Developing N-body Simulations for Exoplanetary Systems

Jonas M. Powell[1]

Received ______________________; accepted ______________________

[1]Department of Astronomy and Van Vleck Observatory, Wesleyan University, Middletown, CT, 06459; `jmpowell@wesleyan.edu`

ABSTRACT

N-body simulations allow scientists from biophysicists to astronomers to test hypotheses about physical systems by creating a computational universe that is governed by the laws that they hope to better understand and observing their outcomes over time. These hypotheses can range from observing the binding characteristics of a DNA junction by modeling atoms and their electromagnetic relationships, to cosmologists developing models for large-scale structures in the Universe, treating galaxies as gravitational points. The simulations calculate the forces exerted by an ensemble of "particles" on each individual in the ensemble, and recalculating those forces in arbitrarily short time steps. However, these calculations can become computationally heavy, up to $O(N^2)$. In this paper I develop a barycentric N-body simulator, applying it to three exoplanetary systems by implementing the two most popular N-body algorithms (a particle-particle, "brute-force" method, and a tree algorithm known as the Barnes-Hut method), and compare their relative efficiency profiles. These models will also allow me to simulate radial velocity curves from each system (and arbitrarily more complex ones), seeing how the potentially chaotic dynamics involved in multi-body orbits manifests in observable data.

Subject headings: N-body simulation — exoplanets: modeling — planetary systems — planets and satellites — techniques: simulation

1. Introduction

Modeling our world has been a fundamental element of science since we began conceptualizing the laws at work in it. Modeling allows us to imagine a fabricated universe that runs only on the fuel (laws, matter, forces, and so on) that we feed it, and to compare the behavior of that little universe to observed behavior in nature. However, these simulations have historically been mental ones (or perhaps ones relying on very simple calculations), where, for example, the scientist might imagine the effects of an increased gravitational force and the effects that would have on nature. However, with the development of increasingly high-powered computers in the last half century, we are now able to translate that process of imagination into a process of actual numbers and calculations that would be impossible to perform by hand. This has allowed us to develop simulations which are complex (both temporally and spatially) far beyond what the human mind can imagine and which return precise and valuable data.

Perhaps the simplest and most common type of simulation is the N-body simulation. An N-body simulation simply takes in an initial set of particles and those particles' initial conditions (position and velocity) and calculates through discrete time-steps their motion due to user defined forces. The beauty of these simulations is their wide-ranging applicability; they work just as well to describe atomic motions in a molecule as they do to describe the galactic motion on Universe scales (or, in this case, the motions of bodies in a planetary system) by changing little more than the forces involved. For example, at the molecular level, electromagnetic forces dominate and gravity is negligible, whereas on cosmic scales those electromagnetic forces have no influence and instead it is gravity that dictates the bodies' motions.

N-body sims are, in essence, tools to solve the N-body problem: ”Given the quasi-steady orbital properties (instantaneous position, velocity and time) of a group of celestial bodies, predict their interactive forces; and consequently, predict their true orbital motions for all future times” (Wikipedia). They are the natural continuations of the two-body problem (understanding how to bodies, gravitationally attracted to one another, will move), but thanks to exponentially increasing computational power, we have been able to find approximate solutions for systems far larger than just the two-body one that has been on the minds of astronomers for so long.

My code was not, of course, the first time an N-body simulation has been written and implemented. There have been countless other implementations of this simulation technique, each geared for some particular task. One of the earliest discussions of these methods is in Harrison et al. (1969) (1), where they introduce many of the challenges that still confront these simulations today, including management of computational cost, appropriate conservation of energy, appropriate time-step length, and numerical integrations of the bodies' equations of motion. Their simulations topped out at N=500 bodies, an impressive number given the state of the computers of almost half a century ago. These simulations soon became valuable for actual science problem solving, as in Aarseth (1979) (2) which performed one of the first attempts at simulating galaxy clustering, using N=1000 bodies. By 1986, the need for increased efficiency for high N was clear and Barnes & Hut (1986) (3) presented the first application of graph theory to subdivide space and to dramatically increase efficiency at a very low cost to accuracy. Since then, N-body simulations have spiraled off into all sorts of different realms, ranging from molecular dynamics to cosmology (as in (4) and (5), respectively). Perhaps the most familiar to the astronomy community, and particularly to those interested in exoplanetary development, is REBOUND (6). Released in 2011 as a package in both C and Python, it was originally designed to model collisional dynamics (like planetary rings). Thanks to its durability, ease-of-use, and widely applicability to many different physical situations, it is now a standard tool for the field. It includes three symplectic integrators (a realm that I did not venture into at all in this project), the ability to be parallelized (a significant advantage for these simulations, since every calculation in each step is made independent of each other calculation, meaning that they are ideal candidates for parallelizing), and an extremely modular code structure, allowing users to easily curate the different modules to suit their needs, or, if need be, to implement their own and add it into the existing software.

