Distinguishing between gravity theories with galaxy peculiar velocity statistics

Robert Olav Fauli

Thesis submitted for the degree of
Master of Science in Astronomy

Institute of Theoretical Astrophysics
University of Oslo

1st of June 2015

Abstract

The ΛCDM model used in modern cosmology has major theoretical problems. To attempt to solve these many modified gravity theories have been proposed. We calculate velocity statistics (streaming pair-wise velocities and the velocity correlation function) for different models (Hu-Sawicki f(R) and symmetron) using data from n-body simulations. We find that the streaming velocities show the clearest deviations from GR. The F5 model deviates quite strongly, while for symmetron models the time of fifth force turn on is more important than the force range. We have not accounted for all the sources of variance and covariance, but suggest running many simulations with different realisations of the initial matter distribution to solve this. For future work we suggest calculating the statistics while binning the halos by their environments, and comparing our simulations to observations.

Acknowledgments

I would like to thank my supervisors, David F. Mota, Claudio Llinares and Philip Bull, for always answering my questions to the best of their ability whenever I came to them for help, and for their positive encouragement during dark times. Thanks to Phil (Philip Bull) for giving me direction, and for rapidly giving detailed constructive feedback on the thesis.

Thanks to my family for providing financial support and helping me take my drivers licence during my time studying.

Contents

Chapter 1

Introduction

Our understanding of the cosmos has changed enormously over the last century. In the beginning of the century we did not know about other galaxies and thought the Universe was static. A static universe had the benefit of not needing to explain a beginning.

When Einstein developed General Relativity (1916) it became apparent that in order to keep the universe static he had to insert a constant into his equations, otherwise the Universe would collapse in on itself or expand rapidly. This was called the cosmological constant.

In 1929 Edwin Hubble made a suprising discovery. It was known at the time that the light from a source approaching an observer would be measured at a shorter wavelength than what was emitted at the source, and conversely a source moving away from the observer would emit radiation that the observer would measure as a longer wavelength than what was emitted from the source. Since the part of the visible spectrum on the shorter wavelength end is blue and the part on the longer wavelength end is red, the light from a source moving towards us will blueshifted, while if the source is moving away from us will be redshifted. The fractional deviation of the observed wavelength from the one emitted at the source is called the *redshift*. Hubble measured the redshift of a type of star called Cepheid variables in nearby galaxies. This type of star changes in brightness periodically over time and the period can be related to the radiative intensity of the star. Knowing how much the star radiates we also know how much radiation we should measure as a function of distance to the star.

Measuring both the redshift and the period of the Cepheid variables, one could therefore determine both the velocity at which the stars were moving closer or farther away from us and how far away they were. The data showed that they were moving away from us and that there was a linear relationship between the velocity and the distance from us. This was formulated as Hubble's law.

In 1998 the "High-z Supernova Search Team" used supernova data from the Hubble Space Telescope to show that the Universe had been expanding more slowly in the past [1], while the "Supernova Cosmology Project" did the same using ground based telescopes the same year [2].

To cause this accelerating expansion, missing energy was needed, and a lot of it (69%, according to the 2013 Planck data [3]). Having no idea what it is, it has been appropriately called "dark energy".

This gave the cosmological constant a new role; instead of keeping the Universe static it would now be responsible for the acceleration of the universe.

The other energy component with the prefix "dark" is *dark matter*. We do not know what dark matter is, but we can see its effects; stars orbit around the centre of their galaxies so fast that if only the matter we can see was there they should escape the galaxy according to General Relativity (GR). Dark matter is the invisible source of this extra gravitational force. Although

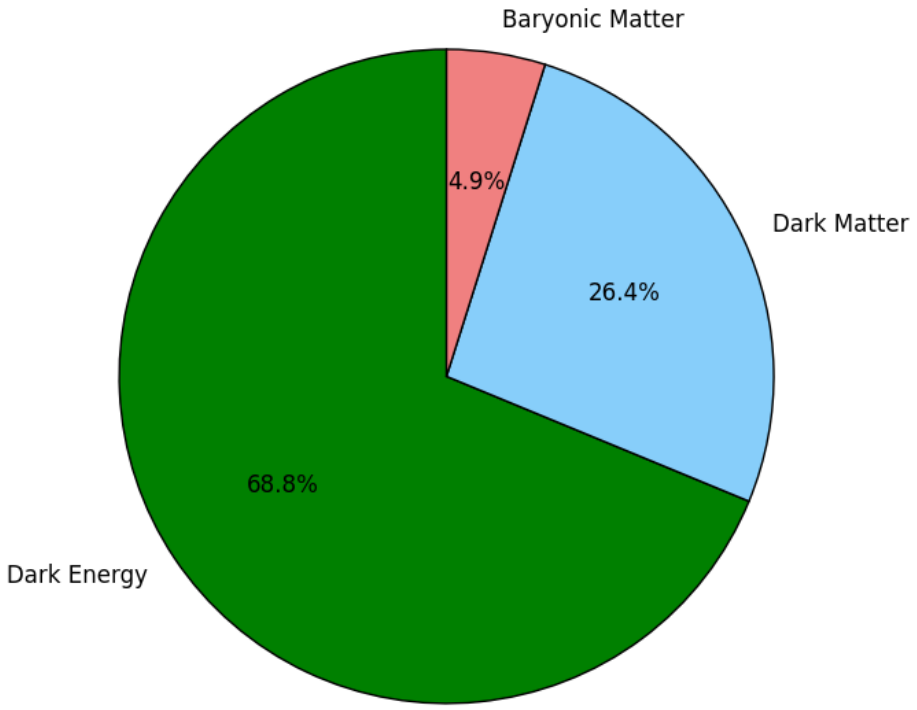


Figure 1.1: Distribution of the energy content in the universe according to the 2013 Planck data [3].

we do not know what dark matter is, some of its properties have been deduced and the more likely type is called Cold Dark Matter (CDM). Cold dark matter consist of non-relativistic particles that do not interact through any other means than gravity.

Today we have a standard model of cosmology called ΛCDM, where Λ is the cosmological constant and CDM stands for Cold Dark Matter. This model emerges from General Relativity (GR), when one assumes that the Universe looks the same in all directions, — the Universe is isotropic, — and that on large scales the Universe looks the same everywhere, — it is homogeneous. ΛCDM has just 6 parameters: the density of baryons, the density of cold dark matter, when the first stars and galaxies were formed reionizing the Universe, the age of the Universe, the amplitude and scale dependence of the initial fluctuations in the matter distribution. The initial matter fluctuations are the end product of what is called inflation, a theory for how the Universe evolved very early in its history. ΛCDM fits the data both from the Cosmic Microwave Background (CMB) [4, 5] and supernovae redshifts [2, 1] very well.

In ΛCDM the cosmological constant Λ, dark energy, is responsible for the Universe's accelerated expansion. Dark energy has a certain set of properties in ΛCDM: its energy density is constant as the Universe expands. From particle physics we know of one thing with this property, — vacuum energy. The problem is that the amount of vacuum energy particle physicists calculate is much higher than what is needed in ΛCDM. Almost all of the vacuum energy must cancel by some mechanism, this is a 55-order fine-tuning [6, 7]. It would seem less mystical

if all of the vacuum energy were cancelled and there was another source for the cosmological constant. So even though ΛCDM fits the data so well, there is still a big problem, although purely theoretical. Thus alternative theories are actively developed, the motivation being to solve the fine-tuning problem of the cosmological constant; it needs to be explained.

Alternatives that have been suggested for the cosmological constant include: adding so far unknown energy content, like different types of exotic fluids, or altering the theory of the space-time geometry of the Universe. The latter go under the category of "Modified Gravity" and include things such as adding extra dimensions and scalar fields [8]. Though many of the alternatives also fit data well, they do not match the simplicity of ΛCDM and in order to justify a more complex theory it should have extra explanatory power, which means that they need to be different enough from ΛCDM that we can detect it. So far we have not detected any such deviations from ΛCDM [3].

The modifications to GR have to be made in a way that makes it plausible that we have not yet detected them by direct measurements. One of the ways of doing this is to add a scalar field (or a few scalar fields) that permeates the universe. These scalar fields cause what is commonly called a "fifth force", (though it is not really another force,) that hides by means of what we call "screening mechanisms" making it really difficult to detect in dense areas like for example the solar system. Two examples of screening mechanisms are *Chameleon* and *Symmetron* screening, which we will come back to later in the thesis.

They also need to be consistent with the way we know the Universe has evolved. A way of testing this is to simulate the universe from an early time until today, then comparing with what we see through observations. If something is different is it because of the modified gravity, or the limitations of Nbody-simulations (resolution, etc, see the Section 2)? Or can we find something that is different in our simulations with modified gravity compared to normal gravity, and if so can we look for it in our own universe?

Although no deviations from ΛCDM of significance have been detected yet, we are constantly accumulating new data restricting how much of a deviation from ΛCDM is theoretically feasible to still be consistent with data. This is done through solar system tests (measuring the acceleration on objects by different means), measuring the properties of the CMB, weak lensing (gravity bends light slightly when it passes by regions with a high matter density, thus distorting the image allowing us to map the dark matter structure), and galaxy/dark matter halo velocity surveys (galaxies will move differently when gravity is different from GR). The main focus for future surveys are based around galaxy surveys, and more accurate measurements of the CMB. Some projects that will accumulate data in the future surveying galaxies and looking at weak lensing are the Euclid mission [9] and the Dark Energy Survey (DES) [10]. And for the CMB we have ACTPol [11] and SPTPol [12]. Galaxy surveys are directly relevant to our the velocity statistics we will look at in our thesis, and so are the velocities of galaxy clusters extracted from CMB surveys using the fact that CMB photons scatter off electrons causing anisotropic radiation, called the kinematic Sunyaev Zel'dovich effect. If we find a signature in the velocity statistics from our simulations we could look for that specifically in the observational data.

1.1 General Relativity

In this section we give a brief overview of the formalism and physical principles of General Relativity. We follow both Sean Carroll's Spacetime and Geometry [13], and Øyvind Grøn's Lecture Notes on General Relativity.

1.1.1 GR building blocks

In cosmology we work on very large scales. While other forces are more important on small scales, on large scales gravity dominates. First we need to get some of the fundamentals out of the way, so in this section we will list the most important expressions in General Relativity with a short explanation of what they mean.

1.1.1.1 The equivalence principle

In GR gravity is not a force; when an object (e.g. a rocket) is in free fall towards the surface of (e.g) a planet there is no force acting on it. To an observer on the surface of the planet it looks like the rocket is accelerating towards the planet with the acceleration g. The principle of equivalence states that there is no way the person inside the rocket can distinguish between being in free fall and being uniformly accelerated by the rocket engines in free space however; the result of any experiments performed inside will be the same. In the Newtonian sense this is the case because the "gravitational charge", m_g ($F_g = -m_g \nabla\phi$) of an object is equal to its inertial mass, m_i ($F = m_i a$), so freely falling objects with the same initial velocity follow the same path in a gravitational field independent of their mass. These paths are called *geodesics*. This is the equivalence principle, and it is a foundational component of GR.

As we will see in Section 1.2.4, Modified Gravity could cause apparent violations of the equivalence principle.

1.1.1.2 Formalism

In General Relativity a lot of formalism has been introduced to simplify expressions when operating with vectors and tensors. One of them is Einstein's summation convention:

$$S = a^\mu b_\mu = a^0 b_0 + a^1 b_1 + a^2 b_2 + a^3 b_3; \tag{1.1}$$

the numbers here are indices. When one upper and one lower index are the same letter we sum over the indices $0, 1, 2, 3$ if it is a greek letter and $1, 2, 3$ in the case of a latin letter.

A vector can then be expressed as:

$$v = v^\mu e_{(\mu)} = v^0 e_{(0)} + v^1 e_{(1)} + v^2 e_{(2)} + v^3 e_{(3)} \tag{1.2}$$

where we have used Einstein's summing convention in the last equality and $e_{(\mu)}$ is the basis spanning the vector space, which the paranthesis indicates.

The basis we will almost always used is $e_{(\mu)} = \partial_\mu = \frac{\partial}{\partial x^\mu}$, we call this the "coordinate basis".

A complementary mathematical construct called the dual-vector $\omega = \omega_\nu \theta^{(\nu)}$ are maps from the vector space to real space $\mathbf{R}$:

$$\omega \cdot v = \omega_\nu \theta^{(\nu)} v^\mu e_{(\mu)} = \omega_\nu v^\mu (\theta^{(\nu)} e_{(\mu)}) = \omega_\mu v^\mu \in \mathbf{R} \tag{1.3}$$

A (k, l) tensor can be transformed into a $(k-1, l+1)$ tensor by contracting it with the metric.

$$g_{\mu\nu} C^\nu{}_{\sigma\lambda} = C_{\mu\sigma\lambda} \tag{1.4}$$

The position of the indices always remains intact.

1.1.1.3 Curvature

Gravity is caused by curvature in spacetime. To describe how spacetime is curved, we use the metric and the Riemann tensor.

When we say "the metric" we can mean two things, either the tensor $g_{\mu\nu}$ or the line-element ds^2. Both are interchangeable in most contexts, — we can get the tensor from the line-element and vice versa. They give us the geometry of the spacetime manifold.

The metric allows us to find the distance between two points on the manifold. The product of two vectors is $u \cdot v = g_{\mu\nu}u^\mu v^\nu$.

In general relativity two different sign conventions are used: $diag(-,+,+,+)$ and $diag(+,-,-,-)$. We will use the former (which is standard in cosmology, however in particle physics the latter is common), meaning that for flat space we have

$$g_{\mu\nu} \equiv \eta_{\mu\nu} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{1.5}$$

The derivative of a vector ∂_μ is not covariant, so to make it covariant one adds a correction term called the connection

$$\nabla_\mu v^\nu = \partial_\mu v^\nu + \Gamma^\nu_{\mu\lambda} V^\lambda \tag{1.6}$$

where $\Gamma^\sigma_{\mu\nu}$ is the connection coefficients.

Christoffel-symbols are the connection coefficients and in coordinate basis are given by:

$$\Gamma^\lambda_{\mu\nu} = \frac{1}{2} g^{\lambda\rho} \left(\partial_\mu g_{\nu\rho} + \partial_\nu g_{\rho\mu} - \partial_\rho g_{\mu\nu} \right). \tag{1.7}$$

The connection coefficients describe how the coordinates have different length and direction from one place to another (like how in polar coordinates $d\theta$ is longer the farther away from origin we get). So if we want to keep a vector/tensor constant while parallel transporting along the radius in polar coordinates and expressed in terms of polar coordinates, the connection would tell us how. The covariant derivative tells us the rate of change of a tensor field relative to if it was parallel transported.

Parallel transporting a vector along a path is to transport it along that path while keeping it constant relative to the path. So the straightest possible line would be one that parallel transports its tangent vector. These straightest possible lines are what we call geodesics, free particles follow geodesics and they are given by the geodesic equation:

$$\frac{\mathrm{d}^2 x^\mu}{\mathrm{d}\tau} + \Gamma^\mu_{\rho\sigma} \frac{\mathrm{d}x^\rho}{\mathrm{d}\tau} = 0 \tag{1.8}$$

When a particle is not free, but a force is working on it, the path is given by:

$$\frac{\mathrm{d}x^\mu}{\mathrm{d}\tau^2} + \Gamma^\mu_{\rho\sigma} \frac{\mathrm{d}x^\rho}{\mathrm{d}\tau} = \frac{q}{m} F^\mu_{\ \nu} \frac{\mathrm{d}x^\nu}{\mathrm{d}\tau} \tag{1.9}$$

τ is here the proper time, the time shown hypothetical clocks following the particles paths.

The Riemann tensor or curvature tensor tells us what happens to a vector that is parallel transported around an infinitesimally small loop consisting of two vectors. The change in the vector after transport can then be expressed as

$$\delta V^\rho = R^\rho_{\ \sigma\mu\nu} V^\sigma A^\mu B^\nu. \tag{1.10}$$

The Riemann tensor can be written as

$$R^{\alpha}_{\sigma\mu\nu} = \Gamma^{\alpha}_{\nu\sigma,\mu} - \Gamma^{\alpha}_{\mu\sigma,\mu} + \Gamma^{\alpha}_{\mu\lambda}\Gamma^{\lambda}_{\nu\sigma} - \Gamma^{\alpha}_{\nu\lambda}\Gamma^{\lambda}_{\mu\sigma} \tag{1.11}$$

The Ricci tensor is the contraction of the Riemann tensor

$$R_{\mu\nu} = R^{\alpha}_{\mu\alpha\nu} \tag{1.12}$$

and the Ricci scalar or scalar curvature is the contraction of the Ricci tensor.

$$R = g^{\mu\nu}R_{\mu\nu} \tag{1.13}$$

Einstein's equation relates the curvature of spacetime to the presence of energy and momentum:

$$G_{\mu\nu} = \kappa^2 T_{\mu\nu} \tag{1.14}$$

where the space-time geometry is on the left side of the equation and the energy content is on the right. One way of deriving the Einstein equation is to start from the Einstein-Hilbert action, which is the action for a vacuum:

$$S_H = \int \sqrt{-g}\ R\ d^n x \tag{1.15}$$

In classical field theory the equations of motion are those for which the action is stationary under variation (the principle of stationary action). By varying the action around the metric, using the principle of stationary action we get Einstein's field equations in a vacuum:

$$R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}R = 0 \tag{1.16}$$

To include matter we include the action for matter S_M, making up the total action:

$$S = \frac{1}{2\kappa^2}S_H + S_M \tag{1.17}$$

where $\kappa = \sqrt{8\pi G}$.

Using the same method as for vacuum one gets the Einstein equation (equation 1.14).

$$G_{\mu\nu} = \kappa^2 T_{\mu\nu} \tag{1.18}$$

where $G_{\mu\nu}$ is the Einstein tensor

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}R \tag{1.19}$$

and $T_{\mu\nu}$ is the stress-energy tensor. For a perfect fluid this is expressed as:

$$T_{\mu\nu} = (\rho + p)u_\mu u_\nu + p g_{\mu\nu}, \tag{1.20}$$

u_μ being the four-velocity of the fluid.

In cosmology we assume perfect fluids almost always, and it is typically a good approximation. The equation of state for one is

$$w = \frac{p}{\rho} \tag{1.21}$$

and for non-relativistic matter we have $w = 0$, for relativistic matter we have $\omega = 1/3$ and we think whatever plays the role of the cosmological constant has $w \approx -1$ [14].

1.1.2 Friedmann equations

In cosmology it is normal to assume that the universe is both isotropic and homogenious. This has also been measured using the cosmic microwave background and galaxy distributions, which look extremely isotropic. Under those assumptions the Friedmann-Lemaitre-Robertson-Walker (FLRW) line-element is a solution of GR.

$$\mathrm{d}s^2 = g_{\mu\nu}\mathrm{d}x^\mu \mathrm{d}x^\nu = -\mathrm{d}t^2 + a^2(t)\left[\frac{\mathrm{d}r^2}{1-kr^2} + r^2\left(\mathrm{d}\theta^2 + \sin^2\theta\ \mathrm{d}\phi^2\right)\right] \tag{1.22}$$

where $a(t)$ is the scale factor describing how large the Universe is relative to today at time t. The time is measured on "comoving" clocks, that is clocks that follow the expansion and thus have constant comoving coordinates. k indicates whether the universe is open, flat or closed.

Spacetime	k	Ω
Open	-1	$\Omega < 1$
Flat	0	$\Omega = 1$
Closed	1	$\Omega > 1$

Table 1.1: What different k values mean, summarized.

Ω in Table 1.1 is the energy in the Universe relative to the critical density. The critical density is the energy density in a flat universe, $\rho_c = 3H^2/8\pi G$. For a species i we use the notation $\Omega_i \equiv \rho_i/\rho_c$, ρ_c being the critical density. Current data [15] indicates that the Universe, if not flat, is at least very close to flat, so we should have $\Omega = \sum \Omega_i = 1$.

The FLRW metric has been successful in explaining the redshifted galaxies observed by Hubble in 1929 through the scale factor. It is not strictly the galaxies who have a velocity in the direction away from us, but the metric is changing "creating" extra space in between us and the galaxies. Similarly when the scale factor was smaller, there was less space between the particles so the Universe was a lot denser.

To determine how the scale factor evolves based on the Universe's energy content we use the Friedmann equations. To find the Friedmann equations we look at the time and space-components of the Einstein equation. The time component gives us Friedmann's first equation (commonly called the Friedmann equation).

$$G_{00} = \frac{3\dot{a}^2}{a^2} + \frac{3k}{a^2} \tag{1.23}$$

$$G_{ii} = -\frac{2\ddot{a}}{a} - \frac{\dot{a}}{a^2} - \frac{k}{a^2} \tag{1.24}$$

Modeling the components of the cosmos as perfect fluids is a good approximation on large scales. A perfect fluid has neither thermal conductivity nor viscosity. So using equation 1.20 we get $T_{00} = \rho$ and $T_{ii} = p$. The first and second Friedmann equations are then:

$$\frac{\dot{a}^2}{a^2} = \frac{\kappa^2}{3}\rho - \frac{k}{a^2} \tag{1.25}$$

$$\frac{\ddot{a}}{a} = \frac{8\pi G}{3}(\rho + 3p) \tag{1.26}$$

If we know the energy content of the Universe, we can therefore determine the evolution of the scale factor. The evolution of the Universe thus depends on the energy make up of the Universe and the energy make up of the universe changes as the Universe evolves. The radiation

energy density evolves as $\rho_r \propto a^{-4}$ as the number density of photons decreases as the volume it occupies expands and additionally the wavelength is stretched. The same logic applies to matter density (baryons and dark matter) except there is no wavelength so $\rho_m \propto a^{-3}$. The energy density of dark energy is however constant. Thus early on radiation dominates, before matter dominates and then dark energy will dominate. The first Friedmann equation tells us that in order for the Universe to start contracting in a flat universe we must have no energy density (to get $\dot{a} = 0$), so as long as our Universe is flat it will always be expanding. The second Friedmann equation tells us that in the radiation dominated era the expansion will decelerate relative to the size of the Universe, while in the matter dominated era it will still decelerate, but less so than during the radiation dominated era as the matter is pressureless. However as soon as dark energy becomes important the sign of $\ddot{a}$ changes and the expansion starts to accelerate and will approach the relative rate of acceleration, $\ddot{a}/a$, for a Universe with only dark energy asymptotically.

Taking another look at the second Friedmann equation we see that any type of energy with an equation of state that has $w < -1/3$ (e.g. a scalar field or a type of exotic fluid) can give $\ddot{a} > 0$ and thus an accelerating universe.

1.1.3 Linear cosmological perturbation theory

In this section, we present some relevant results from linear perturbation theory. We follow the discussion in [16].

The real Universe has inhomogeneities. One can introduce linear perturbations into the metric and stress energy tensor to take into account some aspect of the inhomogeneities, at least on large scales. This allows us to analytically explore how the inhomogeneities evolve over time, which makes it simple and computationally cheap compared to running simulations. When looking at the Universe on large scales, linear theory works very well, however when we are interested smaller scales, scales smaller than roughly 10 Mpc, the non-linear terms we ignore in linear theories become important. In those cases, simulations serve an important role.

One way of adding perturbations to the FLRW metric is by using the Newtonian gauge:

$$\mathrm{d}s^2 = -(1 + 2\Psi)\mathrm{d}t^2 + a^2(t)(1 - 2\Psi)(\mathrm{d}x^2 + \mathrm{d}y^2 + \mathrm{d}z^2) \tag{1.27}$$

where Ψ corresponds to the Newtonian gravitational potential and Φ is the perturbation to curvature [16]. We will only consider scalar perturbations as they are dominant, but vector and tensor perturbations also exist. The solutions in linear theory are well known and can be found in [16]. They are generally given in Fourier space, where perturbations of each wavelength start evolving independently when they are within the particle horizon (wavelength shorter than the distance light has had time to travel since the beginning of the Universe). The initial conditions for the evolution are set by inflation and are Gaussian fractional deviations from the mean density, and statistically isotropic. Here we will present some key results [16]:

The perturbations we are interested in is the perturbations to the matter density, which is given in terms if its fractional deviation from the mean matter density $\delta = (\rho - \bar{\rho})/\bar{(\rho)}$. In Fourier space, $\delta(k, a) = D(a)\delta(k, a = 1)$. D, the growth function, describes how the density perturbation grows with time. It is given by the linear growth equation [17],

$$\ddot{D} + aH\dot{D} - \frac{3}{2}(aH)^2\Omega_M(a)D = 0, \tag{1.28}$$

where the dots are derivatives with respect to conformal time. The linear growth rate describes how quickly the density perturbations grow relative to the scale factor, a, of the universe.

$$f \equiv \frac{\mathrm{d}\ln D}{\mathrm{d}\ln a} \tag{1.29}$$

This can be used to derive the linear velocity field which is given by:

$$v(k,a) = \frac{ifaH\delta(k,a)}{k} \tag{1.30}$$

1.2 Modified Gravity

Modifying gravity is motivated by explaining the accelerated expansion of the universe without the fine-tuning that appears in ΛCDM. There are not any good empirical reasons for modifying GR just yet, just theoretical ones.

Successful Modified Gravity (MG) theories must abide by a set of conditions. They need to explain something that is not explained by standard gravity, like the cosmological constant problem, otherwise we have the same problem as we have with the standard model. For the same reason it should not introduce another fine-tuning, or no progress has been made. The theory needs to be different enough from standard gravity that it can be distinguished from it through its physical effects on the Universe, otherwise, though it might have theoretical advantages in terms of fine-tuning, it does not justify it possibly being more complex; it would need to be simpler than standard gravity, which they generally are not. They also need to be within the current constraints on deviations from GR. This means that in the solar system it must reduce to GR (or extremely close to GR) as GR is well tested within our solar system.

1.2.1 The alternatives

There are many ways of modifying gravity. It is done by adding higher order terms to the action, adding scalar fields, vector fields, tensor fields, or adding extra dimensions, and many other ways [8].

1.2.2 Conformal transformations

In this section we explain what conformal transformations are and why they are useful. We follow the discussion in [13].

A conformal transformation is a changing of scales locally. The metric is multiplied by a time and space dependent function:

$$\tilde{g_{\mu\nu}} = \omega(x) g_{\mu\nu} \tag{1.31}$$

$$\tilde{ds}^2 = \omega^2(x)\ ds^2 \tag{1.32}$$

This changes all non-zero spacetime distances, but leaves spacelike intervals spacelike and timelike intervals timelike. Angles are left alone, leaving the casual structure intact. The null-geodesics ($ds^2 = 0$) do not change either, they are conformally invariant. When changing the metric we consequently also change the Riemann tensor, the Ricci tensor and the Ricci scalar.

These kinds of transformations are very useful, they allow us to get equations in different theories into similar forms so it is easier for us to compare them. We call these different forms *conformal frames*, two of the most important being the Jordan frame and the Einstein Frame. Consistently, variables will be in the Einstein frame when nothing else it indicated; however a tilde over a variable will indicate that it is in the Jordan frame ($\tilde{g}_{\mu\nu}$).

The Einstein frame is found by using the transformation that gets the action in the form [18]:

$$S = \int d^4x\ \sqrt{-g}\frac{R}{2\kappa^2} + \dots \tag{1.33}$$

which is the Einstein-Hilbert action with additional terms.

In this frame our the modifications to general relativity will appear in form of extra energy content (e.g. a new scalar field). In the Jordan frame particles instead follow the geodesics of their metric (WEP satisfied for massless particles) and the energy-momentum tensor is covariantly conserved, $\tilde{\nabla}_\mu \tilde{T}^{\mu\nu} = 0$.

1.2.3 Scalar-tensor Theories

In this section we present the basics of a type of modified gravity theories called scalar-tensor theories. We follow [8] and [13].

Scalar-tensor theories have a long history starting with Brans and Dicke in the 1960s [8] and have a few very useful features.

A scalar field is added that couples with the curvature scalar. This changes the Lagrangian, giving:

$$\mathcal{L} = \frac{1}{2\kappa^2}\sqrt{-g}[f(\phi)R - g(\phi)\nabla_\mu\phi\nabla^\mu\phi - 2\Lambda(\phi)] + \mathcal{L}_m(\mathbf{\Psi}, h(\phi)g_{\mu\nu}) \tag{1.34}$$

Note that we can change to a form where $g(\phi) = 1$ to recover a form where the standard scalar field action, S_ϕ, is one of the terms in the total action. This leaves us with two independent functions.

Using the conformal transformation $h(\phi)g_{\mu\nu} = \tilde{g}_{\mu\nu}$ we get the Jordan frame, and by re-defining the field (using that there are only two free functions) we arrive at a form given by the "coupling parameter" $\omega(\phi)$ and $\Lambda(\phi)$.

1.2.4 Screening mechanisms

Screening mechanisms suppress the effects of modifying gravity when a set of conditions are in place. The point of screening mechanisms is to bypass the stringent constraints that have been placed on gravity by solar system and lab tests [8]. Having a screening mechanism can make an MG theory valid even though it is significantly different from GR when not screened. When looking at screening mechanisms we will approach it from the Einstein frame as is the norm in the literature. In this frame the modification to GR appears in the form of a fifth force that strengthens the gravitational attraction between two objects, thus this force is suppressed in areas that are screened. A consequence of this extra force is that the density perturbations grow a bit faster due to the extra attraction between particles. Similarly lensing is also affected since gravity is effectively stronger and thus objects bend the path of light more.

1.2.4.1 Chameleon

The chameleon mechanism is based on a scalar field that propagates a fifth force over a long range when its mass is small, and a shorter range when its mass is large. The chameleon takes on a large mass when the ambient density is large and a low mass when the ambient density is low. This makes the fifth force hard to detect in dense areas, yet significant on cosmological scales. We follow the discussion in Waterhouse (2006) [19].

