Galaxy Collision Simulation

Adrian Wu

1 Introduction

Galaxies are often a point of interest when it comes to researching the universe around us - their shape, size, and structure are often described as "spiral," where stars orbit a center in the same direction, or "elliptic," where stars are distributed in a complex manner in all directions, though often round in shape. Some small fraction of galaxies are labelled as "peculiar" galaxies because of their unusual shape and size. They may be the result of two galaxies coming close together; their gravitational forces cause them to interact and create these strange shapes. Such galaxies are of interest to study as they can provide deep insight into the inner workings of other galaxies. They represent perturbations and interactions that can help researchers analyze galaxies we can observe, but are too remote to experiment on directly.

We are asked to implement a very simplified model of galaxy collisions, motivated by the work of Alar and Juri Toomre in the 1970's. Despite being a model too simple to handle detailed simulation of galaxy interactions, it will suffice for our reproduction of some of the morphological features observed in actual galactic collisions.

The relevant files are

(a) **nbodyaccn.m** which is a function that outputs the acceleration each particle feels from the surrounding masses.

(b) **tnbodyaccn.m** which contains a script animate two bodies in mutual orbit.

(c) **twobodysim.m** which is a function that performs a simulation of two bodies in mutual orbit and keeps track of where both are at all times.

(d) **t2bodyconverge.m** which is a script that uses twobodysim.m to perform multiple simulations at varying levels, ℓ, to test convergence.

(e) **fastnbodyaccn.m** which is a function that outputs the acceleration for the stars and cores separately, and in a much more efficient fashion.

(f) **randcirclepts.m** which is a function that outputs the random points within some inner and outer ring of a circle, uniformly.

(g) **onegalaxysim.m** which is a script that animates a galaxy with many stars orbiting it, using fastnbodyaccn and randcirclepts.

(h) **twogalaxysim.m** which is a script that animates two galaxies, each with many stars orbiting them, using fastnbodyaccn and randcirclepts.

2 Review of Theory and Numerical Approach

2.1 Gravitational N-Body Problem

This is of interest because we intend to simulate n-many stars around different cores of galaxies. We need to be able to calculate each star's effect on each other.

We begin by considering N many point particles, each with index $i = 1, 2, ..., N$ and mass m_i. Each particle is in 3 dimensional space, and has position vector

$$\mathbf{r}_i(t) = [x_i(t), y_i(t), z_i(t)], \ i = 1, 2, ..., N$$

where the origin is at some arbitrary point. The main force we wish to study is the gravitational force each particle feels from all other particles. We can write this as

$$m_i\mathbf{a}_i = G\sum_{j=1,j\neq i}^{N}\frac{m_i m_j}{r_{ij}^2}\hat{\mathbf{r}}_{ij}$$
$$i = 1, 2, ..., N$$
$$0 \leq t \leq t_{max}$$

where $\mathbf{a}_i = \mathbf{a}_i(t)$ is the acceleration on the ith particle, G is the gravitational constant, and r_{ij} is the magnitude of the separation vector, calculated as

$$\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$$
$$r_{ij} = |\mathbf{r}_j - \mathbf{r}_i|$$

This formula for the acceleration can be rewritten by replacing the unit vector, to get a simpler form

$$m_i\mathbf{a}_i = G\sum_{j=1,j\neq i}^{N}\frac{m_i m_j}{r_{ij}^3}\mathbf{r}ij$$

If we use a coordinate system such that $G = 1$, we can non-dimensionalize the system of equations. We will also use the fact that the second derivative of position is acceleration, cancel out the mass m_i on both sides, and simplify, to get

$$\frac{d^2\mathbf{r}_i}{dt^2} = \sum_{j=1,j\neq i}^{N}\frac{m_j}{r_{ij}^3}\mathbf{r}_{ij}$$

We have obtained a second order differential equation for the vector $\mathbf{r}_i(t)$. We note it does not depend explicitly on mass, so we can use this on massless particles, as we will do later on. Initial conditions will also be required to compute a specific solution.

2.2 Finite Difference Approximation Solution

To solve this second order differential equation, first we must discretize the grid. The domain is $0 \leq t \leq t_{max}$ and we can rewrite it as a mesh using the level parameter, ℓ, as follows.