Implementing an N-body simulation is, conceptually, a fairly simple task: one must simply calculate all the forces on each body in the system for each of some number of time-steps and, according to those forces, move each body as is appropriate. Obviously, however, this quickly results in a large number of computations, and so many variations have been made to simplify the calculations while preserving the simulation's accuracy. Exploring and implementing some of these approaches is the task of the Section 2. In Section 3, I will explore the results obtained with these different simulation approaches and see how each handles a set of canonical planetary systems. In Section 4, I will discuss these results, as well as some of particular issues I had to sort out in this project. I conclude the paper with Section 5, summarizing my findings.

2. Methods

In this project, I have examined several N-body simulation methods, coding two of them. All implementation was performed in Python and written from scratch (i.e., no pre-existing N-body packages were used). Assistance from Ryan helped shape some of the more structural pieces of the code, primarily in the brute-force method's implementation, as I learned about Python classes on the fly through this project. All simulations were executed in two-dimensions, but could be easily scaled to three dimensions in the future. Below are musings on each method.

2.1. Particle-Particle Method

The simplest, most intuitive approach is direct integration, most officially known as the particle-particle method, referring to its calculations of each particle on each other particle, but which is more often simply referred to as the brute force method. This approach involves simply calculating the force exerted on each body in the system by every other body in the system. This force, as far as we are concerned here, is nothing more than Newton's Law of Universal Gravitation:

$$F_{gravity} = m_{total}a = G * M * \sum_{i=1}^{N} \frac{m_i}{r_{i,M}^2}\widehat{r}, \tag{1}$$

where G is the gravitational constant, M is the mass of the body we are observing, m_i is the mass of the i^{th} other body, and $r_{i,M}$ is the distance between those two masses. To use this to predict a body's motion, we must translate it into an acceleration, turn that into a velocity, and then calculate the change in position according to that velocity and our selected time-step (Δt):

$$a_{gravity} = \frac{F}{m_{total}} = G * \sum_{i=1}^{N} \frac{M_i}{r_{i,m}^2}\widehat{r}, \tag{2}$$

$$v = a * \Delta t, \tag{3}$$

$$r_{i+1} = r_i + v\Delta t + \frac{1}{2} * a(\Delta t)^2. \tag{4}$$

We are then able to sum these forces for each body, take a step forward in time, recalculate them, and so on. However, it should be clear that this is a very heavy computation, since for each body in the system, every other body's force contributions must be considered, yielding a cost evaluation in Big-Oh notation of $O(N^2)$. Since longer and more complex simulations are always desirable, it would be nice to find a method that is more efficient than this. However, it is worth noting that the brute force method does have some notable upsides. It is the most accurate method (since other algorithms trade computational efficiency for more approximate simulations), and is relatively simple and intuitive to implement.

2.2. Barnes-Hut Algorithm

The Barnes-Hut method offers a decrease from $O(N^2)$ to $O(N \log N)$ time by approximating far-away groups of bodies as single body located at the group's center of mass, where "far away" is defined by an arbitrary cutoff of the ratio $(\frac{s}{d})$ between the area spanned by the region to be approximated (s) and the distance of the center of mass for that region from the target body (d).

In order to perform this calculation, the simulation's space is represented as a $quad - tree$, using recursive quaternary splitting of regions of space into quadrants until each quadrant has no more than one body in it (Figure 1). Each quadrant is then represented as a node in a graph, below the node representing the larger square that it is a quadrant of and above four sub-nodes of its own. This allows us to take advantage of the efficiency of computational trees to group bodies.

With the bodies divided into their nodes in the quad-tree, we can use the following rules to calculate the forces on a body b, starting at the quad-tree's root node:

1. If the current node is an external node (i.e., it has no sub-nodes), calculate its gravitational force on b. In this case, we know that there must either be zero or one bodies contained in the node, since, if it has no sub-nodes, a body must have been placed either there or in one of its sibling quadrants. Therefore, calculating the node's