In the Einstein frame:

$$S = \int d^4x\ \sqrt{-g}\left(\frac{1}{2\kappa^2}R - \frac{1}{2}\nabla_\mu\psi\nabla^\mu\psi - V(\psi)\right) + S_m\left[g_{\mu\nu}A^2(\psi)\right] \tag{1.35}$$

where $A_i(\phi)$ is the matter coupling to species i and $V(\phi)$ is the potential of the scalar field. This is a very general scalar tensor theory. In the chameleon model we have,

$$A_i^2(\phi) = e^{2\beta_i\phi/M_{PL}} \tag{1.36}$$

Applying the principle of stationary action, varying with respect to ϕ, using the FLRW metric and the Klein-Gordon equation, $\nabla^2\phi = -(\ddot{\phi} + 3H\dot{\phi})$, one gets the cosmological equation of motion for ϕ:

$$\ddot{\phi} + 3H\dot{\phi} = -V_{eff,\phi}(\phi). \tag{1.37}$$

Where $V_{\text{eff},\phi} \equiv \partial V_{\text{eff}}/\partial\phi$ and we have used that the relation between the Einstein and Jordan frame energy densities of a species is $\rho \equiv e^{3(1+\omega_i)\beta_i\phi/M_{PL}}\tilde{\rho}$ to define an effective potential, V_{eff}, consisting of the potential energy of the field itself in addition to the contribution of matter:

$$V_{\text{eff}}(\phi) \equiv V(\phi) + \sum_i \rho_i e^{(1-3\omega_i)\beta_i\phi/M_{PL}} \tag{1.38}$$

When choosing the potential we want to keep in mind the problems we want to solve. Remembering that dark energy has an equation of state of $\omega_{DE} \approx -1$ and that for a scalar field

$$\omega_\phi = \frac{p_\phi}{\rho_\phi} = \frac{\frac{1}{2}\dot{\phi}^2 - V(\phi)}{\frac{1}{2}\dot{\phi}^2 + V(\phi)} \tag{1.39}$$

it is immediately obvious that $\omega_\phi \approx -1$ when $\dot{\phi}^2/2 \ll V(\phi)$. That is, we should have a slow rolling ϕ. Also we want the field to have been in effect for most of the time since the beginning. Additionally $V(\phi) = \rho_{DE}$ and we would prefer to achieve this without having to fine tune a constant in the potential. This sets some conditions on the properties of the potential; it must have a minimum that the scalar field is slowly rolling towards and the potential must gradually flatten towards the minimum.

The mass of the field is

$$m^2 \equiv V_{\text{eff},\phi\phi}(\phi) \tag{1.40}$$

and

$$m^2_{\text{min}} \equiv m^2(\phi_{\text{min}}), \tag{1.41}$$

where $1/m_{min}$ is the characteristic range of the chameleon fifth force and ϕ_{min} is the fields' value at the minimum of the effective potential, which we can find by solving for the ϕ that makes the derivative of the effective potential with respect to ϕ zero.

So if we increase the ambient matter density, then $V_{,\phi}(\phi_{min})$ decreases, and since the field is slowly rolling towards a minimum, it is an increasing function of ϕ, then ϕ_{min} has to decrease as well. This means that $V_{,\phi\phi}(\phi_{min})$ increases as it is a decreasing function of ϕ, in turn increasing m_{min} and thus decreasing the range of the fifth force. This is how the screening occurs. In areas of a high matter density the field takes on a large mass and thus the force from the field (pulling in the same direction as standard gravity) acquires a lower range.

The chameleon force, the fifth force exerted by the chameleon field, is not there in the Jordan frame, as in the Jordan frame particles follow the geodesics. In the Einstein frame however, we see that they do not. This means when we set up the geodesic in the Jordan frame there is no force term, however when we convert to the Einstein frame a force term appears:

$$\frac{\vec{F}_\phi}{m} = -\frac{\beta_i}{M_{Pl}}\vec{\nabla}\phi \tag{1.42}$$

We already know what the time-component of the Einstein tensor (equation 1.24) is and assuming the field couples equally to all matter species we get the Chameleon Friedmann equation,

$$3H^2M_{PL}^2 = \frac{1}{2}\dot{\phi}^2 + V(\phi) + \rho_m e^{\beta\phi/M_{PL}} + \rho_r. \tag{1.43}$$

f(R) Chameleon screening

One f(R) model that exhibits chameleon screening is the Hu & Sawicki model [20]. In this model the action in the Jordan frame is

$$S = \int \mathrm{d}^4x \, \sqrt{-\tilde{g}} \left[\frac{\tilde{R} + f(\tilde{R})}{2\kappa^2} + \mathcal{L}_m \right] \tag{1.44}$$

$$f(R) = -m^2 \frac{c_1(\tilde{R}/m^2)^n}{c_2(\tilde{R}/m^2)^n + 1} \tag{1.45}$$

where $n > 0$ and m^2 is defined as:

$$m^2 \equiv \frac{\kappa^2 \bar{\rho}_0}{3} = (8315 Mpc)^{-2} \left(\frac{\Omega_m h^2}{0.13} \right) \tag{1.46}$$

$\bar{\rho}_0$ being todays' average density. Hu & Sawicki chose that the second derivative of $f(\tilde{R})$ would be above zero and when $m^2/\tilde{R} \to 0$ we can write [20]

$$\lim_{m^2/R \to 0} f(R) \approx -\frac{c_1}{c_2} m^2 + \frac{c_1}{c_2^2} m^2 \left(\frac{m^2}{R} \right)^n. \tag{1.47}$$

We can see straight away that if c_1/c_2 is constant and $c_1/c_2^2 \to 0$ then we would have a cosmological constant. From that they got the modified Friedmann equation

$$\tilde{H}^2 - f_R(\tilde{H}\tilde{H}' + \tilde{H}^2) + \frac{1}{6} f + \tilde{H}^2 f_{RR} \tilde{R}' = \frac{\kappa^2 \bar{\rho}}{3}, \tag{1.48}$$

where $' \equiv \mathrm{d}/\mathrm{d} \ln a$, f_R is the derivative of $f(R)$ and f_{RR} is derivative and double derivative of $f(R)$ respectively with respect to R.

To get an expansion history like in ΛCDM it is needed that $c_1/c_2 \approx 6\Omega_\Lambda/\Omega_m$. This allows us to use only two free parameters instead of the three c_1, c_2 and n. It is common to use n and f_{R0} where the latter is f_R at the present:

$$f_{R0} = -n \frac{c_1}{c_2^2} \left(\frac{\Omega_m}{3(\Omega_m + 4\Omega_\Lambda)} \right)^{n+1} \tag{1.49}$$

When recast to a chameleon scalar-tensor theory the Hu & Sawicki model takes on $\beta = 1/\sqrt{6}$ and the range of the fifth force today is [21]:

$$\lambda_\phi^0 = 3 \sqrt{\frac{(n+1)}{\Omega_m + 4\Omega_\Lambda}} \sqrt{\frac{|f_{R0}|}{10^{-6}}} \ \mathrm{Mpc}/h. \tag{1.50}$$

1.2.4.2 Symmetron

In this section we present and explain the most important principles and results of the symmetron mechanism. We follow the discussion in [22].

In the symmetron model we have a scalar field potential $V(\phi)$ and matter coupling $A(\phi)$ that are symmetric, $V(-\phi) = V(\phi)$ and $A(\phi) = A(-\phi)$. The screening occurs because the field has zero Vacuum Expectation Value (VEV) in high density, and a large VEV in low density areas,

while the matter coupling, $A(\phi)$ determines the strength of the force through its derivative. The simplest potential and matter coupling are:

$$V(\phi) = -\frac{1}{2}\mu^2\phi^2 + \frac{1}{2}\lambda\phi^4, \tag{1.51}$$

$$A(\phi) = 1 + \frac{\phi^2}{2M^2} + \mathcal{O}\left(\frac{\phi^4}{M^4}\right), \tag{1.52}$$

so the smaller the mass scale M is, and the larger the ϕ is, the stronger the fifth force is. The parameter μ is also a mass scale and λ is dimensionless, both determining the shape of the potential. Starting with the Einstein frame action of a scalar-tensor-theory [22]:

$$S = \int \mathrm{d}^4x\ \sqrt{-g}\left[\frac{1}{2\kappa^2}R - \frac{1}{2}\delta_\mu\phi\delta^\mu\phi - V(\phi)\right] + \int \mathrm{d}^4x\ \sqrt{-\tilde{g}}\mathcal{L}_m(\psi, \tilde{g}_{\mu\nu}) \tag{1.53}$$

where we have used the Jordan frame metric in the matter action, $\tilde{g}_{\mu\nu} = A^2(\phi)g_{\mu\nu}$. This results in an equation of motion for the scalar field given by

$$\Box\phi - V_{,\phi}(\phi) = -A^3(\phi)A_{,\phi}(\phi)\tilde{T}, \tag{1.54}$$

$\tilde{T} = \tilde{g}^{\mu\nu}\tilde{T}_{\mu\nu}$ being the trace of $\tilde{T}_{\mu\nu}$. The effective potential, $V_{\mathrm{eff}}(\phi) = V(\phi) + A^3(\phi)A_{,\phi}(\phi)\tilde{\rho}$, for the simple potential is:

$$V_{\mathrm{eff}}(\phi) = \frac{1}{2}\left(\frac{\rho}{M^2} - \mu^2\right)\phi^2 + \frac{1}{4}\phi^4 \tag{1.55}$$

using $\tilde{T} \approx -\tilde{\rho}$ (assuming no pressure). The first term in the effective potential plays a key role, when it become negative we say that symmetry breaks. If the symmetry is not broken, there is no fifth force since the VEV of the field is zero.

In the FLRW medric and assuming non-interacting perfect fluids the Einstein equations, ones again found by varying the Einstein frame action, become

$$\frac{1}{\kappa^2}G_{\mu\nu} = T^{(\phi)}_{\mu\nu} + A^2(\phi)\tilde{T}_{\mu\nu}. \tag{1.56}$$

Like for the Chameleon, a force appears in the geodesic equation when converting to the Einstein frame and for non-relativistic particles this is the equivalent of

$$\ddot{\mathbf{x}} = -\vec{\nabla}(\log A), \tag{1.57}$$

so the logarithm of the matter coupling is the fifth force potential.

In Figure 1.2 we see that when there is sufficiently high density, ϕ is trapped in the middle. However in vacuum the middle is unstable and ϕ will roll down one of the sides. This is in essence how the screening happens. The modified Friedmann equation for the Symmetron is:

$$3M_{Pl}^2H^2 = \frac{1}{2}\dot{\phi}^2 + V_{\mathrm{eff}}(\phi). \tag{1.58}$$

1.2.4.3 Spherical solutions

It is very useful to have a solution of spherically symmetric density distributions, as most objects over a certain size are well approximated by this, like for example dark matter halos and stars. We assume a density function

$$\tilde{\rho}(r) = \begin{cases} \rho_c & : r < R \\ \rho_\infty & : r > R \end{cases} \tag{1.59}$$

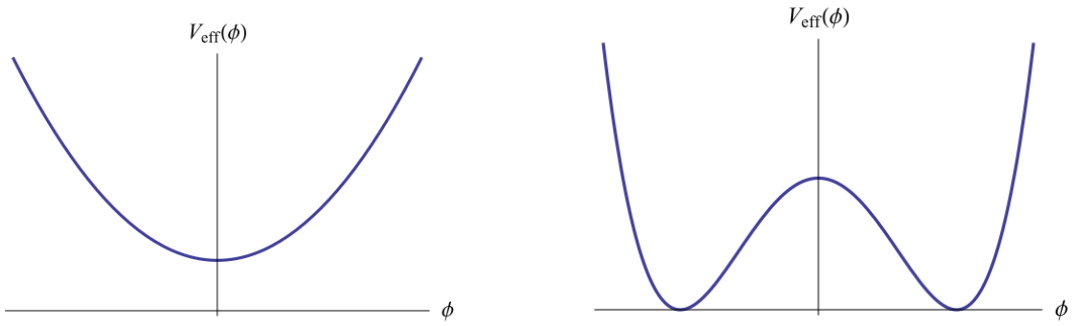


Figure 1.2: An example of what the effective potential, V_{eff}, would look like in an area of high density (a) and an area of low density (b). This is taken from Figure 1 in [22].

where R is the radius of the object. Using that $\nabla^2\phi = V_{\text{eff},\phi}(\phi)$, assuming non-relativistic matter ($\omega_i = 0$), flat space and that there is no time-dependence,

$$\vec{\nabla}^2\phi = V_{,\phi}(\phi) + \frac{\beta}{M_{PL}}\rho e^{\beta\phi/M_{PL}}. \tag{1.60}$$

We set as boundary conditions that the field should be smooth over the origin and that when we go far away from the object $\phi \to \phi_\infty$ where ϕ_∞ is the value of the field in vacuum approaching a distance of infinity away from the closest object. In the symmetron we use $\phi_\infty = \phi_0$.

Outside the sphere the solution is

$$\phi_{out}(r) = A\frac{e^{-m_\infty(r-R)}}{r} + \phi_\infty \tag{1.61}$$

One way of solving for the inside is to divide the interval $[0, R]$ into $[0, R_c]$ and $[R_c, R]$, where we have $\phi \sim \phi_c$ for the first interval and $\phi \gg \phi_c$ for the second. Three different solutions can then be given for both Symmetron and Chameleon screening: the low contrast solution where $R_c = R$, the thick-shell solution where $R_c = 0$ and the thin-shell solution where $0 < R_c < R$. The thick and thin-shell solutions for the Chameleon outside of the sphere are:

$$\phi_{\text{thick}}(r) \approx -\frac{\beta}{4\pi M_{Pl}}M_c\frac{e^{-M_\infty(r-R)}}{r} + \phi_\infty \tag{1.62}$$

$$\phi_{\text{thin}}(r) \approx -\frac{3\beta}{4\pi M_{Pl}}\frac{\Delta R}{R}M_c\frac{e^{-M_\infty(r-R)}}{r} + \phi_\infty \tag{1.63}$$

where $M_c = 4\pi R^3\rho_c/3$ is the mass of the sphere and $\Delta R/R$ is what we call the "thin-shell suppression factor" [19]. For the Symmetron, the solution becomes one.

The thin-shell suppression factor for the Chameleon and the Symmetron are:

$$\frac{\Delta R}{R} \equiv \frac{M_{Pl}(\phi_\infty - \phi_c)}{\beta\rho_c R^2} \tag{1.64}$$

$$\frac{\Delta R}{R} \equiv \frac{M^2}{\rho R^2} = \frac{M^2}{6M_{Pl}^2\Phi} = \frac{\phi_0}{6gM_{\text{Pl}}\Phi} = \frac{M_{\text{Pl}}\phi_0}{g\rho_c R^2} \tag{1.65}$$

where $g \equiv \phi_0\sqrt{\lambda}/\mu$ determines the strength of the fifth force relative to the Newtonian approximation of the standard gravitational force $F_\phi = 2g^2 F_N$. Similarly the strength of the Chameleon fifth force is determined by β, $F_\phi = 2\beta^2 F_N$. We see that the thin-shell suppression factors for the Chameleon and the Symmetron mechanisms are equivalent if we let ϕ_0 be interpreted as the difference between the Chameleon field value far away (ambient Chameleon value) and ϕ_c. If a whole area has a density higher than average then the effective ambient Chameleon value or ϕ_0 is effectively smaller in that area, so objects lying in higher density environments are more likely to be screened. When $\Delta R > R$ we use the thick-shell solution and when $\Delta R << R$ we use the thin-shell solution.

Even the strength of the force from screened ($\Delta R/R \ll 1$) and unscreened ($\Delta R/R \gg 1$) objects on test particles behave similarly:

$$\frac{F_\phi}{F_N}|_{\text{screened}} \simeq 6\xi^2\frac{\Delta R}{R} \ll 1 \tag{1.66}$$

$$\frac{F_\phi}{F_N}|_{\text{unscreened}} \simeq 2\xi^2 \tag{1.67}$$

where $\xi = \beta$ in the Chameleon [23] and $\xi = g$ in the Symmetron [22].

As a consequence of the screening, gravity will appear to work differently on a screened object than an unscreened object, even if their mass is the same, apparently breaking the principle of equivalence. Some examples of consequences of this is that in an unscreened galaxy (e.g. a dwarf galaxy) the gas, which is unscreened, will orbit faster than the stars, (which are screened) [24]. Similarly smaller galaxies should leave voids faster then larger galaxies on average [24].

1.3 Large scale structure

When observing the Universe we are not interested in exactly where every galaxy is or where every star is. This is because this could be different in different realisations of the initial fluctuation set up during inflation. Moreover it does not tell us anything of theoretical use. The overall statistical properties of the Universe on large scales should however be the same, no matter the realisation. These properties could be things like how much galaxies tend to cluster together or how fast they move. This is dependent on what the Universe consists of and which physical laws guide its evolution, so by using many different statistics one can narrow down on a set of viable models. Since initial perturbations are Gaussian, a probability distribution purely characterized by its variance, the second moment, it should be possible to extract all of the information using two-point statistics, which is why two-point statistics are the most common.

Most of the matter in the Universe is dark matter and does not emit any sort of electromagnetic radiation that we have been able to measure. So the only way we are able to observe it is from its gravitational effects on baryonic matter, matter that does radiate. Since the dark matter interacts gravitationally with baryonic matter, the baryons tend to fall into the potential wells of what are called dark matter haloes, large clumps of gravitationally bound dark matter. If the halo is dense enough it can collect enough baryons to form a galaxy. Really large halos can have halos within them, sub-halos, forming a galaxy within each, making a galaxy cluster. We can use the galaxies to trace the dark matter, for example if a halo moves in one direction then its galaxy moves with it. Larger halos are more massive and thus attract each other more then halos of lower mass, and large halos attract large haloes more strongly than smaller halos. So the more massive the halos the more they will cluster, gather in groups. We call this difference in clustering for the “bias”, or galaxy bias when we talk about galaxies specifically.

To measure the different statistics, the nature of baryonic matter is used. One of them is that hydrogen gas has an emission line at 21 cm. By mapping the intensity of it one maps where there is hydrogen gas and how much [25]. This can be done for galaxies with other emission lines, e.g. [26]. Another method is to map galaxy clusters using the Thermal Sunyaev-Zel'dovich effect, which is that when photons from the CMB scatter with high-energy electrons they gain energy, causing small distortions in the CMB [27]. Other methods are for example to use distortions in images of objects where the path of the photons towards us have been changed by the gravity of dark matter halos, causing distortion in the image; this is called gravitational lensing [25].

1.3.1 Dark matter halos

Halos are gravitationally bound objects. Defining properties like their center and the edge of as halo is a bit tricky. In halo catalogues many different definitions are used and it is therefore very important to be aware of which ones were used in order to use it properly.

One of these is whether or not the mass of a halo within another halo, a subhalo, is bart of the mass of its parent halo. If for example we were to compare data from n-body simulations (see Section 2) to that of a gravitational lensing survey, that survey would have a parent halo in its resulting catalogue. So if the catalogue from the nbody simulations has subtracted the mass of the subhalo from the parent halo we would not be comparing the same thing.

The other differences usually are not as important, like whether the edge of a halo is defined by the bound particle furthest from the center or where bound particles have to have zero velocity (or else they would escape, as in not bound).

When we measure the mass of a halo, e.g. by gravitational lensing, the mass we deduce will be all of the mass within the volume of the halos, not just the bound mass.

1.3.2 Galaxy correlation function

1.3.2.1 Analytics

The two-point galaxy correlation function [28] describes how galaxies cluster relative to be placed randomly like in a uniform random Poisson point process. In a uniform random Poisson point process, each point is equally likely to appear in any location independent of the locations of the other points. The number of points in an area follow a Poisson distribution for this kind of process.

In this section we follow Peebles' book: The Large-Scale Structure of the Universe, [28].

Say we have a number density n, then the average number of galaxies in a volume V is $\langle N \rangle = nV$. The probability of finding a galaxy in an infinitesimal volume is $\delta P = n\delta V$. As previously noted, if the galaxies were distributed like a random Poisson process the location of each galaxy would be independent of each other. So what is the probability how having a galaxy in two separate infinitesimal volumes? In this case we have two independent events which means that the probability of both happening is the product of the each events' probability:

$$\delta P = n^2 \delta V_1 \delta V_2 \tag{1.68}$$

When it is a random Poisson process where we place the volumes does not matter. The two-point correlation function describes how the separation of the volumes changes the probability:

$$\delta P \equiv n^2 \delta V_1 \delta V_2 \left[1 + \xi(r_{12})\right] \tag{1.69}$$

$$\xi(r_{12}) = \frac{\delta P}{n^2 \delta V_1 \delta V_2} - 1 \tag{1.70}$$

where r_{12} is the distance between volume one and two. So for the Random Poisson Process (RPP) the correlation function ξ is zero as the separation do not change the probability. When $\xi(r) > 0$ it is more likely that two volumes separated by the distance r both contain an object, we say that the positions are correlated. If $\xi(r) < 0$ the positions are anticorrelated.

Another way of looking at it is that if we know there is a galaxy in one volume then the probability of finding one in another infinitesimal volume a distance r away is:

$$\delta P = n \delta V \left[1 + \xi(r)\right] \tag{1.71}$$

$$\xi(r) = \frac{\delta P}{n \delta V} - 1 \tag{1.72}$$

1.3.2.2 Using halo a catalogue to find the correlation function

This section is a simple example to inform what we will do with the velocity statistics in Chapter 3.

If we have a galaxy catalogue and want to calculate the galaxy correlation function then equation 1.70 does not exactly make it obvious how we should do it. Intuitively we would just count the number of galaxies, bin the distances from a galaxy, count the number of other galaxies within a distance bin and divide by the number density times the volume of the bin. Then if the number of galaxies are equal to nV we get $\xi(r) = 0$ like we should.

In order to calculate the function we have to consider a couple of things. First, a halo at the corner of the volume spanned by the data will have fewer neighbours than one in the middle. Second, if the data is from for example an N-body simulation the boundary contitions are likely to be periodic. This is however not significant on scales much smaller than the data volume.

In order to deal with this problem several estimators ([29]) have been developed to estimate the correlation function. The fundamental idea is to compare the distribution in the data to the one in a randomly generated dataset, where both the geometry and the volume that the data set spans is the same. Most of the estimators are built up from three quantities:

$$DD(s) = \frac{dd(s)}{n_d(n_d - 1)/2} \tag{1.73}$$

$$RR(s) = \frac{rr(s)}{n_r(n_r - 1)/2} \tag{1.74}$$

$$DR(s) = \frac{dr(s)}{n_r n_d} \tag{1.75}$$

where $dd(s)$ is the number of halo-pairs in the actual data set seperated by a comoving distance s, $rr(s)$ is the same but for the random data. $dr(s)$ is the number of actual-random halo pairs (one from the actual data set and one from the random data set) with comoving seperation s.

The most commonly used estimator is the one developed by Landy and Szalay [30].

$$\xi(r) = \frac{DD(r) - 2\ DR(r) + RR(r)}{RR(r)} \tag{1.76}$$

This estimator has been shown to have Poisson variance when there is zero correlation, the actual catalogue volume is big enough and the random catalogue has enough generated data in it.

1.3.3 Bias

Dark matter halos are not distributed exactly like the underlying dark matter distribution. In fact they cluster more, depending on their mass.

The bias is a way of quantifying how much halos cluster relative to the underlying dark matter. Normally one assumes a linear bias which is defined as,

$$b \equiv \left(\frac{\xi_h}{\xi_{dm}} \right)^{1/2} \tag{1.77}$$

Mo & White (1996) [31] showed that a linear bias was a good approximation far into non-linear scales and developed an analytical model based on the Press & Schechter (PS) [32] formalism that PS developed in 1974. This model was shown to be quite accurate when comparing to n-body simulation data and a slight inaccuracy was due to the analytical mass function not matching the one found in the simulations [33].

1.4 Velocity statistics

1.4.1 Peculiar velocities

The only observable component to the velocity of galaxies is the radial velocity, which consists of two parts, velocity due to Hubble expansion and the peculiar velocity: $v = v_h + v_p$ where $v_h = H_0 a \chi$ where χ is the comoving distance to the galaxy from the observer.

Lets call $\vec{x_o}$ the observer position and $\vec{x_h}$ the halo position, then the vector from the observer to the halo is:

$$\vec{x} = \vec{x_h} - \vec{x_o} \tag{1.78}$$

Which gives us the radial velocity, which is the only part of the velocity that we can measure:

$$v_{rad} = \frac{\vec{x} \cdot \vec{v}}{|\vec{x}|} \tag{1.79}$$

In linear perturbation theory we have a relation between the comoving velocity field

$$\vec{v} = i f H_0 \delta(\vec{k}) \frac{\vec{k}}{k^2} \tag{1.80}$$

where $k^2 = k_i k^i$, H_0 is the Hubble constant and f is the linear growth rate (see Section 1.1.3).

The peculiar velocity of an object is due to the sum of the gravitational forces working on it, which means they should be different for different theories of gravitation. Since the velocities are proportional to the growth rate f, which often is appoximated as $f \approx \Omega_m^\gamma$, they can be used to measure it, or γ, like was done in [34].

To find the peculiar velocity we now need to subtract the Hubble part. This is very hard to due accurately as is is hard to measure positions accurately. One of the ways of measuring the position of a galaxy is to use the Tully-Fisher relation, which is an empirical formula for how the luminosity of a spiral galaxy relates to the its rotational velocity, the last of which can be measured by measuring the width of the emission lines from the galaxy (the emission from matter rotating away from us will be redshifted compared to the matter rotating towards us). Once the luminosity of the galaxy has been infered we use the measure flux from it to determine the distance which allows us to separate the redshift caused by Hubble expansion and what is cause the the galaxy's peculiar velocity [25, 35].

1.4.2 Velocity correlation function

The correlation function is a statistic measuring correlations in the peculiar velocity fields, in this section we present it and its approximations following the discussion in [16].

The velocity correlation function is defined as:

$$\xi_v(r) = \langle v_{p,i}\ v_{p,j} \rangle \tag{1.81}$$

where r is the distance between the galaxies.

So it is the average product of the peculiar velocities of objects seperated by a distance r.

The peculiar velocities are measure from an observing point. Halos close to each other will tend have a comoving velocity in the direction of each other on average as they are more likely to be part of the same larger structure, and the closer they are the stronger this trend will be as the other halo becomes a more dominant attractor when it is close. Therefore the velocity correlation function is relatively steep at smaller r.

Using the linear approximation for the velocity field (see Section 1.1.3) and that we can write

$$\xi_v(r) = \langle \vec{v_1} \cdot \hat{r_1}\ \vec{v_2} \cdot \hat{r_2} \rangle\,, \tag{1.82}$$

we can find that a linear approximation of the velocity correlation function is decomposed in a parallel and perpendicular component, where the parallel direction is along the line of sight between the two galaxies.

$$\begin{matrix} \xi_{v,\perp}(r) \\ \xi_{v,\parallel}(r) \end{matrix} = \int_0^\infty \frac{\mathrm{d}k}{2\pi^2 k} P(k) \begin{pmatrix} -f^2 H_0^2 j_0'(kx)/x \\ -f^2 H_0^2 k j_0''(kx) \end{pmatrix} \tag{1.83}$$

where $j_0(x)$ is the zeroth spherical Bessel function and $j_0'(x)$ is the derivative with respect to x.

1.4.3 Streaming pairwise velocity

The streaming pairwise velocity is a measure of galaxies tendency to approach each other [28]. We follow the discussion in [36]

The pair-density weighted relative velocity is given by

$$\vec{v}_{12} = \frac{\langle (\vec{v}_1 - \vec{v}_2)(1+\delta_1)(1+\delta_2) \rangle}{1+\xi(r)} \tag{1.84}$$

where the $\langle \dots \rangle$ indicates that we average over all pairs separated by the distance r. $\xi(r) = \langle \delta_1 \delta_2 \rangle$ is the two-point correlation function.

Assuming Gaussian initial conditions $\vec{v}_{12}$ is related to the

$$v_{12}(r) = -\frac{2}{3} H r f \bar{\xi}(r) [1 + \alpha \bar{\xi}(r)] \tag{1.85}$$

where $f = d\ln D / d\ln a$, $D(a)$ is the linear growing mode solution, $\bar{\xi}(r)$ is defined as $\bar{\xi}(r) \equiv \frac{\xi(r)}{1+\xi(r)}$ and $\xi(r)$ is given by $\xi(r) = (3/r^3)\int_0^r \xi(x) x^2 \mathrm{d}x$; H is the Hubble constant and a is the scale factor as usual.

Chapter 2

N-body simulations

This section is in no way intended to give the reader a detailed understanding of N-body simulations, but rather a feel for how they work. The discussion of [37] has been used extensively in this section.

As the computational power to price ratio keep increases n-body simulations become more and more advanced. The same simulation that used to take a few days to run can now take a few minutes. We can have a higher number of particles and higher resolution on our simulations.

Cosmological n-body simulations are in nearly all cases run with comoving spatial coordinates and periodic boundary conditions on the box which is occupied by a number of particles

When looking at the large scale structure one often only includes dark matter in the simulation as baryonic matter is not that important if one is not interested in for example seperating spiral galaxies from elliptical galaxies.

The initial conditions are made by choosing a cosmological model (e.g. ΛCDM). In most cases Gaussian density fluctuations are used, as generated by inflation, specified by the power spectrum $P(k)$. These are then evolved linearly until typically $z \sim 100$ where the simulation starts. Then particle positions and velocities needs to be generated, normally by the use of the Zel'dovich approximation.

The equations used to calculate the movement of the particles are as follows

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \frac{1}{a}\vec{v} \tag{2.1}$$

$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} + H\vec{v} = \vec{g} \tag{2.2}$$

$$\vec{\nabla} \cdot \vec{g} = -4\pi G a[\rho(\vec{x}, t) - \bar{\rho}(t) \tag{2.3}$$

where $\vec{v}$ is the peculiar velocity, ρ and $\bar{\rho}$ is the mass density and mean mass density respectively. $\vec{\nabla} = \partial/\partial\vec{x}$ is the comoving gradient.

Every particle is evolved following equations 2.1 and 2.2, usually using the leapfrog algorithm. Calculating the gravitational acceleration between each pair of particles takes a lot of time therefore a number of techniques have been developed to avoid having to do so. All having their different take on how to solve equation 2.3, below we outline a few of them.

Tree codes recursively set up a hierarchy of cells containting at least one particle each. And then when calculating the force on a particle it treats groups of particles as one particle using the condition $s/d < \theta$ where s is the cell size and d is the distance from the particle we want to calculate the force on.

The idea behind the particle-mesh method is to divide up the box in a three-dimensional cartesian grid. Then the poisson equation is solved in the following three steps:

1. Assign each particles mass to its nearest points using an interpolation scheme (Cloud-in-Cell is the most common one) on the grid. Each point on the grid now has a mass which is the sum of all the mass assigned to it.

2. The Fourier space solution $\hat{\phi}(\vec{k},t) = -4\pi G a^2 \hat{\rho}(\vec{k},t)/k^2$ to the Poisson equation is calculated from the mass distribution of the grid (using Fast Fourier Transform). And then transform back from Fourier space.

3. Gravity is interpolated from the grid to the particles using the same interpolation scheme as was used in step one, or else particles could exert forces on themselves by their effective mass being positioned elsewhere with the first type of interpolation.

The number of operations used by the PM algorithm to evaluate the force on all the particles is of the order $O(N) + O(N_g \log N_g)$ [37].

The Particle-particle/particle-mesh algorithm is just the PM algorithm, except that one increases the force resolution by calculating the force from individual particle pairs seperated by a relatively short distance (typically less that two or three grid spacings).

