ASTR 400B Homework4

Due: Feb 13th 2020 5 PM

We want to explore how each galaxy in the Local Group moves owing to the gravitational forces they each exert on each other. To do this, we need to compute the center of mass (COM) position and velocity vectors of each galaxy at any given point in time in the simulation.

The simulation is in Cartesian coordinates at all snapshots. At SnapNumber 0 (present day), the MW is located at the origin of this coordinate system x,y,z = (0,0,0). At later snapshots, the MW will move so it is no longer at the origin. Therefore, we need to track the COM motion of the MW, M31 and M33 to find their locations at future snapshots.

1 Beginning the Program: Defining the Class

You will be using a class structure for this solution set.

- Git pull on the course's master repository to see an example script that you can use and modify for this assignment. CenterofMass_Ex.py and centerofmass_ex.ipynb contain the same information to guide you through this assignment.
- Rename this script to *CenterOfMass* and save.
- Functions in this program will call the *Read* function, so *ReadFile* is imported. Be sure this file exists in the same directory.
- You will once again need the files: MW_000.txt, M31_000.txt, M33_000.txt.
- In this program a CLASS called CenterOfMass has been defined.

  class CenterOfMass:

  A tutorial for classes in python can be found here: https://docs.python.org/3/tutorial/classes.html

2 Initialize the Class

Initialize your class so that each object will store data from the simulation file based on particle type.

def __init__(self, filename, ptype):
self.time, self.total, self.data = Read(filename)

self.index = np.where(self.data['type'] == ptype)

self.m = self.data['m'][self.index]

- The index to help you store particle type has already been created for you. The particle masses have also been provided as an example.
- Follow the same procedure as in *ParticleProperties* (Homework 2) to store the mass, position (x,y,z) of particles of a given type.
- You must use "self" to refer to initialize values, eg. *self.data['x'][self.index]*. This stores the data for you when you create an object so that you do not have to read in the data each time you call a function.

The nomenclature "self" is used to refer to quantities that are common to the object. Each function you create must start with the word "self" as an input. E.g.

def NewFunction(self, a):

3 Generic Definition of Center of Mass

- Create a function called *COMdefine* that will generically return the 3D coordinates of the COM (position or velocity) of any given galaxy.
- This function should take as input: the x,y,z coordinates of the particle position or velocity and the mass.
- Remember that the first input for the function within a class is "self"
- This function should return the x,y,z coordinates of the COM (position or velocity)

E.g. for each component of the position or velocity vector (like X) compute:

$$\mathrm{X_{COM}} = \frac{\Sigma x_i m_i}{\Sigma m_i} \tag{1}$$

As shown in Fig. 1, this equation is generic: it would work if you input either the velocity or position vectors.

