Particle Dynamics in a Time-Dependent Kerr Geometry

A Thesis
Presented to
The Division of Mathematics and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Alex Deich

May 2016

Approved for the Division
(Physics)

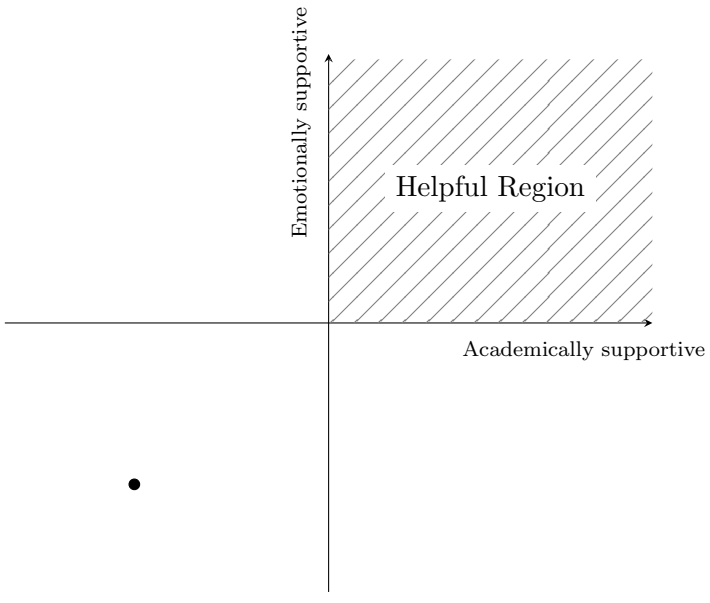
_____

Alison Crocker

Acknowledgements

In thinking about whom to thank, I found that everyone's contribution was some combination of academic and emotional support. People like my mother and father, for example, gave me tremendous emotional support as I worked on this thesis, while someone like my advisor Alison Crocker provided totally necessary academic support. Others, like my officemate Naomi Gendler, were helpful in both areas. When plotted altogether, some of the data created a so-called "helpful region" in the first quadrant (the shaded region below) corresponding to everyone who had a positive academic or emotional effect. Those occupying this region include:

- Abrar Abidi
- Aaron Deich
- Will Holdhusen
- Zuben Scott
- Sidney Vetens
- Alison Crocker
- Joel Franklin
- Naomi Gendler
- Mike Sommer
- Colleen Werkheiser
- Dad
- Ace Furman
- K. Cole Newton
- Mom
- Indy Hsu Liu



Those not in the helpful region:
- Evan Peairs

Table of Contents

List of Figures

Abstract

A spinning mass induces a Kerr spacetime. To investigate particle trajectories near a spinning mass, I present code which integrates the orbits of an arbitrary number of particles in a Kerr spacetime due to a body of arbitrary, time-varying spin. Several particle-particle interactions are considered, and large-scale behavior of disks during a spin-up event is investigated.

Dedication

Thanks to Ryan Campbell for setting me on the academic path which would eventually lead to this thesis. He set me going in a field of study I really love and showed me the joy of the twinklies in the sky.

Introduction

0.1 What is this thesis trying to accomplish?

In this thesis, I set out to make a simulation of many bodies orbiting a spinning central body in a generally relativistic treatment by performing an n-body numerical integration of particle trajectories. What follows is an explanation of why general relativity is desired for this study and what physical systems we might seek to model with such a simulation.

0.2 Why general relativity?

Picture a planet orbiting a star. Or two stars orbiting each other. Or a whole galaxy of stars orbiting a supermassive black hole. How can we predict and explain the motion of these bodies around one another? This thesis presents one particular model, general relativity.

The first model which successfully predicted how these bodies would move around each other was devised by Newton in the 1600's. Newton's model of gravity relies on the idea of gravitational **force**: each body moves according to the sum of forces acting on it by every other body, where the gravitational force is inversely proportional to the distance between bodies.

The Newtonian model was fantastically good at predicting the orbits of the planets, moons, and asteroids that we see in our solar system. It was so good at prediction that it's still used today in supercomputer simulations of galaxy collisions. It predicts to high accuracy many orbital phenomena, e.g. orbital period, ellipticity, asteroid populations, and models of the formation of planetary systems. It was so successful that astronomers could find very few astrophysical phenomena that weren't consistent with it. It seemed like Newton had truly figured out the way orbits and gravity worked.

At the end of the nineteenth century, however, astronomers were getting little hints that Newtonian gravity was somehow deficient at predicting all gravitational phenomena. The first such hint came with a careful observation of Mercury's point of closest approach to the sun (its **perihelion**). A French mathematician, Urbain Le Verrier, first noticed in 1859 that the perihelion was precessing faster than predicted by Newtonian gravity. In a standard Newtonian treatment of two bodies, the perihelion is fixed, but it can move over time as a result of the gravitational tugs of

other planets. Even after accounting for the influence of the known planets at the time, Le Verrier found there was a discrepancy between what he calculated and what was observed. The discrepancy was minuscule: every 100 years, Mercury would be 43 arc seconds further along in its orbit than was predicted by Newtonian gravity. Still, the data were known to high accuracy, and astronomers could not ignore this as measurement error[1].

This observation, combined with some theoretical considerations, were making a compelling case in the early twentieth century that Newtonian gravity was incomplete at best. Notice, though, how incredibly accurate it is. It gets Mercury's precession wrong by arc *seconds* per *century*. It is a testament to the theory that such high precision was required to show that it was wrong.

To solve the deficiencies of Newtonian gravity, Albert Einstein formulated the theory of **general relativity**, which solved not only the problem listed above, but made many more predictions we are still verifying today. General relativity (GR) tells a completely different story from Newtonian gravity. Instead of inducing a force, mass changes the *distance* between points which are near the mass. Furthermore, it says that particles move along the shortest paths between these points, which are not necessarily straight lines. It is a remarkable and nonintuitive theory, with many bizarre implications and predictions.

Of general relativity's many unique predictions, the one which is relevant to this thesis is the prediction that the trajectories of particles orbiting a spinning mass are perturbed as a consequence of the spin. This prediction is totally unique to GR; Newtonian gravity says nothing about anyone's spin.

These perturbations tend to totally defy intuition and are therefore personally fascinating to me. Furthermore, considering these GR effects has interesting observational implications. If we understand these implications well, we are better equipped to interpret observations of astrophysical phenomena.

0.3 Predictive value of this simulation

Although the phenomena considered in this thesis are well understood mathematically, simulation is still required to provide meaningful physical prediction. This is because the relevant mathematics is so cumbersome and impenetrable, the large-scale behavior is not obvious without either significant simplifying assumptions or computer simulation.

What physical scenarios are relevant to a simulation which looks at the GR effects of the spin of a central body on orbiting particles? As mentioned above, the effect on particles' motion due to GR is tiny, especially if you're considering objects the size of the Sun, or any of the things orbiting it. While this simulation can certainly be used to look at solar-sized objects, we can get much more severe behavior by modeling

[1] A few models were proposed to account for the "lost gravity": Maybe Jupiter's mass had been underestimated, or maybe there was an undiscovered giant planet far from the sun. However, Jupiter's mass was well-constrained by observations of the motions of its moons and a distant giant planet would have effects on all planets, not just Mercury.

instead things which are much denser and moving much faster. This will be more helpful in shaping our intuition about the effects of GR.

0.3.1 Black holes

It is for this reason that we restrict ourselves, for the most part, to studying the motion of particles in the vicinity of a spinning **black hole**. A black hole is a massive object of zero size, and therefore has infinite density. They are often extremely massive, and often spinning very quickly. Happily, unless you get too close, an orbit around a black hole looks exactly the same as an orbit around a star or planet; that is, it's an ellipse with the black hole at one focus. So, because black holes represent the extremes of mass and energy, we can probe the exotic behavior we seek. But at the same time, because we know what sort of motion to expect (elliptical), we have a benchmark for what counts as "weird".

A spinning black hole has two unique features not found in things we are familiar with like stars, planets, and moons. These will affect the motion of orbiting particles, and we will look for these in the simulation.

- *Event horizon*: Imagine I had a cannon pointing vertically straight up, and I was recklessly firing cannon balls out of it one after the other. For each cannon ball, I pack slightly more gun powder than the ball before it, and I observe that the balls take longer and longer to return to the ground. (I am very careful to avoid them; I may be reckless but I'm not stupid.) Eventually, I fire a cannon ball so fast that it doesn't come back. This cannon ball was given so much velocity that the Earth's gravity was incapable of pulling it back. We say that this cannon ball has **escape velocity**.

  A defining feature of black holes is that there is an imaginary surface surrounding them from which the escape velocity is equal to the speed of light. This surface is called the **event horizon**. Because nothing can exceed the speed of light, nothing can ever cross the event horizon and return.

- *Ergosphere*: Think of a whirlpool, where water is flowing around and towards a point in the ocean (or lake or whatever). This will serve as a reasonable analogy of a spinning black hole. If you're far from the center of the whirlpool, you will feel a slight tug towards it, and the water will be pushing you around it. In the same way, far from a spinning black hole, there will be a slight acceleration around the center. In both cases, it is easy to resist the motion. As you get closer to the center of the whirlpool however, there might come a point where the angular flow of the water is too much to fight, and there is nothing you can do to stop from being sucked into the middle of the whirlpool. Similarly, there's a point outside the spinning black hole where it's impossible to resist the angular motion. The only difference is that it's literally impossible to resist; even light is obligated to travel in the direction of the spin of the black hole. This region outside a black hole where any particle must co-rotate is called the **ergosphere**. The ergosphere is responsible for some of the most exotic behavior

of orbiting particles. If a particle enters the ergosphere traveling against the direction of spin, it must reverse its motion as it passes into the ergosphere. While such orbits are not stable, and must fall into the center of the black hole, there do exist stable orbits which briefly enter the ergosphere.

Black holes provide laboratories in which to probe the most extreme behavior that GR predicts.

0.3.2 Systems of interest for the simulation

We're looking at groups of multiple particles orbiting massive spinning objects (which can be black holes), and we're looking at how changes to the spin changes the particle's orbits. There are two somewhat unrelated ways we explore this scenario in this thesis.

- *Protoplanetary disks*: First, we can look at a large number of small bodies orbiting in a flat disk. This constitutes a rough model of a **protoplanetary disk**. Protoplanetary disks are disks of debris that form around many massive objects in space. The debris range in size from pebbles to houses. If they form around stars, it is believed they can slowly accrete into planetary systems. Specific to the simulation, a protoplanetary disk would not be modeled with any sort of gravitational force between the bodies that make up the disk. They are too small, and any sort of gravitational force would be negligible. Instead, interaction of the bodies would be modeled by elastic collisions: The bodies are just balls bouncing into each other.

  Some questions we might ask about protoplanetary disks include: How does the structure of the disk depend on the change of the spin of the central body? What would be some observable qualities of such a change?

- *Small numbers of interacting particles*: Second, we can turn on the gravitational interaction ignored above and look at how interacting particles are disturbed by the general relativistic effects. This would be an appropriate model of, say, a solar system orbiting a a black hole. "Small numbers" can be understood to be fewer than 10 or so, and is merely a consideration for restrictions of both time and computation power.

  Some relevant questions about these systems might include: How is the stability of the solar system affected by changes in the spin of the central body? What kind of approximations are reasonable when considering the broader general relativity context?

Finally, it is helpful to remember that *everything* in the universe is spinning. It is impossible to have *exactly* zero angular momentum, so a full treatment ultimately requires something like what is being considered here. Now, it's also the case that any effects of this spin are often negligible, but it is nonetheless true that what I am investigating in this thesis applies to nothing less than literally everything.

0.4 When to use Newtonian gravity and when to use general relativity

I spent a lot of energy above explaining how this thesis is looking at general relativistic effects and why that might be something interesting to pursue. So you might think that when I "turn on" the gravitational interaction between particles orbiting a black hole, I might be "turning on" a GR interaction. Wrong! My simulation treats the particles as Newtonian particles. Why? Well it turns out that calculating GR effects for multiple bodies is an incredibly difficult problem. It can only be done with significant computational power, and was only solved in 1992[2]. So I instead treat the particles as interacting with Newtonian forces, and this is actually okay: as I explained above, Newtonian gravity agrees with GR for all but the most extreme effects. It is an interesting problem in its own right to look at the effects of GR on a large number of bodies interacting with Newtonian forces. It poses many problems without obvious solutions, and is the subject of a lot of this thesis.

0.5 Why a computer simulation?

I said above that a simulation is required because the math is too complicated, but that's only part of the answer. It turns out that when you consider more than three gravitationally interacting bodies, it is *impossible* to write down an equation which gives the position of every particle at any given time. Indeed, even with exactly three interacting bodies, there are only a handful of classes of solutions [3]. The *only* way to figure out how a certain system will change in time is to plop it into a computer and see what happens.

By "plop it into a computer" I mean this: For a very small time interval, pretend as if all the forces between the particles are constant. Then, change all the positions and velocities of the particles according to those constant forces. Recalculate the forces for the next time interval and repeat. This process is expanded in much greater detail in chapter 3, and is called **numerical integration**.

0.6 What will this thesis produce?

I've covered a lot in this introduction. At the end of the day, this is what the reader will be able to take away from this thesis:

- *Simulation code*: Most of the work for this thesis was in writing the *Orbit Integrator for N-body Kerr Solutions* (OINKS), the simulation code. OINKS is a robust Python-based code which I hope others can use to look at other systems near a massive spinning object. The code and documentation is included as an appendix.

[2] Aarseth (2003)
[3] Damour et al. (1992)

- *A quick introduction to general relativity*: General relativity is a complex theory, but I hope to provide a meager introduction in this thesis. I include a discussion of many of the important implications, and a cursory overview of the mathematics involved. I think most importantly, I discuss several arguments to introduce some intuition to the theory.

- *A quick but thorough introduction to numerical methods*: My code uses a fourth-order Runge-Kutta integrator. Why did I chose this integrator? What are the benefits and drawbacks? I also provide pseudocode implementations of the algorithms used.

- *A discussion of my simulations, with both analytic and numerical analyses*: Each scenario I simulate has both analytic and numerical results which are worth discussing. This will also connect each simulation to its physical significance.

Chapter 1

Relativity

What this chapter is and isn't

This thesis is exploring a subtle result from the theory of general relativity (GR), and this chapter attempts to give some intuition about how the theory works. This is not intended to give full derivation of all the GR results used in this thesis, and later chapters will assume the reader has had at least a class in general relativity.

However, I can at least attempt to bring to light some of what I find extremely beautiful about the theory, and explain in prose the thrust of what general relativity says in math. It would be too great a task, however, to give a first-principles derivation of everything, so I assume for *this* chapter that my reader has about the knowledge of a physics student somewhere in the middle of their third term. Sometimes I give a brief sketch of the mathematics involved, sometimes I work through a simple toy example, and sometimes I just discuss it qualitatively. It is my hope that this will give the reader sufficient analytical tools to appreciate what comes later.

Here's an outline of what follows. Note that each section below is titled with the specific idea I think will be helpful for later parts of this thesis:

- Special relativity: Spacetime and 4-vectors. Many of the results used later on derive from several facts about the theory of special relativity, so I will briefly review these before beginning general relativity. This section will also introduce the concept of spacetime, which is an essential ingredient for any treatment of general relativity.

- The math of general relativity: Some differential geometry. I said before that GR says that mass and energy change the *distance* between points in spacetime. What does this mean and what are some consequences?

- Particles take the shortest paths. This thesis concerns the dynamics of particles in a generally relativistic context. How can we derive equations of motion for particles in a curved background?

- Comparison to Newtonian results. This section will, I hope, provide some intuition to the results derived earlier. It will show how GR replicates, and improves upon, those derived from Newtonian gravity.

1.1 Special relativity: Spacetime and 4-vectors

By the beginning of the twentieth century, a substantial amount of empirical evidence had been collected which suggested a surprising result: no matter how fast an observer is going, that observer will always measure light to have the same speed, or,

the speed of light in vacuum, c, is the same in every reference frame.

This fact has important ramifications on the concept of length and time, and by extension, velocity. What follows is a conventional thought experiment to demonstrate these ramifications, and the new mathematical constructions which are required to describe the physics of special relativity.

Suppose there are two laboratories: Lab A and lab B, which is moving with constant velocity v_B to the right with respect to lab A (fig. 1.1). The two labs have their own equipment to measure distance. Lab A will measure distance along the axis x_A, and Lab B will measure along x_B.

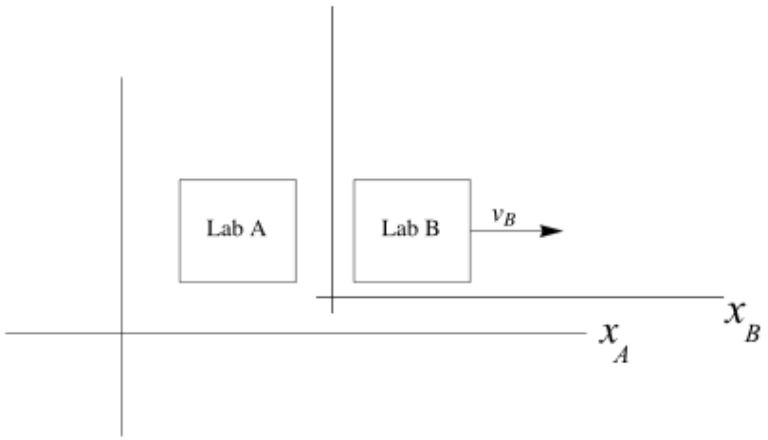


Figure 1.1: Two labs in different inertial reference frames.

