UiO : Institute of Theoretical Astrophysics
University of Oslo

Depicting a Black Hole Merger: A Bridge Between Einstein Toolkit and GYOTO

Daniel Heinesen

Master's Thesis, Spring 2021

Acknowledgements

A special thanks goes to Éric Gourgoulhon and Frederic Vincent from *Observatoire de Paris* in Meudon, Paris. Without them this project wouldn't be possible. Throughout two week in Paris, and countless Zoom conferences they have patiently listened and answered all the stupid questions I have had about LORENE and GYOTO. The trip to Paris would not have been posible without the grant given to me as a STSM by GWVerse. Thanks to my main supervisor David. F. Mota, who didn't give up on me during these one and a half years with ups and downs. Thanks also to Vitor Cardoso who gave me the idea for this thesis, and who have helped me a lot during this project. Thank you to Miguel Zilhão for helping me out with all the nuances of Einstein Toolkit. A huge thanks goes out to Øyvind Christiansen for reading through and correcting all the terrible spelling and grammar mistakes found in the thesis. Last, but not least I will like to give maybe the biggest thanks to my girlfriend Laila Andersland for keeping up with me during these stressful month – and I promise to start helping around the house again now that the thesis is done.

Contents

List of Figures

Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?

Kernighan's Law

Abstract

In this project we create a conversion tool for using numerical general relativity simulations done in Einstein Toolkit in the numerical relativity ray tracer tool *General relativitY Orbit Tracer of Observatoire de Paris*(GYOTO). Due to the different formalisms used by Einstein Toolkit and GYOTO, the numerical relativity framework *Langage Objet pour la RElativité NumériquE* (LORENE) was used to bridge between the two. Simulations of a isotropic Schwarzschild metric and a binary black hole merger initialized with a two puncture method were used as test data to evaluate the tool. The spectral formalism used by LORENE has a spherical topology, meaning that a custom splitting function had to be devised to split the binaries into separate, pseudo-spherical spacetimes. We used Python to read and interpolate the data from Einstein Toolkit, and to call on a separate C-code using LORENE to take care of the spectral transformation. We showed that the process of reading and interpolating the Einstein Toolkit data will lead to numerical errors at order 10^{-5}. This error was also manifest in the final ray tracing. With proper parameterization, the spectral transformation did not show any major additions of error, and could in some cases smooth out errors. We showed that it was possible to transform the black hole binary, but at a much greater computational cost. Results without using the splitting function was also shown to lead to artifacts in the transformed metric. We used drift in the photon momentum norm from a base line of zero to measure errors in the ray tracing. With this we showed that the numerical errors when using the Schwarzschild metric in GYOTO was about $0-10^3$ times higher than ray tracing done with a standard metric made with LORENE, which only was an error increase of 10^{-3} in absolute terms. To use the black hole binary metric in GYOTO, small changes to GYOTO are still needed, and we weren't able to do ray tracing using this metric. We conclude that by using our tool we successfully used GYOTO with a single black hole metric simulated in Einstein Toolkit, and that the tool is capable of transforming black hole binary metric for when this function is implemented in GYOTO.

The code for this project can be found at the repository: https://github.com/dulte/Master

Chapter 1

Introduction

There are few things that catch the imagination more than black holes. For astrophysicists to science fiction authors, black holes have become a staple of the outer limits of physics. They were first proposed back in 1784 by the English clergyman John Michell as an idea, but it wasn't before 1915 that Karl Schwarzschild became the first to solve Albert Einstein's field equation – only a few months after Einstein had published them – and mathematically describe a black hole. For many years, or even decades, black holes were objects which should exist, but were theoretical, exotic objects, which by their nature made them almost impossible to observe.

Almost impossible doesn't mean actually impossible. In 1964 the object *Cygnus X-1* was discovered[12]. After careful observations over many years, it was concluded that this objects most likey must be a black hole[47][1]. One of the most famous observations of a black hole was published in 1998[14], when observations of objects orbiting Sagittarius A*, the object in the center of the Milky Way, showed that Sagittarius A* must be a black hole. This was evidence, but still indirect evidence.

The last 6 years have been a golden age for black holes. In 2015 the first detections of gravitational waves were made[2]. These are waves made by astrophysical objects colliding and making waves in spacetime itself. From the data found in these detections it was proved that the collision must have been between two black holes. This was an even greater evidence that black holes, and especially binary black holes orbiting each other, existed! An even more definitive proof came only four years later when, in April 2019, the Event Horizon Telescope announced that they had managed to take the first direct image of a black hole! While it is impossible to prove that black holes exist[16], these observations strongly points to their existence.

Neither the image nor the detection of gravitational waves came as a shock to scientific community. The LIGO project, which detected the gravitational waves had its start back the the 1960s, and in 1968 the physicist Kip Thorne laid the theoretical ground work for the detection.

Researchers wanted to understand what they were looking for, and for this they

[1] There was a famous bet between Stephen Hawking and Kip Thorne on whether or not this actually was a black hole. Hawking, who bet against to being a black hole, was deemed to have lost the bet.

had to find more complicated solutions of Einstein's field equations than Schwarzschild had found. They quickly understood that this wasn't possible to do with pen and paper, and they needed help from computers. This gave birth to the field of numerical relativity. The field was started in the 1960s but didn't come to maturity before the 1980s and 1990s. The reason was two fold: Firstly the computational power increased drastically in this time period, meaning that now they had the computational power needed to solve the field equations numerically.

The other reasons was that solving the equations turned out to be incredibly difficult. When trying to solve Einstein's field equations numerically for simple binary black holes, the computations often crashed or diverged suddenly. Throughout 1980s and 1990s the researchers found that they had to reshape the field equations using some fancy mathematical tricks.

In the 2000s they had finally found ways of making the simulations stable, and computational power had increased enough that it was possible to simulate complicated binary black holes on even personal computers. This means that you can simulate the collision of black holes emitting gravitational waves on a work station – although it will take some time. This can be done with the powerful framework *Einstein Toolkit*, which is an open source framework which lets everyone run numerical relativity simulations at home. We will be looking a lot at this framework during the thesis.

We did also know what to expect when we first were able to take an image of a black hole. Kip Thorne also have a hand in this subjects, and even made calculations for how a black hole will look like for the movie *Interstellar*. These give approximate solutions for how a black hole will look like. We can also use a numerical method called ray tracing. With this method we send photons towards the a black hole and look at how the photons move. This lets us not only make pretty images, but also look at the physics of the photons as they travel close to the black hole.

What we want is to combine the research fields above. We want to look at photons travelling close to two black holes colliding and creating gravitational waves. This is not an easy task, as these collisions are very difficult to describe with pen and paper. We will be using the aforementioned Einstein Toolkit to simulate the collision, and another program called GYOTO to do the ray tracing, as this is one of the few programs that let us do ray tracing on simulated spacetimes. This will let look at how light behaves around these most extreme objects in the universe.

There is a major hurdle in this task. GYOTO is built upon another numerical relativity framework, LORENE, which used a simulation formalism different for that of Einstein Toolkit. This means that the main task we will be looking at is bridging the gap between the two formalism.

LORENE has built in a spherical topology. This means that it used a spherical coordinate system, meaning that objects deviating too much from this topology, e.g. a black hole binary, will be represented poorly by this formalism. It has the capability to do binary black holes, but this is limited to quasi-circular orbits. If we want to look at the merger itself, then we cannot use LORENE. We will work around this by introducing a splitting function which will separate the binary into separate single black holes. This will enforce a pseudo-spherical topology.

A black hole binary is complicated to get to work on our first try. We will therefore focus on taking a single Schwarzschild black hole from Einstein Toolkit to GYOTO. This lets us try out all the different parts of the method. This is not a trivial results in of itself, and managing to get this Einstein Toolkit simulation to GYOTO would be a good result, and leaves the door open for more simulations done in Einstein Toolkit to be used in GYOTO.

By the end of the thesis we hope to have a working pipeline which can take spacetime simulated in Einstein Toolkit and make it usable for GYOTO. To get the black hole binary to work with GYOTO and to be able to study the effects of gravitational waves might not be possible within the scope of this thesis, but we will lay the ground work, so that either we or other researchers may finish that project later. This will let researchers look at the effects of mergers and the resulting gravitational waves on the passing photons in more detail that other previous approximations have done[20].

1.1 Overview of Thesis

We will start by looking at how to reformulate general relativity in such a way that we can solve the equations numerically. We will then look at Einstein Toolkit and LORENE, two programs/frameworks which we will use to do the simulations. Next we will look at ray tracing, both in the conventional sense and using numerical relativity, and how we can use GYOTO to do this. This will lead us to the methods. Here we will describe how to configure and run both Einstein Toolkit and GYOTO. The main part of the method section will consist of looking at how to convert the Einstein Toolkit data to a formalism which we can use in GYOTO. This will consist of reading and interpolating the data, getting the collocation points needed to do the spectral transformation, handling the symmetries and boundaries of the data and then handing the data over to LORENE for the actual transformation. For the black hole binary we will also need to apply a splitting function, so we will look at this function. We will then evaluate the result of the most crucial steps of the conversion: The reading and interpolation, the spectral transformation and the ray tracing. Finally we will make a conclusion on whether we succeeded in using a Einstein Toolkit metric in GYOTO, and discuss future work.

Part I

Theory

Chapter 2

Worlds Shortest Primer on General Relativity

In this section we assume a familiarity with general relativity. If the reader has problems following along, I recommend reading up on the subject in either [18], or [40] if they are feeling adventures. For those who aren't I will present what I think is the worlds shortest primer on general relativity, what it is and the simplest types of solutions.

Our best understanding of gravity comes from the general theory of relativity as described by Albert Einstein[22]. Combining space and time into a four-dimensional manifold he, he showed that energy – and this matter – bends spacetime. Matter, moving on geodesics, will be affected by this curvature, thus giving rise to gravitation [1]. This relation can be described using the infamous Einstein field equations

$$R_{\mu\nu} - \frac{1}{2}Rg_{\mu\nu} = 8\pi G T_{\mu\nu}, \tag{2.1}$$

where $g_{\mu\nu}$ is the metric tensor, which describes the manifold – more specifically infinitesimal distances on said manifold –; $R_{\mu\nu}$ is the Ricci tensor, and describes the curvature of the manifold; $R = R^{\mu}_{\mu}$ is the Ricci scalar; and $T_{\mu\nu}$ is the energy-momentum tensor, which describes the energy and matter content of space. (2.1) is a set of highly non-linear 2nd order differential equations.

Our goal is to be able to send photons around one or more black holes. This means we have to be able to describe a black hole. This is where (2.1) comes in. If we manage to solve this differential equation we get a description of a black hole. When we have a metric tensor that solves this equation, we can use this to describe how light behaves near the black hole. We will look at how this is done in sec. 5.

Actually solving these equations are very difficult. All but the simplest cases can be solved analytically. We will look at one of the simplest solutions below, namely the Schwarzschild solution.

Note that we can get a lot of the field equations without solving them completely. A very powerful way of looking at solutions to the field equations are perturbation the-

[1] Meaning that gravitation is no longer described as a force, but as a geometrical effect.

ory, where we don't try to find complete solutions but instead look at already known solutions and try to change them a small amount. We will look briefly at the simplest perturbation method in sec. 3.7, which gives us gravitational waves. For more complicated methods, like *Post-Newtonian expansions*, see [37] and [40].

2.1 Empty Space and Black Holes

So what is the simplest solutions for the field equations(2.1)? Well, we can start by saying that we have empty space, in other words vacuum. This means that $T_{\mu\nu} = 0$, since we have no energy[2]. This can only be true if the Ricci tensor $R_{\mu\nu} = 0$.

The simplest vacuum solution is just flat space

$$g_{\mu\nu} = \eta_{\mu\nu} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.2}$$

This is the Minkowski metric, which describes flat space in special relativity. It isn't much to look at, and isn't very interesting when looking at general relativity. We will use it later in the thesis, since it is excellent to use as a test case when ray tracing, since we expect nothing to happen, meaning that if something happens it will be due to numerical error. But we will look into that in sec. 8.7.

We will now look at the solution that might be the most known, namely that of a black hole. So, yes, a black hole is a vacuum solution of the field equations, or at least the normal black hole solutions like Schwarzschild and Kerr solutions. We will look at the Schwarzschild solution here, and then mention the Kerr solution in sec. 5.3.

The Schwarzschild solution is a vacuum solution where we assume the maximum degree of symmetry in spacetime. The metric then takes the form

$$g_{\mu\nu} = \begin{pmatrix} -(1 - r_s/r) & 0 & 0 & 0 \\ 0 & (1 - r_s/r)^{-1} & 0 & 0 \\ 0 & 0 & r^2 & 0 \\ 0 & 0 & 0 & r^2 \sin\theta \end{pmatrix}, \tag{2.3}$$

where $r_s = \frac{2GM}{c^2}$ is the Schwarzschild radius and is an event horizon, meaning that it is the radius from which nothing, not even light can escape.

We will use this metric a lot through out the thesis. But we will not look at the metric in exactly this form. The matrix representation of the metric is not unique, it is instead a result of the coordinates we choose. In (2.3) we have used the so called Schwarzschild coordinates. We are free to change these, and will instead use them in

[2]This is not entirely correct since we have gravitational energy

the so called *isotropic coordinates*. This gives us

$$g_{\mu\nu} = \begin{pmatrix} -\frac{(1-r_s/4R)}{(1+r_s/4R)} & 0 & 0 & 0 \\ 0 & (1-r_s/4R)^4 & 0 & 0 \\ 0 & 0 & (1-r_s/4R)^4 & 0 \\ 0 & 0 & 0 & (1-r_s/4R)^4 \end{pmatrix}, \tag{2.4}$$

where $R = \sqrt{x^2+y^2+z^2}$ and the event horizon now is $r_s/4$. This metric will be used for comparison later in the thesis.

2.2 Two Black Holes

We can then ask what will happen when we use two black holes and let them collide. This will not only lead to some interesting visuals, but also gravitational waves (see sec. 3.7). The problems is that this becomes vastly more difficult. It is possible to make approximations of the period before and after a collision using different approximations, see [37] and [38], but for the collision itself it is more or less impossible. We must instead look to computers to do the job for us. And this is what we will look at in the next section.

Chapter 3

Numerical Relativity and the 3+1 formulation

We have seen that only in highly symmetric cases, like a single black hole, is it possible to solve the field equations analytically. It is possible to use perturbation theory to obtain approximate solutions for gravitational waves [23]. With quite sophisticated tools, like *post-newtonian expansion* it is even possible to find the gravitational waves emitted by merging black holes[37][38]. But if we want the full story of the merger of two black holes such approximations wouldn't be enough: They break down as the black holes actually merge. So there is a need for some method to solve Einstein's field equations (2.1) outside of these idealized situations. It is here numerical relativity comes to the rescue!

3.1 Why a New Formalism?

The goal of numerical relativity is to be able to use computers to solve the field equations (2.1). But does one do that? Equation 2.1 consists of derivatives of the metric $g_{\mu\nu}$ over both space and time. It therefore sounds like we have a Cauchy problem

$$\frac{\partial^n u_i}{\partial t^n} = F\left(t, x_j, u_j, \frac{\partial^k u_i}{\partial x_j^k}\right). \tag{3.1}$$

A Cauchy problem is the normal way of writing a general *partial differential equation*, and something we know how to solve[51]. The problem is the our equation(s) 2.1 treats space and time on equal footing – as a four dimensional spacetime. There is no easy ways of writing it into the form of (3.1). There have been efforts to actually solve (2.1) in this form, but this has been only partially accomplished[45]. We therefore need to reformulate general relativity á la Einstein into a something on the Cauchy form! The most popular form of doing this is the so called 3+1 formalism. This formalism was first formulated by Richard Arnowitt, Stanley Deser and Charles W. Misner in 1959 [21] in an effort to get a Hamiltonian for the use in quantum gravity. James York

then rewrote the formulation in 1979 into the one we use today[1]. This formulation, as we shall see below, was on the form of a Cauchy problem, and it was not long before researchers began to use this formulation to simulate the collapse of stars, black hole mergers, and more. This is why the normal formulation of 3+1 numerical relativity is called the ADM formulation. It soon became clear that this formulation alone wasn't well suited for simulations, since it's mostly weakly hyperbolic[1], meaning that the equations that needed to be solved wasn't stable, in most cases leading to instabilities and divergences. As it became clear that the Laser Interferometer Gravitational-Wave Observatory (LIGO) was beginning to get founding and that the hunt for gravitational waves was nearing, the field of numerical relativity exploded in the late 80's and 90's, and people began to try to find ways of stabilizing the equations of the ADM formalism. Over the year multiple methods of achieving this has be proposed, like the Z4 formulation [10] and the BSSN formulation developed by Thomas W. Baumgarte, Stuart L. Shapiro, Masaru Shibata and Takashi Nakamura in the years 1987-1999. The latter formalism is by far the most popular, and is the one used in most numerical relativity codes today. It is ADM, with the addition of BSSN, we will discuss below.

3.2 The 3+1 Formalism

This is meant as an heuristic introduction to the subject. I will not cite every assumption and steps not explicitly calculated. Thorough derivations can be found in most textbooks on the subjects. This introduction are based on the books from Gourgoulhon[27], Baumgarte & Shapiro [6][7] and Alcubierre [3], and the wonderful lectures given on the subject by Dennis Pollney at the Chris Engelbrecht Summerschool 2020 [44][2]. It is first and foremost Baumgarte & Shapiro I'm basing derivations on, meaning that the notation and order of the derivation is taken from here. I've changed some of the order of the derivation when I found it more pedagogical to do so.

3.2.1 The New Quantities of the 3+1 Formalism, and the First Evolution Equation

As we saw above, one of the problems we had with the normal formulation of general relativity is that time and space is treated as one. We therefore begin with splitting them up. This is done by *foliating* spacetime into discrete 3-dimensional, space-like hypersurfaces, denoted Σ_t. Note that two observers on different parts of one slice don't have to agree on the proper time [3]. The important point is that we have sliced spacetime into non-intersecting 3-dimensional slices. The slices as constants of a global function t, which turns out to be time. Now we have 3d slices living in a time t, therefore a 3+1 formalism.

We now need a way to describe the slices Σ_t. We start be defining a 1-form

[1] To prove this is actually quite hard, and has only been done in specific cases.

[2] This page with the notes has sadly been taken down in the last half a year. I will not use any more citations from this, since it is based on [6].

[3] As one has come to expect in relativity.

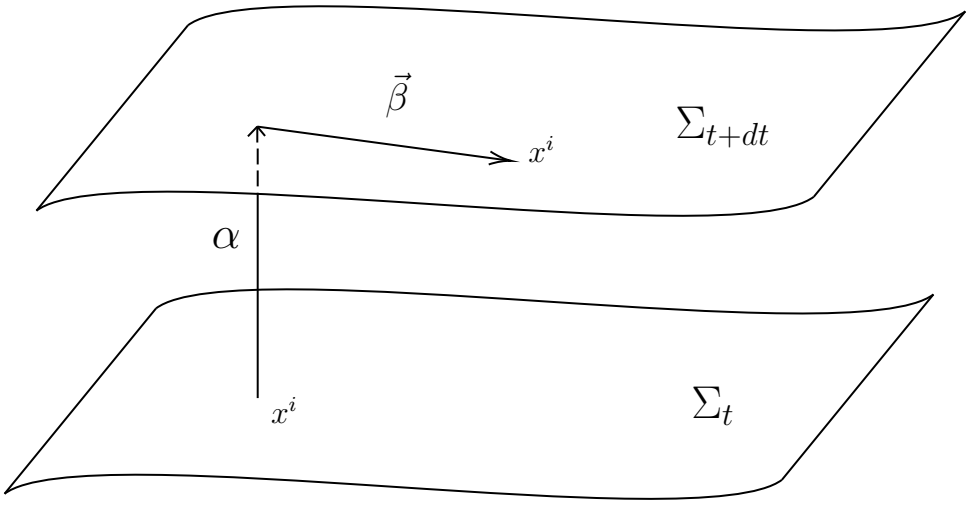


Figure 3.1: Figure illustrating the meaning of the terms defined in the text. Here we have two time sliced Σ_t and Σ_{t+dt} with a given point x^μ. We can see that the lapse function α is the separation in time between the two slices, while the shift vector $\vec{\beta}$ is the distance x^i is shifted in space between the two slices.

$$\Omega_\mu = \nabla_\mu t. \tag{3.2}$$

This is a vector normal to the slice. We then get a normal unit vector to the slice

$$n_\mu = \alpha \Omega_\mu. \tag{3.3}$$

If we say that our function t is time, it is easy to show that

$$\alpha^2 = -\frac{1}{g^{00}}, \tag{3.4}$$

meaning that α describes how time evolve at a given position on the slice. α is what is called *the lapse function*, and is one of the fundamental quantities used in 3+1 formalism. Figure 3.1 shows the interpretation of Σ_t and α together with another quantity $\vec{\beta}$ which will be introduced later.

Before moving on with further description of the slices, we need to know how a tensor is represented in the slices. It is possible to introduce a timelike projection operator with the help of the unit normal vector

$$N^\mu_\nu = n^\mu n_\nu, \tag{3.5}$$

and a spacelike projection operator

$$P^{\mu}_{\nu} = \delta^{\mu}_{\nu} + n^{\mu} n_{\nu}. \tag{3.6}$$

The projection operators are going to help us immensely: We can now take the Einstein equations (2.1) and project different indices into spacelike or timelike parts. This will give us the type of equations we are looking for.

Having this unit normal vector and the projection operators, we can start looking into the curvature of the slices. We need to distinguish between two types of curvature: *Intrinsic* and *extrinsic* curvature. Intrinsic curvature is the curvature we are used to from GR á la Einstein, described by the Riemann tensor. It describes how a vector on the slice changes if it is parallel transported. Since the Riemann tensor is given by just the metric and its first derivatives, we need only to find the metric for the slices. This can be done by simple projecting the metric onto the slices

$$\gamma_{\mu\nu} = P^{\sigma}_{\mu} P^{\tau}_{\nu} g_{\sigma\tau} = g_{\mu\nu} + n_{\mu} n_{\nu}. \tag{3.7}$$

Notice that this is an 3d tensor, and is the metric of the slices and is sufficient to describe the intrinsic curvature.

In our description of GR, extrinsic curvature also plays an important role. Where the intrinsic curvature described the change of a vector on the slice, the extrinsic curvature describes the change of the unit normal vector as it is parallel transported around in the slice. This is a symmetric tensor given as

$$K_{\mu\nu} = -P^{\alpha}_{\mu} \nabla_{\alpha} n_{\nu} = -\nabla_{\mu} n_{\nu} - n_{\mu} a_{\nu}, \tag{3.8}$$

where $a_j = n^{\alpha} \nabla_{\alpha} n_{\nu}$ is called the acceleration. Notice that $K_{\mu\nu}$ is described completely by n_{μ}. With (3.7) and (3.8) we are now able to describe the 3d slices.

We can note here that we are left with two things we are free to choose, namely how we divide up spacetime into slices, and how we travel through them. These freedoms lay in α and the time vector t^{μ}. If we rewrite

$$t^{\mu} = \alpha n^{\mu} + \beta^{\mu}, \tag{3.9}$$

where $\vec{\beta}$ is the *shift vector*. The reason we rewrote this is that now we are guaranteed that t^{μ} is dual to the surface 1-form Ω, and this gives us the free parameters α and $\vec{\beta}$. The interpretation is that α tells the temporal distance between two slices, while β tells us how the spatial coordinates changes from slice to slice. Fig. 3.1 gives an illustration of this. Since these two quantities are free parameters they are treated as gauge choices.

The four quantities α, $\vec{\beta}$, $g_{\mu\nu}$ and γ_{ij} are enough to describe the whole manifold, and are what we are evolving to simulate GR. Having a way of describing spacetime as a set of 3D hypersurfaces is great, but we do not know yet how they interact with matter. We now need to find equation that describe this, and how the slices will evolve with time!

The easiest evolution equation to start with is that of the spatial metric. We'll find this by taking the Lie derivative of the spatial metric along the normal vector

$$\pounds_{\vec{n}}\gamma_{\mu\nu} = n^\sigma \nabla_\sigma \gamma_{\mu\nu} + \gamma_{\sigma(\mu}\nabla_{\nu)} n^\sigma = \cdots = \gamma^\sigma_\mu \nabla_\sigma n_\nu + \gamma^\sigma_\nu \nabla_\sigma n_\mu = -2K_{\mu\nu}. \tag{3.10}$$

For the last step we have used metric compatibility (its covariant derivative is zeros), and that the acceleration is zeros. We now have an expression for the evolution of γ_{ij}, but it is inside a Lie derivative, so we need to deal with that.

A feature of the ADM equations is that they are expressed in a certain set of basis vectors $e^a_{(i)}$, with $i = 1, 2, 3$. They are defined such that lay on a slice

$$\Omega_j e^j_{(i)} = 0. \tag{3.11}$$

The time basis vector we choose to be out time vector (3.9), we get that

$$t^\mu \Omega_\mu = \alpha n^\mu \Omega_\mu + \beta^\mu \Omega_\mu = 1 \Rightarrow t^\mu = (1, 0, 0, 0). \tag{3.12}$$

This alone actually gives us that

$$\pounds_t = \partial_t. \tag{3.13}$$

We can also use eq. 3.3 and 3.11 and write

$$\Omega_\mu e^\mu_{(i)} = -\frac{1}{\alpha} n_\mu e^\mu_{(i)} \Rightarrow n_i = 0. \tag{3.14}$$

Since we know that $\vec{\beta}$ is spatial, we can use 3.9 to show that

$$n^\mu = \frac{1}{\alpha}\left(t^\mu - \beta^\mu\right) = \left(\frac{1}{\alpha}, \frac{\beta^i}{\alpha}\right). \tag{3.15}$$

From this we get a reformulation of the metric in 3+1 formalism

$$\boxed{g_{\mu\nu} = \gamma_{\mu\nu} + n_\mu n_\nu = \begin{pmatrix} -\alpha + \beta_l \beta^l & \beta_i \\ \beta_j & \gamma_{ij} \end{pmatrix}.} \tag{3.16}$$

Now we come back to the Lie derivative of the spatial metric (3.10). Since we have a set of basis vectors, it is possible to rewrite this in a way that we know how to solve. Looking at the Lie derivative in (3.10)

$$\pounds_{\vec{n}}\gamma_{\mu\nu} = \frac{1}{\alpha}\left(\pounds_t - \pounds_{\vec{\beta}}\right)\gamma_{\mu\nu} \tag{3.17}$$

with

$$\pounds_{\vec{\beta}}\gamma_{\mu\nu} = \beta^k \partial_k \gamma_{ij} + \gamma_{kj}\partial_i \beta^k + \gamma_{ik}\partial_j \beta^k. \tag{3.18}$$

One can show that if the connection is symmetric, one can exchange the partial derivative in the definition of the Lie derivative for a covariant derivative [6]. In our case, since we are working on our 3D hypersurface, let $\partial_i \to D_i$, where D_i is the covariant derivative on said hypersurface, given by γ_{ij}. Thus we get

$$\pounds_{\vec{\beta}}\gamma_{\mu\nu} = \beta^k D_k\gamma_{ij} + \gamma_{kj}D_i\beta^k + \gamma_{ik}D_j\beta^k = D_i\gamma_{kj}\beta^k + D_j\gamma_{ik}\beta^k = D_{(i}\beta_{j)}, \tag{3.19}$$

where we have used metric compatibility $D_k\gamma_{ij} = 0$. We can now see that we are left with a nice expression for (3.10)

$$K_{\mu\nu} = \frac{1}{2\alpha}\left(-\partial_t\gamma_{\mu\nu} + D_{(i}\beta_{j)}\right),$$

where we only care about the spatial part. This gives us

$$\boxed{\partial_t\gamma_{ij} = -2\alpha K_{ij} + D_{(i}\beta_{j)}.} \tag{3.20}$$

This is our first evolution equation, and determines how the spatial metric evolve with time. This is the first of equation governing the evolution of our system.

3.2.2 The Constraint Equations and the Second Evolution Equation

To find the rest of the equations needed to do the evolution, it is no surprise that we need to do some rewriting of Einsteins field equations (2.1). The rewriting will consist of doing projections of the field equation. Before we look at the whole set of equations, we will start be applying the projections to the Riemann tensor.

We have two types of projections, one spasial projection and one timelike projection. As we saw above, they are given by eq. 3.6 and eq. 3.5. Each time we apply one of these projection operators, they will project only one index. This means that there are multiple ways we can do the projection of all the indices. One can show that, due to the antisymmetrical properties of the Riemann tensor, there are only three unique ways of doing the projections

$$P^\sigma_\alpha P^\tau_\beta P^\gamma_\mu P^{\delta\,(4)}_\nu R_{\sigma\tau\gamma\delta}, \tag{3.21}$$

$$n^\delta P^\sigma_\alpha P^\tau_\beta P^{\gamma\,(4)}_\mu R_{\sigma\tau\gamma\delta}, \tag{3.22}$$

$$n^\tau n^\delta P^\sigma_\alpha P^{\gamma\,(4)}_\beta R_{\sigma\tau\gamma\delta}, \tag{3.23}$$

where ${}^{(4)}R_{\sigma\tau\gamma\delta}$ is the 4D Riemann tensor – in contrary to the 3D Riemann tensor, which will be used for the 3+1 equations. Doing the actual calculations are not difficult nor technical, but involves a lot of tensor index manipulation, and a lot of time and concentration. A good derivation can be found in [6], or done with Mathematica [44]. After doing the projections we are left with three equations

$$P^\sigma_\alpha P^\tau_\beta P^\gamma_\mu P^{\delta\,(4)}_\nu R_{\sigma\tau\gamma\delta} =^{(3)} R_{\alpha\beta\mu\nu} + K_{\alpha\mu}K_{\beta\nu} + K_{\alpha\nu}K_{\beta\mu}, \tag{3.24}$$

$$n^\delta P^\sigma_\alpha P^\tau_\beta P^{\gamma\,(4)}_\mu R_{\sigma\tau\gamma\delta} = D_\beta K_{\alpha\mu} - D_\alpha K_{\beta\mu}, \tag{3.25}$$

$$n^\tau n^\delta P^\sigma_\alpha P^{\gamma\,(4)}_\beta R_{\sigma\tau\gamma\delta} = \pounds_{\vec{n}}K_{\alpha\beta} + \frac{1}{\alpha}D_\alpha D_\beta\alpha + K^{\sigma\beta}K_{\alpha\sigma}. \tag{3.26}$$

We now have projections for the Riemann tensor, which will help us out when looking at the projections of the field equations.

One more thing we have to look at before tackling the field equations are the energy-momentum tensor $T_{\mu\nu}$. This will be done by decomposing the tensor into a energy density

$$T^{00} = \rho = n^\mu n^\nu T_{\mu\nu}, \tag{3.27}$$

the momentum density

$$T^{0i} = T^{i0} = j^i = P^i_\mu n_\nu T^{\mu\nu} \tag{3.28}$$

and the spatial stress tensor

$$T^{ji} = T^{ii} = S^{ij} = P^i_\mu P^j_\nu T^{\mu\nu}. \tag{3.29}$$

Now we have all we need to do the projection of the field equations.

