         normed=normed, weights=weights)
157
158     if cmin is not None:
159         h[h < cmin] = None
160     if cmax is not None:
161         h[h > cmax] = None
162
163     if z_log:
164         h[h<1.0] = 0.5
165         h = np.log10(h)
166
167     h = gaussian_filter(h, len(h) / 75.0)
168
169     pc = ax.imshow(h[:, ::-1].T, cmap=cmap, extent=[x.min(), x.max(), y.min(), y.max()],
170                    interpolation='gaussian')
171     ax.set_xlim(xedges[0], xedges[-1])
172     ax.set_ylim(yedges[0], yedges[-1])

```

```

172     return h, xedges, yedges, pc
173
174
175 def draw_bin_avgs(ax, x, y, use_log):
176     if use_log:
177         fit_bins = np.logspace(np.log10(x.min()), np.log10(x.max()), num=nfit_bins+1)
178     else:
179         fit_bins = np.linspace(x.min(), x.max(), num=nfit_bins+1)
180
181     mid_bins = (fit_bins[:-1] + fit_bins[1:]) / 2.0
182
183     mean = np.array([])
184     stdev = np.array([])
185     for xmin, xmax in zip(fit_bins[:-1], fit_bins[1:]):
186         mask = np.logical_and(x > xmin, x <= xmax)
187         if mask.sum() > 0:
188             mean_el = y[mask].mean()
189             #stdev_el = y[mask].std() / np.sqrt(len(y))
190             stdev_el = y[mask].std()
191             #stdev_el = stdev / np.sqrt(len(y[mask]))
192         else:
193             mean_el = 0.0
194             stdev_el = -1.0
195         mean = np.append(mean, mean_el)
196         stdev = np.append(stdev, stdev_el)
197
198     mask = (stdev != -1.0)
199     mean = mean[mask]
200     stdev = stdev[mask]
201     mid_bins = mid_bins[mask]
202
203     ax.errorbar(mid_bins, mean, yerr=stdev, fmt='o')
204
205     return ax, mid_bins, mean, stdev
206
207
208
209 def draw_bin_fit(ax, mid_bins, mean, stdev, x_min, x_max, use_log):
210     #fit data
211     if use_log:
212         # coefs = np.polyfit(np.log10(x), y, 1)
213         # coefs, stats = np.polynomial.polynomial.polyfit(np.log10(mid_bins), mean, 1, w=1.0/stdev, full=True)
214         coefs, res, rank, singvals, rcond = np.polyfit(np.log10(mid_bins), mean, 1, full=True)
215     else:
216         # coefs = np.polyfit(x, y, 1)
217         # coefs, stats = np.polynomial.polynomial.polyfit(mid_bins, mean, 1, w=1.0/stdev, full=True)
218         coefs, stats = np.polynomial.polynomial.polyfit(mid_bins, mean, 1, full=True)
219     print 'coefs=%u', coefs
220
221
222     m = coefs[0]
223     b = coefs[1]
224     if use_log:
225         x = np.logspace(np.log10(x_min), np.log10(x_max), 100)
226         y = m * np.log10(x) + b
227     else:
228         x = np.linspace(x_min, x_max, 100)
229         y = m * x + b
230     #y = x**m + b
231     line = ax.plot(x, y, color='green')
232
233     if print_fit_params:
234         if use_log:
235             textstr = '$y=%u\log(x)+%u$' % (m, b)
236         else:
237             textstr = '$y=%umx+%ub$' % (m, b)
238         props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
239         ax.text(0.75, 0.95, textstr, transform=ax.transAxes, fontsize=14,
240                 verticalalignment='top', bbox=props)
241
242     if save_fit_params:
243         with open("fits_to_bins.dat", "a") as fd:
244             fd.write("%g,%g\n" % (m, b))
245
246     return ax
247
248
249 def draw_data_fit(ax, x, y, x_min, x_max, use_log):
250     if remove_zero_strip:
251         mask = (np.abs(y) >= y_epsilon)
252         x = x[mask]
253         y = y[mask]
254
255     #fit data
256     if use_log:
257         coefs, residual, rank, singular_values, rcond = np.polyfit(np.log10(x), y, 1, full=True)
258         # coefs, stats = np.polynomial.polynomial.polyfit(np.log10(mid_bins), mean, 1, w=1.0/stdev, full=True)
259         # coefs, res, rank, singvals, rcond = np.polyfit(np.log10(mid_bins), mean, 1, full=True)
260     else:
261         coefs, residual, rank, singular_values, rcond = np.polyfit(x, y, 1, full=True)
262         # coefs, stats = np.polynomial.polynomial.polyfit(mid_bins, mean, 1, w=1.0/stdev, full=True)
263         # coefs, stats = np.polynomial.polynomial.polyfit(mid_bins, mean, 1, full=True)

```

