Laniakea in a Cosmological Context

or

Detection of galaxy superclusters in simulated cosmological structures

Sergio Daniel Hernández Charpak
200922618

Advisor: Jaime E. Forero-Romero

May 5, 2016

Universidad de los Andes
Facultad de Ciencias, Departamento de Física
Bogotá, Colombia

1 Acknowledgements

I would like to thank my advisor, Jaime E. Forero-Romero, for his guidance during this process, of course my family, Nathalie, Jose Tiberio, Yván David, Gabriel Luis, Nico, Dani and all the others for their great patience and always good advices. I also acknowledge the University of Los Andes for its mayor financial support during my studies. Finally I couldn't have finish this without the support of Andrés, Nicolás, Jorge and all the special friends I am lucky to have.

2 Abstract

Recently Tully et al. (2014) [1] used local cosmic flow information to define our local supercluster, Laniakea. In this work we present a study on large cosmological N-body simulations aimed at establishing the significance of Laniakea in a cosmological context. We explore different algorithms to define superclusters from the dark matter velocity field in the simulation. We summarize the properties of the supercluster population by their abundance at a given total mass and its shape distributions. We find that superclusters similar in size and structure to Laniakea are relatively uncommon on a broader cosmological context. We finalize by discussing the possible sources of systematics (both in our methods and in observations) leading to this discrepancy.

Contents

3 Introduction

In the Universe scene at large scales the galaxy group themselves in structures similar to a filament web which go through large voids regions and cross in regions called superclusters. Even though these structures can be easily detected at simple view, there are different possibilities to delimited them from physical criteria [2].

A proposal to define a supercluster is to use the flow of galaxy in this region of space. Within a supercluster, galaxy tend to flow to the most dense region as a consequence of the gravitational attraction process. In this way, spacial regions where the galaxy flow is convergent represent galaxy superclusters.

Recently a team build a galaxy velocities flow map of the local group in a scale of hundreds of light years [3]. In this map converging points were found and this team identified Laniakea, the galaxy supercluster which includes our galaxy, the milky way [1].

We propose to develop a method to detect a statistical significant number of galaxy supercluster in cosmological simulations. With this we aim to quantify if Laniakea can be considered as an atypical structure in the Universe.

4 Background

4.1 Cosmic flows and peculiar velocities

The peculiar velocity refers to the velocity relative to a rest frame. In the case of the Cosmicflows-2 map, the rest frame is the earth. The Laniakea supercluster of galaxy was mainly identified following an analysis of the peculiar velocities flows. A Wiener filtering method was apply to obtained a better Signal/Noise proportion and separate the peculiar velocities from the cosmic expansion.

4.2 Boundaries of Laniakea - The V-Web Algorithm

The contour of the region were reconstructed with the V-web Algorithm. This algorithm is deeply connected to the velocities as its core is the shear velocity tensor.

4.3 Fractional Anisotropy and Overdensity

5 Methods

As we support the open-source community, we used the N-Body simulation open software Gadget-2 [4] to generate the simulations and source code in C and Python to treat the data obtained from the simulations. The development of the code will be in Github `https://github.com/sercharpak/Monografia/`.

5.1 Simulations

We used the N-Body simulation software Gadget-2 [4] widely used in the scientific community to generate different boxes of dark matter particles (DM). The initial conditions were generated with Springel's N-Genic software. The simulations ran on the HPC cluster at UNIANDES. The main properties (boxsize, number of particles, number of CPUs used and time of the simulation) are described in table 1.

Boxsize [Mpc/h]	DM particles	# CPU	Time [hours]
500	512^3	48	$\tilde{1}0$
250	512^3	48	$\tilde{6}$
500	1024^3	48	$\tilde{7}0$

Table 1: Different Gadget-2 Simulations

By DM particle we mean a particle which only interacts through gravity interaction. In the simulation each particle represents a galaxy. The DM particles are placed in a 3D grid layout first and then are perturbed following Poisson distribution before the simulation start, during the initial conditions generation.