One problem is the where strong clustering occurs the number of particles seperated by short distance can become very large, making the pair summation dominant. To tackle this problem one can place subgrids in highly gense sub-boxes, and then sub-grids within those again if needed. This we call adaptive P3M (AP3M). Using the sub-grids one the algorithm scales as $O(N\ \log N)$ [37]

2.1 Background on the code used

The data used in this project are from simulations run with the ISIS code [21], which is based on RAMSES [38], an open source n-body and hydrodynamical code. RAMSES is a highly parallelized code that uses an Adaptive Refinement Tree (ART) [39] inspired n-body solver and Adaptive Mesh Refinement (AMR) for its hydrodynamics (which we will not describe further here).

ISIS is a modification of RAMSES that adds the ability to do n-body simulations of scala-tensor theories. It solves the following equation

$$\ddot{\mathbf{x}} + 2H\dot{\mathbf{x}} = -\frac{1}{a^2}\nabla\Phi - \mathbf{g}(\phi, \nabla\phi, a), \tag{2.4}$$

using the equation of motion for the metric perturbation Φ:

$$\nabla^2\Phi = \frac{3}{2}\frac{\Omega_m H_0^2}{a}\delta \tag{2.5}$$

and the equation of motion of the field

$$\nabla^2\phi = (V_{,\phi} + A_{,\phi}\rho)a^2 = a^2 S(\rho,\phi) \tag{2.6}$$

where $S(\rho,\phi)$ and $\mathbf{g}(\phi,\nabla\phi,a)$ are functions that the dependent on the theory of gravity [21]. $\mathbf{x}$ is the position of a particle and $\dot{\mathbf{x}}$ is its derivative with respect to time.

ISIS also uses a non-linear multigrid solver rather than the linear one used by RAMSES, meaning that the grid can be divided uneavenly into subgrids. This means that the grid can be adjusted more carefully around where the higher densities are and we can get a more accurate force calculation.

2.2 Background cosmology and initial conditions

We ran three different simulations with the Hu-Sawicki $f(R)$ model [20] (see Section 2.4), four with the symmetron model (see Section 2.3), and one with standard gravity. The simulation is done using a box with 256 Mpc/h sides, holding 512^3 particles of mass $9.26490 \cdot 10^9\ M_{sun}/h$ each. This is a large enough box to study the matter structure at somewhat large cosmological scales, and the particles have a low enough mass that we can identify halos of Milky Way size and upwards reliably [40].

The background cosmology is given by the parameters $\Omega_{b,0} = 0.045$, $\Omega_{cdm,0} = 0.222$, $\Omega_\Lambda = 0.733$ and $H_0 = 71.9$ km/s/Mpc, these are consistent with the WMAP7 best-fit parameters [41, 42].

Gaussian initial conditions were generated using a software package called Cosmics [43]. The same initial conditions can be used in the modified gravity models and standard gravity, as at high redshift the matter density of the universe is high, so in effect the whole Universe is screened. The initial conditions being Gaussion means that the density at a certain point in space follows a Gaussian probability distribution, thus a random number generator (RNG) is needed to specify the density at each point. For the different models in this thesis, the same file with initial conditions was used in each simulation, so we have the same seed in all the simulations.

The simulation starts at $z = 50$. This is well before non-linear effects become important. Past $z = 50$ we have $\Omega_{cdm} = \Omega_m = 0.267$, $\Omega_\Lambda = 0.733$ and $H_0 = 71.9$ km/Mpc/s. Meaning that there are no baryons in the simulation itself. Baryons are important when the universe is small, the density of baryonic particles high, and the pressure between baryons is important making their behavior very different from dark matter which neither exerts nor experience any pressure.

2.3 Symmetron Simulations

For numerical stability, the scalar field is normalized by its vacuum expectation value, defining a new scalar field:

$$\chi \equiv \frac{\phi}{\phi_0} \tag{2.7}$$

where $\phi_0 = \mu/\sqrt{\lambda}$, μ is a mass scale and λ is a dimensionless constant used to form the potential $V(\phi) = -\mu^2\phi^2/2 + \lambda\phi^4/4$.

Normally one defines the fields potential $V(\phi)$ and conformal factor $A(\phi)$ by the parameters μ, λ and M. Here those are changed in favor of three new parameters [21]:

$$\lambda_0 = \frac{1}{\sqrt{2}\mu} \tag{2.8}$$

$$\beta = \frac{\phi_0 M_{Pl}}{M^2} \tag{2.9}$$

$$(z_{SSB} + 1)^3 = a_{SSB}^{-3} = \frac{\rho_{SSB}}{\rho_0} = \frac{\mu^2 M^2}{\rho_0} \tag{2.10}$$

The first parameter, λ_0, determines the length scale of the the fifth force; the second, β, is a dimensionless coupling constant controlling the strength of the force; and z_{SSB} is the redshift at which the symmetry of the potential breaks (the first term in the effective potential becomes negative) and the fifth force is turned on. These parameters are physically more intuitive,

as each parameter is related to the properties of the fifth force rather than the shape of the potential.

Model	λ_0	z_{SSB}	β
SymmA	1	1	1
SymmB	1	2	1
SymmC	1	1	2
SymmD	1	3	1

Table 2.1: The parameters for each Symmetron run. λ_0 is given in units of Mpc/h.

Simulations were run for the parameters given in Table 2.1. Of the models, SymmA will deviate the least from standard gravity, with the force turning on late ($z_{SSB} = 1$) and a coupling constant of one. All of the models have the same force length which was chosen as $\lambda = 1$ Mpc/h, because it is short enough to be physically viable (evade solar system tests and CMB based tests) while exhibiting interesting deviations from ΛCDM . If we use SymmA as a baseline, the other symmetron models have only one of the parameters changed each. This allows us to see what impact changing each of the parameters has on our data, and how they are related in importance.

2.4 f(R) Simulations

The Hu-Sawicki model [20] is, as described in Section 1.2.4.1, an $f(R)$ model recast to a scalar-tensor theory that exhibits Chameleon screening (see Section 1.2.4.1).

As was described in Section 1.2.4.1, the model has two free parameters, f_{R0} and n, where n is the power law index in the ansatz for $f(R)$ and f_{R0} is related to the parameters c_1 and c_2 used in the ansatz, they both are related to the range of the fifth force (at z = 0, today) by [21]:

$$\lambda_0 = 3\sqrt{\frac{n+1}{\Omega_m + 4\Omega_\Lambda}}\sqrt{\frac{|f_{R0}|}{10^{-6}}}\ \text{Mpc}/h. \tag{2.11}$$

Model	n	$\lvert f_{R0}\rvert$	λ_0
F4	1	10^{-4}	23.7
F5	1	10^{-5}	7.5
F6	1	10^{-6}	2.4

Table 2.2: The parameters for each $f(R)$ run. λ_0 is given in units of Mpc/h

The parameters used in the three different simulation runs are given in Table 2.2. They are commonly used in the literature [44, 45, 46, 47, 48], where they are referred to as F4, F5 and F6.

The range of the fifth force increases the larger $|f_{R0}|$ is, so in order of decreasing force length we have F4, F5 and F6. F6 thus deviating the least from standard gravity. The strength of the force does not vary however as we always have $\beta = 1/\sqrt{6}$ in the Hu-Sawicki model. This set of parameters spans the viable parameter space fairly well, while having a large enough $|f_{R0}|$ to exhibit interesting deviations from GR [44]; if $|f_{R0}|$ were to be to small, the theory would be almost indistinguishable from standard gravity.

Note that F4 is however no longer viable [49], although it remains part of the standard parameter set used by the community.

2.5 Limitations

Beyond creation of the initial conditions, dark matter is the only matter component used; the baryon part is replaced by dark matter (we have neglected neutrinos). This should work well when looking at the large scale structure, as the baryons are generally thought to just trace the dark matter [50] with some linear bias.

The force in the simulation is calculated by assigning the mass of the particles to points on a grid and then calculating the force from the grid. Therefore the more points we have on the grid the higher the accuracy of our force calculations. This inaccuracy changes the position and velocity of the particles in the next step. It has been shown that on scales larger than the mean particle separation the dark matter clustering is reliable (in our simulations we have 512^3 particles and mean separation is $0.5\text{Mpc}/h$) when not limited by the box size [51]. In our simulations it should be even better as our grid is of higher resolution where we have a higher density (more particles).

The size of the simulation box can have significant effects on the large scale matter distribution. This is because the size limits the wavelength of perturbation that we can include information from. Wavelengths longer than the sidelength of the box do not contribute to the simulation and we should be very careful looking at scales above 128 Mpc/h, as per Nyquist sampling one needs at least a sample of two (fit two wavelengths in the box) to be able to recover its properties reliably. Power and Knebe (2006) [52] investigated the effects of limiting the box size, starting from a box with a sidelength of 128 Mpc/h. When reducing the box size, the median halo mass within a mass interval was reduced, but not very significantly, for halos below 10^{13} M_{sun}/h, even when using a boxlength of 32 Mpc/h. Power and Knebe (2006) [52] also found that the halo-halo correlation function did not change much when reducing the box size when looking at very small scales (< 2 Mpc/h in this case) however at 10 Mpc/h the discrepancy is quite significant when going from a 128 Mpc/h sized box to one of 64 Mpc/h. We have a larger box (256 Mpc/h) so our correlation function should be reliable a somewhat larger scales, but we still need to be aware of this as it will affect the absolute values of our velocity statistics who depend on the correlation function (through the power spectrum for the velocity correlation function and directly for the pair-wise streaming velocities). Since we are comparing models it is more important to us the the changes in the large scale matter structure between models and the correction to the correlation function due to the box size should be of less importance. In an ideal case we would test this by seeing of our velocity statistics and differences between models changed when varying the box size.

2.6 Halo finding

When the simulation has run its course to $z = 0$, a halo-finder is used to determine the location of gravitationally bound objects from the position of the particles in the box. There are two main categories of halo-finders: ones locating halos through peaks in the matter density, and ones that link together particles closer to each other than a set linking length. The halo-finder that we used is called ROCKSTAR [40], version 0.99.5 (Beta). ROCKSTAR is part of the latter category of halo-finders which are called "Friends of Friends" (FoF) finders.

The way ROCKSTAR works can be described schematically (similar to Figure 1 in [40]) as follows:

First the simulation box is divided into 3D FoF groups. Then, for each of the groups (in parallel as much we one has CPUs for) [40]:

1. Normalize positions and velocities of the particles by the group position and velocity

dispersions.

2. Select a linking length in phase-space (the 6D space $(\mathbf{x}, \mathbf{p})$ of spatial positions and particle momentum) so that 70% of the particles in the group are linked together in subgroups.

3. Repeat step 1 and 2 until no more substructure can be found.

4. Assign particles to the closest halo (in phase-space) in the last level of substructure that was found in step 3.

5. Remove unbound particles, that is particles with a velocity above the escape velocity.

6. Calculate halo properties (halo mass, position, velocity, etc.).

Halo positions are found by taking the average position of a group of particles in the selected halos to make the expected Poisson error $(\sigma_x/\sqrt{N})$ as small as possible. This is the particle selection that most accurately determines the halo position according to [40]. Traditionally halo finders using the maximum density peak in the halo to determine its position have been more accurate than FoF finders, this is because the particle density decreases quickly going towards the edge of a halo and including those particles then made the position of the halos less accurate. ROCKSTAR method closes this gap by not using most of the particles in a halo as when minimizing the expected poison error only the inner 10^3 particles of the 10^6 particle halo will typically be used.

The velocity of a halo is found by calculating the average velocity of the particles within the central 10% of the halo radius, because the galaxy associated with the halo would track the core of the halo [40]. The difference between this velocity and the bulk velocity (using all the particles in the halo) can be up to a few hundred km/s. Both of them can be useful depending on purpose; the former is more useful to us as we study galaxies as tracers if the velocity field, not the dark matter halos.

Figuring out the uncertainty of the halos positions and velocities due the halofinder's ability to recover them is done by measuring them in one time step using the halo-finder, then predicting what the positions and velocities should be in the next time step based on the current position and velocity. This is then compared to what the halo-finder recovers. In the ROCKSTAR paper [40], they performed this procedure and the errors for both the velocities and positions are shown in their Figure 2.1 as a function of halo mass compared to some other finders. We can see that ROCKSTAR resolves velocities extremely well compared to the other finders except for haloes of masses lower than 10^{10} M_{sun}, which we will not use. Uncertainties in halo positions are on the order of a few kpc, which is very small compared to the scales we are interested in. In light of this the uncertainty in the recovery of the halo positions and velocities should be of little importance.

Finally, note that some finders include the masses of subhalos in the mass of the parent halos. This is not done in ROCKSTAR as each particle is only assigned to halos in the lowest level of substructure.

We will look at haloes above 10^{12} M_{sun}, halos with above 100 particles, in order to get trustworthy halo positions and velocities [40, 53], while not knowing the particle size used in Figure 2.1.

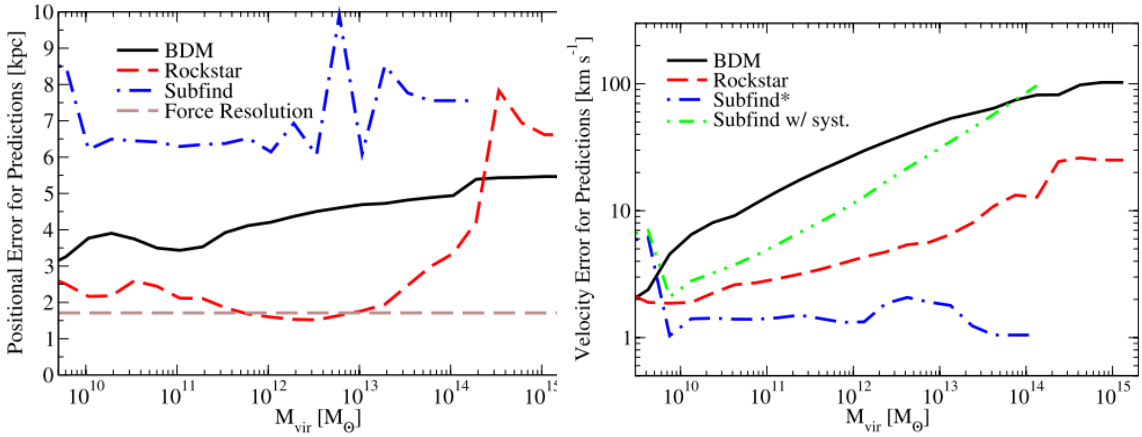


Figure 2.1: Figure 7, taken from [40].

Chapter 3

Method

In this chapter we describe how we calculated the velocity statistics.

3.1 Calculating from simulation data

In this section we will describe how we calculate the velocity correlation function and the pairwise streaming velocities described in Sections 1.4.2 and 1.4.3 using the halo catalogues resulting from the simulations described in Chapter 2.

3.1.1 Correcting for observer position

We want to create a mock observed galaxy catalogue in order be able to make direct comparisons with real observations.

The n-body simulation uses a cube with periodic boundary conditions to simulate the Universe' much large size than the box size. That means that when light and other particles leave the cube through one side of the cube it comes back into the cube from the opposite side instantaneously. As a result, outside our cube there are in effect mirror cubes consisting of halos with mirrored positions. If we place an observer at some point in our cube, often one of the mirrors will be closer to the observer than the halo in the original box. If we limit the distance from the observer that we take data from to be half of a cube sidelength, there will always be only one copy of each halo within this range.

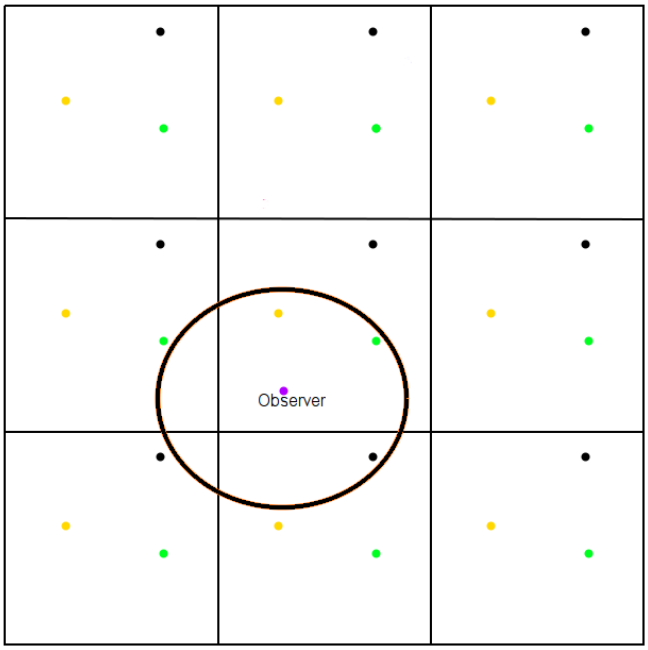


Figure 3.1: A 2D illustration of the box we simulate, with the neighbouring periodic boxes. A circle of radii L_{box} is drawn around the observer with only one copy of each halo within it.

To avoid using the same halo more than once we always choose the copy that is within this range. This can be done in the following way.

For each dimension, if the distance in that dimension from the observer to the original halo is more than half of the cube's sidelength, we adjust that distance in that dimension by subtracting half the sidelength and get the distance to the closest copy.

The velocity of a copy is exactly the same as that of the original in an absolute sense; what we are interested in is however the velocity relative to the observer. When we adjust the position of a halo in a spatial dimension, we also change the sign of its velocity in each dimension and then we can calculate the radial peculiar velocity.

3.1.2 Estimators

An estimator is a formula for estimating the value of the statistic of an underlying distribution based on limited observations. A good estimator should have as low a variance as possible and have as little bias as possible, and unbiased estimator is one who's expected value is the same as the expected value of the statistic it is estimating for the underlying distribution.

3.1.2.1 Velocity correlation function

We were unable to find an appropriate estimator for the velocity correlation function so we used the naive estimator,

$$\xi_v(r) = \frac{\sum_{\text{all pairs with separation } r} v_i v_j}{N_{\text{pairs}}(r)} \tag{3.1}$$

3.1.2.2 Streaming velocities

The pair-wise streaming velocities like we described in Section 1.4.3 uses the relative velocity along the line of sight between each of the galaxies. This is not a problem when using data from the simulations, however in the real universe we can only measure the radial peculiar velocity with any accuracy. To estimate the streaming velocity we can use an estimator [36]:

$$\tilde{v}_{12}(r) = \frac{2\sum(s_A - s_B)p_{AB}}{\sum p_{AB}^2} \tag{3.2}$$

where $s_A = \hat{\mathbf{r}}_A \cdot \mathbf{v}_A$ is the line of sight peculiar velocity part of object A and $p_{AB} \equiv \hat{\mathbf{r}} \cdot (\hat{\mathbf{r}}_A + \hat{\mathbf{r}}_B)$ ($\mathbf{r} = \mathbf{r}_A - \mathbf{r}_B$) the meaning of the different quantities are illustrated in Figure 3.2. The summing signs ($\sum$) indicate that we sum over all pairs with separation r.

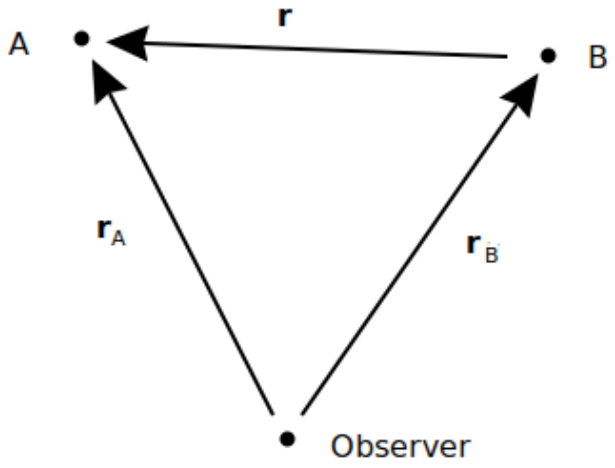


Figure 3.2: An illustration of the geometry for equation 3.2.

This comes from the idea that $\langle s_A - s_B \rangle = v_{12} p_{AB}/2$ and equation 3.2 makes that square difference $\xi^2(r) = \sum_{A,B}[(s_A - s_B) - p_{AB}\tilde{v}_{12}(r)/2]^2$ [36].

The estimator breaks down for very high angle halo pairs [54]; for example, if the angle between a pair of halos is 180 degrees, the p_{AB} becomes zero and so that pair does not contribute to $\tilde{v}_{12}(r)$. So preferably we should have a p_{AB} that also projects correctly for wider angles [54], but as not make too large of a difference when we do not include high-angle pairs as those are effectively down-weighted, thus lowering our effective sample size. In the future one would want to find a projector p_{AB} that holds for larger angles, particularly when using data from observations as the amount of data is then likely to be much more limited.

An alternative choice of estimator would have been [54]

$$v_{12}(r) = \frac{1}{N_{\text{pairs}}(r)} \sum_{i \neq j} (\mathbf{v}_i - \mathbf{v}_j) \cdot \hat{\mathbf{r}} \tag{3.3}$$

where $N_{pairs}(r)$ is the number of pairs in range-bin r and $\hat{\mathbf{r}} = \mathbf{r}/|\mathbf{r}|$ is the unit vector pointing from halo j to halo i.

as then our v_{12} is more accurate as we avoid that wide-angle problem and use the full three dimensional information. However using the estimator (equation 3.2) we can run simulation data and redshift survey data through the same code.

3.1.3 Algorithm

This sub-section describes how the velocity statistics are estimate using the estimators described in sub-section 3.1.2.

At first we need to extract our halo sample and select the right version of each halo as described in section 3.1:

1. Read in data and remove halos outside the selected mass range.
2. Give each halo an ID number.
3. Calculate each halo's position relative the to observer and adjust it and its velocity to match the closest copy to the observer.
4. Calculate the peculiar radial velocity $\hat{\mathbf{x}} \cdot \mathbf{v}$ of each halo relative to the observer and save it in memory.

In order to calculate error bars (see Section 3.1.4), we will need to recalculate the statistic many time, in order to save CPU time it is the useful to save pair-wise calculated values in upper triangle matrices so that when we need to recalculate it we can just look it up in memory instead:

5. For each pair of halos, we then calculate the velocity statistic like so:
   (a) Calculate the distance of separation between halos r and assign the halo-pair to correct range bin.
   (b) Save pair i, j to element $j - i$ in array i with the array of arrays holding the data.

If calculating the velocity correlation function, we save $v_1 v_2$ for each pair in an upper triangle matrix, while when calculating streaming velocities v_{12} (equation 3.2) we save $p_{i,j}$. This is so that when we calculate the error bars later on we do not have to do the same calculations over and over again, but can rather just look the numbers up in a lookup table and thus save a lot of cpu time.

3.1.4 Constructing the errorbars

3.1.4.1 Absolute error

Since the simulation box is of limited size, we also have a finite sample of halos. We need to determine the uncertainty of our velocity statistics due to the limited halo sample. Ideally, if time allowed, we would run lots of simulations with new random seeds, we would then also account for variance in possible initial conditions, but in absense of the time we have to use other methods.

Two common methods are the jackknife and bootstrap methods. Both methods use the data sample to simulate the distribution of data we are missing out (the true "ensemble" distribution) on due to our limited sample size.

In the jackknife method, data points are left out and a new value y_{Ji} is calculated with the estimator so that y_{Ji} is the estimation with halo number i left out of the sample. The variance of the estimator is then:

Algorithm 1 Basic algorithm for calculating the statistics from a halo sample.

```
for i from 0 to N do
  for j from 0 to i − 1 do
    if Halo ID is the same then
      Do nothing.
    else
      bin = Look up range bin.
      if statistic == streaming velocities then
        Numerator_bin = Numerator_bin + 2(v_i − v_j)p_{i,j}
        Denominator_bin = Denominator_bin + p²_{i,j}
      end if
      if statistic == velocity correlation function then
        Numerator_bin = Numerator_bin + v_i v_j
        Denominator_bin = Denominator_bin + 1
      end if
    end if
  end for
end for
for bin from 0 to Nbins − 1 do
  stat_bin = Numerator_bin/Denominator_bin
end for
```

$$\sigma^2_{J\text{mean}} = \frac{(N_{\text{halos}} - 1)}{N_{\text{halos}}} \sum_{i=1}^{N_{\text{halos}}} (y_{Ji} - \bar{y})^2 \tag{3.4}$$

where $\bar{y}$ is the estimation using all halos.

Since calculating the estimator scales in floating points operation as $O(N^2_{\text{halos}})$ as we loop over pairs and in the Jackknife we also need to loop over all halos again, the Jackknife method would go like $O(N^3_{\text{halos}})$.

In the bootstrap method, if our sample size is N we sample N values from our sample with replacement, that means that we pick a random halo from our sample N times to construct a new sample of the same size as the original sample allowing for a halo to be picked more than once. From this new sample we can calculate the statistic we want to find the variance of again. The idea is that if you use the distribution of the original data as a fair representation of the true underlying distribution, one can see how the velocity statistic varies as a function of the random variation in the data. This means that the bootstrap scales in floating point operation as $O(N^2)$, which is why we ended up picking this method.

When choosing the number of resamples it is a tradeoff between spending more CPU time to get more accurate errorbars, or get less accurate errors spending less CPU time. The choice was made to do a thousand resamples. Once all of the new values have been calculated the central 68% of values are within a one σ deviation from the true value, and the central 68% are not necessarily centered on the value of the estimator using the original sample, which means we can get asymmetric error bars.

3.1.4.2 Error propagation

The asymmetric errorbars we got from the bootstrap proved to be quite symmetric when calculating the skewness of $\tilde{v}_{12}$ in each rang bin (see Figure 3.3), so when propagating the errors

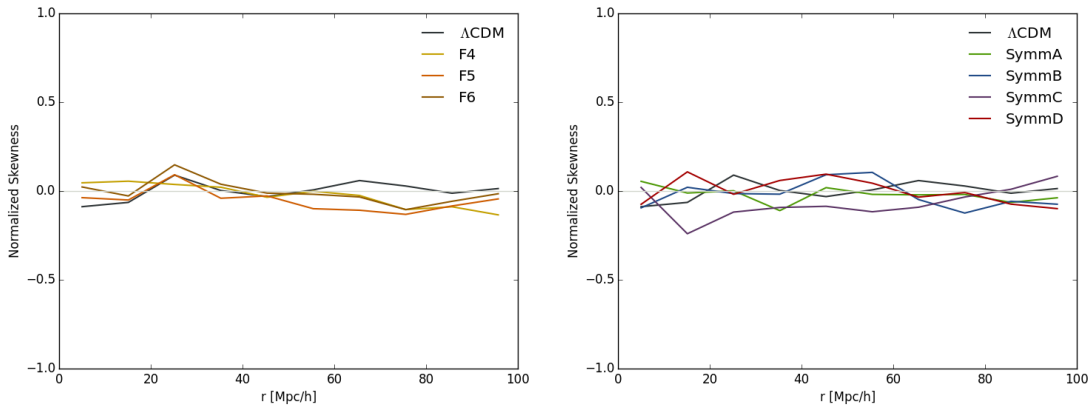


Figure 3.3: Skewness of the $\tilde{v}_{12}$ distribution given by 1000 random halo resamples with replacement. 10 Mpc/h bins are used.

in the calculation of relative deviations from standard gravity (ΛCDM) we assumed symmetric (Gaussian) errors.

To find the errors in the relative deviation from ΛCDM we use the errors from ΛCDM and the alternative theory we are comparing with.

The relative deviation from ΛCDM is expressed as

$$\Delta_y = \frac{y_{\Lambda\mathrm{CDM}} - y_{\mathrm{MG}}}{y_{\Lambda\mathrm{CDM}}}, \tag{3.5}$$

where $y(r)$ is a binned velocity statistic.

To find the error of d we need to propagate the errors of the individual components.

Combining the errors for y in quadrature we obtain

$$\sigma_{\Delta_y} \approx |d| \sqrt{\left(\frac{\sqrt{\sigma^2_{\Lambda\mathrm{CDM}} + \sigma^2_{\mathrm{MG}} - 2\sigma_{\Lambda\mathrm{CDM,MG}}}}{y_{\Lambda\mathrm{CDM}} - y_{\mathrm{MG}}} \right)^2 + \left(\frac{\sigma_{\Lambda\mathrm{CDM}}}{y_{\Lambda\mathrm{CDM}}} \right)^2 - 2\frac{\sigma_{\mathrm{diff,\ \Lambda CDM}}}{y_{\mathrm{diff}}/y_{\Lambda\mathrm{CDM}}}} \tag{3.6}$$

where $y_{\mathrm{diff}} = y_{\Lambda\mathrm{CDM}} - y_{\mathrm{MG}}$, $\sigma_{\Lambda\mathrm{CDM,MG}}$ is the covariance between $y_{\Lambda\mathrm{CDM}}$ and y_{MG}, and $\sigma_{\mathrm{diff,\ \Lambda CDM}}$ is the covariance between y_{diff} and $y_{\Lambda\mathrm{CDM}}$.

There is one problem with assigning errorbars to the relative deviations from ΛCDM, the covariance between models. In Section 2.2 we stated that the initial conditions for each model were exactly the same, so the covariance between models is likely to be very large and we have ignored it, likely leading to larger errorbars than we potentially should have.

3.1.5 Random poisson catalogues

In the streaming velocity plots (Figures 4.1 and 4.2) with 2 Mpc/h bins, there are two curious features:

1. They do not vary as much as the errorbars would suggest, so it looks like there is significant covariance between bins.

2. The graph is "step-like" on scales less than 40 Mpc/h, having range intervals with a constant slope, but with a sudden change in slope in between.

To test whether this was a property of the estimator, or could be due to a programming error instead, random Poisson (uncorrelated) catalogues with different seeds were constructed.

A number of halos were placed with a uniform probability between 0 and L_{box} in each dimension, making any spot in the box equally probable. They were assigned velocities in each dimension using a Gaussian distribution with standard deviations of 200 km/s. If everything is working we would expect to se $\tilde{v}_{12}$ vary around zero. The results for these random catalogues can be found in Figure 3.4, and they do indeed vary around zero.

