Project 1 - PHYS 410

Yuriel Dimayacyac

October 2025

1 Introduction

In computational astrophysics, understanding how galaxies move and interact is a key part of learning how the universe evolves. This report presents a finite-difference simulation of galactic motion based on the Toomre model, a simplified approach that reduces the complex equations of motion into a more computationally-friendly form. In this model, each galaxy is represented by a massive core and several surrounding stars. The goal of this simulation is to recreate the familiar shape of the Antennae Galaxies and show that their structure can emerge purely from gravitational interactions.

The simulation was built in MATLAB using a second-order finite difference algorithm to track how each particle moves over time. Beyond the academic setting of PHYS 410, this project reflects an ability to apply computational methods to physical systems and demonstrates an adaptable approach to problem-solving in astrophysics.

2 Review of Theory

2.1 The Toomre Model

The Toomre Model describes a galaxy with one massive core and an n_s number of stars with vanishing gravitating mass. The stars in this model experience acceleration due to gravity from the core, but their vanishing mass makes the gravity they exert on the other stars and cores negligible. This means that the cost of the simulation is only $O(N)$ for $O(N)$ stars because the gravitational effects from a star are not accounted for in acceleration calculations.

2.2 Equations of Motion

Two equations were used to perform the simulation. Equation (1) below is the differential form of Newton's second law, equated to the law of gravitation, assuming that the net force of the bodies is only the gravitational force exerted on that body due to the structure of the Toomre Model.

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

This equation can be simplified to exclude m_i and be represented as a differential equation, as shown in equation (2) below.

$$\frac{d^2\vec{r_i}}{dt^2} = \sum_j \frac{m_j}{r_{ij}^3}\vec{r_{ij}}$$

This equation is now a differential equation in terms of the vector r_i, where r_i describes the position of the i-th body in space.

$\frac{d^2\vec{r_i}}{dt^2}$ is used to approximate position r_i at future times with the second order centered formula.

$$\frac{d^2\vec{r}}{dt^2} = \frac{\vec{r^{n+1}} - 2\vec{r^n} + \vec{r^{n-1}}}{\Delta t^2}$$

For each time iteration $n+1$, the position of the body is solved using the acceleration value at time n. This process is iterated for all times $t \in [0, tmax]$, where $tmax$ is a predefined value by the user. As a result, the position r represents the position of the body in x, y, z at all discrete times within the given range.

3 Numerical Approach and Implementation

3.1 Finite Difference Grid

The finite difference grid is used to translate continuous motion into discrete motion, which can easily be computed using numerical methods. The time ranges from $t \in [0, tmax]$, and has $2^l + 1$ steps, where l is the discretization level parameter defining the resolution of the simulation. A higher l value corresponds with a more continuous-looking plot. The use of this grid conveniences integration of the aforementioned differential equations, and is ultimately used to solve the position of each body at different times.

3.2 Convergence Testing

Before programming the main simulation, the finite difference approximation algorithm, using the two aforementioned equations of motion, was verified for accuracy. This was necessary because truncation errors can accumulate over time with each discrete approximation of the second derivative, leading to a solution that does not converge correctly. The error in position values between discretization levels is inherently sinusoidal, as it follows a similar form to the true oscillatory motion of the system, but with a different frequency due to the numerical approximation method.

The error between discretization levels should scale with l. For example, if $l = 8$ is adjusted to $l = 9$, the number of time steps doubles (recall that $n \propto 2^l$). Since the position of the body is computed using a second-order scheme, the error should scale by a factor of $2^2 = 4$.

Thus, a convergence test was performed by comparing the numerical solutions at refinement levels from $l = 7$ to $l = 9$, values that were chosen because they provide sufficiently small time steps for near-continuous-looking motion. The x-position of one core in a 2-core, circular orbit system was analyzed. The following is a plot of the discretization error between levels $l = 7, 8$ and $l = 8, 9$.

