N-Body Simulation
or
"What I Did Over Christmas Break"

1 Background

A large part of this thesis concerns the motion of many bodies in the presence of a Kerr black hole. I give each body a Newtonian gravitational potential and look at how that affects their trajectories. I now present the "n-body problem" and explain briefly why it is difficult and what techniques I employ to solve it computationally.

1.1 The N-Body Problem

Presented with two bodies which interact purely gravitationally – planets, stars, grains of dust – I can write an equation which will give me the position of each body at any point in time. This problem has been solved since the 18th century and follows pretty naturally from Newton's observation that the force of gravity between two bodies falls off as the inverse square of the distance between them. We can write this mathematically as

$$\mathbf{F}_1 = \mathbf{F}_{12} = \frac{1}{r_{12}^2} G m_1 m_2 \hat{\mathbf{r}}_{12},$$

for two bodies, of masses m_1 and m_2, respectively. G is the gravitational constant, and r_{12} is the distance between the body 1 and body 2. $\hat{\mathbf{r}}_{\mathbf{12}}$ is a vector that points from one body to the other (Fig. 1a). Here, "$\mathbf{F_1}$" is the force on body 1, and "$\mathbf{F}_{12}$" is the force on body 1 *due to* body 2, so it is read "F-one-two" and not "F-twelve" (and likewise for r_{12}). In this case, $\mathbf{F}_1$ and $\mathbf{F}_{12}$ are equal, but that is not always true as we will see in a second.

As I said, the problem for the gravitational interaction between two bodies is totally solved. However, if I introduce a third body (Fig. 1b), the story is made murky: Now each of the planets or stars or whatever is being pulled simultaneously by *two* other bodies. The force on any one body at a given instant is described by the sum of the forces due to the other two bodies. For instance, the force on the first body would be given by

$$\begin{aligned}\mathbf{F}_1 &= \mathbf{F}_{12} + \mathbf{F}_{13} \\ &= \frac{1}{r_{12}^2} G m_1 m_2 \hat{\mathbf{r}}_{12} + \frac{1}{r_{13}^2} G m_1 m_3 \hat{\mathbf{r}}_{13}\end{aligned}$$

This will move body 1 in some direction, but bodies 2 and 3 are *also* experiencing gravitational pulls and so they too are constantly moving around. So the total force on any given body is constantly changing in direction and magnitude.

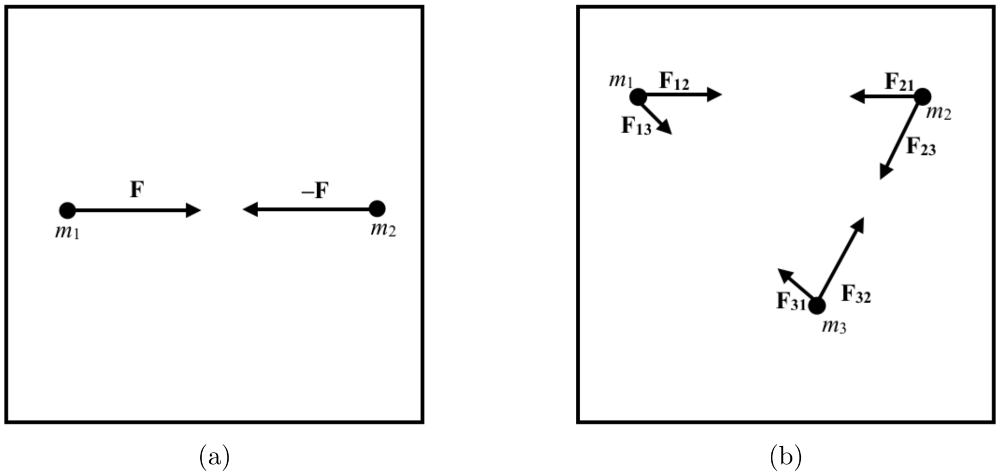

Figure 1: On the left, two bodies are being influenced by each other's gravitational pull. The force that m_2 experiences is equal in magnitude to the force that m_1 experiences, but in the opposite direction. With 3 or more bodies (depicted on the right), "equal and opposite" has no meaning– each body feels the sum of the pull from every other body.