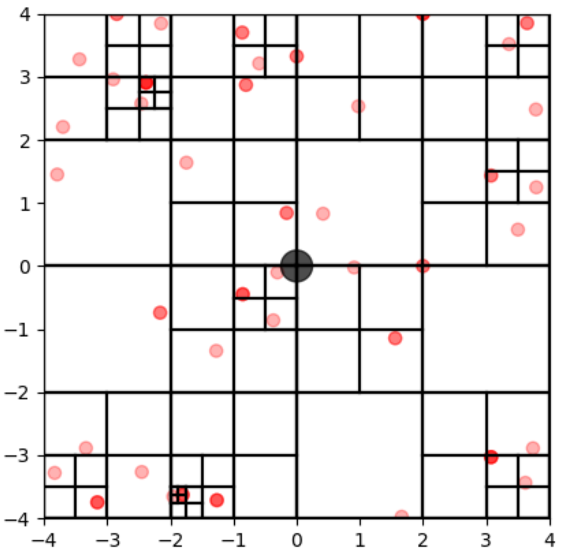

Fig. 1.— *An example of a successfully gridded set of 40 bodies. Each circle is a body, while the black lines represent quadrant boundaries in the graph. Note that no cell has more than one body in it.*

gravitational impact is simply a matter of either calculating the force due to the body contained or due to an empty node ($m_{total} = 0 \rightarrow F = 0$).

2. If the current node is an internal node, calculate $\frac{s}{d}$, where s is the width spanned by the node (its spatial coordinates) and d is the distance from b to the current node's center of mass. Compare $\frac{s}{d}$ to θ, where θ is an arbitrarily chosen cutoff value for this ratio. More on θ below. By convention, I take $\theta = 0.5$.

    (a) If $\frac{s}{d} < \theta$, then the node's mass is bound sufficiently tightly and thus can be approximated as a point source, or a pseudo-body. Calculate the gravitational interaction between b and this pseudo-body, of mass $m_{node,total}$ located at the node's center of mass.

(b) If $\frac{s}{d} > \theta$, then the node's mass is too spread out and cannot be treated as a unified point. Therefore, we must go look individually at each of its four sub-nodes (or spatial quadrants) and repeat (2).

Here we use θ as a tuneable parameter, using the small angle approximation to take $\tan\theta = \frac{O}{A} = \frac{s}{d} \approx \theta$. Thus, θ is measure of the angular spread of the masses in the node. Note that as $\theta \to 0$, the acceptance of internal nodes goes to 0 and the algorithm becomes the brute-force method again, only accepting bodies in external nodes by Eqn. 1.

We come to our evaluation of the algorithm's cost of $O(N \log N)$ by recognizing that the algorithm loops over N bodies and, for each body, makes between two and N calculations, recursively searching through the quad-tree at a cost of $O(\log N)$. Multiplying the two yields a cost of $O(N \log N)$.

2.3. Other Methods

Of course, the two methods that I implemented for this project (particle-particle and Barnes-Hut) are only two (historically, the first two) of many approaches that have been taken to developing these simulations. Some of the alternatives are considered below.

The foundation of the N-body simulation is built on the transition from treating time as a continuous variable to making it a discrete one, allowing us to compute the motions and forces of particles at specific moments and let a series of those discrete moments approximate continuous time. However, the Particle-Mesh (PM) method takes the discretization one step further, transforming space to a discrete variable as well, described by a grid (or "mesh"). One may then give a potential across the mesh, and find the force at each mesh point by taking the gradient of that potential. If a body does not lie precisely on a mesh point, one may interpolate the potential at its position using surrounding points.

The cost of the particle-mesh method can be as low as $O(N)$ (since it only requires a single force calculation - taking the gradient of the potential at the each body's position - for each body), but as the mesh's gridding becomes infinitely small, it slides back to Barnes-Hut's $O(N \log N)$. Consequently, this approach best suited to large-scale simulations such as cosmological runs, where bodies rarely are near enough to require the extremely fine gridding that makes this method costly. One of the early uses of this approach was in Melott et al. (1983) (7), which used such a simulation to compare the large-scale structure development in the Universe under dominance of axions, gravitinos, and photinos.

This weakness of PM can be mitigated by adopting the Particle-Particle/Particle-Mesh method. This approach takes the strengths of the particle-particle method (extremely precise calculations, but computationally heavy) and combines them with the strengths of the particle-mesh method (computationally efficient, but inaccurate at short distances). To do this, particle interactions are simply characterized as either long- or short-range. Short range interactions are then handled by the particle-particle calculations, and long-range ones are the business of particle-mesh approach. While these other methods went without implementation during this project's official duration, they offer an interesting path for future work.

