# PHYS 410 - Project 1

Felipe Garavelli
46427951

October 17, 2025

## Contents

**1 Introduction** **2**

**2 Theory** **2**
2.1 N-Body Problem . . . . . 2
2.2 Toomre Model . . . . . 3

**3 Numerical Approach** **3**
3.1 Finite Difference Grid . . . . . 3
3.2 Initial Conditions for Two-Body Circular Orbit . . . . . 4
3.3 Convergence Testing . . . . . 5

**4 Implementation** **7**
4.1 N-Body Solver Function . . . . . 7
4.2 Acceleration Calculation Function . . . . . 8
4.3 Simulation Setup . . . . . 8

**5 Results** **9**
5.1 Simulation 1 . . . . . 10
5.2 Simulation 2 . . . . . 11

**6 Conclusion** **12**
6.1 Use of AI . . . . . 12

**7 Files** **12**

**A MATLAB Code Listings** **14**
A.1 N-Body Solver Function (`nbody.m`) . . . . . 14
A.2 N-Body Acceleration Function (`nbodyaccn.m`) . . . . . 15

# 1 Introduction

This project investigates the dynamics of galactic collisions through a simplified numerical simulation implemented in MATLAB. The model is inspired by the work of Alar and Jüri Toomre in the early 1970s, who demonstrated that even rudimentary gravitational models can reproduce the large-scale morphological features observed in interacting galaxies, such as the famous Antennae Galaxies. While the present model captures the essential physics governing tidal distortions produced by mutual gravitational attraction, it omits many complex effects. Such as hydrodynamics, dark matter halos, and self-gravity within each galaxy.

The primary objective of this project is to simulate and visualize the evolution of two interacting galaxies, observing the emergence of characteristic tidal features and examining how parameters such as mass ratio and initial conditions affect the resulting morphology.

# 2 Theory

## 2.1 N-Body Problem

In the $N$-body problem we are interested in finding the positions $\mathbf{r}_i$ through time of $N$ particles with masses $m_i$ that interact gravitationally with one another. This problem can be described by using Newtonian gravitational forces. Newton's second law states that $m_i\mathbf{a}_i$ is the sum of all forces on particle $i$. Newton's law of gravitation states that the force between two particles is given by:

$$\mathbf{F}_{ij} = G\frac{m_i m_j}{r_{ij}^3}(\mathbf{r}_j - \mathbf{r}_i). \tag{1}$$

We can use equation (1) and Newton's second law to write the acceleration of particle $i$ as:

$$\mathbf{a}_i = \sum_{j\neq i} \frac{Gm_j}{r_{ij}^3}(\mathbf{r}_j - \mathbf{r}_i), \tag{2}$$

where $r_{ij} = \|\mathbf{r}_j - \mathbf{r}_i\|$ is the distance between particles $i$ and $j$.

We also can re-write $\mathbf{a}_i$ as the second derivative of position with respect to time:

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

and combining this with equation (2) we get the following second order differential equation:

$$\frac{d^2\mathbf{r}_i}{dt^2} = \sum_{j\neq i} \frac{Gm_j}{r_{ij}^3}(\mathbf{r}_j - \mathbf{r}_i). \tag{3}$$

Now we can also use the finite difference method to approximate the second derivative of position with respect to time using the second order centered formula,

$$\left.\frac{d^2\mathbf{r}_i(t)}{dt^2}\right|_{t=t^n} = \frac{\mathbf{r}_i^{n+1} - 2\mathbf{r}_i^n + \mathbf{r}_i^{n-1}}{\Delta t^2}, \tag{4}$$

where $\mathbf{r}_i^n \equiv \mathbf{r}_i(t^n)$ and $t^n = n\Delta t$ for $n = 0, 1, 2, \ldots, n_t$. Combining equations (3) and (4), and setting $G = 1$, we get the following finite difference equation for the $N$-body problem:

$$\frac{\mathbf{r}_i^{n+1} - 2\mathbf{r}_i^n + \mathbf{r}_i^{n-1}}{\Delta t^2} = \sum_{j \neq i} \frac{m_j}{(r_{ij}^n)^3} \left(\mathbf{r}_j^n - \mathbf{r}_i^n\right), \qquad n+1 = 3, 4, \ldots, n_t \tag{5}$$

Solving for $\mathbf{r}_i^{n+1}$ in equation (5) gives the update equation:

$$\mathbf{r}_i^{n+1} = 2\mathbf{r}_i^n - \mathbf{r}_i^{n-1} + \Delta t^2 \sum_{j \neq i} \frac{m_j}{(r_{ij}^n)^3} \left(\mathbf{r}_j^n - \mathbf{r}_i^n\right), \qquad n+1 = 3, 4, \ldots, n_t \tag{6}$$