```

264     print 'coefs=%', coefs, '+/-', residual
265
266
267     m = coefs[0]
268     b = coefs[1]
269     if use_log:
270         x = np.logspace(np.log10(x_min), np.log10(x_max), 100)
271         y = m * np.log10(x) + b
272     else:
273         x = np.linspace(x_min, x_max, 100)
274         y = m * x + b
275     #y = x**m + b
276     line = ax.plot(x, y, color='red')
277
278     if print_fit_params:
279         if use_log:
280             textstr = '$y=%m\log x+b$\n$m=%g$\n$b=%g$' % (m, b)
281         else:
282             textstr = '$y=%mx+b$\n$m=%g$\n$b=%g$' % (m, b)
283         props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
284         ax.text(0.75, 0.95, textstr, transform=ax.transAxes, fontsize=14,
285                 verticalalignment='top', bbox=props)
286
287     if save_fit_params:
288         with open("fits_to_data.dat", "a") as fd:
289             fd.write("%g %g\n" % (m, b, residual))
290
291     return ax
292
293
294 use_log = True
295 #use_log = False
296 z_log = True
297
298 #fit_bins = True
299 #fit_data = True
300
301 print_fit_params = False
302 save_fit_params = True
303
304 use_klypin = True
305
306 remove_zero_strip = True
307 y_epsilon = 0.01
308
309 y_lim = 0.5
310 x_min_lim = 5.33e5 * 100
311
312 #if use_log:
313 #    x_cols = [4, 5, 6, 9, 10, 23, 24, 31, 32, 47, 48, 51, 52, 57, 58] # log10 columns
314 #else:
315 #    x_cols = [17, 18, 77, 78, 91, 92, 93, 94, 97, 98, 99, 100, 107, 108, 111, 112] # nolog columns
316
317 x_cols      = []
318 x_log_cols = [-1]
319 #x_log_cols = [47, 48, -1]
320
321 xlabelss = []
322 xlabelss_log = [r"\mathrm{M_{avg}}"]
323 #xlabelss_log = [r"\mathrm{M_{2LPT}}", r"\mathrm{M_{ZA}}"]
324 #    r"\mathrm{M_{avg}}"
325 #    r"\mathrm{M_{avg}} (\mathrm{\dot{M}})"
326
327 ylabel = r"\mathrm{(M_{2LPT}-M_{ZA})/M_{avg}}"
328
329 #c_source = 'density_profile'
330 c_source = 'rockstar'
331
332 plot_base = 'plots/diff_M--vs--'
333 plot_ext = '.eps'
334
335 #plot_name = 'test.eps'
336 #plot_name = 'c_v_M200c_2lpt.eps'
337 fit_to_binned_data = False
338
339 Rv1_col = 53
340 Rv2_col = 54
341 Rs1_col = 55
342 Rs2_col = 56
343
344 c_2lpt_col = 17
345 c_zza_col = 18
346
347 nbins = 100
348 nfit_bins = 20
349
350 ## c_2lpt, c_zza, chi2_2lpt, chi2_zza
351 #lt_cols = [17, 18, 37, 38]
352 #lt_vals = [100.0, 100.0, 10.0, 10.0]
353 #
354 ## c_2lpt, c_zza, chi2_2lpt, chi2_zza
355 #gt_cols = [17, 18, 37, 38]

```

```

356 #gt_vals = [1.0, 1.0, 0.0]
357
358 lt_cols = []
359 lt_vals = []
360
361 gt_cols = [4, 5]
362 gt_vals = [100, 100]
363
364 eq_cols = [109, 110]
365 eq_vals = [-1, -1]
366
367 ne_cols = []
368 ne_vals = []
369
370 #colormap = cm.PuBuGn
371 #colormap = cm.cubehelix_r
372 #colormap = cm.ocean_r
373 #colormap = cm.rainbow
374 #colormap = cm.gnuplot2_r
375 #colormap = cm.CMRmap_r
376
377 def add_white(orig_map, num):
378     temp_cmap = cm.get_cmap(orig_map, num)
379     vals = temp_cmap(np.arange(num))
380     nfade = num / 7
381     vals[:nfade,0] = np.linspace(1., vals[nfade-1,0], nfade)
382     vals[:nfade,1] = np.linspace(1., vals[nfade-1,1], nfade)
383     vals[:nfade,2] = np.linspace(1., vals[nfade-1,2], nfade)
384     #vals[:nfade,3] = np.linspace(0., vals[nfade-1,3], nfade)
385     #vals[0] = [1.0, 1.0, 1.0, 1.0]
386     #vals[1] = (vals[1] + [1.0, 1.0, 1.0, 1.0]) / 2.0
387     newcmap = mpl.colors.LinearSegmentedColormap.from_list("custom_1", vals)
388     return newcmap
389
390 colormap = add_white('rainbow', 30)
391
392 plot_dest_type = 'paper'
393 if plot_dest_type == 'paper':
394     mpl.rcParams['font.family'] = 'serif'
395     mpl.rcParams['font.size'] = 16
396     mpl.rcParams['axes.linewidth'] = 3
397     mpl.rcParams['lines.linewidth'] = 4
398     mpl.rcParams['patch.linewidth'] = 4
399     mpl.rcParams['xtick.major.width'] = 3
400     mpl.rcParams['ytick.major.width'] = 3
401     mpl.rcParams['xtick.major.size'] = 8
402     mpl.rcParams['ytick.major.size'] = 8
403     mpl.rcParams['xtick.minor.width'] = 2
404     mpl.rcParams['ytick.minor.width'] = 2
405     mpl.rcParams['xtick.minor.size'] = 4
406     mpl.rcParams['ytick.minor.size'] = 4
407
408
409 if __name__ == '__main__':
410     main()

```

## K.2 Concentration (Python)

```

1 #!/usr/bin/env python
2
3 import sys
4 import numpy as np
5 import matplotlib as mpl
6 mpl.use('Agg')
7 import matplotlib.pyplot as plt
8 from matplotlib import cm
9 from scipy import interpolate
10 from scipy.ndimage.filters import gaussian_filter
11 #from ipdb import set_trace
12
13
14
15 def main():
16     # Read in particle files
17     header, halos = read_files(sys.argv[1:], header_line = 3)
18
19     if c_source == 'density_profile':
20         print 'len(halos)=', len(halos)
21         halos = halos[np.isfinite(halos[:,c_2lpt_col])]
22         halos = halos[np.isfinite(halos[:,c_zs_col])]
23         print 'len(halos)=', len(halos)
24
25     print 'Filtering data...'
26     for col, val in zip(lt_cols, lt_vals):
27         halos = halos[halos[:, col] <= val]
28     for col, val in zip(gt_cols, gt_vals):
29         halos = halos[halos[:, col] >= val]
30     for col, val in zip(eq_cols, eq_vals):
31         halos = halos[halos[:, col] == val]
32     for col, val in zip(ne_cols, ne_vals):
33         halos = halos[halos[:, col] != val]

```