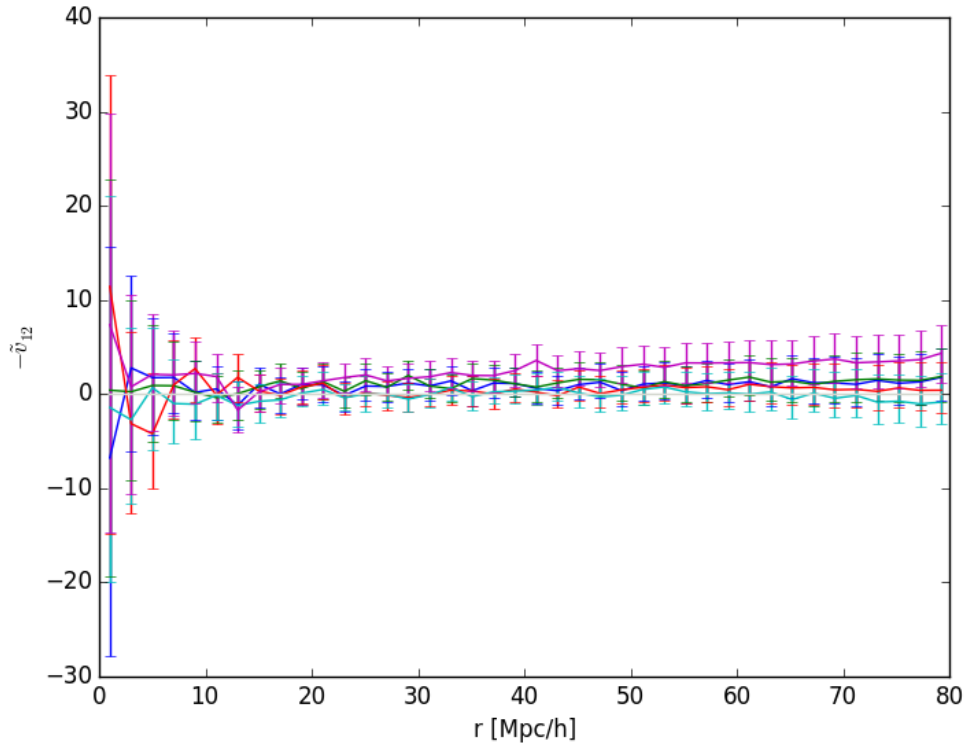


Figure 3.4: $\tilde{v}_{12}$ plotted for random Poisson catalogues (Section 3.1.5). For $\sigma_v = 200$ km/s, where each line is for a different seed. The asymmetric errorbars are determined by the bootstrap method described in Secton 3.1.4.1.

3.2 Validating the results

To reliably be able to base any conclusions from our calculations it is important to check that it is working as intended.

Both the velocity correlation function and the streaming velocities can be approximated from linear theory.

For both approximations I used the matter density power spectrum generated when supplying CAMB [55] with a parameter file holding our cosmology, and solving all of the integrals

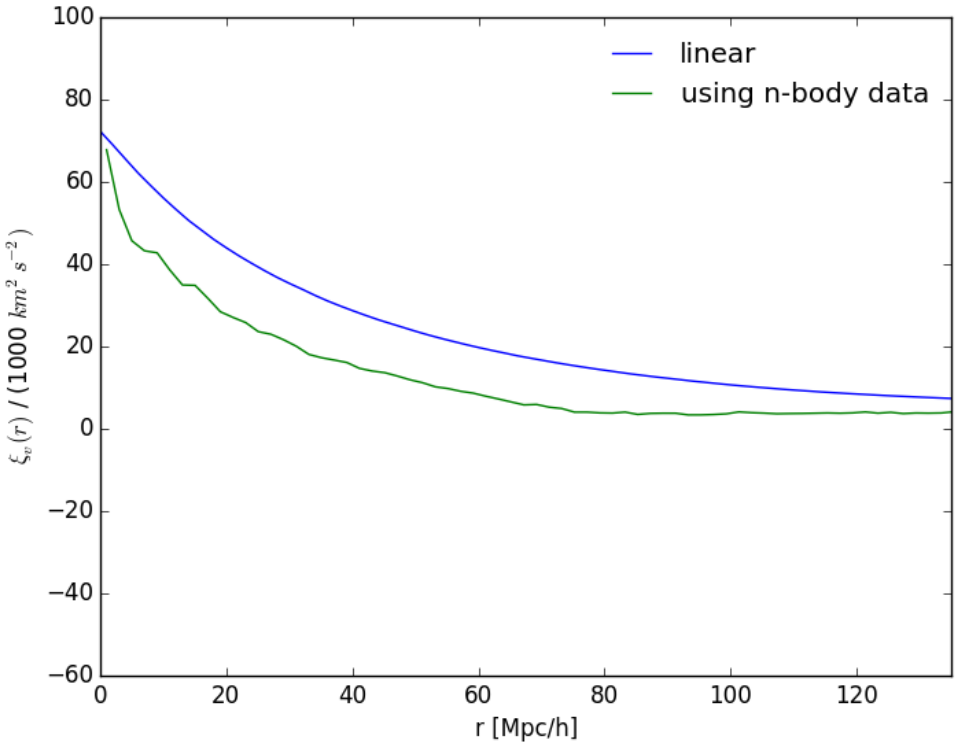


Figure 3.5: Plot showing the linear approximation to the velocity correlation function and the one calculated from the ΛCDM n-body data described in Chapter 2

using the integration function QAG in the GNU Scientific Library [56], Integrating between $k_{min} = 10^{-6}$ h/Mpc and $k_{max} = 31100$ h/Mpc (we are aware that this is an unusual and unnecessarily wide k range). All integrals over k were shown to be stable (within numerical accuracy) when shrinking the integration interval from both sides.

The linear approximations to the parallel and perpendicular components of the velocity correlation function were given in equations 1.83 we made an approximation to $\xi_v(r)$. As can be seen the in Figure 3.5 our approximations do not match what we get from the simulation of ΛCDM .

An analytical model for v_{12} was found in [57], which is valid on quasi-linear scales [36]:

$$v_{12}(x,a) = -\frac{2}{3}Hrf\bar{\bar{\xi}}(x,a)[1+\alpha\bar{\bar{\xi}}(x,a)] \tag{3.7}$$

$$\bar{\bar{\xi}}(x,a) \equiv \bar{\xi}(r,a)[1+\xi(r,a)]^{-1} \tag{3.8}$$

$$\bar{\xi}(r) = \frac{3}{r^3}\int_0^r \xi(x)x^2\mathrm{d}x \tag{3.9}$$

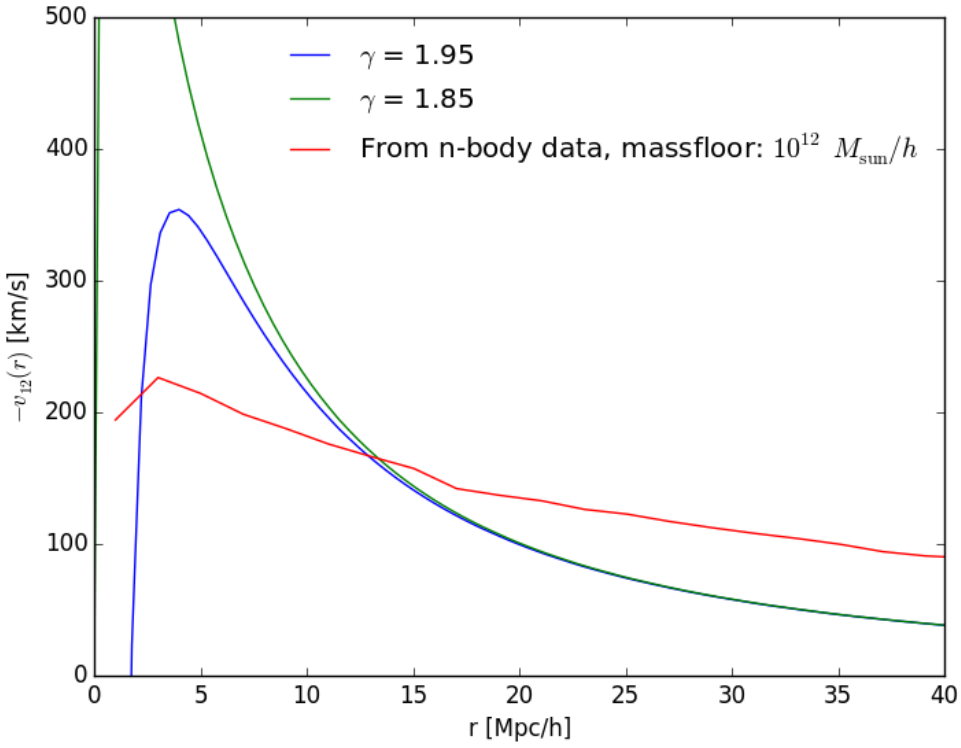


Figure 3.6: v_{12} approximation made using equation 3.7 plotted with the results of using the estimator for $\tilde{v}_{12}$ on the ΛCDM n-body data.

$$\xi(r) = \int \frac{\mathrm{d}k}{k} \frac{k^3 P(k)}{2\pi^2} j_0(kr) \tag{3.10}$$

where $\xi(x, a)$ is the galaxy/halo two-point correlation function at distance scale x when the scale factor is a, and $\alpha = 1.2 - 0.65\gamma$ where γ is the logarithmic slope of ξ at $\xi = 1$.

Juszkiewicz, Springel & Durrer (1998) [57] also suggests that one can use the galaxy correlation function to calculate predict v_{12}, one then just use that function in place of the dark matter correlation function [58] in the approximation. An attempt was made at calculating the halo-halo correlation function using the Landay Szalay estimator [30], but there was problems finding halo-halo correlation functions in the literature where the halo mass intervals used were also stated. In order to verify the halo-halo correlation function we tried to calculate what the ratio between correlation functions using different mass ranges should be and compare this to what we got. This was ultimately unsuccessful, however.

All of the approximations in this section (3.2), the bias calculations and the halo-halo correlation function have one thing in two things in common: they all include integrals over the power spectrum and neither of them work. Therefore one could suspect that something is wrong with the power spectrum that either we use in the approximation, or the one used in the initial conditions of the simulations (these were not found). There was made a custom halo catalogue

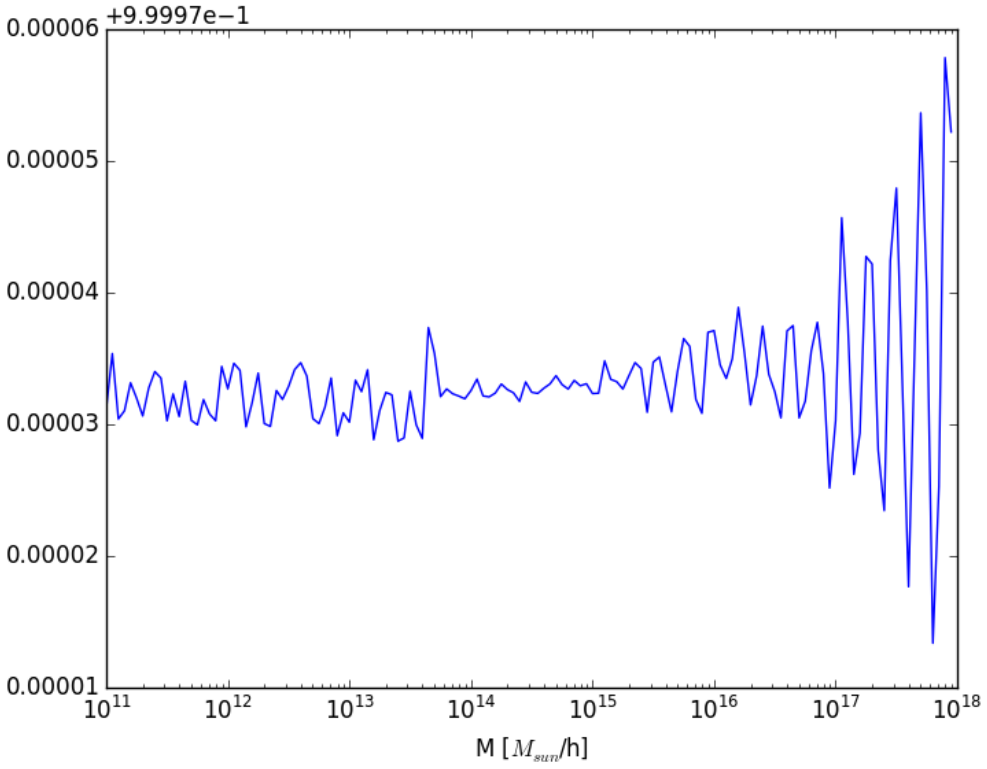


Figure 3.7: The ratio between $\sigma(M) = \int_0^\infty k^2 P(k) W^2(kR) \mathrm{d}k$ calculated using $P(k)$ from CAMB in my halo bias calculation code and what was gotten using the code presented in [59]. We use $M = 4\pi\rho_m R^3/3$ where ρ_m is the matter density of the Universe.

with 4 halos in and both the halo velocity correlation function and the streaming velocities were calculated by hand and shown to match what our code got on the same catalogue.

To summarize, we should be careful concluding anything from the results we get as none of them have been satisfactorily. It is however more likely that the calculation done from the n-body simulation data is correct than the approximations. We must also be aware that there could be something off with the initial powerspectrum in the simulations.

Chapter 4

Results and analysis

In this chapter we present and analyse the velocity correlation functions and pair-wise streaming velocities calculated using the methods described in Section 3.1 and data from the N-body simulations described in Chapter 2.

4.1 Streaming velocities

In this section we will present streaming velocities for the different $f(R)$ and symmetron models and compare them to ΛCDM. Methods for calculating these values can be found in Chapter 3.

The velocity correlation function for $f(R)$ and ΛCDM models is shown in Figure 4.1 for 2 Mpc/h bins. It is important to note that the y-axis is $-\tilde{v}_{12}$, meaning that the larger its value, the more rapidly the halos are *approaching* each other.

$\tilde{v}_{12}$ for all models varies a lot less between the bins than the error bars suggest it should; the scatter in the measured curve is much smaller. This suggests either that there is covariance between the bins, meaning that they vary together, or that the bootstrap errors are incorrect. This is expected, as halos close to each other will be gravitationally attracted by approximately the same set of halos, just with a slightly different force strength. It is possible to predict the covariance using the velocity correlation tensor [36].

For all the models (Figures 4.1 and 4.2) there are "step-like" features; that is, the graph looks almost linear between the second bin ($[2, 4)$ Mpc/h), then suddenly changes gradient at around 18 Mpc/h, remaining linear until around 40 Mpc/h. A test was performed calculating $\tilde{v}_{12}$ for for random Poisson catalogues (Section 3.1.5) to see if these features are real. The features were not seen for these catalogues (Figure 3.4): one possibility is that this is a function of our initial conditions, that if we had another seed we would not see the features. This is easy to test we just need to run the simulation for different seeds and see if the features appear in those as well. Another option could be that the feature appear because there are correlations between bins, but then we would expect not as sudden of a transition between "steps".

$-v_{12}$ peaks for all the models in the second bin; that is, in the range $[2, 4)$ Mpc/h. Going from the peak to $r = 0$, $-v_{12}$ approaches zero. This matches what is found in the literature [36]. If two halos are at almost the same location they are almost as likely to have just passed each other as they are to be approaching each other. At scales where this effect is not dominant galaxies tend to approach each other faster the closer they are to each other as overdensities attract one another [28].

$\tilde{v}_{12}$ flattens out at large r above zero, in theory there should be no correlation approaching homogeneous scales as then there should be on average equally many halos pulling from each

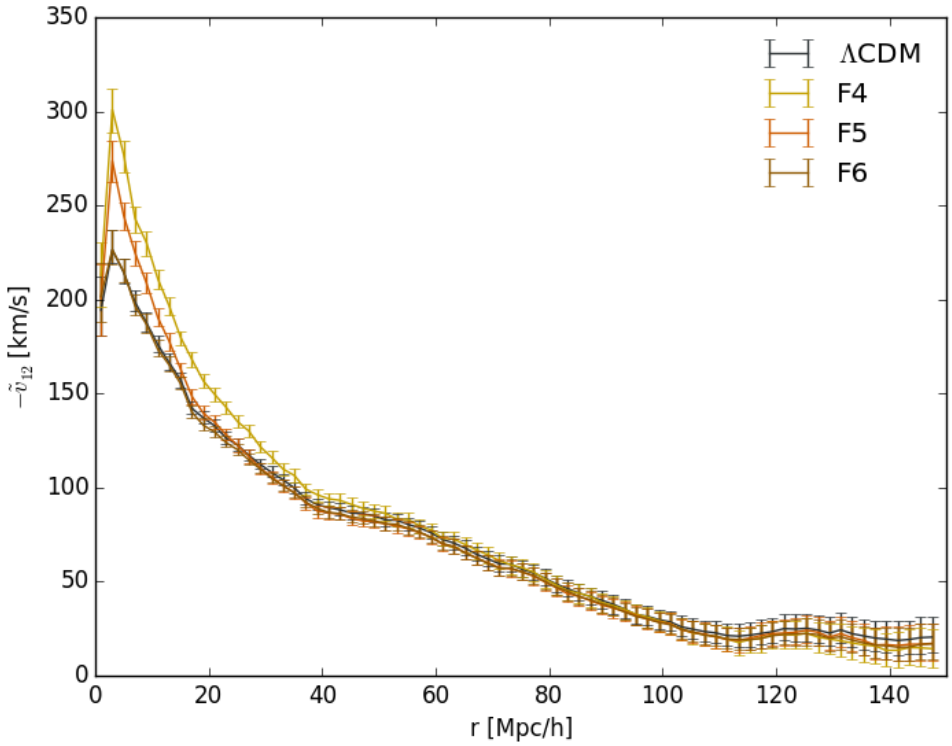


Figure 4.1: $\tilde{v}_{12}$ plotted for the chameleon $f(R)$ theories and ΛCDM, with 2 Mpc/h bins. The asymmetric errorbars are determined by the bootstrap method described in Section 3.1.4.

side.

There is a bump between around 118 Mpc/h and 135 Mpc/h, this is around the same scale at which Barionic Accoustic Oscillations (BAO) causes a bump in the halo and dark matter correlation functions [60].

The error bars increase in size the larger in separation we go to, even though at the larger scales there are more halo pairs, so ordinarily one would think that the error bars would become smaller. However at larger scales the angular separation of the halos in a pair is likely to be large and the pair is therefore downweighted (small projection factor p_{ij}) thus the "effective sample size" is smaller than otherwise.

4.1.1 Comparison between models: f(R)

For the $f(R)$ models one would expect the peak to be the highest for F4, followed by F5, F6 and then ΛCDM, as when the range of the fifth force is longer the fifth force between particles have been working to grow the deviation from ΛCDM for a longer time. In our results F6 has a peak almost identical to (a tiny bit lower than) ΛCDM, while F5 peaks at 47 km/s higher than F6, and F4 around 74 km/s higher than ΛCDM. This is all as expected, other than F6 peaking lower than ΛCDM. We cannot tell if the peaks are at the exact same place, only that they are in the same bin. If we lower the bin size, we could possibly resolve differences in peak locations,

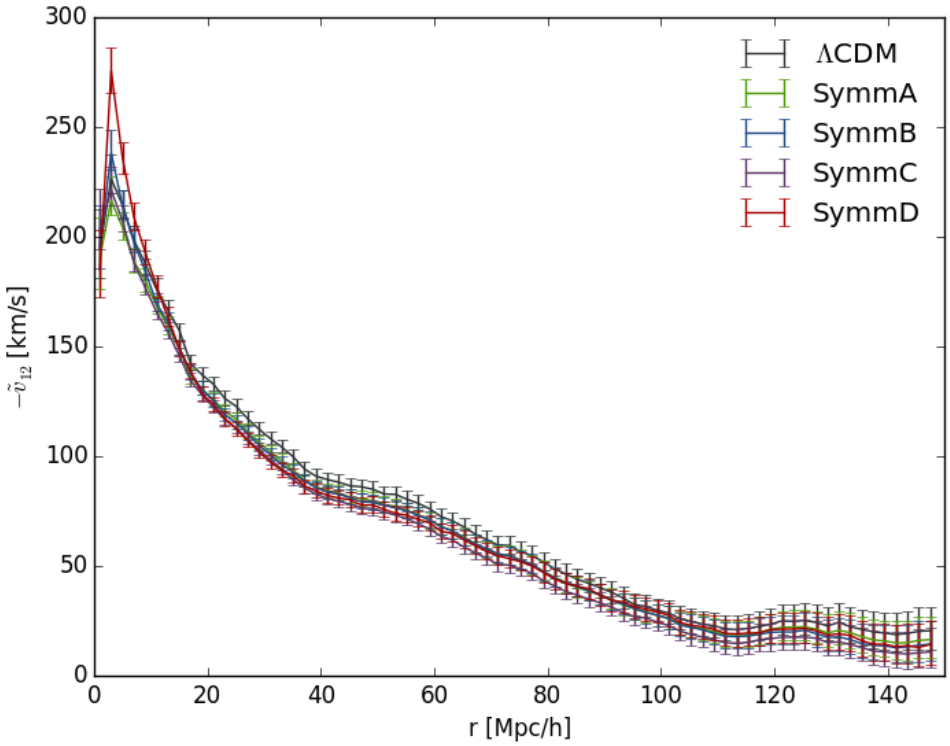
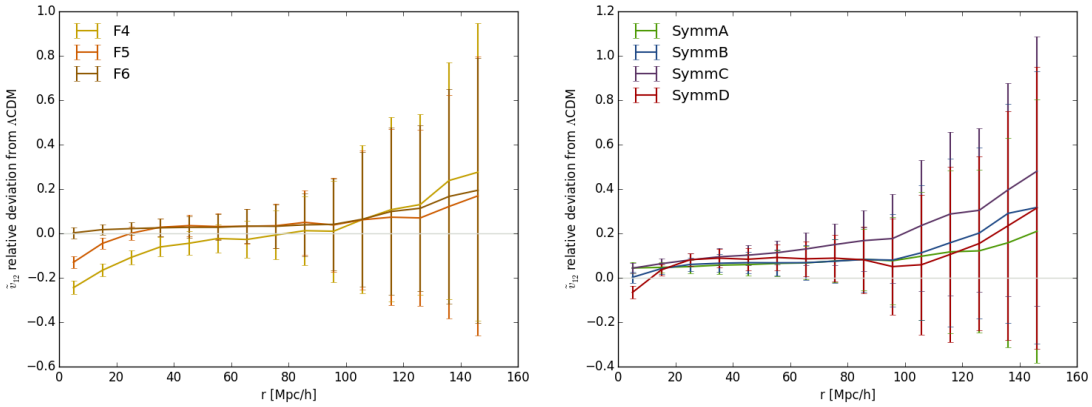


Figure 4.2: $\tilde{v}_{12}$ plotted for the symmetron theories and ΛCDM, with 2 Mpc/h bins. The asymmetric errorbars are determined by the bootstrap method described in Section 3.1.4.

but this would reduce the number of pairs in each bin and so the variance increases. This is the error on $-\tilde{v}_{12}$ however, and since all of the models have the exact same initial conditions, the difference should be real. We are however limited by the force resolution of the simulation — an inaccurate force calculation affects the position and velocity of a particle for all future in the simulation. It can however both overestimate and underestimate the force.

All models decline when we go to larger scales after the peak, F5 deviating noticeably from ΛCDM at ranges up until around 20 Mpc/h, and F4 almost up to 40 Mpc/h. From table 2.2 we have that the range of the force for F5 and F4 are 7.5 Mpc/h and 23.7 Mpc/h, so we have deviations on length scales significantly larger than the force length. This is likely because when the universe was smaller (a was smaller) what corresponds to a length scale of 40 Mpc/h today was smaller. For example, a length scale $r = 10$ Mpc/h today ($z = 0$) corresponds to $r = 5$ Mpc/h at redshift $z = 1$. Another contributing factor to the deviations on such large scales is the covariance between bins. If the fifth force works between two halos separated by a small enough range for the fifth force to work, then they approach each other faster than they otherwise would. A halo approaching both halos, but far enough away for only the fifth force from one of the halos to work will be affected by the fifth force of the halo that does not work on it directly, by that it changes the velocity and position of the halo which is within the fifth force range.

The relative deviations (Figure 4.3a) of the different $f(R)$ theories approach each other, the



(a) (b)

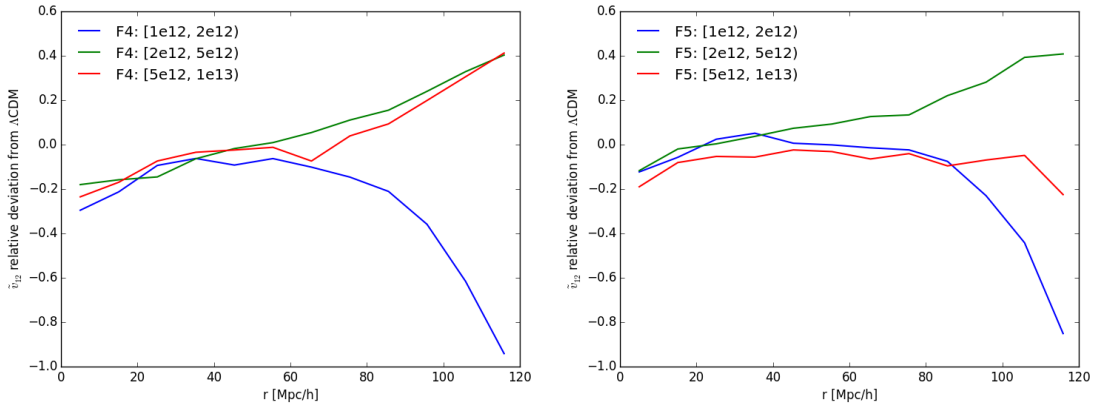
Figure 4.3: $\tilde{v}_{12}$ relative deviation from ΛCDM plotted for all of f(R) theories (left) and symmetron theories (right) with 10 Mpc/h sized bins. The asymmetric symmetric errors are determined by the bootstrap method described in Section 3.1.4, and then propagated following the method in Section 3.1.4.2 assuming no covariance between ΛCDM and the alternative theory.

largest deviations being at small r, systematically as r increases up until 110 Mpc/h. Beyond 110 Mpc/h, the deviations of all the $f(R)$ models increases, F4 increasing the most, then F5 and F6. This is at a scale were the fifth force could not have been working for a significant amount of time. One of the factors that can cause effects on this scale is the box size, but since this happens at a scale beginning at less than half the box size, and we are working with a relative deviation, the difference when we change theories is unlikely to be an effect of much significance. One reason could be that the mass function is distribution of halos with respect to mass is different for the different theories, it is for example known from the literature that changing the minimum mass of the halos used changes the amplitude of the large scale $\tilde{v}_{12}$ [61], this will however require more careful investigation.

4.1.2 Comparison between models: Symmetron

Figure 4.2 shows $-\tilde{v}_{12}$ as a function of the length scale for the symmetron models and ΛCDM.

The symmetron models, like the $f(R)$ models, peak in the second bin. The two highest peaks are for SymmD and SymmB, where SymmB has the lower one. These are the models for which the fifth force has been "turned on" the longest, with SymmD's fifth force turning on at $z = 3$ and SymmB's at $z = 2$ (from Table 2.1), compared with $z = 1$ for SymmA and SymmC. The next peak in height is actually ΛCDM, above SymmC and SymmA in descending order. It makes sense that SymmC has a higher peak than SymmA, as the fifth force in SymmC is stronger, but why are they both lower than ΛCDM? Looking at Figures 4.3a and 4.3b, we see that at larger scales, halos do not tend to approach each other (in terms of comoving distances) as much in the modified gravity theories as for standard gravity (ΛCDM). It could be that what causes this large scales suppression of $\tilde{v}_{12}$ also works on small scales and that the fifth forces of SymmA and SymmC is not a large enough effect to make them peak higher than ΛCDM. The approximation for v_{12} given in equation 3.7 depends on three functions: $H(a)$, $f(a)$ and $\xi(x, a)$. For $a = 1$, which is the time our halo catalogues are for, H is equal for all models in



(a) (b)

Figure 4.4: $\tilde{v}_{12}(r)$ relative deviation from ΛCDM plotted for F4 (a) and F5 (b) with different halo mass ranges (given in units of $M_{\rm sun}/h$ in the legend) and 10 Mpc/h sized bins.

order to match the input value of H_0. The growth rate $f = \mathrm{d}\ln D/\mathrm{d}\ln a$ (see Section 1.1.3) should be larger for both symmetron and $f(R)$ models [17] and thus contributes to a large v_{12}, not smaller. The last factor is a function of the two-point spatial correlation function $\xi(x, a)$. This is a little steeper in modified gravity, as there is extra clustering due to the fifth force [46], but more investigation needs to be done as to how $\bar{\bar{\xi}}(x, a)[1 + \alpha\bar{\bar{\xi}}(x, a)]$ is affected.

4.1.3 Dependence on mass and binning

Here we calculated the streaming velocities for different halo mass ranges. More massive halos are screened for fifth force effects to a larger extent than lower mass halos overall, although the density in the halo environment also impacts the degree of screening significantly [24]. If a halo is located in a high density environment, then the halo does not have to be as massive for the halo to be screened.

In using different mass bins, we would expect the deviations from ΛCDM to be larger for lower masses, particularly in low density environments. Using the same mass bins, but only for halo pairs in high density environments, we should see almost no deviation from ΛCDM however.

To choose the mass bins, we had to compromise between having more mass bins with smaller halo samples, or fewer mass bins with larger samples. Due to the steepness of the mass function (the cumulative mass distribution), we could make the lower massbins much smaller than the higher mass ones. If too low of a samplesize was used, the errorbars would be very large, but the halos also need to be massive enough that their position and velocity can be resolved with good enough accuracy. As we explained in Section 2.6, we use halos more massive than $10^{12}\ M_\odot/h$ (more than a hundred particles).

