1 Introduction

For our computational project, we chose to investigate the n-body problem. As we showed in class, this system is not analytically solvable, so our investigation centers on simulating these masses interactions efficiently so that we can see how the system evolves based on various initial configurations. Solving n-body problems has applications in modeling celestial phenomena, and as a natural extension of the same techniques, other forces could be calculated to model the behavior of gases.

1.1 The Algorithm

For our simulation, we employed the Barnes-Hut algorithm, a tree-based approximation scheme that drops asymptotic computational complexity from $O\left(n^2\right)$ to $O\left(n \log n\right)$ with negligible degradation in accuracy. Our particular implementation can be broken down into the following steps:

(a) First, we initialize the masses as follows:

i) We randomly select the magnitude of mass's displacement from the origin of the space (denoted r) from one of three distributions:

1. A simple linear distribution (uniform range between some start and stop points r_a and r_b)
2. A normal distribution, characterized by a mean radius $\overline{r}$ and a standard deviation σ, yielding a probability density function of
$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\overline{r})^2}{2\sigma^2}}$$
3. A gamma distribution, characterized by two parameters: the "shape", k and, "scale", θ, yielding a probability density function of
$$f(x; k, \theta) = \frac{x^{k-1} e^{-x/\theta}}{\theta^k \Gamma(k)}$$
(where $\Gamma(k)$ is the gamma function. Source: *Gamma Distribution*, Paul E. Johnson).

ii) Next, we choose a second set of distributions (of the three above) from which to sample $\phi_1, \phi_2, \ldots, \phi_{n-1}$, to express the point in n-D spherical coordinates. We then projected the points back into $\mathbb{R}^n$, using the following formulae:

$$\begin{aligned}
x_1 &= r\cos(\phi_1) \\
x_2 &= r\sin(\phi_1)\cos(\phi_2) \\
x_3 &= r\sin(\phi_1)\sin(\phi_2)\cos(\phi_3) \\
&\vdots \\
x_{n-1} &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\cos(\phi_{n-1}) \\
x_n &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\sin(\phi_{n-1})
\end{aligned}$$

Where $\forall i \in \mathbb{N}$ st $1 \leq i \leq n-2, \phi_i \in [0, \pi]$, while for $i = n-1$, we have $\phi_i \in [0, 2\pi]$.

iii) Repeat i) for velocities, but only use a linear distributions for theta in ii) (we want our velocities to show no bias in direction, so that for large numbers of particles, the total momentum of the system will be approximately 0. Since all the thetas are simple linear distributions, we'd expect approximately uniform distribution of $\hat{r} = \mathbf{r}/r$ for large values of n).[1]

[1] Note that for repeatability during our testing phase, we used consistent RNG seeds between runs for each of our generator functions, each of which was itself generated by a static-ly seeded function. However, the current algorithm uses OS-derived RNG to seed the distributions.

iv) Next, we ingest the masses by pushing each generated body to a holding queue for the root node of our global tree structure.[2] At this point, initialization is complete, and we proceed to the main algorithm.

(b) First, the space is recursively subdivided into 2^n n-D subcubes of our overall simulation region (in the the 2-D case, these would be quadrants). Each sub-cube is subdivided further, until it contains either 0 or 1 masses, with an absolute limit on the minimum side-length of any region, which we store as a static float `MIN_LEN` (a tiny fraction of our global simulation width). This data structure can be visualized as a large tree, in which each node has either 0 or n child nodes.

i) At this point, to save on computation, any remaining sub-bodies are merged into a single body at their center of mass, with momentum equal to the sum of their incident momenta. This have a negligible effect on any force-calculations, since the vast majority of mass interactions will occur on length scales orders of magnitude larger than `MIN_LEN`.