```

34
35     m_avg = (halos[:,47] + halos[:,48])/2.0
36     halos = np.column_stack((halos, m_avg))
37     header = np.append(header, 'M_avg')
38
39     if x_min_lim > 0:
40         print 'nhalos=', len(halos)
41         mask = (m_avg >= x_min_lim)
42         halos = halos[mask]
43         print 'nhalos=', len(halos)
44
45     if c_source == 'rockstar':
46         c1 = halos[:, Rv1_col] / halos[:, Rs1_col]
47         c2 = halos[:, Rv2_col] / halos[:, Rs2_col]
48         if use_klypin:
49             mask = (halos[:,4] < 100)
50             c1[mask] = halos[mask, Rv1_col] / halos[mask, 79]
51             mask = (halos[:,5] < 100)
52             c1[mask] = halos[mask, Rv2_col] / halos[mask, 80]
53     if c_source == 'density_profile':
54         c1 = halos[:, c_2lpt_col]
55         c2 = halos[:, c_zs_col]
56
57     dc = 2.0 * (c1 - c2) / (c1 + c2)
58 #dc = c1 - c2
59
60     m1 = halos[:,47]
61     m2 = halos[:,48]
62     dm = 2.0 * (m1 - m2) / (m1 + m2)
63
64     for x_col, xlabel in zip(x_cols, xlabels):
65         make_plot(dc, x_col, halos, header, xlabel, use_log=False)
66     for x_col, xlabel in zip(x_log_cols, xlabels_log):
67         make_plot(dc, x_col, halos, header, xlabel, use_log=True)
68
69     print 'Finished all plots.'
70
71
72 def read_files(files, header_line = None, comment_char = '#'):
73     header = None
74     data = None
75     if type(files) == str:
76         files = [files]
77
78     if header_line != None:
79         with open(files[0], 'r') as fd:
80             for line in range(header_line):
81                 fd.readline()
82             header = fd.readline()
83             if header[0] != comment_char:
84                 print "Header must start with a %s" % comment_char
85                 sys.exit(4)
86             header = header[1:]
87             header = header.split()
88
89     for file in files:
90         print 'Reading file %s...' % (file)
91         if data == None:
92             data = np.genfromtxt(file, comments=comment_char)
93         else:
94             data = np.append(data, np.genfromtxt(file, comments=comment_char), axis=0)
95
96     print 'Finished reading files.'
97     if header_line == None:
98         return data
99     else:
100        return header, data
101
102
103 def make_plot(y, x_col, halos, header, xlabel, use_log):
104     x = halos[:, x_col]
105     col_header = header[x_col]
106
107     print 'generating plot...'
108     fig = plt.figure(figsize=(9.0,6.0))
109     ax = fig.add_subplot(1,1,1)
110     ax = draw_hist2d(ax, x, y)
111     ax = draw_data_fit(ax, x, y, x.min(), x.max(), use_log=use_log)
112     if fit_to_binned_data:
113         ax, mid_bins, mean, stdev = draw_bin_avgs(ax, x, y, use_log=use_log)
114         ax = draw_bin_fit(ax, mid_bins, mean, stdev, x.min(), x.max(), use_log=use_log)
115
116     ax.set_xlim([x.min(), x.max()])
117     #ax.set_yscale("log")
118     ax.set_xlabel(xlabel, fontsize="xx-large")
119     ax.set_ylabel(ylabel, fontsize="xx-large")
120
121     fig.tight_layout()
122     col_header = col_header.replace("/", "over")
123     plot_name = "%s%0.3d%s%s" % (plot_base, '(', x_col, ')', col_header, plot_ext)
124     plt.savefig(plot_name, bbox_inches='tight')
125     print 'finished plot' + plot_name

```

```

126
127
128 def draw_hist2d(ax, x, y):
129     if use_log:
130         xbins = np.logspace(np.log10(x.min()), np.log10(x.max()), num=nbins+1)
131     else:
132         xbins = np.linspace(x.min(), x.max(), num=nbins+1)
133
134     ybins = np.linspace(y.min(), y.max(), num=nbins+1)
135
136     if use_log:
137         ax.set_xscale("log")
138         im = my_hist2d(ax, x, y, bins=[xbins, ybins])
139     else:
140         im = ax.hist2d(x, y, bins=[xbins, ybins], cmap=cmap)
141
142     if y_lim > 0.0:
143         ax.set_ylim([-y_lim, y_lim])
144
145     line = ax.plot([x.min(), x.max()], [0.0, 0.0], 'b--')
146
147
148 def my_hist2d(ax, x, y, bins=10, range=None, normed=False, weights=None,
149               cmin=None, cmax=None, **kwargs):
150     import matplotlib as mpl
151
152     bin_range = range
153     range = mpl.axes._builtins__["range"]
154     h, xedges, yedges = np.histogram2d(x, y, bins=bins, range=bin_range,
155                                         normed=normed, weights=weights)
156
157     if cmin is not None:
158         h[h < cmin] = None
159     if cmax is not None:
160         h[h > cmax] = None
161
162     if z_log:
163         h[h<1.0] = 0.5
164         h = np.log10(h)
165
166     h = gaussian_filter(h, len(h) / 75.0)
167
168     pc = ax.imshow(h[:,::-1].T, cmap=cmap, extent=[x.min(), x.max(), y.min(), y.max()], interpolation='gaussian')
169     ax.set_xlim(xedges[0], xedges[-1])
170     ax.set_ylim(yedges[0], yedges[-1])
171
172     return h, xedges, yedges, pc
173
174
175 def draw_bin_avgs(ax, x, y, use_log):
176     if use_log:
177         fit_bins = np.logspace(np.log10(x.min()), np.log10(x.max()), num=nfit_bins+1)
178     else:
179         fit_bins = np.linspace(x.min(), x.max(), num=nfit_bins+1)
180
181     mid_bins = (fit_bins[:-1] + fit_bins[1:]) / 2.0
182
183     mean = np.array([])
184     stdev = np.array([])
185     for xmin, xmax in zip(fit_bins[:-1], fit_bins[1:]):
186         mask = np.logical_and(x > xmin, x <= xmax)
187         if mask.sum() > 0:
188             mean_el = y[mask].mean()
189             #stdev_el = y[mask].std() / np.sqrt(len(y))
190             stdev_el = y[mask].std()
191             #stdev_el = stdev / np.sqrt(len(y[mask]))
192         else:
193             mean_el = 0.0
194             stdev_el = -1.0
195         mean = np.append(mean, mean_el)
196         stdev = np.append(stdev, stdev_el)
197
198     mask = (stdev != -1.0)
199     mean = mean[mask]
200     stdev = stdev[mask]
201     mid_bins = mid_bins[mask]
202
203     ax.errorbar(mid_bins, mean, yerr=stdev, fmt='o')
204
205     return ax, mid_bins, mean, stdev
206
207
208
209 def draw_bin_fit(ax, mid_bins, mean, stdev, x_min, x_max, use_log):
210     #fit_data
211     if use_log:
212         # coefs = np.polyfit(np.log10(x), y, 1)
213         # coefs, stats = np.polynomial.polynomial.polyfit(np.log10(mid_bins), mean, 1, w=1.0/stdev, full=True)
214         coefs, res, rank, singvals, rcond = np.polyfit(np.log10(mid_bins), mean, 1, full=True)
215     else:
216         # coefs = np.polyfit(x, y, 1)

```