Because this second-order system requires the first two positions to start the iteration, the following initial conditions are needed. The first position is given directly by the initial condition, $\mathbf{r}_i^1 = \mathbf{r}_i(0)$. The second position, $\mathbf{r}_i^2$, can be estimated using a Taylor series expansion, as in Tutorial 3 for the pendulum:

$$\mathbf{r}_i^2 = \mathbf{r}_i^1 + \mathbf{v}_i^1 \, \Delta t + \frac{1}{2}\mathbf{a}_i^1 \, \Delta t^2, \qquad n = 1, 2, \ldots, n_t - 1 \tag{7}$$

where $\mathbf{a}_i^1$ is the acceleration at the first time step, computed using equation (2).

### 2.2 Toomre Model

With the update equation (6) and the initial conditions described above we can simulate the motion of $N$ particles interacting gravitationally. If we want to simulate two galaxies colliding with one another utilizing the equation (6) would be computationally expensive since each star would interact with every other star in the simulation. Toomre simplified this problem by modeling each galaxy as a massive core with mass $m_{ci}$, each core surrounded by $n_{si}$ massless stars. This simplification reduces the number of required interactions, as only the cores interact with one another, while the stars interact only with the cores.

Since the stars have vanishing gravitational masses calculating their initial velocities is straightforward. In the project we assumed that all galaxies have circularly orbiting stars around their cores. We can easily derive the circular velocity of a star by equating its net force to the centripetal force required for circular motion,

$$F_{net} = \frac{G m_s m_c}{r^2} = \frac{m_s v_{circ}^2}{r}.$$

Therefore the velocity of each star with respect to its core with mass $m_c$ is given by:

$$v_{circ} = \sqrt{\frac{G m_c}{r}}.$$

## 3 Numerical Approach

### 3.1 Finite Difference Grid

To solve the $N$-body problem, we must discretize the continuous time domain $0 \leq t \leq t_{max}$ (where $t_{max}$ is chosen by the user). We discretize this domain using a uniform time mesh, where the positions $\mathbf{r}_i$ and velocities $\mathbf{v}_i$ are evaluated at a finite number of time points.

To implement and test convergence (Section 3.3), we define this grid using a level parameter, $\ell$. This parameter controls the resolution of our time mesh. The total number of time points, $n_t$, is given by:

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

This defines a set of $n_t$ discrete times $t_n = (n-1)\Delta t$ for $n = 1, 2, \ldots, n_t$. The uniform time step, $\Delta t$, is determined by the total time $t_{max}$ and the level $\ell$:

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

Increasing the level parameter $\ell$ by one will double the number of time points and halve the time step $\Delta t$, which is ideal for testing the convergence order of our solver.

I'd like to acknowledge that this assumption of a constant $\Delta t$ may not be optimal for all simulations. During close encounters, particles "clump" and experience rapidly changing accelerations, which would ideally be resolved with a much smaller, adaptive time step. However, for this project, this fixed-step approach is convenient and robust.

## 3.2 Initial Conditions for Two-Body Circular Orbit

To validate the $N$-body solver's basic functionality prior to convergence testing, we first simulated the case of a two-body problem. The goal was to ensure the code could reproduce a stable, circular orbit before conducting a more rigorous analysis. This was accomplished by establishing a circular orbit for two point masses, $m_1$ and $m_2$, separated by a distance $R$. We work in the center of mass (CoM) frame, placing the CoM at the origin $(0, 0, 0)$.

The gravitational force $F_g$ between the two masses provides the necessary centripetal force $F_c$ for each body to maintain its orbit.

$$F_g = \frac{G m_1 m_2}{R^2}$$

The radii of the individual orbits around the CoM are $r_1$ and $r_2$, where $R = r_1 + r_2$. From the definition of the CoM, $m_1 r_1 = m_2 r_2$. Solving for $r_1$ and $r_2$ gives:

$$r_1 = \frac{m_2}{m_1 + m_2} R \quad \text{and} \quad r_2 = \frac{m_1}{m_1 + m_2} R \tag{8}$$

The centripetal force for each mass is $F_{c1} = m_1 v_1^2 / r_1$ and $F_{c2} = m_2 v_2^2 / r_2$. Setting $F_g = F_{c1}$ and $F_g = F_{c2}$ and solving for the orbital speeds $v_1$ and $v_2$, we find:

$$v_1 = \frac{\sqrt{G m_2 r_1}}{R} \quad \text{and} \quad v_2 = \frac{\sqrt{G m_1 r_2}}{R} \tag{9}$$

To test the solution, we align the particles on the $x$-axis and direct their initial velocities entirely along the $y$-axis. This configuration is chosen to produce a stable circular orbit in the $x$-$y$ plane. The specific values used for this test are detailed in Table 1, providing a reliable benchmark for our solver.