(c) Once the initial tree has been drawn, we recurse downwards until we reach a leaf node. If the node is empty, we return. Else, there must be only one body inside the region, and we begin calculating the net acceleration on it.

i) To calculate the acceleration on a mass m, we start at the top of the tree, and compare the distance m and the center of the smallest bounding region about N. We denote this distance by d. Let the side length of the bounding region on N be denoted s. We define a "distance" metric, θ, as follows:

$$\theta = \frac{s}{d}$$

If θ is greater than some specific threshold value, then we calculate the center of mass of N, and use Newton's Law of Gravitation on this center of mass to obtain a portion of the force on m. Note that this is a reasonable approximation because, for large distances, the gravitational field from a collection of masses becomes approximately the gravitational field from their center of mass.

ii) Else, If the current region is not far enough away to make this approximation, we check to see if it has child regions, or if it holds a mass (note that because each region has 0 or 1 masses, these are mutually exclusive). If it has a mass, we calculate the acceleration because N and that mass.

iii) If the current region instead has child regions, we recursively call the same update acceleration function on each of them. This will recurse until it finds a region with 1 mass, an empty region, or a region far enough away to approximate. It will then return back to the top of the tree, combining the results of every acceleration calculation to get one total acceleration for N

1.2 The System

The simulation that we developed contains several masses, each of which have (non-relativistic) velocities and positions in a space of arbitrary dimensions. In order to obtain reasonably interpretable results, we chose to generate our plots using a 2-dimensional space, but there is abundant room for future study in analyzing the interactions of bodies in higher dimensions.

(a) The system, when run in 2 dimensions, has $2n$ degrees of freedom: $x_1(t), x_2(t), \ldots, x_n(t)$, and $y_1(t), y_2(t), \ldots y_n(t)$. Each body also has a constant mass.

[2] In the actual codebase itself, it should be noted that each instance of our `Region` class (which stores the data for a single node on our tree) is wrapped in special layering classes, called `Arc` and `Mutex`. The data for the `Arc` class is stored in read-only memory, and contains a reference to the `Mutex` class, which further wraps pointers to the object with a guarding layer that locks out new threads when the memory is being modified. Thus, we can safely have multiple threads checking whether the Region is currently in use (by calling methods on the Mutex) without creating data races.

(b) The system depends on several dimensionless constants. These values are arbitrary, but we chose them with several factors in mind. `DT`, the timestep, needs to be large enough to obtain results in a reasonable amount of time, but small enough that the simulation is reasonably similar to the continuous real-time system. We chose `G` to be 16000 because this is easy to plug into equations to determine the initial conditions that generate simple trajectories (for example, a circular orbit, see below).

2 Analysis

Because this setup is too general to study and get meaningful results, we simulated several sets of initial conditions that have the possibility of giving interesting behavior. These conditions include:

(a) A two-body gravitational simulation with identical masses. Because this is analytically tractable, this configuration is a good way of checking that the simulation gives accurate results. It also gives a way to analyze the error in the simulation caused by rounding and discrete time steps.

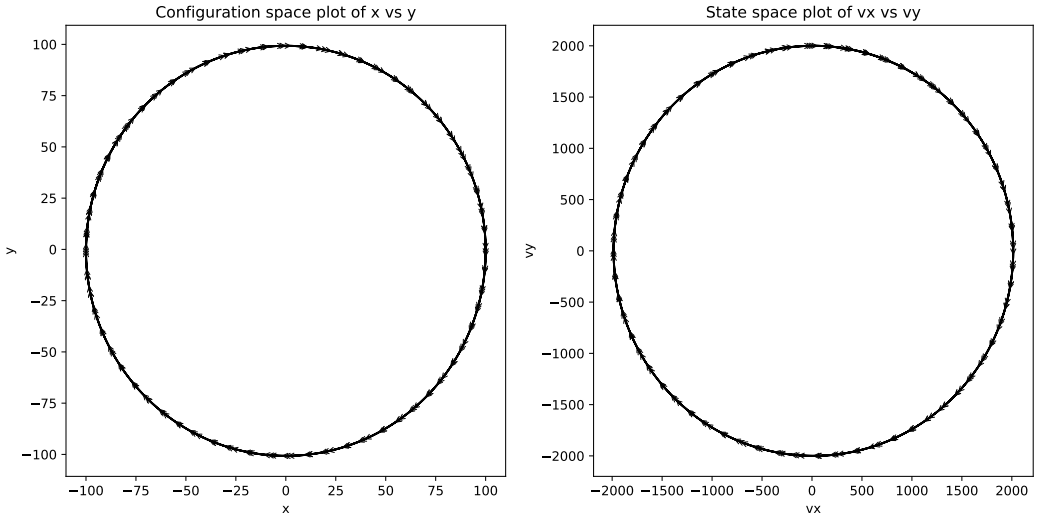
(b) A system with two large objects orbiting, and several smaller objects injected randomly around them, similarly to the three-body simulation showed in class. This simulates a binary star system, with a number of planets or other celestial bodies interacting with it. Some have bound orbits that eventually get close enough to the large bodies that they collide, and some have enough velocity to escape.