The number of time steps is

$$n_t = 2^\ell + 1$$

with the time steps being

$$\Delta t = \frac{t_{max}}{n_t - 1} = 2^{-\ell}t_{max}$$

and the nth time being

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

Next, the equations of motion must be discretized. Acceleration can be approximated using the second order centered formula

$$\left.\frac{d^2\mathbf{r}(t)}{dt^2}\right|_{t=t^n} \approx \frac{\mathbf{r}^{n+1} - 2\mathbf{r}^n + \mathbf{r}^{n-1}}{\Delta t^2}$$

We can substitute this into our previous ODE to get a formula for the future position vector, $\mathbf{r}_i^{n+1}$

$$
\begin{aligned}
\frac{\mathbf{r}_i^{n+1} - 2\mathbf{r}_i^n + \mathbf{r}_i^{n-1}}{\Delta t^2} &= \sum_{j=1, j\neq i}^{N} \frac{m_j}{(r_{ij}^n)^3}(\mathbf{r}_j^n - \mathbf{r}_i^n) \\
\mathbf{r}_i^{n+1} &= \Delta t^2 \left(\sum_{j=1, j\neq i}^{N} \frac{m_j}{(r_{ij}^n)^3}(\mathbf{r}_j^n - \mathbf{r}_i^n) \right) + 2\mathbf{r}^n - \mathbf{r}^{n-1} \\
&= \Delta t^2 \mathbf{a}_i^n(t) + 2\mathbf{r}^n - \mathbf{r}^{n-1}
\end{aligned}
$$

with n ranging from

$$n + 1 = 3, 4, ..., n_t$$

since the previous positions are needed to calculate the future positions. This gives us an iterable formula for the next position of a particle, given we know its current and previous position, as well as the acceleration it feels in its current position.

It is important to note that we need to deal with initial conditions. More specifically, we will need values for the first and second time steps, $\mathbf{r}_i^1 = \mathbf{r}_i(t = 0)$ and $\mathbf{r}_i^2 = \mathbf{r}_i(t = \Delta t)$. This can be done by using the Taylor Expansion to $O(\Delta t^3)$, as this will ensure our overall solution is $O(\Delta t^2)$ accurate. No proof is provided for this statement. Thus, proceeding from the initial conditions we set above, we get

$$
\begin{aligned}
\mathbf{r}_i(\Delta t) &= \mathbf{r}_i(0) + \Delta t \frac{d\mathbf{r}_i}{dt}\bigg|_{t=0} + \frac{\Delta t^2}{2}\frac{d^2\mathbf{r}_i}{dt}\bigg|_{t=0} + O(\Delta t^3) \\
&= \mathbf{r}_i(0) + \Delta t v_i(0) + \frac{\Delta t^2}{2}\mathbf{a}_i(0)
\end{aligned}
$$

Where we used our equation of motion to replace the second derivative, and the velocity to replace the first derivative. Recall we can choose what our initial conditions are, and now we can calculate what our second time step conditions should be.

2.3 Toomre Model

The Toomre model represents an isolated galaxy as a central core with mass m_c, and n_s many stars in circular orbit around it. These stars have vanishing gravitational mass, meaning they do not contribute to gravitational field. This greatly simplifies the calculations, as each star only feels the influence of the cores around it, and not of the other stars. This also means that only other nearby cores can influence the movement of a core.

2.4 Circular Orbit

To obtain circular orbit at a fixed distance, the initial positions and velocities must be set appropriately. Recall that, in circular orbit, the gravitational force and centripetal force perfectly balance, thus the particles do not come closer or go further away from each other. Suppose we have two particles in mutual circular orbit with one another. The origin is at the center of mass of the system, and does not move. The particles are of mass m_1, m_2, and their total mass is $m = m_1 + m_2$. Their positions are on the x axis, and are distances $r_1, -r_2$ from the origin, with their distance to each other being $r = r_1 + r_2$. They each have a tangential velocity which is completely in the y direction, by construction. They are $v_1, -v_2$. They two orbit counter clock wise. We also assumed all values outlined here are positive, and that $G = 1$. Thus, for particle 1, we get that

$$\begin{aligned}
F_g &= F_c \\
G\frac{m_1 m_2}{r^2} &= \frac{m_1 v^2}{r_1} \\
\frac{m_2}{r^2} &= \frac{v_1^2}{r_1} \\
v_1 &= \frac{\sqrt{m_2 r_1}}{r}
\end{aligned}$$