Once the simulation starts, the particles interact only with gravity as time goes by following the cosmological constants. We following are an approximation of the ΛCDM (Lambda cold dark matter) or Lambda-CDM model, our current cosmological model. The involved cosmological constants are described in table 2

.

Ω_0	Ω_Λ	Ω	H
0.3	0.7	0	0.7

Table 2: Cosmological Constants in the Gadget-2 Simulations

Ω_0 corresponds to the Cosmological matter density parameter in units of the critical density at z = 0. Ω_Λ is the cosmological vacuum energy density (cosmological constant) in units of the critical density at z = 0. For a geometrically flat universe, as ours, one has $\Omega_0 + \Omega_\Lambda = 1$. Ω is the baryon density in units of the critical density at z = 0. In our case it is not relevant. Finally, H denotes the Hubble constant at z=0 in units of 100 $km\ s^{-1}\ Mpc^{-1}$. It will be relevant in the discussions at the end of this work.

We then use different algorithms to identify the superclusters within the simulation.

5.2 Approaches

We mainly had two approaches in this work for the search of superclusters. First, a naive one involving the speed very easy to implement 5.2.1. And a second one, more complete, using other concepts 5.2.2.

5.2.1 Naive: the magnitude of the velocity

In a first approach we analyse the distribution of the magnitude of the velocity in the different simulations. We work with all the particles and points given by the simulation. As our final results are not based in this naive approach we leave it as an appendix here. It is explained in further detail in Appendix A.

5.2.2 More complete: V-Web, CIC, the FA and δ

In our second approach, we used directly the V-Web algorithm to form the Fractional Anisotropy (FA) and the Overdensity δ) to have detailed results comparable to Laniakea's.
Finally we apply a Cloud In Cell (CIC) method and a Gaussian smooth to have a more manipulable grid to work with.

We described the V-Web algorithm in section 4.2. Let us remind the velocity shear tensor as

The CIC method is used to transform the box to a grid in which the (i,j,k) have λ_1, λ_2, λ_3 values related to the number of particles in a cell. Each cell has a number of particles in it and hence a total mass equivalent to the sum of the mass of its particles.

And in section 4.3 we saw how the FA and δ are formed and are relevant in this context. We are then looking for regions with δ positive and FA lesser than 1.0.

Our approach is hence very close with the approach used to define Laniakea [1] which is important to validate our later comparison of Laniakea with our detected structures.

5.3 Algorithms

Our main algorithm is a region growing algorithm as described by Gonzalez [5] mainly used in image segmentation. We now proceed to describe it generally.

5.3.1 Region Growing Algorithm: General Description

5.3.2 Region Growing Algorithm with the FA and Overdensity

Our implementation is based in the FA and the Overdensity δ.

5.3.3 Region Growing Algorithm and FoF with the FA and Overdensity

Finally we improved this last method making used of the Friends-of-Friends (FoF) algorithm.

6 Results

6.1 Region Growing Algorithm with the FA and Overdensity

6.2 Region Growing Algorithm and FoF with the FA and Overdensity

6.3 Comparisons with Laniakea

7 Conclusions

8 Future Work

- Understand more in detail how the work done by Hoffman et al[7] in the V-web algorithm, was used to determine Laniakea.
- Write our own Gadget-2 snapshop reader in C or Python which implements the region growing algorithm.
- Optimize our Algorithm. The current results are not sufficient.
- Run a bigger simulation (boxsize: $5 \times 10^6 Mpc/h$ and 1024^3 DM particles.
- Run the algorithm on this simulation.
- Compare the properties of the results with the Laniakea supercluster.
- Discuss the sources of systematics in both methods and observations.

References

[1] R. Brent Tully, Hlne Courtois, Yehuda Hoffman, and Daniel Pomarde. The Laniakea supercluster of galaxies. *Nature*, 513(7516):71–73, September 2014.

[2] J. Richard Gott III, Mario Juri, David Schlegel, Fiona Hoyle, Michael Vogeley, Max Tegmark, Neta Bahcall, and Jon Brinkmann. A Map of the Universe. *The Astrophysical Journal*, 624(2):463–484, May 2005.