Now suppose that Lab A fires a beam of light in the direction of Lab B. Question: After a time Δt, how far does the beam of light travel as measured by Lab A, and how far does it travel as measured by lab B?

For lab A, the answer is straightforward: we know that light travels at speed c, so the distance it travels as measured by lab A must be

$$\Delta x_A = c\Delta t, \tag{1.1}$$

For lab B, the answer seems obvious at first: light travels at speed c, and the lab is traveling at speed v_B, so it must measure the light to travel a shorter distance than lab A:

$$\Delta x_B = (c - v_B)\Delta t < \Delta x_A \tag{1.2}$$

But this cannot be true! Numerous experiments have confirmed to great precision the fact that the speed of light is *always* c, regardless of how fast you're going. So what gives? Well, we've carried out this whole analysis assuming that both labs experience the same Δt. But what if this weren't the case? What if, instead, lab A experienced Δt_A time and lab B experienced Δt_B time and that $\Delta t_A \neq \Delta t_B$? Ignoring, for the moment, what that would mean for two different labs to measure two different times, the measurements of Δx become easier:

$$\Delta x_A = c\Delta t_A \tag{1.3}$$
$$\Delta x_B = c\Delta t_B \tag{1.4}$$

This is, in fact, what happens. The distance as measured by a particular observer is called that observer's **proper length**, and the time as measured by a particular observer is that observer's **proper time**. So this means that whenever you specify the when and where of an event, you also must specify who is observing it. These points constitute the components of an event's position in **spacetime**.

An event in spacetime is specified by an object called a **4-vector**, which is a vector with components for both spatial and temporal information. It's called a *4*-vector for its four components: In the Cartesian coordinate system, it has a time component (also called the 0 component), and one component for x, y, and z. Let's look at (1.3) as an example, keeping in mind that the y and z components are 0. (1.3) describes the position of a beam of light in spacetime as measured by lab A, otherwise known as its 4-position. Classically, we might write a position vector as

$$\mathbf{x} = a\hat{\mathbf{i}} + b\hat{\mathbf{j}},$$

for some vector $\mathbf{x}$ with components in the $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ directions. The difference with (1.3) is that the sign of the spatial component must necessarily be the opposite of that of the temporal:

$$\Delta x_A = -c\Delta t_A + \Delta x_A.$$

Whether it's the temporal or spatial component that gets the minus sign is arbitrary convention. It is called the metric signature. This thesis will use the convention of writing the temporal component as negative, often written as the $(-+++)$ signature. 4-vectors are handily written using **Einstein summation notation**, where a vector is written as

$$x_\mu \text{ for } \mu = 0, 1, 2, 3.$$

As a note on notation, if the index is given by a Roman character rather than greek, the sum is understood to be taken over the spatial components only:

$$x_j \text{ for } j = 1, 2, 3.$$

You can also raise the index, written as x^μ. This has a formal meaning, but the important information for now is that multiplying a raised-index vector by a lowered-index vector implies a sum over that index:

$$x_\mu x^\mu = x_0^2 + x_1^2 + x_2^2 + x_3^2 \tag{1.5}$$

Finally, each additional index is another tensor rank, so x_μ can be interpreted as a vector, $x_{\mu\nu}$ can be interpreted as a matrix, and $x_{\mu\nu\rho}$ is a third-rank tensor.

So we now have the notion of the spacetime 4-vector, and the fact that in order to specify the position and time of a particle, you must also specify who is observing it. This brings up an important point: If you know how one observer describes the 4-position of a particle, how can you get the 4-position in another reference frame? This is solved by the **Lorentz transformation**. I won't go over the derivation here, but a really excellent version is given in Feynman (1964). It's a fun derivation, as it's almost entirely geometric, and yet reveals bizarre facts about reality.

The Lorentz transformation for changing to a frame that is moving entirely in x with respect to another, can be written as the following matrix:

$$\Lambda^\mu_\nu = \begin{pmatrix} \gamma & -\beta\gamma & 0 & 0 \\ -\beta\gamma & \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.6}$$

where $\gamma \equiv \frac{1}{\sqrt{1-v_x^2/c^2}}$ for relative speed v_x, and $\beta \equiv v_x/c$.

The Lorentz transformation can be thought of as a change in basis. One reference frame's proper time and position gives one basis, and by combining them as given in (1.6), one can transform into any other inertial reference frame. This is to say, if I have a 4-position of a particle x_μ in reference frame A, and I know reference frame B is moving with some speed v, I can get the 4-position B would measure (call it x'_ν) by multiplying by the Lorentz factor:

$$x'_\nu = \Lambda^\mu_\nu x_\mu. \tag{1.7}$$

Finally, an important tool in special and general relativity is the spacetime diagram, also known as the **Minkowski diagram**. Minkowski diagrams are methods of plotting trajectories in spacetime. You plot on the vertical axis the time of a particle (really ct: the speed of light gives time units of position) and the particle's position on the horizontal axis (fig. 1.2). As you can see, the path of a beam of light is the 45-degree line. Any trajectory above that line is called "time-like", and any trajectory below it is called "space-like". Trajectories which follow those of light are called "light-like" or "null". Where a trajectory lies relative to the null trajectory is directly related to the length of its 4-position. Null trajectories have norms equal to zero ($x_\mu x^\mu = 0$) while time-like trajectories are less than zero ($x_\mu x^\mu < 0$). Space-like trajectories are greater than 0, but these are not relevant for this thesis. Time-like trajectories describe the movement of classical particles. A consequence of being less than zero is that they can be causally connected, and therefore describe the motion of particles we might be interested in for a physical system.

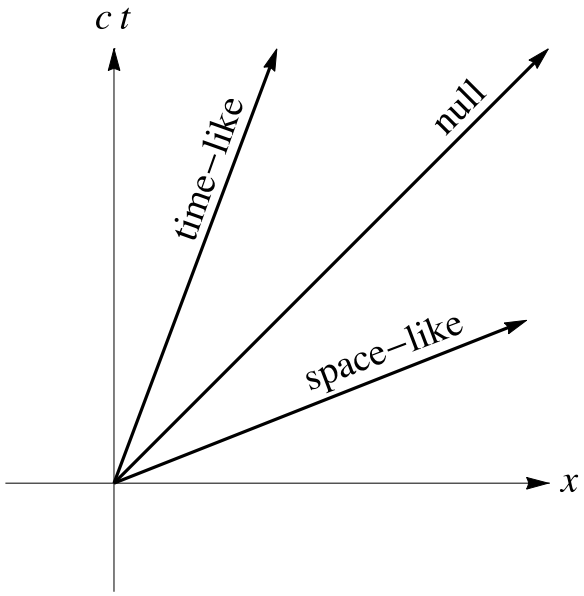


Figure 1.2: A Minkowski, or spacetime, diagram.

1.2 The math of GR: Some differential geometry

This will be a very cursory look at the ideas and mathematical concepts developed by general relativity and used in this thesis. For a completely thorough introduction to general relativity, *Gravitation* by Misner, Thorne, and Wheeler (1973) is the classic text, though I personally enjoy *Einstein's General Theory of Relativity* by Grøn & Hervik (2007).

My brief excursion into the math of general relativity will depend on two ideas which I present as axiomatic.

1. *Mass and energy change the distance between points in spacetime.*

2. *Free particles move along the shortest possible paths between these points.*

The following two subsections will look at what this means mathematically and explore some of the more relevant implications.

1.2.1 Metrics

Suppose you're given a line in a coordinate space. How would you go about measuring the length of that line? I suppose the most obvious way is to simply take a ruler and look at how long it is. But if I have specified the coordinates of the line, that's not necessary, as a dot product will get the job done. For a Cartesian line x_j, the length s is given by:

$$s = \sqrt{x_j x^j} \tag{1.8}$$

This is just the Pythagorean formula. But it's not always this simple. Suppose I have another line which lies on the surface of a sphere (fig 1.3) of radius 2.

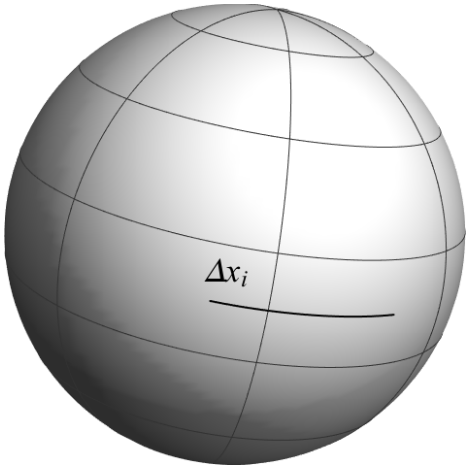


Figure 1.3: A line lying on the surface of a sphere has a different length than on flat ground.

To make it simple, let's say that it's laying along the equator, and takes up an eighth of a circle of arc: $x_{j1} = (2, 0, 0)$, $x_{j2} = (2, 0, \pi/4)$, so the line whose length we want along the surface of the sphere is $\Delta x_j = x_{j2} - x_{j1} = (0, 0, \pi/4)$. If we just naively use (1.8) to calculate the length of Δx_j we get that $s = \pi/4$. But this can't be right, for two reasons: First, $\pi/4$ is a unitless quantity. Second, we know what the answer *should* be, because this is just an eighth of a circumference of a circle of radius 2. So it *should* be $\frac{1}{8}2\pi r = \pi/2$.

So it's clear that we're missing something. Somehow, the length of a vector is hugely dependent upon the coordinates of the space in which that vector lives. This dependence is represented by the **metric tensor**. The metric tensor, or simply the metric, is a second-rank tensor which defines length in a coordinate space. The metric tells you how much of each component is responsible for how much of the length of the vector. For Cartesian "flat" coordinate space, the length is just the dot product:

$$s^2 = x_j x^j = x_1^2 + x_2^2 + x_3^2.$$

This can be written as a multiplication of the vector with an identity matrix:

$$s^2 = x^j g_{ij} x^i \tag{1.9}$$

where g_{ij} is given by

$$g_{ij} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

In this case, g_{ij} is the *Cartesian flat-space metric*, and it is implicit in most dot products we take in math and physics classes. It turns out, though, that (1.9) is totally general for any arbitrary metric space. For the surface of a sphere of radius R, the metric is

$$g_{ij} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & R^2 & 0 \\ 0 & 0 & R^2 \sin^2\theta \end{pmatrix}. \tag{1.10}$$

Then, when we insert this into (1.9), remembering that θ is measured from the north pole and not the equator,

$$g_{ij}|_{R=2,\theta=\frac{\pi}{2}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{pmatrix} \tag{1.11}$$

$$s^2 = \Delta x^j g_{ij} \Delta x^i = \left(\frac{\pi}{4}\right) 4 \left(\frac{\pi}{4}\right) \tag{1.12}$$

$$= \frac{\pi^2}{4}, \tag{1.13}$$

So, the length is $\sqrt{\frac{\pi^2}{4}}$, or $\pi/2$, as desired.

One final note. The above example is vastly simplified, because the metric does not change as the line progresses over the surface of the sphere. But what if there were some more complicated metric which was position dependent, and the path were some arbitrary line? In that case, (1.8) generalizes to

$$s = \int_{x_1}^{x_2} \sqrt{\frac{dx^j}{d\tau} g_{ij} \frac{dx^i}{d\tau}} d\tau, \tag{1.14}$$

where the path has now been parameterized by some affine parameter τ. Equation (1.14) is the definition of arclength for an arbitrary curve in an arbitrary metric.

1.2.2 Mass and energy change distances in spacetime

We are now equipped to define more rigorously what the first axiom I suggested above means. "Mass and energy change the distance between points in spacetime" really means that mass and energy *induce* a metric on spacetime.

In a region where there is no mass or energy, the metric takes the form of the **flat Minkowski metric** which in (t, x, y, z) Cartesian spacetime coordinates looks like

$$\eta_{\mu\nu} \equiv \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{1.15}$$

where the -1 in the (0,0) component enforces the required difference in sign between the temporal and spatial components of a 4-vector discussed above.

But when some mass distribution is introduced, spacetime "warps" so that distances near the mass grow, and time slows down. For instance, a non-spinning spherical mass M induces the so-called **Schwarzschild metric,**

$$g_{\mu\nu} = \begin{pmatrix} -\left(1 - \frac{2GM}{c^2 r}\right) & 0 & 0 & 0 \\ 0 & \left(1 - \frac{2GM}{c^2 r}\right)^{-1} & 0 & 0 \\ 0 & 0 & r^2 & 0 \\ 0 & 0 & 0 & r^2 \sin^2\theta \end{pmatrix} \tag{1.16}$$

General relativity, then, suggests that gravity works in a fundamentally different way than the standard Newtonian story with forces and potentials. Rather than an active force which pushes particles along trajectories, the shape of the spacetime (as determined by the metric) demands that the particles follow specific paths.

I'll look more intently at how particles move in curved spacetimes in the next section, but right now I want to give some intuition (in lieu of any formal argument) about the form of the metric and why the Schwarzschild metric and others make "sense".

Starting with the Schwarzschild metric (1.16), let's compare it to a classical analog. The Schwarzschild metric describes the field outside of a static, uncharged, spherically symmetric mass. This is the standard field studied in Newtonian gravity, which has a potential of

$$\phi(r) = -\frac{GM}{r}. \tag{1.17}$$

$\phi(r)$ is obviously rotationally invariant. If you rotate your reference frame by any angle in any direction, it keeps the same form. The Schwarzschild metric has the same property: Rotate by any angle, and the metric keeps the same form. In particular, there are no off-diagonal elements to the Schwarzschild metric. Every component of a line element contributes only to itself.

Furthermore, in various limits, the Schwarzschild reduces to flat space. Both far from the source, ($r \to \infty$), and when the mass of the central body vanishes ($M \to 0$), the Schwarzschild metric reduces to the Minkowski metric in spherical coordinates.

We can also look at giving some intuition to the metric which is the subject of this thesis, the **Kerr metric**. The Kerr metric describes the spacetime outside a spherically symmetric, spinning mass. In Boyer-Lindquist coordinates, which have the nice property of looking and behaving like spherical coordinates, the Kerr metric has the form

$$g_{\mu\nu} = \begin{pmatrix} -\left(1 - \frac{2Mr}{\Sigma}\right) & 0 & 0 & -\frac{2aMr\sin^2\theta}{\Sigma} \\ 0 & \frac{\Sigma}{\Delta} & 0 & 0 \\ 0 & 0 & \Sigma & 0 \\ -\frac{2aMr\sin^2\theta}{\Sigma} & 0 & 0 & \left(r^2 + a^2 + \frac{2a^2 Mr\sin^2\theta}{\Sigma}\right)\sin^2\theta \end{pmatrix}, \tag{1.18}$$

where

$$\Delta \equiv r^2 - 2Mr + a^2$$
$$\Sigma \equiv r^2 + a^2\cos^2\theta.$$

Notice that in the limit as $a \to 0$, we recover the Schwarzschild metric. The Kerr metric is largely like the Schwarzschild metric, but with two new additions: a, which

is a parameter encoding spin which has units of length, and the off-diagonal elements in the (0,3) and (3,0) spots. This last property is kind of weird, and responsible for really strange behavior of orbiting particles, which I explore later in this thesis. We can still get some kind of intuitive grasp for their existence, however.

The answer starts with one of the motivating assumptions Einstein made when deriving the theory of general relativity. A few years prior, he had shown famously that

$$E = \sqrt{m^2c^4 + p^2c^2}, \tag{1.19}$$

for a particle of rest mass m possessing momentum p. For a static particle, this equates mass with an intrinsic energy so he reasoned that to say that mass is a source for gravity requires also that energy is a source for gravity. This is kind of bizarre in a classical picture– Energy is far less tangible (in a human-intuitive sense) than gravity, and yet apparently it can effect something in exactly the same way as hard rock. Furthermore, if a particle is moving, not only will it have the energy from its rest mass, but there's a contribution to the total energy from the particle's momentum! Apparently, this will change the nature of the particle's gravitational field!

GR deals with this by saying that the metric is proportional to an object called the **stress-energy tensor**. The stress-energy tensor is a second-rank tensor which describes how energy and momentum of an object are moving through space. Each component μ, ν of the stress-energy tensor describes how the momentum of the μ^{th} component of energy/momentum moves through the surface defined by the ν^{th}.

In the case of the spinning spherical mass, the spinning-ness will be reflected in the stress-energy tensor by motion through the 0^{th} surface (time) of the 3^{rd} component of momentum, (the ϕ in Boyer-Lindquist coordinates). Thus, in the mere motion of the spherical mass, there is a way to predict the arcane form of the metric it produces.

Free particles move along shortest paths

The other fact I proposed as axiom was the fact that, given a metric, particles not subject to external potentials move along the shortest paths between points in that metric. In a classical setting, the way to get equations of motions for particles is to minimize their Lagrangian, which is conventionally defined as the difference between the kinetic and potential energies. Looking at the free particle case,

$$L = \frac{1}{2}m\mathbf{v}^2 = \frac{1}{2}m\dot{\mathbf{x}}^2 \tag{1.20}$$

$$= \frac{1}{2}m\dot{\mathbf{x}} \cdot \dot{\mathbf{x}}, \tag{1.21}$$

which should look very familiar. (1.21) is really (1.14), the definition of length of an arbitrary line in an arbitrary metric. Indeed, the former is the square of the latter. Interesting! That means that when you extremize the Lagrangian of a free particle, you're really extremizing the square of the length of the particle's position vector, parametrized by time.