We can now move on to look at the field equations. We start by projecting the Einstein Tensor with the use of the normal vector

$$2n^\mu n^{\nu\,(4)}G_{\mu\nu} = 2n^\mu n^\nu \left({}^{(4)}R_{\mu\nu} - \frac{1}{2}{}^{(4)}Rg_{\mu\nu}\right) = 2n^\mu n^{\nu\,(4)}R_{\mu\nu} +^{(4)} R. \tag{3.30}$$

With this in mind, we can try to apply the spatial metric to Gauss' equation 3.24

$$\gamma^{\beta\nu}P^\sigma_\alpha P^\tau_\beta P^\gamma_\mu P^{\delta\,(4)}_\nu R_{\sigma\tau\gamma\delta} = \gamma^{\tau\delta}P^\sigma_\alpha P^{\gamma\,(4)}_\mu R_{\sigma\tau\gamma\delta} =^{(3)} R_{\alpha\mu} + K_{\alpha\mu}K + K^\sigma_\alpha K_{\sigma\mu}, \tag{3.31}$$

where the first step comes from the fact that the spatial metric operated on by two projection operators yields the spatial metric. We then do a contraction again

$$\gamma^{\alpha\mu}\gamma^{\tau\delta}P^\sigma_\alpha P^{\gamma\,(4)}_\mu R_{\sigma\tau\gamma\delta} = \gamma^{\sigma\gamma}\gamma^{\tau\delta\,(4)}R_{\sigma\tau\gamma\delta} =^{(3)} R + K^2 - K_{\mu\nu}K^{\mu\nu}. \tag{3.32}$$

If we look at the equation in the second step, we see that this can be written as

$$\gamma^{\sigma\gamma}\gamma^{\tau\delta\,(4)}R_{\sigma\tau\gamma\delta} = (g^{\sigma\gamma\sigma}n^\gamma)(g^{\tau\delta} + n^\tau n^\delta)^{(4)}R_{\sigma\tau\gamma\delta} = 2n^\mu n^{\nu\,(4)}R_{\mu\nu} +^{(4)} R. \tag{3.33}$$

Combining (3.30), (3.32) and (3.33) we see that we get

$$2n^\mu n^{\nu\,(4)}G_{\mu\nu} = 2n^\mu n^{\nu\,(4)}R_{\mu\nu} +^{(4)} R =^{(3)} R + K^2 - K_{\mu\nu}K^{\mu\nu}. \tag{3.34}$$

Looking at the Einstein tensor, we know from the field equations that $G_{\mu\nu} = 8\pi T_{\mu nu}$. We also know from (3.27) that $n^\mu n^{\nu\,(4)}G_{\mu\nu} = 8\pi\rho$. This leaves us with

$$\boxed{{}^{(3)}R + K^2 - K_{\mu\nu}K^{\mu\nu} = 16\pi\rho.} \tag{3.35}$$

This is an equation governing the spacetimes we want to simulate. But there are no time derivatives here, meaning that this isn't an evolution equation. This is instead a constraint equation, meaning that this differential equation must hold at all time.

Since is a differential equation determined by the energy density, this is called the *Hamiltonian Constraint*.

We can now do the similar procedures with one timelike and one mixed projection, and using the Codazzi-Mainardi equation 3.25 and the Ricci Equation 3.26, arrive at two more equations:

$$\boxed{D_\mu K - D_\nu K^\nu_\mu = 8\pi j_\mu} \tag{3.36}$$

and

$$\boxed{\begin{aligned}\partial_t K_{ij} = &-D_i D_j \alpha + \alpha\left({}^{(3)}R_{ij} - 2K_{ik}K^k_j + KK_{ij}\right)\\ &-8\pi\alpha\left(S_{ij} - \frac{1}{2}(S-\rho)\right) + \left(\beta^k D_k K_{ij} + D_{(i}\beta_{j)}\right).\end{aligned}} \tag{3.37}$$

As can be seen, this has a time derivative, so this is the *evolution equation for the extrinsic curvature*. Note that it is due to our choice of coordinates that a partial derivative appears in (3.37) instead of a Lie derivative.

This leaves us with a system we can solve. We have two evolution equations (3.20) and (3.37) and two constraint equations, (3.35) and (3.36). These are called the *ADM equations*.

Notice that (3.20) and (3.37) and contain partial differential operators time separate from the spatial derivatives. This means that we can write the equations on the form needed to have a Cauchy problem (3.1)! This means that we have made the mixed derivatives of the field equations into something that we know how to solve numerically! This is what we were looking for when switching to the 3+1 formalism!

But if we now try to solve these equations numerically, we will see that they are very unstable. This is because they at best are strongly hyperbolic in certain ideal situation and at worst weakly hyperbolic or elliptic in the scenarios we want to look at. This is where the BSSN formalism comes in, which we will look at next.

3.3 The BSSN Formulation

From the conception of the ADM equations (3.37), (3.20), (3.35) and (3.36), researchers were plagued with instabilities in the simulations, and all but the most ideal systems were impossible to simulate. This was because the equations are at best weakly hyperbolic, meaning that they are inherently unstable. As mentioned before, there are a lot of proposed solutions to this problem, but the most popular is the one we are going to explain here: The Baumgarte-Shapiro-Shibata-Nakamura (BSSN) formalism.

So how does one go about stabilizing the equations? We want to have strongly hyperbolic equations. The hyperbolicity of the equations are determined by the higher derivatives. In our case, both the evolution equations and the constraint equations contain second derivatives of the gravitational fields. This is why many reformulation schemes introduce new variables, consuming the first derivative. This leaves schemes where we have first derivatives of these new variables, instead of second derivatives of

the gravitational field. This is the case with BSSN. Other formalisms, like Z4, have extra variables, Z_μ, which do not absorb the first derivatives of the metric, but still contribute in the principle part, counteracting the effect of the second derivative of the metric [10].

We start by first rewriting the extrinsic curvature as a sum of its trace and traceless part

$$K_{ij} = A_{ij} + \frac{1}{3}\gamma_{ij}K. \tag{3.38}$$

We will also take the determinants of (3.20) and (3.37), giving us the rewritten evolution equations

$$\partial_t \ln \gamma^{1/2} = -\alpha K + D_i\beta^i \tag{3.39}$$

and

$$\partial_t K = -D^2\alpha + \alpha\left(K^{ij} + 4\pi(\rho + S)\right) + \beta^i D_i K. \tag{3.40}$$

We can now start the actual reformulation.

We start be doing a conformal rewriting of the the spatial metric and the traceless part of extrinsic curvature. We are allowed to rewrite the metric as

$$\bar{\gamma_{ij}} = e^{-4\phi}\gamma_{ij}, \tag{3.41}$$

where ϕ is the conformal factor, and $\bar{\gamma_{ij}}$ is the conformal metric, were we require that the determinant $\bar{\gamma} = 1$. We can do the same with the tracefree part of the curvature

$$\bar{A_{ij}} = e^{-4\phi}A_{ij}. \tag{3.42}$$

If we now take the trace of (3.40) and (3.39) we get the evolution equations of K and $\bar{\gamma}$, and if we subtract these from (3.20) and (3.37) we get evolution equations for $\bar{\gamma_{ij}}$ and $\bar{A_{ij}}$. They are

$$\boxed{\partial_t\phi = -\frac{1}{6} + \beta^i\partial_i\phi + \frac{1}{6}\partial_i\beta^i,} \tag{3.43}$$

$$\boxed{\partial_t K = -\gamma^{ij}D_jD_i\alpha + \alpha\left(\bar{A_{ij}}\bar{A^{ij}} + \frac{1}{3}K^2\right) + 4\pi\alpha\left(\rho + S\right) + \beta^i\partial_i K,} \tag{3.44}$$

$$\boxed{\partial_t\bar{\gamma_{ij}} = -2\alpha\bar{A_{ij}} + \beta^k\partial_k\bar{\gamma_{ij}} + \bar{\gamma_{ik}}\partial_j\beta_k + \bar{\gamma_{kj}}\partial_i\beta_k - \frac{2}{3}\bar{\gamma_{ij}}\partial_k\beta^k,} \tag{3.45}$$

and

$$\boxed{\begin{aligned}\partial_t\bar{A_{ij}} = e^{-4\phi}\left(-(D_iD_j\alpha)^{TF} + \alpha(R_{ij}^{TF} - 8\pi S_{ij}^{TF})\right) + \\ \alpha(K\bar{A_{ij}} - 2\bar{A_{il}}\bar{A_j^l}) + \beta^k\partial_k\bar{A_{ij}}\bar{A_{ik}}\partial_j\beta_k + \\ \bar{Akj}\partial_i\beta_k - \frac{2}{3}\bar{A}\partial_k\beta^k,\end{aligned}} \tag{3.46}$$

where TF markes the tracefree part of the tensor.

In (3.46) we see that we have the Ricci tensor. This can be separated into two parts

$$R_{ij} = \bar{R}_{ij} - 2(\bar{D}_i\bar{D}_j\psi + \bar{\gamma}_{ij}\bar{\gamma}^{lm}\bar{D}_l\bar{D}_m\psi) + 4\left((\bar{D}_i\psi)(\bar{D}_j\psi) - \bar{\gamma}_{ij}\bar{\gamma}^{lm}(\bar{D}_l\psi)(\bar{D}_m\psi)\right). \quad (3.47)$$

The second part is only dependent on the conformal factor ψ, while the first part is the Ricci tensor calculated with the conformal metric $\bar{\gamma}_{ij}$. It is here much of the instability lies. $\bar{\gamma}_{ij}$ contains mixed (second) derivatives of the metric. To combat this, we can instead introduce the variable

$$\bar{\Gamma}^i \equiv \bar{\gamma}^{jk}\bar{\Gamma}^i_{jk} = -\partial_j\bar{\gamma}^{ij}. \quad (3.48)$$

This is called the *conformal connection function*, and as we can see this contains the first derivative of the metric. This only holds if $\bar{\gamma} = 1$. We now get

$$\bar{R}_{ij} = -\frac{1}{2}\bar{\gamma}^{lm}\partial_m\partial_l\bar{\gamma}_{ij} + \bar{\gamma}_{k(i}\partial_{j)}\bar{\Gamma}^k + \bar{\Gamma}^k\bar{\Gamma}_{(ij)k} + \bar{\gamma}^{lm}\left(2\bar{\Gamma}^k_{l(i}\bar{\Gamma}_{j)km} + \bar{\Gamma}^k_{im}\bar{\Gamma}_{klj}\right). \quad (3.49)$$

What we are missing now is an evolution equation for the conformal connection function. It is possible to show that this takes the form

$$\boxed{\begin{aligned}\partial_i\bar{\Gamma}^i = &-2\bar{A}^{ij}\partial_j\alpha + 2\alpha\left(\bar{\Gamma}^i_{jk}\bar{A}^{kj} - \frac{2}{3}\bar{\gamma}^{ij}\partial_j K - 8\pi\bar{\gamma}^{ij}S_j + 6\bar{A}^{ij}\partial_j\psi\right)\\ &+\beta^j\partial_j\bar{\Gamma}^i - \bar{\Gamma}^j\partial_j\beta^i + \frac{2}{3}\bar{\Gamma}^i\partial_j\beta^j + \frac{1}{3}\bar{\gamma}^{li}\partial_l\partial_j\beta^j + \bar{\gamma}^{lj}\partial_j\partial_l\beta_i.\end{aligned}} \quad (3.50)$$

Equations (3.43), (3.40), (3.45), (3.46) and (3.50) are the new evolution equations used in the BSSN scheme. We see that $\bar{\Gamma}^i$ evolves independently of the metric, meaning that (3.48) is a new constraint equation.

This system of equations is much more stable than the ADM equations, and is the scheme used in most numerical relativity codes.

3.4 The Lapse Function and the Shift Vector

In sec. 3.2 we defined (3.9) as a function of the lapse α and the shift vector $\vec{\beta}$. We could do this due to the degrees of freedom found in GR. Since all the 4 components of t^μ are degrees of freedom, we can choose α and the three components β^i freely. Choosing these 4 degrees of freedom is the same as choosing a coordinate system. In the same section we saw that the shape of the time slice Σ is a function of α, meaning that by choosing the lapse we choose the shape of the time slice. We can also look at our choice in β^i as a choice in how spatial points are shifted between time slices.

But how should we choose the lapse and the shift, and why does it matter? In simulating different spacetimes there are a lot of problems we can encounter. The most obvious problems arise around black holes: If there are regions near a black hole where the math breaks down, how can we expects computers to fare any better... At the center of black holes we have real singularities which may cause real problems, but

due to cosmic censorship[41] these will always be covered by a horizon. We will later discuss *puncture* methods of removing such singularities. We also need to handle the singularity at the horizon itself. These are coordinate singularities, meaning that they arise only due to our choice of coordinate system. Since choosing the lapse and the shift are equivalent to choosing coordinate system, we can therefor choose some lapse and shift that avoid singularities.

Avoiding singularities is not the only motivation for choosing lapse and shift, but is the most relevant for our purpose. We will now describe two (out of many) ways of choosing the lapse and the shift. The first is one of the easiest, and the second is a bit more complicated and is the most similar to the one we are going to use.

3.4.1 Geodesic Slicing

The simplest way we can choose the lapse function and a shift vector is to simply choose

$$\alpha = 1, \quad \beta^i = 0. \tag{3.51}$$

It can be shown that the acceleration an observer feels is given as $a_b = D_b \ln \alpha$. With our choice of lapse, this means that $a_b = 0$[4]. Since the observer feels no acceleration, we know that he/she must follow a geodesic, therefore the name geodesic slicing.

While this seems like a good choice, it is not well suited for simulations. The reason is that this slicing easily leads to singularities. For a black hole we have many geodesics that leads into the horizon, and thus into a singularity. Therefore a large portion of the simulation that starts outside of the horizon will follow a geodesic and with time will end up at the horizon, where the simulation will stop due to the break down of the equations. Even for vacuum this will most likely as well. One can show that even in flat space a small perturbation in the form of a gravitational wave packet, will lead to observers moving towards each other and forming a singularity.

All of this means that geodesics slicing, though simple, is not well suited for simulations. We will instead look at a choice that is much better suited.

3.4.2 Harmonic Slicing and the 1+log Slicing

Harmonic coordinates are coordinates which leaves

$$g^{\mu\nu}\Gamma^{\sigma}_{\mu\nu} = 0. \tag{3.52}$$

Since the left side of the equations is also the definition of one of our BSSN quantities, we see that

$$^{(4)}\Gamma^{\mu} = 0. \tag{3.53}$$

We are free to choose $^{(4)}\Gamma^{\mu} = 0$ since we have four degrees of freedom. Now we cannot directly choose the lapse and the shift, but instead we get differential equations for them

[4] This also shows why we cannot have $\alpha = 0$, since this would lead to infinite acceleration.

$$(\partial_t - \beta^j \partial_j)\alpha = -\alpha^2 K \tag{3.54}$$

$$(\partial_t - \beta^j \partial_j)\beta^i = -\alpha(\gamma_{ij}\partial_j \ln \alpha + \gamma_{ij}\Gamma^i_{jk}). \tag{3.55}$$

This is called harmonic coordinates. This is not used that much in numerical relativity, but a close cousin of it is. If we instead of ${}^{(4)}\Gamma^\mu = 0$, we use one of our degrees of freedom to set ${}^{(4)}\Gamma^0 = 0$ and the rest to set $\beta^i = 0$. This is called *harmonic slicing*. We have directly chosen β^i, but, as with harmonic coordinates, the lapse needs to be determined with a differential equation

$$\partial_t \alpha = -\alpha^2 K. \tag{3.56}$$

This yields the solution

$$\alpha = C\gamma^{1/2}, \tag{3.57}$$

where C is a constant of integration dependent only on space.

This slicing has the same advantage as the geodesic slicing in that it is very simple to find. Contrary to the geodesic slicing it is also stable, and can also avoid singularities[5].

The final slicing we will discuss here is the one we will use. It is a generalization of harmonic slicing, and the only difference the addition to the positive function $f(\alpha)$ to the above differential equation

$$\partial_t \alpha = -\alpha^2 f(\alpha) K. \tag{3.58}$$

$f = 1$ gives harmonic slicing, while $f = 0$ gives geodesic slicing. We are going to use $f = 2/\alpha$. This gives us

$$\alpha = 1 + \log \gamma. \tag{3.59}$$

This is known as *1+log* slicing. This still has a simple form, but is much more effective at avoiding singularities than harmonic slicing.

We can also not that if we do not choose $\beta^i = 0$, we can instead solve the differential equation

$$(\partial_t - \beta^j \partial_j)\alpha = -\alpha^2 K \tag{3.60}$$

for the shift. This is called an *advective shift*, and leads to better simulations, especially for moving puncture simulations (binary black holes) which will be discussed later.

We have now seen how to get more stable simulations by being smart when choosing the lapse and the shift vector. We have mentioned black hole simulations and moving puncture simulations without discussing how to actually make such simulations. Next we will discuss how to set up the initial conditions in such a way that we are simulating black holes.

[5]It is not always able to do this. A better method for avoiding all singularities is *maximal slicing*, which is not discussed here.

3.5 Initial Condition and Black Hole Mergers

We now have a set of differential equations we can use to numerically solve Einstein's field equations. But as is the case when solving every differential equation we need initial conditions. We have seen that two of the equations are time independent, namely the constraint equations (3.36, 3.35), and thus should hold for every time step. We can thus use these to create systems which are physical and can be evolved using the evolution equations. This is a convoluted way of saying that we can use the constraint equations, together with some physical intuition about the slicing, to create initial conditions.

This is not as straight forward as it might seem. The constraint are not trivial to solve, so to get the initial conditions will involve solving these complicated differential equations. We will here look at one, and maybe the most used, solution for the initial conditions: the *two puncture method*. This method will yield initial conditions describing n black holes, with mass, velocity and spin. This initial condition will then be used in creating binary black hole mergers.

We will here just give a quick summation of the initial steps of solving for the initial conditions. This will follow [4] and chapter 3 in [6]. For a more detailed calculation see the latter or [19].

The first step is almost identical to what we did when looking the BSSN equations: Doing a conformal transformation of the spatial metric. We will use a slightly different notation than in (3.41), and instead write that

$$\gamma_{ij} = \psi^4 \bar{\gamma}_{ij}, \tag{3.61}$$

where ψ is the *conformal factor* and $\bar{\gamma}_{ij}$ is the *conformal related* metric. The point here is that ψ has absorbed the scale of the metric, meaning that we are free to choose $\bar{\gamma}_{ij}$. This means that, just as in the BSSN formalism, the constraint equations will now be a function of ψ and not γ_{ij}.

Looking at the Hamiltonian constraint (3.35) we see a dependent on the Ricci scalar R. We can use (3.61) and calculate R and get the new Hamiltonian constraint

$$8\bar{D}^2\psi - \psi\bar{R} - \psi^5 K^2 + \psi^5 K_{ij}K^{ij} = -16\pi\psi^5\rho. \tag{3.62}$$

We can first look at the simplest solution for this equation. A simple solution will be a vacuum solution, meaning that $\rho = 0$. We will also let $\beta^i = 0$ (see chapter 3.1.2 in [6] for more detail) which means that $K_{ij} = K = 0$. This means that (3.62) reduces to

$$\bar{D}^2\psi = \frac{1}{8}\psi\bar{R}. \tag{3.63}$$

We then choose the spacetime to be *conformally flat*, meaning $\bar{\gamma}_{ij}$ is Minkowski, and therefore $\bar{R} = 0$. This gives us

$$\bar{D}^2\psi = 0. \tag{3.64}$$

This is a Laplace equation, with the solution

$$\psi = 1 + \frac{\mathcal{M}}{2r}, \tag{3.65}$$

where, if we take $\mathcal{M}$ to be the mass, we see that the spatial metric we retrieve is that of a Schwarzschild metric! But it is a simple Schwarzschild metric, without velocity or spin.

Going back to (3.62) we can now continue the decomposition we started with (3.38), where we take two further decompositions

$$\bar{A}^{ij} = \bar{A}^{ij}_{TT} + \bar{A}^{ij}_{L}, \tag{3.66}$$

$$\bar{A}^{ij}_{L} = \bar{D}^i V^j \bar{D}^j V^i - \frac{2}{3}\bar{\gamma}^{ij}\bar{D}_k V^k, \tag{3.67}$$

where V^i is a vector potential. We can with these retrieve the extrinsic curvature[4] as

$$K_{ij} = \psi^{-2}\left(\partial_j V_i + \partial_i V_j - \frac{2}{3}\delta_{ij}\Delta\vec{V}\right) \tag{3.68}$$

This reduce the Hamiltonian and momentum constraint equations to[4]

$$\Delta\psi + \frac{1}{8}\psi^8 K_{ij}K^{ij} = 0 \tag{3.69}$$

and

$$\Delta\vec{V} + \frac{1}{3}\nabla(\nabla\times\vec{V}) = 0. \tag{3.70}$$

This can be shown to have the solutions

$$\vec{V} = \sum_{n=1}^{N_p}\left(-\frac{7}{4|\vec{x_n}|}\vec{P}_n - \frac{\vec{x_n}\dot{\vec{P}}_n}{3|\vec{x_n}|^3}\vec{x_n} + \frac{1}{|\vec{x_n}|^3}\vec{x_n}\times\vec{S}_n\right), \tag{3.71}$$

where $\vec{P_n}$ can be interpret as the linear momentum and $\vec{S_n}$ is the angular momentum of the nth black hole. We also get

$$\psi = 1 + \sum_{n=1}^{N_p}\frac{\mathcal{M}_n}{2r_n} + u, \tag{3.72}$$

where the last term u has to be solved numerically. So why have we done this? The reason is that if we solve for ψ directly we will have a singularity at $r = 0$, as we would expect from a black hole, meaning that we need to find a way to handle the singularity. Having gone through the steps above we are left with (3.72) where we can see that we have two parts: The first two terms, which are analytical and contains a singularity at $r = 0$; and u, which is called the correction term. It can be shown that u is regular everywhere, meaning that we can now use the Hamiltonian constrain to solve for u without having to worry about the singularity.

Solving for u numerically and using the resulting ψ and K together with the $1+log$ slicing (3.60) we can get the initial conditions for *moving punctures*, with mass, linear and angular momentum given by $\mathcal{M}$, $\vec{P}_n$ and $\vec{S}_n$ respectively. This is the standard way of simulating black holes.

Note that this initial condition will not reduce to Schwarzschild exactly, but instead to something more like a "trumpet" solution[6]. It is actually difficult to get analytical solutions for these solutions, but it is possible to estimate solutions as asymptotical behaviour. See appendix H in [6] for more detail.

Using $N_p = 2$ we get two so called *two punctures method*. But we still have the unknown function u. This needs to be solved numerically. The main method for doing this is described in [4], and is the method used by Einstein Toolkit to get the initial data for binary black holes (see sec. 4.2). This method uses a spectral method(see sec. 4.3) to solve the differential equations at the start of each simulation.

This method can also be used to find the initial condition for a single black hole, as we have seen above as well. But later we will use analytical initial data for single black holes, since it exists.

3.6 Apparent Horizons

We will see situations were we have to find the event horizon of the black hole, the border from which nothing, not even light, can escape. For simple analytical metrics like the Schwarzschild metric this can be found analytically, e.g. $r_{horizon} = 2GM/c^2$. But this is generally not the case, and especially for a black hole merger, the concept of the event horizon becomes unclear. We will instead look at the *apparent horizon*. This is something that is possible to calculate and is a good approximation to the event horizon.

We can define a 2-dimensional surface S living on our spatial hypersurface Σ, with the outward pointing normal vectors s^i. With this we can define the induced metric

$$m_{ij} = \gamma_{ij} - s_i s_j = g_{ij} - n_i n_j - s_i s_j, \tag{3.73}$$

where we recall that n_i is the normal vectors on Σ.

We are interested in how light move around this surface, so we construct a vector parallel to the outgoing null geodesic

$$k^i = \frac{1}{\sqrt{2}}(n^i + s^i). \tag{3.74}$$

With this we can now define the outgoing null geodesic orthogonal to S

$$\Theta = m^{ij}\nabla_i k_j. \tag{3.75}$$

We want to construct this surface so that the null geodesic vanishes, meaning that we have no outgoing light. This means that we have a surface with the same properties as our event horizon. So we want

$$\Theta = 0. \tag{3.76}$$

Finding an expression for this is not trivial, so for more detail see chapter 7.3 in [6]. If we define the surface of the apparent horizon as a scalar function $\tau(x_i) = 0$, with a

radius from the center given as $h(\theta, \psi)$, it can be shown[6] that

$$m^{ij}\left(\frac{\lambda}{r_C}(\delta_{ij} - \sigma_i\sigma_j) + \lambda\partial_i\partial_j h - s_k\Gamma^k_{ij} - K_{ij}\right) = 0, \tag{3.77}$$

where r_C is the distance from the center to some x_i, $\lambda = (\gamma^{ij}D_{ij}\tau)^{-1/2}$, $\sigma_i = \partial r_C$, and $h = h(\theta, \phi)$ is the distance from the center to the apparent horizon, in other words what we are after.

Equation (3.77) is a second order elliptic partial differential equation, meaning that it is not simple to solve. We will not look at how to solve it here. When we are using Einstein Toolkit later the apparent horizon will be found numerically by the Thorn AHFinderDirect[49] if we give it an initial guess for the size and position. We will use this horizon as the event horizon in our conversion.

3.7 Gravitational Waves

The end goal of this project is to be able to use ray tracing on a spacetime containing *gravitational waves* from a black hole merger. We therefore need to look at what gravitational waves are, a bit about how they are created and a bit about how they are generally represented in numerical relativity.

Gravitational waves are a weak field solution to perturbations in the metric

$$g_{\mu\nu} = \eta_{\mu\nu} + h_{\mu\nu}, \tag{3.78}$$

where $\eta_{\mu\nu}$ is the background metric (here Minkowski) and $h_{\mu\nu}$ is the perturbation, with the condition $|h_{\mu\nu}| << 1$. It is normal to rewrite this as

$$\bar{h}_{\mu\nu} = h_{\mu\nu} - \frac{1}{2}\eta_{\mu\nu}h, \tag{3.79}$$

where h is the trace of $h_{\mu\nu}$.

Due to the coordinate freedom of spacetime, we can introduce a gauge, or more precisely the *Lorenz gauge*

$$\nabla_\mu\bar{h}^{\mu\nu} = 0. \tag{3.80}$$

It is possible to show that there are gauge freedoms associated with $\bar{h}_{\mu\nu}$, which means that we can introduce two more gauges, which will leave us with the transverse-traceless version of the perturbation

$$\bar{h}_{\mu 0}^{TT} = 0, \qquad \bar{h}^{TT} = 0. \tag{3.81}$$

The "TT" here indicates that this tensor is transverse and traceless. Plugging this into (3.79) we see that $h_{\mu\nu}^{TT} = \bar{h}_{\mu\nu}^{TT}$.

If we plug this into the field equations we get the solution[37]

$$h_{\mu\nu}^{TT} = -\frac{16}{c^4}T_{\mu\nu}, \tag{3.82}$$

or for a vacuum solution

$$h_{\mu\nu}^{TT} = 0. \tag{3.83}$$

We recognize this as a wave equation for a wave with the wave speed c.

After using all the gauge terms we are left with $10 - 8 = 2$ freedoms. For the transverse-traceless waves this is the two polarization amplitudes h_+ and $h_\times$. They together with the polarization tensors give the vacuum solution

$$h^{TT}{}_{j\,i} = h_+ e_i^+{}_j + h_\times e_i^\times{}_j = \begin{pmatrix} h_+ & h_\times \\ h_\times & -h_+ \end{pmatrix}. \tag{3.84}$$

If we instead want to solve the full equation (3.82) it can be shown that with weak field and slow velocity approximation, we get to the leading order[37]

$$h_{ij}^{TT}(t, \vec{x}) = \frac{1}{r}\frac{2G}{c^4}\Lambda_{ij,kl}(\hat{n})\ddot{M}^{kl}(t - r/c), \tag{3.85}$$

where $\Lambda_{ij,kl}$ is a projection tensor and $\ddot{M}^{kl}(t - r/c)$ is the double time derivative of the quadruple momentum

$$M^{ij} = \frac{1}{c^2}\int d^3x T^{00}(t, \vec{x}) x^i x^j. \tag{3.86}$$

$t - r/c$ is the *retarded time*, meaning that something will happen at a coordinate only after the information has travelled there at the speed of light.

We will of course not calculate all of this, and instead simulate the source of the gravitational waves (the merger). So we need instead to find a way of keeping track of the gravitational wave forms given the simulated metric. For this we, and Einstein Toolkit, use the *Newman-Penrose* scalar ψ_4. This is, in the Newman-Penrose formalism, one of the five complex scalars describing the traceless part of Riemann tensor.

It is possible, using a null tetrad, to show that ψ_4 can be constructed by components of the Riemann tensor (chapter 9.4 in [6]), and using the definition of h_{ij}^{TT} that

$$\psi_4 = \ddot{h}_+ - i\ddot{h}_\times. \tag{3.87}$$

So knowing ψ_4 we know all that is to know about the gravitational wave form.

It is possible to calculate ψ_4 using spin-weighted spherical harmonics[6]. This is what Einstein Toolkit does when calculating ψ_4.

Chapter 4

Numerical Relativity Frameworks

4.1 Why Use Numerical Relativity Frameworks

We have in the theory section described how we can cast general relativity into a form which is possible to solve using computers. So how do we move along and solve the BSSN equations(3.43 3.40 3.45 3.46 3.50)? Trying to do it ourselves would probably take the work more akin to a PhD than to part of a master thesis[1]. So we will instead look at two already existing and well tried frameworks for doing numerical relativity simulation: First we will look at Einstein Toolkit, which we will be using to do most of our simulations. Secondly we will look at LORENE, which uses a different kind of formalism for doing the PDE solving. We will mostly use LORENE for the conversion into its grid formalism (see below), and not to do any actual simulations.

Below we will go through and more thoroughly introduce the frameworks, but before this we will take a look at the reasons we use the specific frameworks. You may also ask why we are using two frameworks, and what the reason is for this thesis. Can we not just simulate a merger and run it through some ray tracing? We hope to be able to answer these questions before we move on to describe the frameworks in more detail.

4.1.1 Why Einstein Toolkit?

Einstein Toolkit[36] is one of, if not *the* largest numerical frameworks out there. It is open source, and has a large community with everything for forums and YouTube lectures to annual meet-ups[2]. It is also quite versatile, with a modular structure making it easy to set up simple simulations as well as add simulations of custom numerical relativity problems. All of this makes Einstein Toolkit the easiest framework to use for

[1] I once tried to ask Dennis Pollney – whom I have cited a couple of times in this thesis – how I could go around to try and make a Z4 numerical relativity simulator[10]. But he just shook his head and would explain how to do it in 1+1 dimensions. So I gave up making a real 3+1 simulation, and continued using Einstein Toolkit...

[2] One of which – London 2019 – I planned to attend. But two days before departure I got very ill, and instead of leaning and discussing Einstein Toolkit with a group of fellow researchers I spent that time in the hospital with severe food poisoning...

most cases.

Einstein Toolkit is also used in most research on black hole mergers and gravitational waves, with many parameter files (see below and sec. 7) already existing and freely available. This meant that the simulation part of the thesis could be minimised and focus could be shifted to more important aspects.

4.1.2 Why GYOTO and LORENE?

LORENE is a framework developed by researchers at *Observatoire de Paris* in Paris, France[26]. This framework is based on the spectral method (see sec4.3) rather than the Cartesian grid with finite difference method. This means that LORENE is more suited for simulations done in a spherical topology, which restricts the simulations done compared to Einstein Toolkit, but instead gives a superior numerical resolution compared to normal finite difference. It has the capability to do binary black holes, but this is limited to quasi-circular orbits. If we want to look at the merger itself, then we cannot use LORENE.

So why are we using LORENE then, since the goal is a black hole merger, which has a more complex topology? The main reason is GYOTO[53]. GYOTO is a relativistic ray tracer, also developed by researchers at *Observatoire de Paris* – with some, but not complete overlap. As seen in sec. 5.3 GYOTO can do ray tracing in both analytical and numerical spacetime. This means that we can use GYOTO on out merger simulated in Einstein Toolkit.

The reason we have to talk about LORENE when talking about GYOTO is that GYOTO is built upon LORENE. So all the positives and negatives of LORENE are present in GYOTO as well. We will see right below what this means for our project.

4.1.3 Need for the Conversion

So we now have a spacetime simulated in Einstein Toolkit and a ray tracer capable of ray tracing using numerical spacetime. So we're done? No, sadly not... There are two problems we need to solve. The first is that Einstein Toolkit uses a Cartesian grid with finite difference and GYOTO and LORENE use a spectral representation. The second is the problem of a merger not having a spherical topology.

The first problem is the main problem of the thesis. A spectral representation is in essence interpolation of the data using Chebyshev polynomials as basis(see sec. 4.3). Going from a spectral representation to "normal" Cartesian representation is more or less trivial, but going the other way takes a bit more effort. LORENE has many useful features for doing this, so this gives LORENE an important role in this transformation. So we want a pipeline capable of taking data from Einstein Toolkit and then converting it to a spectral representation using LORENE. We can then use this data in GYOTO.