[3] R. Brent Tully, Hlne M. Courtois, Andrew E. Dolphin, J. Richard Fisher, Philippe Hraudeau, Bradley A. Jacobs, Igor D. Karachentsev, Dmitry Makarov, Lidia Makarova, Sofia Mitronova, Luca Rizzi, Edward J. Shaya, Jenny G. Sorce, and Po-Feng Wu. *COSMICFLOWS-2* : THE DATA. *The Astronomical Journal*, 146(4):86, October 2013.

[4] V. Springel. The cosmological simulation code gadget-2. *Monthly Notices of the Royal Astronomical Society*, 364(4):1105–1134, 2005.

[5] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Prentice Hall, 3rd ed edition.

[6] Yehuda Hoffman, Ofer Metuki, Gustavo Yepes, Stefan Gottlber, Jaime E. Forero-Romero, Noam I. Libeskind, and Alexander Knebe. A kinematic classification of the cosmic web: The cosmic web. *Monthly Notices of the Royal Astronomical Society*, 425(3):2049–2057, September 2012.

[7] R. Thompson. pyGadgetReader: GADGET snapshot reader for python. Astrophysics Source Code Library, November 2014.

A Approach by the Speed

A.1 Region Growing Algorithm with the Speed

We present here a naive way to approach to the problem.

1. We calculate the magnitude of the velocity (the speed) of each particle.

2. We look for regions where the center has the highest speed and from it the speed decreases while the particle is further away from the center.

3. We define the limits of this region the regions where the speed begins to increase with the distance from the center.

We use a region growing algorithm, a simple image segmentation algorithm used to identify different regions in a image [5].

$f(x, y, z)$ denotes the input data. $S(x, y, z)$ denotes a seed array, with value 1 where the particle can be define as a seed and 0 where not. It is the same size of $f(x, y, z)$. Q denotes a predicate which is to be apply to the input data and determines if the region which starts at the seeds grows or not. Here Q denotes: "if the speed of the close particle is lower than the marked one but greater than a threshold, mark that particle and continue."

1. We choose seeds to begin the growth. In this case we choose the particles with high velocities. Here
$$|v| > th_{high}$$

2. We open a window of particles to look for 2 close particles from the seed s which do not for part of $S(x, y, z)$. Here:
$$window = 20000$$

3. Once we found the 2 close particles c we apply the predicate Q. Here:
$$Q := |v_s| > |v_c| > |th_{low}|$$

4. If the particle c satisfies Q, it is marked, $S(c) = 1$

5. Apply the algorithm to c, and so on (Recursion).

Here we defined empirically different thresholds in order to observer the results.
First we used:
$th_{low} = v_{min} + \frac{\sigma_v}{2}$ and $th_{high} = v_{max} - \sigma_v$.
Secondly we used:
$th_{low} = \vec{v} + 2\sigma_v$ and $th_{high} = \vec{v} + 8\sigma_v$.

The first version of the algorithm was written in python using the module pyGadgetReader [6] to read the data and transform it to NumPy arrays in Python. In Appendix A.3 we attach the source code used.

A.2 Results of the Speed Approach

This approach was tested on the simulation of boxsize 500 Mpc/h and 512^3 dark matter particles (DM) described in table 1.

We first visualize the speed histogram in figure 1.