So, if we're really just talking about length, we can generalize the free particle Lagrangian in the exact way we've been discussing:

$$L = \frac{1}{2} m \dot{x}^\mu g_{\mu\nu} \dot{x}^\nu, \tag{1.22}$$

which, when extremized, returns precisely the equations of **geodesic motion**.

To show this, we can parameterize with respect to an affine parameter τ, and the resulting Euler-Lagrange equation for (1.22) reads

$$\frac{d}{d\tau}\frac{\partial L}{\partial \dot{x}^\mu} - \frac{\partial L}{\partial x^\mu} = 0. \tag{1.23}$$

Following through with the calculations, we arrive at

$$\frac{d^2 x^\alpha}{d\tau^2} + \Gamma^\alpha_{\mu\nu} \dot{x}^\mu \dot{x}^\nu = 0. \tag{1.24}$$

$\Gamma^\gamma_{\mu\nu}$ is the *Christoffel connection*, given by

$$g_{\beta\gamma}\Gamma^\gamma_{\mu\nu} = \frac{1}{2}\left(\frac{\partial g_{\beta\mu}}{\partial x^\nu} + \frac{\partial g_{\beta\nu}}{\partial x^\mu} - \frac{\partial g_{\mu\nu}}{\partial x^\beta}\right), \tag{1.25}$$

which gives a description of how the coordinates change with the metric.

(1.24) is the definition of geodesic motion. For the context of this thesis, a geodesic is defined locally as the shortest possible path between two points (but not generally). In flat space, a geodesic is a straight line, which makes complete sense: Free particles move in straight lines.

Given more complicated metrics, geodesic motion will be more complicated, but it will always minimize the total path length. In the case of the Schwarzschild metric, this minimization conspires to produce elliptical motion about the center.

Note that this is totally different from what Newtonian gravity says. In that scenario, there is no "free" particle– all the particles are being shoved around by whatever gravitational potential they find themselves in. In this case, there is no potential whatsoever. Instead, the mass and energy in a system change the *geometry* of the spacetime itself. The subsequent geometry, then, only allows certain types of motion (called geodesic motion) for an otherwise passive particle.

Finally, the notion of path length is related to the motions of massless and massive particles from earlier. Recall that the definition of a null 4-position vector said that the total length, $s^2 = x^2 + y^2 + z^2 - c^2t^2 = 0$. Well, the same is still true here. The Lagrangian for the motion of a photon is

$$L_{photon} = \dot{x}^\mu g_{\mu\nu} \dot{x}^\nu = 0. \tag{1.26}$$

Similarly, a massive particle has a 4-position with norm is always less than 0, and whose 4-velocity norm is always equal to $-c^2$. Therefore,

$$L_{massive} = \dot{x}^\mu g_{\mu\nu} \dot{x}^\nu = -c^2. \tag{1.27}$$

These facts are extremely helpful later in deriving the exact forms of the equations of motion for particles in curved spacetimes.

Chapter 2

The Kerr Spacetime

This chapter will discuss the features of the Kerr spacetime salient to this thesis. For a more complete treatment, I highly recommend Visser (2008), which covers not only what is discussed here, but goes over some of the metric's more interesting mathematical properties.

Preliminaries

The Kerr metric describes the spacetime around a spinning mass. The Kerr metric, as used in this thesis, is written in Boyer-Lindquist coordinates, and takes the form, for mass M and angular momentum per unit mass a, and units where $G = c = 1$,

$$g_{\mu\nu} = \begin{pmatrix} -\left(1 - \frac{2Mr}{\Sigma}\right) & 0 & 0 & -\frac{2aMr\sin^2\theta}{\Sigma} \\ 0 & \frac{\Sigma}{\Delta} & 0 & 0 \\ 0 & 0 & \Sigma & 0 \\ -\frac{2aMr\sin^2\theta}{\Sigma} & 0 & 0 & \left(r^2 + a^2 + \frac{2a^2Mr\sin^2\theta}{\Sigma}\right)\sin^2\theta \end{pmatrix}, \tag{2.1}$$

where

$$\Delta \equiv r^2 - 2Mr + a^2,$$
$$\Sigma \equiv r^2 + a^2\cos^2\theta.$$

It is sufficient (and recommended) to think of Boyer-Lindquist coordinates as spherical coordinates which have a component for time. Note that these are the physicist's spherical coordinates, and so θ measures the polar angle and goes from 0 to π, while ϕ is the azimuthal angle, ranging from 0 to 2π, and the spatial components of the vector are (r, θ, ϕ) (fig. 2.1).

2.1 Features of the Kerr Spacetime

2.1.1 Event horizon

Immediately you can see that there are values for which components of the metric are undefined. These occur when either Σ or Δ are equal to 0. When Σ equals 0,

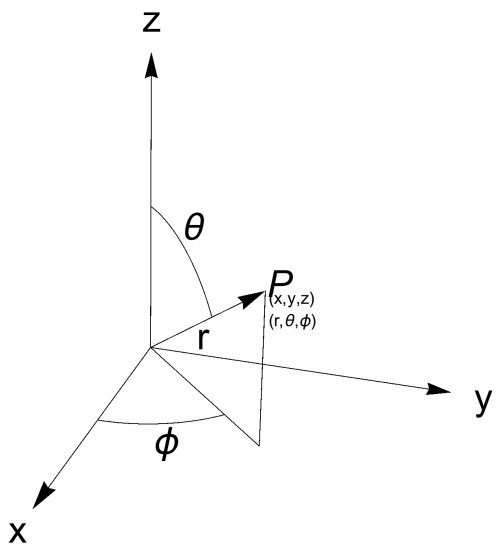


Figure 2.1: The spatial components of the coordinate system assumed for this section.

$$r^2 + a^2 \cos^2 \theta = 0,$$

which is satisfied only when $r = 0$ and $\theta = \pi/2$. This corresponds to the center of the black hole, a which is not a region of interest for us. More interesting is the condition when Δ vanishes:

$$\begin{aligned} r^2 - 2Mr + a^2 &= 0, \\ &\Rightarrow r = M \pm \sqrt{M^2 - a^2}. \end{aligned}$$

The positive root corresponds to the event horizon of a spinning black hole. The negative root, being within the event horizon, is of no interest to a physical simulation, but it is strictly true that a spinning black hole has two event horizons. Note that in the limit as $a \to 0$, this reduces to the standard Schwarzschild radius of $2M$ familiar from static black holes. Looking at the behavior of the metric as the spin goes to 0 is a handy tactic for gaining some instinct about Kerr because the behavior of a Schwarzschild spacetime is much more intuitive.

2.1.2 Ergosphere

A key feature of the Kerr metric is the off-diagonal (0,3) and (3,0) elements coupling the ϕ and t components. This property of the metric demands that any motion in ϕ must also be accompanied by some kind of motion in time. For a more concrete

notion of what this means, consider a coordinate observation of a spaceship outside a Kerr black hole, firing the thrusters such that they remain at rest. If they're at rest according to an observer at infinity (the coordinate reference frame), then the spaceship has a 4-position of $(t, r_0, \theta_0, \phi_0)$. Their 4-velocity, parameterized by coordinate time, is then

$$\dot{x}^\mu = \frac{dx^\mu}{dt} = (1, 0, 0, 0). \tag{2.2}$$

This is to say that all of their motion in the coordinate reference frame is through time, and none of it through space. The length of this tangent vector given a Kerr metric is

$$\dot{x}^\mu g_{\mu\nu} \dot{x}^\nu = -\left(1 - \frac{2Mr}{r^2 + a^2 \cos^2\theta}\right), \tag{2.3}$$

where dots indicate derivatives with respect to coordinate time. Now, in order that this trajectory describe a physical massive particle, it must be less than 0. However, this condition places a requirement on r:

$$-\left(1 - \frac{2Mr}{r^2 + a^2 \cos^2\theta}\right) < 0 \tag{2.4}$$

$$\Rightarrow r > M + \sqrt{M^2 - a^2 \cos^2\theta} \equiv r_{ES}. \tag{2.5}$$

If $r < r_{ES}$, then it is impossible for an observer to maintain (2.2) as their world line, and so it is impossible for them to remain stopped. This requirement on r defines the boundary of the **ergosphere**, which is a bizarre region unique to the Kerr metric. Within the ergosphere, all particles, including light, are compelled to move in the direction the black hole is rotating. The ergospheres and event horizons are plotted for different values of a in fig. 2.2.

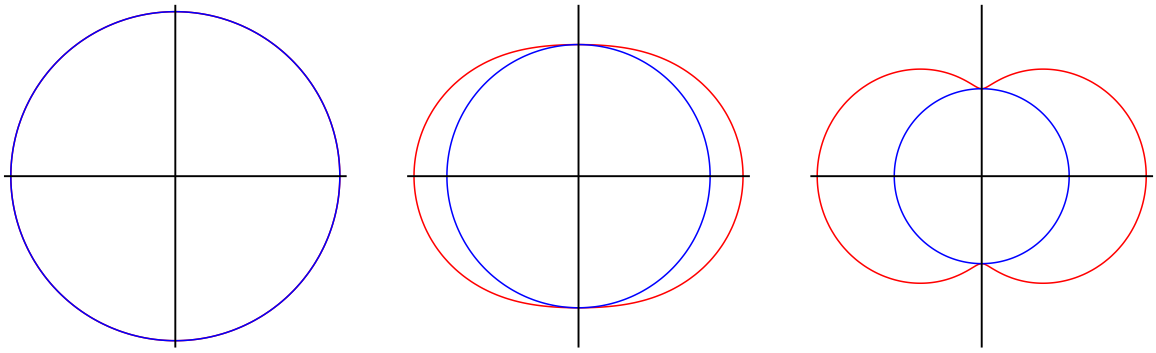


Figure 2.2: The structure of Kerr spacetime for three different values of the a spin parameter. The ergosphere boundary is in red and the event horizon is in blue. On the left, $a = 0$ and we recover the structure of Schwarzschild spacetime. The middle picture has $a = 0.8$ and the right has $a = 0.998$. All of these plots show slices in ϕ. The vertical axis is z and the horizontal is x or y.

To see that they must move in the same direction, consider a similar scenario as above, except now the ship is free to move in ϕ and we parameterize with respect to proper time, τ: $\dot{x}^\mu = (\dot{t}, 0, 0, \dot{\phi})$, where dots now refer to derivatives with respect to proper time.[1]

The norm of this 4-velocity now gives

$$g_{tt}(\dot{x}^t)^2 + 2g_{t\phi}\dot{x}^t\dot{x}^\phi + g_{\phi\phi}(\dot{x}^\phi)^2 = -1, \tag{2.6}$$

where we are explicitly enforcing that the 4-velocity be timelike.

We can divide through by $(\dot{x}^t)^2$ and, defining the time-derivative of the coordinate ϕ,

$$\Phi \equiv \frac{d\dot{x}^\phi}{d\dot{x}^t} = \frac{d\phi}{dt}, \tag{2.7}$$

we get that

$$g_{tt} + 2\Phi g_{t\phi} + g_{\phi\phi}\Phi^2 = \frac{1}{(\dot{x}^t)^2}. \tag{2.8}$$

As a particle approaches the ergosphere, $\dot{x}^t$ gets very large[2], so (2.8) simplifies to a quadratic equation,

$$g_{tt} + 2\Phi g_{t\phi} + g_{\phi\phi}\Phi^2 = 0. \tag{2.9}$$

Finally, solving for Φ, we get

$$\Phi = \frac{-g_{t\phi} \pm \sqrt{g_{t\phi}^2 - g_{tt}g_{\phi\phi}}}{g_{\phi\phi}}, \tag{2.10}$$

which gives us the bounds of how fast the observer must move within the ergosphere. Notice that at the ergosphere boundary, $r = r_{ES}$ and $g_{tt} = 0$. Then the lower bound on Φ is 0, which confirms that we can sit still outside the ergosphere. Whereas the lower bound says that nothing can remain stopped in ϕ, the upper bound says that nothing can move faster than c[3].

2.2 Time-like geodesics: How massive particles move

I described in the GR chapter how one can derive geodesic equations of motion from extremizing the Lagrangian, which was just the line element,

$$L = \dot{x}^\mu g_{\mu\nu}\dot{x}^\nu. \tag{2.11}$$

[1] We choose this toy model because we are interested in how the metric forces motion in ϕ, and the only components of the metric which have any influence over ϕ are those associated with t and ϕ itself.

[2] To see why, consider the Schwarzschild case. It is identical to the Kerr case, but here the event horizon plays the role of the ergosphere. (Indeed, this story is identical for the equatorial Kerr case, where the ergosphere is maximized with a radius of $2M$). For a particle moving only in time, $\dot{x}^\mu g_{\mu\nu}\dot{x}^\nu = -(\dot{x}^t)^2(1 - 2M/r) = -1 \Rightarrow (\dot{x}^t)^2 = (1 - 2M/r)^{-1}$, which goes to infinity as $r \to 2M$.

[3] For an excellent discussion of this, see Visser (2008).

Here I will sketch out how the derivation goes about, without doing very much of the actual work. It should be clear from (2.1) that these equations are not fun to work with. Showing the work would be tedious without being illuminating.

2.2.1 Conserved quantities

We can significantly reduce our workload and give the equations of motion simpler forms by first identifying the conserved quantities of the metric. As with the classical case, we can use the energy and angular momentum as constants of the motion. In Kerr, they become

$$E = -\frac{\partial L}{\partial \dot{t}} = \left(\frac{2aMr\sin^2\theta}{\Sigma}\right)\dot{\phi} + (1 - \frac{2Mr}{\Sigma})\dot{t} \tag{2.12}$$

$$J_z = \frac{\sin^2(\theta)\left(\dot{\phi}\left(2a^4\cos^2(\theta) + 4a^2Mr\sin^2(\theta) + a^2r^2(\cos(2\theta)+3) + 2r^4\right) - 4aMr\dot{t}\right)}{a^2\cos^2(\theta) + r^2} \tag{2.13}$$

Note that in the stationary limit, $a \to 0$, $\Sigma \to r$, and these exactly match the constants from the Schwarzschild case.

In Schwarzschild geometry, these are the conserved quantities associated with the symmetries of the system. However, there is one more quantity for Kerr which arises because Kerr breaks the spherical symmetry of Schwarzschild. While it is symmetric in ϕ, Kerr is not symmetric in θ. This gives the spacetime a "preferred" direction: The frame dragging forces in ϕ are all pushing in ϕ, so we cannot change in θ as we could in Schwarzschild. Doing so would change the direction of the frame-dragging.

Therefore, the field itself has a symmetry, and this gives rise to Carter's constant, given by[4]

$$C = \dot{\theta}^2 + \cos^2\theta\left(a^2(m^2 - E^2) + \left(\frac{J_z}{sin\theta}\right)^2\right). \tag{2.14}$$

These conserved quantities can be used to then vastly simplify the equations of motion, and are helpful in analysis. The equations of motion are[5]

$$\dot{r} = \frac{\left((E(r^2+a^2) - aJ_z^2)^2 - \Delta\left(r^2 + C + (J_z - aE)^2\right)\right)^{1/2}}{\Sigma} \tag{2.15}$$

$$\dot{\theta} = \frac{\left(C - \cos^2\theta\left(a^2(1-E^2) + \frac{J_z^2}{\sin^2\theta}\right)\right)^{1/2}}{\Sigma} \tag{2.16}$$

$$\dot{\phi} = \frac{\frac{J_z}{\sin^2\theta} - aE + \frac{a}{\Delta}(E(r^2+a^2) - aJ_z^2)}{\Sigma} \tag{2.17}$$

[4]Chandrasekhar (1976)
[5]Chandrasekhar (1976)

All of the simulations work in coordinate time, so $\dot{t} = 1$.

However, it should be noted that I specifically *don't* use these forms with the conserved quantities in my code, because I am looking at interactions between particles, namely collisions and Newtonian gravitational interactions. This means that any single particle definitely will not conserve these quantities.

2.3 Comparison with Newtonian Gravity

A comparison with Newtonian gravity is helpful for two reasons. First, I use Newtonian gravity for the particle-particle interactions in my simulation, and it is helpful to see how it's similar to the GR story just described above. Secondly, the GR version of particle motion is pretty complicated and a lot of the gist of it gets lost. However, a lot of the same actors appear in both contexts, and it can help form our notion of GR particle motion to remind ourselves of what happens classically.

2.3.1 Newtonian gravity

Invented in the 1600's to describe mathematically the motions of both planets and apples, Newton's theory of gravity is still used extensively because it's really accurate for anything we might encounter. The main premise of Newtonian gravity from which you can derive planetary motion (among many other phenomena) can be stated like this.

- *A body in space induces an attractive force on another. The force is proportional to the masses of the bodies and inversely proportional to the distance between the bodies.*

Mathematically, you would write this as

$$\mathbf{F} = -\frac{Gm_1m_2}{r^2}\hat{\mathbf{r}}, \tag{2.18}$$

where m_1 and m_2 are the masses of the two bodies, and r is the distance between them. G is called the *gravitational constant*, and it is entirely experimentally derived. $\hat{\mathbf{r}}$ is a vector that points from one body to the other. There is also a minus sign, and this tells us that the force is attractive.

(2.18) is called *Newton's law of universal gravitation*, and it turns out that it is totally sufficient to derive a wide range of phenomena. In particular, it is a common exercise for second- or third-year physics students to derive the equations of planetary motion starting from the law of universal gravitation. The result is that planets orbit in ellipses (fig. 2.3).

This was an early triumph of the theory; Newton was able to totally explain Tycho Brahe's data of the orbits of the planets with this result[6].