3. Results

In this project, I was able to successfully implement each method, creating dynamic visualizations of the resulting orbits, as well as a couple relevant RV plots for the central star for three arbitrarily chosen sets of initial conditions. However, some areas had problems arise that have gone unsolved in the official time allotment of this project. These were, most notably, a bug that significantly over-strengthened the gravitational force in my brute-force implementation and a poor definition of initial velocities for asteroid belts (to allow them to all move in a stable orbit all in the same direction). These mishaps are discussed further in the relevant sections below. Each of these problems affected the final system configurations, but did not affect cost evaluations.

Since writing these simulations is conceptually a fairly simple task, much of the project's difficulty came in code and data management. However, executing the graph in the Barnes-Hut algorithm presented its own unique and interesting challenge. Creating plots (particularly dynamic plots, in the form of .gif files) that showed the progression of the bodies, but more importantly the progression of the tree, became crucially important in order to monitor the system over time.

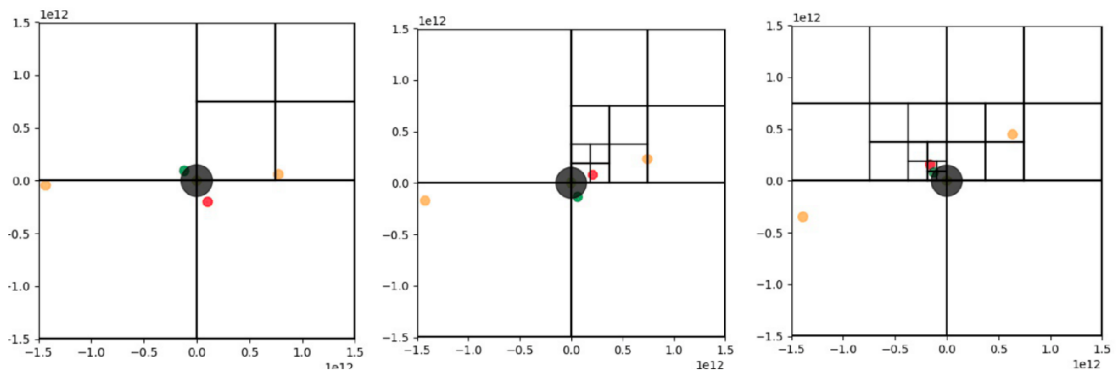

Fig. 2.— *An example of a successfully gridded set of five bodies. Each frame represents a time-step, demonstrating the graph's evolution in time.*

I chose to examine three systems that I felt were representative of a broad range of possible scenarios: first, a simple one planet/one star system, in part to use as a test subject and in part to just study the orbital motions produced in a non-chaotic environment; second, our solar system's planets and a solar-mass star, since it is a familiar subject and one whose relevance is easily understood by people who are not in the field. Finally, I simulated a system with 500 small ($0.05 M_{Earth}$) bodies in a ring from 4.9 - 5.1 AU, as well as our solar system's planets, orbiting a solar-mass star.

3.1. One Planet, One Star

This run was mostly performed to demonstrate systemic stability in each simulation. Consequently, it was where the troubles with the brute-force gravitational attraction showed itself first, as is apparent in Figure 3 (left). One interesting element of this bizarre gravitational twist is that both methods use the same actual force/acceleration/velocity/position updater, meaning that if the problem were arising in the force calculation itself, it should be affecting both methods, which it is not. However, this implementation was working quite well up until quite recently, and so my suspicion is that it is simply a small error somewhere rather than a fundamental issue.

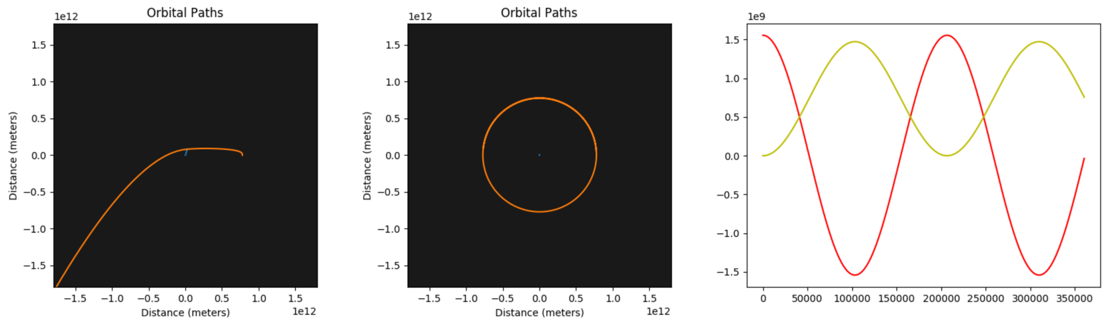

Fig. 3.— *Data resulting from a 17-Earth-year run of a Jupiter-mass planet at a Jovian semi-major axis about a solar-mass star, with half-hour time-steps. LEFT: We see that there is a significant problem with the brute-force implementation, as the bodies are immediately tugged towards each other, leading to the ejection of the planet. CENTER: We see that the Barnes-Hut implementation produces a stable orbit. RIGHT: A radial-velocity plot made using data from the Barnes-Hut run of the central star (yellow), along with a scaled RV plot of the planet (red) to show that they are moving in response to one another.*