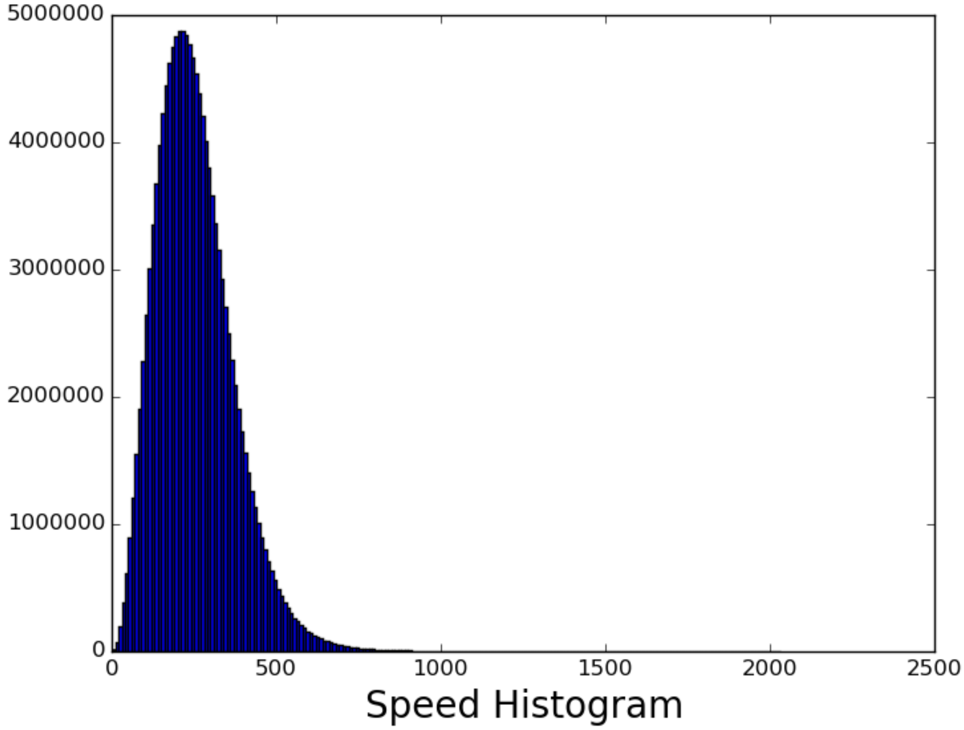

Figure 1: Speed Histogram of the simulation

The histogram resembles a Poisson distribution. This is a consequence of the perturbations generated in the initial conditions generation.

v_{max}	2023.87
v_{min}	0.34
σ_v	117.75

Table 3: Properties of the speed distribution

Our hypothesis is based on the most direct implication of gravity. The lower speed particles will mainly be in the border regions and the higher speed particles will be in the center regions.

For this we must choose a threshold for low velocities and a threshold for high velocities. We obtain: $th_{low} = v_{min} + \frac{\sigma_v}{2} = 59.2$ and $th_{high} = v_{max} - \sigma_v = 1906.12$

We first have a look at the particle distribution of particles with speed higher than the threshold for low velocities.

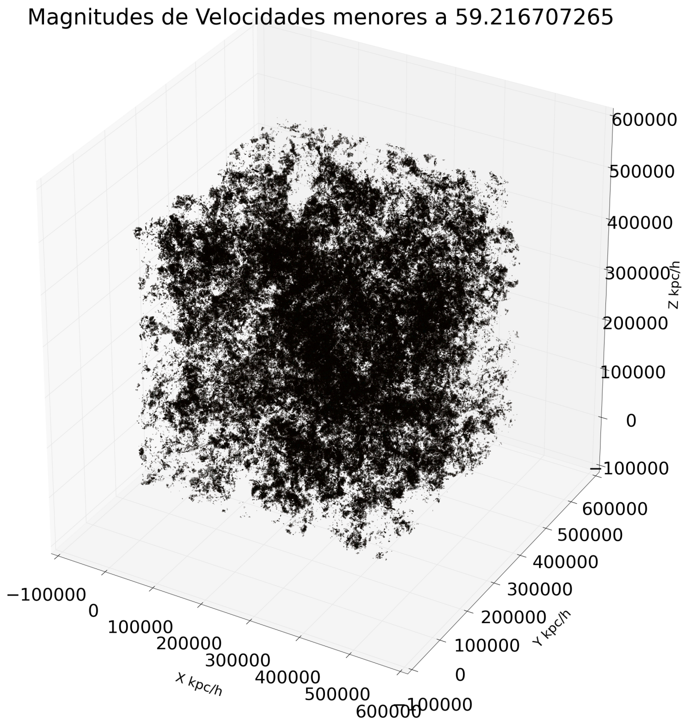

Figure 2: DM particles with $|v| < 59.2$

There seem to be some structures, but it is not clear enough. This is due to the high number of DM particles used in the simulation (512^3).