It turns out, unlike for the 2-body case, it is impossible to write an equation which will totally describe the positions of all three bodies at any point in time. In fact, this is true for *any* number of gravitationally interacting bodies greater than 2. The problem of determining these positions after some amount of time is dubbed the *n-body problem.*

1.2 Solution: Time Stepping

In order to model systems of many gravitationally interacting bodies, then, we need another method, and so we come to *time stepping.* Time stepping comes from the observation that if we know the force on a given particle, we know exactly how that particle will move, *assuming the force is constant.* This follows from the equations of kinematics. For instance, if there is a particle with mass m at position $\mathbf{x}_0$ with velocity $\mathbf{v}_0$, we know that in some time interval Δt, its new position $\mathbf{x}_1$ will be given by

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{v}_0 \Delta t.$$

But if that particle is experiencing some force, $\mathbf{F}$, then its velocity will also change: after time Δt, it will be accelerated to a new velocity, $\mathbf{v}_1$. We need another equation to describe this. The acceleration on the particle is just the force divided by its mass, $\mathbf{a} = \mathbf{F}/m$, and so the particle's position and velocity change in time Δt according to

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{v}_0 \Delta t.$$
$$\mathbf{v}_1 = \mathbf{v}_0 + \mathbf{a} \Delta t.$$

Thus we can totally describe the motion of a particle under a force *assuming the force is constant.* But how does this help us? Section 1.1 was all about how the forces in the n-body problem are extremely *un*constant.

If we're being exact, that's true. In some time Δt, the bodies will move around, and the distances between each body will change, and so the force will change as well. But if we make Δt quite small, then to very good approximation, the force will stay constant *in that time interval.* Then we can move the bodies around as if the force on each is constant in that interval, then at the end of that interval, reevaluate their positions, calculate new forces, and move them again for the same amount of time, and repeat. The smaller we make Δt, the more accurate the approximation is.

Each Δt is usually called a *time step*, and this describes the method of time stepping. The whole process described above is presented as a flowchart in Fig. 2. This approximation technique is how we can probe the behavior of n-body systems. Of course, this can extremely computationally intensive. In a system with 10,000 bodies, calculating the force on any given body in a single time step is a total nightmare for a human. It is only because of computers that time stepping is possible, and even then, there are many optimization methods we must use to make the problem tractable.

2 N-Body Simulation in the Context of this Thesis

Here I cover what I have done so far for n-body solving, the test cases that show it's working, and how I plan to optimize it. This accounts for the minority of my work over break (which is not to say it was easy; it wasn't.). The bulk of the work has been in making it talk to the Kerr RK4 integrator, which is not covered here because that was a big enough effort to warrant its own document, which I am currently working on[1].

2.1 Overview

At the moment, I do no optimization. My first, and so far only code is found in `nbody_examples.nb` a brute-force method which solves Newton's second law (NSL) exactly. It uses a func-

[1] Everything in my code is totally modular, so when I get a better version of the n-body code I can just change which n-body function I call in the Kerr RK4 code. The current n-body is a proof of concept which is slow and crappy, but gets the point across.

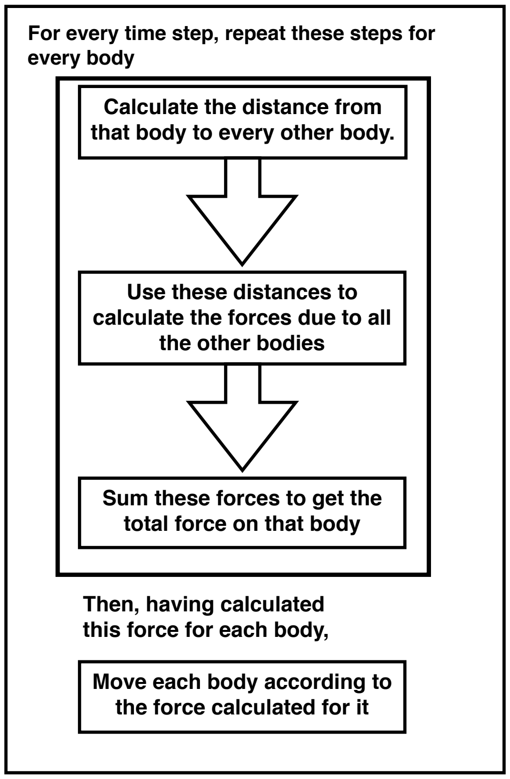
For every time step, repeat these steps for every body
Calculate the distance from that body to every other body.
Use these distances to calculate the forces due to all the other bodies
Sum these forces to get the total force on that body
Then, having calculated this force for each body,
Move each body according to the force calculated for it