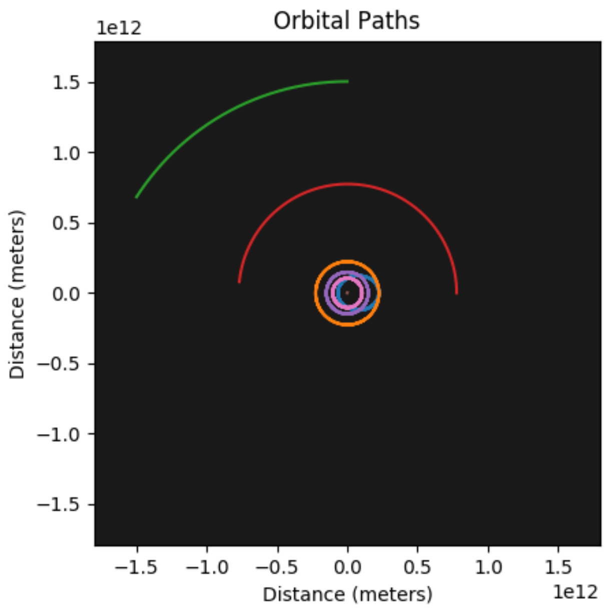

Fig. 4.— *Our solar system, here modeled using the Barnes-Hut method.*

3.2. Our Solar System

The next system I focused on was a model solar system. This struck me as an appropriate option since it gave me easy, known initial conditions (planet masses, distances, and velocities) and so it would just be a question of recreating known trajectories. Although the same troubles with the brute-force method arose here as before, this was more or less a successful endeavor in the case of the Barnes-Hut method (Figure 4); with the exception of Mercury (in blue, showing an extremely elliptical orbit), all the orbits seem pretty good. Furthermore, the high eccentricity of Mercury is not an enormous problem, since it is just an artifact of an imprecisely-given initial velocity (which I found by simply dividing the length of the planet's orbital path by the length of its year, which yields approximate solutions) and the fact that, as the most interior planet, it feels the effects of the discontinuity of time in the simulation the strongest (since the angle it sweeps out in a step is the largest and consequently it suffers the largest inaccuracies during a step relative to planets further out). This could be solved by implementing a variable time-step system, or just shrinking the timestep for all planets.

3.3. Solar System, Plus 500 Small Bodies

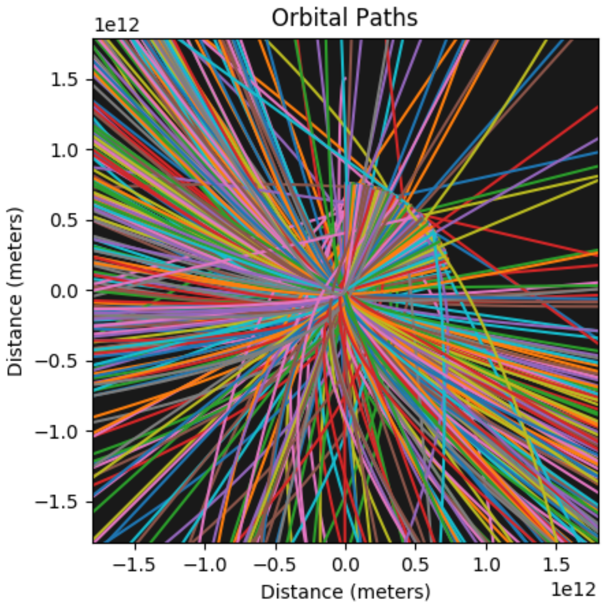

Fig. 5.— *The brute-force simulation, with the gravitational bug described above, results in every body collapsing to the middle and being violently ejected.*

I wanted to simulate a system that mimicked our solar system but with the addition of a rocky belt between Mars and Jupiter (an arbitrarily chosen distance), but this one ended up causing me a lot of trouble (although it resulted in some beautiful plots). In the case of the brute-force method (Figure 5), the problem clearly lies in my bad gravity calculations; each body is just being sucked in to the center and then ejected out of the frame. It is also worth noting that the strange quarter-circle in the upper right is a consequence of my poorly-given initial velocities for the asteroids, as I tried to herd them towards approximately stable counterclockwise orbits. The formula I derived clearly did not work as I had hoped, and so left some strange remnants like this one. Additionally, I only recently noticed that I had used both astronomical units and meters in my formula to calculate what a stable orbital velocity would be, leading to initial velocities of essentially zero for every asteroid, magnifying the strengthened gravity problem by allowing the bodies to immediately fall inward without any resistance.