We produce cuts in the z-direction to visualize in 2-D the speed distribution.

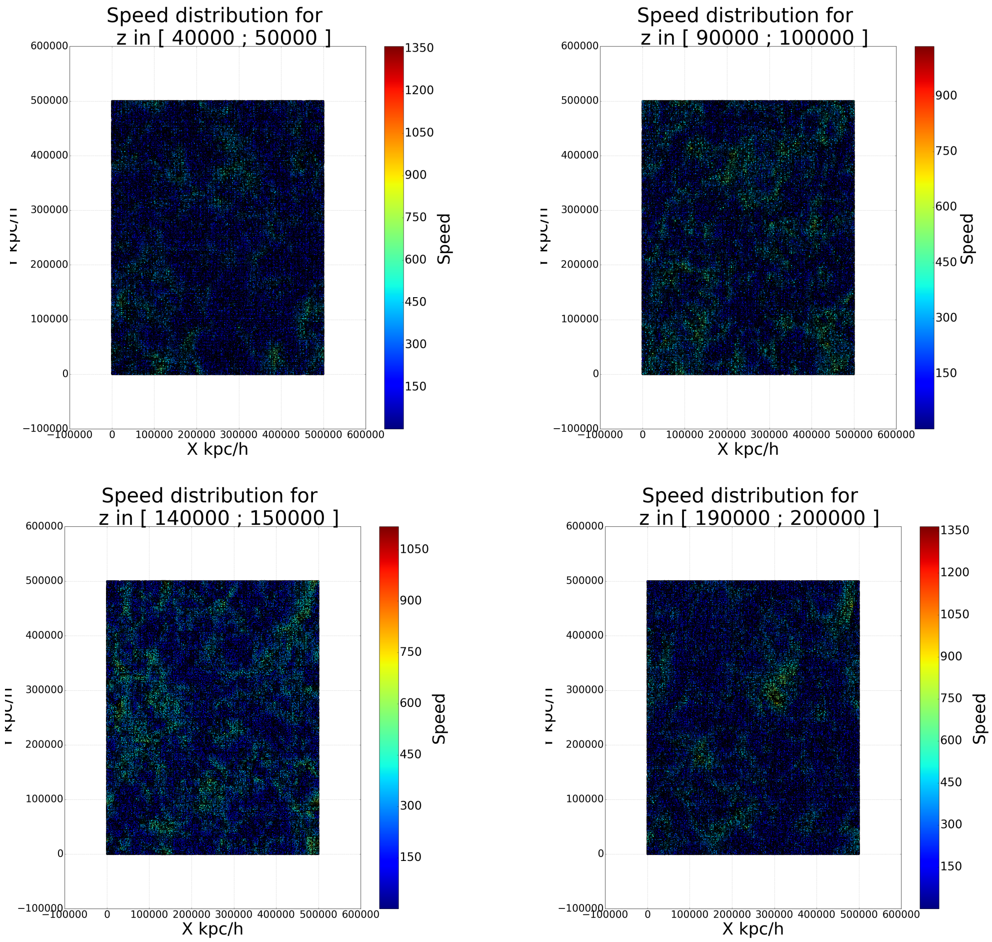

Figure 3: 2D slices of the speed distribution

We can observe in figure 3 intuitive signs of structures related to the speed distribution. We can observe filament structures

It is more clear if we observe from one side the particles with speed greater than a th_{high} and the particles with speed lesser than a th_{low}.