Using the initial conditions detailed in Table 1, the $N$-body solver was run for a total time $t_{max}$ (corresponding to several orbital periods). The resulting trajectories of the two cores are plotted in Figure 1. The plot clearly shows that the particles maintain a stable, circular orbit, which validates the basic functionality of our solver.

Table 1: Initial conditions for the two-body validation test case. The values for $r_1, r_2, v_1, v_2$ are calculated using equation (8) and (9) in Section 3.2 with $R = 1$ and $G = 1$.

| **Body** | **Mass** ($m$) | **Initial Position** ($\mathbf{r}(0)$) | **Initial Velocity** ($\mathbf{v}(0)$) |
|---|---|---|---|
| Core 1 | $m_1 = 1.0$ | $(\ \ 0.333, 0, 0)$ | $(0, \ \ 0.408, 0)$ |
| Core 2 | $m_2 = 0.5$ | $(-0.667, 0, 0)$ | $(0, -0.816, 0)$ |

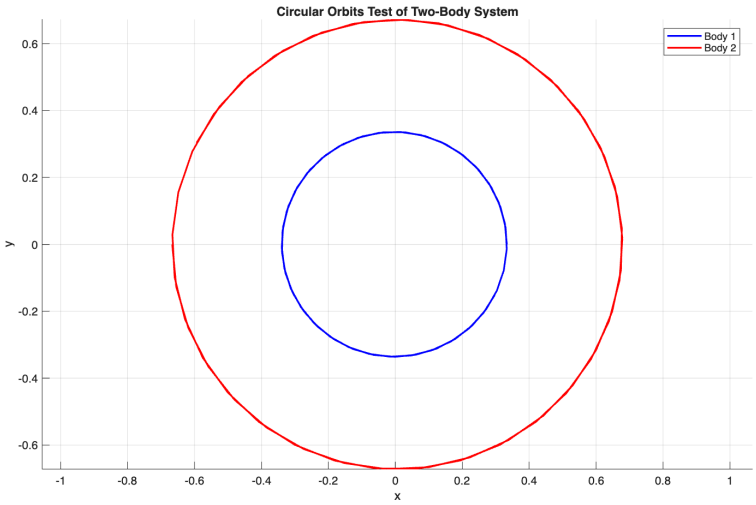


Figure 1: Trajectory of the two-body orbit simulation.

## 3.3 Convergence Testing

To empirically verify the numerical accuracy of our $N$-body solver, we perform a convergence test. This test confirms that the implementation is correct and achieves its theoretical order of accuracy. We use the stable two-body circular orbit from Section 3.2 as our test case and follow the same convergence testing method from Tutorial 3.

The test is conducted by running the simulation four times with identical physical parameters ($t_{max}$, $m_1$, $m_2$, $R$) but with progressively finer time steps. As defined in Section 3.1, we vary the time step by changing the integer-valued level parameter $\ell$. For this test, we used $\ell = 6, 7, 8$, and 9. This produces four distinct solutions for the particle trajectories. We focus on the x-position of the first particle, $x(t)$, generating four solution vectors: $x_6$, $x_7$, $x_8$, and $x_9$.

First, the trajectories from all four levels are plotted on the same axes, as shown in Figure 2. The solutions show general agreement, but there is a visible deviation that grows over time. As expected, the coarsest simulation ($\ell = 6$, red line) diverges the most from the finer, more accurate solutions.

To quantify this convergence, we must compare the solutions at identical time points. We "downsample" the higher-level solutions to match the discrete time grid $t_6$ of the coarsest level. This is done by selecting every 2nd point from $x_7$, every 4th from $x_8$, and every 8th