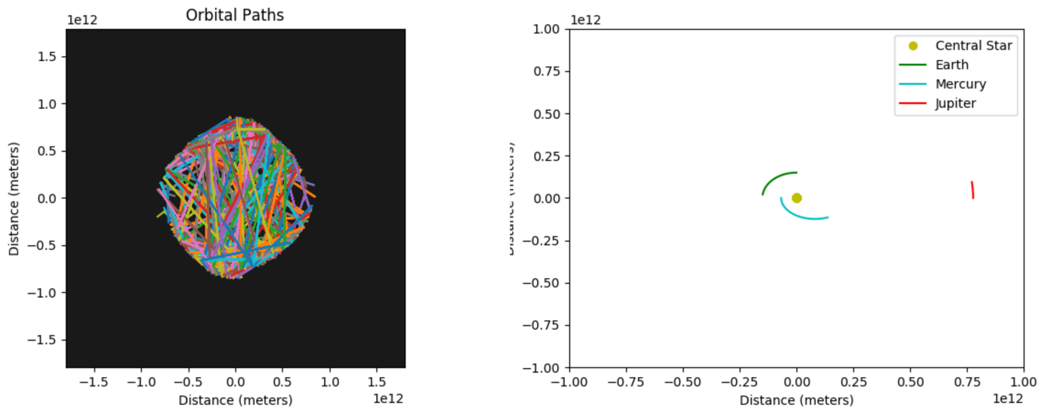

Fig. 6.— *The Barnes-Hut method clearly had some problems, but they were not as bad as it might seem. LEFT: The trajectories of 500 planetesimals plus our solar system's planets. The linear path of each asteroid is cause for concern. RIGHT: The same plot as on the left, but with all the asteroids (and a couple planets) removed to show that the planets are still essentially on their expected orbital path. This shows that the problem was just one of poorly set initial velocities for the asteroids.*

This lack of unit conversion also affected my Barnes-Hut simulation (Figure 6), although in that case, I also somehow managed to flip the equation, meaning that velocities that shrunk towards zero in the brute-force run exploded towards very large numbers in this one. With such high initial velocities, gravity did not really have any effect on each body's path, and so they just followed a straight line from whatever direction they started in. That being said, I'm not sure why none of them escaped the "rubber band ball" shape;

this will be a path for further inquiry in the future. We can confirm that this was the case by observing that the planets (a sample of which are shown in Figure 6, right), which have correct, manually-entered initial velocities, are still moving as expected (recall from Section 3.2 that we expect Mercury to have an elliptical orbit.)

While these troubles are significant and make the actual data collected from the runs essentially useless, they all are simply the effects of numerical carelessness on my part and do not seem to imply any lack of robustness in the algorithm.

3.4. Cost Analysis

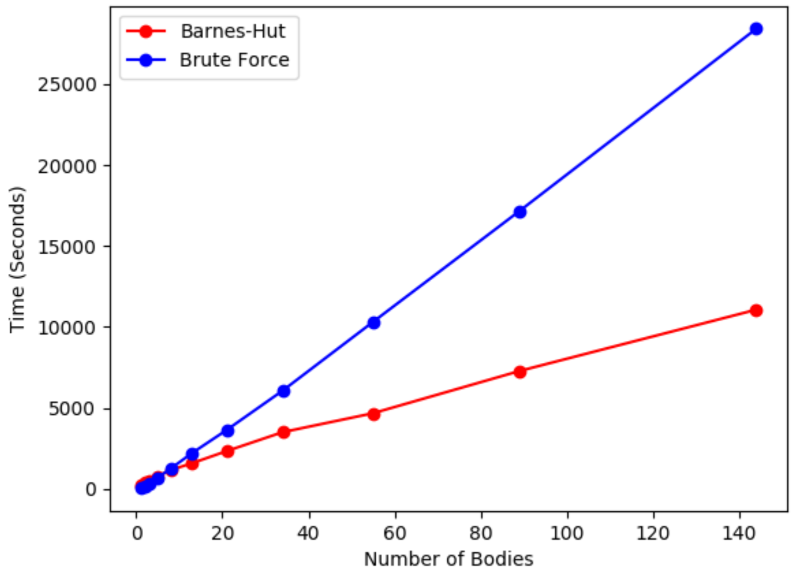

Fig. 7.— *We can see clearly different trends in the two lines, with the Barnes-Hut cost increasing less quickly than the brute-force method.*

One of the main goals of writing two implementations of the N-body simulation, besides simply developing my understanding of each, was to compare the time-efficiency of each algorithm.