The second problem arises from the spherical nature of the spectral representation used in LORENE: It has a spherical topology, meaning that it is based on spherical coordinates. We will see that trying to convert a binary black hole system to LORENE

leads to a poor representation. We therefore need to force spherical topology onto the binary system. We will see how this is done in 6.1.

So to conclude this discussion. Einstein Toolkit is well suited to simulate binary mergers, while our ray tracer program, GYOTO, is based on LORENE, which uses a spectral representation of the 3+1 quantities. This means that we need to convert the data from Einstein Toolkit to a spectral representation. We will also need to take care of the "non-spherical" topology of the binary black hole merger.

But before discussing all of this, we will look at the two numerical relativity frameworks.

4.2 Einstein Toolkit

4.2.1 Introduction

Einstein Toolkit[32, 36, 57] grew out of the community in search for a simple yet versatile and fast framework to simulate numerical relativity, and especially neutron and black hole mergers. Einstein Toolkit is not one monolithic code, but instead a highly modular framework, build to be easy to use and to contribute custom code. While we will use Einstein Toolkit as an umbrella term for the whole framework –as does most of the literature –, Einstein Toolkit actually consist of underlying frameworks, such as *Cactus* and *Carpet*, which works on their own and can be used for simulations outside of numerical relativity.

We will here take a look at the different parts and how Einstein Toolkit is structured. How Einstein Toolkit is used in practise is described in the method section (see sec. 7). We will instead try to give a broader overview here.

4.2.2 Cactus

The main skeleton of Einstein Toolkit is *Cactus*[25]. Cactus was designed to be open source and highly modular, and was originally developed at Louisiana State University for numerical relativity, but has branched out and is now used by other fields of physics. What makes this possible is that while designed for numerical relativity, there are no concrete physics being done by Cactus alone. Cactus is instead focuses on the *High Performance Computing*(HPC) part of the simulation. Cactus is instead the "flesh" (this is the actual term for this part of Cactus) of the simulation, taking care of the memory management as well as communication between all the modules written for Cactus, called *Thorns*. These thorns will therefore not have to think about the memory management of its variables, how to communicate with other thorns or how paralellization is done. It just need to tell Cactus about the variables and its functions, and Cactus will take care of the rest. The author of the thorns can instead focus on the physics. This is in fact the core philosophy of Cactus and Einstein Toolkit: When writing a thorn, the author should not have to care about the inner workings of any of the other thorns or how they communicate, just the physics of their thorn.

Cactus will handle some physics. Most physical simulations will have some kind of functions over a grid, as well as some time evolution of these functions. Cactus will take care of these aspects of the simulation as well. The author of a thorn will only have to give some function over a grid and a differential equations describing the evolution.

4.2.3 Grid Functions

We will throughout the thesis use the term *grid functions* a lot. Einstein Toolkit will normally hold and evolve variables, functions, tensors, etc on a Cartesian grid using finite difference integration. This is contrary to LORENE, as we will see later uses a spectral grid and a spectral method to evolve data. So to distinguish the way Einstein Toolkit holds data to that of LORENE, we will call the Einstein Toolkit data for *grid functions*.

4.2.4 Carpet and Adaptive Mesh Refinement

The part of Cactus that takes care of the memory management, the mesh refinement and the evolution scheme is called the *driver*. There exist many driver for Einstein Toolkit, with the two normally used being *PUGH* and *Carpet*[17]. We will take a small look at the latter, since it provides the most versatile and powerful driver.

Carpet is build upon Cactus and it main feature is its *adaptive mesh refinement*(AMR). Most simulations in physics have different regions in the simulation with different amount of "activity". By this we mean that in some regions of the simulation nothing will happen for the entire simulation, while in other regions we might have so much happening that we get divergence in the integration. We could try the simulation again with a higher resolution, but then a lot of the simulation will have much higher resolution than needed, and the simulation will be unnecessary slow. We can instead use an adaptive mesh refinement, where the simulation refines the resolution in the parts where it is needed. This is done during run time and can be done for both spatial and temporal resolution.

The AMR used in Carpet is the Berger-Oliger algorithm[8]. This algorithm will can make refinements to regions of the simulations of a factor 2, which is one *refinement level*. This will lead to difficulties in what to do on the boundaries of the refinement levels, both in time and space. Thankfully Carpet takes care of all of this, and the user can just used Carpet (with some parameters (see sec. 7). We will not go into details on how this is done, since this is not relevant for the thesis, but if the reader want a good figure to describe the method they can look at fig. 1 in [36].

4.2.5 Thorns

We have now looked at the skeleton and *flesh* of Einstein Toolkit, but up to now we have not discussed any actually numerical relativity. This is where *Thorns* come in. Thorns are the modular parts that together with the flesh make up Cactus – a cactus consists of cactus flesh and thorns, giving their names to the parts of Cactus – and Einstein Toolkit. So the user will use different thorns that do different tasks. Some

thorns will set up initial conditions, some will take care of the positions of the black hole, some will handle the output, some the possible errors. Most of the functionality of Cactus and Carpet is actually also thorns.

Thorns are categorized into so called *arrangements*. These are groups of thorns has similar responsibilities. One example is *EinsteinBase*. This is an arrangement which take care of the variables often used in numerical relativity. In this arrangement some of the thorns we can find are

- ADMBase: Sets up, keep track of, and can convert the normal 3+1 quantities in the ADM formalism.
- *TmunuBase*: Handles the components of the energy-momentum tensor $T_{\mu\nu}$.
- *HydroBase*: Handles the variables needed to do hydrodynamical simulations.

Note that these thorns do not handle the evolution of these variables, that is up to other arrangements and thorn, e.g. *ML_BSSN*[39].

4.2.6 Black Holes, Mergers and Gravitational Waves

We have now seen that thorns are what makes up most of code used to make numerical relativity simulations in Einstein Toolkit. As an example we will look at some thorns used in simulating a single black hole and a black hole merger. To see how this is done in practise, see sec. 7.

Having set up Cactus and a Carpet AMR grid, we can now take care of initial conditions. We can then use the arrangement *EinsteinInitialData*, where we find the thorn *TwoPunctures*[4]. This will set up initial conditions using the two puncture method (see sec. 3.5. This lets us set up binary black holes with arbitrary spin, momentum and mass. By setting the mass of one of the black holes to zeros, we can also make single black holes with this thorn. For the single black hole we can also use the thorn *IDAnalyticBH*[43]. This lets us make initial conditions using analytical values for the metric.

We now have initial conditions but want to evolve the spacetime. For this we can look in the *McLachlan* arrangement. This lets us use the thorn *ML_BSSN*[15] what will use the BSSN formalism to evolve the spacetime.

We have now made some initial conditions and evolved the spacetime. If this is all we add then we will be disappointed to see that the output folder is empty. We need another thorn to actually output the result. In the 3D data we are going to use here, we want to output the data as HDF5 files. We can thus use the thorn *CarpetIOHDF5*, found in the Carpet arrangement. We can tell this thorn which variables and grid functions we want to output. Note that we never told the thorns holding the grid functions how to output. Instead the I/O thorn will take care of all the formatting and outputting. As long as the grid functions exist in the simulation, the I/O will know of it and know how to output it (depending of the type of the function).

There are some more functionality we need to add to the simulation. We want to find and store the horizons of the black holes throughout the simulation. For this we

can use the arrangement *EinsteinAnalysis*. The first thorn we will use from here is *PunctureTracker*. This thorn will track the punctures made by TwoPunture. There are two reasons we do this. The first, and arguably most important reason, is that we want to use the AMR around the punctures. PunctureTracker allows this to be done automatically (for more detail see the section on AMR in [36]). The second reason is to give the position of the black holes to the next thorn *AHFinderDirect*[49]. This thorn will, given a guess for the position and radius of the horizon, calculate the apparent horizons(see sec. 3.6) of the black hole(s).

For a black hole merger we are interested in the gravitational waves generated. For this we can use the thorns *WeylScal4* and *Multipole*. These can generate the Weyl Scalars for the outgoing gravitational waves(see sec. 3.7), which makes it possible for us to look at the gravitational wave patterns of the merger.

4.2.7 Simulation Factory

An other important feature of Einstein Toolkit is Simulation Factory. This is a wrapper for all the parts described above. Simulation Factory is the part which the user will actually interact with when using Einstein Toolkit, and takes care of distributing the code over a cluster or supercomputer, logging and debugging, I/O and actually running the code. To see how it is used, look in sec. 7.

4.3 Spectral Methods

4.3.1 Theory

As mentioned above in section 4.2.2, Einstein Toolkit uses a finite difference method to calculate the derivatives. Finite difference is maybe the most used method to solve differential equations, ordinary and partial. Given some grid x_i with $i = 1, \ldots, N$, we can have defined some grid function $f_i = f(x_i)$. Through a simple Taylor expansion of f_i, it is trivial to find that a first order approximation of the derivative of said function is given as

$$f_i' = f'(x_i) \approx \frac{f_{i+1} - f_i}{x_{i+1} - x_i}. \tag{4.1}$$

This holds to an order of $\Delta x = x_{i+1} - x_i$, which means that given a fixed interval, the error will be inversely proportional to the number of grid points N, so $\Delta x \sim 1/N$. We know that it is possible to find schemes with have a higher order of accuracy, for example central difference

$$f_i' \approx \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2}, \tag{4.2}$$

which is of second order $error \sim 1/N^2$. In general, the best accuracy we can get with a finite difference method is that the error goes as a power law $1/N^n$ for some n'th order scheme.

Instead of representing the functions on a discrete grid, and using finite difference to differentiate the functions, we can instead use a spectral method. We can start by representing our function as an expansion/interpolation over some test function

$$f(x) \approx \sum_{i=0}^{N} a_i g_i(x), \tag{4.3}$$

where $g(x_i)$ is the trial function and a_i is some coefficients. The trial function can for example be $\cos x$ and $\sin x$ for a normal Fourier series – suited for periodical functions –, or, as we will use often, Chebyshev polynomials. The important feature of the test function is that they need to form a basis for some space, so that a function in that space may be expanded in said basis.

Some of the magic of this method is in how derivatives of our function is found. In this representation we have a continuous approximation of our function – this is discrete in the grid method –, represented a known test function and some coefficients, meaning that we can take a normal differentiation of our function

$$f'(x) \approx \sum_{i=0}^{N} a_i g_i'(x), \tag{4.4}$$

where $g'(x_i)$ is analytical and know! This means that a good approximation of $f(x)$ we also have a good approximation of $f(x)$.

So, does this mean that we have a better approximation of the derivative with a spectral method than with a finite difference method? In most cases yes: 4.3 is an approximation where the accuracy is dependent on the number if points use to find the approximation N, but it can be shown that this accuracy goes faster than a power law of N, and can be close to exponential in ideal cases[31]. This means that to get the same accuracy as in the finite difference case, we need a lot fewer points – in some case a couple of order of magnitude lower. This means that with a spectral method, we are able to both speed up the calculation and/or get a more accurate simulation (two sides of the same coin).

We will below see how to find these expansions, and how they are used to solve differential equations.

4.3.2 Expanding In the Test Function

We have seen that when expanding our function to a trial function we get analytical expressions for the derivative and can solve differential equations with close to exponential accuracy. But the question is then: Which (set of) functions should we use as trial functions. A good answer for this is the "Moral principal 1" found in [13] which, paraphrased, says to always use Chebyshev polynomials unless we have periodical function – in which case use a normal Fourier series – or if "you're really, really sure that another set of basis functions is better"[13].

So how do we expand into Chebyshev polynomials? Chebyshev polynomials are polynomials orthogonal with respect to the weight $w(x) = (1-x^2)^{-1/2}$ over the interval

$[-1, 1]$ and defines as $T_n(\cos\theta) = \cos(n\theta)$. From this we can show that gives us the polynomials[31][6]

$$T_0(x) = 1, T_1(x) = x, T_2(x) = 2x^2 - 1, T_{n+1} = 2xT_n(x) - T_{n-1}(x), \tag{4.5}$$

where the last term is a recursive expression used to find all the higher order terms.

Each of the above polynomials have N zeros at

$$x_i = \cos\left(\frac{\pi(k + 1/2)}{N}\right), \qquad k = 0, 1, ..., N - 1 \tag{4.6}$$

and $N + 1$ extrema at

$$x_i = \cos\left(\frac{\pi k}{N}\right), \qquad k = 0, 1, ..., N. \tag{4.7}$$

These points will be very useful when trying to do the expansion.

We can now do the expansion. We want to have an expansion

$$f(x) = \sum_{k=0}^{N} \tilde{f}_k T_k(x), \tag{4.8}$$

where $\tilde{f}_k$ is the kth coefficient of the expansion. To do this expansion, we do as we do in a Fourier expansion, we take the inner product

$$\tilde{f}_k = (f, T_k) = \frac{2 - \delta_{i0}}{\pi} \int_{-1}^{1} T_k(x) f(x) w(x) dx, \tag{4.9}$$

where $w(x) = (1 - x^2)^{-1/2}$ is the weights of the Chebyshv polynomials. This integral is often impossible to solve, so we have to use numerical integration instead. The methods used are normally Gauss integration or Gauss-Lobatto integration[31]. In these methods only a few points are needed to do the integrations. These are the collocation points and are given as the zeros of the Chebyshev polynomials(4.6) when using Gauss integration and the extrema(4.3.2) when using Gauss-Lobatto. Following [6] we can use Gauss-Lobatto to get that

$$\tilde{f}_i = \frac{2}{N c_i} \sum_{k=0}^{N} \frac{1}{c_k} f_k T_k(X_k), \tag{4.10}$$

where $c_i = 2$ at $i = 0$ or $i = N$ and is 1 everywhere else, and X_k are the collocation points. f_k is also evaluated at the collocation points. We can then get the original function from (4.8).

We have now transformed our function into a spectral representation using Chebyshev polynomials. As we will see below in sec. 4.4, we can expand in many other basis. Note that in most code this expansion is done using the *Fast Fourier Transform* (FFT).

4.3.3 Solving Differential Equations

So we have now transformed a function into a spectral representation. We will now look at how we can use a function in a spectral representation to solve differential equations.

A general differential equation can be written as

$$Lu(x) = s(x), \tag{4.11}$$

where L is a linear differential operator, $s(x)$ is some source term and $u(x)$ is the function we want to solve for. We then define the residue for a numerical solution solution $\tilde{u}$ as

$$R = L\tilde{u} - s. \tag{4.12}$$

Since we want to have $\tilde{u}$ as close as possible to $u(x)$ we want the residue to be minimized. There are three main methods for doing this: the first two, *Galerkin* and *Tau* can be found in [31]. The third is the so called the *pseudospectral* or *collocation* method, and consists of evaluating (4.12) at some collocation points.

When using Chebyshev polynomials as an expansion for our functions we have that the collocation points used to evaluate (4.12) are the same as the collocation points we used to do the spectral transformation (4.10).

We can see how we do this is practice. First the source term should be transformed with (4.10). Next note from (4.8) that any derivative of the function $u(x)$ instead becomes a derivative of the Chebyshev polynomials

$$f'(x) = \sum_{k=0}^{N} \tilde{f}_k T_k'(x). \tag{4.13}$$

The derivative of the Chebyshev polynomials are analytical, meaning that we have an analytic expression for the derivative, instead of an approximation which we have in finite difference. Since we have polynomials with a recursive relation we can write the derivatives of the Chebyshev polynomials as some matrix times the polynomials them self[6]

$$T_k' = \sum_{i=0}^{N} D_{lk} T_l(x), \tag{4.14}$$

where the matrix D_{lk} is[31]

$$D_{lk} = \begin{pmatrix} 0 & 1 & 0 & 3 & 0 & \cdots \\ 0 & 0 & 4 & 0 & 8 & \cdots \\ 0 & 0 & 0 & 6 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 8 & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{4.15}$$

From this we also see that the double derivative becomes $D_{l_k}^2$. This means that the differential operator L becomes a normal matrix in the spectral representation:

$L \to L_{lk}$. This means that we have

$$L_{kl}\tilde{u}_k T_l = \tilde{s}_l T_l, \tag{4.16}$$

where the summation over the k and l are implicit. We then evaluate this at the aforementioned collocation points. We know L_{kl}, T_l and $\tilde{s}_l$, meaning $\tilde{u}_k$ is the only unknown. Since we have evaluated this at N collocation points, we have $N+1$ equations for $N+1$ unknowns in the form of a matrix equation. We have thus made the different equation into a matrix inversion problem. This is the magic of the spectral method. Not only that, but since the Chebyshev polynomials have close to a exponential precision, the value of N can be orders of magnitude lower than the resolution normally used in finite difference! This is still not a perfect method, since solving a large matrix inversion can be computationally difficult.

This was a short description of the spectral method for solving differential equations. Below we will look a bit on how this is used with numerical relativity in LORENE.

4.3.4 Using the Spectral Representation without Solving Differential Equations

We have said that we will solve Einstein's field equations in Einstein Toolkit and not in a spectral method. So why involve the spectral representation at all? As we will see below this is because GYOTO use LORENE, which uses spectral representation. But before looking at this we will give a short reason for why to have the functions in a spectral representation, even when not solving differential equations using spectral methods.

Let us say that we now have our function as a numerical representation given by the N spectral coefficients (4.10). We now want to know our function at some arbitrary point x. In a normal numerical representation of a function, we would need to do an interpolation of the data if x wasn't in our original numerical representation. In a spectral representation we don't need to worry about that, we can get this function value of f at any x using (4.8), without having to worry about any interpolation. Thus with N spectral coefficients we can get the function value of f anywhere.

This is not completely true, since the interpolation is hidden in the spectral representation itself, which the function transformed into a Chebyshev basis being the actual interpolation. But as we have seen this interpolation can have exponential precision. This means that we can get the function value at any point with a high precision.

Say we now want to find the derivative of the function. With a normal numerical representation we would need to use some finite difference scheme. But we have seen that the derivative of a spectral representation has an analytical expression (4.13). So we can find the derivatives with a high precision at an arbitrary point.

These features of the spectral representation will be used by GYOTO to quickly and precisely get the 3+1 quantities and their (spatial) derivatives to calculate the geodesic equations(5.13).

4.4 LORENE

Langage Objet pour la RElativité NumériquE or LORENE[26] is a framework, or more precisly a set of C++ classes, used for numerical relativity, numerical astrophysics as well as tensor manipulation created mainly by Eric Gourgoulhon, Philippe Grandclément, Jean-Alain Marck, Jérôme Novak and Keisuke Taniguchi.

The main difference between Einstein Toolkit and LORENE is that LORENE uses a multi-domain spectral method to solve the differential equations associated with numerical relativity instead of the finite difference used by Einstein Toolkit. We have already seen what is meant by a spectral method, but we will look at bit more on the multi-domain part below.

4.4.1 Multi-domain Spectral Methods

We have above discussed how to transform a function into a spectral representation as well as using this to solve differential equations. The real world is rarely this simple. When using this method in numerical relativity we have different areas with different needs for resolutions. We are also using a spherical coordinate system, meaning that we need different basis for θ and ψ. For this we use a multi-domain spectral method[11][31]. This consists of defining different domains with different radii $(r_0, r_1, .., r_{n-2}, r_{n-1}$, where r_{n-1} can go to infinity.

Each of these domains can then have different resolution: N_r, N_θ and N_ϕ. This means that we can do a spectral transformation of our function with a different resolutions at different regions. So if some simulation has something strange happening at $r = 3$ we can define a domain spanning from $r = 2$ to $r = 4$ with a higher resolution than the other domains. After doing all the transformations we need to do some more work to "sew" the regions together[11]. This is thankfully done inside LORENE.

There is one problem here. We have already seen that the Chebyshev polynomials are defined in the region $[-1, 1]$. This means that we need to transform our r into this region, as well as compacting $r = \infty$. This is done with the transformation[53]

$$\zeta(r) = \begin{cases} r/r_0 & \text{for } r \leq r_0 \\ \frac{2r - r_l - r_{l-1}}{r_l - r_{l-1}} & \text{for } r_{l-1} \leq r \leq r_l \\ 1 - \frac{2r_{n-2}}{r} & \text{for } r > r_{n-2}. \end{cases} \tag{4.17}$$

We can then look at the final expression for the spectral transformation, using both Chebyshev polynomials as well as Fourier transforms for the periodical θ and ϕ. From [53] we get

$$f(t, r, \theta, \phi) = \sum_{k=0}^{N_\phi - 1} \sum_{j=0}^{N_\theta - 1} \sum_{i=0}^{N_r - 1} \tilde{f}_{ijk}(t) R_{ij}(\zeta) \Theta_{jk}(\theta) \Phi_k(\phi), \tag{4.18}$$

where

$$\Phi_k(\phi) = \begin{cases} \cos(m_k \phi) & \text{for } k \text{ even} \\ \sin(m_k \phi) & \text{for } k \text{ odd} \end{cases} \tag{4.19}$$

$$m_k = \text{integer part of } k/2, \tag{4.20}$$

$$\Theta_{jk}(\theta) = \begin{cases} \cos(j\theta) & \text{for } m_k \text{ even} \\ \sin(j\theta) & \text{for } m_k \text{ odd} \end{cases} \tag{4.21}$$

$$R_{ij}(\zeta) = \begin{cases} T_{2i}(\zeta) & \text{for } j \text{ even and } r \leq r_0 \\ T_{2i+1}(\zeta) & \text{for } j \text{ odd and } r \leq r_0 \\ T_i(\zeta) & \text{for } r \geq r_0 \end{cases} \tag{4.22}$$

The matrix $\tilde{f}_{ijk}$ is the actual spectral representation of $f(t, r, \theta, \phi)$ and is everything that is needed to get back the original function. This is also what is needed to be able to do the ray tracing with GYOTO, as this matrix lets GYOTO get the original function at any point as well as its derivatives.

4.4.2 Usage

LORENE has the capacity to simulate a wide range of numerical relativity and astrophysical systems, from *MOND*[9] to *strange quark stars*[24]. It is actually also able to create binary black holes[30][28] as well as black hole-neutron star binaries[29]. In fact there exist three Thorns for Einstein Toolkit inside the *EinsteinInitialData* arrangement called *Meudon_Bin_BH*, *Meudon_Bin_NS* and *Meudon_Mag_NS* which uses LORENE to set up initial data for Einstein Toolkit using LORENE. It is also possible to call on LORENE code in Einstein Toolkit using the Thorn *LORENE* in the arrangement *ExternalLibraries*.

We will not use LORENE to such an extent. We will instead use the fact that LORENE is the framework which the ray tracing program GYOTO it built upon (see below). This means that we need to get data from Einstein Toolkit into LORENE, which then can send the data to GYOTO.

The first thing we can use LORENE to, is to do the spectral transformation needed to have the data in the formalism used by GYOTO. LORENE has this capability, and by declaring the domain sizes and resolutions, LORENE can then calculate the needed collocation points needed. If we then give LORENE these points, LORENE is able to do the transformations.

Our transformed data will be the 3+1 quantities discussed in the section on numerical relativity (sec. 3.2). This means that we have one scalar, one 3-vector and two tensors (3×3 matrices)[3]. LORENE is capable of holding all of these different objects, and of doing algebraic, as well as some calculus operations on them. This means that we can do things like inverting the metric as well as transforming everything into a spherical triad, which is needed for GYOTO, inside of LORENE.

All of this is what will be done by the converter, which will be discussed more in the method section.

[3]Technically all of these quantities are tensors, but they are different objects in LORENE.

4.4.3 GYOTO and Ray Tracing in Numerical spacetimes

The reason we have talked so much about LORENE is that this is the numerical relativity framework on which the ray tracing program we want to use, GYOTO, is build upon. This means that we needed a good understanding of LORENE to be able to understand and use GYOTO. We will now look more closely at ray tracing, and use that to introduce GYOTO.

Chapter 5

Ray Tracing

5.1 Introduction

We want to make an image of the black hole merger. But how does one go about this? This is where ray tracing comes in. Ray tracing is a technique where we simulate the path of the photons as it leaves a light source, interacts with objects and finally hits the pixels of the camera – represented by a pixel screen –, creating an image we can see. This way we will have both a fine image of the objects in the scene, but we will also have simulated the effects of the objects on the photons, meaning that we can get some physics from the image, e.g. how the gravitational waves have affected the photons.

The idea of ray tracing is not new, and traces its origins back to the German Renaissance painter Albrecht Dürer, who used a thread going from points on an object to a screen, representing the image[55]. The thread is meant to be the path of the light. This was used as a tool for understanding the path of light and to help paint. We will of course not tie a string at some star and stretch it around a black hole (I would if I could). Instead we will let the computer simulate the path of the photons. Ray tracing with computers has been done since the early 1970s[56], and is now the standard for simulating the effect of light on objects in everything from Pixar movies to video games – especially after the release of NVIDIA's new RTX 2000/3000 graphic cards, which have built in support of extremely effective ray tracing.

Below we will describe in general terms how ray tracing is done. This will be done from a general computer graphics points of view. This will of course not be exactly the same as ray tracing around merging black holes, but many of the concepts are the same. After that we will describe how ray tracing can be done in general relativity, both with a known analytical metric and with a numerical metric.

5.2 Doing Ray Tracing

We will here look at how ray tracing is done. We are going to look at a more general scenario where we have a simple sphere and a light in the scene. This will give a broad understanding about how ray tracing is done. More or less everything here is

transferable to the general relativity case, except that it will be more complex since we have to introduce geodesics for the path of the light instead of assuming that light travels in a straight line (which is technically mean that they travel on a geodesic).

I described above that as the light travelling from the source and hitting the camera is not how ray tracing is done. This would result in many photons being simulated travelling from the light source, but never hitting the camera, thus leading to a very ineffective and wasteful simulation. Ray tracing is rather done in reverse, with the light travelling from the camera, interacting with objects in the scene and finally terminating at a light source or at some distance determined to be infinity and therefore dark. The color of the image is thus determined by the objects the photons passed through or is reflected by, and whether the photon hits a light source or not. When the photon passes through or is reflected by an objects, the property of the object will change that of the object, e.g. a red semitransparent sphere will make the photon more red and decrease the intensity of the photon. This setup with light source and a red sphere is illustrated in fig. 5.1. Here we can see a 2D screen with four pixels. The photons are sent from each pixel at an angle determined by the *field of view* of the image. Here it is about 45°. We see that the bottom photon does not hit anything, meaning that this pixel will be dark. The second pixel (from the bottom) hits the sphere but travels through it and into infinity. This will also be dark. The third photon hits the sphere, bends, exits the sphere and hits the light source. Since it hit the sphere, its intensity is decreased and it becomes red. The top photon hits the light directly and just becomes white. So what we will see is a sphere which is dark at the bottom and red and transparent on the top, with a shining white light above and behind. An example of how this would look can be found in fig. 5.2.

This is a simple description of how ray tracing is done by the computer. In real cases the light and objects in the scene might be more complex and interact with the photons in a more complicated manner. But the general method of sending photons from the screen to interact with the objects is the same, and is also the same when we now move over to ray tracing in general relativity.

5.3 Ray Tracing in General Relativity and GYOTO

We have now briefly looked at how ray tracing works. Above we described it in a general way, and assumed that light moves in straight lines, and only changes direction when it interacts with objects in the scene. We will now move over to ray tracing in general relativity, meaning that we have non-Euclidean space(time) and that photons no longer move in straight lines. Instead they move on so called geodesics. This path is governed by the geodesic equation (5.10), which describes the shortest path between two points. While we can include objects in the scene, such as e.g. stars and accretion disks, the biggest effect on the path of the light is spacetime itself.

All the above steps of doing ray tracing described above still holds, we now have to include a way of calculating the path of the photons as they travel through empty space. It is this effect of the curvature of spacetime on the photons, such as event horizons,

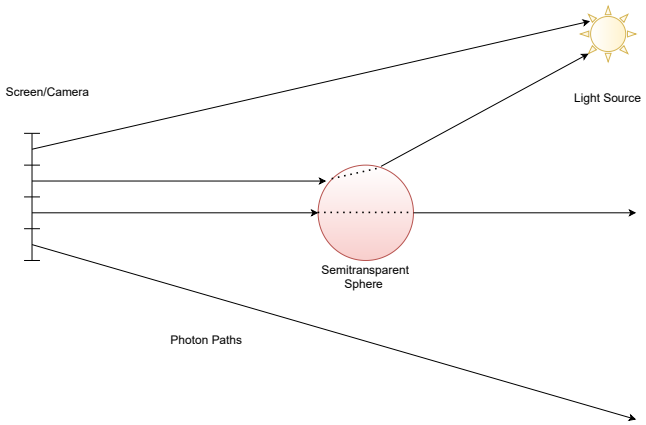


Figure 5.1: An illustration of how ray tracing works. The photons are sent from a screen (here a 2d screen with four pixels). The color of the pixels are determined by what they hit on their path, and if they it a light source. We see that two photons hit nothing (one of them after going through the sphere). The two others hit the light (one after going through the sphere). The image will therefore be a partly lit sphere, with a visible light on the side, in the background.

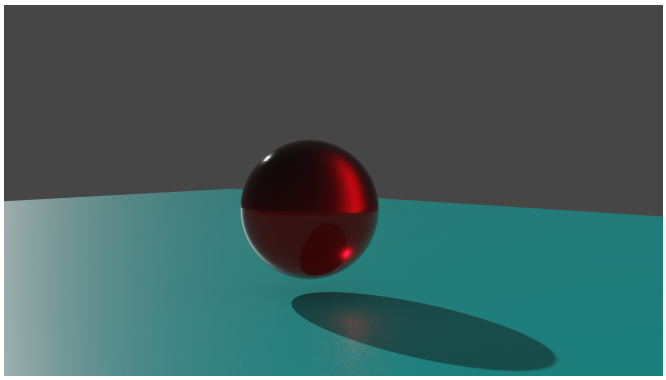


Figure 5.2: A ray tracing of a scene similar to the one described in fig. 5.1. This is a semitransparent red sphere over a light blue floor. There is a light to the left of the sphere, but the light itself is not rendered. This was made in the 3D modelling software Blender.

gravitational waves, etc, which we are interested in. We will look at two ways of doing this. The first assumes that we are ray tracing in a spacetime containing a single and possibly spinning black hole. In this case we can find an analytical expression for the metric. From this metric we can find expression for how photons will move in said spacetime – even without going through the geodesic equation.

For everything but the most trivial situations, an analytical expression for the metric is not possible to find. We thus have to resort to numerical methods for finding the metric. As we have seen earlier, to find numerical metrics we need to use the 3+1 formulation. This means that we need to do the same with the geodesic equation. This will be done below.

We have now described how to do the ray tracing in curved spacetime. The analytical method is not too difficult to implement, and can be done as a fun project without understanding much of the math[1]. For the numerical spacetime the implementation is far more difficult, and to implemt it myself would probably constitute a master thesis all by itself. We will therefore use the program GYOTO[53]. How this program works and how we are going to use it will be described later.

5.4 GYOTO

The main goal of this thesis is to look at how photons behave around the black hole merger, and ultimately how the gravitational waves emitted by the merger affects the light. Having a simulation of the spacetime of the merger, we now need to be able to send light at the black holes. This is where a ray tracer comes in. A ray tracer is program that simulates the path of photons in a given spacetime[2]. This is something which we might write ourselves, and for simple analytical metric like a Schwarzschild or a Kerr metric it is not that much work. But since we are dealing with numerical spacetimes the program becomes more complicated. Thankfully, researchers have been writing ray tracers for numerical relativity since the 70s, to study everything from how stars orbiting black holes will look to accretion disks around supermassive black holes[48] and neutron stars with an atmosphere [54]. Though there are many ray tracers out there, my choice landed on GYOTO [53]

GYOTO (General relativitY Orbit Tracer of Observatorie de Paris) is an open source ray tracer written by F. Vincent, T. Paumard, E. Gourgoulhon and G. Perrin from the Observatorie de Paris [53]. GYOTO is written in C++ and is modular, meaning that it is easy for users to add functionalities to the simulation as plug-ins, without having to edit the main code of the program. It is also possible to use Yorick[3] and Python as scripting languages to run GYOTO simulations.