[6] Zee (1986)

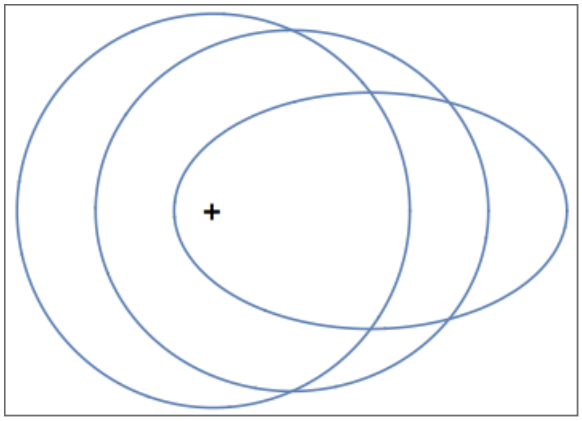


Figure 2.3: The orbits of three different bodies of varying ellipticity around some central body (the plus sign in the middle). This could represent planets orbiting a star, or moons orbiting a planet, or a combination of different types of bodies in the orbit of something much more massive.

2.3.2 Equations of Motion in Newtonian Gravity

However, (2.18) is not the best form of Newtonian gravity to use to analyze the equations of motion. It would be better to write the force of gravity as a potential. This would allow us to stick it into a Lagrangian, and use the Euler-Langrange equations to not only derive the equations of motion, but identify any conserved quantities. These are important because we employ the exact same techniques when we're looking how particles move due to general relativity, and it will be instructive to compare the two theories.

Lagrangian analysis of Newtonian gravity

Starting from (2.18), we can get a potential formulation by integrating along a radial path:

$$Gm_1m_2 \int -\frac{1}{r^2} \cdot d\mathbf{r} = G\frac{m_1m_2}{r} \equiv -\phi(r). \tag{2.19}$$

Where we have given the potential the name $\phi(r)$. Then we can plop this into a Lagrangian and go wild with the calculating.

Recalling the definition of a Lagrangian, $L = T - U$, for kinetic energy T and potential U, the relevant Lagrangian is

$$L = T - \phi(r) = \frac{1}{2}m_1\left(\dot{r}^2 + r^2\dot{\theta}^2\right) - G\frac{m_1m_2}{r}. \tag{2.20}$$

(2.20) describes a the motion of a particle of mass m_1 interacting gravitationally with another particle of mass m_2 (fig. 2.4). An implicit assumption in this is that r is dependent only upon the motion of m_1. This is valid if $m_2 \gg m_1$. We have written the kinetic energy term in polar coordinates because the potential is most naturally written in polar coordinates.

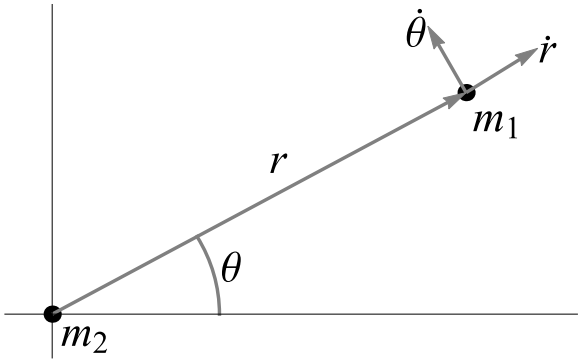


Figure 2.4: A two-body system for which (2.20) is an appropriate description.

Immediately, we see that (2.20) does not depend on θ. This implies a conserved quantity. Employing the Euler-Lagrange equations,

$$\frac{\partial L}{\partial \theta} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) = 0 \tag{2.21}$$

$$0 - \frac{d}{dt} m_1 r^2 \dot{\theta} = 0 \tag{2.22}$$

$$\Rightarrow m_1 r^2 \dot{\theta} = const. \equiv J_z \tag{2.23}$$

$$\Rightarrow \dot{\theta} = \frac{J_z}{m_1 r^2}. \tag{2.24}$$

Where we have identified the angular momentum, J_z. We can use this to rewrite the original Lagrangian, (2.20), with a single variable, r:

$$L = \frac{1}{2} m_1 \left(\dot{r}^2 + \frac{J_z^2}{m_1^2 r^2}\right) - G\frac{m_1 m_2}{r}. \tag{2.25}$$

Now, with (2.25), we can get the actual equation of motion we seek.

$$\frac{\partial L}{\partial r} = -\frac{J_z^2}{m_1 r^3} + G\frac{m_1 m_2}{r^2} \tag{2.26}$$

$$\frac{\partial L}{\partial \dot{r}} = m_1 \dot{r} \tag{2.27}$$

$$\Rightarrow \frac{d}{dt}(m_1 \dot{r}) = -\frac{J_z^2}{m_1 r^3} + G\frac{m_1 m_2}{r^2} \tag{2.28}$$

$$\Rightarrow \ddot{r} = -\frac{J_z^2}{m_1^2 r^3} + G\frac{m_2}{r^2}. \tag{2.29}$$

I find this a helpful reminder of why we care about conserved quantities and where they come from. It is also a neat demonstration that a lot of the machinery of GR is already familiar from classical mechanics. Indeed, in the two-dimensional case where $\theta = \pi/2, \dot{\theta} = 0$, Carter's constant doesn't even appear, and the only conserved quantities for the Kerr metric are exactly those we encounter in the classical problem.

Chapter 3

Numerical Methods

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.

3.1 The N-Body Problem

Presented with two bodies which interact 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[1] century and follows 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. 3.1a). 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.

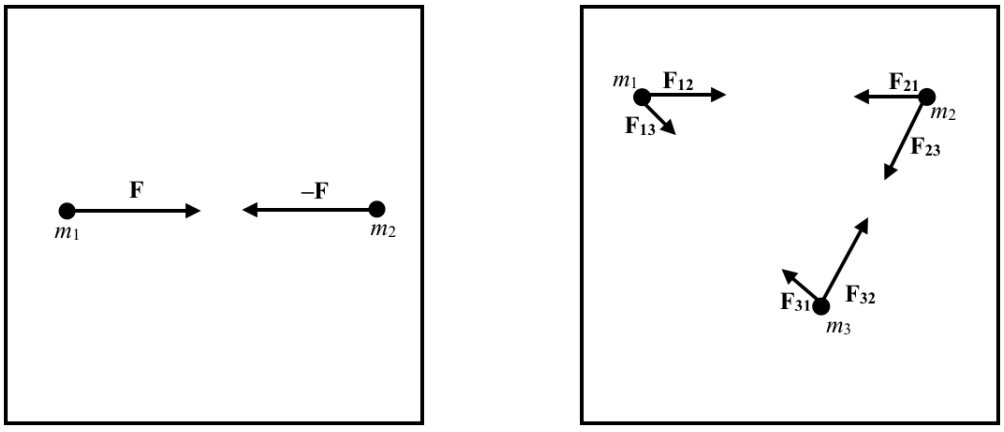
As I said, the problem for the gravitational interaction between two bodies is totally solved. However, if I introduce a third body (Fig. 3.1b), the story is made murky: Now each of the planets or stars or whatever is being pulled simultaneously by *two* other bodies. The force on any one body at a given instant is described by the sum of the forces due to the other two bodies. For instance, the force on the first

[1] Hut & Makino (2007)

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.



(a) (b)

Figure 3.1: 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, except for a small number of specific cases[2]. 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.*

3.1.1 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

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

[2]Aarseth (2003)

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

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{v}_0 \Delta t \tag{3.1}$$

and

$$\mathbf{v}_1 = \mathbf{v}_0 + \mathbf{a}\Delta t. \tag{3.2}$$

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 3.1 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.

Notice that for every single body in every time step, the distance to every other body must be calculated. This means that every time step requires a calculation on every possible pair of bodies. Therefore, the time it takes to compute a single time step will increase with the square of the bodies. For this reason, the n-body problem is solved in what's called n^2 time. This means that the time required increases dramatically with the addition of each new body. There are several techniques[3] to mitigate this limitation, however, and this thesis will employ some of the simpler ones.

Therefore, if a particle starts out at $\mathbf{x}_0$ with speed $\mathbf{v}_0$, we can say to arbitrary accuracy where it will be after N steps by just adding up all the steps:

$$\mathbf{x}_N = \sum_n^N \mathbf{x}_n + \mathbf{v}_{N-1}\Delta t. \tag{3.3}$$

As Δt becomes infinitesimally small, equation (3.3) becomes an integral over the full time period, $t = 0$ to $t = t_f$:

$$\mathbf{x}_N = \int_0^{t_f} \mathbf{v}(t) dt, \tag{3.4}$$

where dt is an infinitesimal unit of time. Thus, this is discretely solving an integral. For this reason, it is called *numerical integration*.

The whole process of numerical integration is presented as a flowchart in Fig. 3.2. This approximation technique is how we can probe the behavior of n-body

[3] see Hut & Makino (2007)

systems. Of course, this can be 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.

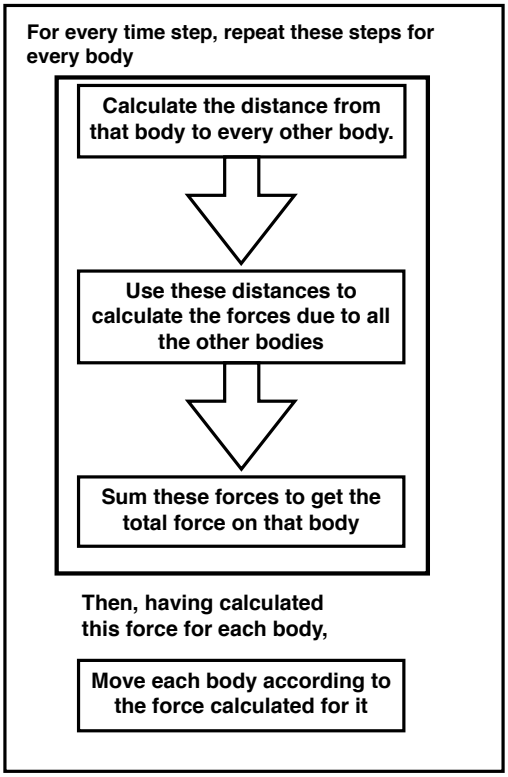


Figure 3.2: The algorithm for time-stepping an n-body system.

3.2 Numerical Calculation in this Thesis

This thesis uses a fourth-order Runge-Kutta method to solve for the trajectories of particles whose equations of motion are given by the Kerr line element. What follows is a description of the method and its specific implementation in the code used for analysis.

3.2.1 Fourth-Order Runge-Kutta

The fourth-order Runge-Kutta (RK4) method is a numerical integration method frequently used in temporal discretization problems.

Let an initial value problem be specified like so:

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}) \tag{3.5}$$

and

$$\mathbf{x}(t_0) = \mathbf{x}_0. \tag{3.6}$$

Where $\mathbf{x}$ represents the (vector) function of time (position, in our case) we want to integrate. In this example, the time-derivative, $\dot{\mathbf{x}}$ be a function of t and $\mathbf{x}$, but it doesn't have to be. The value of the function $\mathbf{x}$ at t_0 is $\mathbf{x_0}$, and both this initial value and the initial time t_0 are given.

RK4, then works like this:
For a given step size dt, the n^{th} steps in position and time are given by

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{dt}{3}\left(\frac{\mathbf{k}_1}{2} + \mathbf{k}_2 + \mathbf{k}_3 + \frac{\mathbf{k}_4}{2}\right), \tag{3.7}$$

and

$$t_{n+1} = t_n + dt. \tag{3.8}$$

The $\mathbf{k}_{1...4}$ are given by:

$$\mathbf{k}_1 = \mathbf{f}(t_n, \mathbf{x}_n), \tag{3.9}$$

$$\mathbf{k}_2 = \mathbf{f}(t_n + \frac{dt}{2}, \mathbf{x}_n + \frac{dt}{2}\mathbf{k}_1), \tag{3.10}$$

$$\mathbf{k}_3 = \mathbf{f}(t_n + \frac{dt}{2}, \mathbf{x}_n + \frac{dt}{2}\mathbf{k}_2), \text{and} \tag{3.11}$$

$$\mathbf{k}_4 = \mathbf{f}(t_n + dt, \mathbf{x}_n + dt\mathbf{k}_3). \tag{3.12}$$

So the $\mathbf{k}_{1...4}$ can be thought of as interstitial time-steps, each using a different method, which are all then combined in some kind of fancy average in (3.7). For the example, it is clear that $\mathbf{k}_1$ is just a straight-forward Euler's method, as it's using information about the slope at the beginning of the interval to update the vector. Likewise, $\mathbf{k}_2$ and $\mathbf{k}_3$ are midpoint methods. To understand exactly why the various

methods are combined like they are in (3.7) requires a full derivation, which can be found in Franklin (2013).

RK4 is well-suited to an adaptive step size technique. However, adaptive step sizing is almost useless for n-body problems. That one particle is experiencing little change time step-to-time step says nothing about what another particle might be doing.

3.2.2 The Equations of Motion

As I derived earlier, equations of motion generated by solving for geodesics in Kerr spacetime are unwieldy and totally unhelpful in forming any sort of intuition. Therefore, for the current discussion I adopt the subscript q_K when referring to coordinate q's equation of motion due to Kerr.

In addition to the acceleration due to geodesic motion, I am probing the accelerations induced by gravitational and collisional interactions with other particles. I include these in the equations of motion by adding to the $\ddot{q}_K$ a $\ddot{q}_{ext.}$ which the the sum of the external forces. Therefore, the derivative vector I use for updating the state vector in the integration (the $\dot{\mathbf{x}}$ in (3.5)) is

$$\mathbf{f} = \begin{pmatrix} \dot{r} \\ \ddot{r}_K + \ddot{r}_{ext.} \\ \dot{\theta} \\ \ddot{\theta}_K + \ddot{\theta}_{ext.} \\ \dot{\phi} \\ \ddot{\phi}_K + \ddot{\phi}_{ext.} \end{pmatrix}. \tag{3.13}$$

3.3 Orbit Integrator for N-Body Kerr Solutions

The code written for this thesis is the Orbit Integrator for N-Body Kerr Solutions, or OINKS (Tagline: *It's a real resource hog!*).

OINKS is a Python package consisting of an RK4 integrator combined with methods for calculating the different particle-particle interactions considered in this thesis. It uses extensively the Python packages NumPy and SciPy, for array handling and some computational functions, AstroPy for file input and output, Matplotlib for plotting, and the command line video tool FFmpeg. It also depends on a custom set of Python functions included in the `OINKSTools.py` file.

The source is included in the appendix of this thesis and, as of this writing, is available on GitHub at http://www.github.com/deichdeich/thesis/code/oinks. The GitHub also includes full documentation and use cases.

3.3.1 Code Design

OINKS is object-oriented. Every simulation is an instance of the `OrbitalSystem` object, which is initialized with the following parameters:

- `Nparticles`: The number of particles for the simulation.
- `M`: The (unitless) mass of the central body.
- `a`: The Kerr spin parameter.
- `dt`: The time step interval.
- `interaction`: Are the particles interacting or not?
- `collisions`: Are the particles colliding? If they are, are they elastically or inelastically colliding?
- `masses`: An array representing the initial mass of each particle.
- `init_state`: This is optional. If specified, it gives the initial state vector of each particle. If unspecified, the particles are given random stable initial conditions. The random initial conditions are handled by the `make_initial_conditions()` function in `OINKSTools.py`. `make_initial_conditions()` uses a configurable range of initial angular momenta and energy to determine random initial stable circular orbits for the particles. The angular momenta and energy are chosen by NumPy's `random.randn()` Gaussian distribution function to get the radii for stable circular orbits. This is done by setting $\dot{r} = 0$, and solving for r. Then, these radii are perturbed by a small amount (a random amount between 1% and 5% of the magnitude radius) to achieve an initial radius which will give some precession. Then, the angular momentum and energy which were used for that calculation are used to give the appropriate value for $\dot{\phi}$. Finally, the ϕ values were given by a flat random distribution. All the simulations started in an equatorial disk, so no calculations for θ were performed.
- `cr`: The collision radius of each particle.
- `save_dir`: The directory to save the output data to.

So, to call this in a script, the user would do something like

```
import oinks

sim = oinks.OrbitalSystem(Nparticles = 10000, M = 1, dt = 0.1, cr = 0.00001,
                          interaction = False,
                          collisions = 'elastic',
                          save_dir = '~/output/')
sim.TimeEvolve(10000, comments = '10e4 non-interacting bodies')
```

This initializes the simulation object in `sim`. To time evolve the system, you would then call the `TimeEvolve` method. `TimeEvolve` takes as the argument the number of time steps as well as any comments to be included in the output data. Calling `TimeEvolve` starts a big loop which evolves the system according to the equations of

motion for Kerr and checks for any other forces which might occur at any given time step. The loop goes like this.

For every particle in every time step:

1. Calculate the numerical values of the equations of motion. This amounts to plugging in the appropriate values of r, θ, ϕ and their derivatives into the equations derived in chapter 2.

2. If `interaction` is set to `True`, then calculate the Newtonian n-body force due to all the other bodies. Add these values to the equation of motion vector. In the interest of optimization, I use a "sphere-of-influence" technique where I only look at the interaction due to particles within a certain radius. This allows me to perform the n^2 n-body calculation on a small fraction of the total particles in the simulation.

3. If `collisions` is not `False`, then look for any colliding bodies. First, I sort the array. I use NumPy's built-in sorter which is an implementation of the QuickSort algorithm. Sorting the array vastly speeds up the search for overlapping points, which is handled by SciPy's built-in `scipy.pdist` function. `scipy.pdist` takes an array of spatial points and returns an array giving the distances between every pair of points.