Although my cost testing did not yield exactly the results I expected, the results I ended up with are at least steps in the right direction and ones that can be fairly easily understood. The most important feature here is that we actually see that the Barnes-Hut implementation is, in fact, saving us time. This is a good thing. In fact, the Barnes-Hut implementation seems to be behaving quite well; by plotting over it a $O(N \log N)$ curve (the ideal that this algorithm is striving towards), scaled by an (eyeballed) factor of 30 to account for the constant-cost operations that occur at each time-step, we find that the two are a very close match (Figure 8).

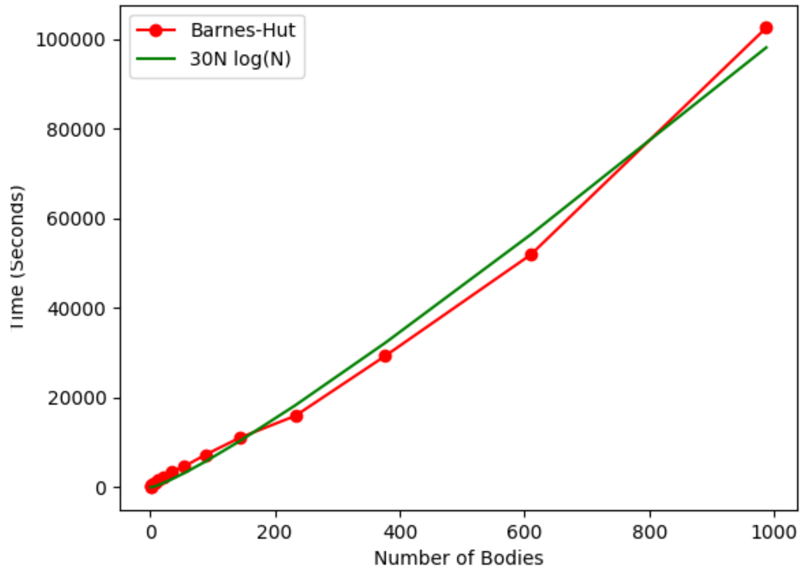

Fig. 8.— *A model $N \log N$ line (green) alongside the actual Barnes-Hut time curve produced by my simulation (red). We see that they are very close.*

Intriguingly, it is actually the brute-force method that is not really behaving as it ought, being best fit by a constant slope with an extremely high coefficient (around 200, which is far more than constant-cost operations could account for), yielding a much closer curve than any sort of exponential function (Figure 9). It is worth noticing here that the numbers are still quite low (the simulation hasn't yet finished running, and so max N is 144), so the curve could become more characteristically exponential as N increases, but for the moment, this is an unsolved quirk.

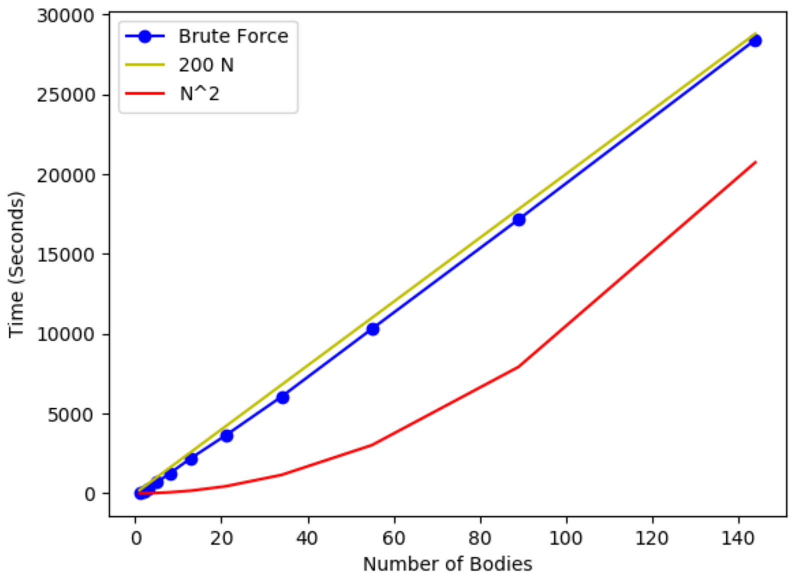

Fig. 9.— *Unexpectedly, the brute-force implementation looks like it is running in approximately linear time, scaled extremely heavily by a factor of 200. Additionally, an exponential function is visibly an extremely poor fit.*

4. Discussion

This project ended up being far more dominated by the implementation process than I expected, but was deeply satisfying. As a relative novice to both astronomy and programming, the project forced me to deeply understand concepts that I had previously glossed over with only a surface-level understanding.