In Figure 4.4b we see that the deviation for F5 from ΛCDM is strongest for the most massive halos. We would have expected the lowest mass bin to deviate the most at this range. In F4 however it does deviate the most for the lowest mass bin, but the middle mass bin deviates less than the most massive halos. [62] showed very little mass dependence for F4 and F5 at 5 Mpc/h; ideally we would run simulations using many different seeds in the initial conditions, it is likely that these differences at larger scales would then be shown to be a feature of high variance. This

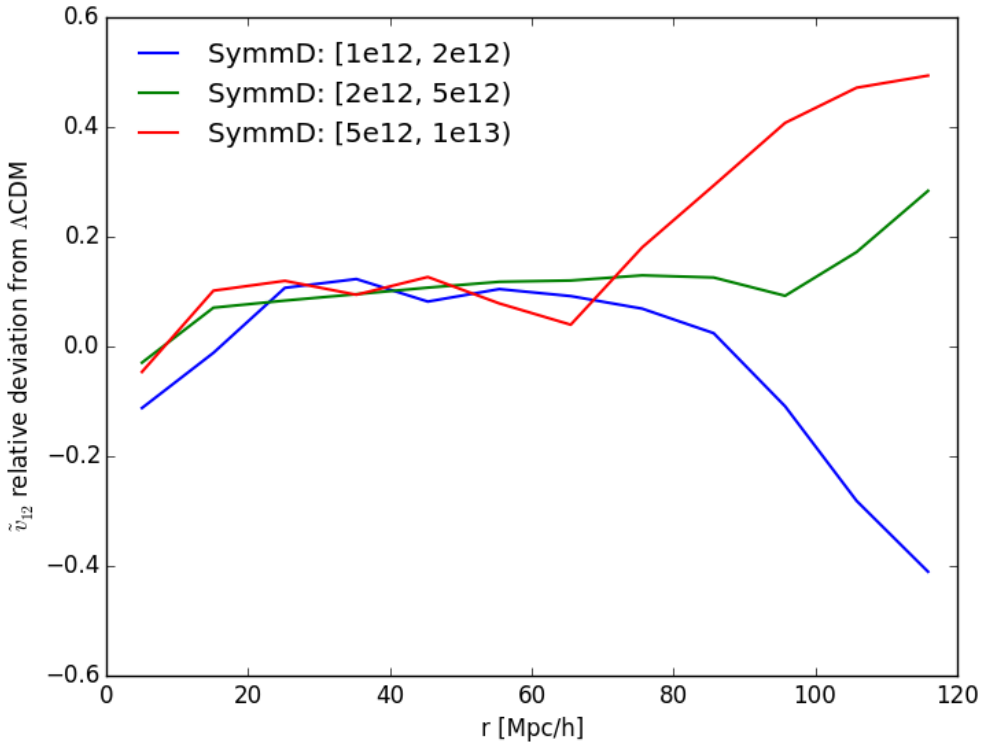


Figure 4.5: $\tilde{v}_{12}$ relative deviation from ΛCDM plotted for SymmD with different halo mass ranges (given in units of M_{sun}/h in the legend) and 10 Mpc/h sized bins.

could be caused by a systematic effect in the environment of different halos, though normally more massive halos are in denser environments. From the bin centered on 75 Mpc/h the lower massbin of F4 and F5 deviates more and more from ΛCDM, at the 115 Mpc/h bin scale halos approach each other almost twice as fast as for ΛCDM. For F4 the two more massive mass bins approach each other, but deviate in the positive direction. For F5 only the middle mass bin deviate in the positive direction at large scales. There is no particularly good reason for these deviations, once again it would be very useful to investigate further by running simulations using different seeds to see if these features are not just due to variance.

In Figure 4.5 we see the relative deviation from ΛCDM for SymmD. Here, the least massive halos have the largest deviations at small range as we would expect, but the same trend as we saw for F5 is here too at scales large than 75 Mpc/h.

4.1.4 Cumulative streaming velocity distribution

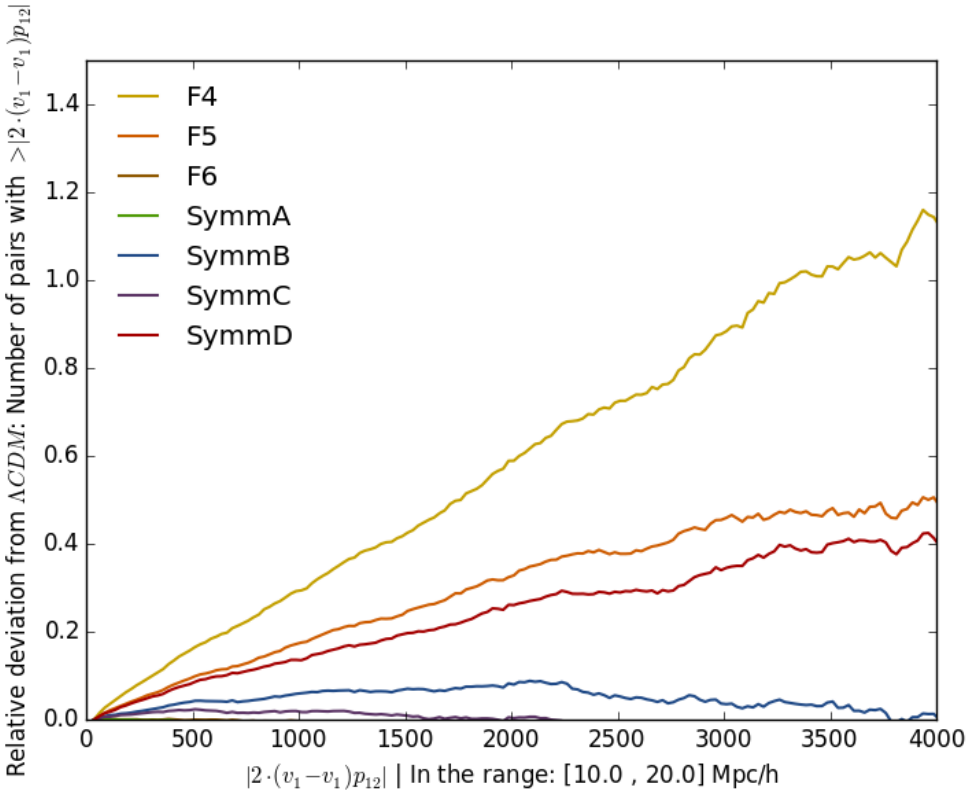


Figure 4.6: Relative deviations from ΛCDM for the cumulative streaming velocity distributions. The distributions have been normalized by the pair count in the range bin.

Figure 4.6 shows the cumulative streaming velocity for the range between 10 and 20 Mpc/h. It is worth noting that the quantity in the plot is a little bit different than the individual pairs' contribution to the streaming velocity, as the estimator (equation 3.2) consists of a sum in the denominator and a sum in the numerator. The quantity we have used here is the one each pair contributes to the numerator, the line of sight velocity difference.

As expected, the difference is the largest for F4 like in the other plots of streaming velocities. Both F4 and F5 deviate more and more from ΛCDM as we move to higher velocities. This could be because lower mass halos generally move faster than more massive ones as gravity is stronger for them (enhanced by fifth force as they are less likely to be screened).

F6 traces ΛCDM extremely well and so does SymmA, as we would expect. F5 and SymmD have relatively similar deviations from ΛCDM, F5 deviating a little bit more than SymmD does. These two models also have quite similar peak heights in $-\tilde{v}_{12}$, with -273 km/s for F5 and -275 km/s. Similarly F4 has the highest peak and also the largest deviation from ΛCDM , while SymmB has the fourth highest peak and also the fourth largest deviation from ΛCDM in the cumulative streaming velocity distribution.

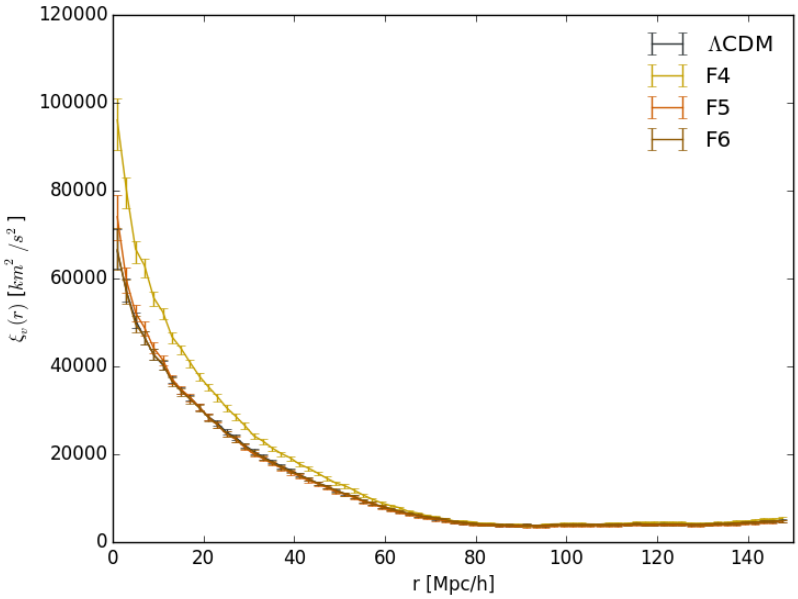


Figure 4.7: $\xi_v(r)$ plotted for the chameleon theories and ΛCDM with 10 Mpc/h sized bins. The asymmetric errorbars are determined by the bootstrap method describe in Section 3.1.4.

4.2 Velocity correlation function

In this section we show how the velocity correlation function described in Section 1.4.2 is affected by using different MGR models. We also show how this depends on halo mass binning. The methods for calculating the velocity correlation function can be found in Chapter 3.

4.2.1 Comparison between models

In Figures 4.7 and 4.8 see that for all models we have a peak and the first range bin and then a decline, this is because then two halos are close together they are likely to be part of the same larger structure and so their radial peculiar velocity relative to the observer is likely to be around the same and have the same sign leading to a positive contribution to $\xi_v(r)$. For all the models the function flattens out above zero meaning that the halo pairs with this large a separation still either move both away or towards the observer. In theory the larger seperation we have between halos the less correlated they should be. As each has less effect on the other. We do not see that here, in fact the correlation increases on scales larger than 130 Mpc/h.

From the relative deviation plots (Figures 4.10 and 4.9) we see that neither of the models vary as much between bins as the errorbars would suggest they should the same reasoning that was applied to the streaming velocities hold here.

F4 deviates strongly from ΛCDM up until around 60 Mpc/h, well outside the errorbars. 60 Mpc/h is much much longer than the fifth force range (around 22 Mpc/h) for F4. The same logic we used for effects on larger scales than the force lengths for the streaming velocities also applies here, in short these scales were smaller at an earlier time. The theories with lower range fifth forces deviate less just like for the streaming velocities.

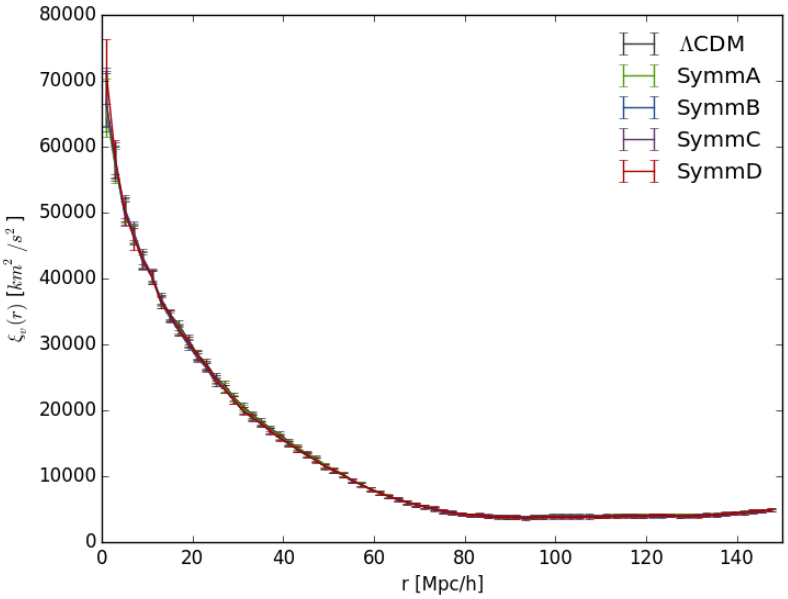


Figure 4.8: $\xi_v(r)$ plotted for the symmetron theories and ΛCDM with 10 Mpc/h sized bins. The asymmetric errorbars are determined by the bootstrap method describe in Section 3.1.4.

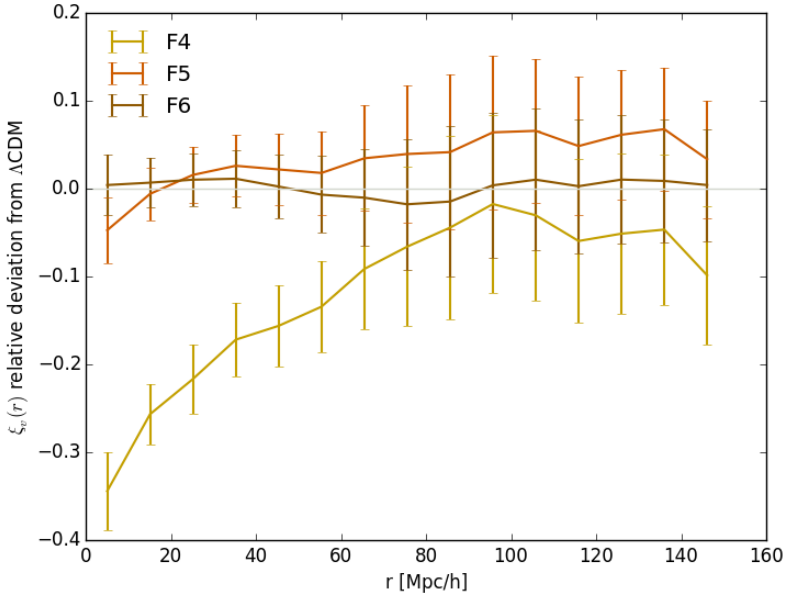


Figure 4.10: $\xi_v(r)$ relative deviation from ΛCDM plotted for all of f(R) theories with 10 Mpc/h sized bins. The asymmetric symmetric errors are determined by the bootstrap method describe in Section 3.1.4 and then propagated following the method in Section 3.1.4.2 assuming no covariance between ΛCDM and the alternative theory.

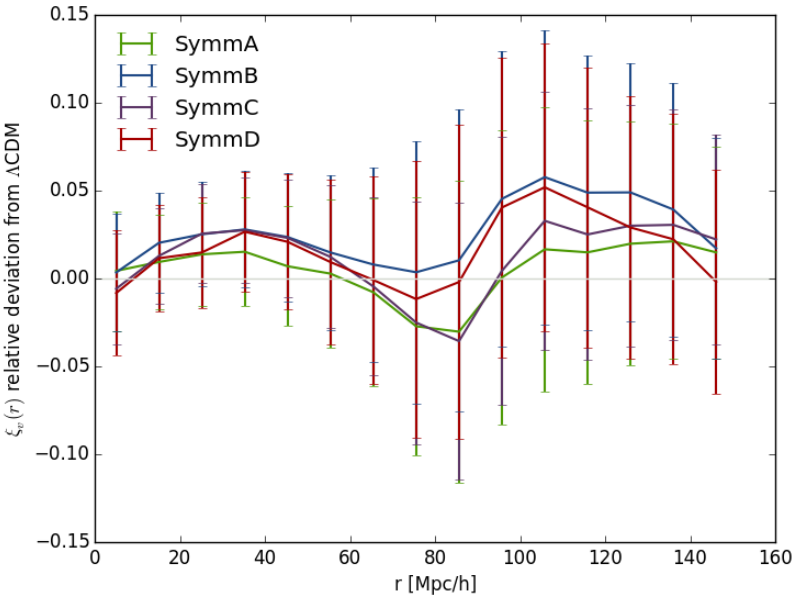
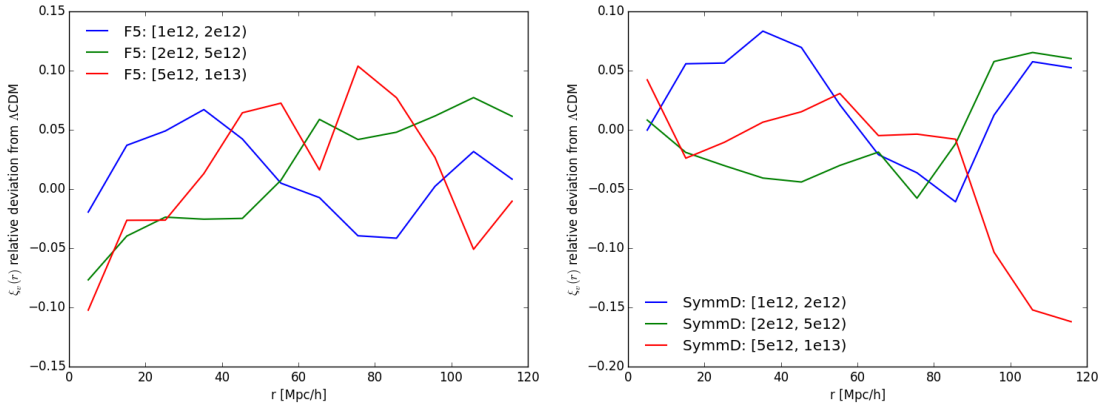


Figure 4.9: $\xi_v(r)$ relative deviation from ΛCDM plotted for all of the symmetron theories with 10 Mpc/h sized bins. The asymmetric symmetric errors are determined by the bootstrap method describe in Section 3.1.4 and then propagated following the method in Section 3.1.4.2 assuming no covariance between ΛCDM and the alternative theory.

F6 traces ΛCDM very well, oscillating around zero deviation. This stronger correlation for F4 and F5 on short range can be explained by that the fifth force binds the halo pairs together more strongly so there is a larger chance of them moving together and the radial peculiar velocity of each halo relative to the observer have the same sign for both halos (positive contribution).

At larger scales than 25 Mpc/h F5 has a lower ξ_v, this is however not true for F6 as it oscillates very closely to ΛCDM. F4 is more correlated than ΛCDM on all scales. What we would expect would be for there to be no difference in correlation on large scales as the fifth force the two halos or the larger structures they are in does not work. The velocities of the halos are still however different than what they are in ΛCDM, so we would expect some deviation that would average out to zero over all realizations. These deviations should have a higher variance for more powerful theories (stronger or/and longer range fifth force) as the halos in general move faster, making the sign of each pairs contribution more important relative to ΛCDM.

In Figure 4.9 the relative deviation from ΛCDM of the symmetron models is plotted for $\xi_v(r)$. We see that they oscillate around ΛCDM, but with around the same "wavelength". This is hard to explain, we have no idea what could cause this. It does however indicate that if we want to detect deviations from GR using the velocity correlation function, the scales most promising for our selection of symmetron models are between 15 and 50 Mpc/h. The 110 to 150 Mpc/h scale also looks promising, but here we need to worry about the box size to a larger extent. Additionally, it is harder to measure $\xi_v(r)$ for large r as distance measurement is harder for galaxies far away from the observer and one needs to survey a very large a to get a good enough sample of pairs.



(a) (b)

Figure 4.11: $\xi_v(r)$ relative deviation from ΛCDM plotted for F5 (a) and SymmD (b) with different halo mass ranges (given in units of M_{sun}/h in the legend) and 10 Mpc/h sized bins.

4.2.2 Dependence on mass and binning

Just like when we used mass bins on the pair-wise streaming velocities, the lighter halos do not deviate more then the other bins at smaller scales like we would expect. The same logic holds here; this could be something that varies with the seed used in the initial conditions to a large extent and what environment the halos are in. Ideally we would want to run many simulations using different realisations of the initial matter distribution and see if we on average would get what we expect, stronger deviations for lighter halos. Also we could calculate ξ_v for halos in different environments (different matter densities in the halos sorroundings). It could be that this would give a much clearer signal than binning by mass.

Chapter 5

Conclusions and Discussion

In modern cosmology we have a model called ΛCDM that fits the data extremely well, but even though there are no major empirical problems with the model, there are some very big theoretical ones. In ΛCDM the needed dark energy content to cause the accelerated expansion of the Universe is much less than the amount of vacuum energy predicted in particle physics, so either all of it cancels somehow and something else is causing the acceleration, or all of it except an extremely small part cancels (amounting to a 55 order fine-tuning [6, 7]). To solve this problem a number of modified gravity theories have been proposed [8]. In order to confirm or rule out these we need to test gravity.

The modified gravity theories need to evade the current constraints on gravity of which the most stringent ones are from experiments and observations in high density environments, e.g. the solar system. In order to do this the modified gravity theories employ what are called screening mechanisms (e.g. chameleon and symmetron screening) to suppress their effects on gravity in high density environments, resulting in an enhanced gravitational force working with different strength on objects that are screened or unscreened (e.g. galaxies of different sizes), causing apparent violations of the equivalence principle. The enhanced gravitational force causes objects to move differently than in standard gravity and because of the screening this deviation from standard gravity should be different for different objects, therefore using velocities to test these theories should be quite promising. Using future surveys there is hope we can measure the velocities well enough to be useful in testing gravity.

A novel part of our work is that we have compared velocity statistics for modified gravity models using different screening mechanisms, as well as comparing the same model using different parameters.

For each screening mechanism we tested different parts of the parameter space in separate simulations, but using the same exact initial conditions for each. This allows us to compare the data from the different simulations and be able to say that it is the difference between the theories that cause the differences and not different initial conditions (e.g. using different seeds).

We looked at two different velocity statistics, the streaming pair-wise velocities and the velocity correlation functions. Different velocity statistics probe the velocity field in different ways. The streaming pair-wise velocities measure the tendency for objects to approach each other, while the velocity correlation functions measure how correlated the peculiar velocities of objects along the line of sight from an observer are with each other. So we probe the velocity field in two different ways, measuring two different properties.

We found that the streaming velocities for ΛCDM for the most part behaved the way we expected, approaching zero when the distance of separation approaches zero, though gener-

ally the galaxies approach each other more rapidly the smaller the separation between them. However, we found "step-like" features between the peak and 40 Mpc/h, and v_{12} never comes close to zero for large scales like one would expect in a homogeneous universe. The amount of deviation from ΛCDM for the $f(R)$ models came in the order we expected with F4 having the largest deviation, then F5 and F6. For the symmetron models we found that changing the time of symmetry breaking (when the force is turned on) had a larger impact on the statistic than changing the force strength. In all of the modified gravity models we found deviations on much larger scales than the force length at the time of measurement ($z = 0$) we did not inquire whether this was consistent with, the redshift dependence of the fifth force however. We also found the streaming velocities are suppressed at large scales compared to ΛCDM , but we were not able to figure out why and would have expected the opposite. We found puzzling deviations from ΛCDM at large scales when using different mass bins, which we were unable to immediately explain; this should be addressed in future work.

In the velocity correlation function ΛCDM behaved as expected, the line of sight peculiar velocities were more and more correlated the lower the distance of separation. However, it flattens out at right above 80 Mpc/h, we would expect it to approach no correlation at large scales. For the $f(R)$ models F4, though no longer a viable model [49], deviated more from ΛCDM, with a larger correlation, the smaller a scale we go to. F5 was more correlated on scales smaller than 20 Mpc/h compared to ΛCDM, while less correlated on larger scales than 20 Mpc/h. For the symmetron models we found deviations from ΛCDM with a wave-like shape, but not understand why.

F6 was hard to distinguish from ΛCDM in either velocity statistic, except its large scale suppression in v_{12} were similar to F5 in magnitude. In the streaming velocities we saw larger deviations from ΛCDM for all models, than in the velocity correlation function, so the pair-wise streaming velocity looked like the more promising statistic when it comes to distinguishing the different theories from standard gravity.

The errorbars are not correct as we have for example not taken into account the covariance between the different models due to using the same seed for our initial conditions. We need to account for the following to make meaningful errorbars:

- Variations in the statistics caused by different realisations of the initial matter density distributions, this should cause the errorbars to be a bit larger.
- The covariance between different models (the change from one realisation to another will be somewhat similar for different models), this should cause the errorbars in the relative deviations to become smaller.
- The covariance between bins, should make the errorbars on the absolute statistic smaller.

All of these can be done by running lots of simulations using different seeds to generate the initial conditions and then using the distribution of the estimations of the statistic. This is extremely computationally expensive and thus takes a very long time [21]. There may be a faster way of doing this, e.g. by solving an informed approximation of the n-body simulation equations [63].

Our linear approximations do not match our results on linear scales, so an even more thorough examination of our approximations to the velocity statistics needs to be done. We should also use the same initial power spectrum as in the simulations and extract the non-linear correlation functions from the data. It is however, more likely that the approximations are wrong than our calculations from the simulations as there are a lot more places to make mistakes when calculating the approximations.

Also we are not sure to exactly how large a scale our results hold due to the limited box size. We could check this by running simulations for different box sizes and then check how

much the statistics vary when adjusting the box size, like was done in [52] with the two-point correlation function.

In future work one would want to calculate the statistics using halos in different environments, in hope of finding some stronger deviations from standard gravity, particularly in low density environments. Then one would want to compare results to observations, like measurements of f [34] and calculations of the statistics themselves using the galaxy/cluster catalogues from redshift surveys and surveys using the kinematic Sunyaev Zel'dovich effect [61].

Bibliography

[1] A. G. Riess, A. V. Filippenko, P. Challis, A. Clocchiatti, A. Diercks, P. M. Garnavich, R. L. Gilliland, C. J. Hogan, S. Jha, R. P. Kirshner, B. Leibundgut, M. M. Phillips, D. Reiss, B. P. Schmidt, R. A. Schommer, R. C. Smith, J. Spyromilio, C. Stubbs, N. B. Suntzeff, and J. Tonry. Observational Evidence from Supernovae for an Accelerating Universe and a Cosmological Constant. AJ, 116:1009–1038, September 1998. doi: 10.1086/300499.

[2] S. Perlmutter, G. Aldering, G. Goldhaber, R. A. Knop, P. Nugent, P. G. Castro, S. Deustua, S. Fabbro, A. Goobar, D. E. Groom, I. M. Hook, A. G. Kim, M. Y. Kim, J. C. Lee, N. J. Nunes, R. Pain, C. R. Pennypacker, R. Quimby, C. Lidman, R. S. Ellis, M. Irwin, R. G. McMahon, P. Ruiz-Lapuente, N. Walton, B. Schaefer, B. J. Boyle, A. V. Filippenko, T. Matheson, A. S. Fruchter, N. Panagia, H. J. M. Newberg, W. J. Couch, and T. S. C. Project. Measurements of Ω and Λ from 42 High-Redshift Supernovae. ApJ, 517:565–586, June 1999. doi: 10.1086/307221.

[3] Planck Collaboration, P. A. R. Ade, N. Aghanim, C. Armitage-Caplan, M. Arnaud, M. Ashdown, F. Atrio-Barandela, J. Aumont, C. Baccigalupi, A. J. Banday, and et al. Planck 2013 results. XVI. Cosmological parameters. A&A, 571:A16, November 2014. doi: 10.1051/0004-6361/201321591.

[4] Planck Collaboration, R. Adam, P. A. R. Ade, N. Aghanim, Y. Akrami, M. I. R. Alves, M. Arnaud, F. Arroja, J. Aumont, C. Baccigalupi, and et al. Planck 2015 results. I. Overview of products and scientific results. *ArXiv e-prints*, February 2015.

[5] C. L. Bennett, D. Larson, J. L. Weiland, N. Jarosik, G. Hinshaw, N. Odegard, K. M. Smith, R. S. Hill, B. Gold, M. Halpern, E. Komatsu, M. R. Nolta, L. Page, D. N. Spergel, E. Wollack, J. Dunkley, A. Kogut, M. Limon, S. S. Meyer, G. S. Tucker, and E. L. Wright. Nine-year wilkinson microwave anisotropy probe (wmap) observations: Final maps and results. *The Astrophysical Journal Supplement Series*, 208(2):20, 2013. URL `http://stacks.iop.org/0067-0049/208/i=2/a=20`.

[6] Steven Weinberg. The Cosmological Constant Problem. *Rev.Mod.Phys.*, 61:1–23, 1989. doi: 10.1103/RevModPhys.61.1.

[7] J. Martin. Everything you always wanted to know about the cosmological constant problem (but were afraid to ask). *Comptes Rendus Physique*, 13:566–665, July 2012. doi: 10.1016/j.crhy.2012.04.008.

[8] T. Clifton, P. G. Ferreira, A. Padilla, and C. Skordis. Modified gravity and cosmology. Phys. Rep., 513:1–189, March 2012. doi: 10.1016/j.physrep.2012.01.001.

[9] J. Amiaux, R. Scaramella, Y. Mellier, B. Altieri, C. Burigana, A. Da Silva, P. Gomez, J. Hoar, R. Laureijs, E. Maiorano, D. Magalhães Oliveira, F. Renk, G. Saavedra Criado,

I. Tereno, J. L. Auguères, J. Brinchmann, M. Cropper, L. Duvet, A. Ealet, P. Franzetti, B. Garilli, P. Gondoin, L. Guzzo, H. Hoekstra, R. Holmes, K. Jahnke, T. Kitching, M. Meneghetti, W. Percival, and S. Warren. Euclid mission: building of a reference survey. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 8442 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 0, September 2012. doi: 10.1117/12.926513.

[10] The Dark Energy Survey Collaboration. The Dark Energy Survey. *ArXiv Astrophysics e-prints*, October 2005.

[11] M. D. Niemack, P. A. R. Ade, J. Aguirre, F. Barrientos, J. A. Beall, J. R. Bond, J. Britton, H. M. Cho, S. Das, M. J. Devlin, S. Dicker, J. Dunkley, R. Dünner, J. W. Fowler, A. Hajian, M. Halpern, M. Hasselfield, G. C. Hilton, M. Hilton, J. Hubmayr, J. P. Hughes, L. Infante, K. D. Irwin, N. Jarosik, J. Klein, A. Kosowsky, T. A. Marriage, J. McMahon, F. Menanteau, K. Moodley, J. P. Nibarger, M. R. Nolta, L. A. Page, B. Partridge, E. D. Reese, J. Sievers, D. N. Spergel, S. T. Staggs, R. Thornton, C. Tucker, E. Wollack, and K. W. Yoon. ACTPol: a polarization-sensitive receiver for the Atacama Cosmology Telescope. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7741 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 1, July 2010. doi: 10.1117/12.857464.

[12] J. E. Austermann, K. A. Aird, J. A. Beall, D. Becker, A. Bender, B. A. Benson, L. E. Bleem, J. Britton, J. E. Carlstrom, C. L. Chang, H. C. Chiang, H.-M. Cho, T. M. Crawford, A. T. Crites, A. Datesman, T. de Haan, M. A. Dobbs, E. M. George, N. W. Halverson, N. Harrington, J. W. Henning, G. C. Hilton, G. P. Holder, W. L. Holzapfel, S. Hoover, N. Huang, J. Hubmayr, K. D. Irwin, R. Keisler, J. Kennedy, L. Knox, A. T. Lee, E. Leitch, D. Li, M. Lueker, D. P. Marrone, J. J. McMahon, J. Mehl, S. S. Meyer, T. E. Montroy, T. Natoli, J. P. Nibarger, M. D. Niemack, V. Novosad, S. Padin, C. Pryke, C. L. Reichardt, J. E. Ruhl, B. R. Saliwanchik, J. T. Sayre, K. K. Schaffer, E. Shirokoff, A. A. Stark, K. Story, K. Vanderlinde, J. D. Vieira, G. Wang, R. Williamson, V. Yefremenko, K. W. Yoon, and O. Zahn. SPTpol: an instrument for CMB polarization measurements with the South Pole Telescope. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 8452 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 1, September 2012. doi: 10.1117/12.927286.

[13] S.M. Carroll. *Spacetime and Geometry: An Introduction to General Relativity*. Addison Wesley, 2004. ISBN 9780805387322. URL `http://books.google.no/books?id=1SKFQgAACAAJ`.

[14] Planck Collaboration, P. A. R. Ade, N. Aghanim, M. Arnaud, M. Ashdown, J. Aumont, C. Baccigalupi, A. J. Banday, R. B. Barreiro, N. Bartolo, and et al. Planck 2015 results. XIV. Dark energy and modified gravity. *ArXiv e-prints*, February 2015.

