Simulating the Orbits of Planets in the Solar System

Rebecca Hamel, Sophia Rubens, Zach Sumners

Introduction

In this project, we constructed a two-dimensional N-body simulation of the solar system using high-order numerical techniques to analyze the stability of the orbits of planets. This paper outlines our implementation and the considerations that informed it, as well as a qualitative assessment of our system's stability for the simulation time we were able to achieve.

N-Body Techniques

N-body techniques are computational methods for simulating dynamic interactions of multiple bodies under the influence of forces such as gravity that are prohibitively computationally intensive to solve analytically. These techniques are useful in various fields of physics, such as astronomy, where they can help model systems including star clusters and planetary systems. By solving the equations of motion for each body in the system, necessarily including interactions, N-body simulations provide insights into the evolution and stability of complex systems over time. Various approaches for implementing N-body techniques exist, each balancing computational efficiency and accuracy. Here, we review features important for understanding our approach.

Simple ODE Solver

The simplest approach to N-body simulations involves directly solving the coupled equations of motion arising from interactions between objects or particles. Using Newton's second law, this can be expressed as the rate of change of velocity for a given body. Since it is simplest to deal with the constant, uniform unit vectors of Cartesian coordinates, we can rearrange the familiar two-dimensional Newtonian gravitational force expression into Cartesian component terms as follows:

$$\frac{d\mathbf{v}}{dt} = -\frac{GM}{r^2}\hat{r} = -\frac{GM}{r^3}\mathbf{r} = -\frac{GM}{r^3}(x\hat{x} + y\hat{y}). \tag{1}$$

From the velocity, the position of the object can be updated. Expanding this to an N-body system, the coupled equations for the positions and velocities of the modelled bodies can be expressed compactly using summation notation. In this scheme, x_i and v_i are the position and velocity vectors of the ith body, M_j is the mass of the jth body, and $r_{ij} = |\sqrt{x_i^2 + y_i^2} - \sqrt{x_j^2 + y_j^2}|$, meaning

$$\frac{d}{dt}[x_i,\ y_i,\ v_{x,i},\ v_{y,i}] = [v_{x,i},\ v_{y,i},\ -\sum_{i\neq j}^{N}\frac{GM_j x_i}{r_{ij}^3},\ -\sum_{i\neq j}^{N}\frac{GM_j y_j}{r_{ij}^3}]. \tag{2}$$

The vector to which the time derivative is applied on the left-hand side of the above equation is called vec in our code. This system of differential equations can be solved numerically by discretizing time into steps. For example, the position at a future time step can be approximated as

$$x_{t+\Delta t} \approx x_t + \Delta t\, v_{\text{avg}}, \tag{3}$$

where v_{avg} is the average velocity in the time step Δt. Naturally, the velocity is updated using the forces calculated at each moment in time. The choice of how to evaluate the average velocity and force during each time step significantly affects the accuracy of the simulation. Since the force is not constant within a step, poor approximations introduce errors. Next, we outline the technique we implemented to mitigate these inaccuracies.

Richardson Extrapolation

Fourth-order Runge-Kutta (RK4) and Leapfrog, two widely used lower-order n-body techniques, do not have the adaptive order and step size properties useful for the kind of low-N, long-term N-body system we consider here. Richardson extrapolation is a numerical technique useful for improving the accuracy of a given method by systematically eliminating leading-order errors. It is widely used in numerical integration of smooth functions to achieve high precision (1).

Given a quantity y with an error of order $O(h^k)$, where h is the step size and k is the precision order of the method (2). By performing the calculation with two different step sizes, h and $h/2$, the extrapolated value can be computed as

$$y(x+H) = \frac{2^k y(h/2) - y(h)}{2^k - 1}, \tag{4}$$

where $y(h)$ and $y(h/2)$ are the approximate solutions obtained with step sizes h and $h/2$, respectively. The process can be repeated iteratively, using results from progressively smaller step sizes to construct a sequence of increasingly accurate estimates. This approach is particularly effective for methods where the error decreases systematically with decreasing step size, such is the case for gravitational N-body simulations.

In the context of N-body simulations, Richardson extrapolation is often employed to refine results from lower-order integrators, such as the modified midpoint method we use here, improving accuracy without significantly increasing computational cost. In our code, we organized our Richardson extrapolates in a matrix, where the rows index higher orders and the columns index evaluations with more substeps (2).