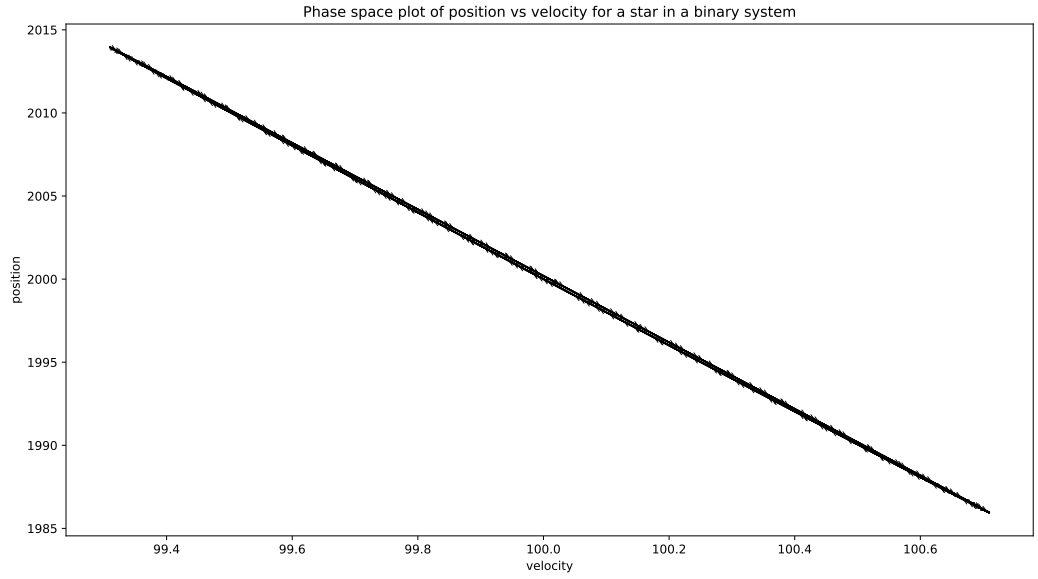
(c) A scattering problem, with two large orbiting masses in the center, and a large set of small masses injected from the side. This shares some elements in common with the binary system (b), and with the scattering calculations done in class.

3 Findings

(a) Placing two bodies with positions and velocities that would generate a circular orbit in Newtonian physics lead (thankfully) to two the bodies orbiting their shared center of mass.

As we can see, the shape of the orbit is circular with a very slight precession. This can be explained in part by the numerical rounding used by the programming language. In addition, the fact that discrete timesteps were used means that, instead of being a true circle, each body's orbit is composed of many linear steps that form a circular shape. This means that the bodies are not interacting continuously throughout their orbits, but only once every time step. Therefore, it is unsurprising that there is a small perturbation from the analytical solution, but as we can see, this is small enough that the simulation can still be used to give an excellent approximation of behavior.

