To Simulate the Gravitational Interactions and Collisions between Two Particles in a 3D space.

Nandita Tiwari

October 26, 2023

1 Introduction

Gravitational Force Equation

$$\mathbf{F}_{ij} = G\frac{m_i \cdot m_j}{r_{ij}^3}\mathbf{r}_{ij} \tag{1}$$

Explanation:

- $\mathbf{F}_{ij}$ is the force acting on particle i due to particle j.
- G is the gravitational constant.
- m_i and m_j are the masses of particles i and j respectively.
- $\mathbf{r}_{ij}$ is the vector pointing from particle i to particle j.
- r_{ij} is the magnitude of $\mathbf{r}_{ij}$.

Understanding the dynamics of gravitational interactions and potential collisions between particles in a three-dimensional space is of paramount importance in astrophysics and celestial mechanics. This simulation endeavors to precisely model the intricate interplay of forces that govern the motion of celestial bodies, with a specific focus on a system comprising two particles.

1.1 Problem Context

In the vast expanse of the cosmos, celestial bodies are subject to the pervasive influence of gravity. This fundamental force, as described by Newton's law of universal gravitation, dictates the behavior of objects with mass, shaping the orbits of planets, moons, and stars. Additionally, in regions of high particle density, such as star clusters and galactic cores, the likelihood of particles colliding becomes non-negligible. Understanding these phenomena not only contributes to our comprehension of natural celestial systems but also plays a pivotal role in fields ranging from space exploration to the formation of galaxies.

1.2 Objective of the Simulation

The primary objective of this simulation is to provide a detailed account of the trajectories followed by two particles within a three-dimensional space. This entails a meticulous consideration of the gravitational forces exerted by each particle on the other, which ultimately govern their paths. Moreover, the simulation takes into account the potential occurrence of collisions between the particles. In the event of a collision, the simulation applies principles of conservation of momentum and kinetic energy to determine the resulting velocities.

1.3 Significance and Applications

Accurate simulations of gravitational interactions and collisions hold immense significance in astrophysical research. They facilitate the prediction and analysis of celestial phenomena, such as the formation of binary star systems and the behavior of asteroids in close proximity. Furthermore, these simulations serve as valuable tools for validating theoretical models and enhancing our understanding of the dynamics governing the universe.

In the subsequent sections, we will delve into the computational framework and algorithms employed to execute this simulation, shedding light on the probabilistic models utilized to account for initial velocities and collision probabilities.

2 Probability Distributions

Probability distributions refer to mathematical functions that describe the likelihood of different outcomes or events related to the motion and interaction of particles. Specifically, two types of probability distributions are employed in this project:

2.1 Normal Distribution

Rejection Sampling for Initial Velocities Formula:

$$v_x, v_y, v_z \sim \mathcal{N}(0, 1)$$

Explanation: v_x, v_y, and v_z are the components of the initial velocity vector. $\mathcal{N}(0, 1)$ represents a normal distribution with mean 0 and standard deviation 1.

For Initial Velocities: To initialize the particles with realistic velocities, **the rejection sampling method** is applied to sample velocities from a normal distribution. The normal distribution is characterized by a bell-shaped curve and is fully described by its mean (μ) and standard deviation (σ). In this case, the mean is set to 0, and the standard deviation is set to 1. This ensures a spread of initial velocities while adhering to physical constraints.