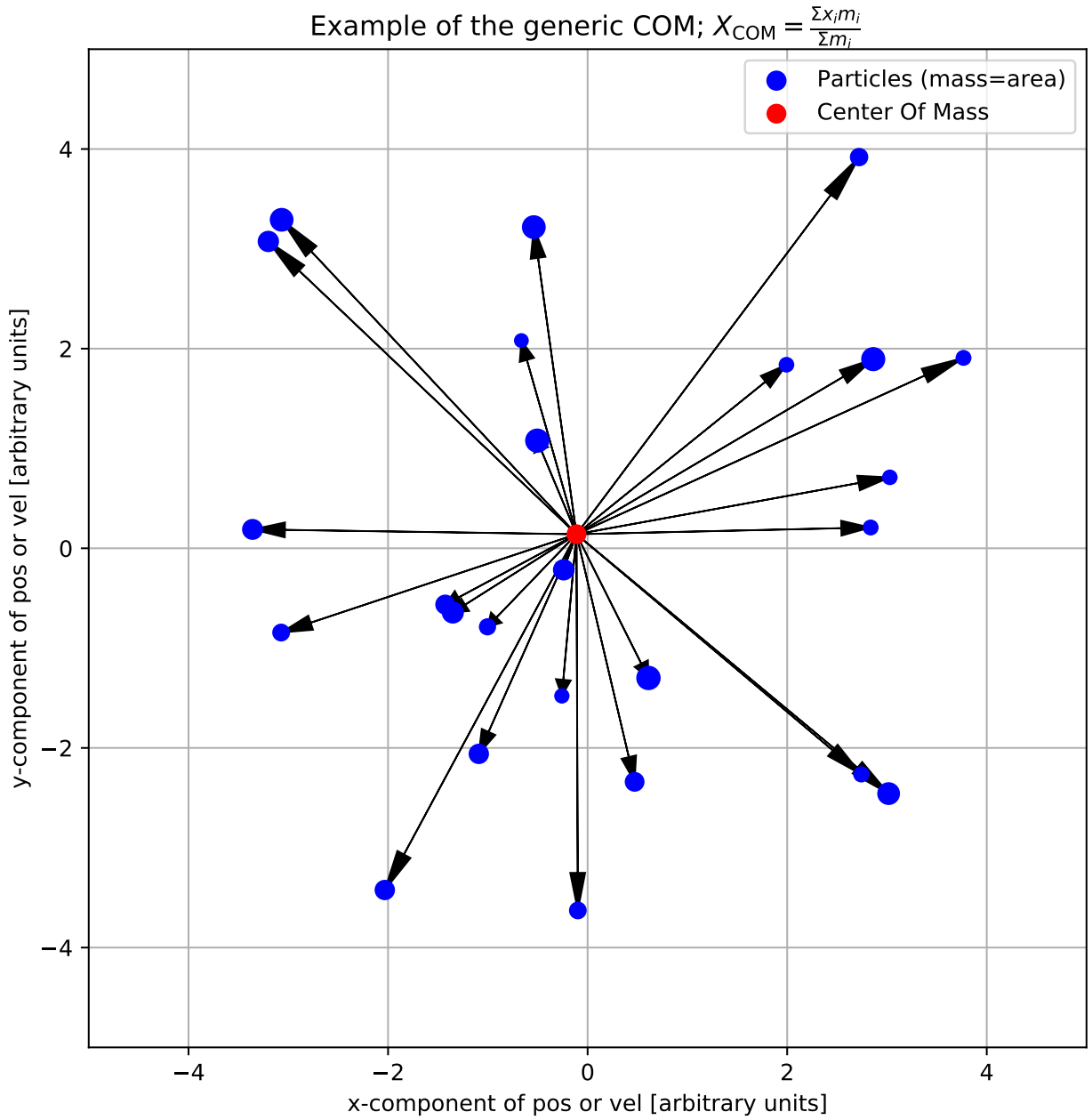

Figure 1: This picture presents an example of the generic definition of center of mass in a 2D plane. In each dimension of either the position or the velocity, the Eq. 1 is applied on the 25 particles (*in blue*) to determine their center of mass (*in red*). (Be careful that the data in your homework is three dimensional.)

4 Refining the COM Position

Create a function called *COM_P* that will call *COMDefine* to determine the center of mass (COM) position and velocity vectors of a given galaxy using particles of a given type.

- The function should take as input: "self" and the tolerance (delta) that will decide whether the COM position has converged.
- The function returns an array with the x, y, z coordinates of the COM position

While you have already created a generic function that returns the COM (*COMdefine*), you need to refine this COM position calculation iteratively (using successively smaller volumes) to make sure the position has converged. Do this by using the following steps.

Step 1: First Guess (see Fig. 2)

- Call *COMdefine* to compute a first estimate for the COM position vector (XCOM, YCOM, ZCOM) using all the particles of the desired type. Store the magnitude of that vector (RCOM).
- When referring to functions already definied in the Class you need to refer to "self" *self.COMdefine*.
- Change the particle positions to the COM reference frame by subtracting the first guess COM position vector from the particle position vectors (self.x - XCOM, etc).
- Create an array that stores the magnitude of the new position vectors of all particles in the COM frame (RNEW).
- Find the maximum 3D separation of the particles from the COM position in the COM frame and reduce it by half (RMAX). You will next refine the COM calculation using this smaller volume to make sure the COM position has converged.