```

217     # coefs, stats = np.polynomial.polynomial.polyfit(mid_bins, mean, 1, w=1.0/stdev, full=True)
218     coefs, stats = np.polynomial.polynomial.polyfit(mid_bins, mean, 1, full=True)
219     print 'coefs=%u', coefs
220
221     m = coefs[0]
222     b = coefs[1]
223     if use_log:
224         x = np.logspace(np.log10(x_min), np.log10(x_max), 100)
225         y = m * np.log10(x) + b
226     else:
227         x = np.linspace(x_min, x_max, 100)
228         y = m * x + b
229     #y = x**m + b
230     line = ax.plot(x, y, color='green')
231
232     if print_fit_params:
233         if use_log:
234             textstr = '$y=%u m %u \\log x %u + b$\n$ m = %g$\n$ b = %g$\n' % (m, b)
235         else:
236             textstr = '$y=%u m x %u + b$\n$ m = %g$\n$ b = %g$\n' % (m, b)
237         props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
238         ax.text(0.75, 0.95, textstr, transform=ax.transAxes, fontsize=14,
239                 verticalalignment='top', bbox=props)
240
241     if save_fit_params:
242         with open("fits_to_bins.dat", "a") as fd:
243             fd.write("%g,%g\n" % (m, b))
244
245
246     return ax
247
248
249 def draw_data_fit(ax, x, y, x_min, x_max, use_log):
250     if remove_zero_strip:
251         mask = (np.abs(y) >= y_epsilon)
252         x = x[mask]
253         y = y[mask]
254
255     #fit data
256     if use_log:
257         coefs, residual, rank, singular_values, rcond = np.polyfit(np.log10(x), y, 1, full=True)
258         # coefs, stats = np.polynomial.polynomial.polyfit(np.log10(mid_bins), mean, 1, w=1.0/stdev, full=True)
259         # coefs, res, rank, singvals, rcond = np.polyfit(np.log10(mid_bins), mean, 1, full=True)
260     else:
261         coefs, residual, rank, singular_values, rcond = np.polyfit(x, y, 1, full=True)
262         # coefs, stats = np.polynomial.polynomial.polyfit(mid_bins, mean, 1, w=1.0/stdev, full=True)
263         # coefs, stats = np.polynomial.polynomial.polyfit(mid_bins, mean, 1, full=True)
264     print 'coefs=%u', coefs, '+/-', residual
265
266     m = coefs[0]
267     b = coefs[1]
268     if use_log:
269         x = np.logspace(np.log10(x_min), np.log10(x_max), 100)
270         y = m * np.log10(x) + b
271     else:
272         x = np.linspace(x_min, x_max, 100)
273         y = m * x + b
274     #y = x**m + b
275     line = ax.plot(x, y, color='red')
276
277     if print_fit_params:
278         if use_log:
279             textstr = '$y=%u m %u \\log x %u + b$\n$ m = %g$\n$ b = %g$\n' % (m, b)
280         else:
281             textstr = '$y=%u m x %u + b$\n$ m = %g$\n$ b = %g$\n' % (m, b)
282         props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
283         ax.text(0.75, 0.95, textstr, transform=ax.transAxes, fontsize=14,
284                 verticalalignment='top', bbox=props)
285
286     if save_fit_params:
287         with open("fits_to_data.dat", "a") as fd:
288             fd.write("%g,%g,%g\n" % (m, b, residual))
289
290
291     return ax
292
293
294 use_log = True
295 #use_log = False
296 z_log = True
297
298 #fit_bins = True
299 #fit_data = True
300
301 print_fit_params = False
302 save_fit_params = True
303
304 use_klypin = True
305
306 remove_zero_strip = True
307 y_epsilon = 0.01
308
```