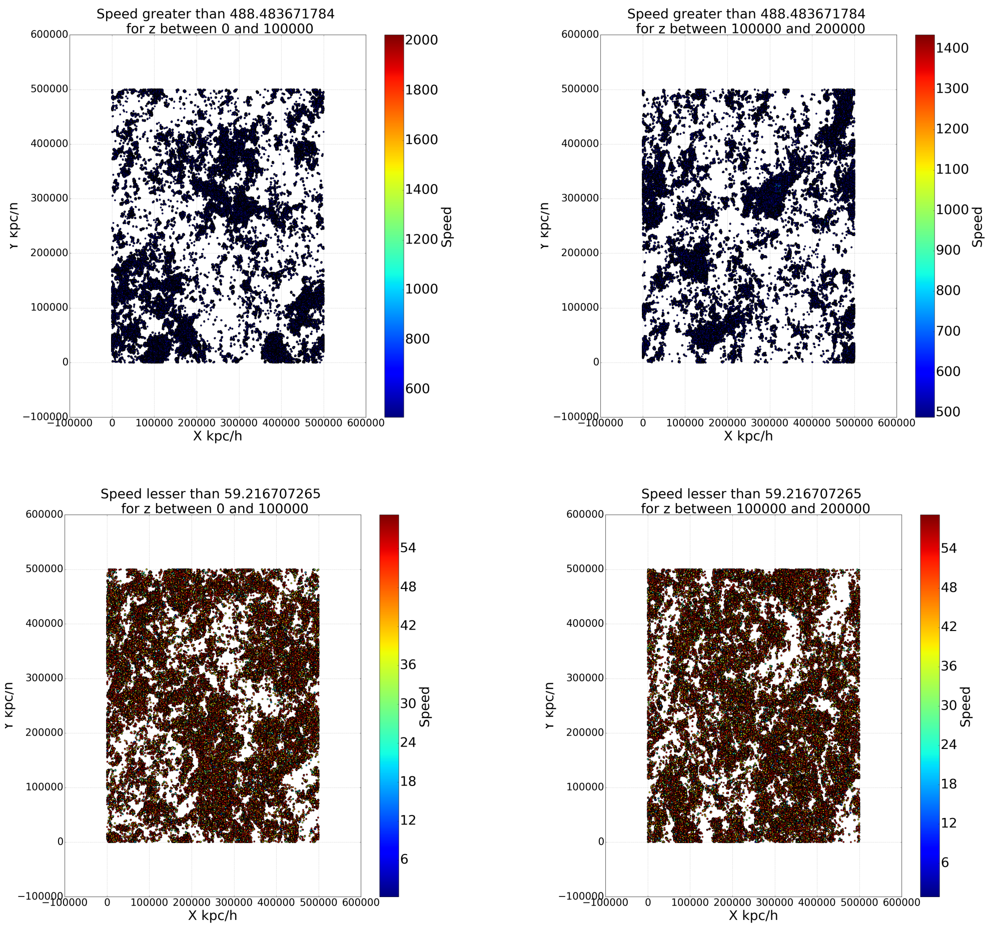

Figure 4: 2D slices of the speed distribution. The upper slices with $|\vec{v}| > th_{high} = 488$ and the two lower slices with $|\vec{v}| < th_{low} = 59$ for the same cuts, $z = \{100, 200\} Mpc/h$ and

As we can see in figure 4 the cuts with low and high thresholds are complementary. It is evident that structures are present.

Finally we apply our region growing algorithm and we obtain the following results.