[1] A good example of how to do this can be found here: https://www.codeproject.com/Articles/994466/Ray-Tracing-a-Black-Hole-in-Csharp

[2] More generally a ray tracer will also simulate objects in the path of the photons, and how the photons scatter and interact with said objects. In our case the spacetime itself is the most important, but gas or optically thin disk might be added around the merger.

[3] A scientific scripting language found at `https://github.com/LLNL/yorick`

GYOTO has two main ways of raytracing. The first is with the use of analytical Kerr metrics (sec. 5.4.1). This is the fastest way of raytracing, since the simulation is highly optimized. This is typically used to look at disks and tori, and stars orbiting Kerr black holes. The downside with this is that it is limited to just Kerr metrics. The second way is raytracing with the use of non-analytical, simulated metrics(sec. 5.4.2). This allows for investigations of more interesting metrics, like our binary black hole merger. According to the authors, it is this feature that makes GYOTO standout compared to other ray tracers. It is also this feature which makes it possible for me to ray trace in a merger metric.

5.4.1 Ray Tracing in an Analytic Metric

The first way of doing raytracing is with an analytical metric; the Kerr metric, describing a rotating black hole. The angular momentum of the black hole is given by aM, where a is the spin parameter and M the mass, so with $a = 0$ this reduces to the Schwarzschild metric. The Kerr metric is given as

$$\begin{aligned} ds^2 = &-\left(1 - \frac{2Mr}{\Sigma}\right) dt^2 - \frac{4Mar\sin^2\theta}{\Sigma} dt d\phi + \frac{\Sigma}{\Delta} dr^2 \\ &+ \theta^2 + \left(r^2 + a^2 + \frac{2Ma^2 r \sin\theta}{\Sigma}\right) \sin^2\theta d\phi^2, \end{aligned} \tag{5.1}$$

where $\Sigma = r^2 + a^2\cos^2\theta$ and $\Delta = r^2 - 2Mr + a^2$. This metric gives rise to three constants of motion: $E = -p_t$ is the energy at infinity, $L = p_\phi$ is the axial component of the angular momentum and $Q = p_\theta^2 + \cos^2\theta[a^2(\mu^2 - E^2) + \sin^{-2}\theta L^2]$ is the so called *Carter constant*. These four constants will be used later to ensure the stability of the integration. One of them is the particle mass squared $\mu^2 = -p_\sigma p^\sigma$. For general particles this is as mentioned the mass, but since we are dealing with photons this will be $\mu^2 = 0$. As we will see later, this identity will be used when checking the procession ray tracing.

To be able to integrate the geodesics we have to rewrite the metric above using Hamiltonian formalism. Following [35] we get

$$\dot{t} = \frac{1}{2\Delta\Sigma}\frac{\partial}{\partial E}(R + \Delta\Theta) \tag{5.2}$$

$$\dot{r} = \frac{\Delta}{\Sigma} p_r \tag{5.3}$$

$$\dot{\Theta} = \frac{1}{\Sigma} p_\theta \tag{5.4}$$

$$\dot{\phi} = -\frac{1}{2\Delta\Sigma}\frac{\partial}{\partial L}(R + \Delta\Theta) \tag{5.5}$$

$$\dot{p_t} = 0 \tag{5.6}$$

$$\dot{p_r} = -\left(\frac{\Delta}{2\Sigma}\right)' p_r^2 - \left(\frac{1}{2\Sigma}\right)' p_\theta^2 + \left(\frac{R + \Delta\Theta}{2\Delta\Sigma}\right)' \tag{5.7}$$

$$\dot{p_\theta} = -\left(\frac{\Delta}{2\Sigma}\right)^{\theta} p_r^2 - \left(\frac{1}{2\Sigma}\right)^{\theta} p_\theta^2 + \left(\frac{R + \Delta\Theta}{2\Delta\Sigma}\right)^{\theta} \tag{5.8}$$

$$\dot{p_\phi} = 0, \tag{5.9}$$

where the dot is the derivative with respect to the proper time, $'$ with the respect to r and $^{\theta}$ with respect to θ, and $\Theta = Q - \cos^2[a^2(\mu^2 - E^2) + \sin^{-2}\theta L]$ and $R = (E(r^2 + a^2) - aL)^2 - \Delta(\mu^2 r^2 + (L - aE)^2 + Q)$.

The above equations are possible to integrate, using a Runge-Kutta 4 integrator with adaptive steps. Each photon is integrated backwards in time. This means that the photon starts at the lens of the camera, with one photon for each pixel of the final image. Each pixel of the camera is associated with one direction, corresponding with one pixel of the background . Thus the initial position and momentum are given.The photon is then integrated backwards until it either: Hits an emitting object, reaches infinity (or some range from an object set as infinity) or gets close to an event horizon.

The constants of motion mentioned above are used to ensure a correct integration. E and L are directly conserved through eq. (5.6) and (5.9), while Q is used to correct $\dot{\theta}$. The magnitude of the momentum $|\vec{p}|$ is also a constant of motion, and is used to correct $\dot{r}$. All of this, together with the RK4 integrator ensure an accurate simulation. (an overview of the accuracy is shown in [53]).

5.4.2 Numerical Metrics

If we use only analytical metrics, as above, then the number of cases we can investigate becomes limited due to the complexity of the field equations. For the more complex cases, we need to figure out a way to do ray tracing on simulated spacetimes. To be able to due this, we need to find a 3+1 decomposition3.2 of the geodesic equation

$$\frac{d^2x^\mu}{ds^2} + \Gamma^\mu_{\alpha\beta}\frac{dx^\alpha}{ds}\frac{dx^\beta}{ds} = 0, \tag{5.10}$$

which describes the path of a photon or a particle. This has been done by the authors of GYOTO (the program we will use to do the ray tracing), and published in [52]. They start with a covariant form of the geodesic equation

$$p_\mu \nabla p^\alpha = 0, \tag{5.11}$$

where p^μ is the 4-momentum. In a 3+1 form, this can be written as

$$p^\alpha = E(n^\alpha + V^\alpha), \tag{5.12}$$

where $n_\mu V^\mu = 0$. $E = -p_\mu n^\mu$ is the energy of the particle as measured by the Eulerian observer, while V^μ is the velocity of the particle on Σ_t, and gives us the 3-velocity of the particle.

By inserting the 3+1 quantities and rewriting the covariant derivatives in a 3+1 form, and doing a lot of algebra, we end up with two sets of equations describing the

position $x^i = X^i(t)$ of the particle

$$\frac{dX^i}{dt} = \alpha V^i - \beta^i$$
$$\frac{dV^i}{dt} = \alpha V^j[V^i(\partial_j \ln\alpha - K_{jk}V^k) + 2K^i_j - {}^3\Gamma^i_{jk}V^k] - \gamma^{ij}\partial_j\alpha - V^j\partial_j\beta^i. \quad (5.13)$$

A detailed calculation can be found in [52].

5.4.3 How the Ray Tracing is Done and Why a Spectral Method

We now have 3+1 form of the geodesic equation. This is solved with a fourth-order Runge-Kutta scheme in GYOTO, using the 3+1 quantities that come from our converter. The reason that the spectral formalism comes in hand here is that eq. (5.13) requires the value of the quantities at arbitrary positions. With a normal grid, an interpolation would be necessary for almost every evaluation, but with the spectral method it is trivial to find the values at an arbitrary position. We have also seen that the spatial derivatives for a function in a spectral representation can be found quasi-analytically (we get an analytical expression containing the spectral coefficients, which are approximations). This means that we do not have to use a differentiation scheme to find the derivatives, but we get them from the spectral representation of the functions themselves. Had the data not been in this representation, GYOTO would have had to find all the derivatives itself, leading to less accuracy. These two reasons make ray tracing in a spectral representation easier.

It is also possible to get the 4-metric $g_{\mu\nu}$ and solve eq. (5.10) directly. This was tried at first, but we later moved on to solving eq. (5.13) due to problems with non-linearity of $g^{\mu\nu}$ and other problems.

When using GYOTO is using a numerical metric, it has historically assumed spherical symmetry. This makes finding the Christoffel symbols in (5.13) much easier. As we will mention again in the method section 10.1.1 to get this project to work, this assumption had to be removed.

Using GYOTO is quite simple, since it runs mostly through XML file, or scripting in Python or Yorick. We will look at how we are going to run GYOTO in this project in sec. 10.2.

5.4.4 Astrophysical Objects

As we have seen the paths of the photons are determined by the spacetime, but until now we have not placed any objects. So while the photons moves and bends, we haven't talked about any light source. Without any light source the photons will register as going to infinity and thus we'll get a black image. We can in GYOTO place astrophysical objects in the scene, which will work as emitting objects giving light to the scene.

The object we will be using most in this thesis is the *fixed star*. As the name suggests, this is a star fixed at some coordinate (x_c, y_c, z_c) with a radius r_{star}. This star will have some absorption,α_ν, and emission,j_ν, coefficients associated with each point

inside of the star. This means that GYOTO can calculate the intensity of the photons passing through the star[53]

$$I_\nu = \int_{s_0}^{s} \exp\left(-\int_{s'}^{s} \alpha_\nu(s'')ds''\right) j_\nu(s')ds'. \tag{5.14}$$

There is another radius associated with the fixed star, namely a parameter called $RMax$. This is not a physical parameter, but instead the radius at which the integrator of GYOTO starts to notice the star. This means that $RMax < r_{star}$. We will see when looking into the ray tracing of a fixed star that this parameter has an effect on the results.

Another objects we will look at is a Page-Thorne disk[42]. This is a type of accretion disk we will use as a final image to show that the ray tracing was a success.

Chapter 6

Splitting the Black Hole Binary

We have now discussed Einstein Toolkit, LORENE and GYOTO. Before we move on to how the converter code is built up, and how to run everything we need to discuss how to deal with the spherical topology of LORENE.

As we have discussed before, LORENE use spherical coordinates when transforming into a spectral representation. This means that it has a spherical topology. In practise this means that LORENE works best when the objects transformed is spherical in nature. It does not need the object to be spherical symmetric, but an object such as a black hole binary, which has a clearly preferred "direction", will become poorly resolved.

We thus want a procedure to make out black hole binary more spherical. The way we will be doing this is by splitting the binary into two "separate" black holes. This is done by centering one of the black holes, and then applying the splitting function with the black hole as the center. This splitting function will leave the black hole we centered on intact, while setting the other black hole to zero. All regions far from the black holes will be multiplied by 0.5 (we will shortly look at the reason).

Our system will now only have one black hole. We can now proceed as normal: Find the value at the collocation points, send the data to LORENE and make the GYOTO file. We will still have some noise in the region where we erased the second black hole, but since the region is zero – instead of e.g. diverging, as it would have been for the diagonal metric elements – the noise is significantly less that if we would have had a black hole there.

We can now center the on the second black hole, and do the same procedure for that, we will now have two GYOTO file, one centered on each black hole with the other black hole set to zeros. If we want to retrieve the actual spacetime we can add the relevant 3+1 quantity from each file. This is the reason we multiplied the regions away from the black holes with 0.5.

We can now look at some more technical details about the splitting function.

6.1 Splitting the Grids and the Splitting Function

We want to make sure that our two grids are well behaved around the object not centered at the origin. We do this by splitting the Einstein Toolkit grid into two grids with the use of the splitting function. The splitting function gives a factor which is multiplied with the grid function we want to spectral transform. This function is given as follows

$$f(\vec{x},\vec{r}_1,\vec{r}_2,R_1,R_2) = \begin{cases} 1 & \text{if } |\vec{x}-\vec{r}_1| < R_1 \\ 0 & \text{if } |\vec{x}-\vec{r}_2| < R_2 \\ S(\vec{x},\vec{r}_1,\vec{r}_2,R_1,R_2) & \text{else,} \end{cases} \tag{6.1}$$

where $\vec{x}$ is a given position in space, $\vec{r}_1$ is the position of the first black hole (centered at the origin), $\vec{r}_2$ is the position of the other black hole, and R_1 and R_2 are balls surrounding the two black holes. We will normally use that $R_1 = R_2 = D/4$ for equal-mass black holes, where D is the distance between the black holes. The function $S(\vec{x},\vec{r}_1,\vec{r}_2,R_1,R_2)$ are comprised of third order smoothing function, and will make it so there is a smooth transition between the two black holes, and so that $\lim_{x\to\infty} f(\vec{x},\vec{r}_1,\vec{r}_2,R_1,R_2) = 1/2$

$$S(\vec{x},\vec{r}_1,\vec{r}_2,R_1,R_2) = \begin{cases} -3\tilde{x}_1^5 + 7.5\tilde{x}_1^4 - 5\tilde{x}_1^3 + 1 & \text{if } |\vec{x}-\vec{r}_1| < R_1 \cdot \frac{D}{R_1+R_2} \\ 3\tilde{x}_2^5 - 7.5\tilde{x}_2^4 + 5\tilde{x}_2^3 & \text{if } |\vec{x}-\vec{r}_2| < R_2 \cdot \frac{D}{R_1+R_2} \\ 1/2 & \text{else,} \end{cases} \tag{6.2}$$

where

$$\tilde{x}_1 = \frac{|\vec{x}-\vec{r}_1| - R_1}{R_1(D/(R_1+R_2)-1)}, \qquad \tilde{x}_2 = \frac{|\vec{x}-\vec{r}_2| - R_2}{R_2(D/(R_1+R_2)-1)}. \tag{6.3}$$

What this function looks like can be seen in figure 6.1.

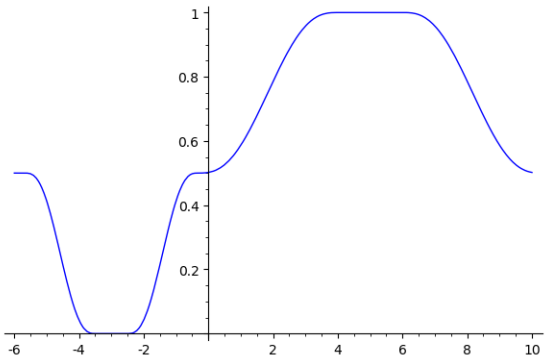


Figure 6.1: Illustration of the splitting function, with black holes at 5 and -3, with $R_1 = 1$ and $R_2 = 1/2$.

This function is $\mathcal{C}^2$, which is necessary when calculating the geodesic of the photon.

Given this splitting function, we can split a given grid function $g(\vec{x})$ from Einstein Toolkit in the following way: $g(\vec{x})_1 = f(\vec{x},\vec{r}_1,\vec{r}_2,R_1,R_2)\cdot g(\vec{x})$ and $g(\vec{x})_2 =$

$(1-f(\vec{x},\vec{r}_1,\vec{r}_2,R_1,R_2))\cdot g(\vec{x})$, where $g(\vec{x})_1$ gets centered on the first black hole and $g(\vec{x})_2$ on the second. We now do the spectral transformation on both of these grids $\tilde{g}(\vec{x})_1$ and $\tilde{g}(\vec{x})_2$. Since the features in spacetime made object not in the center is suppressed, a much more precise spectral representation can be made.

When calculating the geodesics of the photon, GYOTO will calculate $\tilde{g}(\vec{x})_1$ from the first grid, and $\tilde{g}(\vec{x})_2$ from the second. Then using $\tilde{g}(\vec{x}) = \tilde{g}(\vec{x})_1 + \tilde{g}(\vec{x})_2$ it is able to find the geodesics.

Part II

Methods

Chapter 7

Simulating a Black Hole Merger with Einstein Toolkit

The first part of the conversion is to get something to covert, and for this a simulation from Einstein Toolkit is needed. Einstein Toolkit is made to be very easy to run, using only parameter files for the simple systems we are looking at. Binary black hole mergers are also the very reason that numerical relativity and Einstein Toolkit exist, meaning that finding parameter files for these kinds of simulations are very easy. We will look at the parameter files for a isotropic Schwarzschild black hole, a binary merger and the small changes needed to make them work with the converter. In the appendix A the user might find a guide for installing Einstein Toolkit in the same way as I have it installed.

7.1 Single Schwarzschild Black Hole

As mentioned above, to run Einstein Toolkit we need only give it a parameter file describing the spacetime we want to simulate, as well as the methods and output used for the simulation. In this, and the next, subsection we will look at two parameter files, one for a single black hole and one for a binary merger, and explain the most important Thorns and how they are configured.

The first parameter file we will set up a single black hole with a Schwarzschild metric, which can be found in sec. D.1. This will be the metric which we will use to test most of the conversion code, so we want the metric to be as numerical precise as possible, so that numerical errors will be due to the conversion code alone. We will later discuss the main way we tried to ensure this. Note that we will not comment on every thorn and every parameter in this parameter file, only the ones we think make sense to comment, and that is important for our simulation and not something that is in every parameter file.

The first part of the parameter file is used to declare the active thorns.

```
1 ActiveThorns = "
2         (...)
```

```
"
```

Note that this does not need to be done in the start, nor do every active thorn need to be in this one list. We can declare *ActiveThorns* multiple times in the file, and as long as it is done before we use said thorn or any other thorns dependent on it, it should work. But it is easier to read if we start the parameter file with a list of the active thorns.

There many ways of outputting data in Einstein Toolkit. Since we only need the 3+1 quantities for these conversions, we only need to include the following output

```
Activethorns = "CarpetIOHDF5"

# 3D HDF5 Output
CarpetIOHDF5::out3D_every = 2048
carpetIOHDF5::out3D_vars = "
          ADMBase::lapse
          ADMBase::shift
          ADMBase::metric
          ADMBase::curv
"
```

Here we see that the lapse function, the shift vector, the spatial metric and the extrinsic curvature are outputted in 3D. We also see that this is done every 2048th time step. As we will see shortly, the simulation is only 1 iteration long, so the quantities are only outputted at the very first time step.

As mentioned the simulation only run for 1 iteration. This is set as follows

```
Cactus::cctk_itlast = 0
Cactus::terminate = iteration
```

This is done because we are not interested in seeing how the black hole is evolving with time. We are only interested in a snap shot, to see how Einstein Toolkit data is converted. We thus only generate the initial data, run 1 iteration and stop.

Next we need to set up the grid we are doing the simulation on

```
CoordBase::domainsize = "minmax"
CoordBase::xmax = 300
CoordBase::ymax = 300
CoordBase::zmax = 300
CoordBase::xmin =   0.000
CoordBase::ymin =-300
CoordBase::zmin =   0.000
CoordBase::dx   =   2.0
CoordBase::dy   =   2.0
CoordBase::dz   =   2.0


ReflectionSymmetry::reflection_x   = no
ReflectionSymmetry::reflection_y   = no
ReflectionSymmetry::reflection_z   = yes
ReflectionSymmetry::avoid_origin_x = no
ReflectionSymmetry::avoid_origin_y = no
ReflectionSymmetry::avoid_origin_z = no
```

```
CarpetRegrid2::symmetry_rotating180     = yes
```

Here we say that the grid is defined from some minimum and maximum, and that these corners are defined as $\vec{x}_{min} = (0, -300, 0)$ and $\vec{x}_{max} = (300, 300, 300)$. We see that x and z start at 0. This is because we are using symmetries over these axis. This can be seen in the *ReflectionSymmetry* and *CarpetRegrid2* parameters. We can also see that dx, dy and dz are set here. These parameters are quite important since they give the resolution of the grid. Different values for these will be tested to see how they impact the result. Note also that the size of the grid divided by the resolution must be an integer for each axis.

We know that far from the black hole not much happens, but closer we come to the black hole more the metric will change. We therefore make grid refinements closer to the grid

```
carpet::max_refinement_levels             = 10
CarpetRegrid2::num_centres               = 1
CarpetRegrid2::num_levels_1              = 10
CarpetRegrid2::position_x_1              = +0.0
CarpetRegrid2::radius_1             [1] = 128.0 # 1.536
CarpetRegrid2::radius_1             [2] =  64.0 # 0.768
CarpetRegrid2::radius_1             [3] =  16.0 # 0.384
CarpetRegrid2::radius_1             [4] =   8.0 # 0.192
CarpetRegrid2::radius_1             [5] =   4.0 # 0.096
CarpetRegrid2::radius_1             [6] =   2.0 # 0.048
CarpetRegrid2::radius_1             [7] =   1.0 # 0.048
CarpetRegrid2::radius_1             [8] =   0.7 # 0.024
CarpetRegrid2::radius_1             [9] =   0.5 # 0.024
```

This defines some radii from the center where a new grid is created by Einstein Toolkit, with a finer grid resolution. We see that I've used 10 such radii in this simulation.

Now we come to the meat and bone of the setup, namely the point where we define how we want our spacetime to look. There are multiple ways of making a Schwarzschild metric, due to the gauge freedom of numerical relativity. One way is *Two Puncture* (see sec. 3.5) with one mass set to zero. The way we are goinh to do this is to just make the initial metric using an analytical approach. For this we are using the thorn *IDAnalyticBH* [43]. This lets us use analytical metric from common black hole metrics, such as Schwarzschild, Kerr, Nordstrom, etc. This is done with

```
ADMBase::initial_data      = "schwarzschild"
ADMBase::initial_lapse     = "schwarz"
ADMBase::initial_shift     = "zero"
ADMBase::initial_dtlapse   = "zero"
ADMBase::initial_dtshift   = "zero"
idanalyticbh::mass = 1.0
```

The values for the first and second parameters are options from this IDAnalyticBH, which will set up analytical values for the spatial metric, extrinsic curvature and lapse. The shift, as well as the time derivative of the lapse and shift, are set as zero, which is an option with comes from *ADMBase*. The last parameters tells Einstein Toolkit and IDAnalyticBH the mass of the black hole.

Running this parameter file we will get an Einstein Toolkit representation of a Schwarzschild metric. This can then be converted and used for ray tracing.

7.2 Binary Black Hole Merger

Next we want to look at the parameter file for a binary black hole, which can be found in sec. D.2. As mentioned before, one of the main reasons numerical relativity was developed was to solve the problem of a black hole merger. This means that there are a lot of parameter files for binary black holes around. We have used the parameter file for a binary black hole system which can be found accompanying [36] and is one of the example parameter files which will be installed with Einstein Toolkit. We will only changed it to output the same 3+1 quantities as above and changed the size and resolution of the grid.

Most of the parameters described when discussing the single black hole are present in the binary black hole parameter file as well, and build up much of the backbone for the simulation. We will not go over these again, but instead look at some of the changes and additions to the parameter file which adds important information for both initializing the binary black hole, evolving them and outputting some data needed for the conversion. This discussion will not be exhaustive, but will go though the most important parts.

The first and most obvious difference is the different initial conditions

```
ActiveThorns = "TwoPunctures"

ADMBase::metric_type = "physical"

ADMBase::initial_data    = "twopunctures"
ADMBase::initial_lapse   = "twopunctures-averaged"
ADMBase::initial_shift   = "zero"
ADMBase::initial_dtlapse = "zero"
ADMBase::initial_dtshift = "zero"

TwoPunctures::par_b          =  3.0
TwoPunctures::par_m_plus     =  0.47656
TwoPunctures::par_m_minus    =  0.47656
TwoPunctures::par_P_plus [1] = +0.13808
TwoPunctures::par_P_minus[1] = -0.13808
```

Instead of using an analytical metric, we instead tell Einstein Toolkit to use the *Two Puncture Method* (see sec. 3.5) to numerically find the initial conditions of a binary black hole. We can see that mass of each black hole is set here, as well as the momentum – *TwoPunctures::par_P_plus [1]*/*TwoPunctures::par_P_minus [1]*. The parameter *TwoPunctures::par_b* gives the distance between the black holes and the center.

This time we also want to evolve the binary black holes

```
ActiveThorns = "ML_BSSN ML_BSSN_Helper NewRad"

ADMBase::evolution_method         = "ML_BSSN"
ADMBase::lapse_evolution_method   = "ML_BSSN"
```

```
ADMBase::shift_evolution_method   = "ML_BSSN"
ADMBase::dtlapse_evolution_method = "ML_BSSN"
ADMBase::dtshift_evolution_method = "ML_BSSN"

ML_BSSN::harmonicN           = 1      # 1+log
ML_BSSN::harmonicF           = 2.0    # 1+log
ML_BSSN::ShiftGammaCoeff     = 0.75
ML_BSSN::BetaDriver          = 1.0
ML_BSSN::LapseAdvectionCoeff = 1.0
ML_BSSN::ShiftAdvectionCoeff = 1.0

ML_BSSN::MinimumLapse        = 1.0e-8

ML_BSSN::my_initial_boundary_condition = "extrapolate-gammas"
ML_BSSN::my_rhs_boundary_condition     = "NewRad"
Boundary::radpower                     = 2
```

This tells Einstein Toolkit that all of the 3+1 quantities should be evolved using the BSSN formalism. At the bottom we also see that a special kind of boundary conditions are imposed. This is because we have gravitational waves, and thus don't expect Minkowski at the boundaries. The evolution itself will be done using the *Method of Lines*

```
ActiveThorns = "MoL Time"

MoL::ODE_Method             = "RK4"
MoL::MoL_Intermediate_Steps = 4
MoL::MoL_Num_Scratch_Levels = 1
```

Here we can see that this method uses a Runge-Kutta integration of 4th order.

These gravitational waves are tracked by the parameters

```
ActiveThorns = "WeylScal4 Multipole"

Multipole::nradii       = 4
Multipole::radius[0]    = 30
Multipole::radius[1]    = 40
Multipole::radius[2]    = 50
Multipole::radius[3]    = 60
Multipole::ntheta       = 60
Multipole::nphi         = 120
Multipole::variables    = "WeylScal4::Psi4r{sw=-2 cmplx='WeylScal4::
    Psi4i' name='psi4'}"
Multipole::out_every    = 4
Multipole::l_max       = 8
```

This tells Einstein Toolkit to calculate the Weyl scalar (see sec. 3.7), so that the gravitational wave profiles can be calculated later. These will not be important in our case, since we never arrived at the point which we can look at converted spacetimes gravitational waves.

When using the splitting function it is important to know the horizon of the black holes. This is not straight forward to find in numerical relativity, so an approximation is used instead to find the apparent horizon (see sec. 3.6). This is added as follows

```
ActiveThorns = "AHFinderDirect"
AHFinderDirect::find_every = 16
AHFinderDirect::origin_x                                [1] = +3.0
AHFinderDirect::initial_guess__coord_sphere__x_center[1] = +3.0
AHFinderDirect::initial_guess__coord_sphere__radius  [1] =  0.25
AHFinderDirect::which_surface_to_store_info          [1] = 2
AHFinderDirect::set_mask_for_individual_horizon      [1] = no
AHFinderDirect::reset_horizon_after_not_finding      [1] = no
AHFinderDirect::track_origin_from_grid_scalar        [1] = yes
AHFinderDirect::track_origin_source_x                [1] = "
    PunctureTracker::pt_loc_x[0]"
AHFinderDirect::track_origin_source_y                [1] = "
    PunctureTracker::pt_loc_y[0]"
AHFinderDirect::track_origin_source_z                [1] = "
    PunctureTracker::pt_loc_z[0]"
AHFinderDirect::max_allowable_horizon_radius         [1] = 3
```

Note that this is just a taste of the full thorn. The full set of parameters set in the file is about 40 lines, so these are just some important parameters. To find the horizon, the thorn must be given an approximate location and size of the horizon. A numerical method is then used to find the horizon from these initial guesses. This is for the first black hole, and a second block of parameters are also given for the second black hole. We see that this thorn is dependent on an other thorn *PunctureTracker*, which is set up as follows

```
PunctureTracker::track                          [0] = yes
PunctureTracker::initial_x                      [0] = 3.0
PunctureTracker::which_surface_to_store_info[0] = 0
PunctureTracker::track                          [1] = yes
PunctureTracker::initial_x                      [1] = -3.0
PunctureTracker::which_surface_to_store_info[1] = 1
```

This is a thorn that tracks the positions of the black holes as they orbit each other and at last collide. This is used by *AHFinderDirect* because it needs a guess for the positions of the horizons to find the apparent horizons. Since the positions of the black holes change each step, so the initial guess needs to change. This tracker will also output the positions of the black holes, so that converter code can read the positions and apply the splitting function at the correct position.

If we run this parameter file, a simulation of a binary black hole merger will be created. For our conversion only the first steps are needed, so after some iterations the simulation can be terminated, and the data we need will be stored and can be accessed by the converter.

Chapter 8

Conversion of the Data

8.1 Introduction and Overview

One of the bigger challenges faced when ray tracing simulated data from Einstein Toolkit in GYOTO is the difference in the formalisms used by the frameworks. As discussed in section 4.2 Einstein Toolkit uses finite difference on a (adaptive) mesh grid, while GYOTO uses a spectral representation of the functions. This means that we need some code that convert the simulated data from the mesh grid to the spectral grid.

Having a finished simulation from Einstein Toolkit, there are some steps needed to convert the code

1. Read the HDF5 files given by Einstein Toolkit
2. Interpolate over the data, to get the values of the grid functions at arbitrary points (within a boundary)
3. Find the collocation points needed to make the spectral representation
4. Post-processing of the Einstein Toolkit data
5. Do the spectral transformation with the use of the interpolated data and the collocation points
6. Save the spectral representation in a form readable for GYOTO

There are three distinct types of tasks solved here: The first consists of point 1 and two, which handle the data from Einstein Toolkit; points 3, 5 and 6, which handle everything to do with the spectral transformation; and lastly point 4, which has to do with post-processing of the data.

As seen above LORENE can take care of everything that has to do with the spectral transformation, but we still need to take care of the reading of the Einstein Toolkit files, the interpolation and the post-processing.

The first hurdle we need to cross was the reading of the Einstein Toolkit data and its interpolation. Reading of HDF5 files in Python is not that difficult, but since Einstein Toolkit formats the data in its own way – to deal with time steps and the adaptive mesh grid – this was not a trivial task. Thankfully we are not the first having to do this, meaning that there exist tools made for Python to handle this kind of data. The library we are going to use is called PostCactus/PyCactusET [34]. This library can both read the Einstein Toolkit data and interpolate it[1], and was used to read the grid functions.

The position and apparent horizons of the black holes are also needed for the spectral grid as well as the splitting function. This was handled directly by our own code, instead of going through any external library.

With the grid functions, the apparent horizon and the black hole positions handled, the next step is to make the Python script able to communicate with LORENE. This was done by simply calling a LORENE script as a subprocess using Python. When the Python program needs the collocation points, the LORENE script is called and the out-stream of this script was caught by Python, where it is processed to reveal the points. The Python program then does all the interpolation of the grid functions on these collocation points, and the application of the splitting function. The results are then written to a file (with a certain format), and the LORENE script is again called, with the information that it now have to read the processed data and do the transformation.

Having the processed data, LORENE can now do its magic. Making a scalar for the lapse function, a vector for the shift and a tensor for the spatial metric and the extrinsic curvature, the LORENE script can read the files and fill the components of these objects. Having these it can quickly do the spectral transformation and save the results as a file readable to GYOTO.

We will below go through all the points in more detail, and see how all these steps are coded into the converter code. All features discussed below is part of the conversion code. Features from PostCactus are discussed only where explicitly mentioned.

8.2 Reading Data From Einstein Toolkit and Making Interpolations

8.2.1 Reading the Data

The first step of the conversion is to read the simulated data from Einstein Toolkit. Einstein Toolkit uses different file formats to save the simulated data[2], depending on on the dimensions of the saved data, as well as the users choice. For one and two dimensions simple ASCII files are normally used. For the more complicated 3−dimensional cases, as the one we will use here, it is more common to use *Hierarchical Data Format version*

[1] It also have a lot of other, quite practical capabilities, but none that was used at this point.

[2] Here we will just focus on the files that hold the 3+1 quantities. Other quantities like the pressure or ψ_4 will use different conventions.

5 (HDF5 or H5) files. These files can be quite complex, especially since Einstein Toolkit uses adaptive mesh grids. There are good libraries for using HDF5 files in Python[3] and with some elbow grease it would be possible to write code to read the Einstein Toolkit data. But thankfully Einstein Toolkit is large enough that other people have done that before, namely *PostCactus*[34]. This library is able to read many types of Einstein Toolkit, as well as post-processing it for easy plotting. Even though PostCactus can read and interpolate the data, this involves a lot of calls and parameter setting for the user, so we are going to wrap everything in it own class, to make it easier for the user.