Step 2: Refine the Guess (see Fig. 3)

Set up a while loop that continues while the difference between RCOM and a new COM position (RCOM2; computed using half the previous volume) is larger than some tolerance (delta), which you have set as an input value to this function.

- Pick an initial value for the change in COM position between the first guess (RCOM) and the new one you will compute from half that volume (like 1000 kpc, so it will be larger than the input tolerance initially).
- Set up a while loop, that continues while the change in RCOM is larger than the tolerance (delta) that you want for convergence.
- Divide RMAX by half to refine the volume again and repeat the loop

Step 3: Finally, return the converged COM Position Vector (x,y,z) rounded to 2 decimal places.

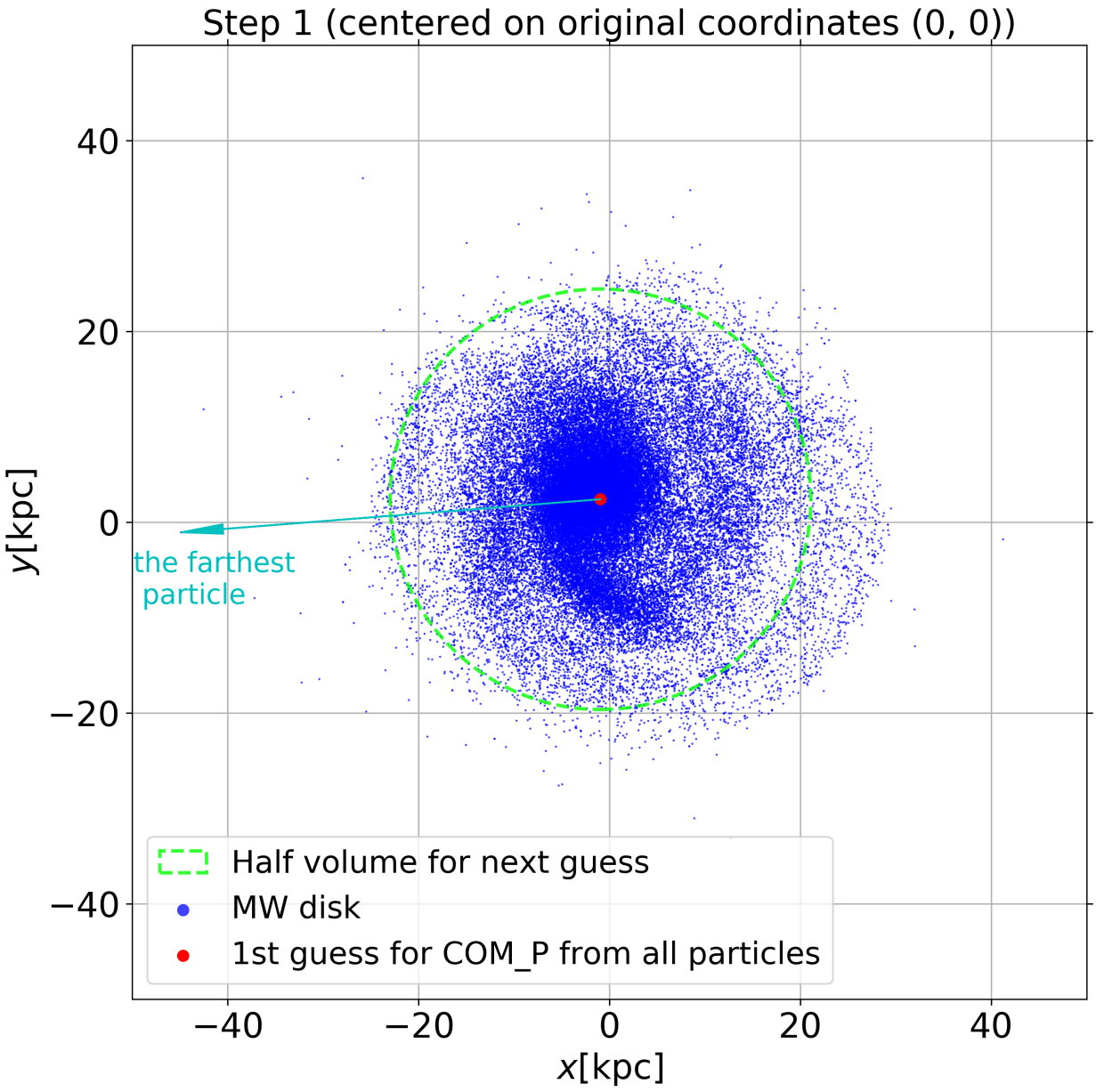

Figure 2: This figure illustrates all the MW disk particles (*in blue*) from the data you are using (projected on the x-y plane). Here we choose the disk particle type as an example. Following the instructions in Step 1, you need to call *COMdefine* to find a first estimate for the COM position vector (*the red point*) using all the particles, and then find the farthest particle (*pointed by the cyan arrow*) with respect to the estimated COM position. All the particles within half the length of the cyan arrow in the estimated COM frame (*inside the green circle*) will be utilized later to refine the COM calculations. Again, note that this picture is in 2D, but your data is in 3D and you should calculate all stuff in 3D.