We can do a similar calculation for particle 2, to find that

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

The center of mass is $R = \frac{m_1 r_1 + m_2 r_2}{m_1 + m_2}$. We now write the total force equation as

$$m_1 r_1'' + m_2 r_2'' = (m_1 + m_2)R'' = F_{12} + F_{21} = 0$$

with $R'' = \frac{m_1 r_1'' + m_2 r_2''}{m_1 + m_2}$. We rearrange these to get that

$$R'' = r_1'' - r_2'' = \frac{F_{12}}{m_1} - \frac{F_{21}}{m_2} = \left(\frac{1}{m_1} + \frac{1}{m_2}\right) F_{12}$$

using the fact that $F_{12} = -F_{21}$. We then get

$$\frac{m_1 m_2}{m_1 + m_2} R'' = F_{12}$$

Solving this differential equation gives us the positions of r_1 and r_2

$$\begin{aligned}
r_1 &= R + \frac{m_2}{m_1 + m_2} r \\
r_2 &= R - \frac{m_1}{m_1 + m_2} r
\end{aligned}$$

We rearrange this, and use the fact that the center of mass of the system is at the origin, to find that

$$\begin{aligned}
r_1 &= \frac{m_2}{m} r \\
r_2 &= \frac{m_1}{m} r
\end{aligned}$$

These inital conditions will give us the desired circular orbits.

3 Numerical Approach

We begin by representing all positions of particles, stars or cores, as 1 by 3 vectors, each column representing the x, y, z component, respectively. Then, for a system with N many particles, an array of shape $N \times 3$ will represent a snapshot of all the particles at specific time. Each row represents a unique particle, and the columns the coordinates of that particle.

Our first function, **nbodyaccn.m**, takes an $N \times 3$ array,r, which represents a snapshot of a system. It also takes a $1 \times N$ array, m, which are the corresponding masses of each particle. Using this information, this function outputs an $N \times 3$ array of the acceleration each particle faces, in each direction. It does so using the formula listed above for the acceleration vector. It iterates over the entire array and assumes all particles contribute to the gravitational field.

To test this function, **tnbodyaccn.m** produces a simulation of two bodies in mutual orbit. It also traces out their path to confirm the orbit is indeed circular. One important thing to note is that we must implement our equations for the first and second time steps correctly in order for the two bodies to properly orbit each other. The first time step values were found using the right positions and velocities for circular orbit, detailed above. The second time step values were found using the Taylor Expansion also described above. This script first creates an $N \times 3 \times nt$ size array, where nt is the number of time steps, to store all the positions of particles at all times. This can be imagined as such; for some time step, nt, the corresponding slice of the 3D array is a snapshot of all the particles at that time. This allows us to keep track of the previous and current time steps. The initial conditions are set, and then the simulation is performed using a for loop and **nbodyaccn.m** to update the positions.

twobodysim.m is a function which does what **tnbodyaccn.m**, but in a function. It uses the same basic principle of storing the positions in a 3 dimensional array, and then iterating over it to update the positions, but it also takes the level and initial conditions as an argument. The level determines the number of time steps, but the maximum time t_{max} can be fixed based on input. This function then outputs the $\mathbf{r}$ vector with all the positions across all the time steps, as well as an array of just the x positions of 1 particle, chosen arbitrarily but consistently.

That simulation function is then used in **t2bodyconverge.m** to check for convergence. This was done using 4 levels, $\ell = 6, 7, 8, 9$, and the differences between levels were scaled and plotted. At each level, a simulation using **tnbodyaccn.m** is produced, and the x values are kept to plot with. All initial conditions were kept the same between levels.