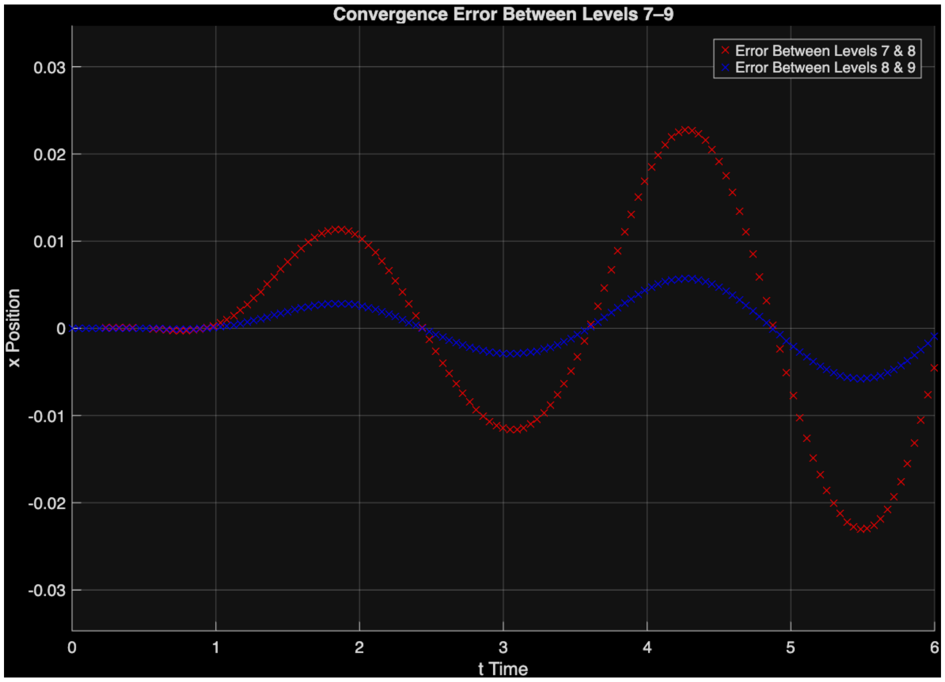

Figure 1: Plotting both error functions with no scaling

As visualized, the error between levels 8 and 9, (plotted in red), follow a similar oscillatory pattern as the error between levels 7 and 8, (plotted in blue), since all other simulation parameters were held constant while l was refined. To verify that the solution converged, the error from levels 8 and 9 were scaled by the ideal factor of 4 and compared against the error graph corresponding to levels 7 and 8. These curves closely align with one another, so convergence is confirmed.