Said class first needs a directory where the simulation is found. This is given as the root folder of the simulation. Next we have to decide how to interpolate the data. Following the instructions of PostCactus we have to define a uniform grid on which to read the Einstein Toolkit data and interpolate it. This is due to the adaptive mesh grid. To be able to do the full interpolation PostCactus will interpolate the adaptive mesh points on to the uniform grid and from there do a final interpolation of the data, which we can access. Inside PostCacus, the gird class is a complicated class that holds a lot of information about how to get function values from the grid. This is completely hidden from the user of the converter. We just have to give the corner of the grid we want to use, as well as the number of grid points. Note that the corners of the grid must be inside the simulated domain (discussed below). If the grid is bigger than the simulated domain, then PostCactus will give the points outside the domain the value 0. This will lead to wrong results! As we will see later, this is the way PostCactus is intended to be used. We will in most of the results instead use another method(see sec. 8.2.4, which is used internally by PostCactus, but is not meant for users to use, therefore lacking some features.

8.2.2 The Simulated Domain

There is an important terminology we will use a bit below, and that is crucial for the result of the conversion. This is the *simulated domain*. Einstein Toolkit will, of course, simulated only a finite spacetime. When we have, e.g., a black hole or a merger the most interesting part of the simulation is close to center. This means in most cases a large part of the spacetime we need to ray trace is not simulated by Einstein Toolkit. It is also so small that the metric has yet to converge to Minkowski. This means that we need to extrapolate the metric (discussed below). How good this extrapolation is, is dependent on how much of the spacetime is simulated by Einstein Toolkit. This part, the spacetime simulated by Einstein Toolkit, is what we call the *simulated domain*. An illustration of the relation between the simulated domain and the other domains are found in fig. 8.1.

8.2.3 Creating an Interpolation

The final step of reading the data is to make a callable function that we can use to get the grid value at an arbitrary point. This can either be in between the defined

[3]https://docs.h5py.org/en/stable/

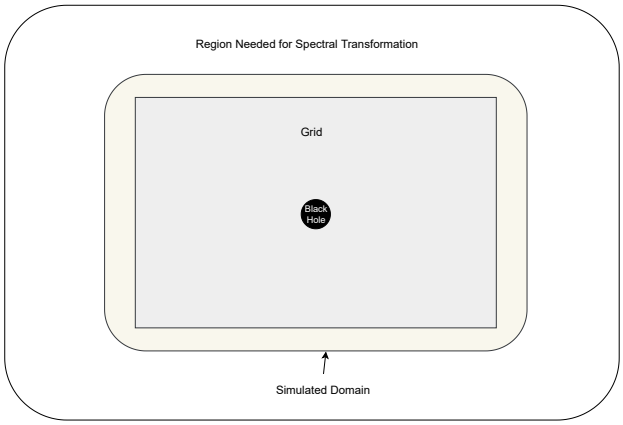


Figure 8.1: Illustration of the relation between the simulated domain and the other domains of the simulation. In the center we find the black hole(s). They exist inside the spacetime simulated by Einstein Toolkit (simulated domain). This is the brownish while field in the illustration. The grid made by the user must be contained by the simulated domain, or else the grid outside the domain will be filled with zeros. The outer most region contains all the other regions and is the values needed by the spectral transformation. Values found here, which are outside of the simulated domain, must be extrapolated from the simulated domain.

gird points or even outside the grid. This means that we need both interpolation and extrapolation. As of now this is done in three ways. The first is based on the way PostCactus does it. But instead of using the function from PostCacus we are going to refactor a bit of the code to do this. This was done so that we can call on the class instance directly with a coordinate (given as a list or an array) and get the value of the grid function, and so that the splines made by the code can be pickled (see below). This way of doing interpolation gives a good interpolation, with splines of order 2-4, giving good results inside the simulated domain. The major problem with this is that the *SciPy* functions used can only extrapolate with a constant value, meaning that we end up with a discontinuity at the end of the simulated domain. The second way instead used linear interpolation, given by SciPy. This interpolation will both interpolate and extrapolate, both with a linear approximation. This means that the interpolation is worse than the above spline, but the extrapolation is better. It is still a problem outside the metric of black holes isn't linear (8.3) outside the simulated domain, so we can risk that the metric can become lower than one and even below zero far from the center (where we expect Minkowski).

Since we have the two ways of doing inter/extrapolation, we might want to use the two methods at different parts of the spacetime, e.g. a precise spline interpolation close to the black hole(s) and a linear extrapolation far away and outside the simulated domain. For this reason we have made two classes for the interpolation/extrapolation (one for each method), and one class that wraps these two, and which is used by

the user. This is named *ReadQuantities*. This class takes a list of geometries and a list which contains the time of the interpolation/extrapolation (linear or spline). ReadQuantities will then make a interpolator for each grid, which is either linear of a spline. We can also give a list of limits for where we want ReadQuantities to switch between the interpolations. If no limits are given, then when we call on the instance to get a function value at a point, ReadQuantities will find the best interpolator to use and give a function value. The way ReadQuantities find the "best" interpolator is quite naive: It looks at the limit of each geometry and assumes that the geometries with corners closer to the center is the one with best resolution. This means that it will use the smallest geometry containing the point given by the user. Since black holes are most complex closer to the center and less resolution is need the farther out the point it, this logic is taken to be sufficient.

8.2.4 The *None* Geometry

There is a third way of reading and interpolating the data from Einstein Toolkit. If we don't give any geometry, or more precisely sets the geometry to *None*, then PostCactus will read the data from Einstein Toolkit using the geometry which Einstein Toolkit used to simulate, and in which the data is stored. This means that there is no initial interpolation to a grid, which is good. The downside is that there is only a linear interpolator with a constant extrapolator given by PostCactus for this kind of data. While this seems to make the spline interpolation superior, this is not necessarily the case, since PostCactus has already used the linear interpolation to make the uniform grid.

This way of reading the data is wrapped in the conversion code in such a way that the user only needs set the geometry parameter in ReadQuantities to *None*. The rest will be taken care of by the code, and everything should work the same way as if the user told the converter to use one or more grids. It is as of now not possible to have a none-geometry together with other geometries, but this will be added. It is also not possible to pickle the interpolation function for this geometry (see below) since no interpolation object is made, and the interpolation is instead done on request.

As we will see in the results, this is the best way of reading data for most data. The reason all the other interpolation code is included and discussed in details, is because the none-geometry was found – which from the side of the creators of PostCactus is internal code and not suppose to be used by the users – late in the master thesis. We also hope that the other integration methods has their place when using the converter.

8.2.5 Geometries Used Later

Throughout the results we will be discussing and comparing three "types" of geometries. The three are single grid, multigrid and none-grid. As mentioned above, the conversion code can use a list of grids when making a the interpolation. From this comes the difference between single grid and multigrid geometry. The single grid is, as the name suggests, just a single grid. The multigrid is any geometry where more than one grid is

used. We would expect this to give a better result as we have more room to define the resolution for different parts of the spacetime. The last geometry is the none-geometry and is the geometry discussed right above.

These three terms will be used ad nauseam in the result section, so it is important to be able to tell them apart.

8.2.6 Pickling the Interpolation

The process of reading and interpolating the data can be a bottleneck, depending on the geometry. For a first time conversion of the Einstein Toolkit data, this is more or less inevitable. But as we will see below, there are a lot of parameters the user can set which impacts the data after it has been read from the HDF5 files, so the we might have to rerun the conversion with parameters that don't impact the reading of the data. To get around this the converter can pickle objects holding the splines. These pickles can then be read by the class at the next run of the conversion (given that the uniform grid is the same). This speeds up this part of the conversion with a factor 20x. We can choose whether to pickle the results or not. Note that the pickles can become very large, so for full simulation they might be too big to load into memory. But for smaller simulations and parameter tests the pickles decrease the runtime significantly.

8.3 LORENE and the Spectral Transformation

8.3.1 Finding the Collocation Points

One of the tedious tasks to have to implement is the actual spectral transformation. As seen in section 4.3 this involves a good deal of compactification of coordinates and fast Fourier transformations to get the spectral transformations.

Thankfully LORENE has most of these capabilities build into it, in a way which is easy to use. LORENE is able to create spectral grids centered at a given position, then give collocation points needed to do the spectral transformation. Given the grid function values at these points, LORENE can, with only one command, transform into a spectral representation.

This means that if we can use LORENE in our conversion code, we will save a lot of work. The big hurdle here is that the conversion code is in Python, while LORENE is in C/C++. There are a lot of libraries to be able to call C/C++ functions in Python – Boost, CTypes and Cython being the ones coming first to mind –, but these takes some effort to get to work with the somewhat involved LORENE code. We will therefore use a more messy but purely Pythonic way of running the C-code. The intent was to only use this temporary, but as time went by temporary became permanent...

8.3.2 Retrieving and Formatting the Collocation Points from LORENE

Instead of calling on the C/C++ functions which run the LORENE code, we instead call on a separate C program using a sub-process, and catch the outgoing stream. The

C-code, in turn, takes a some command-line arguments, telling the program the center of the coordinate patch, and some other arguments having to do with the running mode (the first running mode is described here, the other mode is described below). If the program is told to create the spectral grid, it will use LORENE to define a map of a given size and resolution. This size and resolution is for now hard coded into the C program, so that we have to change the parameters by hand in this program(see sec. 8.8.2). The program will then center the map on the given coordinates of the center, get a list of the coordinates needed for the collocation points and finally print them to screen. After the C-code has made found the collocation points and printed them to the screen, the Python code can catch said stream, and knowing the format at which they are printed, read through the stream and get an array holding all the relevant collocation points.

The details of how the output stream is formatted and how the converter handles it is not important. *Regular Expressions* (Regex) could have been used[4], but instead we just split the string at the relevant places to get the points, since we already know exactly the format of the string. The important part is that resulting array of collocation points is an array consisting of three array, one for x, one for y and one for z. This is turn are arrays with indices representing the domain l, j indicates θ , k indicates ϕ and i indicates r. So if the collocation points is at $r[2]$, $\phi[3]$ and $\theta[0]$ in the second domain, the corresponding Cartesian coordinate we need to evaluate is

$$\text{Collocation Point} = (x[l=1][k=3][j=0][i=2], y[1][3][0][2], z[1][3][0][2]).$$

We now have all the collocation points needed to make a spectral representation. The next sections will discuss how the values are found, and how they are sent back to LORENE for the final transformation.

8.4 Getting the Values at the Collocation Points

We now have a class instance which we can call to get the grid functions from Einstein Toolkit on any arbitrary point, and we have a list of all the collocation points needed by LORENE to do a spectral transformation. So now we can just call on the instance to get the values at the collocation points? No.. We need some more steps to make sure what the grid values are correct and ready for LORENE.

8.4.1 Handling the Boundary

The first two steps are due to the way Einstein Toolkit simulates the spacetimes. Firstly, as mentioned, Einstein Toolkit simulates only a finite spacetime, with a abrupt cut-off point. LORENE and the spectral method on the other hand extend the spacetime out to infinity (though compacted). This means that, depending on the resolution we want

[4]There is a famous saying among programmers which goes "*I just found a problem which can be solved with Regex; now we have two problems...*". Regex is very powerful, but notoriously difficult to use.

LORENE to use, we may get collocation points outside of this cut-off point. As of now this is solved in the crudest way possible, by just giving every points outside the cut-off the same value as the cut-off points. This assumes that spacetime is sufficiently flat at this points that we get away with this. We might also think that even if the spacetime is close to flat, but not entirely, this method might lead to discontinuity in the resulting data. Here the spectral method comes to out rescue, since a spectral transformation is an interpolation. Meaning that even though the data we give it is taken from a discontinuous function, the resulting spectral representation is continuous. So as of now we take this assumptions to hold, but this may have to be changed later, especially if the assumptions show to be false or if more extensive spacetimes are used.

This changes a bit when we use the extrapolation. If we have a linear extrapolation this will be needed, since this might lead the linear approximation to overshoot the Minkowski metric. When using a constant extrapolation, this might not be necessary, but the feature will still be in the code, and will only bound the coordinates at large values.

The main reason for having this feature is that LORENE will request collocation points at infinity. This will, if not handled, cause an error in Python. Thus we must catch the infinities and instead set them to some large number.

8.4.2 Handling the Symmetries

The second fact is that Einstein Toolkit tries everything in its power to reduce simulation time. This means that it uses a lot of symmetries to make the simulation run faster. The most used symmetries are either simulating only 180° and mirroring around one axis, or simulating only one quarter of the spacetime and mirroring around two axis. For a simple Schwarzschild black hole, or even the simplest black hole merger with two identical black holes, the latter method is used. In fig. 8.2 we can see an illustration showing the xy-plane of the simulation given by Einstein Toolkit. Only the original is given by Einstein Toolkit, the rest has to be handled by the converter. This means that we only get one quarter of the data describing the spacetime. The converter code will then simply mirror the data over one or two axis, depending on what the user gives as a parameter (this assumes that the user knows the symmetries of the simulation).

8.4.3 Applying the Splitting Function

The last step has to do with the spherical nature of LORENE, as discussed above(see sec. 6. If we tell the code that we are dealing with a binary, after the extension and mirroring is applied to the collocation points, and the grid value is found, the splitting function (6.1) is multiplied with these values. The splitting function takes the coordinate of the collocation points, the positions of the black holes, and a factor R which determines the size of the smoothing, which by default is the distance between the black holes divided by 4. The position and size of the black holes comes from the data created by the horizon tracker(sec. 3.6. We have now smoothed out one of the black hole, meaning that it is a bit more spherical symmetric.

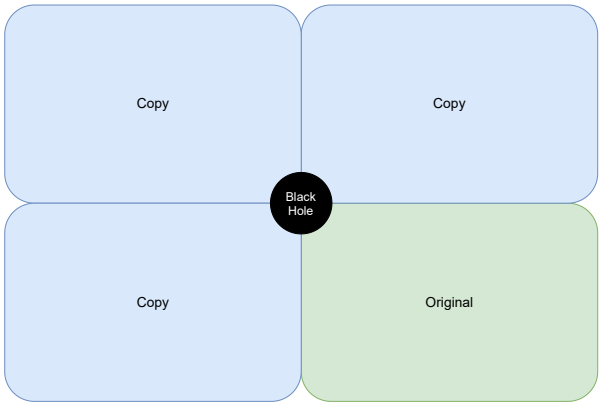


Figure 8.2: Illustration of the xy-plane of the simulation. Only the part marked *original* is given by the Einstein Toolkit simulation. The copies have to be made by the converter. This is done by simply mirroring around the z-axis.

Since we are going to split the binary black holes into two grid, each centered on one of the black holes, we need to find the position of the black holes. Einstein Toolkit also needs to keep track on the positions of the black hole, and their horizons(sec. 3.6). This means that all the information we need about the positions and horizons of the black holes already exist as time series in the same folder as the grid functions. This means that we can read them straight from these files. The code for doing this is taken from example code found in [36].

If we tell the code that we are only converting a single object, such as a single black hole, this part of the conversion is omitted automatically. We can also choose not to apply the splitting function when converting the black hole binary, but this serves only a illustrative purpose, and will be used to show that the splitting function is necessary.

8.4.4 Flattening the Results

We can now write the values at the collocation points to file and hand it over to a C program running LORENE. The files are chosen to be flat, so to be easier to read back in C. The array is flattened in the following way

```
for index,domain in enumerate(d):
    for k in domain.keys():
        for j in domain[k].keys():
            for r in range(len(domain[k][j])):
                flatten_values.append(d[index][k][j][r])
```

This means that in the C-code we can reverse this flattening with the following

formula

$$d[l][k][j][i] = \text{flatten\_array}\left[\left(\sum_{0<=m<l} nr[m] * nt[m] * np[m]\right) + k * nt[l] * nr[l] + j * nr[l] + i\right], \tag{8.1}$$

where $nr[l]$ is an array holding the resolution of r in domain l, and $nr[l]$ and $np[l]$ are the same for respectively θ and ϕ. These arrays are know to the C-code. We now have a way of finding the values at the collocation points and saving them to a file. Next is to do the spectral transformation with LORENE.

8.5 The Final GYOTO Formatting with LORENE

8.5.1 Running the Last Conversion

We now have one file per quantity, each containing the function value at the collocation points for said quantity. We now need to do the spectral transformation. As mentioned above, LORENE can do this for us. The same C code that gave us the collocation points have another mode that lets us do the spectral transformation and create a file usable by GYOTO.

There are two ways of running the code to make the GYOTO file. The first is do let the conversion code do it for you, by setting the `do_gyoto_conversion` to true when running the `analyze_bbh` function. The conversion code will then run the C code using a sub-process at the right time, with the right parameters. The only thing the user must be careful with is that they have converted all the 3+1 quantities, or else the code will crash. The second way is to run the C program in the terminal. Here the user have to manually give the origin of object, the mode of the program (1 for final conversion), whether it is the first or second body and the number of the time step. This will then create the GYOTO file.

Note that since we have split the two (or more) black hole, we will have one file for each black hole. They are later combined inside of GYOTO (see sec.10.1.2.

8.5.2 What Happens Inside the C Code

When the C program is called, either through the converter or the terminal, it will go through a couple of steps to do the spectral transformation. The first step is to create the objects which can represent the 3+1 quantities. This is done with the LORENE objects `Scalar` for the lapse α, `Vector` for the shift β_i and `Sym_tensor` for the 3-metric γ_{ij} and the extrinsic curvature K_{ij}. The program will then read in the files containing the 3+1 quantities. All the above objects are then filled with these quantities. Since each component of these objects are a function of the above described domain index l, ϕ index k, θ index j and r index i. This means that the program will have to reformat the flatten array, read from the file. A function will therefore use eq. 8.1 to do this.

All the objects (tensors) are now filled with their respective components. With the simple `object.std_spectral_base();` LORENE will now do the spectral trans-

formation using the information about the function values at the collocation points. We now have a spectral representation of the grid functions. But as mentioned above, GYOTO use a spherical coordinate. This means that we have to do a change of basis, to a spherical triad. Again, this is done with a simple command in LORENE: `.change_triad(map.get_bvect_spher())`.

We now have the objects we need to do ray tracing. The final step is just to make a file usable by GYOTO. LORENE lets us make a file, and simply save the objects to that file, with the `object.sauve(file)` command. We must also add `Metric(gamma).con().sauve(file_out)`, this simply saves the inverse metric to the file, which is needed in GYOTO. After saving all the objects to file, we have successfully made a file usable by GYOTO!

Note that we also can do a lot of plotting with LORENE. Many of the plots found in the result section of the data after conversion, are made with LORENE.

8.6 Parallelization

Even after removing the bottleneck for reruns of the conversion, we want the conversion to run faster. To do this we parallelize the program with *Message Passing Interface*(MPI)[5], which lets us run the program in parallel on different core. We will not parallelize the C code, but rather the Python code, meaning MPI is the easiest choice.

As mentioned above, each 3+1 quantity is read and converted independently of the others. This means that we quite easily can parallelize the conversion. A wrapper for the conversion and MPI was then made, *mpi_converter.py*. This program simple distribute the 16 different 3+1 quantities (or fewer if the user don't want to convert every quantity), among the number of cores the program is given – limited be either the number given by the user, the number of cores on the used computer or the total number of quantities. The program then just runs the conversion independently on each core. This gives a speedup of up to 16x. Note that conversion code will not be able to run the final formatting to GYOTO, since cores converting each quantity finish at different times. This instead has to be done manually.

There is another way of running the code in parallel. The value for each collocation point is also found independently of all the others, meaning that be can distribute the task for finding the values to different cores or threads. This will need a more invasive refactoring of the code, and is thus not done yet. The conversion runs much faster than the ray tracing done by GYOTO, so further optimizing is not needed yet.

8.7 Test Cases

When testing if the code works there many steps that can go wrong. The main steps that might go wrong are either the reading and interpolation of the data, retrieving of

[5]https://mpi4py.readthedocs.io/en/stable/

the collocation points, bounding the coordinates and the final processing by LORENE to make the GYOTO files. To be able to test the code without the limitation of using simulated data, a couple of test cases are implemented. These test cases consist of analytical metrics, meaning that when the converter finds the values of collocation points a function is called, giving the analytical value at each points, instead of having to read the Einstein Toolkit data and bound the coordinates. This will therefore test all the implementation of LORENE.

The first test is a simple Minkowski metric

$$\alpha = \gamma_{ii} = 1, \qquad \beta_i = K_{ij} = 0. \tag{8.2}$$

This is a trivial metric, but gives us the possibility to manually plot and inspect the values of the metric after the conversion using LORENE – this is of course possible with all metrics, but the constant value of the Minkowski metric makes it easy to spot errors in metric components arising from the conversion.

The second test is a Schwarzschild metric in isotropic coordinates

$$\alpha = \frac{1 - \frac{1}{2r}}{1 + \frac{1}{2r}}, \quad \gamma_{ii} = \left(1 + \frac{1}{2r}\right)^4, \quad \beta_i = K_{ij} = 0. \tag{8.3}$$

This lets us test a more interesting metric. GYOTO has a lot of its own test cases for black holes, both analytical and simulated with LORENE. This means that there are a lot of data we can compare our converter with. So after using our converter on the above metrics, we can do ray tracing on them, and compare the results with raytracing done in metrics made by LORENE.

Note that the above test cases are for cases with a single (or no) black hole. This is in other words only tests the pipeline with LORENE, and not the overall method with splitting. But if the tests gives good results, then we can assume that the LORENE pipeline is correct, and any errors occurring in the two-body system should be due bugs in other parts of the code.

8.8 Parameters Used in the Conversion

There are a lot of parameters that go into the conversion. Most of them are given when the user calls on the conversion function in the Python code. Some of the parameters are also found in the C code, and need the C code to be recompiled. We will here discuss the parameters which have an impact on the conversion. Parameters like whether or not to pickle, the simulation folder, etc. is discussed in the documentation. An overview of all the parameters are found in tab. 8.8.

8.8.1 Parameters found in the Python Code

Inside the Python code we find parameters that first and foremost have to do with the reading of data from Einstein Toolkit. Maybe the most important parameter the

user sets is the type of geometry. In most cases the none-geometry will yield the best results, but there might be other cases where another type of geometry is preferred. In this case the user needs to set is the size and resolution of the geometry which the Einstein Toolkit will be interpolated on. The size needs to be big enough that it encompasses most of the spacetime, but if it is bigger than the simulated domain, it will fill the extra points with zeros. The resolution defines how many points the geometry will have. Generally the more the better, but this of course comes at the cost of runtime and memory usage. In this case the parameter determining limits also becomes important.

8.8.2 Parameters found in the C Code

The second set of important parameters is found in the C code. As we saw in sec. 4.3 the spectral transformation is an interpolation where the degree (and thus precision) is determined by the number of collocation points we use. This number of collocation points is something the user has to define in the C code. Remember as well that we split the spacetime into different domains, where the resolution can be different depending on the complexity of the spacetime in said domain. So the user will first have to define the number of domains they want to use; secondly they need to define the r limit of each domain, and lastly they must define the number of collocation points in each domain. As for the domain limits, it is wise to define the domain so one domain contains one feature of the spacetime: so, for example, one region which holds the center black hole, one that contains the region in between the black holes, one that contains the second black hole and one that contains the region outside the black holes. The number of collocation points will also need to be greater for regions where the spacetime varies a lot and varies non-monotonic. These parameters are the ones that has the most effect on the results. With a low number of collocation points, the conversion will give rise to artifacts in the resulting spacetime.

8.8.3 Future Plans

As mentioned the user will have to set the parameters in both the Python code and the C code, and then recompile the C code. The plan is to implement a wrapper for the whole library, which makes it easier to use for the user. This will let the user just fill a parameter file with all the discussed parameters (and some more to do with folder and saving of the splines), and then run a terminal command to run the conversion, either from scratch or rerun an existing conversion with different parameters.

Name	Type	Example	Found in	Effect
Grids	List of Lists or `None|`	$[[[x_1, y_1, z_1], n_1], [[x_2, y_2, z_2], n_3], \ldots]$ or `None|`	Python file, when ReadQuantities are initialized	Higher resolution gives high runtime, low resolution gives bad interpolation of metric
Limits	List	$[10, 20, 250]$	Python file	The limits where, in the case of a multigrid, ReadQuantity switches between grids.
Interpolation Type	List of Strings	["linear", "spline", ...]	Same as Grids	Important for extrapolation, and interpolation near center.
Spline Order	Int	4	Same as Above. Only used for spline interpolation	Smoothness of interpolation
Number of Spectral Domains	Int	nz=6	Found in C code. Need compilation after change.	Higher number gives better spectral representation, but higher runtime
Number of Points in each Domain. One each for r, θ and ϕ	Number of Ints	$[52, 52, 52, 22, 12]$	Same as above	Same length as number of domains. Higher numbers give better resolution but higher runtime.
Domain Limits	List of doubles	r_limits[] = {0., 2, 4., 6, 8, 20, __infinity}	Same as above	Should be one longer than the number of domains. Each domain should encompass one interesting part of the spacetime. Last should be __infinity

Table 8.1: Table showing the most important parameters, how to use them, where to find them and how this affects the conversion. These are only the ones which directly impact the conversion. There are many more parameters which has to do with folder, reading and writing data. These are found in the code documentation.

Chapter 9

A Closer Look at the Conversion Code

As mentioned, the code for the converter consists of Python code and C code. The code itself can be found at the Github repository `https://github.com/dulte/Master`. We will first take a look at the structure of the code, and where to run the code from. We will so quickly look at how to run the code, and how to use the most usual features in the code. A deeper look at what is run when, and how the methods inside the code are called, can be found in appendix B.

9.1 Structure of the Conversion Code

The code is yet to be made into a proper Python package, so as of now the structure of the code is quite rigorous. In the source folder we can find the following structure

```
src/
    - C/
        - old_code  (Of no importance)
        - Makefile
        - get_point.C
    - Python/
        - old_code  (Of no importance)
        - Analytics/
            - error_plotter.py
            - image_analytics.py
        - EinsteinToolkitToGYOTO/
            - example.py
            - mpi_converter.py
            - requirements.txt
            - ET2G/
                - __init__.py
                - ReadQuantities.py
                - ETQuantities.py
                - ETInterpolater.py
```

The two must important folders here are `src/C/` and `src/Python/EinsteinToolkitToGYOTO/`. The first holds the necessary C code. If the user wants to change come of the C-code parameters, then the important file `get_points.C` can be found here. This is the C code that runs LORENE. As long as the user has installed LORENE and GYOTO, it should be enough to run the Makefile to compile the code.

The second folder contains the Python package, called EinsteinToolkitToGYOTO or ET2G. In this folder we can see two files called `example.py` and `mpi_converter.py`. These files show the normal ways of using the conversion code. All programs using the conversion code must be running from here!!! This is because of the hard coded paths in the code. This will all be changed in the final wrapper/user interface. Here we can also find the requirements for the Python code.

In the folder `src/Python/EinsteinToolkitToGYOTO/ET2G/` we find the actual converter code. `ETQuantities.py` contains the two classes that can read the Einstein Toolkit data and interpolate it, either using linear interpolation or a spline. `ReadQuantities.py` is a wrapper wrapping the two ETQuantities classes, and makes it easier for the user to use. This is the class that should be used for the reading. `ETInterpolater.py` holds the last class, and is the code responsible for all the rest of the code: Communicating with LORENE, evaluating the collocation points, handling symmetries and boundaries, etc. The user will use ReadQuantities and ETInterpolater together, as the former will read and interpolate the data while the later will use the data to do the spectral transformation. Figure 9.1 shows how the classes are connected and their methods.

9.2 Using the Code

Remember to install all the requirements before using. For the C code LORENE and GYOTO is needed. For the Python code the requirements are found in `requirements.txt`. Most of the packages should be installable with *pip*. PostCactus can be found at [34]. Remember that this code is for Python2.7, due to the fact that PostCactus wasn't ported to Python3 when this project started.

In *example.py* we can find examples on how to run a conversion of analytical metric as well as reading and plotting some simulated data. We will not look at that here, but from what we will be discussing, the other examples will be self evident.

We will here be looking at some of the code used in `mpi_converter.py`. This is a *MPI4Py* code for running the conversion in parallel. We will ignore the middle code, which sets up MPI and distribute the quantities among the cores as best as it can. The interesting code is that at the bottom. To run a conversion we can use the code

```
1 nb_bodies = 1
2 linear = True
3 smooth = True
4 it = 0
5
6 inter = ETInterpolater(folder, nb_bodies)
7
8 g = None
9 limits = [300]
```

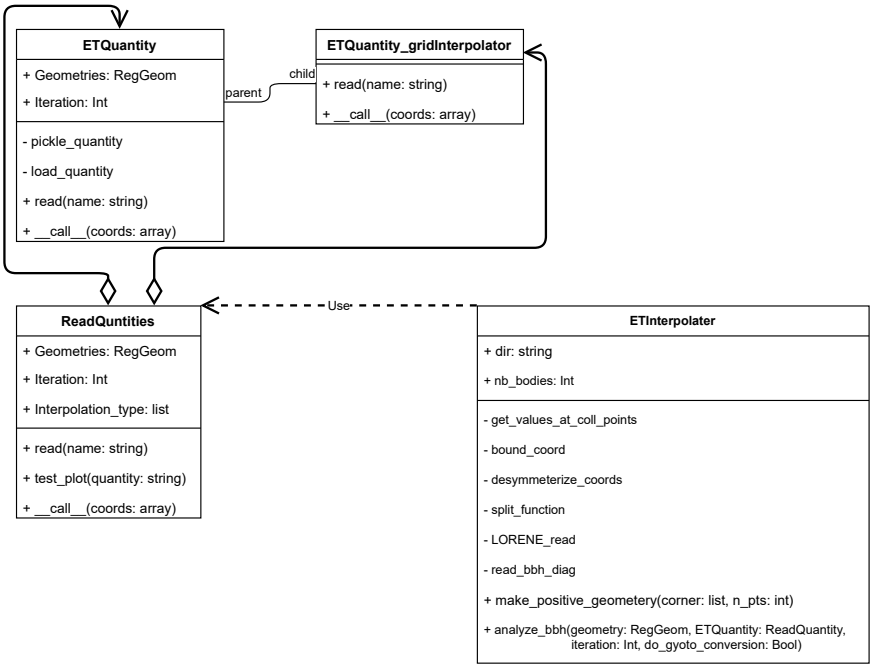


Figure 9.1: A simplified class diagram showing the most important parameters, methods and relations between the different classes in the Python code. We see here that *ReadQantities* is build upon *ETQuantities* and *ETQuantities_ gridInterpolator*, where the latter of the two is inherent from the former. *ETInterpolater* uses ReadQuanties, as is shown here. Not all private methods and parameters are shown here, just the important ones, which does something discussed in the text. The private methods do not have all the parameters listed, since they are not important for the user.

```
10
11 et_q = ReadQuantities([g], it, folder, pickle_folder=pickle_folder,
       pickle=False, linear=linear,limits=limits)
12
13 inter.analyse_bbh(g, et_q, [it],quantities=rank_quantities, test=False,
       do_gyoto_converstion=False, split=smooth)
```

This code will set up a conversion of a single black hole using a none-geometry, and then run the conversion without the final conversion to the GYOTO file. The first four lines gives some basic information to the code: We have one object to convert, we will use a linear interpolation (since we have a none-geometry this is unimportant), we want to apply the splitting function (since we have only one object, this will be ignored) and we are at iteration 1 of the Einstein Toolkit simulation. Line 6 makes an instance of the ETInterpolater class. This is normally done first since it makes the geometries. In line 8 we set the geometry to be *None*, which means a none-geometry. Line 11 then reads the data. The *folder* parameter is the path to the Einstein Toolkit simulation. Finally, at line 13, we tell the ETInterpolater instance to analyse and convert the data. This is called *analyse binary black hole* since this code first was made for binaries, but it will handle single objects as well. Notice that `do_gyoto_converstion=False`, which means that the final conversion to the GYOTO file won't be made. Instead the user is left with one file for each quantity for each object, and has to do the conversion them self, using the C-code

```
1 ./get_points x y z mode body it
```

Here x, y and z are the center of the black hole, the *mode* should be 1 to do the final conversion, *body* is 1 and *it* should be 0 as this is the first iteration.