Before producing the galaxy simulations, it was found that the previous function to find the accelerations was too slow. This was because it used nested for loops to calculate the acceleration. To remedy this, **fastnbodyaccn.m** was created, which implements the same equations of motion, but in a much more efficient way. Instead of looping over all the particles, **fastnbodyaccn.m** takes in an $N \times 3$ array for the star positions, and an $ncore \times 3$ array for the core positions. N is the number of stars, and $ncore$ is the number of cores. To prevent nested for loops, we loop over the array of core positions. For a core, an array is created that contains the position of that core, but N many times. This results in another $N \times 3$ array, containing just the position of this core. The reason this is done is so that the acceleration of the stars can be calculated with just one array operation, again, using the same equation of motion. Thankfully, MATLAB allows array operations to be done position-wise, so this reduces the need for nested looping. This is then repeated for all the cores. To determine the acceleration of the cores due to other cores, however, a nested for loop was used. The number of cores is going to be so small in our simulations, compared to the number of stars, that this will be a very negligible cost on computation time. This resulted in a much more efficient function.

randcirclepts.m produces the random positions of the stars in a galaxy. It takes a minimum and maximum radius for the position of the stars, thus allowing it to distribute stars in a "donut" shape. It also takes n, which is the number of stars to generate, as well as the position of the core these stars are to be distributed around. The positions are randomly generated as follows; first, a random angle is uniformly sampled n-many times; second, the the radius is randomly sampled and scaled to fit within the range give; third, the angle and radius are combined to create the x and y coordinates, using simple trigonometry. These positions, as well as the corresponding angles, are output. These angles will become important later.

A single galaxy core was first simulated to test the functions. By looking for circular orbit when both stationary and moving, we can confirm the code is working as intended. **onegalaxysim.m** takes advantage of the much faster **fastnbodyaccn.m** to calculate the acceleration and future positions of all the stars and core. The inital velocities of the stars are set using the same formulae as shown above, but this time, the positions were chosen randomly by **randcirclepts.m**. This meant that the formula for tangential velocity could give us a magnitude, but the specific vector direction needed to be calculated. Using the array of angles output by **randcirclepts.m** for all the stars, the required direction for tangential velocity would be $\pm\frac{\pi}{2}$ of this angle, which the choice determining if the stars would orbit clockwise or counter clockwise. The choice is arbitrary for now. The velocity vector was then found using trigonometry, and the galaxy displayed normal behaviour when stationary and at the origin. By changing the initial condition of the core, the entire galaxy can be shifted just by shifting everything by the same value. To get the stars to still orbit properly, the distances from the core to the star needed to be calculated carefully. Initially, I had the position of the star determine its tangential velocity for orbit. But I quickly realized that shifting my core would invalidate these calculations. So to fix this, the distance from the core to the star had to be used to determine tangential velocity. To make the galaxy as a whole move, an initial velocity needed to be given to all the particles. The stars remained in orbit of the core even when moving.

twogalaxysim.m produces a simulation and animation of two galaxies in motion. It takes advantage of the much faster **fastnbodyaccn.m** to do the calculations, as well as **randcirclepts.m** ability to produce random stars at any core position. Following the same steps as **onegalaxysim.m**, a simulation was created, and was working. But when trying to scale up the number of stars from 10 to 100 to 1000, the animation became much much slower. This was because of the plotting function used. It used a for loop over all the particles in r to plot, which became much slower when the number of particles increased. So the for loop was removed, and one plot command was used to plot all the stars at once. This greatly improved the speed of the particle, and, alongside the changes to **fastnbodyaccn.m**, allowed for even 10 000 stars per core to be simulated with 2000 time steps.

While different scripts were used to calculate the convergence testing and galaxy collisions portions, the equations of motion are the same.

4 Results

Below are some figures from the convergence testing and galaxy collision simulations. Some interesting shapes of the cores and stars are displayed. The initial conditions that resulted in these shapes were found through "brute force." The example collision gave a rough guide on how to achieve similar results. By positioning the galaxies in certain spots and choosing velocities such that their collisions are "glancing," the shapes of the galaxies seemed most peculiar. However, when the stars go too close to the cores, non-physical shapes were created. Through some simple guessing, initial conditions were found to create the plots below. The direction of orbit was also varied in the initial conditions, but more success was found when the orbits of the galaxies were in the same direction.