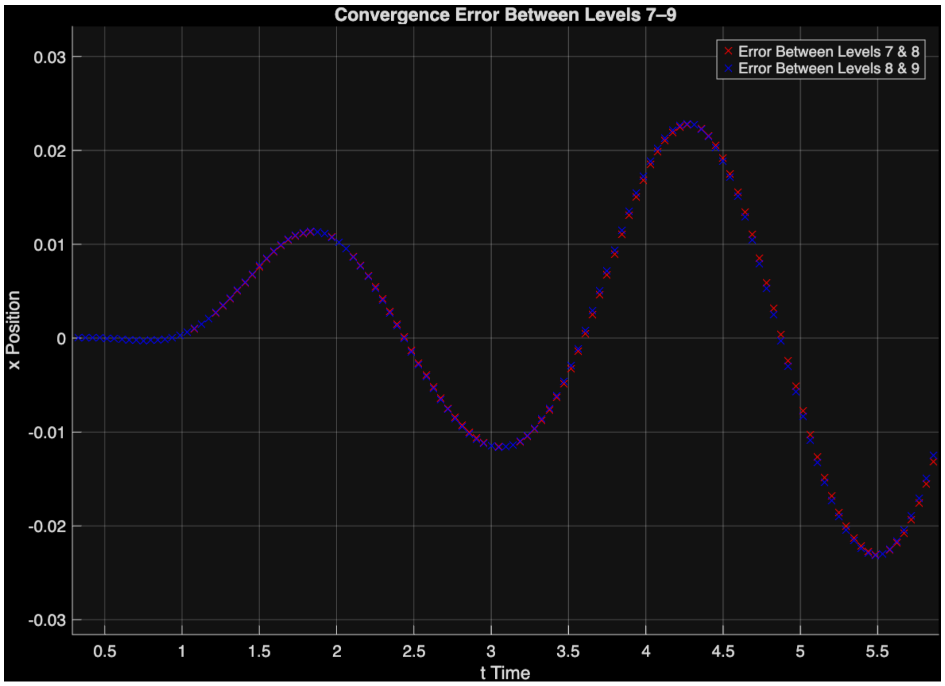

Figure 2: Plotting both error functions with scaling

In order to eliminate human error from observing a visual graph, the simulation includes code that prints the factor between the median of the error between levels 8 and 9 and the median of the error between levels 7 and 8. Following the same reasoning as before, this value is expected to be 4, and the code displayed that the factor is, indeed, around 4.

Testing Convergence on Initial Code

```
zero_filter = abs(dx89_core1) > 1e-12; % Avoid division by 0
ratio_pointwise = zeros(size(dx89_core1));
ratio_pointwise(zero_filter) = dx78_core1(zero_filter) ./ dx89_core1(zero_filter);

fprintf('Median ratio dx78/dx89 = %.3f\n', median(ratio_pointwise(zero_filter)));
fprintf('||dx89||_inf / ||dx78||_inf = %.3f \n', ...
    norm(dx89_core1, inf) / max(eps, norm(dx78_core1, inf)));
```

Result of Convergence Test

```
Median ratio dx78/dx89 = 3.986
||dx89||_inf / ||dx78||_inf = 0.251
```

3.3 Initializing each galaxy

The Toomre model requires that the stars follow circular orbits around their core, so their velocities were initialized to follow the assumption that the gravitational force on a body was its centripetal force $F_G = F_c$. The following derivation uses this equation to get equations for the velocity of each body in a 2-body gravitational system.

$$F_G = \frac{G * m_1 * m_2}{r^2}$$
$$F_c = \frac{mv^2}{r}$$
$$\frac{G * m_1 * m_2}{r^2} = \frac{mv^2}{r}$$

From non-dimensionalizing the system, the simulation works with values that represent relationships between order of magnitude. This simplifies the computation and makes this applicable to, potentially, other systems. As a result, the following is true for the simulation:

$$G = 1$$

For body 1,

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

For body 2,

$$v_2^2 = \frac{G * m_1 * r_2}{r^2} \rightarrow v_2 = \frac{\sqrt{m_1 * r_2}}{r}$$

These equations are used in the code of *initializegalaxy.m* to initialize both stars and cores. To verify whether these equations reflect circular motion, the following snapshot shows the simulation with only one galaxy initialized. (Note that the y-axis is stretched, resulting in an image that does not appear perfectly symmetrical.)