Bulirsch-Stoer Method

The Bulirsch-Stoer Method is a high-accuracy numerical integration technique, particularly suited to solving systems of ordinary differential equations (ODE) in problems requiring high precision. This method combines the idea of Richardson extrapolation with the modified midpoint method to achieve high-order accuracy while maintaining computational efficiency.

The Bulirsch-Stoer method uses the modified midpoint method to compute a sequence of approximate solutions with increasingly smaller step sizes, followed by Richardson extrapolation to estimate the solution at zero step size.

1. Modified midpoint method: Starting from an initial condition $r(t_k)$, where here the function r represents the bodies' position, the midpoint method divides the time interval $[t_n, t_{k+1}]$ into substeps m, applying the midpoint rule iteratively to approximate $r(t_{k+1})$.

$$y_{k+1} = y_{k-1} + 2hf(t_k, y_k), \quad k = 1, 2, \ldots, m, \tag{5}$$

   for a function f representing the right-hand side of the coupled-first order recasting of our Newtonian gravity n-body problem.

2. Richardson extrapolation: Once solutions for several values of m are obtained, they are used to estimate the solution as $m \to \infty$. This is done by extrapolating the sequence of results to a zero-step size, eliminating leading-order truncation errors.

The method proceeds iteratively, refining the number of substeps m until the solution converges within a given tolerance. The Bulirsch-Stoer method is particularly efficient for problems where the solution is smooth, as it minimizes the number of force evaluations required to achieve a given accuracy.

For highly chaotic systems or systems with frequent close encounters, the method's efficiency can degrade because of the need for very small step sizes to maintain accuracy. Fortunately, neither of these properties appear in our simple 2D model of the solar system with only nine bodies, so the Bulirsch-Stoer method remains a reliable technique for the scope of this project.

Analysis

We initialized our simulations using present-day orbital parameters for solar system bodies including orbital periods, semi-major axes, longitudes of the ecliptic, planet masses, etc. (3; 4; 5). We used SI units throughout our simulations (6; 7). The following section outlines how we chose other values, such as the step size, the error tolerance, and presents our findings.

Step Size

To determine the ideal step size, we had to consider two factors that may shorten or lengthen the runtime: the number of steps and the number of recursions per step. On the one hand, decreasing the step size results in a larger number of steps to cover the same period of time, but the number of recursions required to reach the error tolerance will be smaller. However, increasing the step size results in fewer steps required, but the number of recursions will increase significantly. Ultimately, the optimal step size results from where these factors' effects are balanced. See Figure 1 below.

Going deeper into the substep-doubling recursion is more computationally intensive than taking shorter step sizes given a large number of steps (such is the case when simulating long-term stability), since the substep division follows a power law (2^k), which grows more quickly than simply accumulating additional steps.

To find the optimal step size, we ran simplified test runs with a range of logarithmically spaced step sizes and calculated the time it took to simulate 5e4 years of solar system evolution (see Figure 1). From this, we found the most solar system time could be traversed for a given program runtime was with a step size of $\Delta t \approx$2.2e6 seconds. Based on some print statements we used temporarily when conducting our step size experiments, the typical number of step size doublings for this optimal step is $k = 13$. The results we present below use this step size.

Substep halving limit

We set the error tolerance of the Richardson extrapolation to be double precision ($\sim$1e-12) to achieve the highest accuracy allowed by the machine. Given that the scales of the measurements were of the order of $\sim$1e13m for the positions, an error tolerance of $\pm$1m was chosen. We set the maximum number of doublings (k) to 15, just exceedingthe observed typical number of doublings for our optimal step size—allowing a bit of flexibility in the case of numerical drift, but preventing runaway step size doubling for unconverged steps. The maximum number of iterations for the Richardson extrapolation function was also set to match these values.

Performance Considerations

We initially hoped to parallelize our code to reduce the runtime. Based on cProfile-ing, our pairwise acceleration and position calculation functions constituted the most significant bottlenecks in evaluation. The computational burdens of our other routines were comparatively trivial, so we did not pursue performance enhancing strategies for these.