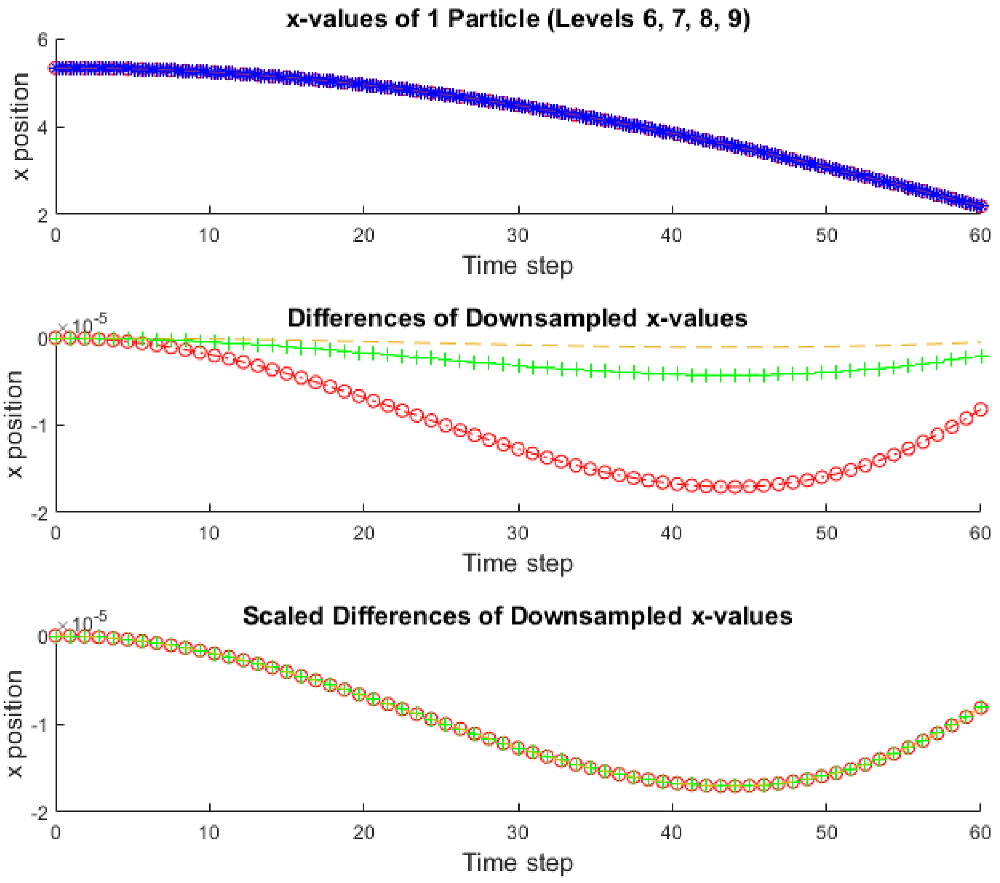

Figure 1: The plots above show the convergence test results for the equations of motion in a two body, mutual orbit, system. The top plot shows the x values for the different levels. The middle plot shows the difference in x values between levels, down sampled to compare the same points. The bottom plot shows the same comparison but with the differences scaled according to the level, with larger levels being scaled more. The plots show good convergence, as the bottom plot has near coincidence between the lines. The magnitude of the differences are all $O(10^{-5})$. As $t \to \infty$, the leading order term in the Finite Differences Approximation will be $O(\Delta t^2)$, so our implementation appears to have this same order in error.

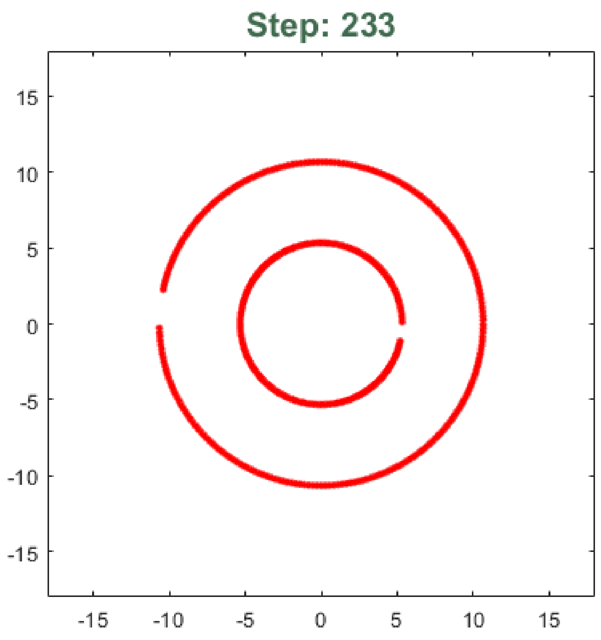

Figure 2: The plot above shows a snapshot of two bodies, each with distinct masses, in mutual orbit, with their path traced out. This was done to show that the equations of motion are indeed correct, and show the expected behaviour. The masses are rotating counter clockwise, and trace out circular orbits. The inner and outer circle each represent one of the masses.