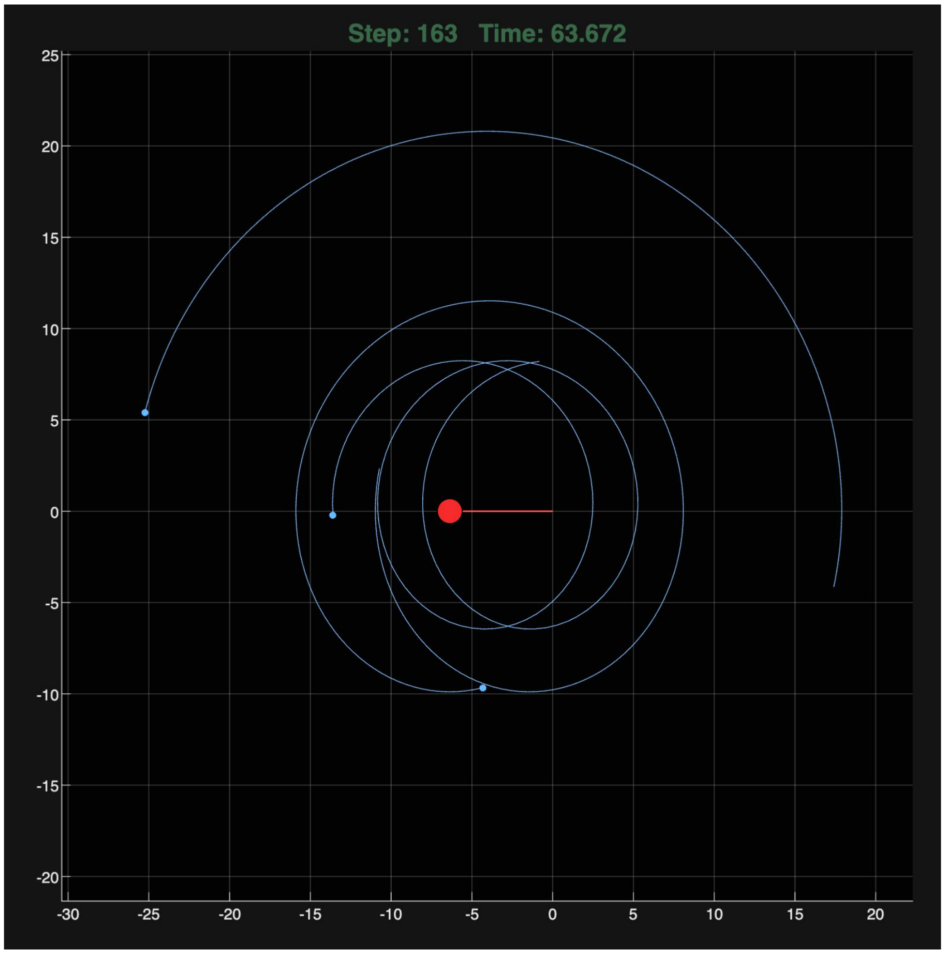

Figure 3: One Galaxy Simulation with Circular Motion

When adding stars to the system, it is important to note that stars have vanishing mass. Their respective cores do not have a circular orbit velocity, as they are not expected to orbit around the stars. The cores, however, are initialized with a velocity that allows them to reach each other's proximity in order to display the gravitational effects of the collision. Because the cores have an initial velocity, the velocity of stars must sum this velocity, as the previous equations only give the velocity of the stars in the core's inertial time frame.

```
    v_star(i,:) = v_mag*t_direction + v_core; % initial velocity of star should be
        relative to the core's velocity
```

The position of each galaxy was set to a distance where the gravitational force of the opposing galaxy was negligible. This is so the gradual effects of the opposing galaxy's gravitational field can be observed as the cores and stars approach each other. To implement this, the cores were separated by a distance 200 on the x-axis. They were given x-velocities such that they would slowly, without losing their grip on their stars, move closer to each other. They were also given a y-component of velocity, so they would not collide as soon as they met the same x-coordinate.

4 Results

The simulation was conducted with the following initial conditions:

- Core 1 position: $(x, y) = (-100, 0)$
- Core 2 position: $(x, y) = (100, 0)$
- Core Masses: $m_1 = 50, m_2 = 50$
- Stars distributed randomly about each core

The resulting figure shows a snapshot of the animation (the full animation can be viewed here: Link to AVI Animation.