```

309 y_lim = 0.5
310 x_min_lim = 5.33e5 * 100
311
312 #if use_log:
313 # x_cols = [4, 5, 6, 9, 10, 23, 24, 31, 32, 47, 48, 51, 52, 57, 58] # log10 columns
314 #else:
315 # x_cols = [17, 18, 77, 78, 91, 92, 93, 94, 97, 98, 99, 100, 107, 108, 111, 112] # nolog columns
316
317 x_cols = []
318 x_log_cols = [-1]
319 #x_log_cols = [47, 48, -1]
320
321 xlabel = []
322 xlabel_log = [r"\mathbf{M_{avg}}"]
323 xlabel_log = [r"\mathbf{M_{2LPT} (M_{\odot})}", r"\mathbf{M_{ZA} (M_{\odot})}", r"\mathbf{M_{avg} (M_{\odot})}"]
324 #
325 ylabel = r"\mathbf{(c_{2LPT}-c_{ZA}) / c_{avg}}"
326
327 #c_source = 'density_profile'
328 c_source = 'rockstar'
329
330 plot_base = 'plots/diff_c--vs--'
331 plot_ext = '.eps'
332
333 #plot_name = 'test.eps'
334 #plot_name = 'c_v_M200c_2lpt.eps'
335 fit_to_binned_data = False
336
337 Rv1_col = 53
338 Rv2_col = 54
339 Rs1_col = 55
340 Rs2_col = 56
341
342 c_2lpt_col = 17
343 c_zza_col = 18
344
345 nbins = 100
346 nfit_bins = 20
347
348 ## c_2lpt, c_zza, chi2_2lpt, chi2_zza
349 #lt_cols = [17, 18, 37, 38]
350 #lt_vals = [100.0, 100.0, 10.0, 10.0]
351 #
352 ## c_2lpt, c_zza, chi2_2lpt, chi2_zza
353 #gt_cols = [17, 18, 37, 38]
354 #gt_vals = [1.0, 1.0, 0.0, 0.0]
355 #
356 lt_cols = []
357 lt_vals = []
358
359 gt_cols = [4, 5]
360 gt_vals = [100, 100]
361
362 eq_cols = [109, 110]
363 eq_vals = [-1, -1]
364
365 ne_cols = []
366 ne_vals = []
367
368 colormap = cm.PuBuGn
369 #colormap = cm.cubehelix_r
370 #colormap = cm.ocean_r
371 #colormap = cm.rainbow
372
373 def add_white(orig_map, num):
374     temp_cmap = cm.get_cmap(orig_map, num)
375     vals = temp_cmap(np.arange(num))
376     nfade = num / 7
377     vals[:nfade,0] = np.linspace(1., vals[nfade-1,0], nfade)
378     vals[:nfade,1] = np.linspace(1., vals[nfade-1,1], nfade)
379     vals[:nfade,2] = np.linspace(1., vals[nfade-1,2], nfade)
380     #vals[:nfade,3] = np.linspace(0., vals[nfade-1,3], nfade)
381     #vals[0] = [1.0, 1.0, 1.0, 1.0]
382     #vals[1] = (vals[1] + [1.0, 1.0, 1.0, 1.0]) / 2.0
383     newcmap = mpl.colors.LinearSegmentedColormap.from_list("custom_1", vals)
384     return newcmap
385
386 colormap = add_white('rainbow', 30)
387
388 plot_dest_type = 'paper'
389 if plot_dest_type == 'paper':
390     mpl.rcParams['font.family'] = 'serif'
391     mpl.rcParams['font.size'] = 16
392     mpl.rcParams['axes.linewidth'] = 3
393     mpl.rcParams['lines.linewidth'] = 4
394     mpl.rcParams['patch.linewidth'] = 4
395     mpl.rcParams['xtick.major.width'] = 3
396     mpl.rcParams['ytick.major.width'] = 3
397     mpl.rcParams['xtick.major.size'] = 8
398     mpl.rcParams['ytick.major.size'] = 8

```

```

401     mpl.rcParams['xtick.minor.width'] = 2
402     mpl.rcParams['ytick.minor.width'] = 2
403     mpl.rcParams['xtick.minor.size'] = 4
404     mpl.rcParams['ytick.minor.size'] = 4
405
406
407 if __name__ == '__main__':
408     main()

```

### K.3 PBS Submission Script (Bash)

```

1 #!/usr/bin/env bash
2 #PBS -M djsissom@gmail.com
3 #PBS -m bae
4 #PBS -l nodes=1:ppn=1
5 #PBS -l pmem=4000mb
6 #PBS -l mem=4000mb
7 #PBS -l walltime=1:00:00
8 #PBS -o out.log
9 #PBS -j oe
10
11 minsnap=0
12 maxsnap=61
13
14 # Change to working directory
15 echo $PBS_NODEFILE
16 cd ${PBS_O_WORKDIR}
17
18 rm -v fits_to_bins.dat
19 rm -v fits_to_data.dat
20
21 for ((snap=$minsnap; snap<=$maxsnap; snap++)); do
22
23     if [ $snap -lt 10 ]; then
24         j=00$snap
25     elif [ $snap -lt 100 ]; then
26         j=0$snap
27     fi
28
29     new_plot_dir=snap${j}_plots
30
31     if [ ! -e plots_all_snaps/${new_plot_dir} ]; then
32         mkdir plots_all_snaps/${new_plot_dir}
33     fi
34
35     {
36         echo "Starting box${i} snap${j}..."
37         ./residual_hist.py ~/projects/simulations/rockstar/box{1,2,3}/crossmatch/snap{j}/halos.dat > plots/out.
38         mv plots/* plots_all_snaps/${new_plot_dir}/.
39         echo "Finished snap{j}"
40     }
41     #} &
42
43 done
44
45 wait
46
47 # - end of script

```

## Appendix L

### Alternate Differential Distribution Redshift Trends Code (Python)