Figure 2: The algorithm for time-stepping an n-body system.

tion `SingleParticleNewtonianForce[{state},{masses},i]` which is also found in the `ThesisTools.m` Mathematica package. `SingleParticleNewtonianForce` has three arguments, the first two of which are arrays: `state` and `masses`. `state` has the phase space coordinates of each particle and has the form $\{\{\{x_1, y_1\},\{\dot{x}_1, \dot{y}_1\}\},\ldots,\{\{x_n, y_n\},\{\dot{x}_n, \dot{y}_n\}\}\}$ for n particles. `masses` has the form $\{m_1,\ldots,m_n\}$. The third argument, `i`, is the index of the particle being evaluated. The function returns $\{\ddot{x},\ddot{y}\}$, the total acceleration on the i^{th} particle (I know, I know, the function is misnamed).

`nbody_examples.nb` then has its own little function, `GetForces`, which loops through each particle and returns an array with the total acceleration on every particle. A third function calls `GetForces` at each time step, and updates the phase space accordingly. (all of the files mentioned are on the GitHub, natch).

2.2 Tests

Here are a couple of tests I ran to make sure it can handle simple cases. Fig. 3 has the initial conditions

$$\begin{aligned}
x_0 &: \pm 1 \\
\dot{x}_0 &: 0 \\
y_0 &: 0 \\
\dot{y}_0 &: \pm .3
\end{aligned}$$

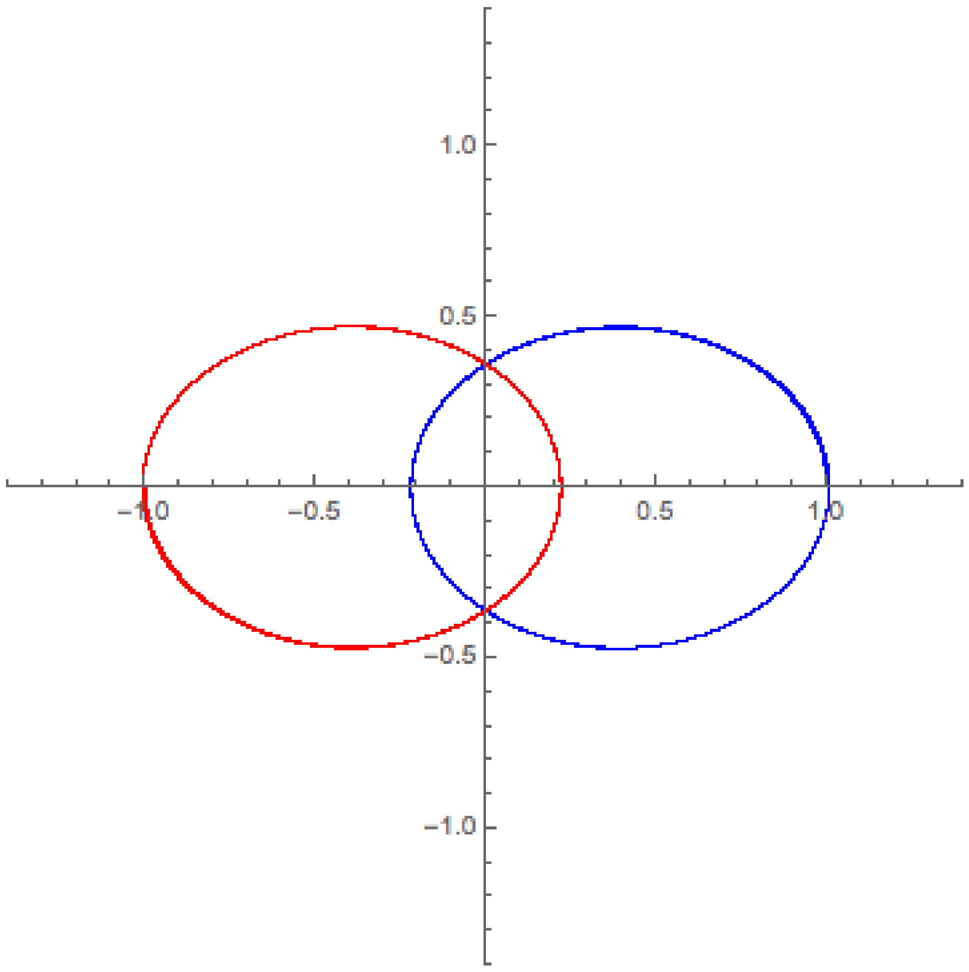

Figure 3: Two bodies with symmetric initial conditions in a nice closed orbit

And Fig. 4 has three bodies with the same initial condition, except for the third body which is starting with