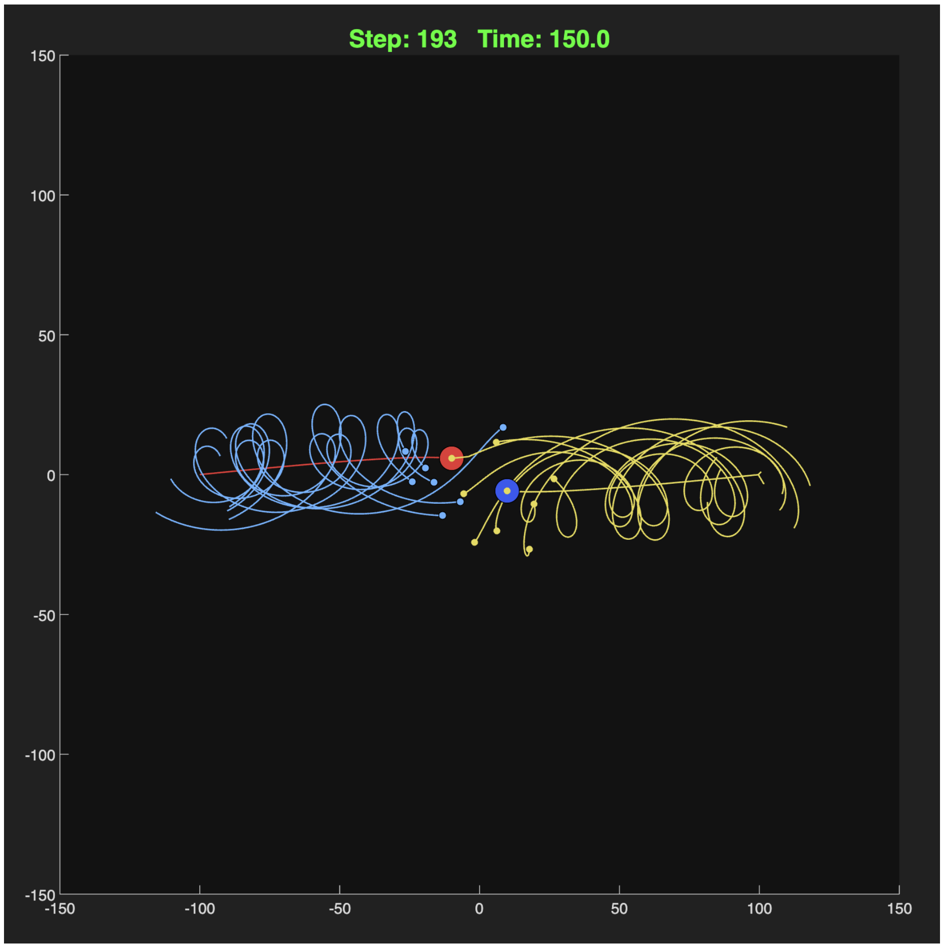

Figure 4: Snapshot of Simulation

The figure below displays the behaviour of the two galaxies a few time steps after they reach proximity of one another. The graph shows that, as the cores interact with one another, the stars stray from their trajectory, either flying away if they gain enough energy from the gravity of the cores or getting consumed by the cores. This can explain the structure of Antennae Galaxies, where the motion caused by the gravity from massive moving cores causes surrounding stars to form ’swooping’ structures, like the ones seen in the graph. These visual features can be compared to the provided picture of an Antennae Galaxy. One limitation of this MATLAB simulation is that it does not show how the mass of a star can start to tear off its body due to the intense gravitational pull of the core. This is a big feature of the Antennae structure that is not revealed by the MATLAB simulation; however, we get to visualize the general motion through this program.