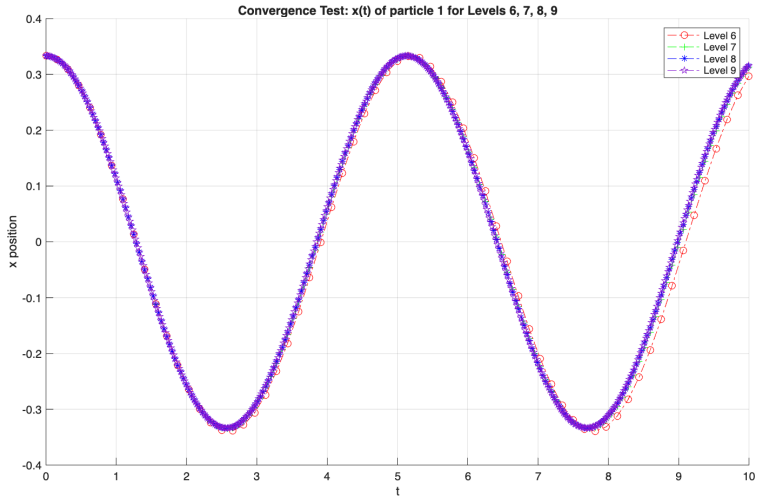


Figure 2: Trajectory x(t) of particle 1 for levels 6, 7, 8, and 9. The solutions converge as the time step decreases ($\ell$ increases).

from $x_9$. We can then compute the differences between successive solutions:

$$\Delta x_{6,7} = x_6 - x_{7,ds}$$
$$\Delta x_{7,8} = x_{7,ds} - x_{8,ds}$$
$$\Delta x_{8,9} = x_{8,ds} - x_{9,ds}$$

These differences, which serve as a direct measure of the solution error, are plotted in Figure 3. The plot clearly shows that the magnitude of the error decreases significantly as $\Delta t$ gets smaller.

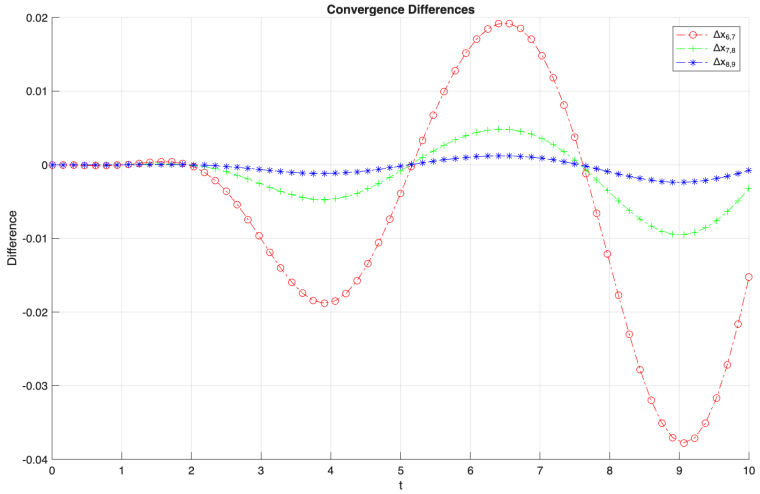


Figure 3: Unscaled differences between solutions at successive levels.

Since we are using a second-order finite difference algorithim its error $E$ should scale with the time step as $E \propto (\Delta t)^2$. Since each increase in level $\ell$ halves the time step ($\Delta t \to \Delta t/2$), the error should decrease by a factor of $2^2 = 4$. We can test this hypothesis directly. If the

method is second-order, we expect the following relationship:

$$\Delta x_{6,7} \approx 4 \times \Delta x_{7,8} \approx 16 \times \Delta x_{8,9}$$

To visualize this, we scale the finer differences and plot them against the coarsest difference. Figure 4 plots $\Delta x_{6,7}$, $4 \times \Delta x_{7,8}$, and $16 \times \Delta x_{8,9}$ on the same axes. The three curves are nearly coincident, which provides strong visual evidence that our solver is, as expected, second-order accurate.

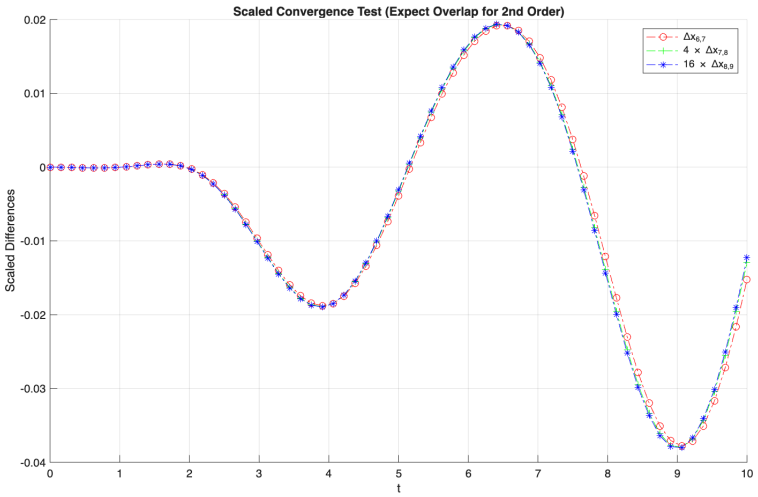


Figure 4: Scaled differences. The overlap of the three curves confirms the theoretical second-order convergence of the solver.