[15] Planck Collaboration, P. A. R. Ade, N. Aghanim, M. Arnaud, M. Ashdown, J. Aumont, C. Baccigalupi, A. J. Banday, R. B. Barreiro, J. G. Bartlett, and et al. Planck 2015 results. XIII. Cosmological parameters. *ArXiv e-prints*, February 2015.

[16] S. Dodelson. *Modern Cosmology*. Academic Press. Academic Press, 2003. ISBN 9780122191411. URL `http://books.google.it/books?id=3oPRxdXJexcC`.

[17] T. Baker, P. Ferreira, and C. Skordis. A fast route to modified gravitational growth. Phys. Rev. D, 89(2):024026, January 2014. doi: 10.1103/PhysRevD.89.024026.

[18] P. Brax. Lectures on Screened Modified Gravity. *ArXiv e-prints*, November 2012.

[19] T. P. Waterhouse. An Introduction to Chameleon Gravity. *ArXiv Astrophysics e-prints*, November 2006.

[20] W. Hu and I. Sawicki. Models of f(R) cosmic acceleration that evade solar system tests. Phys. Rev. D, 76(6):064004, September 2007. doi: 10.1103/PhysRevD.76.064004.

[21] C. Llinares, D. F. Mota, and H. A. Winther. ISIS: a new N-body cosmological code with scalar fields based on RAMSES. Code presentation and application to the shapes of clusters. A&A, 562:A78, February 2014. doi: 10.1051/0004-6361/201322412.

[22] K. Hinterbichler, J. Khoury, A. Levy, and A. Matas. Symmetron cosmology. Phys. Rev. D, 84(10):103521, November 2011. doi: 10.1103/PhysRevD.84.103521.

[23] J. Khoury and A. Weltman. Chameleon cosmology. Phys. Rev. D, 69(4):044026, February 2004. doi: 10.1103/PhysRevD.69.044026.

[24] L. Hui, A. Nicolis, and C. W. Stubbs. Equivalence principle implications of modified gravity models. Phys. Rev. D, 80(10):104002, November 2009. doi: 10.1103/PhysRevD.80.104002.

[25] L.S. Sparke and J.S. Gallagher. *Galaxies in the Universe: An Introduction*. Cambridge University Press, 2007. ISBN 9781139462389. URL `https://books.google.no/books?id=N8Hngab5liQC`.

[26] L. Anderson, É. Aubourg, S. Bailey, F. Beutler, V. Bhardwaj, M. Blanton, A. S. Bolton, J. Brinkmann, J. R. Brownstein, A. Burden, C.-H. Chuang, A. J. Cuesta, K. S. Dawson, D. J. Eisenstein, S. Escoffier, J. E. Gunn, H. Guo, S. Ho, K. Honscheid, C. Howlett, D. Kirkby, R. H. Lupton, M. Manera, C. Maraston, C. K. McBride, O. Mena, F. Montesano, R. C. Nichol, S. E. Nuza, M. D. Olmstead, N. Padmanabhan, N. Palanque-Delabrouille, J. Parejko, W. J. Percival, P. Petitjean, F. Prada, A. M. Price-Whelan, B. Reid, N. A. Roe, A. J. Ross, N. P. Ross, C. G. Sabiu, S. Saito, L. Samushia, A. G. Sánchez, D. J. Schlegel, D. P. Schneider, C. G. Scoccola, H.-J. Seo, R. A. Skibba, M. A. Strauss, M. E. C. Swanson, D. Thomas, J. L. Tinker, R. Tojeiro, M. V. Magaña, L. Verde, D. A. Wake, B. A. Weaver, D. H. Weinberg, M. White, X. Xu, C. Yèche, I. Zehavi, and G.-B. Zhao. The clustering of galaxies in the SDSS-III Baryon Oscillation Spectroscopic Survey: baryon acoustic oscillations in the Data Releases 10 and 11 Galaxy samples. MNRAS, 441:24–62, June 2014. doi: 10.1093/mnras/stu523.

[27] C.-P. Ma and J. N. Fry. Nonlinear Kinetic Sunyaev-Zeldovich Effect. *Physical Review Letters*, 88(21):211301, May 2002. doi: 10.1103/PhysRevLett.88.211301.

[28] P.J.E. Peebles. *The Large-scale Structure of the Universe*. Princeton series in physics. Princeton University Press, 1980. ISBN 9780691082400. URL `http://books.google.no/books?id=0_BPaHFtX1YC`.

[29] M. Vargas-Magaña, J. E. Bautista, J.-C. Hamilton, N. G. Busca, É. Aubourg, A. Labatie, J.-M. Le Goff, S. Escoffier, M. Manera, C. K. McBride, D. P. Schneider, and C. N. A. Willmer. An optimized correlation function estimator for galaxy surveys. A&A, 554:A131, June 2013. doi: 10.1051/0004-6361/201220790.

[30] S. D. Landy and A. S. Szalay. Bias and variance of angular correlation functions. ApJ, 412:64–71, July 1993. doi: 10.1086/172900.

[31] H. J. Mo and S. D. M. White. An analytic model for the spatial clustering of dark matter haloes. MNRAS, 282:347–361, September 1996.

[32] W. H. Press and P. Schechter. Formation of Galaxies and Clusters of Galaxies by Self-Similar Gravitational Condensation. ApJ, 187:425–438, February 1974. doi: 10.1086/152650.

[33] R. K. Sheth, H. J. Mo, and G. Tormen. Ellipsoidal collapse and an improved model for the number and spatial distribution of dark matter haloes. MNRAS, 323:1–12, May 2001. doi: 10.1046/j.1365-8711.2001.04006.x.

[34] A. Johnson, C. Blake, J. Dossett, J. Koda, D. Parkinson, and S. Joudaki. Searching for Modified Gravity: Scale and Redshift Dependent Constraints from Galaxy Peculiar Velocities. *ArXiv e-prints*, April 2015.

[35] K. L. Masters, C. M. Springob, M. P. Haynes, and R. Giovanelli. SFI++ I: A New I-Band Tully-Fisher Template, the Cluster Peculiar Velocity Dispersion, and H_0. ApJ, 653:861–880, December 2006. doi: 10.1086/508924.

[36] P. G. Ferreira, R. Juszkiewicz, H. A. Feldman, M. Davis, and A. H. Jaffe. Streaming Velocities as a Dynamical Estimator of Ω. ApJ, 515:L1–L4, April 1999. doi: 10.1086/311959.

[37] E. Bertschinger. Simulations of Structure Formation in the Universe. ARA&A, 36:599–654, 1998. doi: 10.1146/annurev.astro.36.1.599.

[38] R. Teyssier. Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES. A&A, 385:337–364, April 2002. doi: 10.1051/0004-6361:20011817.

[39] A. V. Kravtsov, A. A. Klypin, and A. M. Khokhlov. Adaptive Refinement Tree: A New High-Resolution N-Body Code for Cosmological Simulations. ApJS, 111:73–94, July 1997. doi: 10.1086/313015.

[40] P. S. Behroozi, R. H. Wechsler, and H.-Y. Wu. The ROCKSTAR Phase-space Temporal Halo Finder and the Velocity Offsets of Cluster Cores. ApJ, 762:109, January 2013. doi: 10.1088/0004-637X/762/2/109.

[41] D. Larson, J. Dunkley, G. Hinshaw, E. Komatsu, M. R. Nolta, C. L. Bennett, B. Gold, M. Halpern, R. S. Hill, N. Jarosik, A. Kogut, M. Limon, S. S. Meyer, N. Odegard, L. Page, K. M. Smith, D. N. Spergel, G. S. Tucker, J. L. Weiland, E. Wollack, and E. L. Wright. Seven-year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Power Spectra and WMAP-derived Parameters. ApJS, 192:16, February 2011. doi: 10.1088/0067-0049/192/2/16.

[42] E. Komatsu, K. M. Smith, J. Dunkley, C. L. Bennett, B. Gold, G. Hinshaw, N. Jarosik, D. Larson, M. R. Nolta, L. Page, D. N. Spergel, M. Halpern, R. S. Hill, A. Kogut, M. Limon, S. S. Meyer, N. Odegard, G. S. Tucker, J. L. Weiland, E. Wollack, and E. L. Wright. Seven-year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Cosmological Interpretation. ApJS, 192:18, February 2011. doi: 10.1088/0067-0049/192/2/18.

[43] E. Bertschinger. COSMICS: Cosmological Initial Conditions and Microwave Anisotropy Codes. *ArXiv Astrophysics e-prints*, June 1995.

[44] B. Li, W. A. Hellwing, K. Koyama, G.-B. Zhao, E. Jennings, and C. M. Baugh. The non-linear matter and velocity power spectra in f(R) gravity. MNRAS, 428:743–755, January 2013. doi: 10.1093/mnras/sts072.

[45] G.-B. Zhao, B. Li, and K. Koyama. N-body simulations for f(R) gravity using a self-adaptive particle-mesh code. Phys. Rev. D, 83(4):044007, February 2011. doi: 10.1103/PhysRevD.83.044007.

[46] W. A. Hellwing, B. Li, C. S. Frenk, and S. Cole. Hierarchical clustering in chameleon f(R) gravity. MNRAS, 435:2806–2821, November 2013. doi: 10.1093/mnras/stt1430.

[47] C. Corbett Moran, R. Teyssier, and B. Li. Chameleon $f(R)$ gravity on the Virgo cluster scale. *ArXiv e-prints*, August 2014.

[48] E. Jennings, C. M. Baugh, B. Li, G.-B. Zhao, and K. Koyama. Redshift-space distortions in f(R) gravity. MNRAS, 425:2128–2143, September 2012. doi: 10.1111/j.1365-2966.2012.21567.x.

[49] H. Wilcox, D. Bacon, R. C. Nichol, P. J. Rooney, A. Terukina, A. K. Romer, K. Koyama, G.-B. Zhao, R. Hood, R. G. Mann, M. Hilton, M. Manolopoulou, M. Sahlen, C. A. Collins, A. R. Liddle, J. A. Mayers, N. Mehrtens, C. J. Miller, J. P. Stott, and P. T. P. Viana. The XMM Cluster Survey: Testing chameleon gravity using the profiles of clusters. *ArXiv e-prints*, April 2015.

[50] M. Kuhlen, M. Vogelsberger, and R. Angulo. Numerical simulations of the dark universe: State of the art and the next decade. *Physics of the Dark Universe*, 1:50–93, November 2012. doi: 10.1016/j.dark.2012.10.002.

[51] T. Hamana, N. Yoshida, and Y. Suto. Reliability of the Dark Matter Clustering in Cosmological N-Body Simulations on Scales below the Mean Separation Length of Particles. ApJ, 568:455–462, April 2002. doi: 10.1086/338970.

[52] C. Power and A. Knebe. The impact of box size on the properties of dark matter haloes in cosmological simulations. MNRAS, 370:691–701, August 2006. doi: 10.1111/j.1365-2966.2006.10562.x.

[53] J. Tinker, A. V. Kravtsov, A. Klypin, K. Abazajian, M. Warren, G. Yepes, S. Gottlöber, and D. E. Holz. Toward a Halo Mass Function for Precision Cosmology: The Limits of Universality. ApJ, 688:709–728, December 2008. doi: 10.1086/591439.

[54] S. Bhattacharya, A. Kosowsky, J. A. Newman, and A. R. Zentner. Galaxy peculiar velocities from large-scale supernova surveys as a dark energy probe. Phys. Rev. D, 83(4):043004, February 2011. doi: 10.1103/PhysRevD.83.043004.

[55] Antony Lewis and Sarah Bridle. Cosmological parameters from CMB and other data: a Monte- Carlo approach. *Phys. Rev.*, D66:103511, 2002.

[56] Gnu scientific library: Qag adaptive integration, May 2015. URL `http://www.gnu.org/software/gsl/manual/html_node/QAG-adaptive-integration.html#QAG-adaptive-integration`.

[57] R. Juszkiewicz, V. Springel, and R. Durrer. Dynamics of Pairwise Motions. ApJ, 518:L25–L28, June 1999. doi: 10.1086/312055.

[58] S. Bhattacharya and A. Kosowsky. Dark energy constraints from galaxy cluster peculiar velocities. Phys. Rev. D, 77(8):083004, April 2008. doi: 10.1103/PhysRevD.77.083004.

[59] S. G. Murray, C. Power, and A. S. G. Robotham. HMFcalc: An online tool for calculating dark matter halo mass functions. *Astronomy and Computing*, 3:23–34, November 2013. doi: 10.1016/j.ascom.2013.11.001.

[60] B. Bassett and R. Hlozek. *Baryon acoustic oscillations*, page 246. 2010.

[61] E.-M. Mueller, F. de Bernardis, R. Bean, and M. Niemack. Constraints on gravity and dark energy from the pairwise kinematic Sunyaev-Zeldovich effect. *ArXiv e-prints*, August 2014.

[62] W. A. Hellwing, A. Barreira, C. S. Frenk, B. Li, and S. Cole. Clear and Measurable Signature of Modified Gravity in the Galaxy Velocity Field. *Physical Review Letters*, 112 (22):221102, June 2014. doi: 10.1103/PhysRevLett.112.221102.

[63] H. A. Winther and P. G. Ferreira. A Fast Route to Non-Linear Clustering Statistics in Modified Gravity Theories. *ArXiv e-prints*, March 2014.

Appendices

Appendix A

Code

A.1 Calculating velocity statistics

A.1.1 main.cpp

```
1  // Standard Library and others:
2  #include <iostream>
3  #include <cmath>
4  #include <fstream>
5  #include <sstream>
6  #include <string>
7  #include <vector>
8  #include <math.h>
9  #include <stdlib.h>
10 #include <time.h>
11 #include <omp.h>
12 #include <chrono>
13
14 // Debugging:
15 #include <cassert>
16
17 // Self written:
18 #include "../gen_purpose/functions.h"
19 #include "../gen_purpose/statistics.h"
20 #include "velo_tools.h"
21 #include "bookkeeping.h"
22
23 #define N_cpu 64
24 #define runtime_mins std::chrono::duration_cast<std::chrono::
       milliseconds>(std::chrono::high_resolution_clock::now()-start).
       count()/(1000.0*60.0)
25
26 using std::cout; using std::endl;
27 using std::vector; using std::string;
28
```

```
29 vector<vector<double> > velocorrfunc(vector<double> o, vector<vector
       <double> > H, vector<vector<double> >::size_type N, double
       boxsize, int binsize, vector<double>& cos2, vector<double>&
       counter, string stat, int sampling = 0, int vbin = 0);
30 vector<double> velocityCorrection(vector<double> dists, vector<
       double> o, vector<double> v, double boxsize);
31 vector<double> positionCorrection(vector<double> d, double boxsize);
32 vector<vector<double> > std_jackknife(vector<vector<double> >
       samples);
33 vector<vector<vector<double> > > bootstrap(string stat, string cosmo
       , int samples, vector<double> o, vector<vector<double> > relH,
       double boxsize, int binsize, vector<double> xvec, string ext);
34 void pairBins(const vector<vector<double> > &Halos, vector<vector<
       int> > &pB, int binsize);
35 void pairCosThetas(const vector<vector<double> > &Halos, vector<
       vector<double> > &pT, int binsize);
36
37 #define NDEBUG
38
39 auto start = std::chrono::high_resolution_clock::now();
40
41 vector<vector<int> > pB;
42 vector<vector<double> > pT;
43
44 int main(int argc, char** argv)
45 {
46     // Debugging section, does not run if NDEBUG is defined:
47     #ifndef NDEBUG
48
49     string teststat = "vrel";
50
51     int testbinsize = 2;
52     double testboxsize = 3;
53     int testboxsize_int = sqrt(3*testboxsize*testboxsize);
54     vector<double> testobs = {1.5, 1.5, 1.5};
55     int testsumlength = testboxsize_int/testbinsize + 1;
56     cout << testsumlength << endl;
57     vector<double> testcorr(testsumlength);
58     vector<double> testcounter(testsumlength);
59     vector<vector<double> > testHalos;
60     if (teststat == "vcorr")
61     {
62         testHalos.push_back({0, 2.3, 2.3, 123, 144, 102, 0});
63         testHalos.push_back({1.2, 2.9, 1.4, 70, -120, -56, 0});
64         testHalos.push_back({0.8, 2.1, 0.3, -180, -4, 28, 0});
65         testHalos.push_back({0.3, 1.3, 2.3, 90, -24, -13, 0});
66     }
67     else if (teststat == "vrel")
68     {
69         testHalos.push_back({0, 2.3, 2.3, 123, 144, 102, 0});
```

```
70          testHalos.push_back({1.2, 2.9, 1.4, 70, -120, -56, 0});
71      }
72
73      auto testN = testHalos.size();
74      vector<vector<double> > testrelH = posrad(testobs, testHalos,
        testboxsize);
75      //cout << "a" << endl;
76
77      for (decltype(testN) i = 0; i != testN; ++i)
78          testrelH[i].push_back(i);
79
80      //cout << "b" << endl;
81      pairBins(testrelH, pB, testbinsize);
82      //cout << "c" << endl;
83      if (teststat == "vrel")
84      {
85          pairCosThetas(testrelH, pT, testbinsize);
86      }
87
88      //cout << "d" << endl;
89      vector<vector<double> > testsamples = velocorrfunc(testobs,
        testrelH, testN, testboxsize, testbinsize, testcorr, testcounter,
         teststat);
90
91      //vector<vector<double> > samp = velocorrfunc(o, relH, N,
        boxsize, binsize, corr, counter, stat, sampling);
92      //cout << "e" << endl;
93      vector<double> testx = {1, 3, 5};
94      //cout << "f" << endl;
95      write("testcorr.dat", testx.data(), testcorr.data(),
        testsumlength);
96      cout << "End of test." << endl;
97      while (testx[0] == 1) ;
98      #endif
99
100     string stat = argv[1];//"vrel";//"vcorr_noesti";//"vcorr";//"
        vcorr_noesti";
101     string cosmo = argv[2];//"lcdm";
102
103     //Observer posisions from (0,1) in  each timension:
104     double xpos = 0.5; double ypos = 0.5; double zpos = 0.5;
105
106     string binscale = "linear";
107     double boxsize = 256;
108
109     bool errors = true;//true;
110     int binsize = std::stoi(argv[5]);
111     double massFloor = std::stod(argv[4]);
112     double massCeiling = std::stod(argv[7]);
113     if ((massCeiling < massFloor) and (massCeiling >= 0))
```

```
114         cout << "Invalid massbin!!!" << endl;
115     int boxsize_int = sqrt(3*boxsize*boxsize);
116     int sumlength = boxsize_int/binsize + 1;
117     int samples = std::stoi(argv[3]);//3;
118     int sampling = 0;
119     if (samples == 0)
120     {
121         sampling = 1;
122     }
123     else if (samples == -1)
124     {
125         sampling = 2;
126     }
127     cout << "Sampling is set to option: " << sampling << endl;
128     std::stringstream ext;
129     ext << "binscale_" << binscale << "_binsize_" << binsize << "
    _massfloor_" << massFloor;
130
131     vector<double> corr(sumlength);
132     vector<double> counter(sumlength);
133     int Nran;
134
135     unsigned int N_c = 7;
136     vector<int> col = {2, 8, 9, 10, 11, 12, 13};
137     int header = 19;
138     vector<vector<double> > Halos;
139     if (cosmo != "random")
140     {
141         if (cosmo == "random_lcdm")
142             Halos = load("../../data/runs/run_lcdm/halos_0.0.ascii",
    N_c, col);
143         else
144             Halos = load("../../data/runs/run_" + cosmo + "/halos_0
    .0.ascii", N_c, col);
145
146         double m;
147         for(decltype(Halos.size()) i = 0; i<Halos.size(); ++i)
148         {
149             m = Halos[i][0];
150             Halos[i].erase(Halos[i].begin());
151             Halos[i].push_back(m);
152         }
153
154         // Eliminating halos with masses below the set mass floor:
155         auto itHalos = Halos.begin();
156         while (itHalos != Halos.end())
157         {
158             if (massCeiling < 0)
159                 if ((*itHalos)[6] < massFloor)
160                     itHalos = Halos.erase(itHalos);
```

```
161                 else
162                     ++itHalos;
163             if (massCeiling >= 0)
164                 if ((*itHalos)[6] < massFloor or (*itHalos)[6] >
    massCeiling)
165                     itHalos = Halos.erase(itHalos);
166                 else
167                     ++itHalos;
168         }
169
170         if (cosmo == "random_lcdm")
171         {
172             Nran = 10000;
173             Halos = ranSelectCat(Halos, Nran);
174         }
175     }
176
177     double vstd;
178     unsigned int seed;
179
180     if (cosmo == "random")
181     {
182         Nran = std::stoi(argv[4]);
183         vstd = std::stod(argv[6]);//200;
184         seed = std::stoi(argv[7]);//time(NULL);
185         Halos = genRanCat(Nran, vstd, boxsize, seed);
186     }
187
188     auto N = Halos.size();
189     cout << "Number of halos above mass floor: " << N << endl;
190
191     //Set observer position:
192     vector<double> o = {xpos*boxsize, ypos*boxsize, zpos*boxsize};
193     vector<vector<double>> relH = posrad(o, Halos, boxsize);
194
195     for (decltype(N) i = 0; i != N; ++i)
196         relH[i].push_back(i);
197
198     cout << "Starting to calculate binnings... | At runtime: " <<
    runtime_mins << " min." << endl;
199     pairBins(relH, pB, binsize);
200     cout << "Done calculating binnings." << " | At runtime: " <<
    runtime_mins << " min." << endl;
201
202     if (stat == "vrel")
203     {
204         cout << "Starting to calculate p-s at runtime: " <<
    runtime_mins << " min." << endl;
205         pairCosThetas(relH, pT, binsize);
```

```
206         cout << "Done calculathing p-s at runtime: " << runtime_mins
        << " min." << endl;
207     }
208
209     vector<vector<double> > samp = velocorrfunc(o, relH, N, boxsize,
        binsize, corr, counter, stat, sampling);
210
211     vector<double> xvec(sumlength);
212     double step = sqrt(3*boxsize*boxsize)/((double)sumlength - 1.0);
213     for (int i = 0; i<sumlength; ++i) xvec[i] = i*step + 0.5*step;
214
215     //string velotxt, counttxt, sampletxt;
216     int vbincount = samp[0].size();
217     vector<double> vvec(samp[0].size());
218
219     vector<string> filenames = filenamegen(stat, cosmo, ext.str(),
       Nran, vstd, seed, sampling, samples, massCeiling);
220     string velotxt = filenames[0];
221     string counttxt = filenames[1];
222     string sampletxt = filenames[2];
223
224     write(velotxt.c_str(), xvec.data(), corr.data(), sumlength);
225     write(counttxt.c_str(), xvec.data(), counter.data(), sumlength);
226
227     if (sampling == 1)
228     {
229         for (decltype(samp.size()) i = 0; i<samp[0].size(); ++i)
230             vvec[i] = (i-vbincount/2.0)*25.0 - 12.5;
231     }
232     else if (sampling == 2)
233     {
234         for (decltype(samp.size()) i = 0; i<samp[0].size(); ++i)
235             vvec[i] = (i+1.0)*25.0 + 12.5;
236     }
237     samp.push_back(vvec);
238     writeMatrix(sampletxt, samp);
239     cout << "Runtime is: " << runtime_mins << " mins." << endl;
240
241     string corrstr, counterstr;
242     vector<vector<vector<double> > > sampvec;
243     if (sampling == 2)
244         errors = false;
245     if (errors)
246     {
247         sampvec = bootstrap(stat, cosmo, samples, o, relH, boxsize,
       binsize, xvec, ext.str());
248
249         string corrstr = filenames[3];
250         string counterstr = filenames[4];
251
```

```
252             writeMatrix(corrstr, sampvec[0]);
253             writeMatrix(counterstr, sampvec[1]);
254         }
255         /*
256         if (errors)
257         {
258             cout << "Calculating standard deviations..." << endl;
259             vector<vector<double> > std_mean = std_jackknife(samples);
260             auto N_bins = std_mean.size();
261 
262             for (decltype(N_bins) i = 0; i != N_bins; ++i)
263             {
264                 std_mean[i].insert(std_mean[i].begin(),1,xvec[i]);
265             }
266 
267             writeMatrix("std_jack.dat", std_mean);
268         }
269         */
270         return 0;
271 }
272 
273 void pairBins(const vector<vector<double> > &Halos, vector<vector<
        int> > &pB, int binsize)
274 {
275     auto N = Halos.size();
276     vector<int> empty;//(N);
277     double boxsize = 900;
278     double dist;
279     int bin;
280     for (decltype(N) i = 0; i < N; ++i)
281     {
282         pB.push_back(empty);
283         for (auto j = i; j<N; ++j)
284             pB[i].push_back(0);
285     }
286     #pragma omp parallel for  ordered num_threads(N_cpu) schedule(
        dynamic) private(dist, bin)
287     for (decltype(N) i = 0; i < N; ++i)
288     {
289         for (auto j = i; j<N; ++j)
290         {
291             dist = rfind(Halos[i][0], Halos[i][1], Halos[i][2],
        Halos[j][0], Halos[j][1], Halos[j][2], boxsize, 0);
292             bin = dist/binsize;
293             if (bin < 0 or bin > 221)
294                 cout << bin <<  " i: " << i << "j: " << endl;
295             if (i == j)
296                 bin = 0;
297             pB[i][j-i] = bin;
298         }
```

```
299     }
300 }
301
302 void pairCosThetas(const vector<vector<double> > &Halos, vector<
    vector<double> > &pT, int binsize)
303 {
304     auto N = Halos.size();
305     vector<double> empty;
306     double boxsize = 900;
307     double cos_theta, theta;
308     vector<double> d1(3);
309     vector<double> d2(3);
310     vector<double> ru;
311     vector<double> d1u(3);
312     vector<double> d2u(3);
313     double rul, d1l, d2l;
314     double p;
315     //#pragma omp parallel for  ordered num_threads(N_cpu) schedule(
    dynamic)
316     for (decltype(N) i = 0; i < N; ++i)
317     {
318         pT.push_back(empty);
319         for(auto j = i; j<N; ++j)
320         {
321             pT[i].push_back(0);
322         }
323     }
324     #pragma omp parallel for  ordered num_threads(N_cpu) schedule(
    dynamic) private(d1, d2, ru, p, rul, d1l, d2l, d1u, d2u)
325     for (decltype(N) i = 0; i < N; ++i)
326     {
327         d1u = {0, 0, 0}; d2u = {0, 0, 0};
328
329         for (auto j = i; j<N; ++j)
330         {
331             if (i == j)
332                 p = 0;
333             else
334             {
335                 d1 = {Halos[i][0], Halos[i][1], Halos[i][2]};
336                 d2 = {Halos[j][0], Halos[j][1], Halos[j][2]};
337                 ru = vec_subtract(d1, d2);
338                 rul = vec_length(ru);
339                 d1l = vec_length(d1);
340                 d2l = vec_length(d2);
341                 for (int k = 0; k<3; ++k)
342                 {
343                     ru[k] = ru[k]/rul;
344                     d1u[k] = d1[k]/d1l;
345                     d2u[k] = d2[k]/d2l;
```

```
346                 }
347                 p = dot_product(ru, vec_add(d1u, d2u));
348                 #ifndef NDEBUG
349                 cout << "ru: (" << ru[0] << ", " << ru[1] << ", " <<
    ru[2] <<")" << endl;
350                 cout << "d1u: (" << d1u[0] << ", " << d1u[1] << ", "
    << d1u[2] << ")" << endl;
351                 cout << "d2u: (" << d2u[0] << ", " << d2u[1] << ", "
    << d2u[2] << ")" << endl;
352                 #endif
353             }
354
355             pT[i][j-i] = p;
356         }
357     }
358 }
359 vector<vector<vector<double> > > bootstrap(string stat, string cosmo
    , int samples, vector<double> o, vector<vector<double> > relH,
    double boxsize, int binsize, vector<double> xvec, string ext)
360 {
361     auto N = relH.size();
362     vector<double> nullVec = xvec;
363     for (decltype(N) i = 0; i != nullVec.size(); ++i)
364         nullVec[i] = 0;
365     vector<double> fourVec = {0, 0, 0, 0};
366     vector<vector<double> > s;
367     vector<vector<double> > corr(samples+1, nullVec);
368     vector<vector<double> > counter(samples+1, nullVec);
369     vector<vector<double> > ranHorg(N, fourVec);
370     vector<vector<double> > ranH;
371     corr[0] = xvec; counter[0] = xvec;
372     bool seeded = false;
373     std::mt19937 generator(42u);
374     if (!seeded)
375     {
376         generator.seed(static_cast<unsigned int>(time(NULL)));
377         seeded = true;
378         std::cout << "Seeded uniform RNG engine for bootstrap." <<
    std::endl;
379     }
380     boost::uniform_int<> uni_dist(0, N-1);
381     boost::variate_generator<std::mt19937&, boost::uniform_int<> >
    uni(generator, uni_dist);
382
383     //#pragma omp parallell num_threads(N_cpu)
384     //{
385     //vector<vector<double> > ranH(N, fourVec);
386     #pragma omp parallel for ordered num_threads(N_cpu) schedule(
    dynamic) private(ranH,s) shared(N, o, boxsize, binsize, corr,
    counter, stat, ranHorg)
```

```
387     for (int i = 0; i < samples; ++i)
388     {
389         ranH = ranHorg;
390         if (i%100 == 0)
391             cout << "Sample number " << i+1 << " out of " << samples
        << ". | Calculated by thread number " << omp_get_thread_num() <<
        ". | At runtime: " << std::chrono::duration_cast<std::chrono::
        milliseconds>(std::chrono::high_resolution_clock::now()-start).
        count()/(1000.0*60.0) << " min." << endl;
392
393         for (decltype(N) j = 0; j != N; ++j)
394         {
395             ranH[j] = relH[uni()];
396         }
397         s = velocorrfunc(o, ranH, N, boxsize, binsize, corr[i+1],
        counter[i+1], stat);
398     }
399
400     return {corr, counter};
401 }
402
403 vector<vector<double> > velocorrfunc(vector<double> o, vector<vector
        <double> > H, vector<vector<double> >::size_type N, double
        boxsize, int binsize, vector<double>& corr, vector<double>&
        counter, string stat, int sampling, int vbin)
404 {
405     double v_max = 1e6;
406     double v_min = -v_max;
407     double vbinSize = 25;
408
409     int vbincount;
410     if (sampling == 1)
411         vbincount = 2*v_max/vbinSize + 2;
412     else if (sampling == 2)
413         vbincount = v_max/vbinSize + 1;
414     double dist;
415     int bin, vbinSelect;
416     auto N_bins = corr.size();
417     vector<double> cos2(N_bins);
418     vector<double> empty(vbincount);
419     vector<vector<double> > samples;
420     for (decltype(N_bins) i = 0; i<N_bins; ++i)
421         samples.push_back(empty);
422
423     double theta;
424     double cos_theta;
425
426     vector<double> p1(3);
427     vector<double> p2(3);
428     vector<double> d1(3);
```

```
429     vector<double> d2(3);
430     vector<double> v1Vec(3);
431     vector<double> v2Vec(3);
432
433     double v1, v2; // radial velocities
434     double p, d1l, d2l, rul, over, under;
435     vector<double> d1u(3);
436     vector<double> d2u(3);
437     vector<double> ru(3);
438     int i_ind, j_ind;
439     for (decltype(N) i = 0; i<N; ++i)
440     {
441         //if (i%10 == 0) cout << i << " | Runtime: " << runtime_mins
    << " min." << endl;
442         for (auto j = i+1; j<N; ++j)
443         {
444             if (H[i][4] > H[j][4])
445             {
446                 i_ind = H[j][4];
447                 j_ind = H[i][4];
448             }
449
450             else
451                 i_ind = H[i][4];
452                 j_ind = H[j][4];
453             if (i_ind == j_ind)
454                 continue;
455             bin = pB[i_ind][j_ind-i_ind];
456
457             v1 = H[i][3];//radial_velocity(d1, v1Vec);
458             v2 = H[j][3];//radial_velocity(d2, v2Vec);
459
460             #ifndef NDEBUG
461             cout << "i, j: " << i << ", " << j << endl;
462             cout << "dist: " << dist << endl;
463             cout << "bin: " << bin << endl;
464             cout << "vi, vj: " << v1 << ", " << v2 << endl;
465
466             #endif
467
468             if (stat == "vcorr_esti")
469             {
470                 over = v1*v2*cos_theta;
471                 under = cos_theta*cos_theta;
472                 corr[bin] += over;
473                 cos2[bin] += under;
474             }
475             else if (stat == "vcorr")
476             {
477                 over = v1*v2;
```

```
478                 under = 1;
479                 corr[bin] += over;
480                 cos2[bin] += under;
481             }
482             else if (stat == "vrel")
483             {
484                 p = pT[i_ind][j_ind-i_ind];
485                 over = 2*(v1 - v2)*p;
486                 under = p*p;
487                 corr[bin] += over;
488                 cos2[bin] += under;
489                 #ifndef NDEBUG
490                 cout << "p: " << p << endl;
491                 #endif
492             }
493             counter[bin] += 1;
494             if (sampling == 1)
495             {
496                 //over = -120;
497                 //under = 1;
498                 vbinSelect = vbincount/2 + (int)((over/under))/
    vbinSize;
499                 if (vbinSelect < 0)
500                     vbinSelect = 0;
501                 else if (vbinSelect > vbincount-1)
502                     vbinSelect = vbincount-1;
503                 samples[bin][vbinSelect] += 1;
504             }
505 
506             else if (sampling == 2)
507             {
508                 vbinSelect = (int)abs(over)/vbinSize;
509                 if (vbinSelect > vbincount-1)
510                     vbinSelect = vbincount-1;
511                 samples[bin][vbinSelect] += 1;
512             }
513         }
514     }
515     for (decltype(N_bins) i = 0; i<N_bins; i++)
516     {
517         //if (samples[i].size() != counter[i])
518         //  std::cerr << "The number of samples in bin " << i << "
    is not equal to the pair count!!!" << endl;
519         //else
520         //  ;
521         if (cos2[i] != 0.0)
522         {
523             corr[i] = corr[i]/cos2[i];//counter[i];
524         }
525         //else
```

```
526             //  cout << "cos2 has a 0 value." << endl;
527         }
528         return samples;
529     }
530
531     vector<double> velocityCorrection(vector<double> dists, vector<
        double> o, vector<double> v, double boxsize)
532     {
533         double boxsize_half = boxsize/2.0;
534         for (int i = 0; i!= 3; ++i)
535         {
536             if (abs(dists[i]) > boxsize_half)
537                 v[i] = -v[i];
538         }
539
540         return v;
541     }
542
543     vector<double> positionCorrection(vector<double> d, double boxsize)
544     {
545         double L = boxsize/2.0;
546         for(int i = 0; i<3; ++i)
547         {
548             if (d[i] > L) d[i] -= boxsize;
549             else if(d[i] < -L) d[i] += boxsize;
550         }
551         return d;
552     }
553
554     vector<vector<double> > std_jackknife(vector<vector<double> >
        samples)
555     {
556         auto N_bins = samples.size();
557         vector<double>nullVec(2);
558         vector<vector<double> > std_mean(N_bins, nullVec);
559         #pragma omp parallel for num_threads(N_cpu) schedule(dynamic)
560         for (decltype(N_bins) i = 0; i<N_bins; ++i)
561         {
562             if (samples[i].size() == 0) continue;
563             if (true)//i%10 == 0)
564                 cout << "\n" << "Bin number " << i << ": Samplesize to
        jackknife: " << samples[i].size() << endl;
565                 cout << "Calculated by thread number: " <<
        omp_get_thread_num() << "  At runtime: " << std::chrono::
        duration_cast<std::chrono::milliseconds>(std::chrono::
        high_resolution_clock::now()-start).count()/(1000.0*60.0) << "
        min." << endl;
566             std_mean[i][0] = bootstrap(samples[i], std_mean[i][1], "symm
        ");
567         }
```

```
568
569     return std_mean;
570 }
```

