*Zach Warren*

*Final Project*

*Code:*

**import** numpy **as** np  
**import** scipy.spatial **as** spatial  
**from** scipy.spatial.distance **import** euclidean  
**import** pandas **as** pd  
**import** matplotlib.pyplot **as** plt  
  
*#universal variables*data = **'DM\_2p8M.dat'**output\_coords = **'coords\_and\_flags9.csv'**output\_halos = **'halos9.csv'**cube\_length = 141.3  
particle\_mass = 1.4\*10\*\*10  
  
*#get data*x,y,z = np.loadtxt(data,unpack=**True**)  
xyz=list(zip(x,y,z))  
numData = len(xyz)  
  
*#create dataframe to hold final values  
#and to hold initial points and eventually flags*haloDF = pd.DataFrame(columns=[**'Mass'**, **'Center of Mass'**, **'RMS Radius'**, **'Potential Issue'**], index=np.arange(1,len(xyz)))  
  
d = {**'x'**:x, **'y'**:y, **'z'**:z}  
haloCoords = pd.DataFrame(d)  
  
*#r\_link is the mean interparticle separation times .2  
#which is the number density*r\_link = .2\*(cube\_length/np.power(numData,1/3.))  
  
*#create flag array, tree, and halo count for flag*flag = np.linspace(0,0,len(x))  
potential\_particles = 0  
tree = spatial.cKDTree(xyz)  
halo\_count = 0  
  
**for** i **in** range(0,numData):  
 **if**(i % 1000 == 0):  
 print(i)  
 *#make sure the particle hasn't been used* **if** (flag[i] == 0):  
 point\_list = [i]  
 coords = [xyz[i]]  
 halo\_count = halo\_count + 1  
 flag[i] = halo\_count  
  
 *#iterate through all points in point list* **for** p **in** point\_list:  
 neighbors = tree.query\_ball\_point(xyz[p],r\_link)  
 *#make sure there are some neighbors* **if** neighbors:  
 **for** j **in** neighbors:  
 **if** x[j] - r\_link < 0 **or** x[j] + r\_link > cube\_length \  
 **or** y[j] - r\_link < 0 **or** y[j] + r\_link > cube\_length \  
 **or** z[j] - r\_link < 0 **or** z[j] + r\_link > cube\_length:  
 haloDF[**'Potential Issue'**][halo\_count] = **'yes'** potential\_particles+=1  
 *#make sure the particle hasn't been used already* **if** (flag[j] == 0):  
 point\_list.append(j)  
 coords.append(xyz[j])  
 flag[j] = halo\_count  
  
 total\_points = len(coords)  
 center\_of\_mass = tuple(map(**lambda** y: sum(y) / float(len(y)), zip(\*coords)))  
  
 *#get distance from center for rms* dist\_from\_center = []  
 **for** p **in** coords:  
 dist\_from\_center.append(euclidean(p,center\_of\_mass))  
  
 rms\_radius = np.sqrt(np.mean(np.square(dist\_from\_center)))  
  
 *#store data for each halo* haloDF[**'Mass'**][halo\_count] = 1.4\*10\*\*10\*total\_points  
 haloDF[**'Center of Mass'**][halo\_count] = center\_of\_mass  
 haloDF[**'RMS Radius'**][halo\_count] = rms\_radius  
  
haloDF = haloDF[haloDF.Mass > 1.4\*10\*\*11]  
haloDF = haloDF.reset\_index()  
  
  
haloDF.to\_csv(output\_halos)  
haloCoords[**'flag'**] = flag  
haloCoords.to\_csv(output\_coords)  
  
print(**'done'**)  
print(halo\_count)  
print(potential\_particles)  
  
f, plts = plt.subplots(2,1,figsize=(15,15))  
  
*#get number of halos with mass above a certain threshold*x = np.linspace(12,15,30)  
  
count\_mass = np.linspace(0,0,30)  
  
i=0  
  
**for** mass **in** x:  
 **for** row **in** haloDF[**'Mass'**]:  
 **if** np.log10(row) >= mass:  
 count\_mass[i]+=1  
 i+=1  
  
print(count\_mass)  
  
num\_dens\_halos = [np.log10(x/(np.power(cube\_length, 3)\*.1)) **for** x **in** count\_mass]  
log\_rms = np.log10(haloDF[**'RMS Radius'**].astype(float))  
log\_mass = np.log10(haloDF[**'Mass'**].astype(float))  
  
plts[0].plot(x,num\_dens\_halos)  
plts[0].set\_ylabel(**'log(dN/Vdm)'**)  
plts[0].set\_xlabel(**'log(M)'**)  
plts[0].set\_title(**'Number Density of Halos vs. Mass'**)  
  
plts[1].scatter(log\_rms,log\_mass)  
plts[1].set\_ylabel(**'log(rms radius)'**)  
plts[1].set\_xlabel(**'log(M)'**)  
plts[1].set\_title(**'RMS Radius vs. Mass'**)  
  
  
plt.show()