# 4 Implementation

## 4.1 N-Body Solver Function

The `nbody.m` function serves as the main driver for the simulation. It accepts the total simulation time `tmax`, the discretization `level`, and the initial conditions (masses `m`, positions `r0`, and velocities `v0`).

First, it establishes the finite difference time grid based on the `level` parameter, as defined in Section 3.1. As a practical implementation detail, rather than calculating `deltat` from the level, the time vector `t` is first generated using `linspace`. The constant time step `deltat` is then simply found by taking the difference `t(2) - t(1)`. A critical step is the initialization of the first two time steps. The position at $n = 1$ is simply the initial position `r0`. The position at $n = 2$ is then calculated using the second-order Taylor expansion from equation (7), which requires a single, initial call to the acceleration function `nbodyaccn`.

Once initialized, the function proceeds to the main time-stepping loop. This loop is highly efficient, as it iterates only over time (from $n = 2$ to $n_t - 1$) and not over the $N$ particles by calculating their next position all at once. At each time step, it computes the acceleration $\mathbf{a}_i^n$ for all particles and then applies the second-order central difference formula to find the next positions $\mathbf{r}^{n+1}$ using equation (6).

## 4.2 Acceleration Calculation Function

The `nbodyaccn.m` function is the most computationally intensive part of the simulation, as it must compute the gravitational interactions of thousands of stars for each time step. A key optimization, discussed in Section 2.2, is implemented here.

Our galaxy models consist of a core particles ($m > 0$) and a large number of massless star particles ($m = 0$). The function first identifies the indices and masses of only the core particles.

It then loops from $i = 1$ to $N$, calculating the acceleration for every particle (both cores and stars). However, the force calculation inside the loop only sums the contributions from the particles with mass by already knowing its indices. This optimization reduces the computational complexity from $\mathcal{O}(N^2)$ to a much more manageable $\mathcal{O}(N \times N_{core})$. In our project, $N_{core}$ is typically 2, while $N$ can be several thousand.

Within the loop, operations are vectorized using `vecnorm` and element-wise arithmetic to compute the forces from all cores onto particle $i$ simultaneously. A check is included to set the distance to infinity (`Inf`) for self-interaction (i.e., when particle $i$ is itself a core), preventing a division-by-zero.

## 4.3 Simulation Setup

Before simulating a full collision, I implemented the `toomre_galaxy.m` function to initialize the galaxies. This function was first tested by generating a single, isolated galaxy to ensure the model was stable and that the stars would correctly orbit the central core in stable circular paths, as expected. The initial parameters for this validation run are specified in Table 2. The simulation was run for several rotational periods, and the result is shown in Figure 5. The figure confirms the system's stability, with the blue dots (stars) maintaining their circular orbits around the central red dot (core). The `toomre_galaxy.m` function was implemented to generate galaxies rotating only about the $z$-axis, with options for either clockwise (CW) or counter-clockwise (CCW) spin.

Table 2: Initial conditions for the single galaxy validation test.

| **Parameter** | **Value** |
|---|---|
| Number of Stars ($N_{star}$) | 500 |
| Core Mass ($M_{core}$) | 1.0 |
| Initial Core Position | (0, 0, 0) |
| Initial Core Velocity | (0, 0, 0) |
| Min. Disk Radius ($R_{min}$) | 0.1 |
| Max. Disk Radius ($R_{max}$) | 0.5 |
| Spin Axis | CCW [z-axis] |

With the galaxy generation function validated, a top-level script constructs the initial state of the two interacting galaxies. Each galaxy is generated using `toomre_galaxy.m`. Once the two individual galaxy models are created, their respective particle lists are combined. The initial position matrices (`r01, r02`) and velocity matrices (`v01, v02`) are concatenated