4. If `collisions` is `elastic`, then every pair of points in the output from `scipy.pdist` which is less than the collision radius `cr` are collided in a classical momentum- and energy-conserving collision. Here, saying points are "collided", means the velocity components of their state vectors are updated according to the equations of hard elastic collisions. If three or more points are within `cr`, then the closest two points are used.

5. If, on the other hand, `collisions` is `inelastic`, then every pair of points in the output from `scipy.pdist` which is less than the collision radius `cr` are combined into a new particle. In this case, it iterates until there are no more points within the collision radius. Again, they are combined using conservation of momentum and energy. In this case, all the colliding particles are deleted from the state array, and a new particle with aggregate momentum and mass is added to the state array at the site of collision.

6. Finally, each particle is updated according to their equation of motion.

OINKS saves data to disk 1000 time steps at a time. The state array is outputted time step-by-time step to a file in the Flexible Image Transport System (FITS) format. The FITS format is the standard format for astronomical observation data, but it has several features helpful for this context: It has built-in structures for headers and comments, it is very easy to store the data type of each array, and the code available for writing FITS files is faster than, for example, NumPy's CSV module.

When the integration is finished, there are several functions for plotting movies of the data. These use Matplotlib to plot both density heatmaps and spatial scatter

plots of the data frame-by-frame. They then use the command-line tool FFmpeg to compile all the frames into an MP4 video file.

When it works, OINKS is extremely helpful at analyzing the behavior of particles in a Kerr spacetime. Because every simulation is an object, the user can load up previously calculated data in a new instance, and make new plots, or run the integration from a different initial point very easily.

Chapter 4

Physical Relevance of this Simulation

Recall that this thesis is investigating "particle dynamics in a time-dependent Kerr geometry". We now know well what a Kerr spacetime is. But how can it change in time, and what are the actual particles that could be affected? This chapter will look at these questions. I will describe to what extent my simulations are capable of describing physical systems.

4.1 Reasons a Kerr Spacetime could change in time

4.1.1 Stellar Evolution

The end of a massive star's life is spectacular. As the star depletes its nuclear fuel, it is no longer able to support its outer layers from the gravitational pull of the mass within the star. The star then collapses in a series of increasingly dramatic structural failures. First, the intense pressure from the outside layers falling in will force all of the electrons into the lowest possible energy states. By the Pauli exclusion principle, only two electrons can occupy a state at once. Therefore, if a state is filled, the electrons fight back, exerting a force against the pull of gravity. If the star cannot overcome this so-called "electron degeneracy pressure", the collapse stops, and the star becomes a *white dwarf*. This is the fate for stars the size of our sun.

If the star is about an order of magnitude more massive than the sun (written as $10M_\odot$ where $M_\odot$ is the mass of the sun), electron degeneracy pressure will not be enough to create a barrier to gravitational collapse, and the star will push on to undergo a **supernova**. In this case, the electrons are collided into the protons in the nuclei of the atoms. This collision produces neutrons, and soon all the matter in the star is converted into a neutron soup. These objects are for this reason called **neutron stars**. If the resulting neutron star is approximately $5M_\odot$, then it has the potential to push on an become a black hole.

The point is that at every stage mentioned above, the star shrinks significantly while maintaining a large fraction of its mass. Remember, too, that the star neces-

sarily starts out spinning, even if it is spinning very slowly. So, if the star shrinks in radius, and changes its mass, then its angular momentum per mass changes.

But angular momentum per mass is exactly what's encoded in the a parameter in the Kerr metric. Therefore, stellar evolution, and in particular supernovae, constitute a change in the metric.

Supernovae

Supernovae are some of the most complicated astrophysical events in the universe. They're still not fully understood, and a full treatment requires a generally relativistic magnetohydrodynamic simulation. Furthermore, the supernova is a tremendous explosion, releasing huge amounts of energy in nontrivial mechanisms. A thorough explanation of supernovae is too complex for this setting, but Carroll and Ostlie have an excellent overview of the processes in chapter 8 of *An Introduction to Modern Astrophysics*.

For the context of this thesis, it is enough to know that a supernova occurs when a star explodes at the end of its life. In most varieties of supernova, the star sheds its outer layers incredibly quickly, leaving behind a dense core, often a neutron star. This shedding of its outer layers will tend to make the star lose about 10% of its mass. The collapse of the core leads to an increase in angular velocity of about a hundred orders of magnitude (one author has called core-collapse supernova "the highest known $\dot{\omega}$ in the universe"[1]), and mass loss of around 10 to 20 percent[2]. Additionally, the explosion is not homogenous across the surface of the star. Instead, most of the energy is expelled out of one region. This gives the star a "kick" in whichever direction the inhomogeneity is pointing. Furthermore, this "kick" is not necessarily radial, and can impart angular momentum to the star[3]. This is where we'd expect to see a change in the a spin parameter in the Kerr metric.

These are all the things which relate to supernovae we will seek to investigate with the simulation. Again, supernovae are extremely complex and the simulation presented here will not come close to providing a full treatment of the processes involved. However, the phenomena discussed above absolutely exist, and our simulation is absolutely capable of probing them. Indeed, the effects considered here would be quite mild, so by not considering anything *but* them, we are can more easily see how they manifest than if we were doing a full treatment.

4.1.2 What particles could be affected?

Circumstellar disks

Stars are often observed to have flat disk-like structures orbiting them. These are called **circumstellar disks** and they have extremely complicated dynamics, depending on the size and composition of the matter composing the disk. Circumstellar disks

[1] Lyne & Graham-Smith (2012)
[2] Carroll & Ostlie (2006)
[3] Lai et al. (2001)

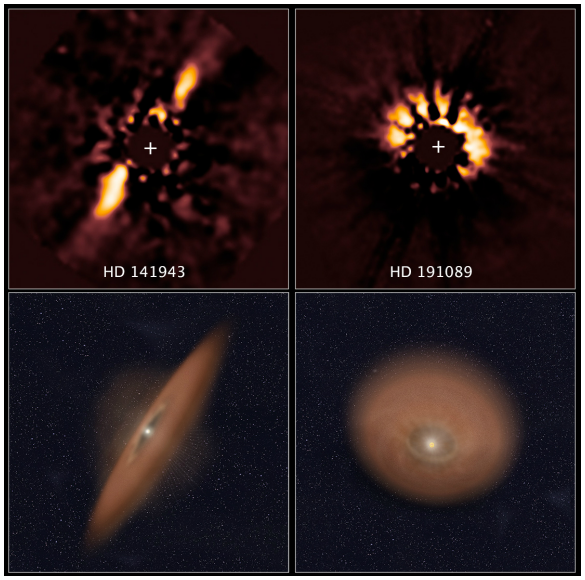


Figure 4.1: At the top, data from a 1997 observation from HST's Near Infrared Camera and Multi-Object Spectrometer shows some structure on the stars HD 141943 and HD 191089. It was later reprocessed in 2014 with image analysis algorithms which revealed a much clearer disk. From Soummer et al. (2014).

which form around young stars are called **protoplanetary disks**, and are composed of dust and gas, and are best modeled as a continuous fluid. Furthermore, they are extremely short-lived; they either accrete into planets or fall back into the star within a few hundred million years. For this reason, the simulation I present here will not apply very well to protoplanetary disks.

Instead, I will be looking at a more general classification of disk called **debris disks** which have been found around mature stars, and, importantly, neutron stars and pulsars[4]. Debris disks can be made of small pebble-sized objects which have the nice property of being decently described as billiard balls knocking into each other. This is easy to simulate. Furthermore, debris disks can be relatively free of difficult-to-model things like dust, gas, and magnetic fields.

Debris disks are huge, extending up to 45AU from the central body (1AU $\approx 1.5 \times 10^8$km is an *astronomical unit*, the mean distance from the Earth to the sun). They are composed of rocky bodies which can range in size from millimeters to tens of meters. The mass of a disk is usually on the order of $.01M_\odot$ to $1M_\odot$[5]. Spread over a flat circle of radius 1.5AU, this gives a highest mass density of about $\rho = .14M_\odot AU^{-2}$. When the disk is illuminated by a nearby radiation source, like a star or neutron star, the disk emits in the infrared. These disks can then be imaged by telescopes like the Hubble Space Telescope (HST), an example observation from which is given in fig. 4.1.

[4]Lyne & Graham-Smith (2012)
[5]Garcia (2011)

So, armed with my n-body simulation, I can ask the question: How does a debris disk respond to the changes in the metric that a supernova or glitch would enact? How does this response change depending on how the disk is oriented to the spin?

Planetary systems

The other type of physical system this code is well-suited for is small numbers of a dozen or so gravitationally interacting bodies. It is natural to assume that the "central body" in this case would be something like the sun, and the particles orbiting it would be planets. But this is less interesting than you might think: Recall from the previous chapter that the effect of general relativity on these particles is *tiny*. The only reason that the central body can be solar mass in a disk simulation is that there are so many orbiting particles that we can hope to see large-scale effects from a tiny perturbation in the metric. Furthermore, the interactions between the orbiting bodies would be pretty uninteresting: Spacing them out like planets in a solar system would result in very minor n-body behavior.

It is much more interesting to instead look at a system of bodies orbiting something which itself is orbiting the central body for which we calculate the general relativistic effects (fig. 4.2). Such a system corresponds to, say, moons around a planet which orbits a gigantic star, or planets around a star which orbits a supermassive black hole. Actually, because the latter system will almost definitely not experience a change in rotational rate or mass of the central body, we will restrict any mapping to physical reality to the moon-around-planet case. However, the problem of n-body dynamics in a curved spacetime is generally quite interesting in its own right, so wether this specific example is terribly physical is not of great personal concern.

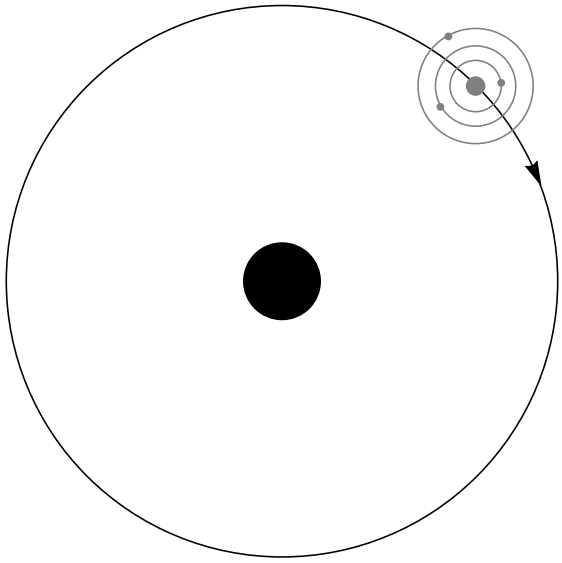


Figure 4.2: An exaggerated top-down view of a system for which n-body dynamics would be interesting. The large black circle would be the object for which general relativistic effects would be calculated, while the gray circles would have a Newtonian interaction.

Chapter 5

Results

Here I present the results of the simulations I performed. The n-body simulations did not work, so in lieu of any simulation output for those, I instead provide a discussion of what went wrong and how to correct it.

5.1 Disk Simulations

In this section, I present the outputs of four disk simulations: 1) an equatorial disk whose central body changes in spin, 2) an equatorial disk whose central body changes in spin and angle θ relative to the disk, 3) an equatorial disk whose central body spins up retrograde to the direction of the disk's orbit, and 4) the same as (3), but with a change in θ relative to the disk. The second and fourth scenarios allows us to probe the θ-dependent dynamics, which turn out to be quite interesting.

There were limitations in performing all of these simulations, largely having to do with particle number. Simulations of more than around 10^4 particles required a prohibitively long time, so to mimic the mass and number density of real disks, I was confined to making my particles orbit only a few Schwarzschild radii from the central body.

Furthermore, I ran out of time before I was able to give my simulations "real" numbers. All of the output data is still in units where $G = c = 1$. Given these limitations, these simulations are not making predictions about exactly what values one would observe in a spin-up event. Rather, they show broad behavioral trends and provide some intuition about the processes which occur.

This fundamental limitation of the simulation freed me up to only consider the most extremal cases, where the spin step-functions from $a = 0$ to $a = 0.99$ instantaneously. This emphasizes any behavior we would see.

The full videos for these simulations are available at the time of this writing at the "adthesis" YouTube channel. In general, they are much more exciting and illuminating than the static images presented here.

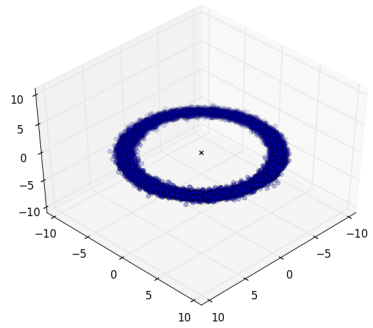


Figure 5.1: $t = 0$. The x represents the center of the central body.

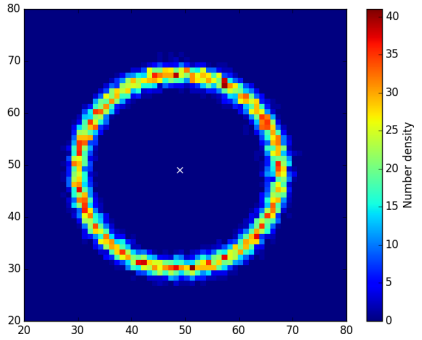


Figure 5.2: A heatmap of the disk at initialization. Color corresponds to number density.

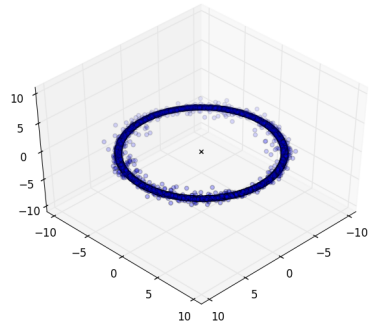


Figure 5.3: The disk at time step 1300. A dense ring has formed.

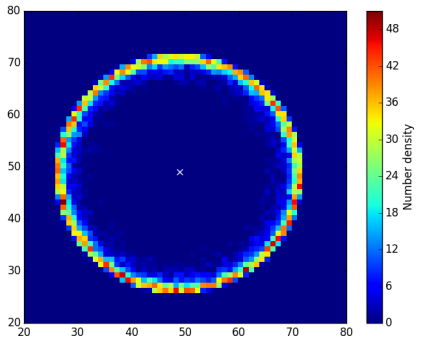


Figure 5.4: The heatmap corresponding to fig. (5.3).

5.1.1 Spin-up, with no change in axis.

In this simulation, 10^4 particles were generated and allowed to run for 40,000 time steps, which corresponds to between 10 and 20 orbits. The particles were initialized in an equatorial disk with $\theta = \pi/2$. The initial radii ranged from about 4 to 5 Schwarzschild radii. When the central body spun up, the spin remained equatorial, so no motion in θ ever occurred. Fig (5.1) is what the disk looked like at the first time step.

The most surprising features of this simulation are density rings which form and then dissipate. First, a single ring forms about a hundred time steps after the spin-up. This ring forms on the inner edge of the disk, and spreads out until the whole disk is consolidated in the ring. This is also how the secondary and tertiary rings form: always on the interior, and spreading out. Three rings formed before any structure dissipated. (figs. 5.3 and 5.6).

These rings expand and shrink before dissipating. The dissipation is due to the particle collisions; when the particles are allowed to pass through each other, the

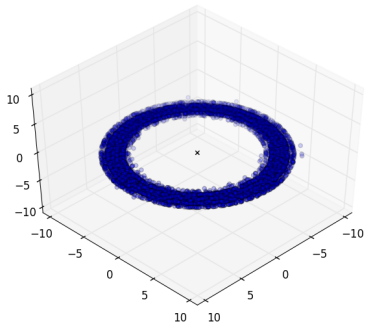


Figure 5.5: The disk at time step 2500. Several rings have formed.

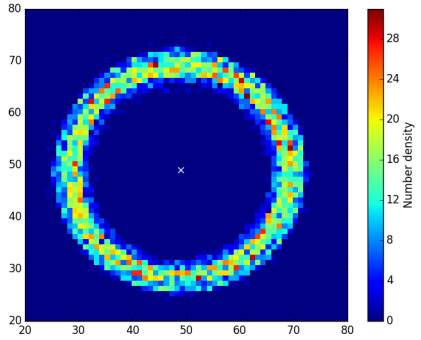


Figure 5.6: The heatmap corresponding to fig. (5.5).

rings remain. Furthermore, the larger the collision radius, the shorter the rings exist, although this relationship was not thoroughly examined.

The formation of many rings is somewhat surprising. Apparently, in the transfer of angular momentum during the spin-up, the order of the disk increases. We can get a grasp for why by taking the ratio of orbital angular momentum before and after the spin-up. Before the spin-up (with $a = 0$), the particle has angular momentum J_z, and after it has J'_z. Dividing one by the other, we have

$$\frac{J_z}{J'_z} = \frac{2r^2\dot{\phi}\left(a^2\cos^2(\theta) + r^2\right)}{\dot{\phi}\left(2a^4\cos^2(\theta) + 4a^2Mr\sin^2(\theta) + a^2r^2(\cos(2\theta) + 3) + 2r^4\right) - 4aMr\dot{t}} \tag{5.1}$$

Fixing everything but r, we find that the ratio between the initial and final angular momenta asymptotes as a function of radius (fig. 5.8). The ratio approaches 1 at infinity, but it decreases rapidly nearer the central body. This means that particles closer to the central body will have a much greater change to their angular momenta than particles further away. Because of this deferential treatment of nearby particles, the orbits will tend to bunch up, as the nearby particles experience a greater change in their orbital angular momentum and energy. This effect can be seen in fig. 5.7, which shows the trajectories of two particles taken from the initial state array of the simulation in fig. 5.3. The inner particle experiences a greater transfer of angular momentum than the outer.