A.1.2 velo_tools.h

```
1  #ifndef VELO_TOOLS
2  #define VELO_TOOLS
3
4  #include <iostream>
5  #include <cmath>
6  #include <random>
7  #include <boost/random/uniform_real.hpp>
8  #include <boost/random/normal_distribution.hpp>
9  #include <boost/random/variate_generator.hpp>
10
11 #include "../gen_purpose/functions.h"
12
13 using std::cout; using std::endl;
14
15 inline double radial_velocity(std::vector<double> x, std::vector<
       double> v);
16 std::vector<std::vector<double> > posrad(std::vector<double> o, std
       ::vector<std::vector<double> > H, double boxsize);
17 inline std::vector<double> velocityCorrection(std::vector<double> d,
        std::vector<double> o, std::vector<double> v, double boxsize);
18 inline std::vector<double> positionCorrection(std::vector<double> d,
        double boxsize);
19
20 std::vector<double> pairRanger(std::vector<std::vector<double> > H);
21
22 std::vector<std::vector<double> > genRanCat(const unsigned int N,
       const double vstd, const double boxsize, const unsigned int seed
       = time(NULL));
23 std::vector<std::vector<double> > ranSelectCat(std::vector<std::
       vector<double> > Org, const int Npick, unsigned int seed = -1);
24
25 #include "velo_tools.cpp"
26 #endif
```

A.1.3 velo_tools.cpp

```
1  double radial_velocity(std::vector<double> x, std::vector<double> v)
2  {
3      // Parameters:
4      // x: vector (x,y,z) from the  position of the stationary
       observer to the halo.
5      // v: vector with (v_x, v_y, v_z) velocity of the halo.
6
7      // Returns:
8      // v_r: The radial velocity of the halo seen from the observer.
9
10     double v_r = dot_product(x, v)/sqrt(dot_product(x, x));
11     return v_r;
12 }
13
14 std::vector<std::vector<double> > posrad(std::vector<double> o, std
       ::vector<std::vector<double> > H, double boxsize)
15 {
16     auto N = H.size();
17     double vr;
18     std::vector<double> nullVec(4);
19     std::vector<std::vector<double> > relH(N, nullVec);
20     std::vector<double> d(3); std::vector<double> v(3);
21     for (decltype(N) i = 0; i != N; ++i)
22     {
23         relH[i][0] = H[i][0]-o[0]; relH[i][1] = H[i][1]-o[1]; relH[i
       ][2] = H[i][2]-o[2];
24         d = {relH[i][0], relH[i][1], relH[i][2]};
25         v = {H[i][3], H[i][4], H[i][5]};
26         v = velocityCorrection(d, o, v, boxsize);
27         d = positionCorrection(d, boxsize);
28         vr = radial_velocity(d, v);
29         relH[i][3] = vr;
30     }
31
32     return relH;
33 }
34
35 std::vector<double> pairRanger(std::vector<std::vector<double> > H)
36 {
37     std::vector<double> r;
38     auto N = H.size();
39     double boxsize = 900;
40     for (decltype(N) i = 0; i != N; ++i)
41     {
42         for (decltype(N) j = 0; j != i; ++j)
43         {
44             r.push_back(rfind(H[i][0], H[i][1], H[i][2], H[j][0], H[
       j][1], H[j][2], boxsize));
```

```
45          }
46      }
47
48      return r;
49  }
50
51  std::vector<std::vector<double> > genRanCat(const unsigned int N,
        const double vstd, const double boxsize, const unsigned int seed)
52  {
53      // Generates random catalogue for use in velostats.
54      // Format: {x, y, z, vx, vy, vz}
55      std::mt19937 generator(42u);
56      cout << "asd" << endl;
57      generator.seed(static_cast<unsigned int>(seed));
58      boost::uniform_real<> uni_dist(0,boxsize);
59      boost::variate_generator<std::mt19937&, boost::uniform_real<> >
        uni(generator, uni_dist);
60      cout << "a" << endl;
61      boost::variate_generator<std::mt19937, boost::
        normal_distribution<> >
62          norm(std::mt19937(seed),
63                  boost::normal_distribution<>(0, vstd));
64
65      std::vector<double> nullVec(7);
66      std::vector<std::vector<double> > cat(N, nullVec);
67      for (unsigned int i = 0; i<N; ++i)
68      {
69          cat[i][0] = uni();
70          cat[i][1] = uni();
71          cat[i][2] = uni();
72          cat[i][3] = norm();
73          cat[i][4] = norm();
74          cat[i][5] = norm();
75      }
76
77      return cat;
78  }
79
80  std::vector<std::vector<double> > ranSelectCat(std::vector<std::
        vector<double> > Org, const int Npick, unsigned int seed)
81  {
82      if (seed == -1)
83          seed = time(NULL);
84      std::mt19937 generator(42u);
85      generator.seed(static_cast<unsigned int>(seed));
86      boost::uniform_int<> uni_dist(0,Org.size()-1);
87      boost::variate_generator<std::mt19937&, boost::uniform_int<> >
        uni(generator, uni_dist);
88
89      std::vector<std::vector<double> > New;
```

```
90      for (int i = 0; i<Npick; ++i)
91      {
92          New.push_back(Org[uni()]);
93      }
94
95      return New;
96  }
```

A.1.4 bookkeeping.h

```
1 #ifndef BOOKKEEPING_H
2 #define BOOKKEEPING_H
3
4 std::vector<std::string> filenamegen(std::string stat, std::string
      cosmo, std::string ext, int Nran, double vstd, unsigned int seed,
       int sampling, int samples, double massCeiling);
5
6 #include "bookkeeping.cpp"
7 #endif
```

A.1.5 bookkeeping.cpp

```
1  std::vector<std::string> filenamegen(std::string stat, std::string
       cosmo, std::string ext, int Nran, double vstd, unsigned int seed,
        int sampling, int samples, double massCeiling)
2  {
3
4      std::string corrstr, counterstr;
5      std::stringstream n, mC;
6      n << samples;
7      mC << massCeiling;
8
9      std::string velotxt, counttxt, sampletxt;
10     std::stringstream vstdstr, Nranstr, seedstr;
11     vstdstr << vstd; Nranstr << Nran; seedstr << seed;
12
13     std::string folder;
14     if (cosmo != "random" and cosmo != "random_lcdm")
15     {
16         folder = "cosmoruns/";
17         if (massCeiling > 0)
18             ext = ext + "_massCeiling_" + mC.str();
19         velotxt = folder +  stat + "_" + ext + "_" + cosmo + ".txt";
20         counttxt = folder + "count_" + ext + "_" + cosmo + ".txt";
21         sampletxt = folder + "samples/samples_" + ext + "_" + cosmo
       + ".txt";
22         if (sampling == 2)
23             sampletxt = folder + "samples/samples_over_" + ext + "_"
        + cosmo + ".txt";
24         corrstr = folder + "bootstrap_" + stat + "_" + ext + "_" +
       cosmo + "_" + n.str() + ".txt";
25         counterstr = folder + "bootstrap_count_" + ext + "_" + cosmo
        + "_" + n.str() + ".txt";
26     }
27
28     else if (cosmo == "random")
29     {
30         velotxt = "randomcat/" + stat + "_vstd_" + vstdstr.str() + "
       _Nran_" + Nranstr.str() + "_seed_" + seedstr.str() + ".txt";
31         counttxt = "randomcat/count_" + vstdstr.str() + "_Nran_" +
       Nranstr.str() + "_seed_" + seedstr.str() + ".txt";
32         sampletxt = "randomcat/sample_" + stat + "_vstd_" + vstdstr.
       str() + "_Nran_" + Nranstr.str() + "_seed_" + seedstr.str() + ".
       txt";
33
34         corrstr = "randomcat/bootstrap_random_" + n.str() + "_" +
       stat + "_vstd_" + vstdstr.str() + "_Nran_" + Nranstr.str() + "
       _seed_" + seedstr.str() + ".txt";
35         counterstr = "randomcat/bootstrap_count_" + n.str() + "_" +
       stat + "_vstd_" + vstdstr.str() + "_Nran_" + Nranstr.str() + "
```

```
    _seed_" + seedstr.str() + ".txt";
36
37          if (sampling == 2)
38              sampletxt = "randomcat/sample_over_" + stat + "_vstd_" +
     vstdstr.str() + "_Nran_" + Nranstr.str() + "_seed_" + seedstr.
    str() + ".txt";
39
40      }
41
42      else if (cosmo == "random_lcdm")
43      {
44          folder = "ranlcdmruns/";
45          velotxt = folder + stat + "_" + ext + "_" + cosmo + "_Npick_
    " + Nranstr.str() + ".txt";
46          counttxt = folder + "count_" + ext + "_" + cosmo + "_Npick_"
     + Nranstr.str() + ".txt";
47          sampletxt = folder + "samples/samples_" + ext + "_" + cosmo
    + "_Npick_" + Nranstr.str() + ".txt";
48          if (sampling == 2)
49              sampletxt =  folder + "samples/samples_over_" + ext + "_
    " + cosmo + "_Npick_" + Nranstr.str() + ".txt";
50          corrstr = folder + "bootstrap_" + stat + "_" + ext + "_" +
    cosmo + "_" + n.str() + "_Npick_" + Nranstr.str() + ".txt";
51          counterstr = folder + "bootstrap_count_" + ext + "_" + cosmo
     + "_" + n.str() + "_Npick_" + Nranstr.str() + ".txt";
52      }
53
54
55
56      return {velotxt, counttxt, sampletxt, corrstr, counterstr};
57  }
```

A.2 Approximating the velocity correlation function

A.2.1 main.cpp

```
1  // Standard library:
2  #include <iostream>
3  #include <sstream>
4  #include <cmath>
5  #include <vector>
6  #include <string>
7  #include <cassert>
8
9  // GNU Scientific Library:
10 #include <gsl/gsl_errno.h>
11 #include <gsl/gsl_matrix.h>
12 #include <gsl/gsl_odeiv2.h>
13 #include <gsl/gsl_sf_bessel.h>
14 #include <gsl/gsl_spline.h>
15 #include <gsl/gsl_integration.h>
16
17 // Selfwritten files:
18 #include "../gen_purpose/functions.h"
19 #include "../gen_purpose/statistics.h"
20
21 struct  spline {gsl_interp_accel* acc; gsl_spline* splined;};
22
23 #include "tools.cpp"
24 #include "calc.cpp"
25
26 using std::cout; using std::endl;
27 using std::vector; using std::string;
28
29 int testfunc(double x, const double y[], double f[], void* params);
30 int testjac(double x, const double y[], double *dfdy, double dfdx[],
       void *params);
31
32 //We use that approximation f = Omega_m**0.6.
33
34 int main()
35 {
36     //Testing:
37     #ifndef NDEBUG
38     /*
39     vector<double> xt = {3.0, -2.5, 0.09};
40     vector<double> yt = {-8.9, 4.2, -7.3};
41     cout << angle_vector(xt, yt) << endl;
42     assert(dot_product(xt, yt) == -37.857);
43     cout << "angle_vector(xt, yt) == 2.48315: " << angle_vector(xt,
       yt) << endl;;//abs(acos(-37.857/(sqrt(15.2581)*sqrt(150.64)))));
```

```
44      */
45
46      // Example 1:
47      vector<double> Ex1_o = {1.5, 1.5, 1.5};
48      vector<double> Ex1_p1 = {2.1, 2.9, 0.0};
49      vector<double> Ex1_p2 = {0.6, 2.3, 0.0};
50      vector<double> Ex1_theta = anglepos(Ex1_p1, Ex1_p2, Ex1_o, 3);
51      cout << "Example 1: " << "\n";
52      cout << "theta_1 = " << Ex1_theta[0] << "\n";
53      cout << "theta_2 = " << Ex1_theta[1] << "\n";
54      cout << "|x| = " << Ex1_theta[2] << "\n";
55      cout << "cos(theta_1)*cos(theta_2) = " << cos(Ex1_theta[0])*cos(
        Ex1_theta[1]) << "\n";
56      cout << "sin(theta_1)*sin(theta_2) = " << sin(Ex1_theta[0])*sin(
        Ex1_theta[1]) << endl;
57      cout << "\n";
58
59      cout << "Example 1 reversed:" << endl;
60      Ex1_theta = anglepos(Ex1_p2, Ex1_p1, Ex1_o, 3);
61      cout << "theta_1 = " << Ex1_theta[0] << "\n";
62      cout << "theta_2 = " << Ex1_theta[1] << "\n";
63      cout << "|x| = " << Ex1_theta[2] << endl;
64      cout << "cos(theta_1)*cos(theta_2) = " << cos(Ex1_theta[0])*cos(
        Ex1_theta[1]) << "\n";
65      cout << "sin(theta_1)*sin(theta_2) = " << sin(Ex1_theta[0])*sin(
        Ex1_theta[1]) << endl;
66      cout << "\n";
67
68      cout << "Example 2:" << endl;
69      vector<double> Ex2_o = {1.5, 1.5, 1.5};
70      vector<double> Ex2_p1 = {1.5, 2.5, 1.5};
71      vector<double> Ex2_p2 = {1.5, 0.5, 1.5};
72      vector<double> Ex2_theta = anglepos(Ex2_p1, Ex2_p2, Ex2_o, 3);
73      cout << "theta_1 = " << Ex2_theta[0] << "\n";
74      cout << "theta_2 = " << Ex2_theta[1] << "\n";
75      cout << "|x| = " << Ex2_theta[2] << endl;
76      cout << "cos(theta_1)*cos(theta_2) = " << cos(Ex2_theta[0])*cos(
        Ex2_theta[1]) << "\n";
77      cout << "sin(theta_1)*sin(theta_2) = " << sin(Ex2_theta[0])*sin(
        Ex2_theta[1]) << endl;
78      cout << "\n";
79
80      int l = 0;
81      //while (l == 0) ;
82      #endif
83      //#ifdef NDEBUG
84
85      // Setting cosmological parameters:
86      double h = 0.719;
87      double H0 = h*100;
```

```
88      double Omega_m = 0.2670;
89      //double f = pow(Omega_m, 0.6);
90
91      double k_min;
92      double k_max;
93      double massFloor = 7.0e12; // M_sun/h
94      double L = 256.0; // sidelength of simulation box in Mpc/h.
95      int binsize = 2; // Mpc/h.
96
97      vector<double> o = {0.5*L/2,0.5*L/2,0.5*L/2}; // Observer
        position in the box.
98      cout << "Observer position set to, x: " << o[0] << "   y: " << o
        [1] << "   z: " << o[2] << endl;
99
100     // Importing powerspectrum and multiplying P(k) by h^2 to get km
        ^s as units for the velocity correlation:
101     //vector<vector<double> > P = load("kVector_simu.txt", 2, {0,
        1}, 4);
102     vector<vector<double> > P = load("test_matterpower.dat", 2, {0,
        1});
103     //vector<vector<double> > P = load("../../philcode/velocorr/
        pspec.dat", 2, {0, 1});
104     vector<double> kvec(P.size()); vector<double> Pvec(P.size());
105     for (decltype(P.size()) i = 0; i<P.size(); ++i)
106     {
107         kvec[i] = P[i][0]; Pvec[i] = P[i][1]/(h*h);
108     }
109
110     // Setting integration boundaries:
111     k_min = kvec[0]; k_max = kvec[kvec.size()-1];
112
113     // Allocating for and splining the power spectrum:
114     gsl_interp_accel* ps_acc = gsl_interp_accel_alloc();
115     gsl_spline* ps_spline = gsl_spline_alloc(gsl_interp_cspline, (
        int)P.size());
116     gsl_spline_init(ps_spline, kvec.data(), Pvec.data(), (int)P.size
        ());
117
118     cout << "ps splined" << endl;
119
120     //myparams params = {H0, Omega_m, 1.0, ps_acc, ps_spline};
121
122     // Setting minimum and maximum halos seperation:
123     double r_min = 0.01;
124     double r_max = sqrt(3*L*L);
125
126     // Allocating arrays to hold the parallell and perpendicular
        velocity correlation functions from linear theory:
127     int N2 = 1000;
128     double dr = (r_max - r_min)/(double)(N2-1);
```

```
129     double* r = new double[N2];
130     double* xi_para = new double[N2];
131     double* xi_perp = new double[N2];
132
133     // Calculating the parallell and perpendicular velocity
        correlation functions from linear theory:
134     const double hstart = 1e-2;
135     const double epsabs = 0;
136     const double epsrel = 1e-5;
137     cout << "k-range: [" << k_min << ", " << k_max << "]" << endl;
138     cout << "Selected k-range for integration: [" << k_min << ", "
        << k_max << "]" << endl;
139     gsl_function F_para, F_perp;
140     F_para.function = &para;
141     F_perp.function = &perp;
142     size_t limit = 1e6;
143     double result, error;
144     gsl_integration_workspace* w;
145     for (int i = 0; i<N2; i++)
146     {
147         w = gsl_integration_workspace_alloc(limit);
148         if (i % 100 == 0) cout << i << endl;
149         r[i] = r_min + i*dr;
150         struct myparams par = {H0, Omega_m, r[i], ps_acc, ps_spline
        };
151         F_para.params = &par;
152         F_perp.params = &par;
153         gsl_integration_qag(&F_para, k_min, k_max, epsabs, epsrel,
        limit, 6, w, &result, &error);
154         xi_para[i] = result;
155         gsl_integration_workspace_free(w);
156         w = gsl_integration_workspace_alloc(limit);
157         gsl_integration_qag(&F_perp, k_min, k_max, epsabs, epsrel,
        limit, 6, w, &result, &error);
158         xi_perp[i] = result;
159         gsl_integration_workspace_free(w);
160     }
161
162
163     // Allocating for and splining the two components:
164     gsl_interp_accel* para_acc = gsl_interp_accel_alloc();
165     gsl_interp_accel* perp_acc = gsl_interp_accel_alloc();
166     gsl_spline* para_spline = gsl_spline_alloc(gsl_interp_cspline,
        N2);
167     gsl_spline* perp_spline = gsl_spline_alloc(gsl_interp_cspline,
        N2);
168     gsl_spline_init(para_spline, r, xi_para, N2);
169     gsl_spline_init(perp_spline, r, xi_perp, N2);
170
171
```

```
172     /*// Use this if importing the two components from somewhere
        else:
173     vector<vector<double> > Imp_para = load("../../philcode/velocorr
        /para.dat", 2, {0, 1});
174     vector<vector<double> > Imp_perp = load("../../philcode/velocorr
        /perp.dat", 2, {0, 1});
175     auto N_Imp = Imp_para.size();
176     vector<double> Im_para(N_Imp); vector<double> Im_perp(N_Imp);
177     vector<double> r_imp(N_Imp);
178     for (decltype(N_Imp) i = 0; i<N_Imp; ++i)
179     {
180         r_imp[i] = Imp_para[i][0]/h;
181         Im_para[i] = Imp_para[i][1]; Im_perp[i] = Imp_perp[i][1];
182     }
183     r_min = r_imp[0];
184     r_max = r_imp[N_Imp-1];
185
186     gsl_interp_accel* para_acc = gsl_interp_accel_alloc();
187     gsl_interp_accel* perp_acc = gsl_interp_accel_alloc();
188     gsl_spline* para_spline = gsl_spline_alloc(gsl_interp_cspline,
        N_Imp);
189     gsl_spline* perp_spline = gsl_spline_alloc(gsl_interp_cspline,
        N_Imp);
190
191     gsl_spline_init(para_spline, r_imp.data(), Im_para.data(), N_Imp
        );
192     gsl_spline_init(perp_spline, r_imp.data(), Im_perp.data(), N_Imp
        );
193
194     write("para_imp.dat", r_imp.data(), Im_para.data(), N_Imp);
195     write("perp_imp.dat", r_imp.data(), Im_perp.data(), N_Imp);
196     */
197
198     // Initializing spline struct holding the splines of the two
        components:
199     spline sp_para = {para_acc, para_spline};
200     spline sp_perp = {perp_acc, perp_spline};
201
202     // Writing the two components to one file each:
203     write("para.dat", r, xi_para, N2);
204     write("perp.dat", r, xi_perp, N2);
205
206     // Releasing arrays:
207     delete[] r; delete[] xi_para; delete[] xi_perp;
208
209     cout << "Calculated and interpolated both the parallel and the
        perpendicular component." << endl;
210
211     // Reading in halo catalogue:
```

```
212     string haloCatLoc = "../../data/runs/run_lcdm/halos_0.0.
        ascii_stripped";
213     vector<vector<double> > halos = load(haloCatLoc, 4, {0, 1, 2,
        6});
214     auto ithalos = halos.begin();
215     while (ithalos != halos.end())
216     {
217         if ((*ithalos)[3] < massFloor)
218             ithalos = halos.erase(ithalos);
219         else
220             ++ithalos;
221     }
222     cout << "Number of halos above mass floor: " << halos.size() <<
        endl;
223
224     // Calculation the peculiar velocity correlation function from
        using a prediction for each halo
225     // pair from linear theory:
226     vector<double> xi = linVeloCorr(halos, o, L, r_min, r_max,
        binsize, &sp_para, &sp_perp);
227
228     int L_int = sqrt(3*L*L); // Maximum distance between two halos
        in the box.
229     int N_bins = L_int/binsize + 1; // Number of distance bins.
230
231     int N_halos = halos.size();
232     vector<double> x(N_bins);
233     double dx = (r_max - r_min)/(N_bins-1);
234     for (int i = 0; i != N_bins; ++i)
235         x[i] = i*dx+0.5*dx;
236
237     write("xi_tot.dat", x.data(), xi.data(), N_bins);
238
239     // For plotting the integrand of the xi components:
240     double I[1] = {0};
241     struct myparams paramsI = {H0, Omega_m, 50.0, ps_acc, ps_spline
        };
242     double y[2] = {0, 0};
243     double klogmin = log(k_min);
244     double klogmax = log(k_max- 0.0001);
245     int NI = 2000;
246     double logstep = (klogmax - klogmin)/(NI-1);
247     vector<double> Iperp(NI); vector<double> Ipara(NI);
248     vector<double> kvec_pl(NI);
249     for (int i = 0; i<NI; ++i)
250     {
251         kvec_pl[i] = exp(klogmin + i*logstep);
252     }
253     int status;
254     /*cout << "Calculating integrands..." << endl;
```

```
255     for (int i = 0; i != NI; ++i)
256     {
257         //cout << i << " " << kvec_pl[i] << endl;
258         status = perp(kvec_pl[i], y, I, &paramsI);
259         Iperp[i] = I[0];
260         I[0] = 0.0;
261         status = para(kvec_pl[i], y, I, &paramsI);
262         Ipara[i] = I[0];
263         I[0] = 0.0;
264     }
265
266     write("Iperp.dat", kvec_pl.data(), Iperp.data(), NI);
267     write("Ipara.dat", kvec_pl.data(), Ipara.data(), NI);
268     //#endif
269     */
270     #ifndef NDEBUG
271     // Testing integration:
272     int ntest = 100;
273     double test_xmin = k_min*r_min;
274     double test_xmax = 100;
275     double test_logdiff = (log(test_xmax) - log(test_xmin))/(ntest
        -1);
276     vector<double> j0int(ntest);
277     vector<double> j1int(ntest);
278     vector<double> xtest(ntest);
279     for (int i = 0; i != ntest; ++i)
280         xtest[i] = exp(log(test_xmin) + i*test_logdiff);
281
282     //gsl_interp_accel* ps_acc = gsl_interp_accel_alloc();
283     //gsl_spline* ps_spline = gsl_spline_alloc(gsl_interp_cspline,
        100);
284
285     struct myparams j0 = {0, 2.0, 3.0, ps_acc, ps_spline};
286     struct myparams j1 = {1, 2.0, 3.0, ps_acc, ps_spline};
287     const double testhstart = 1e-8;
288     const double testepsabs = 1e-8;
289     const double testepsrel = 0.0;
290     /*
291     for (int i = 1; i != ntest; ++i)
292     {
293         j0int[i] = integration(testfunc, testjac, testhstart,
        testepsabs, testepsrel, j0, test_xmin, xtest[i]);
294         j1int[i] = integration(testfunc, testjac, testhstart,
        testepsabs, testepsrel, j1, test_xmin, xtest[i]);
295     }
296
297     write("j0.dat", xtest.data(), j0int.data(), ntest);
298     write("j1.dat", xtest.data(), j1int.data(), ntest);
299     */
300     #endif
```

```
301     return 0;
302 }
303
304
305 int testfunc(double x, const double y[], double f[], void* params)
306 {
307     struct myparams *mu = (struct myparams *)params;
308     //cout << mu->a << endl;
309     //cout << mu->b << endl;
310     if (mu->a == 0)
311         f[0] = sin(x)/x; // j0(x)
312     else if (mu->a == 1)
313         f[0] = sin(x)/(x*x) - cos(x)/x; // j1(x)
314     return GSL_SUCCESS;
315 }
316
317 int testjac(double x, const double y[], double *dfdy, double dfdx[],
318             void* params)
319 {
320     struct myparams *mu = (struct myparams *)params;
321     //cout << mu << endl;
322     gsl_matrix_view dfdy_mat
323             = gsl_matrix_view_array(dfdy, 1, 2);
324     gsl_matrix *m = &dfdy_mat.matrix;
325     gsl_matrix_set(m, 0, 0, 0.0);
326     gsl_matrix_set(m, 0, 1, 0.0);
327     if (mu->a == 0)
328         dfdx[0] = cos(x)/x - sin(x)/(x*x); // Derivative of j0(x)
329     else if (mu->a == 1)
330         dfdx[0] = (x*cos(x) - 2*sin(x))/pow(x, 3) + (x*sin(x) + cos(
    x))/(x*x); // Derivative of j1(x)
331     return GSL_SUCCESS;
332 }
```