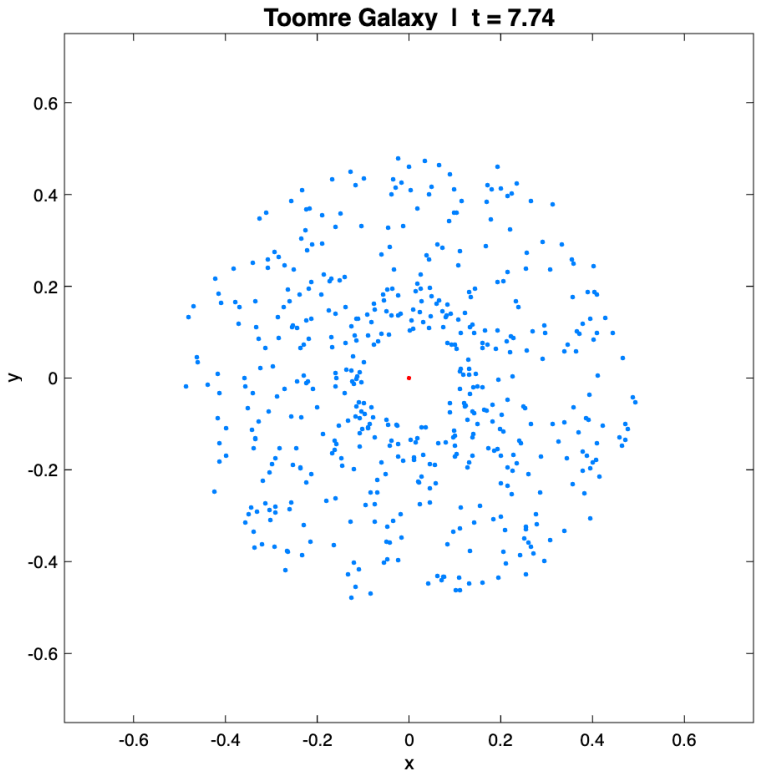


Figure 5: Trajectory of tracer particles in a stable, isolated disk galaxy, confirming the `toomre_galaxy.m` function.

to form the global `r0` and `v0` arrays. These arrays, along with the combined mass vector `m`, serve as the complete set of initial conditions for the `nbody.m` solver.

The main script then executes the `nbody` function for a specified total time `tmax`. After the simulation is complete, the solver returns the full $N \times 3 \times n_t$ position array `r` (where $N$ is the total number of particles) and the time vector `t`. These outputs are then passed to a separate `animation.m` function, which processes the data to create a dynamic visualization of the particle trajectories and the morphological evolution of the collision over time.

# 5 Results

Here are some interesting results from several simulations of interacting galaxies. By varying the initial conditions of the two-galaxy system we can use the $N$-body solver to model a wide range of interacting morphologies.

Before presenting the final visualizations, it's important to note a key flaw in the Toomre model encountered during testing. The model does not include gravitational softening. Consequently, any star particle that passes extremely close to a core experiences a near singular gravitational force, resulting in an unphysically massive acceleration causing it to fly away.

## 5.1 Simulation 1

Our first simulation models two galaxies with equal core masses but different initial velocities and disk sizes. The specific initial conditions for this run are detailed in Table 3.

Table 3: Initial conditions for Simulation 1.

| Parameter | Blue Galaxy | Green Galaxy |
|---|---|---|
| $M_{core}$ | 0.2 | 0.2 |
| $N_{star}$ | 5000 | 5000 |
| $\mathbf{r}_{core}$ | (2, 4, 0) | (-2, -4, 0) |
| $\mathbf{v}_{core}$ | (-0.1, 0.1, 0) | (0.2, 0, 0) |
| $R_{min}$ | 0.7 | 0.2 |
| $R_{max}$ | 3.5 | 4.1 |
| Disk Spin | CCW | CW |

The resulting interaction is shown in Figure 6 with each galaxy represented by a different colour. As the galaxies pass each other, their mutual gravitational pull strips stars from the outer disks, creating prominent, long tidal tails. The bridge of stars connecting the two galaxies is visible.