*Output:*

*Total Halos: 1245449*

*Potential issues with boundaries: 595072*

*Number of halos in each mass bin:*

*[ 2.72800000e+03 2.15800000e+03 1.68500000e+03 1.33800000e+03*

*1.05900000e+03 8.08000000e+02 6.28000000e+02 4.80000000e+02*

*3.72000000e+02 2.88000000e+02 2.26000000e+02 1.63000000e+02*

*1.30000000e+02 9.50000000e+01 6.90000000e+01 4.60000000e+01*

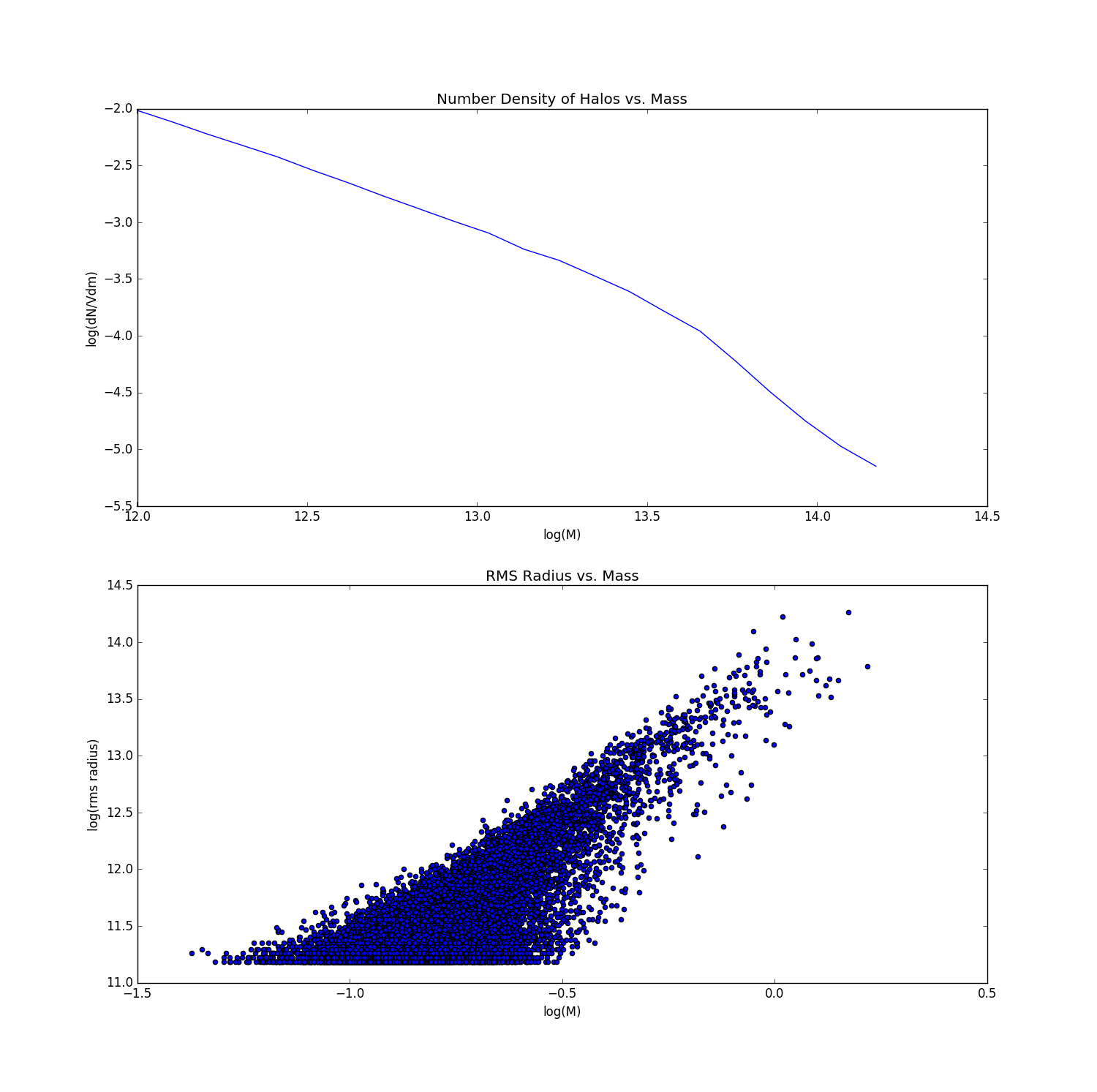
*3.10000000e+01 1.70000000e+01 9.00000000e+00 5.00000000e+00*

*3.00000000e+00 2.00000000e+00 0.00000000e+00 0.00000000e+00*

*0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00*

*0.00000000e+00 0.00000000e+00]*

*Charts:*

**