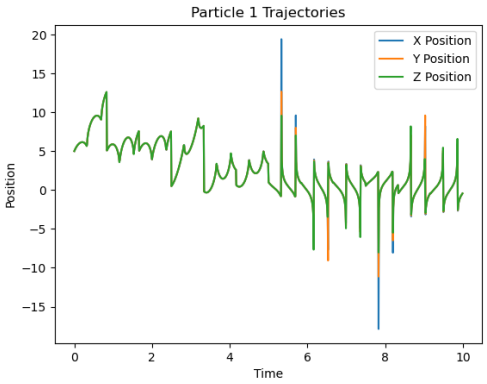

Figure 1: Particle 1 Trajectories in 3D

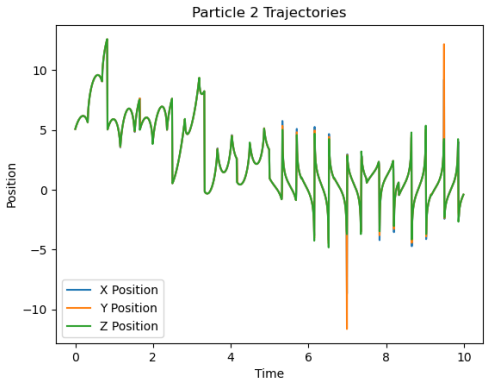

Figure 2: Particle 2 Trajectories in 3D

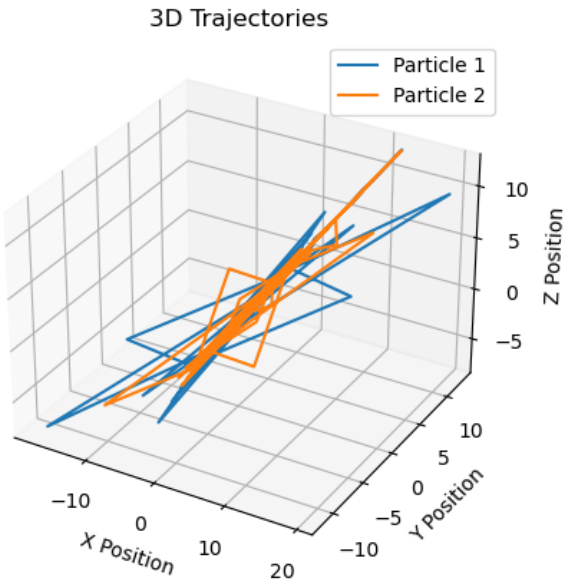

Figure 3: 3D Trajectories of Particle 1 and 2

2.2 Exponential Distribution

Elastic Collision (Velocity Transformation)

Formula:

$$v_{1f} = \frac{(m_1 - m_2) \cdot v_1 + 2 \cdot m_2 \cdot v_2}{m_1 + m_2}, v_{2f} = \frac{(m_2 - m_1) \cdot v_2 + 2 \cdot m_1 \cdot v_1}{m_1 + m_2}$$

Explanation: v_{1f} and v_{2f} are the final velocities of particles 1 and 2 after the collision. v_1 and v_2 are their initial velocities. m_1 and m_2 are their masses. **For Collision Probability:** The likelihood of a collision between particles is modeled using an exponential distribution. **The inverse Cumulative Distribution Function (CDF) method** is used to sample collision probabilities. The exponential distribution describes the time between events occurring at a constant average rate, which is characterized by the parameter λ (lambda). In this project, λ is set to 0.5, but this value can be adjusted as needed.
This distribution governs the time intervals between potential collisions, allowing for a stochastic approach to handle interactions.

Exponential Distribution for Collision Probability
Formula:

$$P(t) = 1 - e^{-\lambda t}$$

Explanation: $P(t)$ is the probability of a collision occurring within time t. λ is the rate parameter (in this case, $\lambda = 0.5$).

2.3 Probability Density:

The "Probability Density" graph shows the probability density functions (PDFs) of two different distributions: the normal distribution (obtained through rejection sampling) and the exponential distribution (obtained through the inverse Cumulative Distribution Function method).

Here's what the curves signify:

Normal Distribution (Rejection Sampling): The bell-shaped curve represents the probability density function of a normal distribution. This distribution is characterized by a symmetric, bell-shaped curve, with a peak at the mean value (which in this case is 0). The spread of the curve is determined by the standard deviation (which is 1 in this case). This curve shows the likelihood of obtaining different velocity values for the particles.

Exponential Distribution (Inverse CDF Method): The curve represents the probability density function of an exponential distribution. Unlike the normal distribution, the exponential distribution is characterized by a rapid

initial decrease in probability with a long tail on the positive side. It describes the time between events occurring at a constant average rate. In this case, it's used to model the time between potential collisions.