```
1 #!/usr/bin/env python
2
3 import sys
4 import numpy as np
5 import matplotlib as mpl
6 mpl.use('Agg')
7 import matplotlib.pyplot as plt
8 from scipy.special import gamma as Gamma
9 from scipy.special import psi as digamma
10 from ipdb import set_trace
11
12
13 def main():
14     if (len(sys.argv[1:]) == 4):
15         data1 = read_files(sys.argv[1], header_line = None)
16         data2 = read_files(sys.argv[2], header_line = None)
17         data3 = read_files(sys.argv[3], header_line = None)
18         rsnap_data = read_files(sys.argv[4], header_line = None)
19     else:
20         print 'need 4 files'
21         sys.exit(15)
22
23     if fit_mean_trend:
24         with open(statsfile, 'w') as fd:
25             fd.write("#plot_uslope_uslope_err_uintercept_uintercept_err\n")
26
27     if minsnap > 0:
28         data1 = data1[data1[:,0] >= minsnap]
29         data2 = data2[data2[:,0] >= minsnap]
30         data3 = data3[data3[:,0] >= minsnap]
31
32     z = 1.0 / rsnap_data[:,1] - 1.0
33     if (len(data1) == len(data2)) and (len(data1) == len(data3)):
34         z = z[-len(data1):]
35     else:
36         sys.exit(16)
37
38     data1 = np.column_stack((data1, z))
39     data2 = np.column_stack((data2, z))
40     data3 = np.column_stack((data3, z))
41
42
43 #-----#
44 # make mean and stdv plots
45 #-----#
46
47 for (data, ylabel, label, name) in zip([data1, data2, data3], ylabels1, labels1, names):
48     print "Making %s plot..." % (name + '_xvals')
49     fig = plt.figure(figsize=(9.0, 6.0))
50     ax = fig.add_subplot(111)
51
52     ax = make_plot(ax, data[:,z_col], data[:,peak_col], err = None, color = 'black', marker='o', linestyle='-', label=None)
53
54     for (x_val_col, color) in zip(x_val_cols, colors1):
55         ax = make_plot(ax, data[:,z_col], data[:,x_val_col], err = None, color = color, marker='o', linestyle='--', label=None)
56
57     #if add_rms_line:
58     #    ax = make_plot(ax, data[:,z_col], data[:,data_rms_col], color = 'green', linestyle=':')
59
60     #if fit_mean_trend:
61     #    ax, slope, slope_err, intercept, intercept_err = add_fit(ax, data[:,z_col], data[:,mean_col], err = data[:,mean_err_col], color='red')
62     #    save_fits(statsfile, name, slope, np.sqrt(slope_err), intercept, np.sqrt(intercept_err))
63
64     #ax.legend(loc='lower right')
65     ax.set_xlim(z[0] + 1.0, z[-1] - 1.0)
66     #ax.invert_xaxis()
67
68     ax.set_xlabel(xlabel, fontsize='x-large')
69     ax.set_ylabel(ylabel, fontsize='x-large')
70
71     fig.tight_layout()
72     fig.savefig(plot_base + name + '_xvals' + plot_ext, bbox_inches='tight')
73
74 #-----#
75
76 for (data, ylabel, label, name) in zip([data1, data2, data3], ylabels2, labels1, names):
77     print "Making %s plot..." % (name + '_sumfrac')
78     fig = plt.figure(figsize=(9.0, 6.0))
79     ax = fig.add_subplot(111)
```

```

80     for (sum_frac_col, color) in zip(sum_frac_cols, colors2):
81         ax = make_plot(ax, data[:,z_col], data[:,sum_frac_col], err = None, color = color, marker='o',
82         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',
85             linestyle='--', label=None)
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 print 'Finished all plots.'
95
96
97 def make_plot(ax, x, y, err=None, color='black', marker='None', linestyle='None', label=None):
98     if err == None:
99         if label == None:
100             ax.plot(x, y, color=color, marker=marker, linestyle=linestyle)
101         else:
102             ax.plot(x, y, color=color, marker=marker, linestyle=linestyle, label=label)
103     else:
104         if label == None:
105             ax.errorbar(x, y, yerr=err, color=color, marker=marker, linestyle=linestyle)
106         else:
107             ax.errorbar(x, y, yerr=err, color=color, marker=marker, linestyle=linestyle, label=label)
108
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\n" % (name, slope, slope_err, intercept, intercept_err))
166
167
168 plot_dest_type = 'paper'
169 if plot_dest_type == 'paper':

```

```

170     mpl.rcParams['font.family'] = 'serif'
171     mpl.rcParams['font.size'] = 16
172     mpl.rcParams['axes.linewidth'] = 3
173     mpl.rcParams['lines.linewidth'] = 4
174     mpl.rcParams['patch.linewidth'] = 4
175     mpl.rcParams['xtick.major.width'] = 3
176     mpl.rcParams['ytick.major.width'] = 3
177     mpl.rcParams['xtick.major.size'] = 8
178     mpl.rcParams['ytick.major.size'] = 8
179
180 #colors = ['red', 'green', 'blue']
181 colors = ['black', 'black', 'black']
182 labels1 = [r'$\mathfrak{M}_{\mathrm{vir}}$', r'$X_{\mathrm{off}}$']
183 names = ['c_rockstar', 'Mvir', 'Koff']
184 xlabel = 'Redshift'
185 ylabel1 = [r"$\Delta_{\mathrm{c}}(f_{\mathrm{h}}, z)$", r"$\Delta_{\mathrm{M}}(\mathrm{vir})(f_{\mathrm{h}}, z)$", r"$\Delta_{\mathrm{M}}(\mathrm{vir}, \mathrm{peak})$",
186             r"$\Delta_{\mathrm{X}}(\mathrm{off})(f_{\mathrm{h}}, z)$", r"$\Delta_{\mathrm{X}}(\mathrm{off}, \mathrm{peak})$"]
187 ylabel2 = [r"$f_{\mathrm{h}}(\Delta_{\mathrm{c}}, z)$", r"$f_{\mathrm{h}}(\Delta_{\mathrm{M}}(\mathrm{vir}), z)$", r"$f_{\mathrm{h}}(\Delta_{\mathrm{X}}(\mathrm{off}), z)$"]
188 plot_base = 'plots/'
189 plot_ext = '.eps'
190
191 statsfile = 'plots/stats.dat'
192
193 z_col = -1
194 snap_col = 0
195 mean_col = 7
196 mean_err_col = 8
197 var_col = 9
198 var_err_col = 10
199 skew_col = 3
200 skew_err_col = -2
201 #skew_col = 7
202 #skew_err_col = 8
203 #kurt_col = 4
204 kurt_col = 13
205 kurt_err_col = 14
206 beta_col = 13
207 beta_err_col = 14
208
209 data_mean_col = 1
210 data_rms_col = 15
211
212
213
214 peak_col = 1
215 x_val_cols = np.array([4, 6, 8]) + 2
216 sum_frac_cols = np.array([2, 4, 6, 8]) + 2 + 9
217 doublesum_frac_cols = sum_frac_cols + 9
218
219 colors1 = ['red', 'green', 'blue']
220 colors2 = ['blue', 'green', 'red', 'black']
221
222 offset = 0.06
223 #offset = 0.0
224
225 minsnap = 39
226 #minsnap = None
227
228 fit_mean_trend = False
229 add_rms_line = False
230
231
232 if __name__ == '__main__':
233     main()

```

## Appendix M

### Miscellaneous Scripts

#### M.1 Directory Structure Setup (Bash)