Returning to the heavier functions, since our pairwise acceleration calculator informs evaluation where parts of an array need to be evaluated sequentially, no extensive

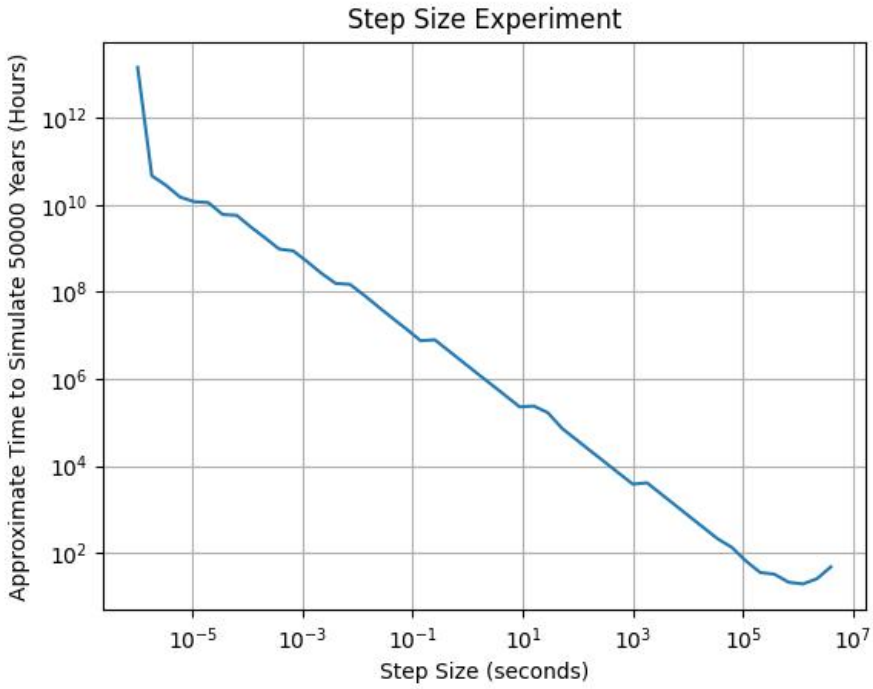

Figure 1: Logarithmically spaced step size test run done to determine what step size resulted in the shortest run-time to simulate 5e4 years of solar system evolution. From this, we found that the step size that resulted in the optimal run-time was a step size of $\Delta t \approx$1.1e6 seconds. This also resulted in a value of $k = 13$. Unfortunately, we were not able to extend this plot to significantly larger step sizes because the accumulated errors became too large for the integration to converge, even when we tried augmenting the maximum number of allowed step size doublings.

parallelization was possible. We tried using numba's nearly-just-in-time compiler wrapper for our pairwise position calculator, but noticed no significant improvement in performance. For our final simulation runs, we stuck with our "idiomatic numpy" version of this function, which we include here in our final program. Despite our abortive parallelization efforts, by optimizing our step size we achieved noticeable speedup in evaluation, on the order of 100% faster than our initial test runs with a naïvely chosen step size.

Evaluating Stability

To determine the stability of the orbits of planets in the solar system, we examined the bodies' long-term orbital behaviour qualitatively. We ran the simulation for a period of 22000 years. Although our graphics reveal jagged orbits for the innermost planets with the shortest orbital periods, this is merely because our optimal step size means that the position of each inner planet is not characterized at very many points per revolution around the sun. Once you take this into account and combine it with the fact that the planets have slightly elliptical orbits, we observe the expected behaviour: when plotting multiple orbits of each planet, the argument of periapsis appears to precess around the sun, leading to the observed "flower-like" curves. Restricting ourselves to the instructive plots associated with plotting the last few thousand points from our longest run, we can still glean intuition about the stability of the solar system from our simulation. See Figure 2 below. We see that the long-term behaviour of each planet exhibits the elliptical orbits we'd expect in the case of a stable system. In other words, if we'd been able to comfortably plot the whole time series of our longest simulation run, we'd expect to see a torus of sorts for each planet, with inner and outer radii equal to the planet's perihelion and aphelion.

Notably, however, it turned out to be *not* particularly instructive to plot the entire time series of our longest (270000-steps characterizing 22000 years) run because the barycenter of the solar system translates according to our integrator, leading to the impression of a "squashed slinky" in the plots where the ellipses are so close together that, for any easily legible line width, we see a pill-shaped blob of overlapping ellipses where additionally only the curve for Neptune (which we plotted last) is easily visible. Although the solar system barycenter does orbit the Milky Way, we did not include these dynamics in our simulation, so the translation of the barycenter we observe is merely an unphysical numerical artifact.