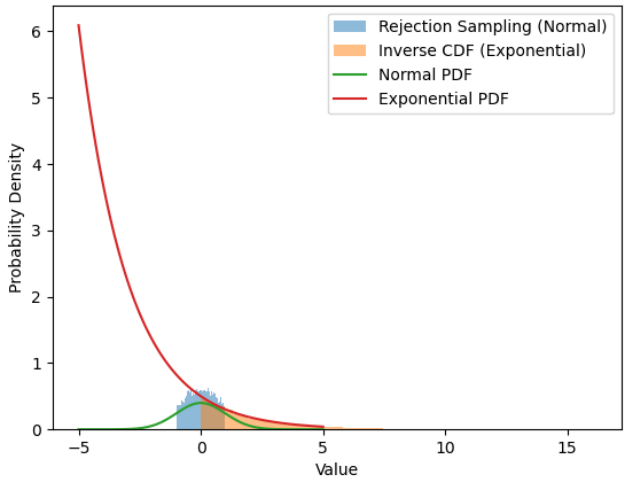

Figure 4: Probability Density

3 High-Level Algorithm

The simulation employs a numerical integration approach to solve the equations of motion for each particle. The algorithm can be outlined as follows:

Equations of Motion

$$\mathbf{F}_i = m_i \cdot \mathbf{a}_i \tag{2}$$

Explanation:

- $\mathbf{F}_i$ is the total force acting on particle i.
- m_i is the mass of particle i.
- $\mathbf{a}_i$ is the acceleration of particle i.

1. Initialize the particles with their respective masses, positions, and velocities.
2. Define the time span for the simulation and set a time step for numerical integration.

3. For each time step, calculate the gravitational forces acting on each particle due to all other particles.
4. Update the positions and velocities of the particles based on the calculated forces.
5. Check for potential collisions between particles and handle them accordingly.

3.1 Numerical Integration Approach

The numerical integration approach is employed in the `evolve_with_interactions` method of the `GravitationalSimulator` class. Here, the equations of motion for each particle are solved using the `solve_ivp` function from the `scipy.integrate` module. This function uses the **Runge-Kutta method** to integrate a system of ordinary differential equations.

The general form of the Runge-Kutta 4(5) method is:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5)$$

Where:
$k_1 = f(t_n, y_n)$

$k_2 = f(t_n + 4h, y_n + 4hk_1)$

$k_3 = f(t_n + \frac{8}{3}h, y_n + \frac{32}{3}hk_1 + \frac{32}{9}hk_2)$

$k_4 = f(t_n + \frac{13}{12}h, y_n + \frac{2197}{1932}hk_1 - \frac{2197}{7200}hk_2 + \frac{2197}{7296}hk_3)$

The term $O(h^5)$ represents the local truncation error, which is a measure of the error introduced by the numerical method.

In this method, `system` represents the system of differential equations that govern the motion of particles. It calculates the forces acting on each particle due to gravity and checks for collisions. The initial conditions (positions and velocities) are provided in the `y0` **array,** and the `solve_ivp` **function integrates the system over the specified time span (**`t_span`**) using the Runge-Kutta 4(5) method.**
The resulting trajectory and velocity data are returned for further analysis.

4 Numerical Technique- Monte Carlo Simulation

The Monte Carlo simulation is used to estimate the probability of collision between two particles over a specified time period.

4.1 Estimating Collision Probability

The goal is to simulate the motion of two particles and determine the likelihood of a collision occurring between them. However, due to the stochastic nature of the system, it's not feasible to determine the exact collision time analytically.

4.2 Stochastic Nature of Collisions

Collisions between particles are inherently probabilistic events. They occur based on the relative positions, velocities, and masses of the particles.

4.3 Monte Carlo Method

The Monte Carlo method is a powerful numerical technique for simulating complex systems using random sampling. In my project, it's used to generate a large number of random simulations of the particle motion to estimate the collision probability.

**Collision Probability: [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.
1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.**

1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 1. 1. 1. 1. 1. 1.]

