Physics 410: Project 1, Galaxy Collisions

Drew Spencer

October 16, 2019

1 Introduction

In this project, I created a model of two galaxies colliding which is based off the model by Alar and Juri Toomre. While my simulation is not to be considered "ideal" it models the basic principles of two galaxies morphing and distorting through their interactions with each other. In this model we have two separate central cores that will have any n amount of stars rotating circularly around them respectively. The stars have no mass in our model and their acceleration is only influenced by their respective core. This makes the stars have no influence on each other nor on the core. However the two separate cores will have gravitational influence on each other ıand every star in the model whether the star is in its orbit, or the other core. Initially the influence between the two cores is not felt until they become relatively close to each other. Once this happens all the stars in either orbit begin to feel the gravitational field of the other core and the morphing begins to happen. As the stars don't enter into our acceleration equation, it is very easy to implement thousands of stars per galaxy. In my galaxy collision model, I implement a two dimensional collision.

2 Breakdown of Mathematical Model

For the mathematical model, we were given a guideline to follow on N-body Gravitational problem. We also worked through many equations in class including, Finite Difference Approximations for the Newton's equation of motion. The breakdown of these two portions is shown below with brief explanations as to what is happening with each step.

2.1 The Gravitational N-Body Problem

For the first portion of our assignment we were given a guideline to follow on **The Gravitational N-Body Problem**. For this we are looking at Newtonian gravitational forces (attractive) and how each particle interacts with one another though this. Coupling Newton's second law with the law of gravitation, we can get the basic equations of motion in vector form. N denotes the number of particles in the system.

$$m_i \mathbf{a}_i = G \sum_{j=1, j\neq i}^{N} \frac{m_i m_j}{r_{ij}^2} \hat{\mathbf{r}}_{ij} \qquad i = 1, 2, ..., N \tag{1}$$

While G is Newton's gravitational constant, we take this number to be 1 to non-dimensionalize our problem. This allows us to use "natural" units and simplify the problem.

Where $\vec{a}_i(t)$ is the acceleration of the i-th particle which is also the second derivative of position giving us

$$\mathbf{a}_i(t) = \frac{d^2\mathbf{r}(t)}{dt^2}$$

and the magnitude of the separation vector is

$$r_{ij} = [(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2]^{\frac{1}{2}}$$

and we have the unit vector of the position between these two particle given by

$$\hat{\mathbf{r}}_{ij} \equiv \frac{\mathbf{r}_j - \mathbf{r}_i}{r_{ij}}$$

Combining all of these we arrive at and dividing by m_i

$$\frac{d^2\mathbf{r}(t)}{dt^2} = \sum_{j=1, j\neq i}^{N} \frac{m_j}{r_{ij}^3}\mathbf{r}_j - \mathbf{r}_i \qquad i = 1, 2, ..., N \tag{2}$$

Seeing that this is a differential equation of second order, we know we will need to have some initial conditions to find the solutions. The initial conditions we need are position of the cores and the velocity of the cores (the derivative of position). The stars initial positions and velocities will all be based off of the initial conditions of the cores

2.2 Finite Difference Approximation (Discretization)

For the Finite Difference approximation, we will have a domain of $0 \leq t \leq t_{max}$ We also have a uniform time mesh (constant time step) and we also use a level parameter define by l. The level parameter will be for means of convergence testing. The level parameter also enables us to specify the mesh grid. Below summarize these parameters.

$$n_t = 2^l - 1$$

$$\Delta t = \frac{t_{max}}{n_t - 1}$$

$$t^n = (n-1)\Delta t \qquad n = 1, 2, ..., n_t$$

The finite difference approximation, as we derived in class, lets us approximate the second derivative of position (acceleration).

$$\frac{d^2\mathbf{r}_i}{dt^2} \approx \frac{\mathbf{r}_i{}^{n+1} - 2\mathbf{r}_i^n + \mathbf{r}_i^{n-1}}{\Delta t^2} \tag{3}$$

We can take this approximation and substitute it in to equation (2) and then isolate for $\vec{r_i}^{n+1}$ which is the our position at advanced time steps. This will be how we calculate each new position from time step 3 onward.

$$\mathbf{r}_i^{n+1} = 2\mathbf{r}_i^n - \mathbf{r}_i^{n-1} + \Delta t^2 \sum_j \frac{m_j}{r_{ij}^3}\mathbf{r}_{ij} \qquad n = 2, 3, ..., n_t - 1 \tag{4}$$

Arriving at this equation, we now have our equation of motion that will be used to advance our problem. To do this we need to know $\mathbf{r}_i^n$ and $\mathbf{r}_i^{n-1}$. We know these as these are the initial conditions that we input into the system or position and velocity.