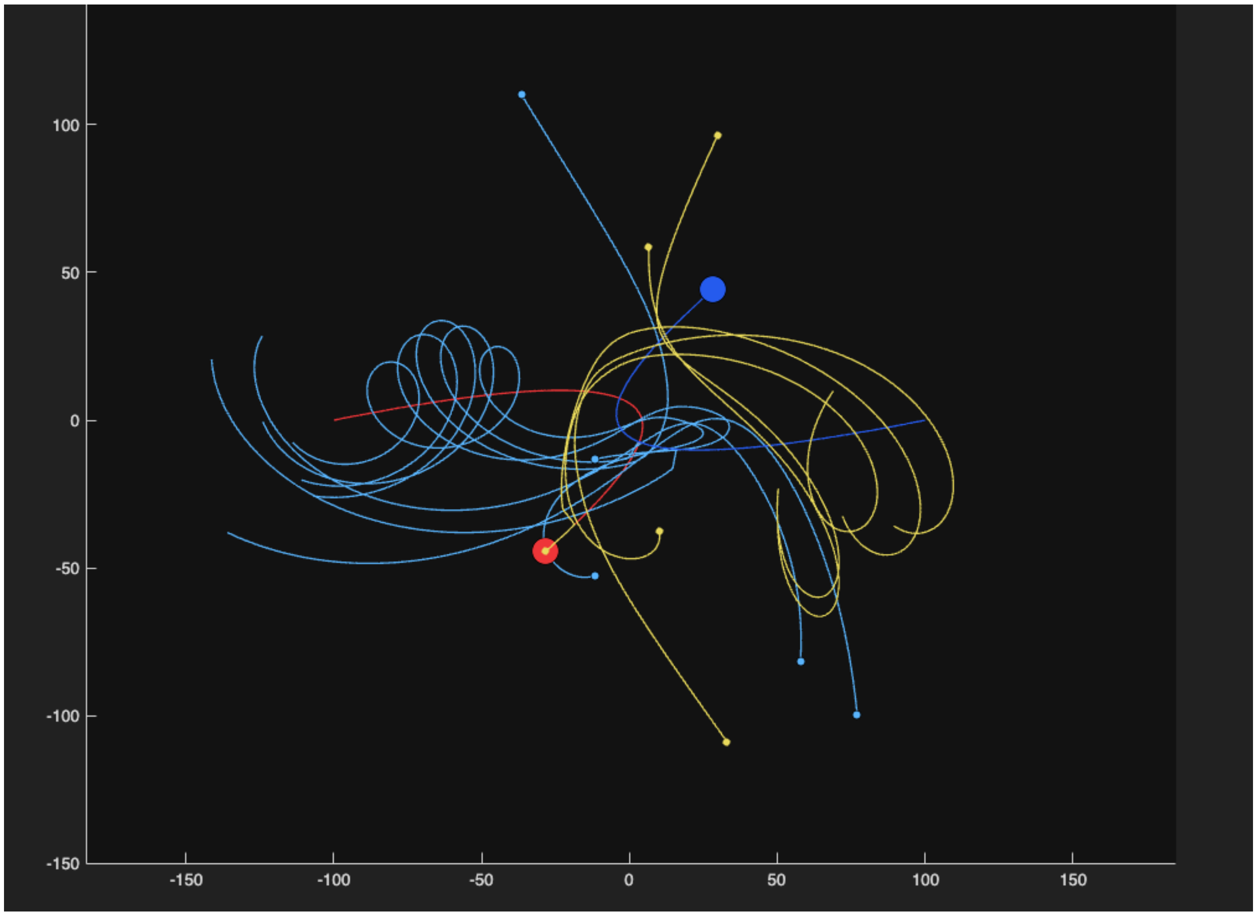

Figure 5: Antennae Structure

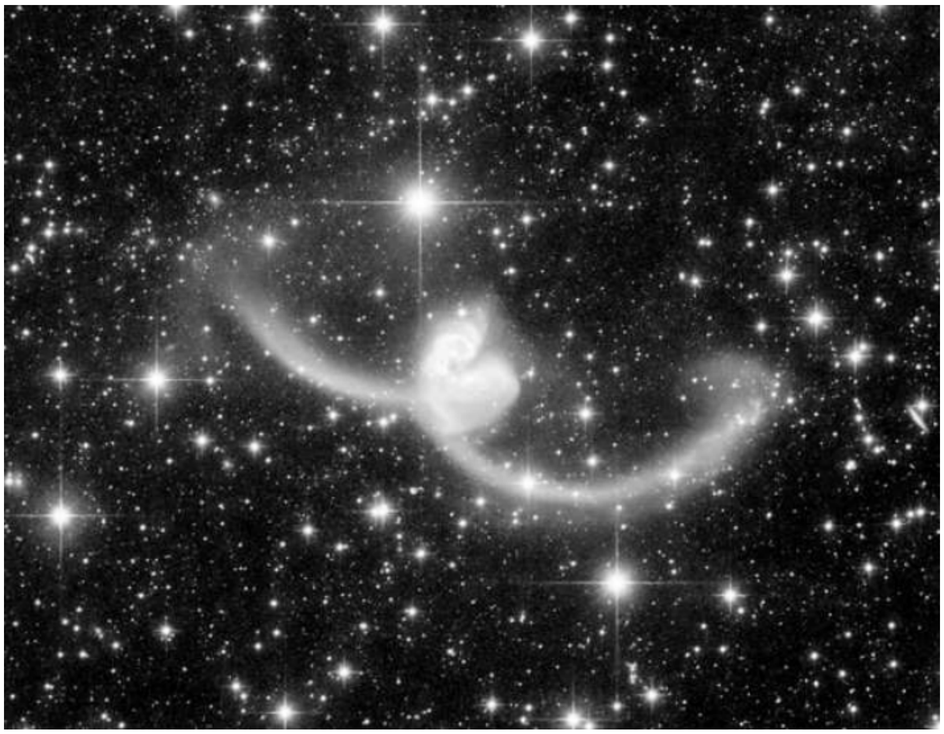

Figure 6: Figure 1 from Assignment Handout (NGC4038-39)

5 Discussion/Conclusions

One of my main errors was with convergence testing, where I realized that my code did not meet the expected second-order convergence testing. I noticed this when simulating two cores in my simulation and plotting their x-coordinates over time. Keeping everything constant, I changed the l-level value and interpolated their results onto the same time vector. As a result, I noticed that the error of the level 7-8 graph did not scale by 4 to coincide with the error of the level 6-7 graph. This should happen because, following a second-order algorithm, increasing the time resolution by 2 should ideally yield a factor of 4 jump in the plot. However, by using the following test code, I determined that there was a factor of 2.227 between my plots.

I also noticed that initialization was a key factor in the success of my simulation. If I set the core to move too fast, it will travel faster than the binding energy allows; thus, the stars appear to follow a corkscrew trajectory. However, the stars still circle the core according to the core's inertial frame, but from an outsider perspective, they appear as such. In order to show circular motion of the stars, the velocity of the core must be the same order of magnitude as the circular motion of the stars, so I set the velocity of the core to be 0.1. To be safe, the velocity of each star was implemented to be offset by the velocity of its corresponding core, as according to the relative velocities equation.