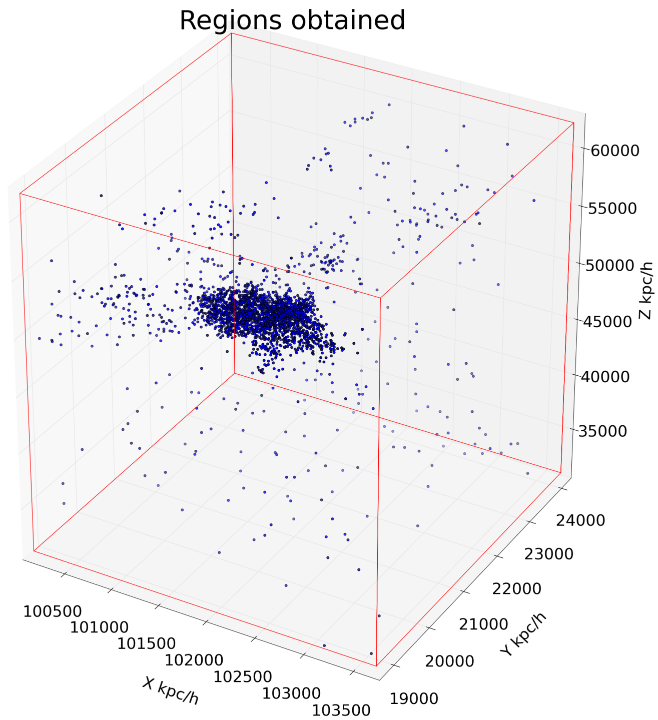

Figure 5: Region identified by the algorithm

And, changing the thresholds to: $th_{low} = \bar{v} + 2\sigma_v = 488.48$ and $th_{high} = \bar{v} + 8\sigma_v = 1195.0$
We obtain:

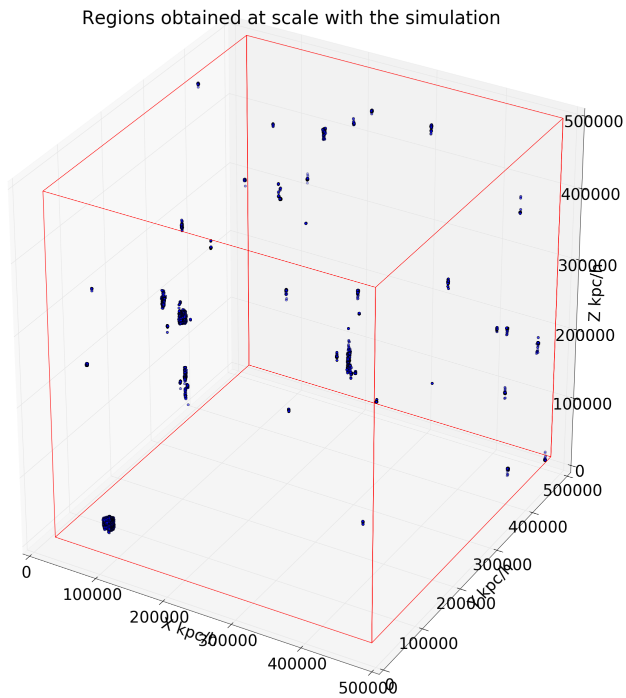

Figure 6: Region identified by the algorithm with the second thresholds

We have different observations:

1. The details of the regions in figures 5 and 6 seem accurate with what we would expect. It is more dense in the center and less dense in the borders.

2. The regions obtained have a dimension of only a few Mpc/h (Laniakea's dimension is two orders of magnitude higher). This could mean that Laniakea is atypical.

3. There is first only one region identified. This indicates than all the high velocity particles are congregated in only one region (the detected region). If we set lower th_{high} we can see we get more regions.

4. This is certainly a good naive approach to the problem. Nevertheless for more accurate results it needs to be drastically changed.

The script in Appendix A.3 was executed in a student's laptop and later on the HPC cluster. For a significant number of particles, the laptop limited memory (8GB) crashes. Running in the HPC solves this issue. results.