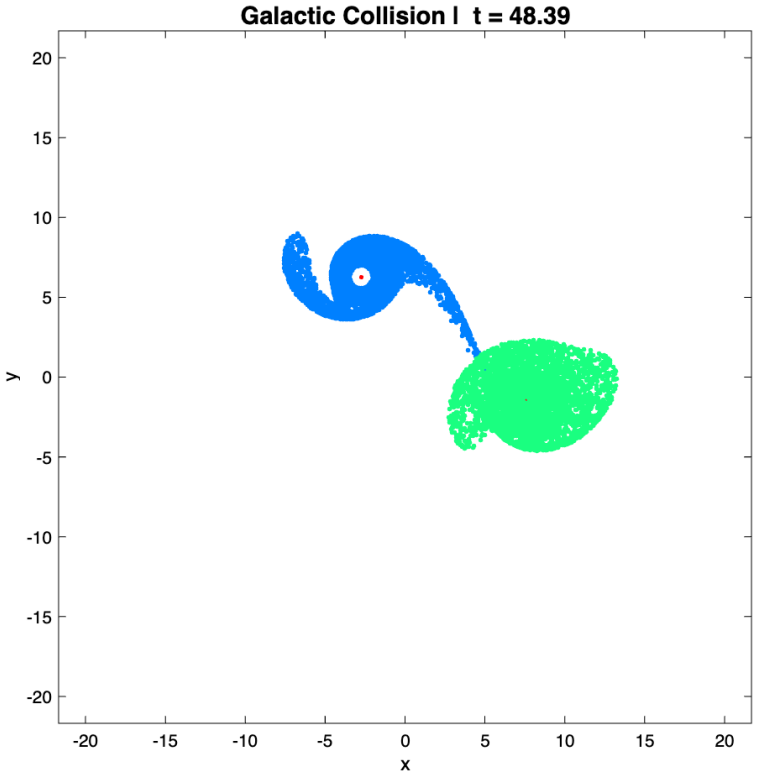


Figure 6: Snapshot of Simulation 1 at $t = 48.39$, showing the formation of long tidal tails and a stellar bridge.

## 5.2 Simulation 2

In this second simulation, I explored the effect of unequal core masses and different initial velocities with galaxies rotating in the same direction. This effect created a less complex interaction as the heavier galaxy remained mostly intact while the lighter galaxy was heavily distorted. The initial conditions for this run are summarized in Table 4.

Table 4: Initial conditions for Simulation 2.

| **Parameter** | **Blue Galaxy** | **Green Galaxy** |
|---|---|---|
| $M_{core}$ | 1.5 | 2.0 |
| $N_{star}$ | 5000 | 5000 |
| $\mathbf{r}_{core}$ | (4, 0, 0) | (-3, 0, 0) |
| $\mathbf{v}_{core}$ | (0, -0.4, 0) | (0, 0.3, 0) |
| $R_{min}$ | 1.7 | 1.1 |
| $R_{max}$ | 3.5 | 3.3 |
| Disk Spin | CCW | CCW |

The resulting interaction is shown in Figure 7. The heavier galaxy (green) remains relatively undisturbed, while the less massive galaxy (blue) is significantly distorted, forming a pronounced tidal tail. The interaction highlights how mass ratios influence the dynamics of galactic collisions.

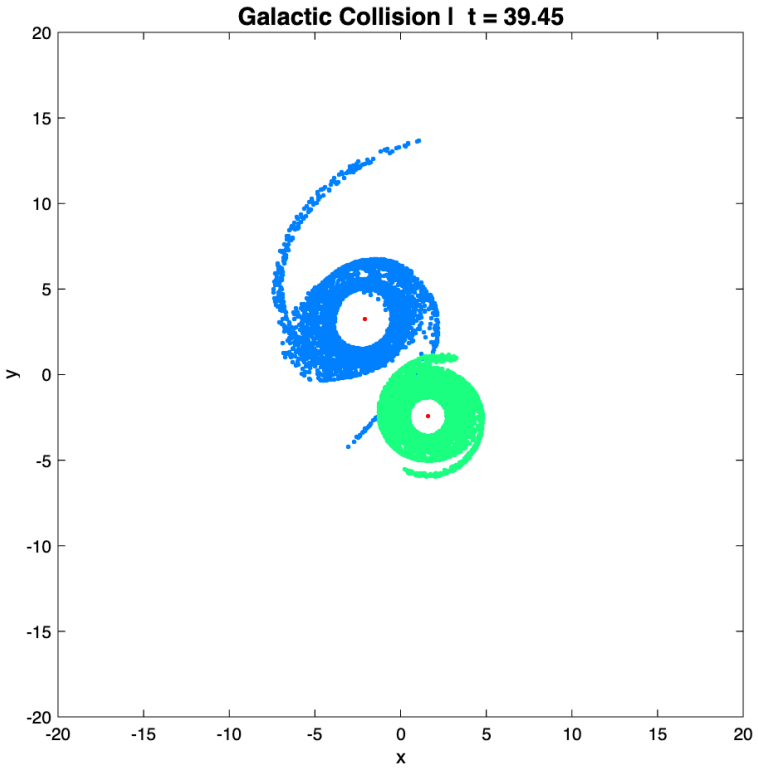


Figure 7: Snapshot of Simulation 2 at $t = 39.45$.

# 6 Conclusion

In this project, I successfully developed, tested, and implemented a gravitational $N$-body solver in MATLAB to simulate the dynamics of galactic collisions. The solver, based on a second-order central difference scheme, was validated through a two-body circular orbit test and a convergence test. The convergence analysis confirmed the solver's theoretical second-order accuracy, giving us strong confidence in its implementation. Using this solver, we modeled galaxies as a massive central core and a disk of massless stars according to Toomre's model. The simulations successfully reproduced a range of intersting morphologies. The results clearly demonstrated the high sensitivity of a collision's outcome to its initial parameters, such as the mass, velocity, and orientation of the interacting galaxies.