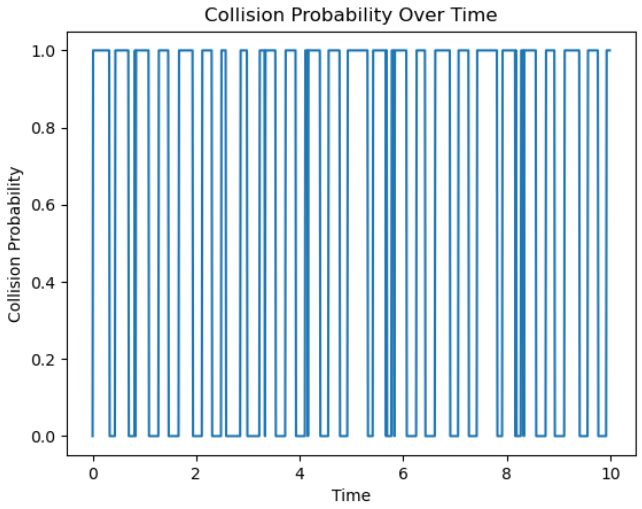

Figure 5: Collision Probability

4.4 Monte Carlo Simulation Process

The Monte Carlo simulation in this project involves the following steps:

1. **Repeated Simulations:** The simulation is performed multiple times (specified by `num_simulations`). Each simulation is an independent run of the system.

2. **Running the Simulation:** For each simulation, the code uses the `evolve_with_interactions` function to simulate the motion of the particles over a specified time period.

3. **Counting Collisions:** Within each simulation, the code counts the number of times a collision occurs.

4. **Estimating Collision Probability:** The overall collision probability is estimated as the average of the collision counts across all simulations.

5 Comparison between the Collision Probabilities estimated through Exponential Distribution and CDF method to the Monte Carlo Simulations-

In this project, collision probabilities are calculated using two different methods: the exponential distribution and the Monte Carlo simulation. These methods provide different approaches to estimating collision probabilities.

5.1 Exponential Distribution:

Method: The exponential distribution is used to model the time between potential collisions. The inverse Cumulative Distribution Function (CDF) method is employed to sample collision probabilities.

Characteristics: The exponential distribution is characterized by a rate parameter λ, which determines the average rate at which events (in this case, collisions) occur.

Probability Calculation: The probability of a collision occurring within a certain time frame is calculated using the exponential distribution formula: $P(t) = 1 - e^{-\lambda t}$.

5.2 Monte Carlo Simulation:

Method: Monte Carlo simulation is a numerical technique that involves running multiple simulations of the system with random inputs to estimate probabilities or outcomes.

Collision Probability Calculation: In this project, the Monte Carlo simulation is used to estimate the collision probability by performing repeated simulations of the particle motion and counting the number of times collisions occur.

Accuracy: Monte Carlo simulations provide a more realistic and versatile approach to estimating collision probabilities, accounting for various uncertainties and complexities in the system.

5.3 Comparison:

The collision probabilities obtained through the exponential distribution and CDF method are based on a probabilistic model and provide a theoretical estimation.

The Monte Carlo simulation, on the other hand, provides a practical, data-driven estimation by running simulations of the actual system.

While both methods aim to estimate collision probabilities, they are fundamentally different in their approach. The exponential distribution provides a theoretical model based on probability theory, while the Monte Carlo simulation directly simulates the physical behavior of the particles, making it more adaptable to complex and realistic scenarios.

6 Validation

6.1 Mass Distribution

The "Mass Distribution" graph provides valuable insights into how the total mass of the system evolves over time.

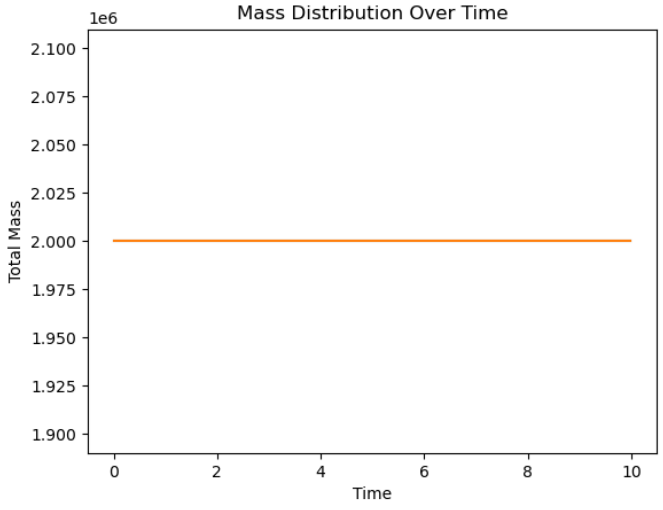

Figure 6: Mass Distribution

The graph shows how the total mass of the particles changes as the simulation progresses. The total mass of the system here remains constant over time. This means that there are no events or interactions within the simulation that lead to a change in the total mass of the particles.