The formation of multiple rings comes from the intersection of all of the slightly-different ellipses. Each time they intersect, a ring is formed. It is interesting to note that although this effect comes from general relativity, it could have been derived classically simply by looking at an orbital system which gets a radially-dependent kick in angular momentum.

5.1.2 Spin-up, change in axis.

Again, 10^4 particles were generated and allowed to run for 17,000 time steps. The particles were initialized in an equatorial disk with $\theta = \pi/2$. At time step 300, the

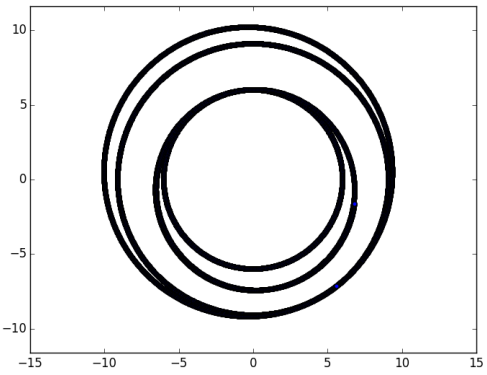


Figure 5.7: An interior particle will experience a greater change in angular momentum and energy than an exterior particle.

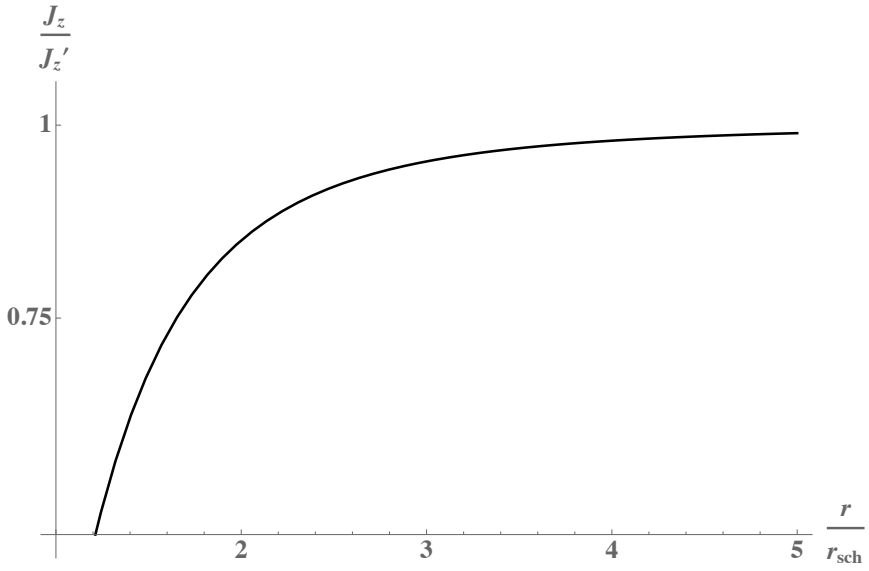


Figure 5.8: Ratio of angular momenta before and after the kick. r is measured in units of Schwarzschild radii. The further from the central body, the less an effect the kick has on the angular momentum.

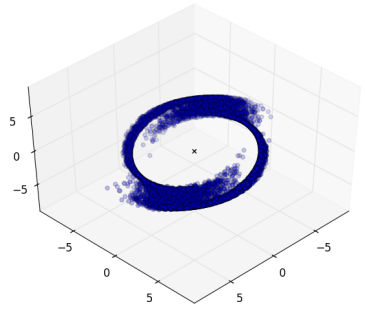


Figure 5.9: Time step 818. The central body has increased in spin and changed the angle relative to the disk.

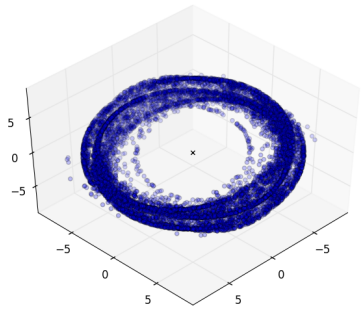


Figure 5.10: Time step 10,565. The orbits have preferentially aligned themselves along certain polar angles corresponding to particles of similar initial radii.

central body spun up but also changed its polar angle relative to the disk by $\pi/6$ radians. This induced polar motion of the particles in the disk.

The initial conditions look identical to the previous scenario, so those plots will not be reproduced. Furthermore, the density plots become unhelpful here, because there is no obvious 2D plane in which to slice the data in a way which would be helpful.

Almost immediately after spinning up, the particles divide into two regions (fig. 5.9). Why they divide into exactly two could be due to a natural grouping of particles, stemming from the projection of their position and velocity vectors onto the angular momentum vector of the central body. This is to say, this dot product will be greater than 0 for half of the particles and less than 0 for the other half.

There's a similarity with the previous scenario, as well. In this case, not only is the change in angular momentum dependent on radius, but the force in the polar coordinate is as well. Therefore, we expect to see some "preferred" polar angles that the particles will cluster around.

Indeed, that is exactly what happens (fig. 5.10). To show that they really are clustered, fig. 5.11 shows the distribution of particles in θ.

5.1.3 Retrograde spin changes

These were the last simulations performed. Again, 10^4 particles were initialized in an equatorial disk, and again, the change in spin was introduced at the 300^{th} time interval. In the first, they remained in the plane, and in the second, the spin-up was accompanied by a change in θ. In comparison with the prograde spin changes, a lot of what happens to the disk is familiar: rings form, and there are preferred angles in θ that the particles like to group around. However, there are two key differences, stemming from the fact that this retrograde spin-up *subtracts* angular momentum from the particles.

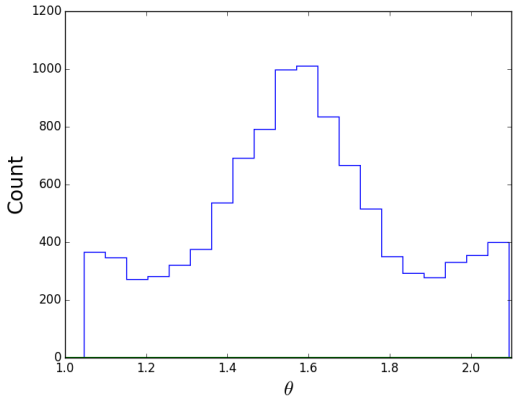


Figure 5.11: A histogram of angles in theta at time step 10,565. There are certain orbits the particles prefer to be in, corresponding to their initial conditions at the time of the spin-up.

First, the rings form in reverse: Instead of densifying at the middle and moving outward, now the particles fall sparsely towards the center before accumulating there in a dense ring. Second, there are some particles which lose sufficient angular momentum to be sucked up by the central body. This leads to a "purging" phase where the central body rapidly eats up all the particles which have lost sufficient energy and momentum (figs. 5.12, 5.13). This is very fast, lasting only a few hundred time steps. After this phase, it is clear the disk is lacking particles within a certain radius (figs. 5.14, 5.15).

The fact that the purge has a defined start and end is made clear in figs. 5.16 and 5.17, plots of the number particles over time. After a sharp plunge, the number of particles stays constant. The off-θ case is much more dramatic than the equatorial case. One possible cause for this could be the θ-dependence of (5.1). As you can see in fig. 5.18, when all other variables are held constant, the ratio between the initial and final angular momenta increases as θ moves from $\pi/2$ during a retrograde spin up. This means that the final angular momentum of a particle is less the further that particle is from the equator. Therefore, a system with a large displacement in θ will experience more particle loss.

This dependence of the angular momentum on θ is surprising and non-intuitive. A similar relationship is found when we look at the dependence on θ of the radial acceleration (fig. 5.19). This relationship can help explain why the off-theta simulation purges of its particles so much sooner after the kick than the equatorial scenario.

5.2 N-Body simulation: Why didn't it work?

I was unable to find a set of initial conditions for two Newtonian particles which maintained a stable orbit (fig. 5.20). I attempted to create a weak analogue to a star-planet system, where the body labeled 2 in (fig. 5.20) would be 6 or so orders of

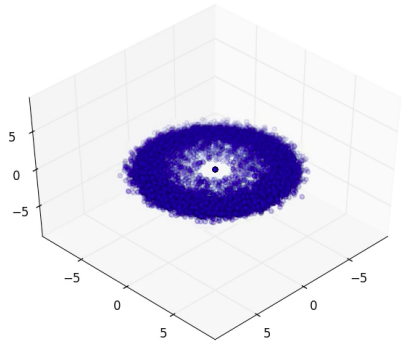


Figure 5.12: Time step 4000. The equatorial disk "purging" of all the particles which lost sufficient momentum to fall into the central body.

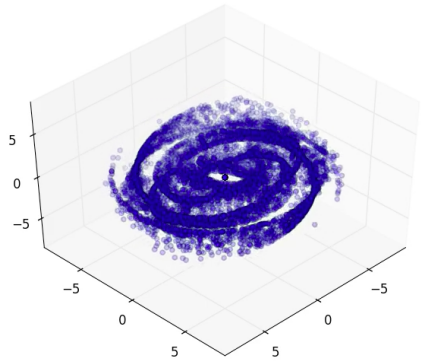


Figure 5.13: Time step 4000. The off-θ version of the left figure.

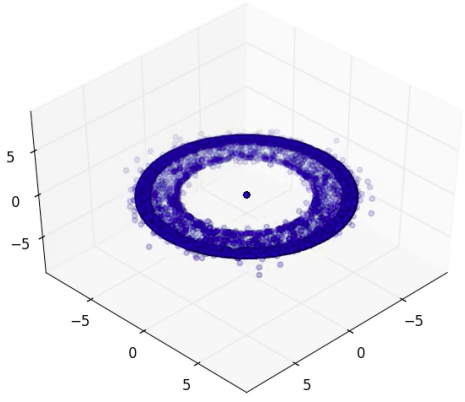


Figure 5.14: Time step 8000. The smallest orbit corresponds to the largest angular momentum which survived the purge.

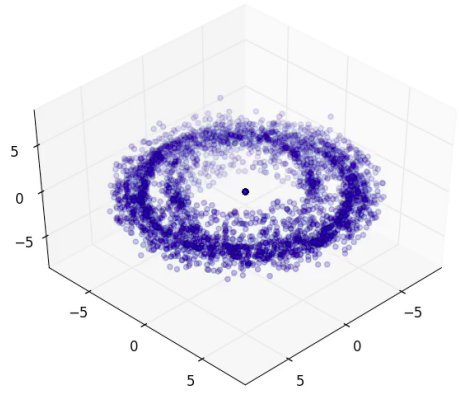


Figure 5.15: Time step 8000. The off-θ version of the left figure.

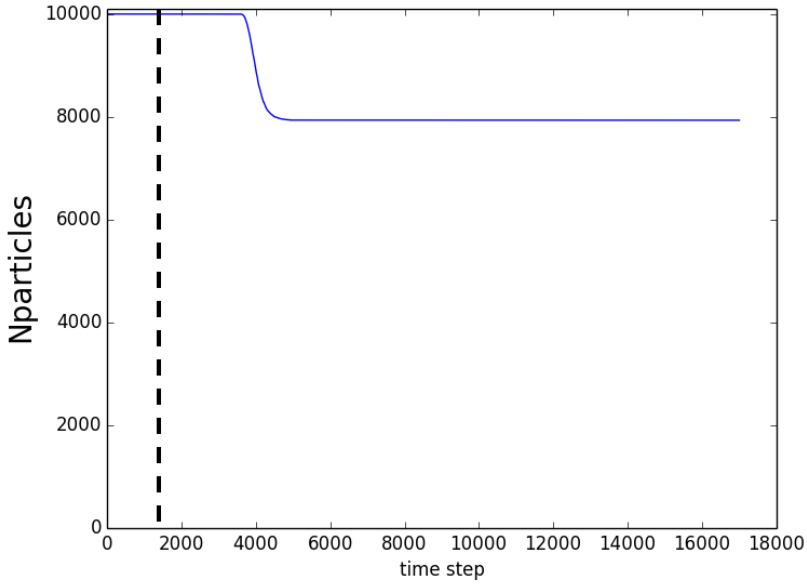


Figure 5.16: The number of particles over time for the system in figs. 5.12 and 5.14. In both this and fig. 5.17, the dashed line represents the moment of spin-up.

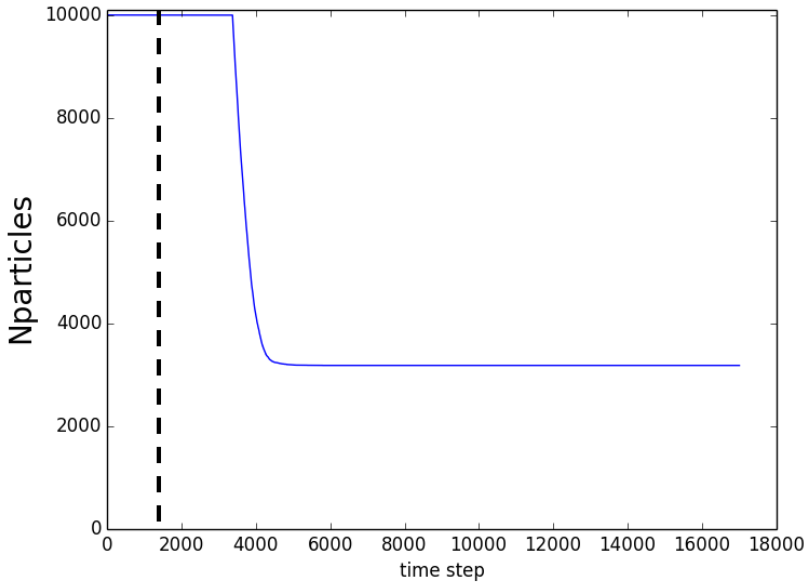


Figure 5.17: The off-θ version of the above figure. The particles deplete much more quickly than in the equatorial case.

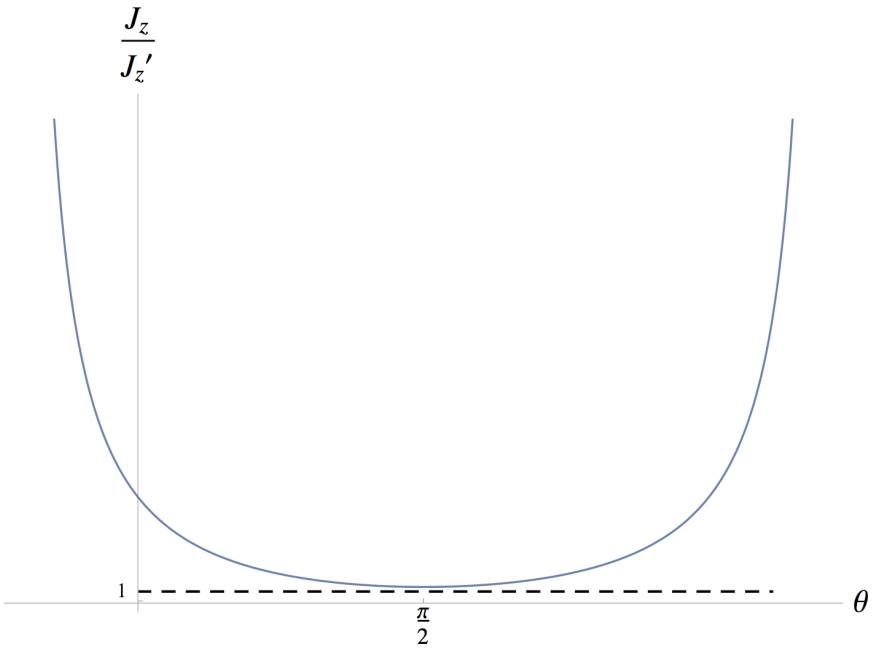


Figure 5.18: The θ dependence of (5.1). As the particle moves in θ from the equator, its final angular momentum diminishes.

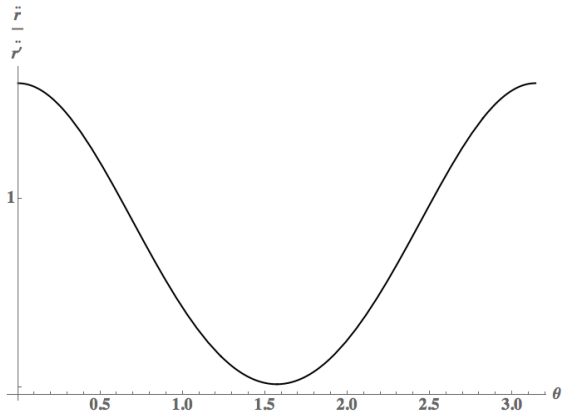


Figure 5.19: The dependence on θ of the radial acceleration before and after the spin-up.

magnitude more massive than body 3. While I had an easy time making them stable on a flat background, as soon as I calculated their motion on the Kerr background, the smaller body was either ejected or was swallowed by the massive, GR body (body 1). A script was written to try 10^3 different initial conditions of varying initial angular momenta of body 3, but none were found which survived more than 1 orbit around body 2.

Figure 5.20: An example of an n-body system. Body 1 sources the GR effects.