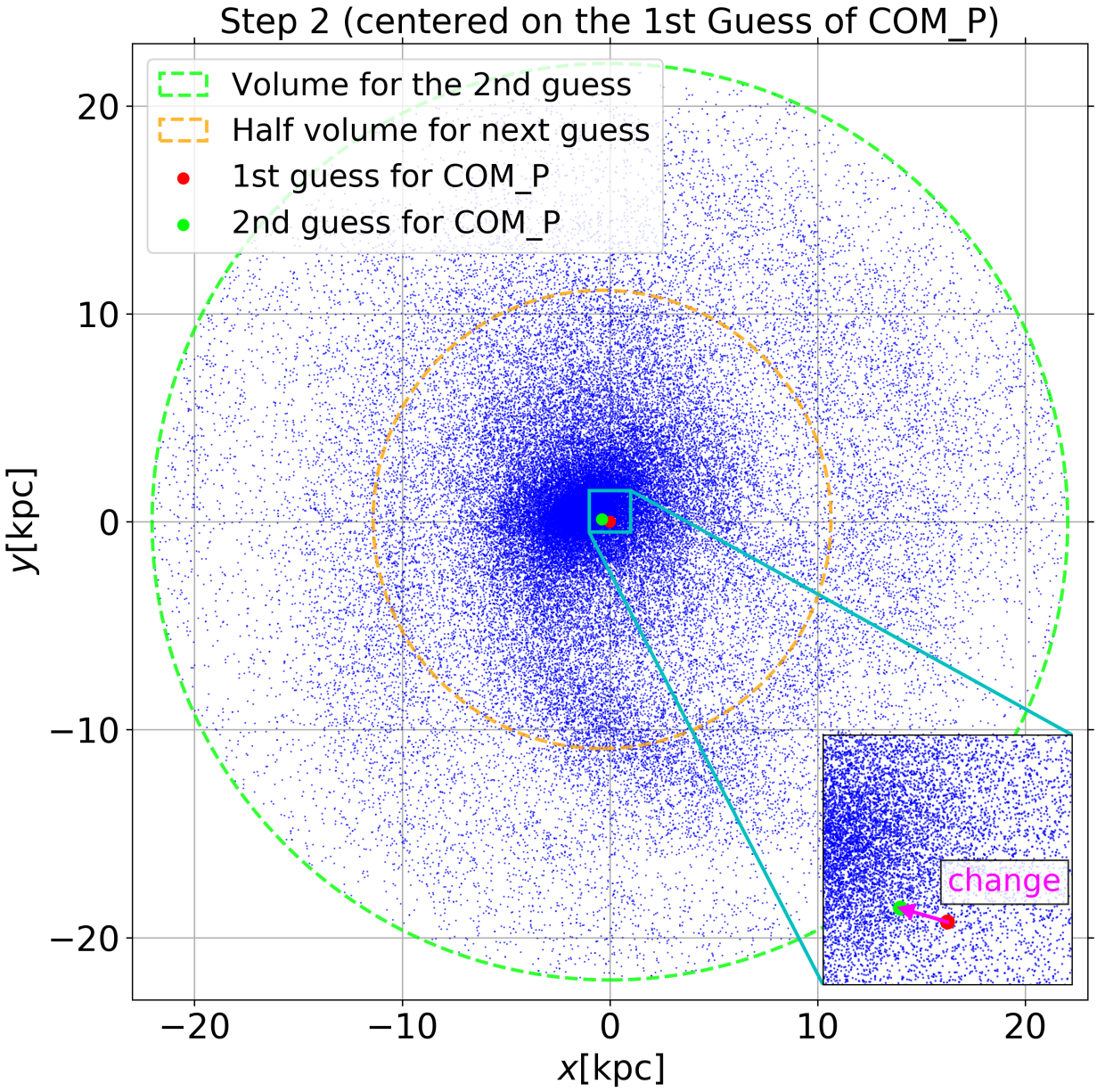

Figure 3: This figure illustrates the next step to refine the COM calculations. In this figure, only the particles within the green circle (as in Fig. 2) are shown. Also, all particles are drawn in the frame of the first COM guess (*the red point*). Following the instructions in Step 2, you need to call *COMdefine* to find the 2nd guess of the COM (as denoted by *the green point*) using all the particles inside this green circles, and then quantify the change in COM position between two guesses (as denoted by *the magenta arrow)*). If the magnitude of the change vector is larger than the input tolerance (delta), then you need to perform such refinement steps again and again until the magnitude of the change vector is smaller than the input tolerance (delta). For example, the orange circle (whose radius is half the radius of the green circle) indicates where you should perform the 3rd guess.

5 Computing the COM Velocity

Now that you know the COM position, you can compute the COM velocity by creating a function *COM_V*.

- Store the velocities of all particles that are within 15 kpc from the COM position. Note that you already initialized self.vx, self.vy, self.vz in the beginning of the class structure so you will only have to mask these based on the particles within 15 kpc.
- Use *COMdefine* to compute and store the COM velocity.
- Return the COM Velocity Vector (vx, vy, vz), rounded to two decimal places.
- The end of the provided example scripts show you how to create an object of the class and apply the methods of the class to that object (i.e. call COM_P on the MW's data file).

6 Testing Your Code

1. What is the COM position and velocity vector for the MW, M31 and M33 at Snapshot 0 using Disk Particles only (use 0.1 kpc as the tolerance so we can have the same answers to compare) ? In practice, disk particles work the best for the COM determination. Recall that the MW COM should be close to the origin of the coordinate system (0,0,0).
2. What is the magnitude of the current separation and velocity between the MW and M31? From class, you already know what the relative separation and velocity should roughly be (Lecture2 Handouts; Jan 16).
3. What is the magnitude of the current separation and velocity between M33 and M31?
4. Given that M31 and the MW are about to merge, why is the iterative process to determine the COM is important?

7 Homework Submission

- You must DOCUMENT your code. Explain each step.
- Create a directory called Homework4. Save your code and answers to section 6 in that directory. Your answers to section 6 should be saved EITHER:
    1. As part of your Jupyter notebook solution.
    2. Take a screen shot of your python output (with the relevant print statements for each quantity) from the command line.
- Upload your Homework4 directory to your public 400B_yourlastname repository on GitHub