```
1 #!/usr/bin/env bash
2
3 minsnap=0
4 maxsnap=61
5
6 minbox=1
7 maxbox=3
8
9 for ((i=$minbox; i<=$maxbox; i++)); do
10   if [ ! -e ../box$i ]; then
11     mkdir -v ../box$i
12   fi
13   if [ ! -e ../box$i/2lpt ]; then
14     mkdir -v ../box$i/2lpt
15   fi
16   if [ ! -e ../box$i/za ]; then
17     mkdir -v ../box$i/za
18   fi
19   if [ ! -e ../box$i/crossmatch ]; then
20     mkdir -v ../box$i/crossmatch
21   fi
22
23 cp -v run_*.pbs ../$box$/
24 cp -v postprocess.sh ../$box$/
25
26 for ((snap=$minsnap; snap<=$maxsnap; snap++)); do
27   if [ $snap -lt 10 ]; then
28     j=0$snap
29   elif [ $snap -lt 100 ]; then
30     j=0$snaps
31   fi
32
33   if [ ! -e ../box$i/2lpt/snap$j ]; then
34     mkdir -v ../box$i/2lpt/snap$j
35   fi
36   if [ ! -e ../box$i/za/snap$j ]; then
37     mkdir -v ../box$i/za/snap$j
38   fi
39
40 cp -v -r proto/* ../$box$/2lpt/snap$j/.
41 cp -v -r proto/* ../$box$/za/snap$j/.
42
43 ln -v -s ~/projects/data/2lpt/box$i/2lpt_512_z300_PM_$j ../$box$/2lpt/snap$j/particles/2lpt_512_z300_PM_$j
44 ln -v -s ~/projects/data/za/box$i/za_512_z300_PM_$j ../$box$/za/snap$j/particles/za_512_z300_PM_$j
45
46 echo /home/sissomdj/projects/simulations/rockstar/box$i/2lpt/snap$j/particles/2lpt_512_z300_PM_$j > ../$box$/
47 echo /home/sissomdj/projects/simulations/rockstar/box$i/za/snap$j/particles/za_512_z300_PM_$j > ../$box$/za/
48 snap$j/particles/snapnames.lst
49
50 echo "BGC2_SNAPNAMES=\\"/home/sissomdj/projects/simulations/rockstar/box$i/2lpt/snap$j/particles/snapnames.
51 lst\"">> ../$box$/2lpt/snap$j/onenode.cfg
52 echo "BGC2_SNAPNAMES=\\"/home/sissomdj/projects/simulations/rockstar/box$i/za/snap$j/particles/snapnames.lst
53 "\"">> ../$box$/za/snap$j/onenode.cfg
54 done
55
56 done
```

#### M.2 CROSSMATCH Setup (Bash)

```
1 #!/usr/bin/env bash
2
3 minsnap=0
4 maxsnap=61
5
6 minbox=1
7 maxbox=3
8
9 for ((i=$minbox; i<=$maxbox; i++)); do
10   if [ ! -e ../$box$i/crossmatch ]; then
11     mkdir -v ../$box$i/crossmatch
12   fi
13
14 cp -v run_crossmatch.pbs ../$box$/
15
16 for ((snap=$minsnap; snap<=$maxsnap; snap++)); do
```

```

17     if [ $snap -lt 10 ]; then
18         j=0$snap
19     elif [ $snap -lt 100 ]; then
20         j=0$snaps
21     fi
22
23     if [ ! -e ../box$i/crossmatch/snap$j ]; then
24         mkdir -v ../box$i/crossmatch/snap$j
25     fi
26
27     cp -v -r crossmatch_proto/* ../box$i/crossmatch/snap$j/.
28
29     echo "OUTPUT_DIR\uuuuuuuu/home/sissomdj/projects/simulations/rockstar/box$i/crossmatch/snap$j" >> ../box$i/
30     crossmatch/snap$j/rockstar_2lpt.param
31     echo "FIRST_GROUPDIR\uuuu/home/sissomdj/projects/simulations/rockstar/box$i/2lpt/snap$j/halos" >> ../box$i/
32     crossmatch/snap$j/rockstar_2lpt.param
33     echo "SECOND_GROUPDIR\uuuu/home/sissomdj/projects/simulations/rockstar/box$i/za/snap$j/halos" >> ../box$i/
34     crossmatch/snap$j/rockstar_2lpt.param
35     echo "OUTPUT_DIR\uuuuuuuu/home/sissomdj/projects/simulations/rockstar/box$i/crossmatch/snap$j" >> ../box$i/
36     crossmatch/snap$j/rockstar_za.param
37     echo "FIRST_GROUPDIR\uuuu/home/sissomdj/projects/simulations/rockstar/box$i/za/snap$j/halos" >> ../box$i/
38     crossmatch/snap$j/rockstar_za.param
39     echo "SECOND_GROUPDIR\uuuu/home/sissomdj/projects/simulations/rockstar/box$i/2lpt/snap$j/halos" >> ../box$i/
39     crossmatch/snap$j/rockstar_za.param
39 done
39 done

```

### M.3 Individual Snapshot ROCKSTAR Run Script (Bash)