If we instead want to use a multigrid, we can change the relevant parts of the code to

```
1 g0 = inter.make_positive_geometry([-10,-10, -10], 100)
2 g1 = inter.make_positive_geometry([-50,-50, -50], 100)
3 g2 = inter.make_positive_geometry([-200,-200, -200], 200)
4 limits = [5, 50, 100]
5
6 et_q = ReadQuantities([g0,g1,g2], it, folder, pickle_folder=
      pickle_folder, pickle=False, linear=linear,limits=limits)
```

This will create three different geometries. The limit list tells code when to switch geometries. For a single grid, we only need give one geometry and one limit.

The Python parameters can be changed in the way we have seen here. The C parameters will have to be changed inside of the `get_points.C`. Here the block

```
1 /*
2 ////////////////////////////////////////////////////////
3     User defined variables
4     Change to change conversion!
5 ////////////////////////////////////////////////////////
6 */
7 int nz = 6 ;  // Number of domains
8
```

```
9  // Domain Resolutions
10 int nr_array[]  = {25, 25, 25, 25, 25, 25};
11 int nt_array[]  = {7,7,7, 7,7,7};
12 int np_array[]  = {4,4,4, 4,4,4};
13
14 //Type of Domain
15 int type_r[] = {RARE, FIN, FIN,FIN,FIN, UNSURR};
16
17 // Domain Limits
18 double r_limits[] = {0.,0.51, 1, 2, 4, 8, __infinity} ;
```

can be used to change the parameters. This can be found at around line 35 in the code. Remember to compile afterwards. Also note that `type_r` and the domain resolution arrays need to be at the same length as `nz`. The code will not necessary tell you is this is not done, and will lead to much unnecessary debugging...

Chapter 10

Adapting and Using GYOTO with Converted Data

We have now looked at how to simulate and convert the Einstein Toolkit data. We can now look at how to use GYOTO. We will first look at some changes that need to be added to GYOTO to make it work with our metrics. We will then look at how to set up a parameter file for GYOTO and run it. We will also discuss the outputs and how this is used to evaluate the results.

10.1 Changes Made to GYOTO

10.1.1 Using GYOTO without Spherical Symmetries

GYOTO was first and foremost made to use either analytical metric 5.4.1 or metrics simulated in LORENE 5.4.2. Both of these options are metrics with a spherical topology. This leads to lots of simplifications if both the metric and especially the equations of motion (5.13), where all the dependencies on ϕ and θ disappear.

Einstein Toolkit uses some symmetries when running the simulations (see sec. 7.1) to save on time and computational resources, but these are at best a 180° mirroring of one of the Cartesian axis and not a spherical symmetry. The whole point of introducing a splitting function was to circumvent the spherical symmetries of LORENE and GYOTO. So this meant that GYOTO needed to be extended to use metric without spherical symmetry.

Most of difference between with and without spherical symmetries is in the Christoffel symbols. Thankfully the creators of GYOTO have calculated all the Christoffel symbols for this general case. This was circulated in an internal document and never published. In the appendix F I've added the parts of this document which give said Christoffel symbols. This is done with the permission of the author.

10.1.2 Using Two Metrics in GYOTO

For our two metrics describing the binary black hole system, some changes are needed to GYOTO. As we have discussed above, the metric is split up in such a way that the sum of the metrics/files becomes the whole metric. Since GYOTO is made to use only one file as the metric per time step (as would be expected from a normal simulation), changes to GYOTO are needed to get this system to work.

GYOTO should be able to read the two files independently, and then, when integrating the photons, it should ask for the value for each 3+1 quantity from each of the files. GYOTO can then sum the two values for each quantity and get the actual metric at this point. These changes are not yet implemented to GYOTO. This means that we won't be able to test out the ray tracing on the binary black hole system. The implementation is being worked on, and will be made usable at a later data.

Note that both the converter and this change to GYOTO will be made so that we can use an arbitrary number of objects.

10.1.3 Additions to the Source Code of LORENE and GYOTO

Most of the changes that have been made to GYOTO have been done by Frederic Vincent at my request, meaning that I have not changed much of the GYOTO source code myself. There are three minor changes I have done to the source code of LORENE and GYOTO.

The first change is to LORENE, and is crucial for the conversion pipeline to work! The conversion code ask LORENE to print all the collocation points to stream, where it can pick it up. The way LORENE prints out the collocation points have a set precision which is way too small, and will lead to errors. In the LORENE file `tbl.C` we changed the `precision` call in the function `ostream& operator<<`(for me at line 394) to be `o.precision(15)`. This prints out the collocation points with a much greater precision.

The second change is to GYOTO, and is needed to get the photon norm drift, which we will use for testing the conversion. In the file `WorldlineIntegState.C` the line `cout << Norm << norm_ << " " << coord[1]`
`<< " " << coord[2] << " " << coord[3] << endl;` is added to the function `void Worldline::Integ`
(for me at line 104). This will print the photon norm and the (r, θ, ϕ) coordinates to screen when using GYOTO. This will happen for every photon at every integration step, so it is wise to pipe this output into a text file.

The last change is minor, and is only to get the Page-Thorne disk (see sec. 5.4.4) to work. The calculation of the disk is dependent on the spin of the black hole. For this reason, GYOTO won't let us use this object with a numerical metric, since the spin is not defined. But since we want to use the disk anyways, we need to change the source code of GYOTO. In the file `PageThorneDisk.C` and the function `void PageThorneDisk::updateSpin` (for me at line 90) we need to comment out all the switch statement which can check the type of metric and can trigger an error. We then force the spin to be zero by adding the line `aa_=0.0`. In the function below,

`void PageThorneDisk::metric` we will also need to remove the if-statement checking the metric type and causing an error. This lets us use the Page-Thorne disk.

10.2 Running GYOTO

From the converter we now have a *.d* file with all the LORENE objects needed to describe our spacetime. This means that we can ask GYOTO to use this file, possibly together with some other astrophysical object, to ray trace. GYOTO makes this quite easy. While it possible to make scripts for GYOTO using Python or Yorick, for us it suffices to use GYOTO in the terminal. To run GYOTO from the terminal we use the command

```
gyoto [xml file] [output file]
```

This takes an Extensible Markup Language (XML) file specifying the scene (this will be discussed below), and an output file, normally in a *.fits* format. This output file can then be read by, for example, the program *SAOImageD9*[33].

10.3 The Anatomy of the XML File

All the important settings for how the ray tracing is done can be found in the XML file. Here we will define the type of metric, integrator, camera and astrophysical objects will be used. The two XML files used for my ray tracings can be found in appendix E.1. We will now go through all the XML tags and setting used in the file that is used for most of the results, the XML file for a fixed star.

First of all, everything inside the script can be found inside the tags

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Scenery>
   (...)
</Scenery>
```

This tells GYOTO that this is XML, and that everything inside these tags are the scenery used for the ray tracing.

The next step is to set up the metric used for the ray tracing

```
<Metric kind = "NumericalMetricLorene">
    <MapAf/>
    <Horizon>0.51</Horizon>
    <File>/home/dulte/Documents/Skole/Master/Gyoto_files/Metrics/</File>
    <AxisymCirc/>
</Metric>
```

The property *kind* tells GYOTO what kind of metric we want to use. In this case we have a numerical metric from LORENE[1]. In this case we will have to give the file where the numerical metric can be found, the horizon where GYOTO will stop integrating and whether or not to force symmetries in the integration. Note that there is a strange

[1] In the full script there is also an example showing how to use an analytical Kerr metric.

convention with the file names of the numerical metric: The metric file(s) must be inside a folder with the name *Metrics* which is located as given in the XML file. The files inside this folder must have the name *metric00001.d*. If the number at the end of the file name increases (e.g. *metric00002.d*) GYOTO will assume that this is the next time step.

The next line should be

```
<Integ31/>
```

This indicates to GYOTO that we are using integration of the 3+1 quantities, instead of using the 3+1 quantities to reconstruct the normal 4D objects used in GR.

We then need to set up the camera/screen

```
<Screen>
    <Position>1000. 250. 1.483 0.</Position>
    <Time unit="geometrical_time">1000.</Time>
    <FieldOfView> 0.05 </FieldOfView>
    <Resolution>30</Resolution>
  </Screen>
```

The position here is given in the coordinate system (t, r, θ, ϕ). The time should be the same as t in position. The field of view describes angle of the image. The resolution is the number of pixels for the height/width. So a resolution of 30 will give a $30x30$ image, meaning that 900 photons will have been ray traced.

The next part depends on the objects we want to look at in the ray tracing. Without any objects the image will be black, since there are nothing for the photons to interact with. To see an effect of the spacetime, and to be able to compare the converted result with analytical results we need to have an object. We choose to use a *fixed star*. This is simply a simple star placed in the center of the spacetime. Note that this will only affect the intensity of the photons (see sec. 5.4.4), and not the geodesic. So even though there is a star there, the spacetime will be Minkowski if nothing is given by the numerical metric. The XML for this is

```
<Quantities>Intensity</Quantities>

<Astrobj kind = "FixedStar">
    <Radius> 3.972 </Radius>
    <Position> 0 0 0 </Position>
    <Spectrum kind="PowerLaw">
      <Exponent> 0 </Exponent>
      <Constant> 1. </Constant>
    </Spectrum>
    <OpticallyThin/>
    <RMax>0.</RMax>
</Astrobj>
```

The tag at the top tell GYOTO that we want to look at the intensity. We see that the *kind* property of the *Astrobj* tag tells us that we are looking at a fixed star. The important tags inside here is radius and the position, which are self-explanatory, and the *RMax* tag. This tag tells GYOTO when to start to check whether or not the photon is inside the star. When the photon is inside this radius, the adaptive integration steps

need to be shorter, so that the photon does not overshoot the start of the actual star. This means that more time is needed for the integration. We therefore want to tell GYOTO when to check for the star. Notice that RMax= 0 here. This is the same as removing the star, but without having to comment the whole section of the XML file out. Running GYOTO without the star is useful when looking at the errors (see below). If you want to ray trace with the star, RMax= 20 is a good choice for this radius. Since we have $r = 3.972$, we will also look at RMax= 5.

The last two lines are technical information we want to give to GYOTO