One problem could have been the different length scales involved. The orbit of body 2 around body 1 is much much greater than 3 around 2. In physical reality, it would be many orders of magnitude different, but the simulations I ran only had a difference of factors of 3 or 4. Correcting this would mean not only calculating new initial conditions for the much smaller secondary orbit of body 3, but also have a much smaller time step to accommodate the smaller length scale.

Chapter 6

Future Work

Towards the end of this thesis, I encountered two problems I find fascinating, and are something I am eager to work on in the future. They concern the very thing which failed: Newtonian gravity in a curved background. Doing Newtonian gravity in any curved background has some difficulties whose solutions are potentially really interesting. As a scalar field, a the strength of a Newtonian potential is only dependent on the distance between the interacting particles. However, on a curved background, distance is really poorly defined! It is highly path-dependent, and differs depending on if you're a photon, or if you're a massive particle. Indeed, the solution I present in this thesis is inherently flawed: I was only ever calculating the Pythagorean, flat distance between the particles [1]. The two extensions I would like to work on are potential solutions to this problem.

6.1 Photon firing

While the graviton is still purely hypothetical, most models give it no mass. This has several attractive aspects, not least of which would be the fact that this would enable it to travel at c, something we otherwise assume [2]. Therefore, one solution, proposed to me by Joel Franklin, could be to calculate the null geodesic connecting the two particles, and use the spacetime length of that vector to calculate the Newtonian motion. This would be a not-too-difficult computational problem: It is a boundary value problem, so I would use something like the bisection method to hone in on a geodesic which connected the two particles. The function I would be minimizing would be the distance (in 4-space) between the target particle and the end of the geodesic. I would know when to stop calculating the null geodesic by watching its r value. If the r value went past that of the particle I would be done, as I know the geodesic couldn't backtrack and increase in r.

This would have bizarre consequences on the dynamics of the particles. In fig. 6.1, several null geodesics of unit length starting from a particle outside a black hole

[1] Note, however, that this is not why I failed– this would only mean that my simulation would lose accuracy the closer you got to the central body.

[2] Grøn & Hervik (2007)

are plotted. It is not clear what would happen if one uses these to for the length in a Newtonian potential. The point labeled A in fig. 6.1 is "closer" to the particle than the point labeled B. It is unclear whether such a system would have any stable orbits.

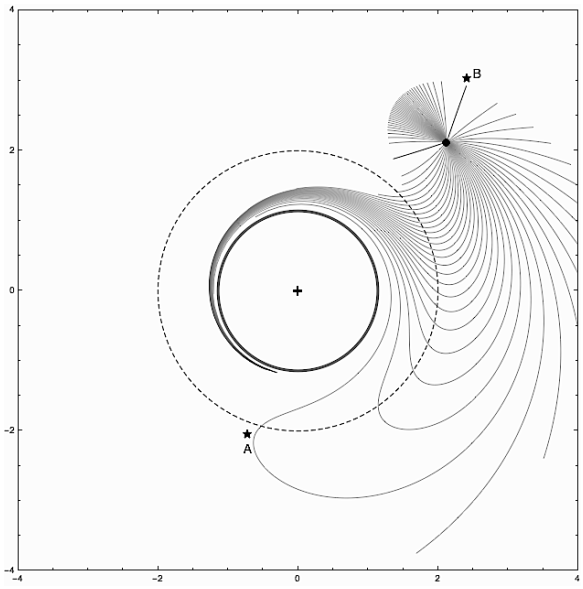


Figure 6.1: Null geodesics of unit length. Using these for calculating a Newtonian potential, point A would feel a greater force than point B. The "+" is the center of the black hole for $a = 0.99$. The solid line is the event horizon and the dashed line is the ergosphere.

6.2 Scalar field equations

The other option would be to calculate the field equations for a scalar field in the Kerr metric. To mimic the flat-space scalar Newtonian field, this would mean extremizing the field action,

$$L = \int \phi_{,\mu} g^{\mu\nu} \phi_{,\nu} d^4x, \tag{6.1}$$

where $g^{\mu\nu}$ is the Kerr metric.

Carrying through the calculations, one would arrive at the vacuum field equation:

$$\phi^{\mu}_{;\mu} = 0, \tag{6.2}$$

which looks very promising, as this is precisely the geodesic equation (the semi-colon refers to the covariant derivative). This would mean that the field lines move along geodesics in exactly the way that the above method assumed a priori. However, it would necessarily not be in a vacuum; the field is being sourced by the particle of interest. Therefore, the field equation would not be exactly the geodesic equation, but be equal to some mass density ρ:

$$\phi^{\mu}_{;\mu} = \rho. \tag{6.3}$$

This field could then be coupled to a particle, and the equations of motion derived. These equations of motion, then, could be integrated to investigate the motions in of a particle in such a field.

Conclusion

In this thesis, I wrote computer code which integrates the orbits of particles in a time-dependent Kerr spacetime. My simulations of large numbers of particles in a disk showed that there, the particles do not undergo uniform changes in orbit, but experience changes of varying severity, depending on their initial conditions at the time of spin-up. This leads to the formation of ring structures in the disk. If the central body spins retrograde to the direction of disk orbit, then particles lacking sufficient angular momentum and energy can collide with the central body. The investigations of n-body behavior were not successful. No stable Newtonian orbit could be found in the curved spacetime.

Appendix A

OINKS source code

```
"""
Orbit Integrator for N-body Kerr Solutions (OINKS)
"It's a real resource hog!"

This version is for 3d disk simulation

Author: Alex Deich :: http://github.com/deichdeich
Date: February, 2016
"""

from __future__ import division, print_function
import numpy128 as np
import time
import matplotlib.pyplot as plt
import sys
import os
import OINKSTools as farts
import scipy.stats
from astropy.io import fits
from scipy.spatial.distance import pdist,squareform

class OrbitalSystem(object):
    def __init__(self,
                 Nparticles,
                 M=1,
                 a = None,
                 dt=0.01,
                 interaction = False,
                 collisions = 'elastic',
                 masses = None,
                 init_state = None,
                 cr = 0.001,
                 save_dir = "./output"):

        self.start_particles = Nparticles
        self.Nparticles = Nparticles
        self.interaction = interaction
        self.M = M
        self.dt = dt
        self.a = a
        self.cr = cr
        self.event_horizon = 0
        self.cleanup = []
        if init_state is not None:
            self.use_state = np.copy(init_state)
        else:
            self.use_state = None
        self.save_dir = save_dir
        self.init_state = np.copy(init_state)
        self.state = np.copy(init_state)
        self.collision_dict = {}
        self.collisions=collisions
        self.collided_particles = np.array([])
        self.fname_list = []
        self.dirname = ""
        self.plotdata = 0
        self.skip_mkdir=False
        self.nsteps = 0

        self.old_collisions = []

        if self.Nparticles == None and self.init_state is not None:
            self.Nparticles = len(self.init_state)

        if self.a == None:
            self.a = farts.a0
```

```
    if init_state is not None:
        if init_state.shape == (Nparticles,2,3):
            self.init_state = init_state
        else:
            raise ValueError("Initial state not the right shape: ",init_state.shape,
            "\nShould be ",(Nparticles,2,3))

    if masses == None:
        self.masses = np.zeros(Nparticles)+0.0001
    else:
        if len(masses) == Nparticles:
            self.masses = masses
        else:
            raise ValueError("Mass list must be of length ", Nparticles,
                        "\nGiven length:", len(masses))

    self.collision_radius = .05

print('sjkkjds')
def SingleParticleDerivativeVector(self, kstate, particle, t):
    pstate = kstate[particle]
    r = pstate[0,0]
    theta = pstate[0,1]
    phi = pstate[0,2]
    if(r < self.event_horizon+0.2):
        if particle not in self.cleanup:
            self.cleanup.append(particle)
    f = np.array(([pstate[0,0],pstate[1,0]],
                  [pstate[0,1],pstate[1,1]],
                  [pstate[0,2],pstate[1,2]]))

    rdd = -((pow(self.a(t),2) - 2*self.M*f[0,0] + pow(f[0,0],2))*((4*self.M*pow(f[0,0],2))/
    pow(pow(self.a(t),2)*pow(np.cos(f[1,0]),2) + pow(f[0,0],2),2) - (2*self.M)/(pow(self.a(t),
    2)*pow(np.cos(f[1,0]),2) + pow(f[0,0],2)) + (4*(self.M - f[0,0])*(pow(self.a(t),2)*pow(np.cos(f[1,0]),2)
    + pow(f[0,0],2))*pow(f[0,1],2))/pow(pow(self.a(t),2) - 2*self.M*f[0,0] + pow(f[0,0],2),2) + ((-2*self.M
     + 2*f[0,0])*(pow(self.a(t),2)*pow(np.cos(f[1,0]),2) + pow(f[0,0],2))*pow(f[0,1],2))/pow(pow(self.a(t),2) -
    2*self.M*f[0,0] + pow(f[0,0],2),2) - (2*f[0,0]*pow(f[0,1],2))/(pow(self.a(t),2) - 2*self.M*f[0,0] +
    pow(f[0,0],2)) - 2*f[0,0]*pow(f[1,1],2) - (4*self.a(t)*self.M*pow(f[0,0],2)*f[2,1]*pow(np.sin(f[1,0]),2))/
    pow(pow(self.a(t),2)*pow(np.cos(f[1,0]),2) + pow(f[0,0],2),2) +
    (2*self.a(t)*self.M*f[2,1]*pow(np.sin(f[1,0]),2))/(pow(self.a(t),2)*pow(np.cos(f[1,0]),2) + pow(f[0,0],
    2)) - (2*f[2,1]*pow(np.sin(f[1,0]),2)*(pow(self.a(t),4)*pow(np.cos(f[1,0]),4)*f[0,0]*f[2,1] +
    2*pow(self.a(t),2)*pow(np.cos(f[1,0]),2)*pow(f[0,0],3)*f[2,1] + pow(f[0,0],5)*f[2,1] +
    self.a(t)*self.M*pow(f[0,0],2)*(1 - self.a(t)*f[2,1]*pow(np.sin(f[1,0]),2)) + pow(self.a(t),
    3)*self.M*pow(np.cos(f[1,0]),2)*(-1 + self.a(t)*f[2,1]*pow(np.sin(f[1,0]),2))))/pow(pow(self.a(t),
    2)*pow(np.cos(f[1,0]),2) + pow(f[0,0],2),2) - (2*f[0,1]*(-2*f[0,0]*f[0,1] + pow(self.a(t),
    2)*f[1,1]*np.sin(2*f[1,0])))/(pow(self.a(t),2) - 2*self.M*f[0,0] + pow(f[0,0],2))))/(2.*(pow(self.a(t),
    2)*pow(np.cos(f[1,0]),2) + pow(f[0,0],2)))


    Tdd = -((4.*pow(self.a(t),3)*np.cos(f[1,0])*f[0,0]*f[2,1]*pow(np.sin(f[1,0]),3))/pow(pow(self.a(t),
    2)*pow(np.cos(f[1,0]),2) + pow(f[0,0],2),2) - (2.*pow(self.a(t),2)*f[0,0]*np.sin(2*f[1,0]))/
    pow(pow(self.a(t),2)*pow(np.cos(f[1,0]),2) + pow(f[0,0],2),2) + (pow(self.a(t),2)*pow(f[0,1],
    2)*np.sin(2*f[1,0]))/(pow(self.a(t),2) - 2.*f[0,0] + pow(f[0,0],2)) + pow(self.a(t),2)*pow(f[1,1],
    2)*np.sin(2*f[1,0]) + (2.*self.a(t)*f[0,0]*f[2,1]*np.sin(2*f[1,0]))/(pow(self.a(t),2)*pow(np.cos(f[1,0]),2)
    + pow(f[0,0],2)) - (f[2,1]*(8*pow(self.a(t),6)*pow(np.cos(f[1,0]),4)*f[2,1] + 4*pow(self.a(t),
    4)*pow(np.cos(f[1,0]),2)*(5 + np.cos(2*f[1,0]))*pow(f[0,0],2)*f[2,1] + 8*pow(self.a(t),2)*(2 +
    np.cos(2*f[1,0]))*pow(f[0,0],4)*f[2,1] + 8*pow(f[0,0],6)*f[2,1] + 16.*self.a(t)*pow(f[0,0],3)*(-1 +
    2*self.a(t)*f[2,1]*pow(np.sin(f[1,0]),2)) - 2.*pow(self.a(t),3)*f[0,0]*(8 - 4*self.a(t)*(3 +
    np.cos(2*f[1,0]))*f[2,1]*pow(np.sin(f[1,0]),2)))*np.sin(2*f[1,0]))/(8.*pow(pow(self.a(t),
    2)*pow(np.cos(f[1,0]),2) + pow(f[0,0],2),2)) - 2*f[1,1]*(-2*f[0,0]*f[0,1] + pow(self.a(t),
    2)*f[1,1]*np.sin(2*f[1,0])))/(2.*(pow(self.a(t),2)*pow(np.cos(f[1,0]),2) + pow(f[0,0],2)))


    Pdd = (2*(-(f[0,1]*(pow(self.a(t),4)*pow(np.cos(f[1,0]),2)*pow((np.cos(f[1,0])/(np.sin(f[1,0]))),
    2)*f[0,0]*f[2,1] + 2*pow(self.a(t),2)*pow((np.cos(f[1,0])/(np.sin(f[1,0]))),2)*pow(f[0,0],3)*f[2,1] +
    pow((1/np.sin(f[1,0])),2)*pow(f[0,0],5)*f[2,1] + 1.*self.a(t)*pow(f[0,0],2)*(pow((1/np.sin(f[1,0])),2) -
     self.a(t)*f[2,1]) + 1.*pow(self.a(t),3)*(-pow((np.cos(f[1,0])/(np.sin(f[1,0]))),2) +
     self.a(t)*pow(np.cos(f[1,0]),2)*f[2,1]))) - ((np.cos(f[1,0])/(np.sin(f[1,0])))*pow((1/np.sin(f[1,0])),
     2)*f[1,1]*(8*pow(self.a(t),6)*pow(np.cos(f[1,0]),4)*f[2,1] + 4*pow(self.a(t),4)*pow(np.cos(f[1,0]),
     2)*(5 + np.cos(2*f[1,0]))*pow(f[0,0],2)*f[2,1] + 8*pow(self.a(t),2)*(2 + np.cos(2*f[1,0]))*pow(f[0,0],
     4)*f[2,1] + 8*pow(f[0,0],6)*f[2,1] + 16.*self.a(t)*pow(f[0,0],3)*(-1 +
     2*self.a(t)*f[2,1]*pow(np.sin(f[1,0]),2)) - 2.*pow(self.a(t),3)*f[0,0]*(8 - 4*self.a(t)*(3 +
     np.cos(2*f[1,0]))*f[2,1]*pow(np.sin(f[1,0]),2))))/8.))/((pow(self.a(t),2)*pow(np.cos(f[1,0]),2) +
     pow(f[0,0],2))*(pow(self.a(t),4)*pow((np.cos(f[1,0])/(np.sin(f[1,0]))),2) + 2.*pow(self.a(t),2)*f[0,0] +
     pow(self.a(t),2)*(-1 + 2*pow((1/np.sin(f[1,0])),2))*pow(f[0,0],2) + pow((1/np.sin(f[1,0])),
     2)*pow(f[0,0],4)))

    # The Kerr metric
    G = np.array([[f[0,1], f[1,1], f[2,1]],
                  [rdd, Tdd, Pdd]])

    return(G)


def UpdateStateVectorRK4(self,t):
    self.event_horizon = self.get_event_horizon(t)
    new_state = np.ndarray((self.Nparticles,2,3))

    for particle in xrange(self.Nparticles):

        kstate = np.copy(self.state)
```

```
    k1 = self.dt * self.SingleParticleDerivativeVector(kstate,particle, t)
    kstate[particle] = np.copy(self.state[particle]) + k1/2
    k2 = self.dt * self.SingleParticleDerivativeVector(kstate,particle,t+(self.dt/2))
    kstate[particle] = np.copy(self.state[particle]) + k2/2
    k3 = self.dt * self.SingleParticleDerivativeVector(kstate,particle,t+(self.dt/2))
    kstate[particle] = np.copy(self.state[particle]) + k3
    k4 = self.dt * self.SingleParticleDerivativeVector(kstate,particle,t+self.dt)
    new_state[particle] = np.copy(self.state[particle]) + (1/3)*(k1/2 + k2 + k3 + k4/2)

    """
    if t > 29.0 and t<29.0+(self.dt):
        print('\nkick\n')
        new_state[particle,0,1] += np.pi/6 * np.cos(new_state[particle,0,2])
    """

#Get rid of gobbled or ejected particles
if self.cleanup != []:
    new_state = np.delete(new_state,self.cleanup,axis=0)
    self.remove_particle(particle)
    for particle in self.cleanup:
        print("\n***particle {} shit the bed at step {}***".format(particle,int(t/self.dt)))
        print("***particle {} removed***".format(particle))
        self.cleanup.remove(particle)
        if self.cleanup == []:
         print("***cleanup completed***")
self.cleanup = []


big_particle_list = []
big_vector_list = []
new_masses_list = []


if self.collisions == 'elastic':

    distances = squareform(pdist(self.state[:,0,:], metric='sph'))
    ind1, ind2 = np.where(distances < 2 * self.collision_radius)
    unique = (ind1 < ind2)
    ind1 = ind1[unique]
    ind2 = ind2[unique]
    new_collisions = zip(ind1,ind2)
    for i in new_collisions:
     if i not in self.old_collisions:
     i1 = i[0]
     i2 = i[1]

         m1 = self.masses[i1]
         m2 = self.masses[i2]

         r1 = new_state[i1,0,0]
         P1 = new_state[i1,0,1]
         T1 = new_state[i2,0,3]
         r2 = new_state[i2,0,0]
         P2 = new_state[i2,0,1]
         T2 = new_state[i2,0,2]

         r1d = new_state[i1,1,0]
         P1d = new_state[i1,1,1]
         T1d = new_state[i1,1,2]
         r2d = new_state[i2,1,0]
         P2d = new_state[i2,1,1]
         T2d = new_state[i2,1,2]

         new_state[i1, 1] = [(r1d*(m1-m2) + 2*m2*r2d)/(m1+m2),
                             (P1d*(m1-m2) + 2*m2*P2d)/(m1+m2),
                             (T1d*(m1-m2) + 2*m2*T2d)/(m1+m2)]
         new_state[i2, 1] = [(r2d*(m2-m1) + 2*m1*r1d)/(m1+m2),
                             (P2d*(m2-m1) + 2*m1*P1d)/(m1+m2),
                             (T2d*(m2-m1) + 2*m1*T1d)/(m1+m2)]
    self.old_collisions = new_collisions

 elif self.collisions == "inelastic":
    if self.collision_dict != {}:
        for key in self.collision_dict:
         particles = self.collision_dict[key][1]
         m = 0
         x = self.collision_dict[key][0][0]
         y = self.collision_dict[key][0][1]
         z = self.collision_dict[key][0][1]
         mvx = 0
         mvy = 0
         mvz = 0
         for particle in particles:
             m += self.masses[particle]
         for particle in particles:
             mvx += (self.state[particle][1][0]*self.masses[particle])
             mvy += (self.state[particle][1][1]*self.masses[particle])
             mvz += (self.state[particle][1][1]*self.masses[particle])
             big_particle_list.append(particle)
         new_masses_list.append(m)
        new_particle = [[x,y,z],[mvx/m,mvy/m,mvz/m]]
```

```
            big_vector_list.append(new_particle)

            self.masses = np.delete(self.masses,big_particle_list,axis=0)
            self.masses = np.append(self.masses,np.array(new_masses_list),axis = 0)
            new_state = np.delete(new_state,big_particle_list,axis=0)
            new_state = np.append(new_state,np.array(big_vector_list),axis = 0)
            self.Nparticles = len(new_state)
            self.collision_dict = {}
            self.collided_particles = np.array([])

    return(new_state)

def TimeEvolve(self,nsteps,comments,write=True):
    self.cleanup = []
    self.nsteps = nsteps
    t=0
    ##Get init_state
    if self.use_state == None:
        self.cleanup = []
        self.state = np.copy(farts.make_initial_conditions(self.start_particles,
                                                           self.M,
                                                           self.a(0)))
        self.init_state = np.copy(self.state)
        self.Nparticles = len(self.state)
    else:
        self.state = np.copy(self.init_state)
    print("Got initial conditions for {} particles".format(self.start_particles))

    primary = self.get_header(nsteps,comments)
    frame0 = self.get_hdu()

    filenum = farts.get_filenum(self.save_dir,self.start_particles)
    self.dirname = "{}/nbody_3d_{}_{}".format(self.save_dir,self.start_particles,filenum)
    os.mkdir(self.dirname)
    os.mkdir("{}/data".format(self.dirname))
    hdulist = fits.HDUList([primary,frame0])
    total_time = 0
    savenums = nsteps/1000
    print('\n\n')
    for step in xrange(1, nsteps):
        stepstart = time.time()
        self.state = self.UpdateStateVectorRK4(t)
        framen = self.get_hdu()
        hdulist.append(framen)
        t += self.dt
        end = time.time()
        steptime = end-stepstart
        total_time += steptime
        avg = total_time/step
        perc = 100*((step+1)/nsteps)
        sys.stdout.write('\rFrame {} of {} completed ({}%).  Step: {}, Total: {}s, Sim time: {}, Estimated time remaining: {}s. Nparticles: {}'.format(step+1,
                                                          nsteps,
                                                          '%0.1f'%perc,
                                                          '%0.4f'%steptime,
                                                          '%0.4f'%total_time,
                                                          '%0.2f'%t,
                                                          '%i'%(((avg * nsteps)+1)-total_time),
                                                          '%i'%self.Nparticles))
        sys.stdout.flush()
        if step%1000 == 0:

            if write == True:
             print("\nWriting to disk...")
             fname = "{}/data/{}.fits".format(self.dirname,step)
             hdulist.writeto(fname,clobber=True)
             print("Frames {} - {} written at {}".format(step-1000,step,fname))
             hdulist = fits.HDUList([primary])
             self.fname_list.append(fname)

    if len(hdulist)!=1:
        print("\nWriting to disk...")
        fname = "{}/data/{}.fits".format(self.dirname,step)
        hdulist.writeto(fname,clobber=True)
        print("Frames {} written at {}".format(step+1,fname))
        self.fname_list.append(fname)

def get_header(self,nsteps,comments=""):
    prihdr = fits.Header()
    prihdr["NPARTS"] = self.Nparticles
    prihdr["PARTGRAV"] = self.interaction
    prihdr["DT"] = self.dt
    prihdr["BHMASS"] = self.M
    if self.a:
        prihdr["SPINFUNC"] = True
    else:
        prihdr["SPINFUNC"] = False
    prihdr["NSTEPS"] = nsteps
    prihdr["COMMENTS"] = comments
    prihdr["COLRAD"] = self.cr
    prihdu = fits.PrimaryHDU(header=prihdr)

    return(prihdu)
```

```
    def get_hdu(self):
        state_transpose = self.state.T
        frame = fits.BinTableHDU.from_columns([fits.Column(name='R',format='E',array = state_transpose[0][0]),
                                               fits.Column(name='T',format='E',array = state_transpose[1][0]),
                                               fits.Column(name='P',format='E',array = state_transpose[2][0]),
                                               fits.Column(name='Rd',format='E',array = state_transpose[0][1]),
                                               fits.Column(name='Td',format='E',array = state_transpose[1][1]),
                                               fits.Column(name='Pd',format='E',array = state_transpose[2][1]),
                                               fits.Column(name='MASS',format='E',array = self.masses)])
        return(frame)

    def remove_particle(self,particle):
        self.state = np.delete(self.state,self.cleanup,axis=0)
        for key in self.collision_dict:
            if particle in self.collision_dict[key][1]:
                self.collision_dict[key][1].remove(particle)
        self.Nparticles = len(self.state)

    def get_event_horizon(self,t):
        return(self.M + np.sqrt(self.M**2 - self.a(t)**2))



#####  Plotting Tools  #####
    def movie(self,type=None,function = "count"):
        if self.skip_mkdir == False:
            os.mkdir("{}/movies".format(self.dirname))
            os.mkdir("{}/movies/spatial".format(self.dirname))
            os.mkdir("{}/movies/spatial/frames".format(self.dirname))
            os.mkdir("{}/movies/heatmap".format(self.dirname))
            os.mkdir("{}/movies/heatmap/frames".format(self.dirname))
        if type == 'spatial':
            self.make_spatial()
        elif type == 'heatmap':
            self.make_heatmap(function)
        elif type == None:
            self.make_heatmap(function)
            self.make_spatial()

    def make_heatmap(self,function):
        print("\n")
        print("Creating animated heatmap...")
        plt.figure()
        n=1
        for fname in self.fname_list:
            wholedata = fits.open(fname)
            for i in xrange(1,len(wholedata)):
             data = wholedata[i].data
                bins = scipy.stats.binned_statistic_2d(data['X'],
                                                       data['Y'],
                                                       data['MASS'],
                                                       statistic=function,
                                                       bins=50,
                                                       range=[[-30,30],[-30,30]])

                x,y = np.where(~np.isnan(bins[0]))
                plt.scatter(x,y,c=bins[0][np.where(~np.isnan(bins[0]))],
                                                         marker='s',
                                                         edgecolors='none',
                                                         s=200)
                t = n*self.dt
                plt.xlim(0,48)
                plt.ylim(0,48)
                circle1 = plt.Circle((24,24), radius = self.get_event_horizon(t), color = 'r', fill = False)
                fig = plt.gcf()
                fig.gca().add_artist(circle1)
                fig.gca().axes.get_xaxis().set_visible(False)
                fig.gca().axes.get_yaxis().set_visible(False)
                plt.axes().set_aspect('equal')
                plt.title('Nparticles: {}'.format(len(data['X'])))
                cb = plt.colorbar()
                plt.clim(0,50)
                cb.set_label('Number density')
                plt.scatter(24,24,marker="x", color="white")
                fname = "{}/movies/heatmap/frames/{}.png".format(self.dirname,n)
                plt.savefig(fname)
                plt.clf()
                sys.stdout.write('\rFrame {} completed'.format(n))
                sys.stdout.flush()
                n+=1
            wholedata.close()
        os.system("ffmpeg -framerate 300 -i {}/movies/heatmap/frames/
        %d.png -c:v libx264 -r 30 -pix_fmt yuv420p {}/movies/heatmap/
        out.mp4".format(self.dirname,self.dirname))

    def make_spatial(self):
        from mpl_toolkits.mplot3d import axes3d
        print("\n")
        print("Creating 3D position space movie...")
        n=1
     plotrange = self.state[0,0,0]+.25*self.state[0,0,0]
```

```
    fig = plt.figure()
    for fname in self.fname_list:
        wholedata = fits.open(fname)
        for i in xrange(1,len(wholedata)):
            ax = fig.add_subplot(111,projection = '3d')
            data = wholedata[i].data
            ax.scatter(data['R']*np.sin(data['T'])*np.cos(data['P']),
                       data['R']*np.sin(data['T'])*np.sin(data['P']),
                       data['R']*np.cos(data['T']),alpha = 0.3)
            ax.scatter(0,0,0,marker="x", color="black")
            t = n*self.dt
            ax.set_xlim3d(-plotrange,plotrange)
            ax.set_ylim3d(-plotrange,plotrange)
            ax.set_zlim3d(-plotrange,plotrange)
            ax.view_init(45,45)
            #plt.title('Nparticles: {}'.format(len(data['R'])))
            fname = "{}/movies/spatial/frames/{}.png".format(self.dirname,n)
            plt.savefig(fname)
            plt.clf()
            sys.stdout.write('\rFrame {} completed'.format(n))
            sys.stdout.flush()
            n+=1
        wholedata.close()
    print('\n')
    os.system("ffmpeg -framerate 300 -i {}/movies/spatial/frames/
    %d.png -c:v libx264 -r 30 -pix_fmt yuv420p {}/movies/spatial/
    out.mp4".format(self.dirname,self.dirname))

def plot_n(self):
    ns = np.ndarray(len(self.nsteps))
    i=0
    for fname in self.fname_list:
        data = fits.open(fname)
        ns[i] = len(data[1].data)-1
        i+=1
    ns = ns/self.init_particles
    plt.plot(ns)
    plt.set_xlabel('time step')
    plt.set_ylabel('\r$n_step/N$')


def plottraj3d(self):
    from mpl_toolkits.mplot3d import Axes3D
    print("Plotting 3D trajectory...")
    n = 0
    trajectory = np.ndarray((self.nsteps,3))
    for fname in self.fname_list:
        wholedata = fits.open(fname)
        for i in xrange(1,len(wholedata)):
            data = wholedata[i].data
            trajectory[n,0] = data['R']
            trajectory[n,1] = data['T']
            trajectory[n,2] = data['P']
            n += 1
    fig = plt.figure()
    ax = fig.add_subplot(111,projection='3d')
    plotrange = self.state[0,0,0]+.25*self.state[0,0,0]
    print(plotrange)
    ax.set_xlim3d(-plotrange,plotrange)
    ax.set_ylim3d(-plotrange,plotrange)
    ax.set_zlim3d(-plotrange,plotrange)
    ax.scatter(trajectory[:,0]*np.sin(trajectory[:,1])*np.cos(trajectory[:,2]),
                trajectory[:,0]*np.sin(trajectory[:,1])*np.sin(trajectory[:,2]),
                trajectory[:,0]*np.cos(trajectory[:,1]))
    plt.show()
```