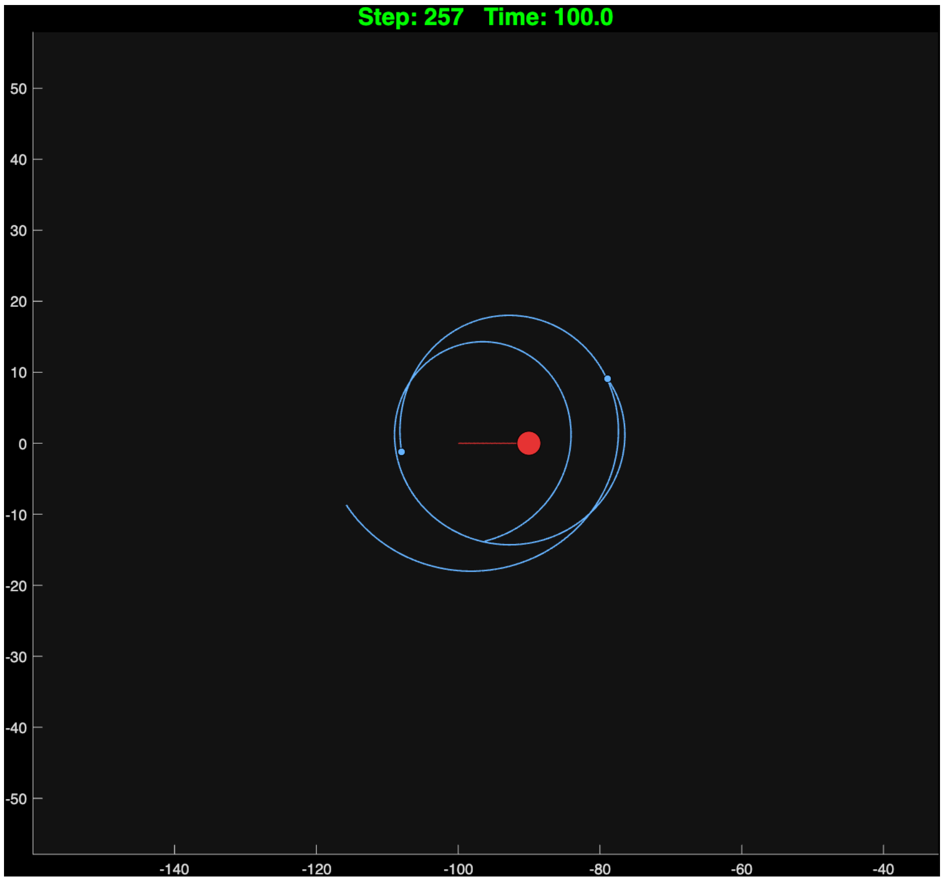

Figure 7: Galaxy with stars following circular motion

Selecting the appropriate mass and distance parameters to reproduce accurate motion was also a big challenge. Because the simulation used nondimensionalized units, there was no absolute scale for such quantities. Several tests with invalidly initialized values caused unexpected behaviour. This was mostly due to a lack of understanding of how nondimensionalized units worked in computational simulations. Now, I recognize that one of their benefits is the implementation of order of magnitude in calculations, so keeping that in mind, I made sure to keep related values close in order. An example of this appears in my code, where I used the mass of the core (its order of magnitude) to define a range that the position of a star should be bounded by. This helped ensure that circular motion was present in my simulations.

```
% Assign min val and max val according to order of mass
minVal = -m_core;
maxVal = m_core;

% Generate random initial 3D position vector from min val to max val
r_star = r_core + [minVal + (maxVal - minVal)*rand(n_stars,2),zeros(n_stars,1)];
size(r_star);
```

There are notable errors in this simulation, namely, the inaccuracy of energy conservation. This finite-difference scheme does not exactly conserve energy, but since we are using this at sufficiently small Δt, the error in total energy will oscillate around a mean value, so there is numerical stability.

In terms of accuracy, some assumptions were made to finalize this simulation. A major error in this simulation was the behaviour of the system when the distance between two bodies approaches 0. The acceleration equation tends to blow up at this limit, so I had to intervene this condition with the assumption that stars will be consumed by a core when they are too near. The implementation from *script_project1_twogalaxies.m* is shown below:

```
% Star - Core reaction
        for i = 1 : N_star

            total_acceleration = zeros(1,3);

            capture_core = 0;

            for j = 1: N_cores

                % Calculate acceleration based on gravity
                accel = nbodyaccn(r_s(i,:,n), r_c(j,:,n), mass_c(j));

                % Superpose acceleration of i-th star due to j-th core
                total_acceleration = total_acceleration + accel;

                if dot(squeeze(r_s(i,:,n))-squeeze(r_c(j,:,n)), squeeze(r_s(i,:,n)
                    )-squeeze(r_c(j,:,n))) < capture_radius^2
                    capture_core = j;
                end

            end

            % Add advanced position into position vector
            if capture_core ~= 0
                r_s(i,:,n) = r_c(capture_core,:,n);
                r_s(i,:,n+1) = r_c(capture_core,:,n+1);
            else
                r_s(i,:,n+1) = 2*r_s(i,:,n) - r_s(i,:,n-1) + total_acceleration *
                    delta_t^2;
            end
        end
```

The use of generative AI was limited to searching native MATLAB functions that could perform tasks more efficiently. An example was the use of the *rand*() function, as seen in a snippet from *initializegalaxy.m* below:

```
    r_star = r_core + [minVal + (maxVal - minVal)*rand(n_stars,2),zeros(n_stars,1)
        ];
```

Generative AI helped me learn how to range random numbers, allowing me to effectively test different initial star positions and confirm the flexibility of my code. Tools like ChatGPT were not used to generate any functions or algorithms.

Overall, the simulation successfully reproduced the qualitative behaviour of a two-galaxy interaction, demonstrating how Antennae Galaxies form their fascinating structures. The results highlight how the equations of motion can be simulated through Finite Difference methods to understand physical phenomena. This project also emphasized the importance of selecting physically consistent parameters, enforcing the connection between rigorous theoretical modeling and computational methods, especially in astrophysical simulations.