$$x_0 : 0$$
$$\dot{x}_0 : -.3$$
$$y_0 : 1$$
$$\dot{y}_0 : 0$$

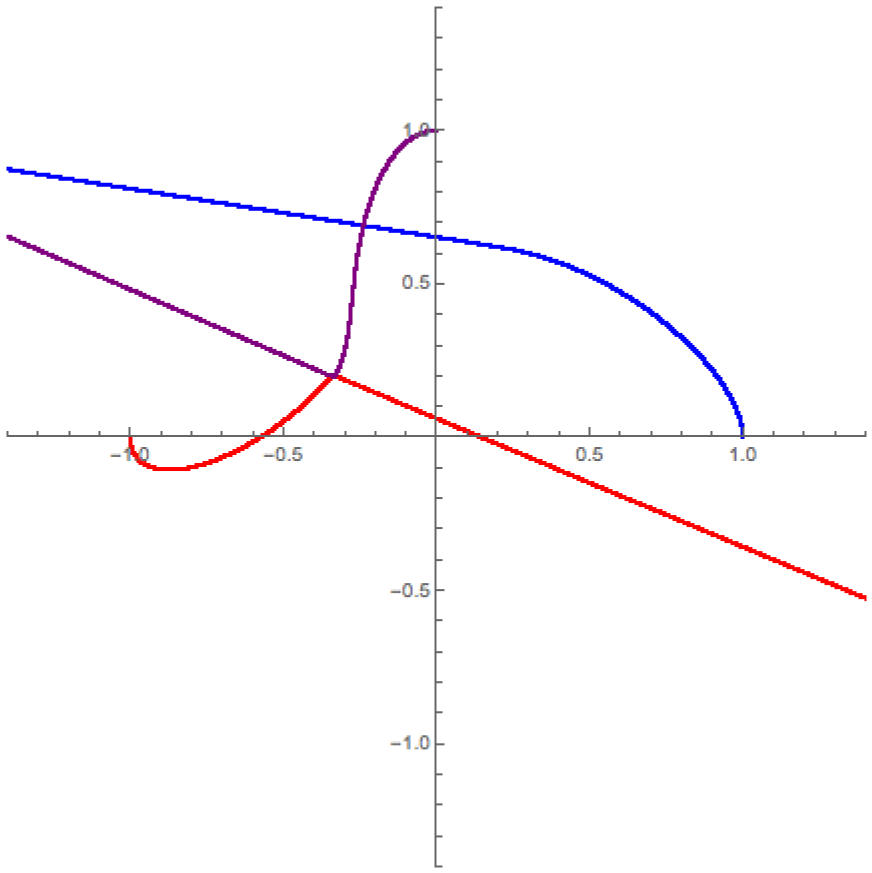

Figure 4: Three bodies. I have also gotten cute solar-system-ey things going but those take a while to set up.

2.3 Problems and Where It's Going

- It has really bad error accumulation

  There's a good example of this in the `nbody_example.nb` notebook, but basically this code accumulates error really badly. There are numerous methods to cut down on error, but I have not done enough research to know which one I'll use.

- There's no optimization

  This is the dumbest n-body code you could possibly write. I need to cut down on the computation. I think the first thing I'll try is a sphere-of-influence approach, where each particle's gravity cuts off after a certain distance from it (with the distance depending on the particle's mass). I think I might look into an adaptive step size, but I don't know how well that applies to many particles.

  The Piet Hut resource I mentioned last semester also has some good ideas.

- I want to include inelastic collisions.

  I often get crazy high forces and speeds because the particles can get arbitrarily close (as seen in Fig.4). I might say that if two particles are within a certain epsilon, they have collided. I then remove them both from the state vector and replace them with a particle that has their aggregate mass.

  Other than these things, I'm pretty happy. It definitely needs to get more sophisticated, but it's working well enough now for the combined Kerr-Nbody code, which you'll see a report on quite soon.