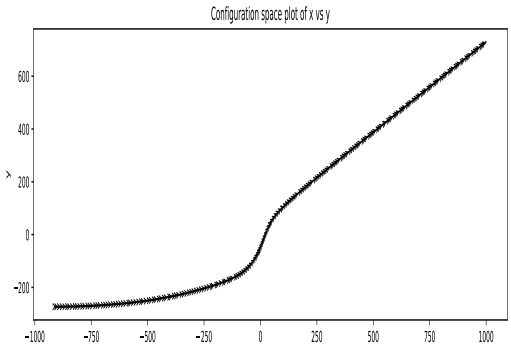


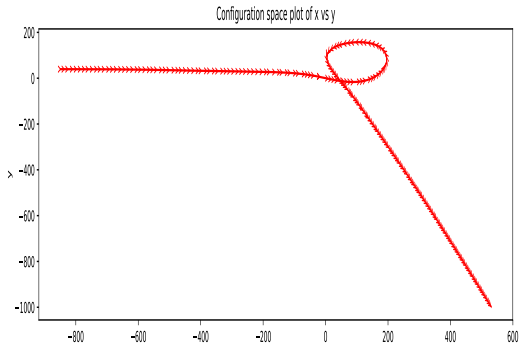


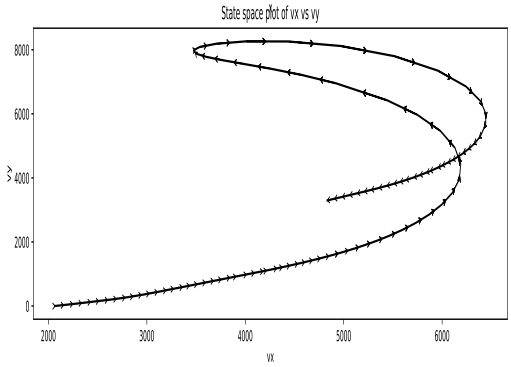
(b) As shown in the attached video (`https://youtu.be/BcRRBVifNFI`), each of the masses injected into the binary system has slightly different initial conditions, but their final conditions are vastly different. Most are ejected from the simulation, some collide and are absorbed by the large masses, and a few continue to orbit in an unstable configuration around the large masses.

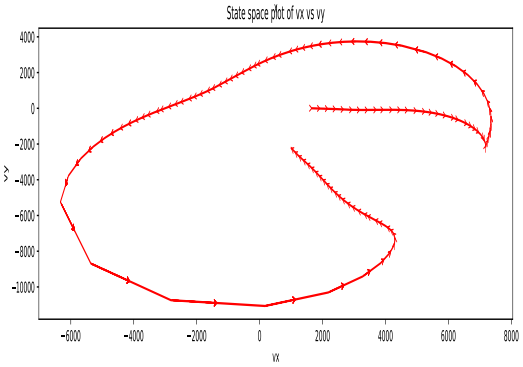
(c) We see again in the following plots that, even for small changes in initial conditions, bodies have completely different trajectories both in phase space and in configuration space. Because of the size limits of our simulation, we saw widely varying scattering results very every impact parameter that was tested. However, there is room for futher investigation by testing much higher impact parameters to find the ones for which bodies are negligibly scattered by the binary system.

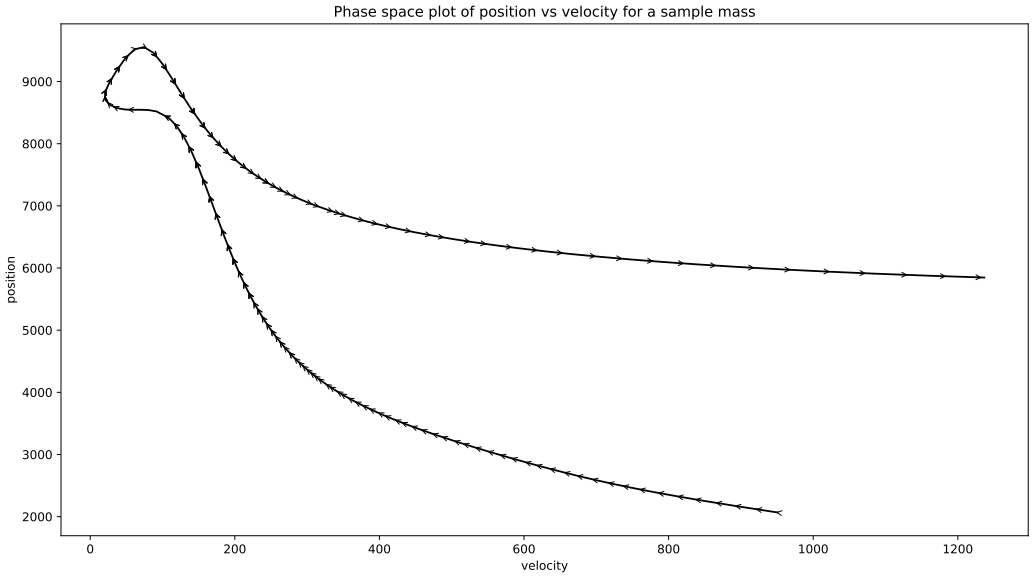
The following plots track two masses injected from the left and scattering off of a system of two large bodies orbiting circularly about their barycenter.