A primary challenge throughout the project was managing the computational cost associated with the N-body problem. The $\mathcal{O}(N^2)$ complexity of a naive acceleration calculation is computationally hard for a large number of particles. It's initial implementation would take minutes for 1000 stars to be calculated. This was mitigated by a key optimization in the `nbodyaccn.m` function, which treated only the two cores as massive sources of gravity. This reduced the complexity to a much more manageable $\mathcal{O}(N)$, allowing for simulations with thousands of stars to be run in a reasonable amount of time.

The model, while successful, operates under several key physical simplifications. The most significant of these is the use of massless stars, which neglects the self-gravity of the galactic disk and any collisions from occuring. Since all stars are attracted only to the core, under some initial conditions, stars might get too close to the core and fly away due to their near zero proximity. In its current form, however, this project has served as an effective demonstration of the fundamental physics driving galactic collisions and provided a robust framework for $N$-body simulation.

## 6.1 Use of AI

I used ChatGPT and Gemini as a guide to help me understand the methods and clarify concepts. All code was written by me. AI was primarily used for formatting tables in my LaTeX report, generating MATLAB plots, and helping with the LaTeX equations. ChatGPT was very helpful in explaining how the `bounce.m` function worked and how I could implement similar logic in my `animation.m` function, as well as make the simulation look like a dark sky. It also tought me some of the limitations of the Toomre model and different ways in mitigating stars from flying away such as gravitational softening which I did not implement.

# 7 Files

The following MATLAB files were created for this assignment:

- **FDA:** `nbody.m`, `nbodyaccn.m`
- **Testing:** `nbody_convergence_test.m`, `galaxy_dynamics.m`
- **Galaxy Setup:** `toomre_galaxy.m`

- **Visualization:** `animation.m`
- **Collision script:** `galaxy_collision.m`

# A MATLAB Code Listings

## A.1 N-Body Solver Function (`nbody.m`)

```
function [t, r] = nbody(tmax, level, m, r0, v0)
% nbody   Solves the n-body gravitational problem using an FDA.
%         similar implementation to tutorial 3.
%
%   Input arguments
%       tmax:       (real scalar) Final simulation time.
%       level:      (integer scalar) Discretization level.
%       m:          (real vector [1xN]) Masses of the n-particles.
%       r0:         (real Nx3 matrix) Initial positions
%       v0:         (real Nx3 matrix) Initial velocities
%
%   Output arguments
%       t:  (real vector) Time points (length nt = 2^level + 1).
%       r:  (real Nx3xnt array) Positions of the particles

    N = numel(m);
    % --- Finite Difference Grid ---
    nt = 2^level + 1; % number of time steps
    t = linspace(0.0, tmax, nt);
    deltat = t(2) - t(1);
    r = zeros(N, 3, nt);
    % --- Set initial conditions ---
    r(:,:,1) = r0;
    a0 = nbodyaccn(m, r0); % acceleration at t=0
    r(:,:,2) = r0 + v0 * deltat + 0.5 * a0 * deltat^2;

    % --- Time-stepping loop ---
    for n = 2:nt-1
        a = nbodyaccn(m, squeeze(r(:,:,n))); % squeeze to make Nx3
        r(:,:,n+1) = 2*r(:,:,n) - r(:,:,n-1) + deltat^2 * a;
    end
end
```

Listing 1: The MATLAB function for the n-body simulation.

## A.2 N-Body Acceleration Function (nbodyaccn.m)

```
function [a] = nbodyaccn(m, r)
%  nbodyaccn Computes accelerations for an N-body
%            gravitational system.
%
%  Input arguments
%       m: Vector of length N containing the particle masses
%       r: N x 3 array containing the particle positions
%
%  Output arguments
%       a: N x 3 array containing the computed accelerations
%

    N = numel(m);
    a = zeros(N,3);
    m = m(:);

    idx_core = find(m > 0);
    m_nonzero = m(idx_core);
    r_core = r(idx_core, :);

    for i = 1:N
        % Vector from i to all cores
        rij = r_core - r(i, :);
        dist3 = vecnorm(rij, 2, 2).^3;

        % If current particle is one of the massive ones, skip self
        self_idx = (idx_core == i);
        dist3(self_idx) = Inf;

        % Compute sum over massive bodies
        a(i, :) = sum(rij .* (m_nonzero ./ dist3), 1);
    end
end
```

Listing 2: The MATLAB function for the n-body acceleration.