```
<MinimumTime> -1e4 </MinimumTime>
 <NThreads> 1 </NThreads>
```

The minimum time is the smallest value the adaptive integration step can take. The *NThreads* is the number of threads GYOTO is allowed to use. For numerical metric, GYOTO can sadly not use more than one thread, so this line is technically redundant, but is kept in so it can be used if someone wants to use this script for an analytical ray tracing.

We will also use another XML file to create our last image of a Page-Thorne disk. This XML file can be found in the appendix E.2.

10.4 Looking at Errors in the Raytracing

We want to be able to compare the results of our ray traced numerical metrics and some proven results In our case a numerical metric from LORENE. As we have seen, it is possible to make analytical metrics with GYOTO. The reason that we are using a numerical metric from LORENE is to have the best case numerical metric to compare with. If one looks compare the analytical metric from GYOTO with the numerical one from LORENE, they will give very similar results. This is not added to this thesis to save space.

We have made two methods for comparing the results to see how good the conversion is. The first method we will use is to ray trace using an object in the path of the photons, and look at the resulting images, i.e. the $.fits$ files. The object is a *fixed star* and is an object included with GYOTO. If all of our images are of the same metric, e.g. Schwarzschild, with the same mass, then we know – from the no hair theorem and from looking at the expression for the metric – the spacetimes should be identical, and thus the images should be the same. The only difference should be from numerical error coming from the conversion. Including using SAOImageD9 we can also use *Astropy*[5] to read the data into Python. This lets us compare the the pixel values of the images to look at the differences.

This method is a bit crude and does not give any help in finding where possible errors come from. The next method is based on the fact that the norm of a photon $\mu^2 = g_{\mu\nu}p^\mu p^\nu = 0$ should be constant along a null geodesic. Since GYOTO uses numerical integration, this will not hold for a full integration. We can therefore plot the norm and see how it differs from 0 along the geodesic. Numerical errors coming from the conversion will impact the norm of the photons, so by looking at the norm of the

converted metric compared with norm of the analytic metric and the metric simulated in LORENE, we can see how much our results differ and where on the geodesics errors occur. This lets us look at the effect of both the interpolation if the Einstein Toolkit data and the domain size/resolution we used in LORENE.

These are the two methods we will be using to evaluate our ray traced metrics. We will be using a LORENE metric simulated with the LORENE file `kerr_QI.C`. This is written by the creators, and is included with LORENE. We will assume that the metric we gets from this LORENE simulation is the best we can get from LORENE, and is therefore the standard we want to compare our results with. This file can be found with the other examples given by LORENE. Together with this C file a parameter file is given. Here it is important to set the mass to 1 and the spin to 0. This will give us a good comparison for the data simulated in Einstein Toolkit. We will use comparisons of the fixed star image and the photon norm for this metrics and our converted metric, either our analytical test metrics or the metric simulated in Einstein Toolkit, to evaluate how good our results are. If we get results equal or similar to the LORENE metric we can assume that we have succeeded in our goal of using a Einstein Toolkit metric in GYOTO.

At the end we will make an image of a Page Thorn disk. This is mostly to have a final showcase result.

10.5 Using *Dumb* Parallelization

GYOTO is normally fully integrated with MPI, and can parallelize the ray tracing with a single parameter. Sadly this is not implemented for numerical metrics. This means that the only way we can use parallelization is using a *dumb* parallelization. When running GYOTO in the command line we can add the parameters `--imin` and `--imax` to choose the minimum and maximum column of pixels that will be ray traced, and `--jmin` and `--jmax` for the the minimum and maximum row. This means that one can separate the whole screen into many subscreens. This way we can run one subscreen in one terminal and another subscreen in another terminal. Since the pixels/photons are completely independent, all the resulting images can be combined into an image of the whole screen. In this way we can run the ray tracing on multiple cores without using MPI.

Part III

Results

Chapter 11

Note on Units

All the important units in this section is normalized with respect to solar masses. The reason for this is that the unit system used by Einstein Toolkit measures both mass and length in solar masses. This means that no units are given in the plots nor the text, and where either mass or length is seen, these should be taken as being measured in solar masses. For the purpose of this thesis, the actual lengths are not important, so no conversion of units are done.

This is made a bit more confusing by the contour plotting functions of LORENE, where *km* is used. The conversion between the unitless units and km is just a factor 10. So where we have contour plots spanning from -200 to 200km, this will actually show the simulated data from -20 to 20.

Chapter 12

Reading from Einstein Toolkit

The first part of the conversion is the reading of the Einstein Toolkit data. As we have seen, most of the process is done by the Python library PostCactus, with a couple of parameters given by the user. We will here look at the results of the reading and interpolation of data. First we will look at how the different types of geometries affect the data, finding the best type of geometry. Having the "best" type of geometry we will look at how the grid size will affect the quality of the data. This will be done using a single black hole, since this lets us compare the read data with analytical expressions. We will then look at how the data for a binary black hole system is read.

12.1 Single Black Hole

We start by looking at a single black hole simulated in Einstein Toolkit using analytical initial conditions. These initial conditions are given by a isotropic Schwarzschild metric(2.4), meaning we can compare the results with analytical expressions. For this analytical metric we have only two non-zero quantities, the lapse function α and the diagonal components of the spatial metric $\gamma_{xx} = \gamma_{yy} = \gamma_{zz}$. So α and γ_{xx} will be the quantities used to compare the results

12.1.1 Effect of Different Types of Geometries

First we will look the different types of geometries used to read data from Einstein Toolkit, as well as the different interpolation methods (spline or linear interpolation). As mentioned in the method section the type of geometries we have are a single uniform grid, multiple uniform grids with different resolutions and sizes, and a none-geometry where PostCacus reads the geometry as given by Einstein Toolkit. The first two geometries will have the possibility for a linear or a spline interpolation, while the third will only allow for a linear interpolation.

In table 12.1 we see the results of the different geometries on the lapse function created by Einstein Toolkit. The parameter file for this simulation can be found in the appendix D.1. dx= 2 was for this test – see below for the discussion of effect of different

Type	Geometry Details	Mean Error	Max Error	Run Time[Sec]
Multigrid Spline	Radii: [10, 20, 100, 300]; Resolutions: [100, 100, 100, 100]; Limits: [5,10, 50,250]	9.04×10^{-5}	8.87×10^{-3}	37.59
Multigrid Linear	Radii: [10, 20, 100, 300]; Resolutions: [100, 100, 100, 100]; Limits: [5,10, 50,250]	1.01×10^{-4}	4.41×10^{-3}	36.71
Single Grid Spline	Radius: 300; Resolution: 300	0.023	0.54	25.68
Single Grid Linear	Radius: 300; Resolution: 300	0.014	0.35	24.06
None	-	6.13×10^{-6}	2.85×10^{-5}	1.01

Table 12.1: Here we can see the results of reading of the lapse function α from Einstein Toolkit for different geometries. The parameter file can be found in the appendix D.1. For this dx= 2 was used. The errors are the absolute error between the simulated and analytical expression over a radius of 20. We can see that all the geometries gives quite good results, with the none-geometry being the most accurate and fastest, with the single grid being the worst when talking about the error.

values of dx on the error. We will look at bit on the error using this table and use this to judge the usage of each type of geometry. Since this might seem a bit naive, we only compare one example of each geometry, but as we will see a bit later, there are some hidden obstacle which will make the choice of geometry simple, independently of the grid sizes and resolutions used.

Note that all the times in the table 12.1 are meant to be a comparison between the methods and will differ between different computers and different runs (and altitude, humidity, moon phases, star alignments and all the other things that make computers run at different running times...). All of these runs were made consecutively on the *beehive5* computer on *Institute for Theoretical Astrophysics* at UiO[1].

The multigrid geometry uses four different uniform grids with radii 10, 20, 100 and 300, all of them with resolution 100. The reason for these parameters are trial and error, with the most important radius being 300, which is the furthest radius simulated by Einstein Toolkit using the mentioned parameter file. The limits are the limits at which the converter switches from a finer grid to a coarser. It is important that this limit is smaller than the radius of a grid. The errors are the absolute error between the simulated and analytical expression over a radius of 20. We see that for the spline interpolation the results are on order 10^{-3} to 10^{-5}, which is good, but as we will see later not nearly as good as we need to get good ray tracing results. The linear

[1] https://www.mn.uio.no/astro/english/services/it/help/basic-services/compute-resources.html

interpolation gives errors which are a bit higher for the mean error and a bit lower for the max error.

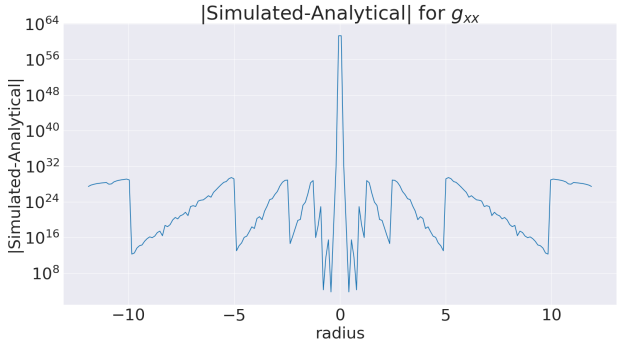
The single grid uses, as is self explanatory, only a single grid. Here the radius of the grid is the same as the one in the muligrid case. The resolution is one which is fast and accurate. Higher resolution will lead to a small improvement in the error, but will lead to much longer run time. We see here a small improvement in run time, but it is still comparable to the multgrid case, which makes sense this the multigrid have to interpolate four grids that are smaller. The results are sadly much worse, with the error being of order 100 larger. This is the case for both the spline and the linear interpolation, with the linear interpolation being a bit better.

Looking at the none-geometry we see much better result! Not only is the errors $10-100$ times better, but the runtime is only a fraction of both the single grid and the multigrid. This is most like since the none-geometry reads the data on a geometry as they are given by Einstein Toolkit, which the other geometries first linearly interpolate the data onto the uniform geometries. When the user now asks for a value at a point, the none-geometry will do a simple linear interpolation, while the other geometries can do either a linear or spline interpolation. To summarize this, going from Einstein Toolkit data to output to the user only require one interpolation using the none-geometry and two interpolations for the other geometries.

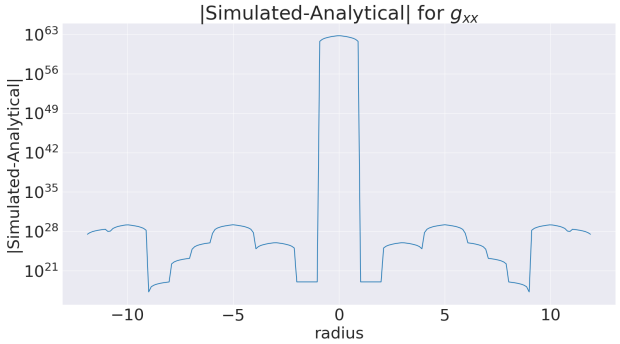
The above results for the interpolation of the lapse function indicates that the none-geometry is the way to go. These results could give room for this to be discussed and some use for the other geometries might have been found. We will now look at the results for g_{xx}, which will make all other geometries other than the none-geometry obsolete.

In table 12.2 we can see the results for the reading of g_{xx}. Now the results are completely different. We now can see that for both single grid and multigrid, independently of the type of interpolation, give absolute errors of around 10^{60}. One can argue that this might happen close to the center, where g_{xx} rapidly diverges, meaning that an absolute error of this magnitude might be reasonable. Remember that if a diverging error exists only close to the center, this will drive up the mean and max error, giving a false impression that we have an erroneous result. As we will see shortly, this is indeed the case for the none-geometry, but sadly not for the single grid and multigrid geometries.

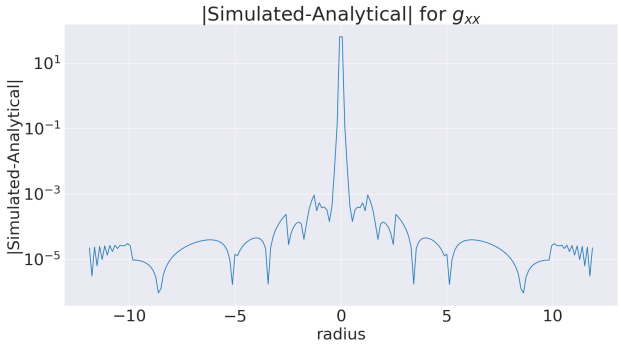
In fig. 12.1 we see the absolute error for g_{xx} plotted radially along $y = z = 0$(the x-axis). The center of the spacetime, and in this case the black hole, is at radius$= 0$. The radius indicates the distance from the spacetime/black hole center, with positive being to the right and negative being to the left. While having a negative radius might be strange, we will use radius or r as the measure of distance along the x-axis from the center of the spacetime (which coincides with the center of the black hole for the single black hole) for the error plots. We can see that for the single grid and multgrid, using a linear interpolation, we have an error at 10^{60} for the whole plot, while the none-geometry have an error at $< 10^{-3}$ everywhere except from the center. This means that we can point to this as the reason for the high mean and max error. The increase in the center happens inside the event horizon $r_{horizon} = 0.5$, meaning that they won't impact the ray tracing.



((a)) Absolute error using four grids in a multigrid. We can see that the error is of order 10^{60}. The sudden decreases we can see is due to change in which grid is used.



((b)) Absolute error using a single grid with radius and resolution 300. We can see that the error is of order 10^{60}. The sudden decreases we can see is due to the grids used for AMR in Einstein Toolkit.



((c)) Absolute error using a none-geometry. We see here that the error is at a manageable level, except from the center. Due to this increase happening inside the event horizon, it will not impact the results. The sudden decreases we can see is due to the grids used for AMR in Einstein Toolkit.

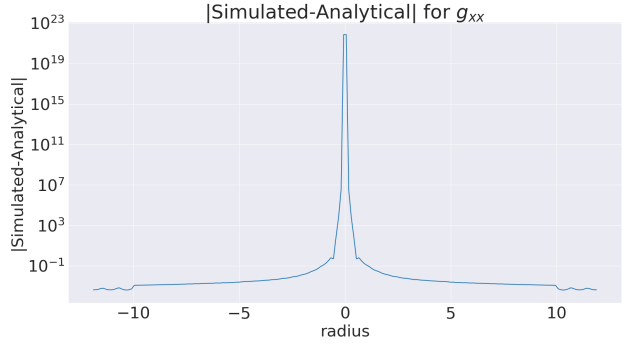
Figure 12.1: The absolute error for g_{xx} for a single black hole with isotropic Schwarzschild metric, plotted against the distance/radius away from the black hole (at radius $r = 0$). We see that using the single grid and multigrid geometries, with a linear interpolation, gives us errors at 10^{60} (fig. 12.1(b) and 12.1(a)), while the none-geomtery gives us errors at levels we can accept (fig. 12.1(c)).

Type	Geometry Details	Mean Error	Max Error	Run Time[Sec]
Multigrid Spline	Radii: [10, 20, 100, 300]; Resolutions: [100, 100, 100, 100]; Limits: [5,10, 50,250]	2.40×10^{59}	1.87×10^{61}	40.12
Multigrid Linear	Radii: [10, 20, 100, 300]; Resolutions: [100, 100, 100, 100]; Limits: [5,10, 50,250]	2.22×10^{59}	2.20×10^{61}	36.16
Single Grid Spline	Radius: 300; Resolution: 300	3.57×10^{61}	6.20×10^{62}	25.50
Single Grid Linear	Radius: 300; Resolution: 300	2.61×10^{61}	5.87×10^{62}	23.47
None	-	0.64	63.47	0.77

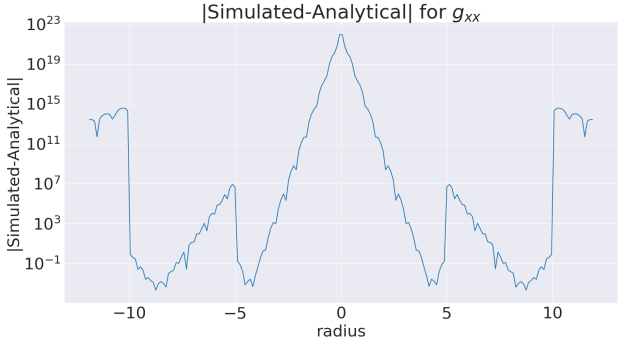
Table 12.2: Here we can see the results of reading of g_{xx} from Einstein Toolkit for different geometries. The parameter file can be found in the appendix D.1. For this dx= 2 was used. The errors are the absolute error between the simulated and analytical expression over a radius of 12. We can see that we now get outlandish results, with the errors at around 10^{60}. This indicates that something is wrong. The none-geometry is still the best, but still has more error than wanted.

The error for the single and multigrids means that these types of geometries are unsuitable for actual use. What the reason for this error in reading and interpolating the data is unknown, especially since the lapse function behaves well. We can also ask whether this behaviour is always present. The answer seems to be no. In fig. 12.2 we have plotted the same as in fig. 12.1, but an other parameter file has been used (this can be found in the appendix D.3 or accompanying [36]). This parameter file generates the same single black hole, but uses a two puncture method instead of analytical initial conditions. We can see here that when we use a multigrid with a linear interpolation (fig. 12.2(a)) we get results more akin to the one we got when using the none-geometry. The metric is still not very usable, only reaching 10^{-1}. We can also see that the error increases quickly outside of the event horizon, meaning it would impact our results. In fig. 12.2(b) we have the same plot, but with a spline interpolation. We still get areas with a small error, but we still have spikes at around 10^{23}, meaning that the data is useless. The sudden drop in error corresponds well with the limits of the grids used, pointing to the boundaries of the grids having a large impact.

The reasons for the difference when using these two parameter files is unknown. The two main differences between the parameter files are the smaller amount of mesh refinements and the use of a two puncture method in the latter parameter file. The two puncture method method should not make the reading better, since they both give values at the same points, and the analytical method also give a more precise value for the metric. The smaller amount of mesh refinements might be to blame, but we can



((a)) Absolute error using four grids in a multigrid with a linear interpolation. We see that contrary to fig. 12.1 we now have a low error. We can also see that the error starts to increase outside the event horizon.



((b)) Absolute error using four grids in a multigrid with a spline interpolation. We see that compared to fig. 12.1 we now have a lower error, but still we have extremely high spikes.

Figure 12.2: The absolute error for g_{xx} for a single black hole with isotropic Schwarzschild metric created using a two puncture method. We have two multigrids, one with a linear interpolation and one with a spline. We see that the linear interpolation gives much lower error than fig. 12.1. For the spline we have much lower error, but we still see spikes reaching 10^{23}.

question whether throwing out the mesh refinement only to get a geometry inferior to the none-geometry to work is a good idea.

All of this means that we can use the multigrid system in some cases, but in all cases the none-geometry is superior, and the easiest to use. In fig. 12.3 we see a 2D intensity plot of the error of g_{xx} using the none-geometry. We can clearly see squares in the plot, where these are a wave pattern corresponding to a drop in error. These squares are the grids used by Einstein Toolkit when doing the adaptive mesh refinement. We can also see these drops as sudden decreases in error in fig. 12.1(c).

Having looked at the different geometries we can conclude that the none-geometry is the best geometry to use. Not only does it give the best results, but also the fastest read time (see tab. 12.1 and 12.2). This geometry will have increased run time when a lot of points are use, since it does the interpolation each time we need a point. The other geometries will do the interpolation beforehand, and can even store the interpolation object, making it much faster to read at a later time. The other geometries might work in some cases, such as in fig. 12.2(a), but they are worse and less consistent than the none-geometry, meaning that even though they might be much faster for larger and repeated runs, we have to conclude that the none-geometry is the best and safest to use.

For all the results below (except for when we try reading in a binary black hole 12.2) we will only be using the none-geometry!

The reason why the other geometries are so well developed and discussed even though they are inferior to the none-geometry, is that the latter was throughout and

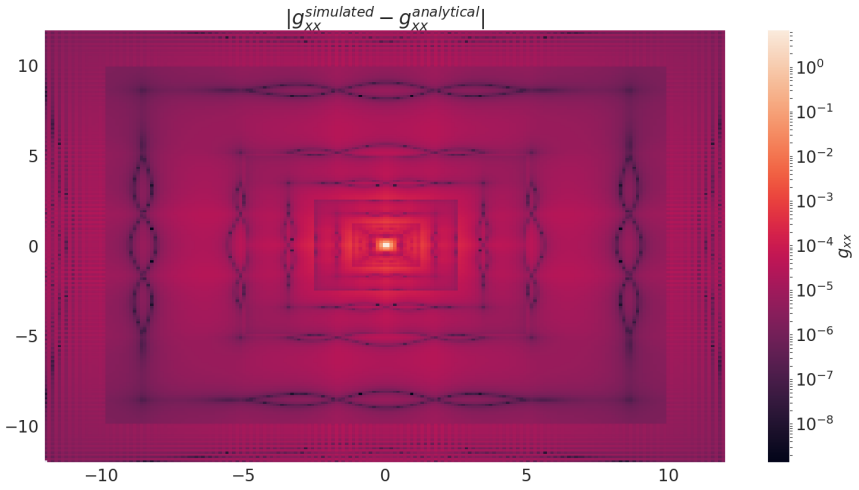


Figure 12.3: An intensity plot of the error of g_{xx} for a single black hole using a none-geometry. We can here clearly see the square grids used by Einstein Toolkit for the mesh refinement. We see some wave patterns spanning the borders of the grids. The reasons for these are unknown, but surprisingly these patterns has a lower error than the surroundings.

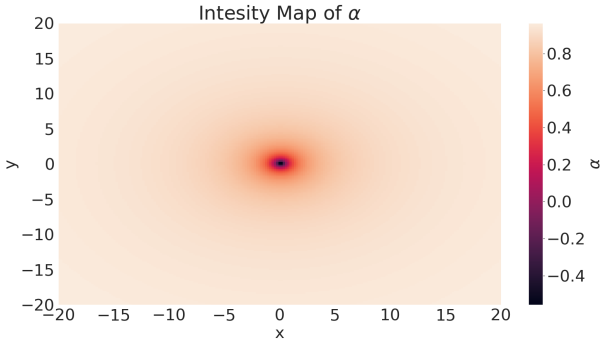
implemented much later in the thesis, meaning that all of the conversion code was developed with the single and multigrid geometries. All the test were done with the two puncture single hole (fig. 12.2(a)), meaning that only later did the major problems with these geometries become clear.

We can now look at how the lapse function and the diagonal metric coefficients will look like using the none-geometry. These can be found in fig. 12.4. Here we can see that the lapse function α starts close to zero at the center and gradually increases to one at infinity. g_{xx} is also one at infinity and will stay at this value until it get close to the center, when it will diverge quickly, leaving almost no contours to be seen. These results will be used to compare with when we look at the results after the conversion to LORENE.

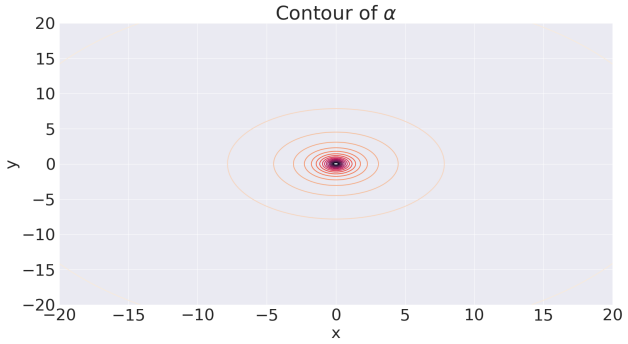
12.1.2 Effect of Different Grid Sizes

Having looked at the different geometries we can use to read the data from Einstein Toolkit, we can now turn to the simulation done in Einstein Toolkit. When running a simulation in Einstein Toolkit we can specify the size of the simulation as well as the grid. Above we mentioned that we used a grid resolution of $dx = dy = dz = 2$ (from here on only called dx) for all the test. We will now try to see how the resolution impacts the simulation time and the read data.

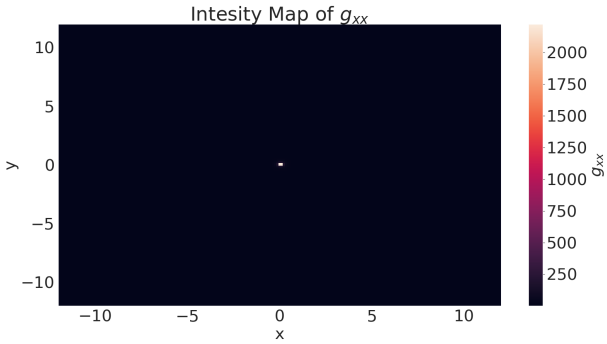
We will be using a simulation size of 300 for all the simulation and use of seven



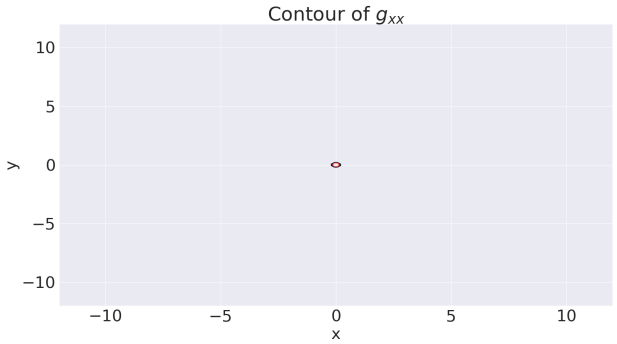
((a)) An intensity map of α.



((b)) A contour plot of α



((c)) An intensity map of g_{xx}



((d)) A contour plot of g_{xx}

Figure 12.4: The results of the reading and interpolation of the Einstein Toolkit data for a single black hole using a none-geometry. These will be the results we use as a comparison to the data after conversion to LORENE

dx	RAM [GByte]	Could Run	Mean Error	Max Error
1	388.21	No	-	-
1.5	131.78	No	-	-
1.875	77.96	Yes	6.97×10^{-5}	8.03×10^{-4}
2	65.71	Yes	7.45×10^{-5}	9.29×10^{-4}
2.5	37.05	Yes	1.12×10^{-4}	1.26×10^{-3}
3	23.91	Yes	1.50×10^{-4}	2.07×10^{-3}
4	12.73	Yes	2.15×10^{-4}	3.67×10^{-3}

Table 12.3: Overview of the different grid resolution dx and memory requirements used for each run. All of the simulation were run on the *beehive*5 node at ITA, so I've indicated whether this node was able to run the simulation or not. We can also see the mean and max absolute error for each dx. Here we have only taken the error outside of the event horizon $r_{horizon} = 0.5$. The errors are calculated using g_{xx} and a none-geometry.

different values for dx. Out of these the two smallest needed more memory than the computer we ran the simulation on had. This is to show that the required memory grows exponential (or worse) with lower dx, meaning that we can not strive for smaller and smaller dx's.

In table 12.3 we see the results for the different resolutions dx. We can also see the mean and max absolute error over the range plotted. We will look at these shortly.

In fig. 12.5 we see the different absolute errors for different dx. This is the same single black hole simulation as discussed above (app. D.1) with a none-geometry. We can see that when dx increases the plots becomes more erratic. This is most likely due to more points having to be interpolated from points further apart, leading to numerical error.

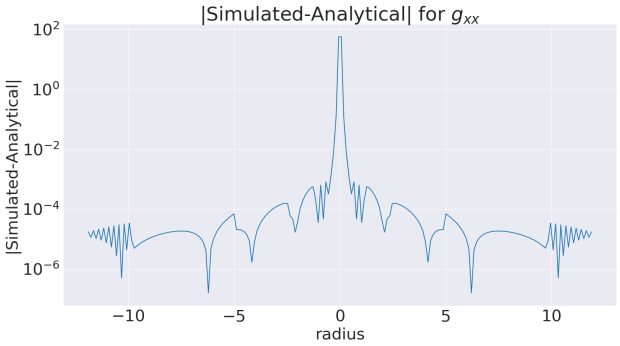
We don't see much from these plots when it comes judging the improvement in the absolute error. For this we have to go back to table 12.3, where we can see the mean and maximum absolute errors. We saw previously that the error increase drastically inside of the event horizon. These errors are therefore calculated using $r > r_{horizon} = 0.5$.

We can see both the errors increase as dx increases. This increase is in fact linear, with a slope of 6.98×10^{-5} for the mean error and 1.35×10^{-3} for the maximum error. This means that we don't get much improvement for decreasing dx

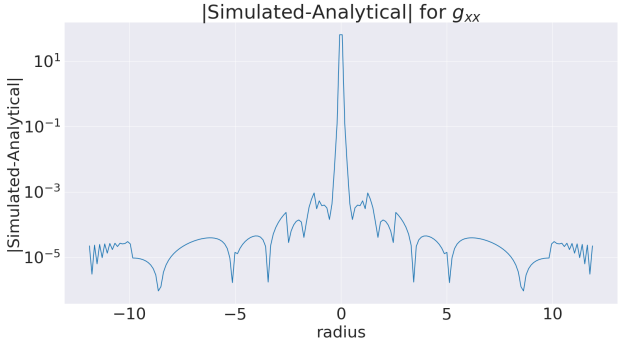
To conclude, while the system requirement increases exponential with a decrease in dx, the error only decreases linearly. This means that the choice of dx mostly comes down to the computer one runs the simulation on. We will see later that we get an error in the ray tracing more or less comparable to the numerical error from the reading of the data, but as we have seen now we can not decrease this error much more by decreasing dx. We will look more at this later.

For all the conversion and ray tracing below we will continue to use $dx = 2$.

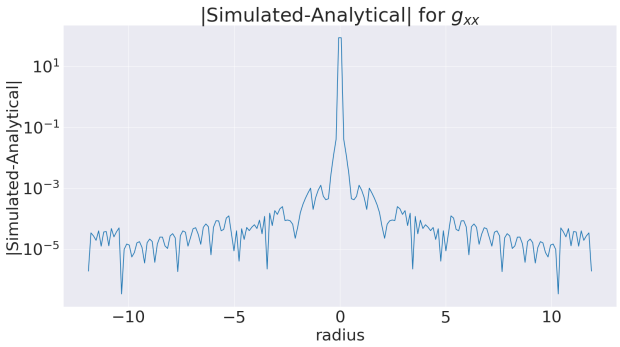
We have thus found that the optimal geometry for reading a single black hole is none-geometry, and that there is no reason to decrease dx of the Einstein Toolkit



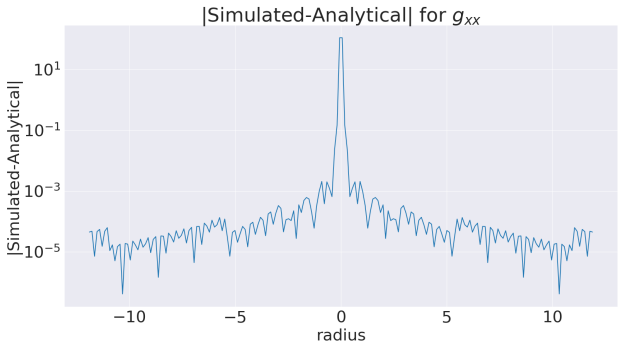
((a)) $dx = 1.875$



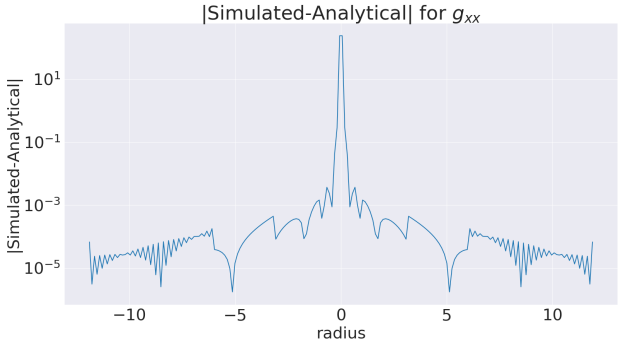
((b)) $dx = 2$



((c)) $dx = 2.5$

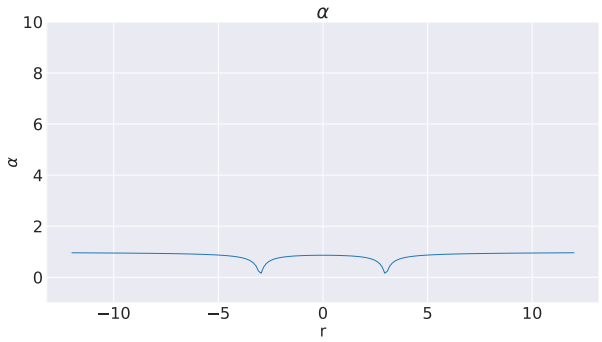


((d)) $dx = 3$

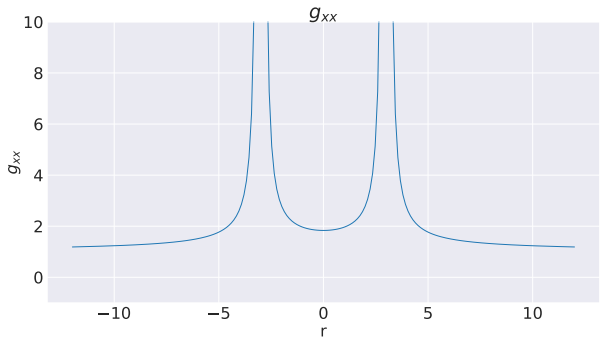


((e)) $dx = 4$

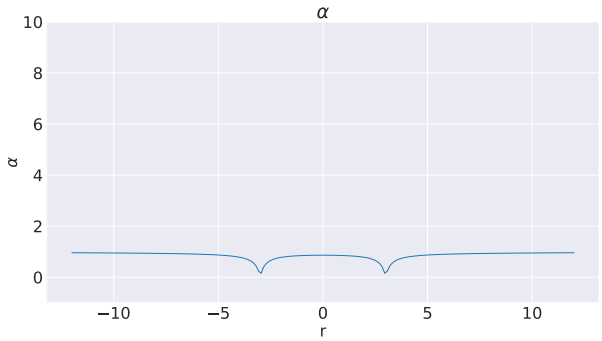
Figure 12.5: The absolute error for g_{xx} for a single black hole using a none-geometry. Different grid resolutions dx are used to show how the error decrease with dx



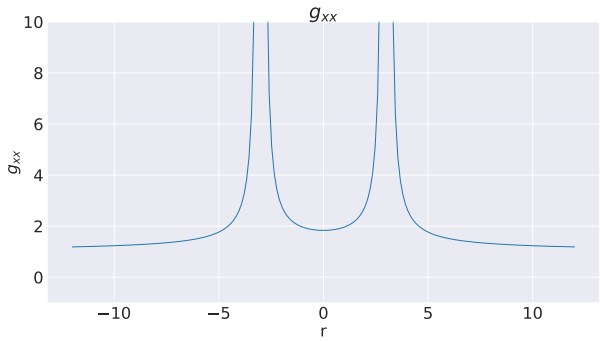
((a)) α with multi geometry.



((b)) g_{xx} with multi geometry.



((c)) α with none-geometry.



((d)) g_{xx} with none-geometry.

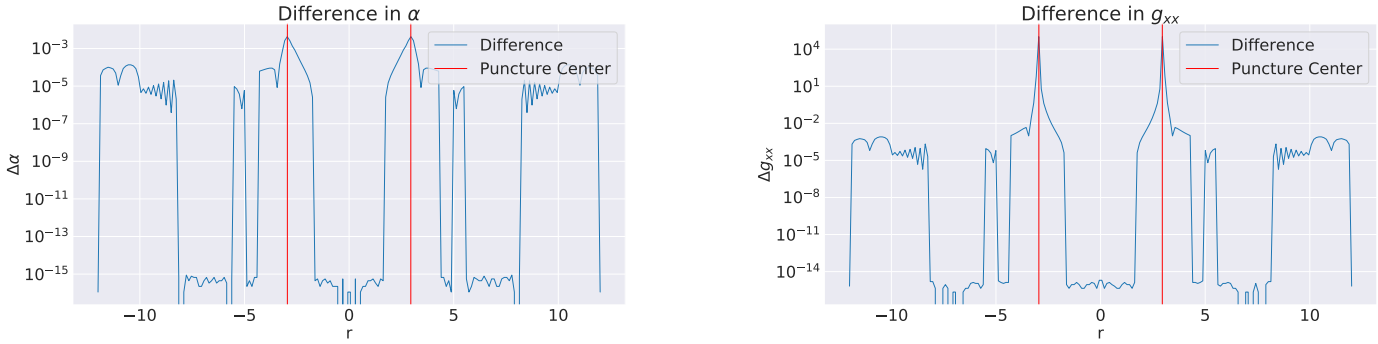
Figure 12.6: The results of the reading and interpolation of the Einstein Toolkit data for a binary black hole system. Here the α and g_{xx} are functions of r along the x-axis. We see that they are more or less equal, meaning that the multigrid geometry worked for this simulation.

simulation more than $dx = 1.875$. We will now move on to a binary black hole system.

12.2 Binary Black Holes

We can now look at a binary black hole system, and see how this fares when read into the converter. This binary black holes is, as discussed in sec. 7, the parameter file found in appendix D.2.

We concluded above that the none-geometry was by far the best geometry to use when reading the data, followed by the multigrid and lastly the single grid geometry. Since this simulation used a two puncture method for initial data, which we saw could work with a multigrid geometry, we will try to read the data using both none-geometry and a multigrid geometry. The multigrid geometry is the same as in table 12.1 and 12.2.



((a)) α with multi geometry.

((b)) g_{xx} with multi geometry.

Figure 12.7: The difference between the none-geometry and the multigrid geometry for a simulated binary black hole system simulated in Einstein Toolkit. We see that difference are quite small. There are sudden increases in the difference. They seem to be associated with the grid sizes in the multigrid geometry.

In fig. 12.6 α and g_{xx} are plotted as functions of r along the x-axis. In appendix C the reader can find the intensity plots and contour plots for these results (fig. C.11 and C.6). We can see that the plots are more or less identical. From the reading of the single black hole we expected the multigrid geometry to give results with values close to 10^{60}, while we don't see this here at all.

Since we don't have any analytical expressions to compare these results with, we instead have to compare them against eachother. In fig. 12.7 we have plotted the difference between the figures we looked at above. We see that there are very little difference between the two types of geometries. The center of the black holes/location of the punctures are marked with a red line. We see that the difference occurs near where we have limits for the different grids used in the multigrid. We have the highest amount of error close to the black holes. This is most likely due to the AMR having a very fine refinement close to the punctures. This will most likely lead the none-geometry to be much more precise than multigrid.

Contrary to the single black hole case, we found that the multigrid geometry also works well with the binary case. Why this is the case is a bit of the mystery. Just like in the single black hole simulated with two puncture, we might have a result which looks good, but still is inferior to the none-geometry. We have no concrete method of checking if the none-geometry is the best, so we will use what we learned from the single black hole case: The multigrid geometry might work well, but we have seeded enough doubt in it, and seeing that the none-geometry is superior in every other case, we choose to use the none-geometry going forward with the binary black hole system as well.

Chapter 13

Conversion to LORENE

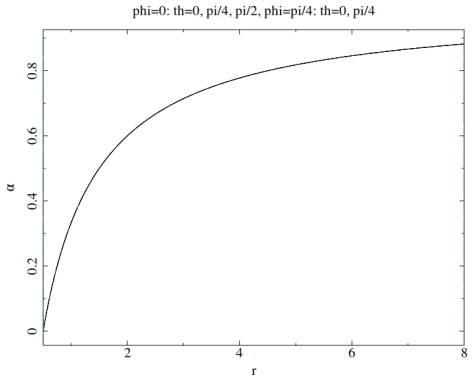
We have now looked at the reading of the Einstein Toolkit data, as well as comparison of the data of a single black hole metric with the isotropic Schwarzschild metric. We can now move on to the next step in the process: Transforming the data to a spectral representation with LORENE. We will first try out the conversion with the test metrics we made (sec. 8.7). We will so look at the metric for the single black hole. This lets us, once again, compare the results with the analytical expression for the isotropic Schwarzschild metric(2.4). We will lastly look at the binary black hole metric, where we will look at both how the resolution affects the results and what happens if we don't include the splitting function (sec. 6.1).