2.3 Derived equations for Positions

For this portion, we break down the positions of each core, and the stars surrounding each core. We know that the stars in our problem have no mass and don't influence any gravitational forces on other particles. Each core however influences all particles in the system, that is the other respective core and all stars in both orbits. Thus we need 3 equations to be broken down, the first core on the second core, the second core on the first core, and the effect the star feel from both cores. These three equations were solved through equation (4), we know the cores only influence each other but the stars are influenced by both core, thus the last term on the stars equation is the sum of the effect by each core.

$$\mathbf{r}^{n+1} = 2\mathbf{r}_1^n - \mathbf{r}_1^{n-1} + \Delta t^2 \left(\frac{m_2}{r_{12}^3}\mathbf{r}_{12}\right) \tag{5}$$

$$\mathbf{r}^{n+1} = 2\mathbf{r}_2^n - \mathbf{r}_2^{n-1} + \Delta t^2 \left(\frac{m_1}{r_{21}^3}\mathbf{r}_{21}\right) \tag{6}$$

$$\mathbf{r}^{n+1} = 2\mathbf{r}_i^n - \mathbf{r}_i^{n-1} + \Delta t^2 \left[\frac{m_1}{r_{i1}^3}\mathbf{r}_{i1} + \frac{m_2}{r_{i2}^3}\mathbf{r}_{i2}\right] \tag{7}$$

Core ones position update is equation (5), core twos update is position equation (6) and the i-th stars position is updated via equation (7).

Each of these 3 equations require initial conditions in order to be solved. The initial conditions $\vec{r_i}^1$ are given (just the initial position of the cores/stars). The initial conditions $\vec{r_i}^2$ are found by Taylor series expansion just as in the non linear pendulum:

$$\mathbf{r}_i(\Delta t) = \mathbf{r}_i(0) + \Delta t\frac{d\mathbf{r}_i(0)}{dt} + \frac{1}{2}\Delta t^2\frac{d^2\mathbf{r}_i(0)}{dt^2} + O(\Delta t^3) \tag{8}$$

Here we were given initial condition to start with, these were later manipulated to see how certain conditions affected the progression of the system. Since we were given the initial condition we were able to use them to start the program running and see how we did with our breakdowns of certain equations. Seeing that we have the initial condition we can re-write equations

$$\mathbf{r}_1^2 = \mathbf{r}_1(\Delta t) = \mathbf{r}_1(0) + \Delta t\mathbf{v}_1(0) + \frac{1}{2}\Delta t^2\left[\frac{m_2}{r_{12}(0)^3}\mathbf{r}_{12}(0)\right] \tag{9}$$

$$\mathbf{r}_2^2 = \mathbf{r}_2(\Delta t) = \mathbf{r}_2(0) + \Delta t\mathbf{v}_2(0) + \frac{1}{2}\Delta t^2\left[\frac{m_1}{r_{21}(0)^3}\mathbf{r}_{21}(0)\right] \tag{10}$$

$$\mathbf{r}_i^2 = \mathbf{r}_i(\Delta t) = \mathbf{r}_i^2(0) + \Delta t\frac{d\mathbf{r}_i(0)}{dt} + \frac{1}{2}\Delta t^2\left[\frac{m_1}{r_{i1}(0)^3}\mathbf{r}_{i1}(0) + \frac{m_2}{r_{i2}(0)^3}\mathbf{r}_{i2}(0)\right] \tag{11}$$

Now that we have all these equations, we have all the necessary components to begin working on coding the project. Equations (9) gives core ones scond position using the initial conditions given. We know that $\frac{d\mathbf{r}_i(0)}{dt} = \mathbf{v}_i(0)$ where $i = 1, 2$ for each core. Seeing as we input the initial velocities I changed the notation accordingly. Since we don't know the initial stars velocity, it's velocity is left in the more basic form of $\frac{d\mathbf{r}_i(0)}{dt}$. These three equations enable us to move forward and see if we broke them down properly.