```

1 #!/bin/bash
2
3 echo "Cleaning\u00d7old\u00d7files..."
4 if [ -e out.log ]; then
5     mv -v out.log out.log.bak
6 fi
7 if [ -e server.out ]; then
8     mv -v server.out server.out.bak
9 fi
10 if [ -e clients.out ]; then
11     mv -v clients.out clients.out.bak
12 fi
13 if [ -e auto-rockstar.cfg ]; then
14     rm -v auto-rockstar.cfg
15 fi
16 if [ $(ls halos/* 2> /dev/null | wc -l) != "0" ]; then
17     rm -rv halos/*
18 fi
19
20 echo "Submitting\u00d7run\u00d7script..."
21 echo "qsub run_rockstar.pbs"
22 qsub run_rockstar.pbs

```

### M.4 All Snapshots ROCKSTAR 2LPT PBS Submission Script (Bash)

```

1 #!/usr/bin/env bash
2
3 #PBS -M djsissom@gmail.com
4 #PBS -m bae
5 #PBS -l nodes=1:ppn=10
6 #PBS -l pmem=3000mb
7 #PBS -l mem=30000mb
8 #PBS -l walltime=6:00:00
9 #PBS -o out_2lpt.log
10 #PBS -j oe
11
12 echo $PBS_NODEFILE
13 cd $PBS_O_WORKDIR
14
15 for snapdir in 2lpt/*; do
16     # Change to working directory
17     echo Working on $snapdir...
18     cd $PBS_O_WORKDIR/$snapdir
19
20     # Start the server
21     rockstar -c onenode.cfg &> server.out &
22
23     # Wait for auto-rockstar.cfg to be created
24     perl -e 'sleep 1 while (!(-e "halos/auto-rockstar.cfg"))';
25     mv halos/auto-rockstar.cfg .
26
27     # Execute the reader processes
28     mpieexec -verbose -n 1 rockstar -c auto-rockstar.cfg >> clients.out 2>&1 &
29     sleep 20
30
31     # Execute the analysis processes
32     mpieexec -verbose -n 8 rockstar -c auto-rockstar.cfg >> clients.out 2>&1
33

```

```

34 # - end of script
35 done

1 #!/usr/bin/env bash
2
3 #PBS -M djsissom@gmail.com
4 #PBS -m bae
5 #PBS -l nodes=1:ppn=10
6 #PBS -l pmem=3000mb
7 #PBS -l mem=30000mb
8 #PBS -l walltime=6:00:00
9 #PBS -o out_za.log
10 #PBS -j oe
11
12 echo $PBS_NODEFILE
13 cd $PBS_O_WORKDIR
14
15 for snapdir in za/*; do
16   # Change to working directory
17   echo Working on $snapdir...
18   cd $PBS_O_WORKDIR/$snapdir
19
20 # Start the server
21 rockstar -c onenode.cfg &> server.out &
22
23 # Wait for auto-rockstar.cfg to be created
24 perl -e 'sleep 1 while (!(-e "halos/auto-rockstar.cfg"))'
25 mv halos/auto-rockstar.cfg .
26
27 # Execute the reader processes
28 mpixec -verbose -n 1 rockstar -c auto-rockstar.cfg >> clients.out 2>&1 &
29 sleep 20
30
31 # Execute the analysis processes
32 mpixec -verbose -n 8 rockstar -c auto-rockstar.cfg >> clients.out 2>&1
33
34 # - end of script
35 done

```

## M.6 All Snapshots ROCKSTAR Post-Process Script (Bash)

```

1 #!/usr/bin/env bash
2
3 startdir='pwd'
4
5 for snapdir in {2lpt,za}/*; do
6   echo Working on $snapdir...
7   cd $startdir/$snapdir
8
9   ./postprocess
10
11 done
12
13 # - end of script

```

## M.7 All Snapshots CROSSMATCH PBS Submission Script (Bash)

```

1 #!/usr/bin/env bash
2
3 #PBS -M djsissom@gmail.com
4 #PBS -m bae
5 #PBS -l nodes=62:ppn=1
6 #PBS -l pmem=3000mb
7 #PBS -l mem=186000mb
8 #PBS -l walltime=1:00:00
9 #PBS -o out_crossmatch.log
10 #PBS -j oe
11
12 echo $PBS_NODEFILE
13 cd $PBS_O_WORKDIR
14
15 for snapdir in crossmatch/*; do
16   # Change to working directory
17   echo Working on $snapdir...
18   cd $PBS_O_WORKDIR/$snapdir
19
20 {
21   mpixec -verbose -n 1 crossmatch rockstar_2lpt.param > out_2lpt_first.log 2>&1
22   mpixec -verbose -n 1 crossmatch rockstar_za.param    > out.za_first.log    2>&1
23   echo "Finished:$snapdir"
24 } &
25
26 done
27
28 wait
29 # - end of script

```

## M.8 All Snapshots Density Profile PBS Submission Script (Bash)

```
1 #!/usr/bin/env bash
2
3 #PBS -M djsissom@gmail.com
4 #PBS -m bae
5 #PBS -l nodes=124:ppn=1
6 #PBS -l pmem=4000mb
7 #PBS -l mem=496000mb
8 #PBS -l walltime=1:00:00
9 #PBS -o out_density_profile.log
10 #PBS -j oe
11
12 echo $PBS_NODEFILE
13 cd $PBS_O_WORKDIR
14
15 for snapdir in {2lpt,za}/snap*/halos; do
16     # Change to working directory
17     echo Working on $snapdir...
18     cd $PBS_O_WORKDIR/$snapdir
19
20     {
21         mpiexec -verbose -n 1 density_profile halos_0.*.bgc2 > density_profile_out.log 2>&1
22         echo "Finished $snapdir"
23     } &
24
25 done
26
27 wait
28 # - end of script
```