Gravitational N-Body Simulation

Multi-Body Dynamics under Newtonian Gravity

Joseph Barton

February 2026

Abstract

This document outlines the direct summation method for simulating the gravitational evolution of N point masses. The system of $6N$ coupled first-order ODEs (positions and velocities in 3D) is integrated numerically using `scipy.integrate.odeint` or `solve_ivp`. The code computes pairwise inverse-square forces, conserves energy/momentum to machine precision (for small N), and visualizes orbits, trajectories, and stability. Source available in the repository. Link at end of paper.

1 Introduction

The gravitational N-body problem models the motion of celestial bodies (planets, stars, particles) interacting solely via Newton's law of universal gravitation. For $N \geq 3$, no general closed-form solution exists, and most configurations are chaotic. Direct summation (brute-force pairwise forces) is simple, exact in the Newtonian limit, and ideal for small-to-moderate N ($N \lesssim 1000$ without optimizations).

This project accompanies a Python implementation that solves the full N-body equations, includes initial condition setups (e.g., solar system-like, random clusters, figure-8 three-body), and produces animations or static plots of motion.

2 Theoretical Background

Each body i (mass m_i, position $\vec{r}_i$, velocity $\vec{v}_i$) feels gravitational acceleration from all others:

$$\vec{a}_i = \sum_{j \neq i} G \frac{m_j (\vec{r}_j - \vec{r}_i)}{|\vec{r}_j - \vec{r}_i|^3}, \tag{1}$$

where G is the gravitational constant (often set to 1 in astronomical units for simplicity).

The equations of motion are second-order:

$$\begin{aligned} \dot{\vec{r}}_i &= \vec{v}_i, \\ \dot{\vec{v}}_i &= \vec{a}_i(\{\vec{r}_k\}). \end{aligned} \tag{2}$$

This yields a state vector of length $6N$ (3 positions + 3 velocities per body).

To integrate numerically, flatten into a 1D array and define a derivative function that computes all pairwise accelerations via vectorized NumPy operations (avoiding slow loops for better performance).

Global quantities monitored include: - Total energy $E = T + V$, where kinetic $T = \sum_i \frac{1}{2} m_i v_i^2$ and potential $V = -\sum_{i<j} G\frac{m_i m_j}{r_{ij}}$, - Linear momentum $\vec{P} = \sum_i m_i \vec{v}_i$ (should be conserved), - Angular momentum $\vec{L} = \sum_i m_i \vec{r}_i \times \vec{v}_i$.

3 Numerical Solution

Integration uses `scipy.integrate.odeint` (or adaptive `solve_ivp` with 'RK45'/'DOP853' for better long-term stability). Softening parameter ϵ is sometimes added to the denominator $(r_{ij}^2 + \epsilon^2)^{3/2}$ to avoid singularities in close encounters.

Typical setups: - Units: $G = 1$, masses ~ 1, distances $\sim$ 1–10, - Time steps adaptive or fixed with monitoring of energy drift, - N = 2–100 bodies (e.g., Sun + planets, hierarchical triples, Plummer sphere clusters).

Key results: - Stable periodic orbits (e.g., restricted 3-body Lagrange points), - Chaotic scattering/ejections in three-body interactions, - Energy conservation within 10^{-10}–10^{-14} over many orbits for symplectic-like behavior in high-precision integrators, - Visualizations: 3D/2D trajectories with trails, center-of-mass frame animations, phase-space plots.

4 Conclusion

The N-body problem benchmarks numerical accuracy, force computation efficiency, and long-term stability in gravitational dynamics. This implementation highlights vectorization for speed, conservation checks, and flexible initial conditions—core skills for astrophysics simulations and scientific computing.

Repository: `https://github.com/jbarton727/Portfolio`