A.3 Code example for Region Growing Algorithm - Approach by the Speed

Listing 1: Region Growing Algorithm Code

```python
from pygadgetreader import *
import matplotlib
matplotlib.use('Agg')
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np

snap_file_sim='/lustre/home/ciencias/fisica/pregrado/sd.hernandez204/Outputs/28_02_2016/snapshot_005'

requested='dmcount'
n_dm=readheader(snap_file_sim,requested)
print "Number of Dark Matter particles"
print n_dm

array_pos=readsnap(snap_file_sim,'pos','dm')
array_vel=readsnap(snap_file_sim,'vel','dm')

array_magnitud_vel=np.zeros(n_dm)
for i in range(0,n_dm):
    array_magnitud_vel[i]=np.linalg.norm(array_vel[i,:])
print "The speed array has been calculated"

longitud = (int) (array_pos.shape[0])
print "The lenght of the position array should be the same number of particles"
print str(longitud)

array_copy=array_pos.copy()
array_results=np.zeros((longitud,4))
array_results[:,:-1] = array_copy
print "The results array has been created."

vel_max = np.max(array_magnitud_vel)
vel_min = np.min(array_magnitud_vel)
vel_std = np.std(array_magnitud_vel)
vel_mean = np.mean(array_magnitud_vel)

print "Vel_max, Vel_min, std_vel, vel_mean"
print vel_max, vel_min, vel_std, vel_mean
thresh_v_hig = vel_mean + 8.0 * vel_std
thresh_v_low = vel_mean + 2.0 *  vel_std
print "Thresh_high, Thresh_low"
print thresh_v_hig, thresh_v_low

#Gets the indexes of the seeds
indexes = np.where(array_magnitud_vel>(thresh_v_hig))[0]
print 'There are: '+ str(indexes.shape[0]) +' seeds'

def region_accretion(l):
    #Resize the window if necesary
    size_window_up = n_window
    if(((l+size_window_up) > longitud)):
        size_window_up = longitud - l
    size_window_down = n_window
    if(((l-size_window_down) < 0)):
        size_window_down = 0 + l
    distancias_l_up = np.zeros(size_window_up)
    distancias_l_down = np.zeros(size_window_down)
    distancias_l_up[:] = 1000000
    distancias_l_down[:] = 1000000
    if(size_window_up == size_window_down):
        for k in range(0,size_window_up):
            if(((l+k) < longitud)):
                if((array_results[l+k,3] == 0)):
                    distancias_l_up[k]=np.sqrt(np.sum((array_results[l,0:2] - array_results[l+k,0:2])**(2.0)))
            if(((l-k) > 0)):
                if((array_results[l-k,3] == 0)):
                    distancias_l_down[k]=np.sqrt(np.sum((array_results[l,0:2] - array_results[l-k,0:2])**(2.0)))
    else:
```

```
        for k in range(0,size_window_up):
            if(((l+k) < longitud)):
                if((array_results[l+k,3] == 0)):
                    distancias_l_up[k]=np.sqrt(np.sum((array_results[l,0:2] - array_results[l+k,0:2])**(2.0)))
        for k in range(0,size_window_up):
            if(((l-k) > 0)):
                if((array_results[l-k,3] == 0)):
                    distancias_l_down[k]=np.sqrt(np.sum((array_results[l,0:2] - array_results[l-k,0:2])**(2.0)))
    distancias_l_up[0]=1000000
    distancias_l_down[0]=1000000
    closest_l_up = np.min(distancias_l_up)
    index_closest_up = np.argmin(distancias_l_up)
    closest_l_down = np.min(distancias_l_down)
    index_closest_down = np.argmin(distancias_l_down)
    #Now si if these validate the condition
    #It has to see if it got any new candidates (not marked particles)
    if ((closest_l_up != 1000000)):
        vel_l = array_magnitud_vel[l]
        vel_closest_up = array_magnitud_vel[l + index_closest_up]
        if( (vel_l>= vel_closest_up>=thresh_v_low)&(array_results[l+index_closest_up,3]==0)):
            #it is marked
            array_results[l + index_closest_up,3] = 1
            #Recursion
            l_up = l + index_closest_up
            region_accretion(l_up)
    if ((closest_l_down != 1000000)):
        vel_l = array_magnitud_vel[l]
        vel_closest_down = array_magnitud_vel[l + index_closest_down]
        if( (vel_l >= vel_closest_down>=thresh_v_low)&(array_results[l-index_closest_down,3]==0)):
            #it is marked
            array_results[l - index_closest_down,3] = 1
            #Recursion
            l_down = l - index_closest_down
            region_accretion(l_down)

#Define the default size of the search window
n_window=10000
#Now mark the seeds
for i in indexes:
    array_results[i,3] = 1
#Now do the recursion
for i in indexes:
    #Recursion
    region_accretion(i)

print 'The regions have: '+ str(len(array_results[array_results[:,3] == 1,:])) +' galaxies.'
print 'Starting with points with V > '+ str(thresh_v_hig)
print 'Ending at points with V < ' + str(thresh_v_low)

#Now it gets the points of the regions
array_pos_region=array_results[(array_results[:,3] == 1),:]
```