There were several nagging conceptual issues that haunted me as I developed this code. One such problem was keeping the Barnes-Hut algorithm from becoming $O(N^2 \log N)$, since in each time-step, in addition to making the force calculations, the graph structure had to be recreated (the bodies must be resorted) to adapt to the bodies' new positions. This extra factor of N pushes the algorithm from being an improvement over the $O(N^2)$ of the particle-particle method to a decrease in performance thanks to an additional cost of $\log N$. This was solved, though, by simply sorting each body in as it had its force-calculations performed. The intuition one might have saying that such an approach would not work because every body must be in place for any body's total acceleration to be calculated, as originally occurred to me, is solved by recalling that in calculating those forces, we are actually using the positions from the previous time-step. Therefore, by retaining the previous time-step's graph, we can simply use that to calculate where each body moves and sort it into a new graph, one body at a time. While there were many other problems that arose - generally small, simple errors in my code, math, or physics - I thought that this issue was one of the more satisfying issues I had to solve. In the end, though, one core goal of this project - to write a more efficient implementation than the particle-particle method using the Barnes-Hut algorithm - was accomplished, as shown in Figure 7. That was quite satisfying, as was having successfully implemented these algorithms generally; watching GIFs of the planets in orbit with the graph restructuring itself constantly was a pretty great feeling.

But ultimately, so what? As my simulations have begun to show, the potential for these algorithms to be used to significantly inform our understanding of circumstellar (or even circumplanetary) disks. With a fully-implemented N-body simulator, one might study the ability of planets to swap orbits with other planets, as is believed to have happened in our own solar system. Or perhaps one might study, as Ryan did, mean-motion resonances. The authors of REBOUND have even used their simulator to model Saturn's rings. One could add in other forces or laws (beyond the simple gravity used here) to begin exploring the effects of the ice line on planetesimal growth (doing this has been on my mind a lot recently). The benefit in all these cases is, of course, that these systems are (literally) unimaginably complex, meaning that having the ability to simulate them lets us observe directly the effects of various physical laws.

5. Conclusions and Future Work

In this project, I developed two N-body simulation algorithms using two traditional algorithms, particle-particle and Barnes-Hut. The project ended up being more an exploration of the computational side of developing the algorithms (managing large blocks of code producing large amounts of data over long-period runs) rather than a particularly close examination of the scientific principles resulting from the code. Ultimately, though, I was able to push through lots (but not all) of mistakes and create data that model, at least approximately, the physical scenarios that they are meant to replicate. In addition to the model trajectories being fairly reasonable, the two algorithm's computational-cost profiles were ended up being approximately what they should be. Future work (this winter, for fun) will likely include developing a particle-mesh implementation and translating the code to three dimensions.

Fig. 10.— *Some beautiful mistakes and testing.*

REFERENCES

Harrison, D.; Gay, W.; & Effron, H. M.; *Algorithm for the Calculation of the Classical Equations of Motion of an N-body System* 1969: Journal of Mathematical Physics

Aarseth, S. J., J. R. Gott III, & E. L. Turner *N-body Simulations of Galaxy Clustering: Initial Conditions and Galaxy Collapse Times* 1979: The Astrophysical Journal

Barnes, J., & Hut, P. *A hierarchical O(N log N) force-calculation algorithm* 1986: Nature

Berendsen, H., van der Sproel, D. & R. van Drunen *GROMACS: A message-passing parallel molecular dynamics implementation* 1995: Computer Physics Communications

Kim, J., Park, C., Rossi, G., Lee, S. M., & Gott, J. R. *The New Horizon Run Cosmological N-body Simulations* 2011: Journal of the Korean Astronomical Society

Rein, H., Liu, S. *REBOUND: An open-source multi-purpose N-body code for collisional dynamics* 2012: Astronomy & Astrophysics

Melott, A.L.; Einasto, J.; Saar, E.; Suisalu, I.; Klypin, A.A.; Shandarin, S.F.; *Cluster analysis of the nonlinear evolution of large-scale structure in an axion/gravitino/photino-dominated universe* 1983: Physical Review Letters

This manuscript was prepared with the AAS LaTeX macros v5.0.