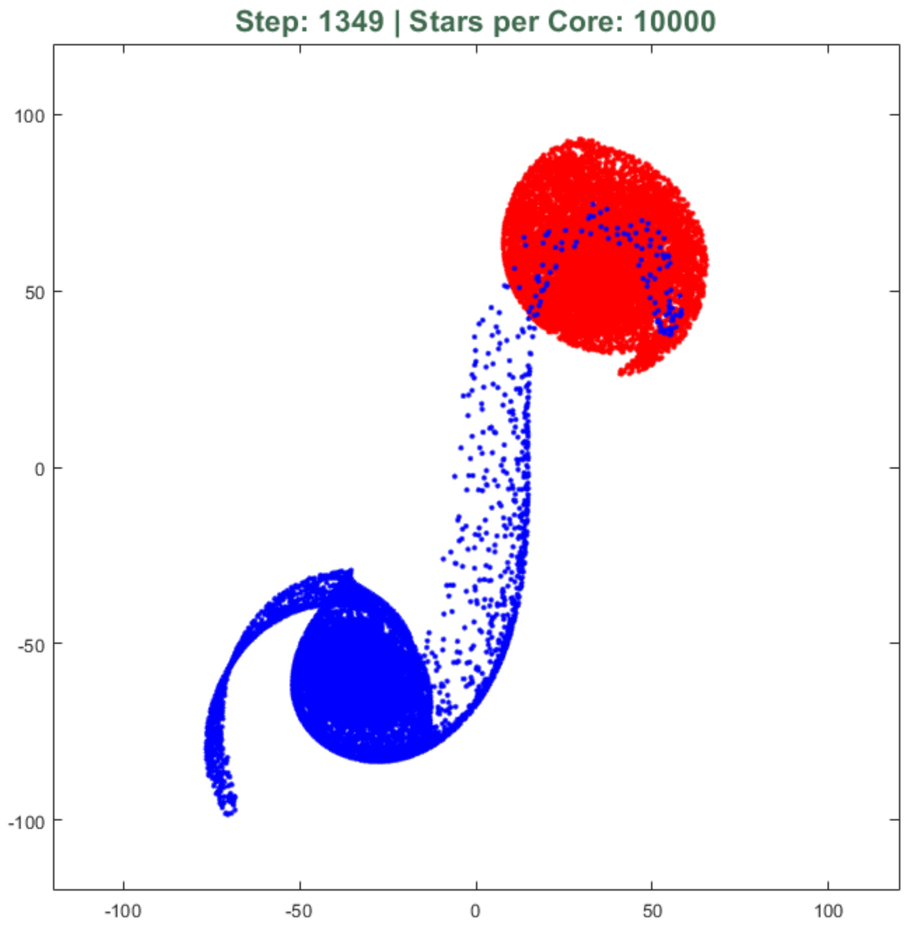

Figure 3: The plot above shows a snapshot of two galaxies interacting with each other. Each core has 10^4 stars orbiting it, and both cores have these same mass. This simulation was done with a level $\ell = 11$. The initial conditions were chosen such that this would happen.