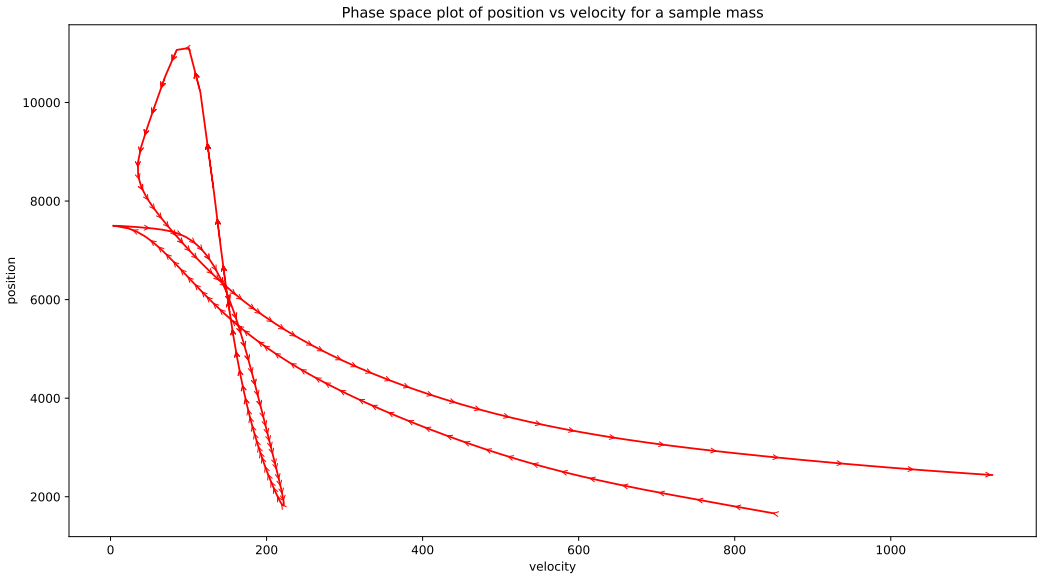












4 Conclusion

The predominant theme in analyzing our findings for each system was that final state differed significantly even for small changes in the initial conditions. From this we conclude that our system is chaotic. This is not at all surprising — after all, the n-body problem is known to be analytically intractable. We further conclude that, although our simulation contains very small numerical errors due to rounding and discrete time steps, these are small enough (and accumulate slowly enough) that we can extract meaningful data about real interactions from our simulation.

The bulk of the work involved in this project was coercing Rust into compiling a working simulation. However, now that it is finally functional, there are several very interesting questions we would like to pursue further. In particular, simulating larger collections of masses will likely yield interesting emergent properties, such as oscillations among large groups of bodies.[3] It would also be interesting to add a correction term to acceleration, like we saw in class, to account for some of the effects of general relativity. Furthermore, although calculations of energy are difficult based on the program framework, it would be very interesting to see the evolution of the total kinetic and potential energy of the system over time, and check if results are consistent with statmech, and also examine how the corrective GR term affects the distribution. Finally, we'd like to set up n-dimensional simulations that are symmetric across certain planes through $\mathbb{R}^n$, and look at how they behave when projected onto the $x - y$ plane.

Finally, here's another small demo of our program running: `https://youtu.be/P-yFOgdxxb0` (Note that the simulation is running slowly because my computer was low on battery, not because the simulation is slow)

And finally, a link to our git repo: `https://github.com/redpanda1234/barnes-rust`

[3] Currently, the largest simulation we have been able to run at a reasonable pace is about 25,000 masses, but we'd like it to go faster.