Appendix B

OINKSTools.py source code

```
"""
OINKSTools.py: Helper functions for OINKS.

Author: Alex Deich :: http://github.com/deichdeich
Date: February, 2016
"""
from __future__ import division
import os
import numpy128 as np

def elastic_collision(state,masses):
    raise ValueError("whoops!")


def dict_check(some_dict, thing):
    what_key= -1
    for key in some_dict:
        if thing in some_dict[key][1]:
            what_key = key
    return what_key


def get_filenum(dir,npart):
    nums = [0]
    for filename in os.listdir(dir):
        if(filename[:5] == "nbody"):
            pieces = filename.split("_")
            if int(pieces[-2]) == npart:
                nums.append(int(pieces[-1]))
    return(max(nums)+1)


def make_initial_conditions(nparticles,bh_m,a0):
    vecs = np.zeros((nparticles,2,3))
    for vec in xrange(nparticles):
        r_init = np.nan
        i=1
        while np.isnan(r_init):
         energy = np.random.normal(5,0.05)
         jz = np.random.normal(7,.05)
         r_init = (1/(8 * bh_m)) * (4*(a0**2) - (a0**2) * energy**2 + jz**2 + np.sqrt((-(a0**2) * (-4 +
energy**2) + jz**2)**2 - 48 * (-(a0**2) * energy + jz)**2 * bh_m**2))
         i+=1
         if i == 100000:
         raise ValueError("no real initial radius available")
        r_init += np.random.normal(0.5,0.25)
        phi = 2*np.pi*np.random.random(1)
        rd = 0
        phid = (pow(a0,5)*bh_m + 2*pow(a0,3)*bh_m*r_init*(-2*bh_m + r_init) + a0*bh_m*pow(r_init,
        2)*pow(-2*bh_m + r_init,2) - np.sqrt(-(pow(r_init,3)*pow(pow(a0,2) + r_init*(-2*bh_m + r_init),
        2)*(pow(a0,4)*bh_m*(-1 + pow(rd,2)) - pow(a0,2)*r_init*(2*bh_m*r_init + pow(r_init,2)*pow(rd,2)
        + pow(bh_m,2)*(-4 + pow(rd,2))) + bh_m*pow(r_init,2)*(-4*pow(bh_m,2) + 4*bh_m*r_init +
        pow(r_init,2)*(-1 + pow(rd,2)))))))/((pow(a0,2)*bh_m - pow(r_init,3))*pow(pow(a0,2) +
        r_init*(-2*bh_m + r_init),2))

        vecs[vec] = [[r_init, np.pi/2, phi],
                     [rd,0,phid]]
    return(vecs)


def a0(x):
        return(0)
```

References

Aarseth, S. J. (2003). *Gravitational n-Body Simulations*. Cambridge University Press.

Carroll, B. W., & Ostlie, D. A. (2006). *Introduction to Modern Astrophysics*. Pearson.

Chandrasekhar, S. (1976). *Mathematical Theory of Black Holes*. Oxford, 2 ed.

Damour, T., Sofel, M., Chongming, & Xu (1992). *Relativistic Gravity Research With Emphasis on Experiments and Observations: Proceedings of the 81 WE-Heraeus-Seminar Held at the Physikzentrum, Bad Honnef, Germany 2–6 September 1991*, chap. The general relativistic N-body problem, (pp. 46–69). Berlin, Heidelberg: Springer Berlin Heidelberg. `http://dx.doi.org/10.1007/3-540-56180-3_2`

Feynman, R. P. (1964). *Lectures in Physics*, vol. 2. Addison-Wesley.

Franklin, J. (2010). *Advanced Mechanics and General Relativity*. Cambridge University Press.

Franklin, J. (2013). *Computational Methods for Physics*. Cambridge University Press.

Garcia, P. J. V. (Ed.), (2011). *Physical Processes in Circumstellar Disks Around Young Stars*. University of Chicago Press.

Grøn, Ø., & Hervik, S. (2007). *Einstein's General Theory of Relativity with Modern Applications in Cosmology*. Springer.

Hut, P., & Makino, J. (2007). Moving stars around. `http://www.artcompsci.org/kali/pub/msa`

Jefremov, P., & Tsupko, O. Y. (2015). Innermost stable circular orbits of spinning test particles in Schwarzschild and Kerr space-times. *Physical Review D*, *83*, 222–229.

Lai, D., Chernoff, D. F., & Cordes, J. M. (2001). Pulsar Jets: Implications for Neutron Star Kicks and Initial Spins. *Astrophysical Journal*, *79*, 120–125.

Lyne, A., & Graham-Smith, F. (2012). *Pulsar Astronomy*. Cambridge University Press.

Misner, C. W., Thorne, K. S., & Wheeler, J. A. (1973). *Gravitation*. W. H. Freeman.

Petschek, A. G. (Ed.), (1990). *Supernovae*. Springer-Verlag.

Soummer, R., Perrin, M. D., Pueyo, L., Choquet, É., Chen, C., Golimowski, D. A., Hagan, J. B., Mittal, T., Moerchen, M., N'Diaye, M., Rajan, A., Wolff, S., Debes, J., Hines, D. C., & Schneider, G. (2014). Five Debris Disks Newly Revealed in Scattered Light from the HST NICMOS Archive. *Astrophysical Journal*, *87*, 340–344.

Visser, M. (2008). *The Kerr spacetime*, chap. The Kerr spacetime: A brief introduction. University of Wellington Press.

Zee, A. (1986). *Fearful Symmetry: The Search for Beauty in Modern Physics*. Princeton University Press.