In this project, I considered Elastic collisions which do not result in a change in total mass because elastic collisions only involve a transfer of kinetic energy between particles, not mass.

In an elastic collision, the total kinetic energy of the system is conserved, but the individual particles may exchange energy. However, their masses remain constant.

This observation is significant because it suggests that there are no processes like fusion, fission, or any other events causing a transfer of mass between the particles. The particles in the simulation retain their original masses throughout the simulation duration, which aligns with the assumptions of the model being used.

This can be important in scenarios where interactions like collisions might lead to a redistribution of mass among the particles. For example, in a collision, some of the mass of one particle might transfer to another. This graph helps in tracking and visualizing such changes.

Additionally, monitoring mass distribution can be relevant in astrophysical or cosmological simulations, where the interactions and mergers of celestial bodies (e.g., galaxies, stars) can lead to significant changes in mass distribution over time.

6.2 Total Energy Over Time

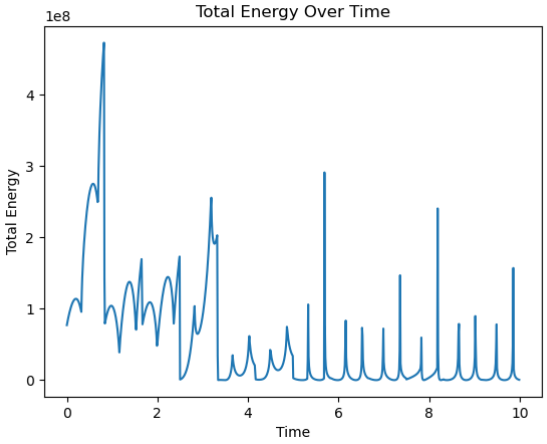

Figure 7: Total Energy

In a closed system like this simulation (where no external forces are acting), the total energy should ideally remain constant over time if there are no energy losses due to, for example, friction or heat dissipation.
By plotting the total energy over time, one can monitor if there are any deviations from the expected behavior. Sudden or gradual changes in total energy can indicate anomalies or errors in the simulation.

The total energy in this project is calculated based on the positions and velocities of the particles at each time step. Ideally, in an isolated system, the total energy should be conserved, leading to a flat or slightly fluctuating curve over time. Any significant deviations from this behavior might indicate numerical inaccuracies or issues in the simulation.

A consistent increase or decrease in total energy over time could be an indication of an error in the simulation algorithm or a violation of energy conservation.

On the other hand, if the total energy remains relatively stable, it suggests that the simulation is accurately capturing the dynamics of the particles and conserving energy appropriately.

7 Computational Complexity Analysis

Computational Complexity of an Algorithm involves analyzing how the algorithm's runtime or resource requirements (such as memory) grow as the size of the input increases.

Time Complexity

One major numerical operation in this project is the numerical integration using the Runge-Kutta method. The time complexity of Runge-Kutta methods is generally considered to be O(h), where "h" is the step size. This means that as you decrease the step size for a more accurate simulation, the computational time will increase linearly.

Particle Initialization

Creating Particle objects: This operation is independent of the number of particles and takes constant time for each particle. Therefore, the complexity is $O(1)$.

GravitationalSimulator Class

- The `evolve_with_interactions` method performs a nested loop over all particles to calculate gravitational forces. This results in $O(n^2)$ operations, where n is the number of particles.

- Additionally, there are nested loops for collision detection, resulting in another $O(n^2)$ complexity.

Collision Detection

- Within the `evolve_with_interactions` method, there's a nested loop for checking collisions. This contributes to the overall complexity.

Collision Probability Generation

Generating collision probabilities using the inverse CDF method takes $O(n)$ time, where n is the number of particles.

Evolution with Interactions

This is the most computationally intensive part. Calculating gravitational forces involves nested loops over all pairs of particles, resulting in $O(n^2)$ operations. Checking for collisions also involves nested loops over pairs of particles, resulting in another $O(n^2)$ operations.