A.3 Approximating pair-wise streaming velocities

A.3.1 v12.cpp

```
1  // Standard library:
2  #include <iostream>
3  #include <sstream>
4  #include <cmath>
5  #include <vector>
6  #include <string>
7  #include <cassert>
8
9  // GNU Scientific Library:
10 #include <gsl/gsl_errno.h>
11 #include <gsl/gsl_matrix.h>
12 #include <gsl/gsl_odeiv2.h>
13 #include <gsl/gsl_sf_bessel.h>
14 #include <gsl/gsl_spline.h>
15 #include <gsl/gsl_integration.h>
16
17 // Selfwritten files:
18 #include "../gen_purpose/functions.h"
19 #include "../gen_purpose/statistics.h"
20
21 struct  spline {gsl_interp_accel* acc; gsl_spline* splined;};
22
23 #include "tools.cpp"
24 #include "calc.cpp"
25
26 using std::cout; using std::endl;
27 using std::vector; using std::string;
28
29 int testfunc(double x, const double y[], double f[], void* params);
30 int testjac(double x, const double y[], double *dfdy, double dfdx[],
       void *params);
31
32 //We use that approximation f = Omega_m**0.6.
33
34 int main(int argc, char** argv)
35 {
36     //Testing:
37     #ifndef NDEBUG
38     #endif
39
40     // Setting cosmological parameters:
41     double h = 0.719;
42     double H0 = h*100;
43     double Omega_m = 0.2670;
44     double f = pow(Omega_m, 0.6);
```

```
45
46      double k_min;
47      double k_max;
48      double massFloor = 7.0e12; // M_sun/h
49      double L = 256.0; // sidelength of simulation box in Mpc/h.
50      int binsize = 2; // Mpc/h.
51
52      vector<double> o = {0.5*L/2,0.5*L/2,0.5*L/2}; // Observer
        position in the box.
53      cout << "Observer position set to, x: " << o[0] << "   y: " << o
        [1] << "   z: " << o[2] << endl;
54
55      // Importing powerspectrum and multiplying P(k) by h^2 to get km
        ^s as units for the velocity correlation:
56      vector<vector<double> > P = load("kVector_simu.txt", 2, {0, 1},
        4);
57      //vector<vector<double> > P = load("test_matterpower.dat", 2,
        {0, 1});
58      //vector<vector<double> > P = load("../../philcode/velocorr/
        pspec.dat", 2, {0, 1});
59      vector<double> kvec(P.size()); vector<double> Pvec(P.size());
60      for (decltype(P.size()) i = 0; i<P.size(); ++i)
61      {
62          kvec[i] = P[i][0]; Pvec[i] = P[i][1];
63      }
64
65      // Setting integration boundaries:
66      k_min = kvec[0]; k_max = kvec[kvec.size()-1];
67
68      // Allocating for and splining the power spectrum:
69      gsl_interp_accel* ps_acc = gsl_interp_accel_alloc();
70      gsl_spline* ps_spline = gsl_spline_alloc(gsl_interp_cspline, (
        int)P.size());
71      gsl_spline_init(ps_spline, kvec.data(), Pvec.data(), (int)P.size
        ());
72
73      cout << "ps splined" << endl;
74
75      //myparams params = {H0, Omega_m, 1.0, ps_acc, ps_spline};
76
77      // Setting minimum and maximum halos seperation:
78      double r_min = 0.0001;
79      double r_max = sqrt(3*L*L);
80
81      // Allocating arrays to hold the correlation function from
        linear theory:
82      int N2 = 5000;
83      double dr = (r_max - r_min)/(double)(N2-1);
84      vector<double> r(N2);
85      vector<double> xi(N2);
```

```
86
87      k_min = k_min*1; k_max = k_max/1.0;
88      cout << "k_min: " << k_min << " | k_max: " << k_max << endl;
89      const double epsabs = 1e-2;
90      const double epsrel = 1e-3;
91      size_t limit = 1e6;
92      double result, error;
93      gsl_integration_workspace* w;
94      gsl_function F_xi;
95      F_xi.function = &Icorrfunc;
96      for (int i = 0; i<N2; ++i)
97      {
98          if (i%100==0) cout << i << endl;
99          w = gsl_integration_workspace_alloc(limit);
100         r[i] = r_min + i*dr;
101         struct myparams par = {H0, Omega_m, r[i], ps_acc, ps_spline
        };
102         F_xi.params = &par;
103         gsl_integration_qag(&F_xi, k_min, k_max, epsabs, epsrel,
        limit, 6, w, &result, &error);
104         xi[i] = result;
105         gsl_integration_workspace_free(w);
106     }
107
108     write ("analxi.txt", r.data(), xi.data(), xi.size());
109     cout << "Wrote analytical correlation function to analxi.txt."
        << endl;
110
111     // Allocating for and splining the correlation function:
112     gsl_interp_accel* xi_acc = gsl_interp_accel_alloc();
113     gsl_spline* xi_spline = gsl_spline_alloc(gsl_interp_cspline, (
        int)xi.size());
114     gsl_spline_init(xi_spline, r.data(), xi.data(), (int)xi.size());
115
116     double gamma = std::stod(argv[1]);
117     double alpha = 1.2 - 0.65*gamma;
118     double xibb;
119     gsl_function F_v_12, F_xiBar;
120     F_v_12.function = &Iv_12;
121     F_xiBar.function = &IxiBar;
122     vector<double> v_12(N2);
123     for (int i = 0; i<N2; ++i)
124     {
125         if (i%100==0) cout << i << endl;
126         w = gsl_integration_workspace_alloc(limit);
127         r[i] = r_min + i*dr;
128         struct myparams par = {H0, Omega_m, r[i], ps_acc, ps_spline
        };
129         //F_v_12.params = &par;
130         F_xiBar.params = &par;
```

```
131         //gsl_integration_qag(&F_v_12, k_min, k_max, epsabs, epsrel,
        limit, 6, w, &result, &error);
132         gsl_integration_qag(&F_xiBar, r_min, r[i], epsabs, epsrel,
        limit, 6, w, &result, &error);
133         //v_12[i] = -H0*r[i]*2*pow(Omega_m, 0.6)*result/(1+xi[i]*h);
134         xibb = result/(1.0+xi[i]);
135         v_12[i] = -(2.0/3.0)*H0*r[i]*f*xibb*(1.0+alpha*xibb)/h;
136         gsl_integration_workspace_free(w);
137     }
138     std::stringstream analv_12txt;
139     analv_12txt << "analv_12_gamma_" << gamma << ".txt";
140     write(analv_12txt.str().c_str(), r.data(),  v_12.data(), (int)
        v_12.size());
141
142     return 0;
143 }
```

A.4 Files used to approximate velocity statistics

A.4.1 tools.cpp

```
1  using std::cout; using std::endl;
2  using std::vector; using std::string;
3  using std::stringstream;
4
5  struct myparams {double a; double b; double c;
6                   gsl_interp_accel* acc; gsl_spline* spline;};
7
8  vector<double> anglepos(vector<double> p1, vector<double> p2, vector
       <double> o, double L)
9  {
10     // Finding x1 x2 and x in figure 9.8 in Dodelson:
11
12     // Electing the closest copy of each halo to the observer
13     // assuming the simulation box has periodic boundaries.
14     // If this is undesired one can just set the boxlength insanely
       long.
15
16     // x1 and x2 if we say that the observer is at origo:
17     vector<double> x1, x2;
18     x1.push_back(p1[0] - o[0]); x1.push_back(p1[1] - o[1]);
19     x1.push_back(p1[2] - o[2]);
20     x2.push_back(p2[0] - o[0]); x2.push_back(p2[1] - o[1]);
21     x2.push_back(p2[2] - o[2]);
22     double L2 = L/2.0; // Half of the box length;
23
24     if (x1[0] > L2) x1[0] -= L;
25     else if (x1[0] < -L2) x1[0] += L;
26     if (x1[1] > L2) x1[1] -= L;
27     else if (x1[1] < -L2) x1[1] += L;
28     if (x1[2] > L2) x1[2] -= L;
29     else if (x1[2] < -L2) x1[2] += L;
30
31     if (x2[0] > L2) x2[0] -= L;
32     else if (x2[0] < -L2) x2[0] += L;
33     if (x2[1] > L2) x2[1] -= L;
34     else if (x2[1] < -L2) x2[1] += L;
35     if (x2[2] > L2) x2[2] -= L;
36     else if (x2[2] < -L2) x2[2] += L;
37
38     // The vector between the halos:
39     vector<double> x(3);
40     x = {x2[0] - x1[0], x2[1] - x1[1], x2[2] - x1[2]};
41
42     // theta_1 and theta_2:
43     // theta_2 is the angle between x1 and theta_1 is the :
```

```
44      double xl = sqrt(dot_product(x, x));
45      double x1l = sqrt(dot_product(x1, x1));
46      double x2l = sqrt(dot_product(x2, x2));
47
48      double theta_1
49          = angle_vector(x1, x);
50      double theta_2
51          = angle_vector(x2, x);
52
53      return {theta_1, theta_2, xl};
54  }
55
56  // 3d window function in fourier space:
57  double W2(double kR)
58  {
59      double w;
60      w = 3.0*(sin(kR) - kR*cos(kR))/pow(kR, 3.0);
61      return w*w;
62  }
63
64  // Perpendicular component:
65  double perp(double k, void* params)
66  {
67      struct myparams *mu = (struct myparams *)params;
68      double P = gsl_spline_eval(mu->spline, k, mu->acc);
69      double x = mu->c;
70      double kx = k*x;
71      double j1_0 =
72              (kx*cos(kx) - sin(kx))/(kx*kx);//j'_0(kx)
73      // j_0(kx) = sin(kx)/kx.
74
75      double f2 = pow(mu->b, 2.0*0.6);
76      double H02 = pow(mu->a, 2.0);
77      double f = -P*f2*H02*j1_0/(2*M_PI*M_PI*kx);
78
79      return f;
80  }
81
82  // Parallell component:
83  double para(double k, void* params)
84  {
85      struct myparams *mu = (struct myparams *)params;
86      double P = gsl_spline_eval(mu->spline, k, mu->acc);
87      double x = mu->c;
88      double kx = k*x;
89      double j2_0 =
90              2*sin(kx)/pow(kx, 3.0) - sin(kx)/kx - 2*cos(kx)/(kx*kx);
    //j''_0(kx)
91      // j_0(kx) = sin(kx)/kx.
92
```

```
93      double f2 = pow(mu->b, 2.0*0.6);
94      double H02 = pow(mu->a, 2.0);
95      double f = -P*f2*H02*j2_0/(2*M_PI*M_PI);
96      /*cout << "P: " << P << "\n";
97      cout << "k: " << k << "\n";
98      cout << "r: " << x << "\n";
99      cout << "f2: " << f2 << "\n";
100     cout << "H02: " << H02 << "\n";
101     cout << "j2_0: " << j2_0 << "\n";
102     cout << "para: " << f << endl;
103     */
104     return f;
105 }
106
107 // Correlation function without constants:
108 double Icorrfunc(double k, void* params)
109 {
110     struct myparams *mu = (struct myparams *)params;
111     double P = gsl_spline_eval(mu->spline, k, mu->acc);
112     double kr = k*(mu->c);
113     double xi = k*k*P*sin(kr)/(kr*2*M_PI*M_PI);
114
115     return xi;
116 }
117
118 double IxiBar(double x, void* params)
119 {
120     struct myparams *mu = (struct myparams *)params;
121     double xi = gsl_spline_eval(mu->spline, x, mu->acc);
122     double r = (mu->c);
123     double xiB = 3.0*xi*x*x/pow(r, 3.0);
124
125     return xiB;
126 }
127
128 double Iv_12(double k, void* params)
129 {
130     struct myparams *mu = (struct myparams *)params;
131     double P = gsl_spline_eval(mu->spline, k, mu->acc);
132     double kr = k*(mu->c);
133     double H_0 = mu->a;
134     double j_1 = sin(kr)- kr*cos(kr)/(kr*kr);
135     double I = k*k*P*j_1/kr;
136
137     return I;
138
139 }
```

A.4.2 calc.cpp

```
using std::cout; using std::endl;
using std::vector; using std::string;

vector<double> linVeloCorr(vector<vector<double> > halos, vector<
    double> o, double L, double r_min, double r_max, int binsize,
    spline *para, spline *perp)
{
    gsl_interp_accel* para_acc = para->acc; gsl_spline* para_spline
    = para->splined;
    gsl_interp_accel* perp_acc = perp->acc; gsl_spline* perp_spline
    = perp->splined;
    int L_int = sqrt(3*L*L); // Maximum distance between two halos
    in the box.
    int N_bins = L_int/binsize + 1; // Number of distance bins.

    vector<double> xi(N_bins);
    vector<double> cnt(N_bins);

    int bin; double xpara, xperp;
    vector<double> thetas;
    vector<double> p1(3);
    vector<double> p2(3);
    vector<double> A(N_bins);
    vector<double> B(N_bins);
    vector<double> para_comp(N_bins);
    vector<double> perp_comp(N_bins);
    double A_temp; double B_temp;
    int N_halos = halos.size();
    for (int i = 0; i != N_halos; ++i)
    {
        //cout << "i: " << i << endl;
        if (i%100 == 0)
            cout << i << endl;
        for (int j = 0; j != i; ++j)
        {
            //cout << "j: " << j << endl;
            p1 = {halos[i][0], halos[i][1], halos[i][2]};
            p2 = {halos[j][0], halos[j][1], halos[j][2]};
            thetas = anglepos(p1, p2, o, L);
            //cout << r_max << "c " << thetas[2] << " i, j " << i <<
    ", " << j << endl;
            if (thetas[2] < r_min or thetas[2] > r_max)
                continue;
            else
            {
                xpara = gsl_spline_eval(para_spline, thetas[2],
    para_acc);
```

```
41                  xperp = gsl_spline_eval(perp_spline, thetas[2],
    perp_acc);
42                  bin = thetas[2]/binsize;
43                  A_temp = sin(thetas[0])*sin(thetas[1]);
44                  B_temp = cos(thetas[0])*cos(thetas[1]);
45                  A[bin] += A_temp;
46                  B[bin] += B_temp;
47                  perp_comp[bin] += A_temp*xperp;
48                  para_comp[bin] += B_temp*xpara;
49                  xi[bin] += A_temp*xperp + B_temp*xpara;
50                  cnt[bin] += 1;
51              }
52          }
53      }
54      cout << " after " << endl;
55
56      for (int i = 0; i != N_bins; ++i)
57      {
58          if (cnt[i] != 0)
59          {
60              xi[i] = xi[i]/cnt[i];
61              A[i] = A[i]/cnt[i]; B[i] = B[i]/cnt[i];
62              perp_comp[i] = perp_comp[i]/cnt[i];
63              para_comp[i] = para_comp[i]/cnt[i];
64          }
65      }
66
67      // Output of linear constants:
68      vector<vector<double> > Out;
69      vector<double> x(N_bins);
70      double dx = sqrt(3*L*L)/(N_bins-1);
71      for (int i = 0; i != N_bins; ++i)
72      {
73          x[i] = i*dx;
74          Out.push_back({x[i], A[i], B[i], perp_comp[i], para_comp[i
    ]});
75      }
76      writeMatrix("components.dat", Out);
77
78      return xi;
79  }
```

A.5 General tool set

A.5.1 functions.h

```
1  #ifndef FUNCTIONS_H
2  #define FUNCTIONS_H
3  #include <iostream>
4  #include <fstream>
5  #include <sstream>
6  #include <string>
7  #include <vector>
8  #include <math.h>
9  #include <stdlib.h>
10 #include <time.h>
11 #include <omp.h>
12 #include <random>
13
14 double int_trapez(double (*dydx)(double x, double y), double x_min,
       double x_max, double y_ini, int N, double R);
15 void load(const char* filename, double **A, int N_r, int N_c, int *
       columns,      const char* folder = "");
16 std::vector<std::vector<double> > load(std::string filename, int N_c
       ,
17          std::vector<int> columns, int header = 0, std::string folder
        = "");
18 int lineCount(std::string fileloc);
19
20 // Writing to file:
21 void write(const char* filename, double *x, double *A, int Nhalos,
       const char* folder = "");
22 void writeMatrix(std::string filename, std::vector<std::vector<
       double> > A);
23
24 double rfind(double x1, double y1, double z1, double x2, double y2,
       double z2, double boxsize, int trigger=0);
25 double r2find(double x1, double y1, double z1, double x2, double y2,
        double      z2, double boxsize, int trigger=0);
26
27 inline
28 double dot_product(std::vector<double> x, std::vector<double> y);
29 inline
30 double angle_vector(std::vector<double> x, std::vector<double> y);
31 inline
32 double vec_length(std::vector<double> x);
33 inline
34 std::vector<double> vec_add(std::vector<double> x, std::vector<
       double> y);
35 inline
```

```
36 std::vector<double> vec_subtract(std::vector<double> x, std::vector<
       double> y);
37
38 double variance(double *x, int N);
39
40 #include "functions.cpp"
41 #endif
```

A.5.2 functions.cpp

```
1  #include "functions.h"
2  //using namespace std;
3
4  double int_trapez(double (*dydx)(double x, double y), double x_min,
       double x_max, double y_ini, int N, double R)
5  {
6      double dy;
7      double dx = (x_max - x_min)/(double)(N-1);
8      double y = y_ini;
9      double x = x_min;
10     for (int i = 0; i<N-1; i++)
11     {
12         dy = (dydx(x, R) + dydx(x+dx,R))/2.0;
13         y += dy*dx;
14         x += dx;
15     }
16     return y;
17 }
18
19
20 void load(const char* filename, double **A, int N_r, int N_c, int *
       columns, const char* folder)
21 {
22     std::string line;
23     int rows, cols, col_i;
24     double dummy;
25     //double A[44524][10];
26     std::stringstream fileandfolder;
27     fileandfolder << folder << filename;
28     std::ifstream pFile (fileandfolder.str().c_str());
29     if (pFile.is_open())
30     {
31         rows = 0;
32         while(rows < N_r) //!pFile.eof()
33         {
34             getline(pFile, line);
35             std::stringstream ss(line);
36             //cout << rows << "\n";
37             cols = 0;
38             col_i = 0;
39             while(col_i < N_c) // ss >> A[rows][cols]
40             {
41                 //std::cout << rows << " " << cols << " " << col_i
       << std::endl;
42                 if (cols == columns[col_i])
43                 {
44                     ss >> A[rows][col_i];
45                     col_i++;
```

```
46                  }
47                  else {
48                      ss >> dummy;}
49                  //cout << cols << "\n";
50                  cols++;
51              }
52              rows++;
53          }
54          pFile.close();
55      }
56      else
57          std::cout << "Unable to open file: " << filename << std::
        endl;
58  }
59
60  std::vector<std::vector<double> > load(std::string filename, int N_c
        , std::vector<int> columns, int header, std::string folder)
61  {
62      std::string line;
63      int rows, cols, col_i;
64      double dummy;
65
66      std::string fileandfolder;
67      fileandfolder = folder + filename;
68      std::ifstream pFile (fileandfolder.c_str());
69      if (pFile.is_open())
70      {
71          std::vector<std::vector<double> > A;
72          rows = 0;
73          std::vector<double> inivec(N_c);
74          for (int i = 0; i<header; ++i)
75              getline(pFile, line);
76          while(!pFile.eof()) //!pFile.eof()
77          {
78              getline(pFile, line);
79              std::stringstream ss(line);
80              //cout << rows << "\n";
81              cols = 0;
82              col_i = 0;
83              //std::cout << "A.size(): " << A.size() << std::endl;
84              A.push_back(inivec);
85              while(col_i < N_c) // ss >> A[rows][cols]
86              {
87                  if (cols == columns[col_i])
88                  {
89
90                      ss >> A[rows][col_i];
91                      ++col_i;
92                  }
93                  else {
```

```
94                      ss >> dummy;}
95                  //cout << cols << "\n";
96                  ++cols;
97              }
98              ++rows;
99          }
100         pFile.close();
101         A.pop_back();
102         //std::cout << "rows: " << rows << " " << A.size() << std::
    endl;
103         return A;
104     }
105     else {
106         std::cout << "Unable to open file: " << filename << std::
    endl;
107     }
108 }
109
110 int lineCount(std::string fileloc)
111 {
112     std::string line;
113     std::ifstream pFile (fileloc.c_str());
114     if (pFile.is_open())
115     {
116         int lcnt = -1;
117         while(!pFile.eof())
118         {
119             getline(pFile, line);
120             ++lcnt;
121             //std::cout << lcnt << std::endl;
122         }
123         return lcnt;
124     }
125     else
126     {
127         std::cout << "Could not open file given to lineCount." <<
    std::endl;
128         return -1;
129     }
130 }
131
132 void write(const char* filename, double *x, double *A, int Nhalos,
    const char* folder)
133 {
134     std::stringstream fileandfolder;
135     fileandfolder << folder << filename;
136     std::ofstream file(fileandfolder.str().c_str());
137     for (int i = 0; i<Nhalos; i++)
138     {
139         file << x[i] << " " << A[i] << std::endl;
```

```
140     }
141     file.close();
142 }
143
144 void writeMatrix(std::string filename, std::vector<std::vector<
    double>> A)
145 {
146     std::ofstream file(filename);
147
148     for (auto ittA = A.begin(); ittA != A.end(); ++ittA)
149     {
150         for (auto itsubA = ittA->begin(); itsubA != ittA->end(); ++
    itsubA)
151             file << *itsubA << " ";
152         file << std::endl;
153     }
154     file.close();
155 }
156
157 double rfind(double x1, double y1, double z1, double x2, double y2,
    double z2, double boxsize, int trigger)
158 {
159     double xdist = x2-x1;
160     double ydist = y2-y1;
161     double zdist = z2-z1;
162     double boxsize_half = 0.5*boxsize;
163
164     if (trigger != 0) {
165     //Correct for periodic boundary conditions:
166     if (xdist > boxsize_half) xdist -= boxsize;
167     else if (xdist < -boxsize_half) xdist += boxsize;
168
169     if (ydist > boxsize_half) ydist -= boxsize;
170     else if (ydist < -boxsize_half) ydist += boxsize;
171
172     if (zdist > boxsize_half) zdist -= boxsize;
173     else if (zdist < -boxsize_half) zdist += boxsize;}
174
175     //std::cout << "xdist, ydist, zdist: " << xdist << ", " << ydist
    << ", " << zdist << std::endl;
176     double r = sqrt(xdist*xdist + ydist*ydist + zdist*zdist);
177     return r;
178 }
179
180 double r2find(double x1, double y1, double z1, double x2, double y2,
     double z2, double boxsize, int trigger)
181 {
182     double xdist = x2-x1;
183     double ydist = y2-y1;
184     double zdist = z2-z1;
```

```
185     double boxsize_half = 0.5*boxsize;
186
187     if (trigger != 0) {
188     //Correct for periodic boundary conditions:
189     if (xdist > boxsize_half) xdist -= boxsize;
190     else if (xdist < -boxsize_half) xdist += boxsize;
191
192     if (ydist > boxsize_half) ydist -= boxsize;
193     else if (ydist < -boxsize_half) ydist += boxsize;
194
195     if (zdist > boxsize_half) zdist -= boxsize;
196     else if (zdist < -boxsize_half) zdist += boxsize;}
197
198     double r = xdist*xdist + ydist*ydist + zdist*zdist;
199     return r;
200 }
201
202 double dot_product(std::vector<double> x, std::vector<double> y)
203 {
204     double dp = 0;
205     auto N = x.size();
206     if (N != y.size())
207         std::cerr << "To take the dot product between two vectors
    they need to be of an equal number of elements." << std::endl;
208     for (decltype(N) i = 0; i<N; ++i)
209     {
210         dp += x[i]*y[i];
211     }
212
213     return dp;
214 }
215
216 double angle_vector(std::vector<double> x, std::vector<double> y)
217 {
218     double theta = acos(dot_product(x, y)/(vec_length(x)*vec_length(
    y)));
219     //std::cout << "Inside angle_vector: " << std::endl;
220     //std::cout << vec_length(x) << std::endl;
221     //std::cout << vec_length(y) << std::endl;
222     //std::cout << dot_product(x, y) << std::endl;
223
224     return theta;
225 }
226
227 double vec_length(std::vector<double> x)
228 {
229     double sum = 0.0;
230     for (decltype(x.size()) i = 0; i<x.size(); ++i)
231     {
232         sum += x[i]*x[i];
```

```
233     }
234
235     return sqrt(sum);
236 }
237
238 std::vector<double> vec_add(std::vector<double> x, std::vector<
    double> y)
239 {
240     auto N = x.size();
241     std::vector<double> z(N);
242     if (N != y.size())
243         std::cerr << "Trying to add together two vectors with a
    different number of elements in them!" << std::endl;
244     for (decltype(N) i = 0; i != N; ++i)
245         z[i] = x[i] + y[i];
246     return z;
247 }
248
249 std::vector<double> vec_subtract(std::vector<double> x, std::vector<
    double> y)
250 {
251     auto N = x.size();
252     std::vector<double> z(N);
253     if (N != y.size())
254         std::cerr << "Trying to add together two vectors with a
    different number of elements in them!" << std::endl;
255     for (decltype(N) i = 0; i != N; ++i)
256         z[i] = x[i] - y[i];
257     return z;
258 }
259
260 double variance(double *x, int N)
261 {
262     double mean = 0.0;
263     double sum = 0.0;
264     for (int i = 0; i<N; i++)
265     {
266         sum += x[i];
267     }
268     mean = sum/(double)N;
269     std::cout << "Mean: " << mean << std::endl;
270     sum = 0.0;
271     double dummy;
272     for (int i = 0; i<N; i++)
273     {
274         dummy = x[i] - mean;
275         sum += dummy*dummy;
276     }
277     double var = sum/(double)N;
278     return var;
```

```
279 }
280 /*
281 double rng_uniform_real(double min, double max)
282 {
283     mt19937 generator(42u);
284     generator.seed(static_cast<unsigned int>(time(NULL)));
285     boost::uniform_real<> uni_dist(min,max);
286     boost::variate_generator<mt19937&, boost::uniform_real<> > uni(
    generator, uni_dist);
287     return uni();
288 }
289 */
```

A.5.3 statistics.h

```
1  #ifndef STATISTICS_H
2  #define STATISTICS_H
3
4  #include <iostream>
5  #include <vector>
6  #include <string>
7  #include <random>
8  #include <algorithm>
9
10 #include <boost/random/uniform_int.hpp>
11 #include <boost/random/variate_generator.hpp>
12
13 bool seeded = false;
14
15 double jackknife(std::vector<double> v, double& jackmean);
16 double bootstrap(std::vector<double> v, double& bootmean, std::
       string option = "asymm");
17
18 #include "statistics.cpp"
19 #endif
```

A.5.4 statistics.cpp

```
1  double jackknife(std::vector<double> v, double& jackmean)
2  {
3      auto M = v.size(); double sum;
4      //std::cout << M << std::endl;
5      double dp = 0;
6      jackmean = 0;
7      std::vector<double> p_m(M);
8      for (decltype(M) i = 0; i<M; ++i)
9      {
10         sum = 0;
11         for (decltype(M) j = 0; j<M; ++j)
12         {
13             if (j != i)
14                 sum += v[j];
15             else
16                 continue;
17         }
18         p_m[i] = sum/(M-1);
19         jackmean += p_m[i];
20     }
21     jackmean = jackmean/M;
22     double fac;
23     for (decltype(M) i = 0; i<M; ++i)
24     {
25         fac = p_m[i] - jackmean;
26         dp += fac*fac;
27     }
28     dp = sqrt((double)(M-1)*dp/(double)M);
29
30     return dp;
31 }
32
33 double bootstrap(std::vector<double> v, double& bootmean, std::
       string option)
34 {
35     int straps = 10000;
36     int N = v.size();
37     std::vector<double> n(N);
38     std::vector<double> p(straps);
39     //std::vector<int> indicies(N);
40     std::mt19937 generator(42u);
41     if (!seeded)
42     {
43         generator.seed(static_cast<unsigned int>(time(NULL)));
44         seeded = true;
45         std::cout << "Seeded uniform RNG engine in STATISTICS_H." <<
       std::endl;
46     }
```

```
47      boost::uniform_int<> uni_dist(0, N-1);
48      boost::variate_generator<std::mt19937&, boost::uniform_int<> >
        uni(generator, uni_dist);
49
50      double p_m;
51      for (int i = 0; i != straps; ++i)
52      {
53          //if (i%100 == 0) std::cout << i << std::endl;
54          p_m = 0;
55          for (int j = 0; j != N; ++j)
56          {
57              p_m += v[uni()];
58          }
59          p_m = p_m/N;
60          p[i] = p_m;
61      }
62
63      std::sort(p.begin(), p.end());
64
65      int a_int = 0.16*(double)straps; int b_int = 0.84*(double)straps
        ;
66      double std;
67      if (option == "symm")
68          std = (p[b_int] - p[a_int])/2.0;
69      else if (option == "asymm")
70          std = p[a_int];
71          bootmean = p[b_int];
72
73      return std;
74  }
75
76  /*
77  double bootstrap(std::vector<std::vector<double> > v, double&
        bootmean, std::string option)
78  {
79      int straps = 10000;
80      int N = v.size();
81      std::vector<double> n(N);
82      std::vector<double> p(straps);
83      //std::vector<int> indicies(N);
84      std::mt19937 generator(42u);
85      if (!seeded)
86      {
87          generator.seed(static_cast<unsigned int>(time(NULL)));
88          seeded = true;
89          std::cout << "Seeded uniform RNG engine in STATISTICS_H." <<
        std::endl;
90      }
91      boost::uniform_int<> uni_dist(0, N-1);
```

```
92      boost::variate_generator<std::mt19937&, boost::uniform_int<> >
        uni(generator, uni_dist);
93
94      double p_m;
95      for (int i = 0; i != straps; ++i)
96      {
97          //if (i%100 == 0) std::cout << i << std::endl;
98          n[i] = v[uni()];
99          p_m = p_m/N;
100         p[i] = p_m;
101     }
102
103     std::sort(p.begin(), p.end());
104
105     int a_int = 0.16*(double)straps; int b_int = 0.84*(double)straps
        ;
106     double std;
107     if (option == "symm")
108         std = (p[b_int] - p[a_int])/2.0;
109     else if (option == "asymm")
110         std = p[a_int];
111         bootmean = p[b_int];
112
113     return std;
114 }
115 */
```