Conclusion

The goal of this project was to implement high-order numerical techniques to analyze the stability of the orbits of planets in the solar system. We implemented a Bulirsch-Stoer method to predict the positions of planets using an optimized step size. We were successful in simulating the orbits of the planets in the solar system over a period of over 50 thousand years. In our results, we observed the planets' precession and saw that none of the solar system bodies we tracked in our simulation diverged to infinity, converged to the barycenter, or otherwise deviated from the ellipses we expected. Therefore, we can conclude that for at least the timeframe we simulated, our solar system is stable. In the future, we could make our conclusions more robust by generalizing our model to three dimensions; including the next-most-important bodies, such as the Galilean moons of Jupiter; and examining the effects of perturbing bodies on the system's stability.

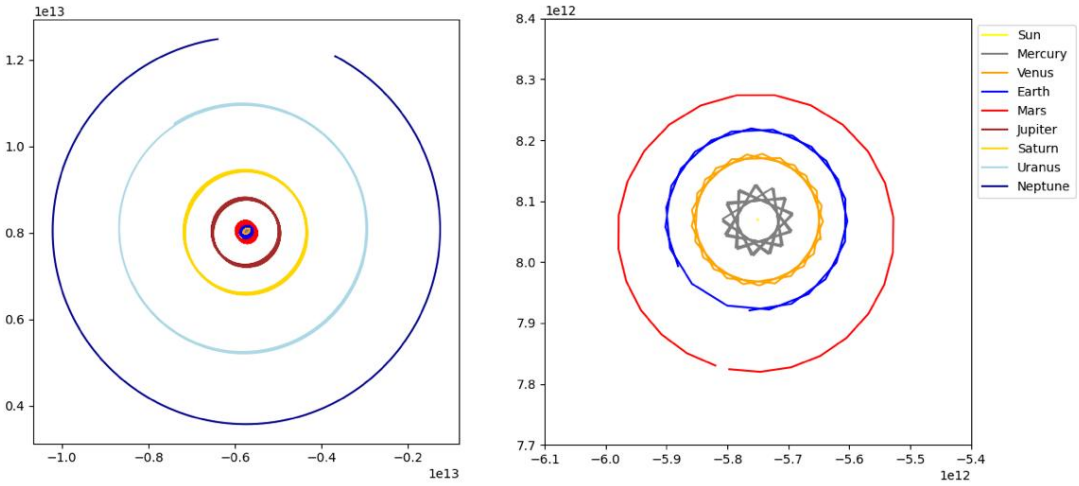

Figure 2: Snapshot of the tail end of our 22000-year simulation. Left: Orbital periods of the outer planets, showing the final 148 years. Right: Orbital period of the inner planets, showing the final 2 years. The jagged appearance of the innermost planets' contours is due to aliasing, not any physical instability. For more details, see the conclusion.

Division of workload

Collaborative development of integrator: Zach, Sophia (equal contribution)
High performance computing: Zach
Visualizations: Zach
Profiling and parallelization: Sophia, Zach (equal contribution)
Report: Rebecca (writing), Sophia (revising, technical comments)

Bibliography

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 3 ed., 2007.

[2] Wikipedia contributors, "Richardson extrapolation – Wikipedia, The Free Encyclopedia." `https://en.wikipedia.org/wiki/Richardson_extrapolation`, 2024. Accessed: 2024-12-04.

[3] NASA's Solar System Exploration, "Planetary Comparison: Orbital Periods." `https://solarsystem.nasa.gov/planet-compare/`, 2024. Accessed: 2024-12-04.

[4] Jet Propulsion Laboratory (JPL), "Approximate Positions of the Planets." `https://ssd.jpl.nasa.gov/planets/approx_pos.html`, 2024. Accessed: 2024-12-04.

[5] Jet Propulsion Laboratory (JPL), "Planetary Physical Parameters." `https://ssd.jpl.nasa.gov/planets/phys_par.html`, 2024. Accessed: 2024-12-04.

[6] Wikipedia contributors, "Astronomical unit – Wikipedia, The Free Encyclopedia." `https://en.wikipedia.org/wiki/Astronomical_unit`, 2024. Accessed: 2024-12-04.

[7] National Institute of Standards and Technology (NIST), "Value of the Gravitational Constant (G)." `https://physics.nist.gov/cgi-bin/cuu/Value?bg`, 2024. Accessed: 2024-12-04.