3 Creating the Galaxies

3.1 Making the two cores

Now is the portion where I began writing code in MATLAB. Following the guidelines I started off by trying to code the two cores using the equations broken down above. As these are all vectors that get updated we needed to make arrays that could hold each value of position from initial to final. Seeing as we were asked to create a system that could be used in 3 dimension, I made a position array for the cores that took three columns, one the x position, y position, and z position. Initially my script had outputs that would be registered, a time array, position matrix, and a velocity matrix. The function took the arguments of tmax, level, initial position core 1, initial position core 2, initial velocity core 1, initial velocity core 2, mass core 1, mass core 2). These inputs were then use to create all the arrays and implemented in the equations (9) and (10) as initially we were told to try coding 2 cores in concentric orbit of each other. The way I found the initial conditions was laid out to us in the N-Body handout. The breakdown went as follows:

From the N-body handout we have $\mathbf{r}_1(0) = (r_1, 0, 0)$, $\mathbf{r}_1(0) = (-r_2, 0, 0)$, $r = r_1 + r_2$, $m = m_1 + m_2$, and we are given the four equations, 2 for position and two for velocity. These equation are as follows:

$$r_1 = \frac{m_2 r}{m} \tag{12}$$

the same derivation follows for r_2 and we get the following result:

$$r_2 = \frac{m_1 r}{m} \tag{13}$$

We know that the velocities can be broken down from centripetal motion. Again we have a non-dimentionalized system where G=1 and were given these equation in our handout.

$$v_1 = \sqrt{\frac{m_2 r_1}{r^2}}$$

$$v_1 = \frac{\sqrt{m_2 r_1}}{r} \tag{14}$$

Similarly, the velocity v_2 is:

$$v_2 = \frac{\sqrt{m_1 r_2}}{r} \tag{15}$$

To induce a circular orbit, we know the velocities will need to be perpendicular to the separation vector between the cores, otherwise we would get a wonky orbit or not orbit at all $v_1(0) = (0, v_1, 0)$ and $v_2(0) = (0, -v_2, 0)$.

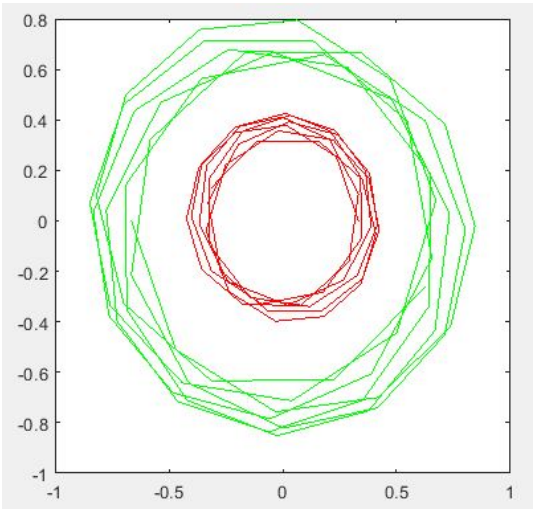
Seeing that we have these relations, we can use any arbitrary values and derive everything accordingly. After talking with a couple friends we decided that the best approach would be to keep it simple and use the value listed below:

$$m_1 = 1 \qquad m_2 = 0.5$$

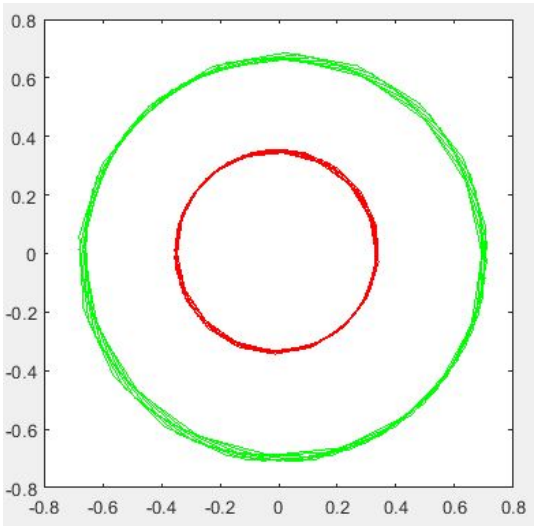
$$r_1 = 1/3 \qquad r_2 = 2/3$$

$$v_1 = \sqrt{\frac{1}{6}} \qquad v_2 = \sqrt{\frac{2}{3}}$$

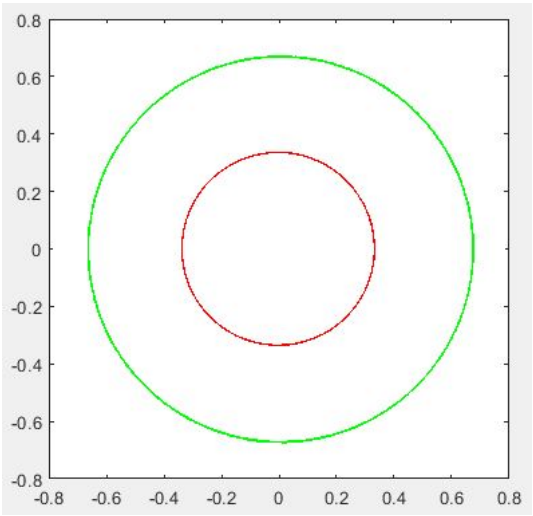
Now using these conditions and the equations broken down earlier, we were able to, after a few trial and error runs, create concentric circular orbit of the two cores. The plots below



(a) Circular orbits of cores with $l = 6$



(b) Circular orbits of cores with $l = 7$

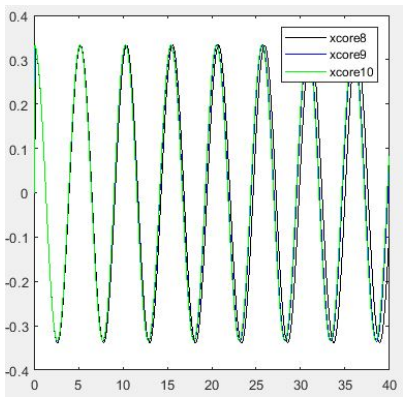


(c) Circular orbits of cores with $l = 8$

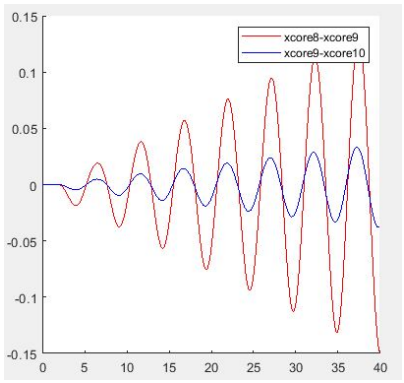
These 3 graphs represent the circular orbit or the two cores. We can see that are definitely in circular orbit with each other. As it can clearly be seen the level of each system/run highly influences the shape/smoothness of the curve. This is because as the level increases, the times steps get finer and finer (smaller) which gives a better approximation of the path traced by the core. There wasn't too much difference from level 8 onward and I believe this showed enough to understand how the level affects the system.

3.2 Convergence Testing

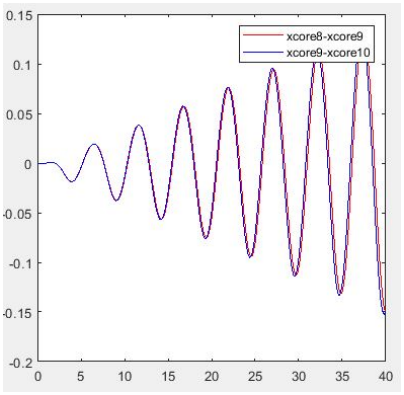
The next step after getting the cores to have a circular orbit with each other was to do a convergence test of the orbit. We were shown in a tutorial how to do this with the nonlinear pendulum. The method was almost the exact same implementation for the cores. Here we could test either the x or y position of either core as the basis of the code is the same for both cores. We were asked to use level 8, 9, and 10 for this test. To see if the system passed the convergence test it was a fairly straight forward implementation. What i did was write a convergence.m script that ran the same code 3 times with each system 1 running level 8, system 2 running level 9 and system 3 running level 10. The graphs of position vs time were then all plotted together. In my test I used the x position of each system vs time. The first plot was strictly the position of the 3 systems graphed against each other. Second plot was the difference between the position of level 8 and 9 (xcore8 - xcore9), and the difference between level 9 and 10(xcore9 - xcore10). Next we took to see if the system was on the $O(\Delta t^2)$, I multiplied the (xcore9-xcore10) by 4. The reason for doing this is that each level should increasing the grid spacing by an order of 2, then the difference should go as $2^2 = 4$ between the plots. If multiplying the above (xcore9-xcore10) by 4 lined up with the difference of (xcore8 - xcore9) on a plot then we know our system is of the order (Δt^2).



(d) Position vs Time



(e) Differences between plots



(f) Scaled difference

With the explanation above on the plots and analyzing the actual plots above, it is clear to see that we are on the $O(\Delta t^2)$ and the convergence test is passed.