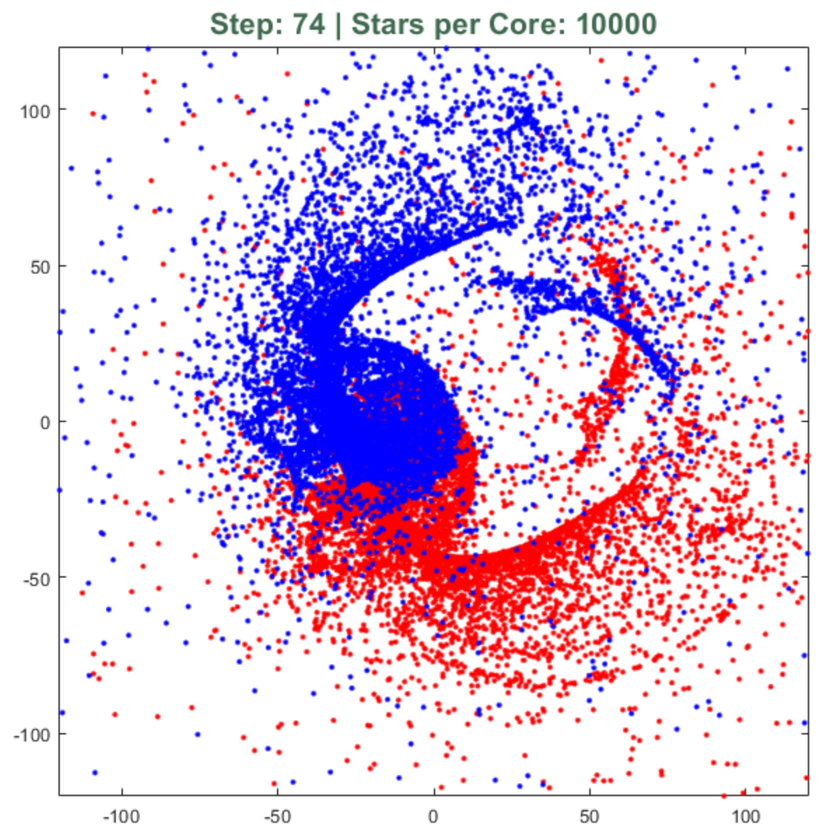

Figure 4: The plot above shows a snapshot of two galaxies interacting with each other in a non-physical way. This set of initial conditions ended up going beyond the limits of the FDA. For example, the stars go too close to the cores, and they gained a large acceleration and shot off.

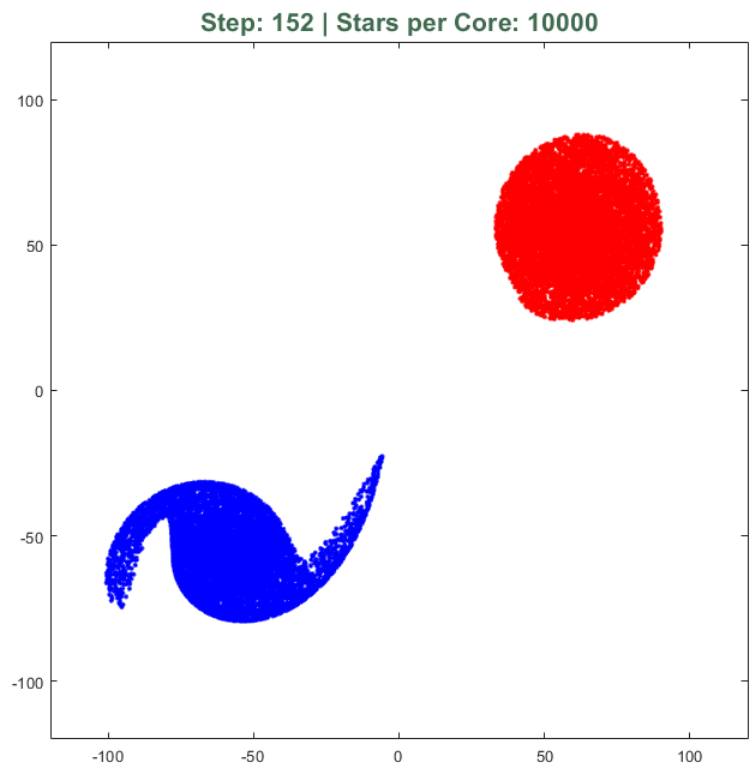

Figure 5: The plot above shows a snapshot of two galaxies interacting, resulting in interesting shapes of the galaxies.