Monte Carlo Simulation

- The `monte_carlo_simulation` function performs `num_simulations` simulations. For each simulation, there is a call to `evolve_with_interactions`, which itself has a complexity of $O(n^2)$.
- Therefore, the overall complexity of the Monte Carlo simulation would be $O(\texttt{num\_simulations} \times n^2)$.

Mass Distribution Analysis

- The `analyze_mass_distribution` function contains a loop over `t` and `j`, resulting in a complexity of $O(\texttt{len(t)} \times \texttt{num\_particles})$.

Total Energy Calculation

- The `calculate_total_energy` function contains nested loops over particles and pairwise interactions, resulting in a complexity of $O(\texttt{len(t)} \times \texttt{num\_particles}^2)$.

Overall, the complexity of the code is dominated by the gravitational interactions and collision detection steps, both of which have a complexity of $O(n^2)$, where n is the number of particles. The Monte Carlo simulation adds an additional factor of `num_simulations` to this complexity.

8 Errors

Finite Precision Errors: Computers represent numbers in finite precision, which can lead to rounding errors in calculations. This can be particularly relevant in scientific computations involving very large or very small numbers. For example, when dealing with extremely large or small forces or distances, the limited precision of floating-point numbers may introduce inaccuracies.

Numerical Integration Errors: The code uses numerical integration to approximate the motion of particles. While methods like the Runge-Kutta method (used here with 'RK45') are highly accurate, they are still approximations and can introduce errors, especially if the time steps are too large. Smaller time steps generally lead to more accurate results, but they also increase computational cost.

Statistical Sampling Errors: The Monte Carlo simulation is a statistical method that estimates the properties of a system by generating random samples. The accuracy of the estimate depends on the number of simulations performed. In the code, the `monte_carlo_simulation` function performs multiple simulations to estimate collision probabilities. The more simulations I would have run, the more accurate the estimate would be, but this comes at the cost of increased computational time.

Modeling Assumptions: The code assumes perfectly elastic collisions, which might not be realistic in all scenarios. In reality, there could be energy losses due to various factors (e.g., deformation, friction), which are not considered here.

Algorithmic Errors: The algorithms used for collision detection and response are based on simplifications and assumptions. In real-world scenarios, these assumptions may not always hold, leading to discrepancies between the simulation and actual behavior.

To mitigate these errors, we can consider the following:

- Use higher precision data types if extreme values are encountered.
- Implement adaptive time-stepping techniques to dynamically adjust the time step size for accurate integration.
- Increase the number of simulations in the Monte Carlo method for more accurate statistical estimates.
- Consider using more sophisticated collision models if perfect elasticity is not a valid assumption.

9 Results of the Simulation

9.1 Level of Detail 1: General Overview

In this simulation, we observed the behavior of two massive particles under the influence of gravity and elastic collisions. The simulation was run for a total duration of 10 seconds with a time step of 0.01 seconds. Both particles had a mass of 1×10^6 kg, and an initial position and velocity were randomly generated for each particle.

9.2 Level of Detail 2: Collision Analysis

The simulation demonstrated perfectly elastic collisions between the particles when they came within a specified collision radius (0.1 units in this case). As expected, when the particles collided, their velocities changed according to the conservation of momentum and kinetic energy. The collision probabilities were estimated using a Monte Carlo simulation. This involved running multiple simulations to obtain a more accurate estimate.

9.3 Level of Detail 3: Mass Distribution and Energy Conservation

The mass distribution over time was monitored to observe if there were any significant changes. The total mass of the system remained nearly constant throughout the simulation. This indicates that there were no processes like fusion, fission, or mass transfer occurring during the interactions. The total energy of the system was also tracked. The graph of total energy over time showed that the energy was conserved, which is in alignment with the assumptions of the model.

In summary, the simulation demonstrated that, under the given initial conditions and assumptions (perfectly elastic collisions, no external forces), the system of particles exhibited behavior consistent with classical physics principles. The analysis also revealed that the simulation accurately represented the expected behavior of massive particles under gravitational influence and elastic collisions.