3.3 Stars

Since the convergence test was passed, we know the system is of the proper order, and we can move on to adding in orbiting stars for each core. Using equation (11) we know how we can calculate the advanced position of the stars, but we need to initialize the stars and give them their initial positions and velocities. This is done by the following method: We know the stars create a circle around the core and we need a position that would be dependent on theta, and an initial random value. For the velocity component we know that we can breakdown the tangential velocity into x and y components to give initial speeds. This coupled with equations (14) and (15) we were able to figure out the x and y velocities for each star about its respective core. The equations found were as follows below.

$$x_1 = x + r(\cos\theta) \tag{16}$$

$$y_1 = y + r(\sin\theta) \tag{17}$$

$$v_{ix} = v_x \mp \sqrt{\frac{m_i}{r}}(\sin\theta) \tag{18}$$

$$v_{iy} = v_y \pm \sqrt{\frac{m_i}{r}}(\cos\theta) \tag{19}$$

where i = 1,2 for the respective core it is orbiting.
Seeing as the stars are orbiting around the cores, it made sense to add the initial velocities of the cores to the stars. Since the stars move with the core, they will need some form of "kick" to be able to do this and not blow up away from the core itself. The same went for the position of the star, as we need it to be within a certain radius of the core itself, would couldn't just put the star anywhere in the system, it had to be connected with a core.

3.4 Animations, the fun part!

The animations were fairly straight forward to breakdown from the file given to us called bounce.m There was ample explanation in the file to help us create the animations of our galaxies moving. The hardest part was breaking down what we actually needed and didn't need from the file. Turn out there was a few critical lines to do this and the rest was to define different parameters that we didn't need to change.

While the animation part was pretty straight forward, once we ran the animations it was very clear that there were certain portions of our code that need some fixing as many of the stars seemed to shoot off the map in all directions. This was where we bypassed a crucial "hint" in our handout about having conditions on the radius of the stars orbit. What was happening was the stars were getting to close to the core and the system "diverged or broke" and the stars just shot off. While this was funny to see the first few times, it became increasingly worrying when the problem wasn't getting fixed. After a couple days of leaving the code to refresh the brain, a friend of mine derived a little equation for the radius to limit how close the stars orbit around the core. After implementing this change there were no issues of stars blowing up before interacting with the other core. I would say i played around with the initial conditions for about 5 hours in total just to get a decent collision. Most resulted in the galaxies blowing up which i have a couple plots included of this happening. After watching the animation provided as a "solution" I tried to break down the dynamics of the system as close as possible and finally got something similar to the animation provided.

4 Screenshot of Animations

Here I ran the code with 3 different conditions on the direction of orbit of the stars. I show 2 different plots of each over time of the two galaxies interacting. One section is the stars of each galaxy rotating counterclockwise, another is one cores stars rotation clockwise and the other counterclockwise, and the third is both clockwise. I was surprised at the difference it made to their interactions as I needed to change the initial condition just to make them even influence each other when they were both rotating clockwise.

4.1 Both Counterclockwise

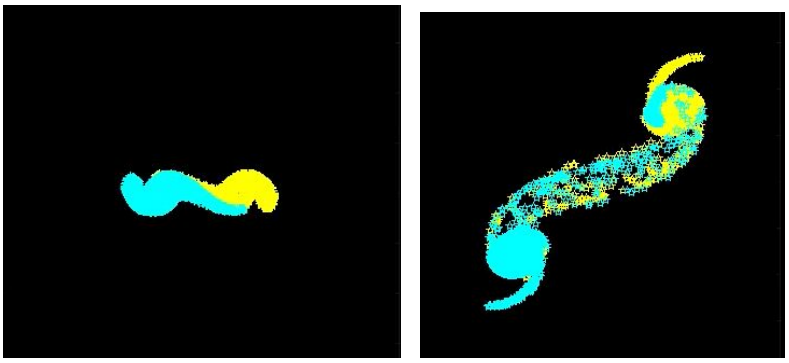
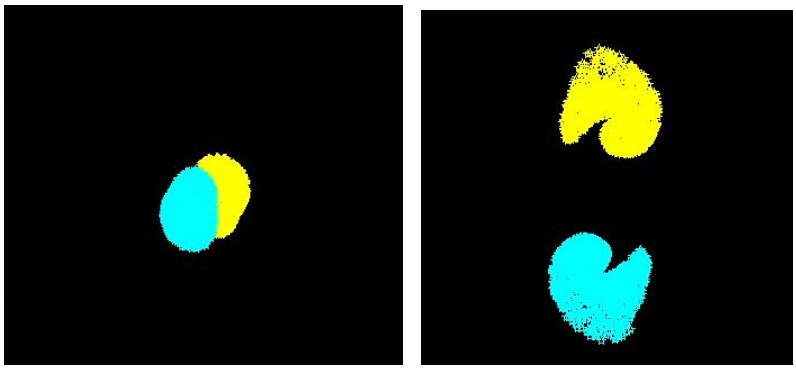