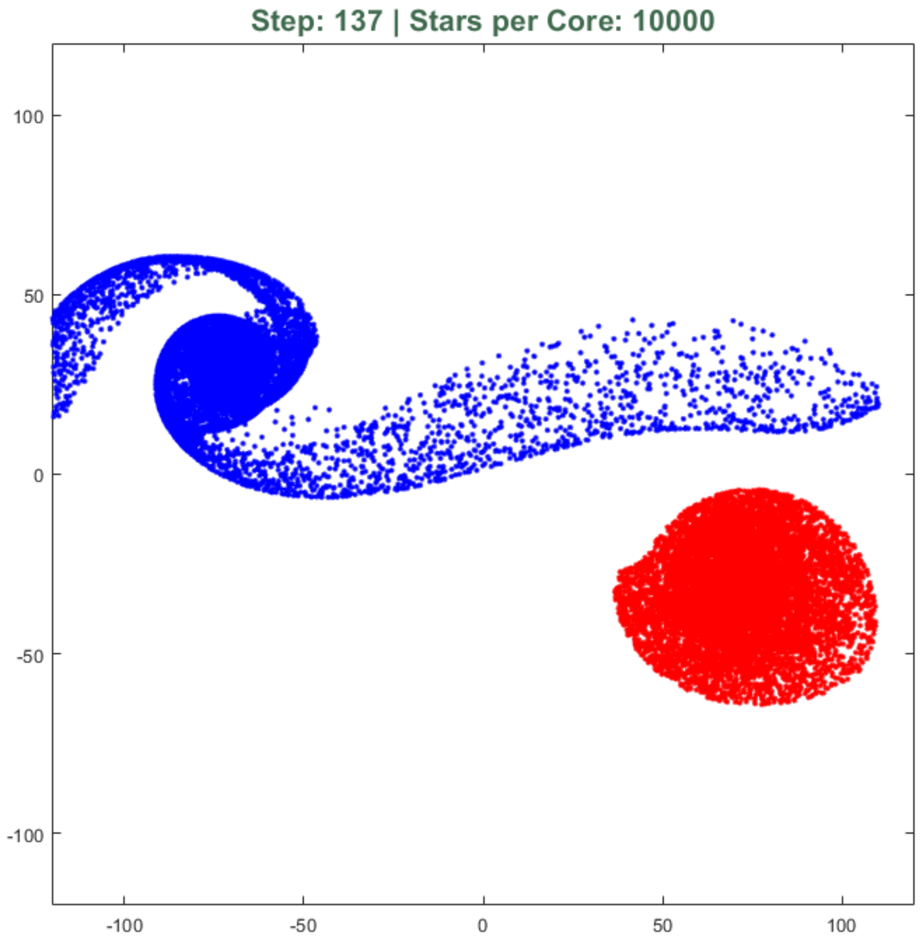

Figure 6: The plot above shows a snapshot of two galaxies interacting, resulting in interesting shapes of the galaxies.

5 Discussion

The equations of motion implemented show good convergence, and the scripts that test it display expected behaviours. The galaxy simulations, though simple in construction, appear to be a good tool to study the possible shapes galaxies can have when in near proximity with one another. This near proximity causes stars in either galaxy to feel the forces of each other, leading to these peculiarities. Despite this, the limits of the model, and Finite Difference Approximations, can be seen in the non-physical shapes that would show up.

An interesting discovery was the minimum radius needed for stars to not fly towards the core, leading to them shooting off. In my simulation, I found the closest r_{min} was $r_{min} \geq 2$. Anything less than 2 and the stars would shoot off into the core. For the 2 galaxy simulation, I used $r_{min} = 5$ and changed the size of the marker for the core to be large enough to fill in the gap between the stars and core. In addition, I also found that, for some single galaxy experiments, the simulation would not behave as expected when the level $\ell \leq 6$. This could be due to the large t_{max} set, resulting in very large Δt being used. I did not try $\ell > 11$, but there may be some lower limit to the size of Δt before the FDA breaks down.

One area of pain when implementing the code was optimization. Since the number of stars is so large, especially with two galaxies, the code must be efficient, else the simulation will take far too long to run. The largest contributing factor to improving this was using array operations instead of loops. This reduced the time it took to, say, calculate the acceleration on each particle, or the next set of positions, by a great amount. In addition, the plotting process itself was very time consuming at first, due to the loops used. But by passing through the appropriate array index, the plotting became much faster. Initially, the simulation of two galaxies took 15 minutes to run, at a level $\ell = 6$. But by making the two fixes above to make it more efficient, it took seconds to run and produce the animation.

Another struggle was determining the right tangential velocity to have that would result in circular orbit. The struggle came with trying to get the angle and position of a star relative to the core. Though I tried many different ways to try and calculate it based purely off of the two positions, the easiest way was for the **randcirclepts.m** function to just output it. This saved a lot of calculation headache, and gave me a way to find the necessary components of tangential velocity.