Figure 1: Both galaxies have counterclockwise rotation

For these plots , the rotations of both galaxies are counterclockwise. This was the first set of animations I made and took a long time to get the initial conditions correct as mentioned earlier. The initial condition I ended up using for these plots are as follows:

$m_1 = 8, m_2 = 8,\ r_1 = (-4, -9, 0), r_2 = (4, 9, 0), v_1 = (0.9, 0.4, 0), v_2 = (-0.9, -0.4, 0)$. There is 5000 stars in each galaxy for a total of 10000 stars. The level was 11 and tmax was 40.

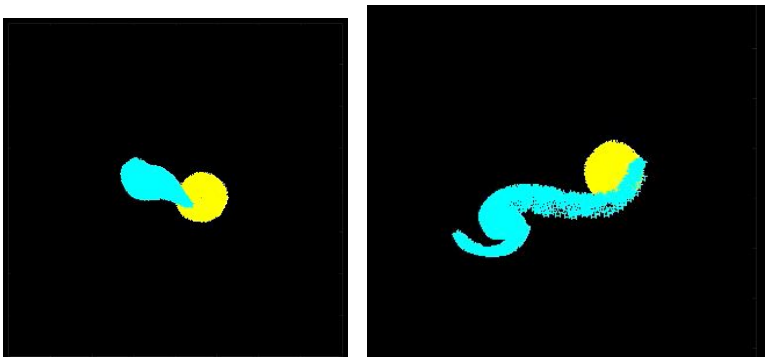
4.2 Both Clockwise



For these plots, I changed the rotation parameter to make them both rotate clockwise now. I was very surprised at how much the initial conditions needed to be changed here. With the same parameters as the counterclockwise rotation, the galaxies didn't interact with each other at all. So i changed the conditions a few times to make them closer and closer and eventually go them to show an interaction and was very surprised at what proceeded to happen. The initial condition I ended up using for these plots are as follows:

$m_1 = 8, m_2 = 8,\ r_1 = (-2, -5, 0), r_2 = (2, 5, 0), v_1 = (0.9, 0.4, 0), v_2 = (-0.9, -0.4, 0)$. There is 5000 stars in each galaxy for a total of 10000 stars. The level was 11 and tmax was 40.

4.3 Opposite Rotation



Here, the blue galaxy is rotating counterclockwise, and the yellow is rotating clockwise. At first it looked as though the both galaxies would interact similar to them both rotating counterclockwise. It became very clear once the interaction started that this was no the case. The counterclockwise galaxy was morphed very strongly where as the yellow galaxy only squished a little bit. The dynamics of this is fairly interesting and i don't know the underlying implications of why this happens without really looking into it. The initial conditions were the exact same as both galaxies rotating counterclockwise. They are as follows:

$m_1 = 8, m_2 = 8$, $r_1 = (-4, -9, 0), r_2 = (4, 9, 0), v_1 = (0.9, 0.4, 0), v_2 = (-0.9, -0.4, 0)$. There is 5000 stars in each galaxy for a total of 10000 stars. The level was 11 and tmax was 40.

5 Function and Inputs for main function

function [t, BIG] = toomre(tmax, level, r1, r2, v1, v2, m1, m2, rot1, rot2, N)

This is my main function to make the animations. The inputs and Outputs are as follows :

5.1 Inputs

- tmax = max time
- level
- r1 = core 1 initial position
- r2 = core 2 initial position
- v1 = initial velocity core 1
- v2 = initial velocity core 2
- m1 = mass core 1
- m2 = mass core 2
- rot1 = parameter for rotation core 1 (¿= 1 CCW rotation, ¡ 1 CW rotation)
- rot2 = parameter for rotation core 1 (¿= 1 CCW rotation, ¡ 1 CW rotation)
- N1 = number of stars core 1
- N2 = number of stars core 2

5.2 outputs

- t = time array
- BIG = Matrix of positions, velocities of cores and stars. Can index to find specific elements.

6 Functions

test.m was the function used for the convergence test.
test1.m was the function used for finding the circular orbits
convergence.m was the function to test for convergence
toomre.m was the main function to run all of my code and create the animations.