13.1 Conversion of Test Cases

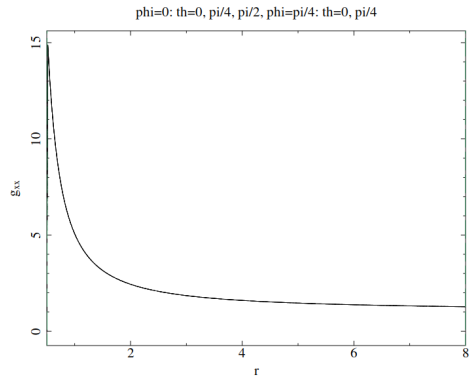
The test cases (sec. 8.7) let us look at how the spectral transformation pipeline works without having to worry about numerical errors from the reading of the data. If there is something wrong with this pipeline, we should be able to iron this out before throwing the real data on it.

The results for the Minkowski metric are just printouts of the constant coefficients $\alpha = g_{xx} = 1$. Plots of the results can be found in fig. C.1, but due to the automatic limits in the class used by LORENE to plot, the maximum and minimum of the plots hide the results, so the plots are moved to the appendix C, but we have here $\alpha = 1$ and $g_{xx} = 1$. Three domains were used in this conversion, with the limits $[0.5, 8, \infty]$ and the resolution $n_r = 25$, $n_\theta = 7$ and $n_\phi = 4$.

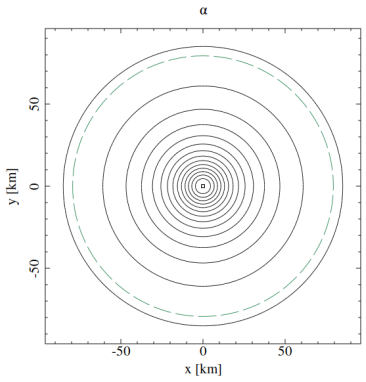
This result does not show us much, but if we instead look at the converted Schwarzschild metric in fig. 13.1 we see that we have successfully transformed some analytical metric into a spectral representation. The transformation parameters are the same as the Minkowski metric above. Since LORENE is used to plot the contour plots instead of matplotlib, it is not trivial to compare 13.1(c) and 13.1(d) with 12.4(b) and 12.4(d), but if we look at 13.1(a) and 13.1(b) we see plots which seems to have the same form as we would expect from the expression for an isotropic Schwarzschild metric (2.4). Note that 13.1(c) and 13.1(d) have multiple plots in them. These plots are values of



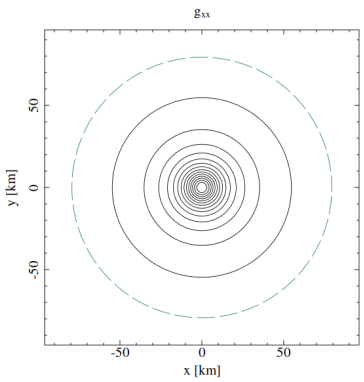
((a)) Plot of the lapse function α after the conversion.



((b)) Plot of the spatial metric coefficient g_{xx} after the conversion.

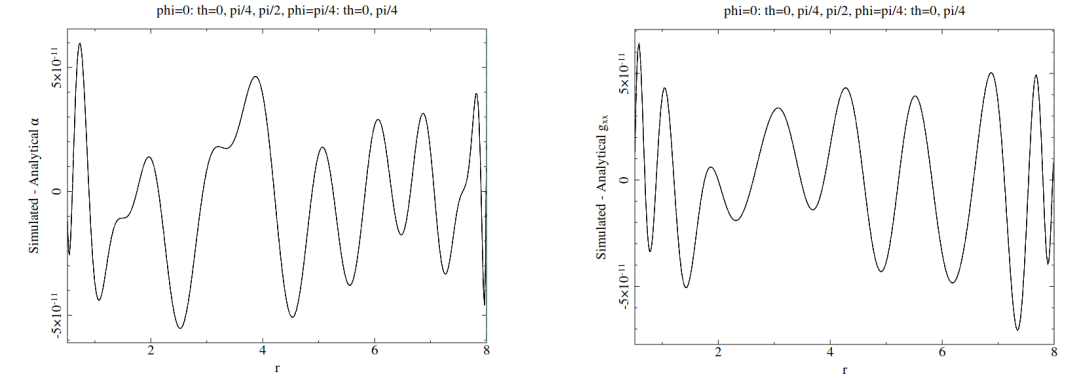


((c)) Contour plot of the lapse function α after the conversion.



((d)) Contour plot of the spatial metric coefficient g_{xx} after the conversion.

Figure 13.1: Here we can see plots of the lapse function α and the spatial metric coefficient g_{xx} for an analytical isotropic Schwarzschild metric after conversion. Three domains where used in this conversion, with the limits $[0.5, 8, \infty]$ and the resolution $n_r = 25$, $n_\theta = 7$ and $n_\phi = 4$.



((a)) Plot of difference between the converted α and the analytical expression.

((b)) Plot of difference between the converted g_{xx} and the analytical expression.

Figure 13.2: Here we can see plots of the difference between the lapse function α and the spatial metric coefficient g_{xx} for an analytical isotropic Schwarzschild metric after conversion and the analytical expression found in (2.4). We can see that the difference is around 10^{-11}. Three domains where used in this conversion, with the limits $[0.5, 8, \infty]$ and the resolution $n_r = 25$, $n_\theta = 7$ and $n_\phi = 4$.

the functions radially for different values of θ and ϕ. Since the Schwarzschild metric is spherical symmetric, all of these plots are the same. When we move on to the binary system, we will see cases where this is not true.

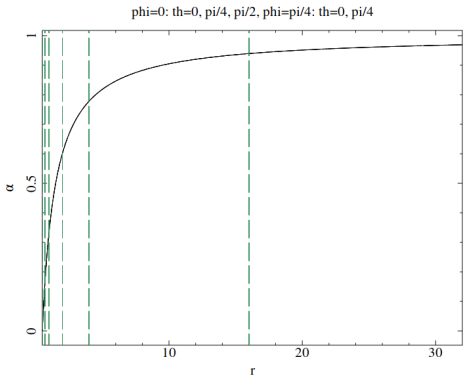
To be sure we have successfully transformed the data we compare the converted metric with the analytical expression. In fig. 13.2 we can see such a comparison. We can see that for both the quantities we have a difference of around 10^{-11}. This is so close that we can in practise call them identical. We might be able to lower this a bit if we are smart with domain limits and resolutions, but as we will see later, this error will be overshadowed by other numerical errors.

We can therefore conclude that our pipeline for transforming data into a spectral representation with the use of LORENE works. We can now safely on to Einstein Toolkit data.

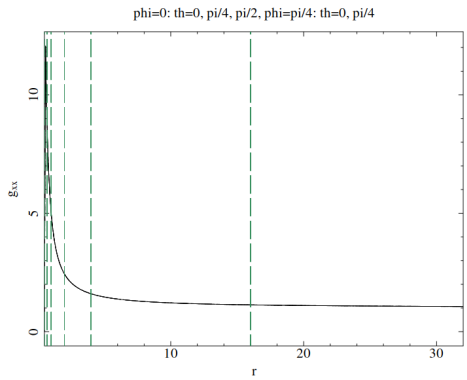
13.2 Single Black Hole

We can now try to convert our Einstein Toolkit simulation of a single black hole to a spectral representation. We will look at five different case, all with different domain limits and/or resolution. Table 13.1 shows the different cases we will look at.

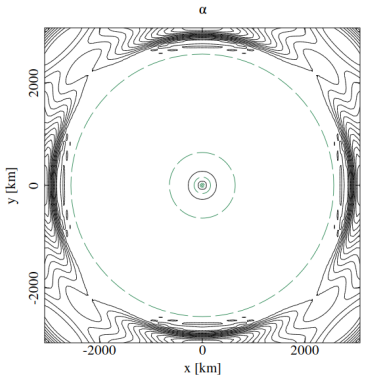
We start by looking at fig. 13.3 we see the same plots as we saw for the analytical case in fig. 13.1. Here *case 5* from the table is used to see a best case scenario. We see that the results are the same as we got with the analytical Schwarzschild metric, meaning that we have successfully transformed the Einstein Toolkit data to LORENE!



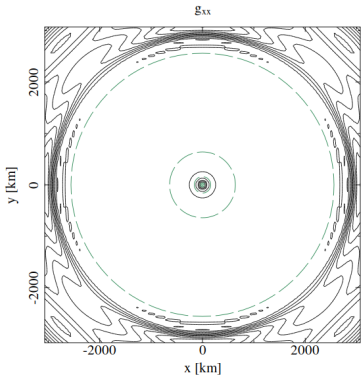
((a)) Plot of the lapse function α after the conversion.



((b)) Plot of the spatial metric coefficient g_{xx} after the conversion.



((c)) Contour plot of the lapse function α after the conversion.



((d)) Contour plot of the spatial metric coefficient g_{xx} after the conversion.

Figure 13.3: Here we can see plots of the lapse function α and the spatial metric coefficient g_{xx} for a single black hole data from Einstein Toolkit, after conversion. The green dashed lines indicates the domain limits. We have used *Case 5* from table 13.1.

Case	Domain	Domain Limits	Resolutions	Time [Min]
1	3	[0.5,8, ∞]	$n_r = 25$, $n_\theta = 7$,$n_\phi = 4$	0.017
2	6	[0.5,1,4,16,32, ∞]	$n_r = 25$, $n_\theta = 7$,$n_\phi = 4$	0.021
3	6	[0.5,1,4,16,32, ∞]	$n_r = 51$, $n_\theta = 15$,$n_\phi = 8$	0.108
4	6	[0.5, 1, 2, 8, 32, ∞]	$n_r = 51$, $n_\theta = 15$,$n_\phi = 8$	0.108
5	9	[0.5, 0.7, 1, 2, 4, 16, 64, 256, ∞]	n_r =[51, 51, 51,51, 51, 51, 25, 25, 25], n_θ =[15, 15, 15, 15, 15, 15, 11, 11, 11, 11],$n_\phi = 8$	0.093

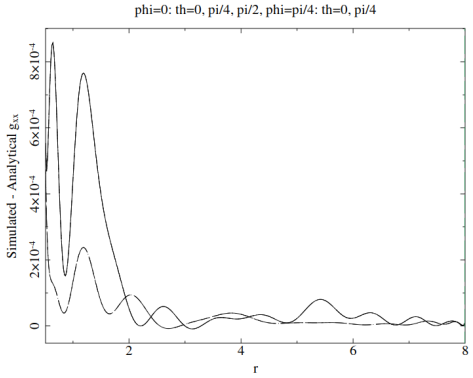
Table 13.1: Table showing the parameters used when converting the single black hole data. Note: For the resolution, when only a single number is given, this means that all the domains have the same resolution.

This is a bold statement, especially when looking at 13.3(c) and 13.3(d), where we see a lot of noise after a sudden radius. The noise is in the outer most domain, the compacted domain going to infinity. This is will discussed more below.

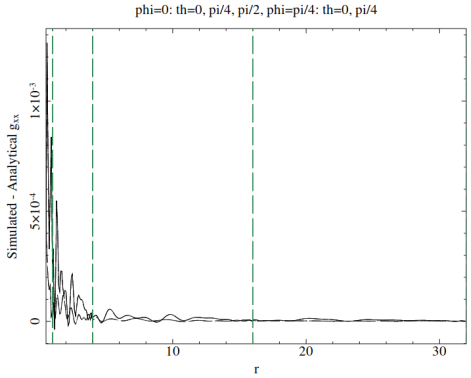
To backup our claim that we have succeeded in the conversion we will need to look at the converted data compared with the analytical metric. Fig. 13.4 show the difference between the converted Einstein Toolkit metric component g_{xx} and the analytical expression. The same plot but for the lapse can be found in the appendix C in fig. C.12. Note that as we have done in most cases before, only the values outside the event horizon $r > r_{horizon} = 0.5$ is plotted and discussed.

The first case 13.4(a) have the same parameters as we had in the text cases.This means that we only have one finite domain(other than the domain near the center), with boundaries 0.5 and 8 – this is the reason there are no green lines in the first plot. We see that we here have a difference up to 8×10^{-4}. This difference becomes less and less the further we move away from the $r = 0$. In 13.4(b) and 13.4(c) three extra domains have been added, and we are using first the same resolution and then doubling – note that n_r and n_θ must be odd. Here we can actually see a increase in the difference at the tallest spikes, but with a faster decrease in the difference after the first spike. It seems that the domain at $r = 1$ is where the decrease happens, meaning that we "lock in" the error in the first domain with this other domain. We can also see that between the other domains the difference seems to decrease.

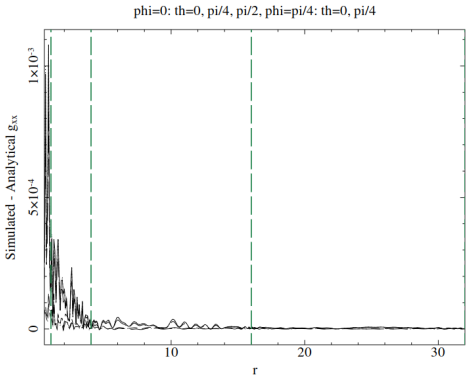
This is due to how LORENE does the transformation. Say that we have a set of points and function values we are going to use Chebyshev polynomials to interpolate. If some of the function values have some numerical error, then we are going to get some error in the spectral coefficients, since we are trying to fit this numerical error into the interpolation. This will lead to the errors we are looking at here. If we now contain the numerical error in its own domain, this domain will have a larger error since



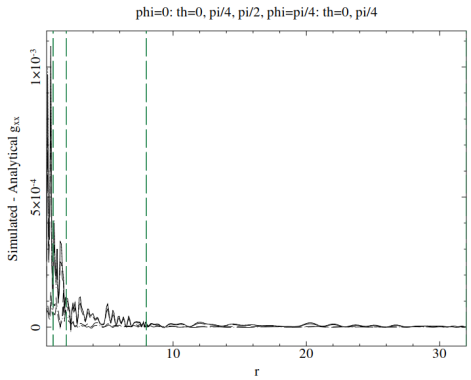
((a)) Difference between simulated and analytical g_{xx} for case 1.



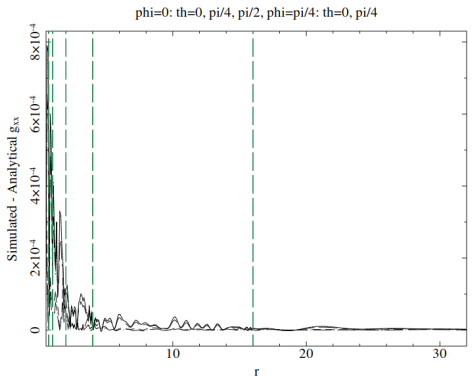
((b)) Difference between simulated and analytical g_{xx} for case 2.



((c)) Difference between simulated and analytical g_{xx} for case 3.

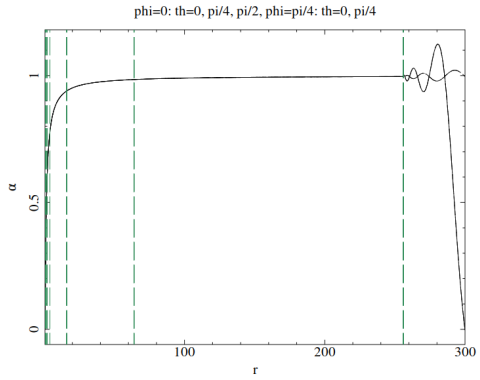


((d)) Difference between simulated and analytical g_{xx} for case 4.

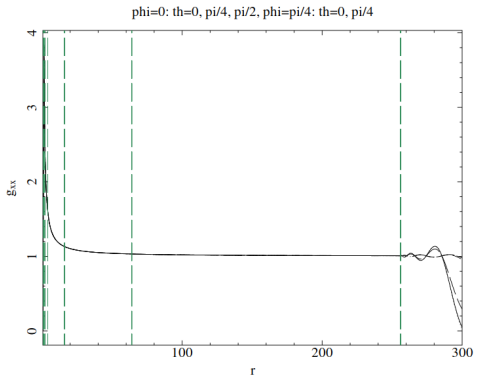


((e)) Difference between simulated and analytical g_{xx} for case 5

Figure 13.4: Here we can see the difference between the simulated and analytical g_{xx} after conversion. The domain parameters are taken from table 13.1. We can see that the differences are around 10^{-4} and dependent on domain limit and resolution. The green dashed lines indicates the domain limits.



((a)) α for distances outside the finite domains.



((b)) g_{xx} for distances outside the finite domains.

Figure 13.5: We can here see α and g_{xx} using case 5 for distances outside of the furthest finite domain. We can see that the data here becomes unusable.

the numerical error takes up a larger part of the data. But the other domain will not have this error, and therefore have a better fit. As we have seen, the error in the data from Einstein Toolkit tends to get larger closer to the center, so we can contain this numerical error by having multiple domains close to the center. We can not eliminate the numerical error completely, but we can keep the error from affecting photons not passing too close to the center.

In 13.4(d) we have done exactly this, by inserting more domains close to the center, and as we can see we still have a high spike close to center, but we now have an even faster decrease in the difference. This is taken even further in 13.4(e), where we also have increased the resolution. We can now see that we have a spike close to the height of the first case, but with the difference decreasing faster.

Before we conclude the single black hole case, we will look at what happens when we go too far out from the center. We saw in 13.3(c) and 13.3(d) that we got some strange noise far out from the center. We can also see this in fig. 13.5. In the last domain, the one going to infinity, we suddenly have large oscillations. These happens for a simple reason: As we have mentioned before, the interpolation of the Einstein Toolkit data only works within the simulated domain. In other words, we have no extrapolation with the none-geometry, except from a constant value. We therefore at some point in the last domain go from a smooth function to some constant value. This leads to a discontinuity in the data, meaning that when LORENE tries to interpolate over this discontinuity we get large oscillations.

There are three solutions to this problem. The first is to use another interpolation in the Python code. The linear interpolation for single grid and multigrid geometries have a linear extrapolation, which can make the spectral transformation more smooth (but not completely smooth). But as we have seen, we are not guaranteed a good interpolation of the data if we use these geometries, and are instead better off using the

none-geometry. The second solution is to choose smart constant values after the last finite domain, i.e. that of Minkowski space. This will lead to a smaller jump when this constant value kicks in, but since the simulated metric won't converge to a Minkowski metric, we will still get a jump, only a bit smaller. The third solution is to ignore the problem. If we only use the data which is within the last domain, then we do not have to deal with this problem. The downs side of this is that we only can ray trace within the simulated domain from Einstein Toolkit. We will use both the second and the third method throughout the results.

We have now showed that we can convert the Einstein Toolkit data of a single black hole to a spectral representation in LORENE. We still get some numerical error, but by using smart domain limits and resolutions we can contain this error close to the center. We have also seen that this error is more or less on the same range as the error we got from reading the Einstein Toolkit data. This indicates that we might not be able to decrease the error in LORENE much more. In fact adding a higher spectral resolution to LORENE might actually increase the errors again, since we then overfit the Chebyshev interpolation instead of smoothing them out due to the other ("more correct") function values.

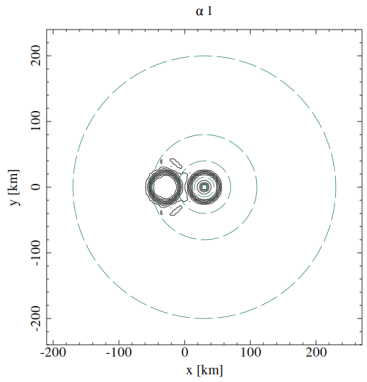
13.3 Binary Black Holes

We have now seen that we can make spectral representations in LORENE of data from Einstein Toolkit. Until now we have only used data with the same spherical topology as LORENE, meaning that we haven't needed to do anything with the with the data from Einstein Toolkit before trying convert it. This is not the case with the binary black hole system. As we have discussed in the theory and method sections we no longer have a spherical topology, and we need to apply the splitting function to the data to make it into two spherical topological systems, one for each black hole – meaning that if we have more objects, we will need to make more systems, as the conversion code is capable of doing.

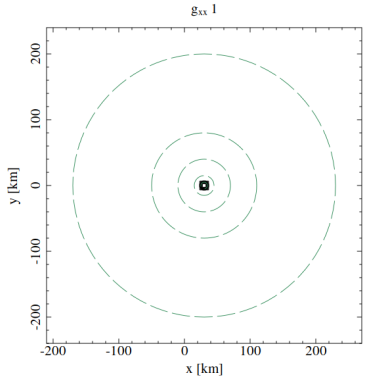
We will first look at two cases with the splitting function applied, but with different resolutions. We will then look at the case with the best resolution, but without the splitting function. This lets us see how both the resolution and the splitting function affects the results.

13.3.1 Effect of Different Resolution

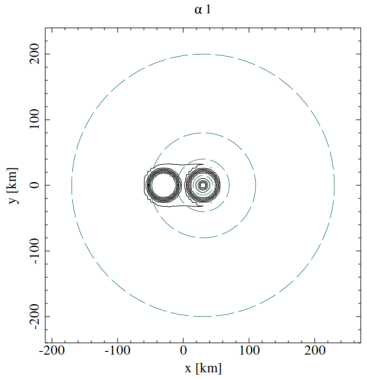
In fig. 13.6 we can find the contour plots after the conversion of a binary black hole system. For all the plots we have the domain limits $[0.5, 1.5, 4, 8, 20, \infty]$, but for the top plots the resolution is $n_r = 51$, $n_\theta = 21$ and $n_\phi = 40$, while for the bottom plots the resolution is $n_r = [135, 135, 135, 135, 67, 57]$, $n_\theta = [51, 51, 51, 51, 51, 31]$ and $n_\phi = [142, 142, 142, 122, 102, 62]$. Note that all of these plots are from the perspective of the black hole at $x = +3$. For the perspective of the other black hole, see fig. C.13 in appendix C. This means that we only have half of the picture, and we have combine the two to get the full 3+1 quantities, which we will look at shortly.



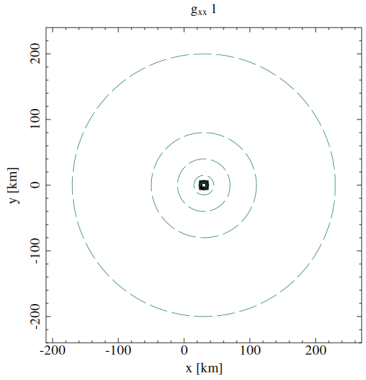
((a)) Contour plot of α for a converted binary black hole system. The conversion with the lowest resolution.



((b)) Contour plot of g_{xx} for a converted binary black hole system. The conversion with the lowest resolution.

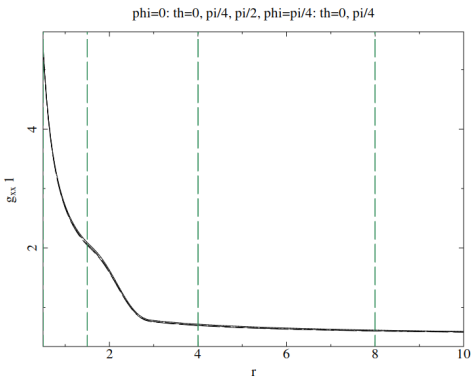


((c)) Contour plot of α for a converted binary black hole system. The conversion with the highest resolution.

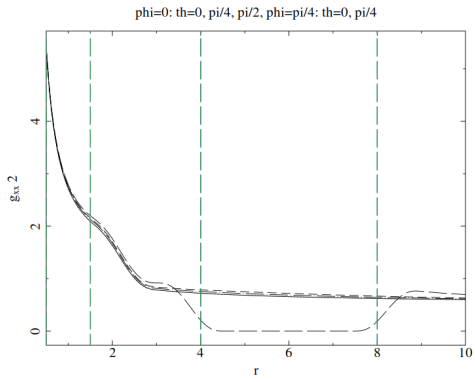


((d)) Contour plot of g_{xx} for a converted binary black hole system. The conversion with the highest resolution.

Figure 13.6: Contour plots of α and g_{xx} for a binary black hole system after conversion. For all the plots we have the domain limits $[0.5, 1.5, 4, 8, 20, \infty]$, but for the top plots the resolution is $n_r = 51$, $n_\theta = 21$ and $n_\phi = 40$, while for the bottom plots the resolution is $n_r = [135, 135, 135, 135, 67, 57]$, $n_\theta = [51, 51, 51, 51, 51, 31]$ and $n_\phi = [142, 142, 142, 122, 102, 62]$. All the plots are of the black hole located at $+3$ at the x-axis, the results for the other black hole is located in the appendix in fig. C.13.



((a)) g_{xx} radially for different values of θ and ϕ for the black hole at x=+3.



((b)) g_{xx} radially for different values of θ and ϕ for the black hole at x=-3.

Figure 13.7: g_{xx} radially for different values of θ and ϕ for both of the black holes. All the radial plots goes from left to right, so the splitting function is only visible for the black hole at $x = -3$, since it is to the left of the other black hole. The resolution is that of the highest resolution in 13.6.

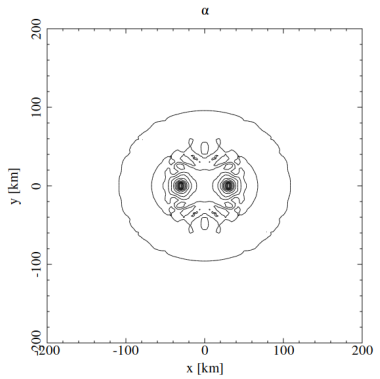
We can see from these plots that for the lapse we have contours in the center which looks like the lapse of the single black hole, and on the left side we have some other region where the lapse decreases. This is the region where the splitting function is applied and the lapse is decreasing smoothly to zero. For g_{xx} we see no such thing. This is because we here don't ignore the center, and the increase in g_{xx} near the center makes the careful decrease of the smoothing function invisible.

The lapse in 13.6(b) is the lapse with the lowest resolution, which we can see from the noise around the region of the splitting function. In C.13(b), where the resolution is higher, the noise is less prevalent.

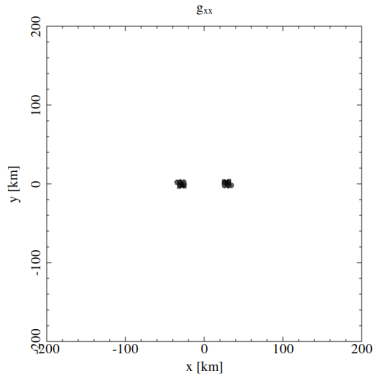
In fig. 13.7 we see g_{xx} for both the black hole at $x = +3$ and that at $x = -3$. All the radial plots goes from left to right, so the splitting function is only visible for the black hole at $x = -3$, since it is to the left of the other black hole. We can therefore see in 13.7(b) a dip in one of the radial plots, where it hits the region of the splitting function. The rest of the rays will not this this region and will therefore all be the same. Notice also that the function value converge to 0.5 instead of 1 as we expect from Minkowski. This is also due to the splitting function, so that when we sum the two function we retrieve Minkowski far away.

To get a read comparison between the two resolutions we need to add the quantities from the two black holes. In fig. 13.8 we see the sums of the two black holes for the two different resolutions. Here we can, for the lapse, see a clear difference, with the lapse with the lowest resolution having a high amount of noise. The one with the highest resolution seems to be very similar to the plots we found in fig. C.8.

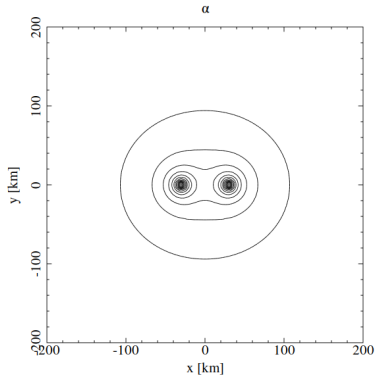
While we don't have a analytical expression to compare the results with, we can see



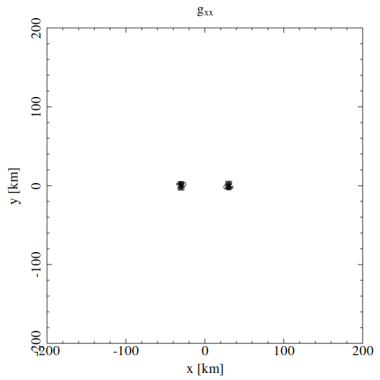
((a)) Contour plot of the sums of the two α for a converted binary black hole system. The conversion with the lowest resolution.



((b)) Contour plot of the sums of the two g_{xx} for a converted binary black hole system. The conversion with the lowest resolution.



((c)) Contour plot of the sums of the two α for a converted binary black hole system. The conversion with the highest resolution.



((d)) Contour plot of the sums of the two g_{xx} for a converted binary black hole system. The conversion with the highest resolution.

Figure 13.8: Contour plots of the sums of the two α's and two g_{xx}'s for a binary black hole system after conversion. For all the plots we have the domain limits $[0.5, 1.5, 4, 8, 20, \infty]$, but for the top plots the resolution is $n_r = 51$, $n_\theta = 21$ and $n_\phi = 40$, while for the bottom plots the resolution is $n_r = [135, 135, 135, 135, 67, 57]$, $n_\theta = [51, 51, 51, 51, 51, 31]$ and $n_\phi = [142, 142, 142, 122, 102, 62]$.

a clear trend that with higher resolution we get results more similar to the results before the conversion. So why not crank the resolution to max? Well, if we look at the runtime we have 1.55 min for the lower resolution and 24.78 min for the higher resolution. The files produced are also of size 104,42MB and 1,65GB respectively, meaning that the increase in resolution costs a lot when it comes to space and time (pun intended).

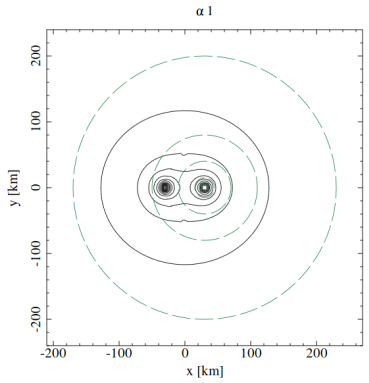
13.3.2 Effects of the Splitting Function

We can now look at how the splitting function affects the conversion of the binary black holes system. We have seen that we can get good results with the splitting function, but for all we know we can get a better result with a smaller resolution without the splitting function. To test this we plot the result above for the higher resolution, but without the splitting function.

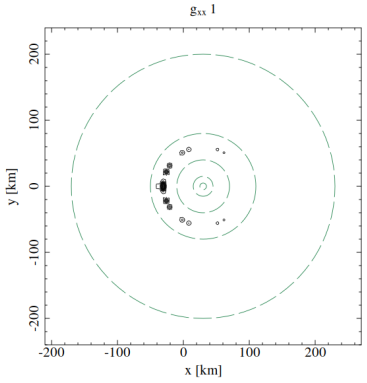
In fig. 13.9 we see the results with out the splitting function. For g_{xx} we can see clear problems, with artifacts appearing in a sphere around the center. This leads some of the same artifacts to appear in the sum also. For the lapse function the results looks better. Even when only using the function centered at the black hole at $x = +3$ we get a result which look fine, even though one can see some artifacts in a sphere passing though the second black hole. The sum looks more or less like the sum with the splitting function.

To see the difference we zoom a bit in. In fig. 13.10 we see a zoomed in version of the sum of the two lapses, with and without the splitting function. We can now see that without the splitting function there is actually some artifacts present. We can therefore conclude that we need the splitting function.

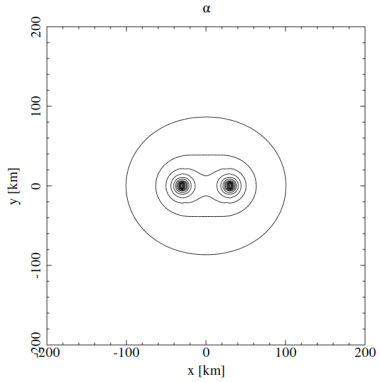
We have thus managed to convert a binary black hole system into a spectral representation using LORENE. We saw that resolution makes the spectral representation better, but that this is at the cost of both time and computer space. The size of the final output file will also have a huge impact on the run time of GYOTO, and will take hours to days for a 1.6GB file. We have also seen that the splitting function was necessary. We expected this due to the spherical topology of LORENE, but we have now seen the effect of not including it.



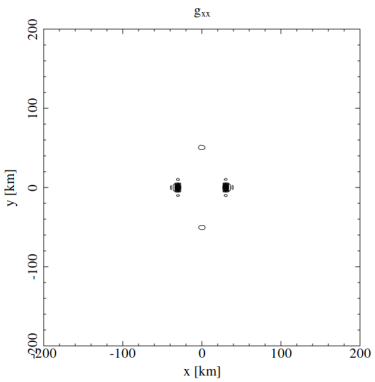
((a)) Contour plot of α for a converted binary black hole system without the splitting function.



((b)) Contour plot of g_{xx} for a converted binary black hole system without the splitting function.

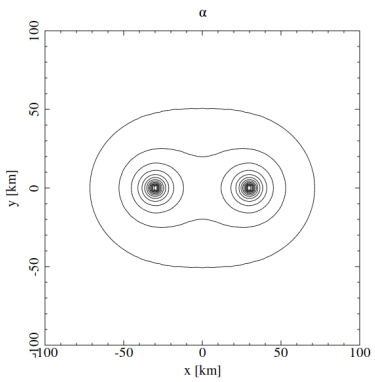


((c)) Contour plot of the sums of the two α for a converted binary black hole system without the splitting function.

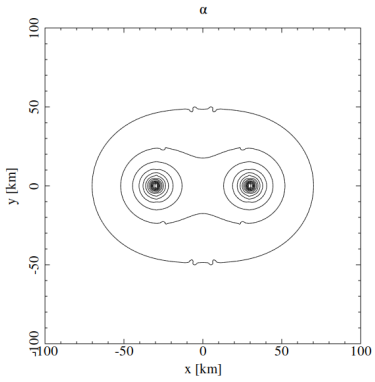


((d)) Contour plot of the sums of the two g_{xx} for a converted binary black hole system without the splitting function.

Figure 13.9: Contour plots of α and g_{xx}'s for a binary black hole system after conversion without the splitting function. The the domain limits are $[0.5, 1.5, 4, 8, 20, \infty]$, and the resolution is $n_r = [135, 135, 135, 135, 67, 57]$, $n_\theta = [51, 51, 51, 51, 51, 31]$ and $n_\phi = [142, 142, 142, 122, 102, 62]$. The dashed lines are the domain limits.



((a)) Zoomed in contour plot of the sum of the two α's for a converted binary black hole system.



((b)) Zoomed in contour plot of the sum of the two α's for a converted binary black hole system without the splitting function.

Figure 13.10: Zoomed in contour plots of the sum of the two α's for a binary black hole system after conversion with and without the splitting function. The the domain limits are $[0.5, 1.5, 4, 8, 20, \infty]$, and the resolution is $n_r = [135, 135, 135, 135, 67, 57]$, $n_\theta = [51, 51, 51, 51, 51, 31]$ and $n_\phi = [142, 142, 142, 122, 102, 62]$.

Chapter 14

GYOTO Results

We have now looked at how the data is read into the converter code and then given to LORENE for the conversion to a spectral representation. The LORENE code will finally make a file which we can give to GYOTO to do the actual ray tracing.

We will in this section first look at the results of ray tracing using a metric created by LORENE alone (so no use of Einstein Toolkit or my conversion code). We will then move on to the test cases, and finally the single black hole simulated in Einstein Toolkit.

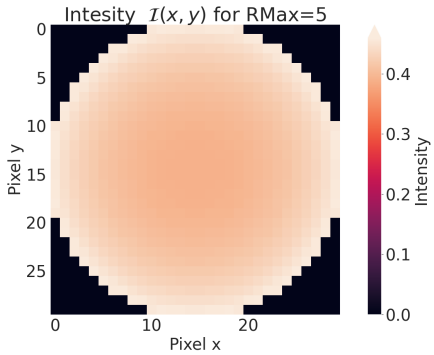
For all the results we will look at a *fixedstar* with different integration radii of *RMax=20* and *RMax=5*(see sec. 10.4). Here we will look at the images which are created, and how the differ. This star has a radius of $R = 3.972$. The camera is placed at $r = 250$ from the center, and has a resolution of 30×30 pixels and field of view of 0.5. We will also look at the photon momentum norm for both of said cases, as well as the case where we have no such star. For the photon momentum norm, any drift away from 0 will be considered as error. The terms drift, norm and error will be used interchangeably when talking about the photon norm in this chapter, but are the same thing.

We will sadly not look at the binary black hole system, since we weren't able to implement the modifications to GYOTO needed to get the addition of multiple metric to work. The represented results will instead show that the use of Einstein Toolkit metrics with GYOTO is possible, and that only said modifications are needed to get the binary black hole system to work in GYOTO.

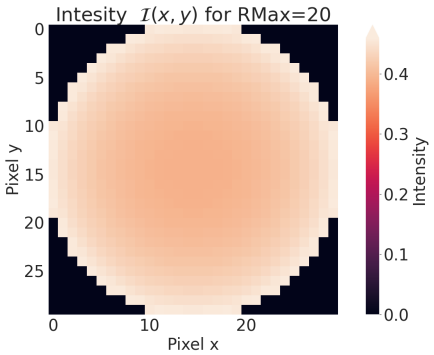
14.1 Metric From LORENE

We will start by looking at the metric created by LORENE alone. Since this is made without using our code or Einstein Toolkit, we will use this as a standard to compare all the other results. The metric is simulated with the program *kerr_ QI.C*, with mass 1, domain limits $[0.51, 1, 2, 4, 8, \infty]$ and resolution $n_r = 25$, $n_\theta = 7$ and $n_\phi = 4$.

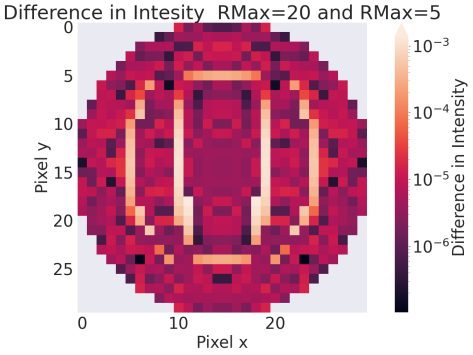
In fig. 14.1 we can see the results of the ray tracing of a fixed star. GYOTO has a cutoff point where it doesn't include the object in the integration. This radius is called



((a)) Image of a fixed star with $RMax = 5$.

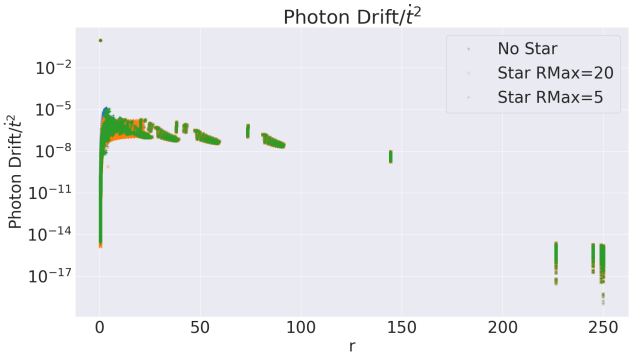


((b)) Image of a fixed star with $RMax = 20$.

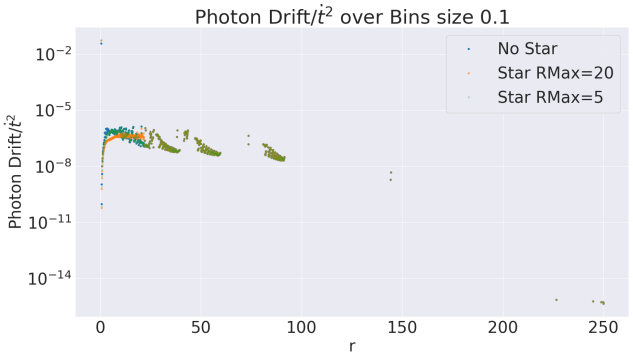


((c)) The difference in pixel values for the two values of $RMax$.

Figure 14.1: Plots showing the intensity of the fixed star. The value at which GYOTO starts integrating with the star is given by $RMax = 5$ and $RMax = 20$. 14.1(c) shows the difference in intensity. The metric used here is a Schwarzschild metric simulated in LORENE with mass 1, domain limits $[0.51, 1, 2, 4, 8, \infty]$ and resolution $n_r = 25$, $n_\theta = 7$ and $n_\phi = 4$. The parameters for the ray tracing can be found in sec. 10.3 or in the appendix E.1.



((a)) Scatter plot showing the drift of the photon momentum. Here all the photons at all the time steps/radii are included.



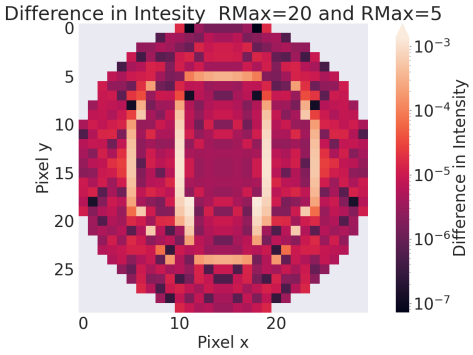
((b)) Plot showing the drift of the photon momentum. Instead of plotting all the radii here we have made bins of 0.1. We then take the mean of all the values in side each bin.

Figure 14.2: The drift of the norm of the photon momenta from the initial value of 10^{-16} plotted as for different radii. This shows The metric used here is a Schwarzschild metric simulated in LORENE with mass 1, domain limits $[0.51, 1, 2, 4, 8, \infty]$ and resolution $n_r = 25$, $n_\theta = 7$ and $n_\phi = 4$. The parameters for the ray tracing can be found in sec. 10.3 or in the appendix E.1.

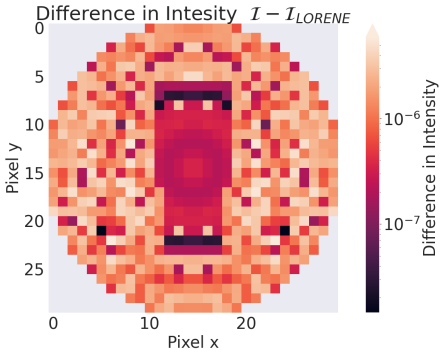
$RMax$ in the gyoto XML file, which can be found in sec. 10.3 or in the appendix E.1. Here, as well as in all plots of of the fixed star, we will be looking at $RMax = 5$ and $RMax = 20$. The plots of the different $RMax$' can be seen in 14.1(a) and 14.1(b). We see that we have successfully made a picture of a star. By them self, the stars with different $RMax$ don't tell us so much. 14.1(c) shows the difference between the intensities of the two images. We expect the difference to be zero, but it is in fact of order 10^{-6} to 10^{-3}.This tells us that $RMax$ actually have something to say when it comes to the integration of the photons, even though this value is well outside of the star (meaning that we still are in a region were we should only have vacuum).

In fig. 14.2 we see drift of the norm of the photon momenta from the initial value of 10^{-16} plotted as for different radii from the center of the black hole. The left plot shows all the integration steps for all the photons, while in the right plot we have placed all the values into bins along the r-axis with size 0.1 and taken the average of each bin. Notice that we don't have the drift alone, but instead normalized it with the time derivative of the time component of the metric $\dot{t}$. This is done to correct for divergent behaviour near the event horizon. We can here see that there is a drift in the photon momentum. We expect the norm to stay constant as the photons move along null geodesics, but this isn't the case here. Near the event horizon we have a norm drift of 10^{-6}, and even some values of 0.1. We can take this drift to be a consequence of numerical integration error.

We can also observe the effects of the adaptive integration steps GYOTO uses. All the steps should be plotted here, so where there are no points in the plot, the integrator



((a)) The difference in pixel values for the two values of $RMax$.



((b)) The difference of a fixed star with $RMax = 20$ made using the test case and one made using LORENE.

Figure 14.3: Plots showing the intensity of the fixed star. 14.3(a) shows the difference in the intensity between $RMax = 5$ and $RMax = 20$, and 14.3(b) is the difference between a fixed star with $RMax = 20$ made with the test case and the one we made using the LORENE metric. The test case uses domain limits $[0.51, 1, 2, 4, 8, \infty]$ and resolution $n_r = 25$, $n_\theta = 7$ and $n_\phi = 4$. The parameters for the ray tracing can be found in sec. 10.3 or in the appendix E.1.

never evaluates any photons.

If we look at fig. 14.2 we see a radius at which the three different ray tracings starts to diverge from one another. This is easiest to see for the case with $RMax = 5$(orange) and the one with $RMax = 20$(green). This radius is 20, in other words, where GYOTO starts to take the star into consideration in $RMax = 20$. We see that for this case(orange) the spread of the values of the drift of the norm becomes larger. We will look at that later when we look at the single black hole simulated in Einstein Toolkit.

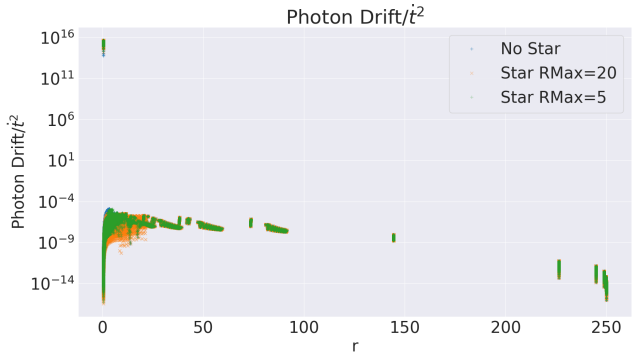
We have now successfully ray traced our first numerical metric, but one which is made by the creators of GYOTO to be used in GYOTO. We can therefore use this as a comparison for both our test case and our simulated black hole. This means that all of the parameters for the fixed star and the type of plots will be the same for the other cases below, making it simple to compare with this metric.

14.2 Test Case

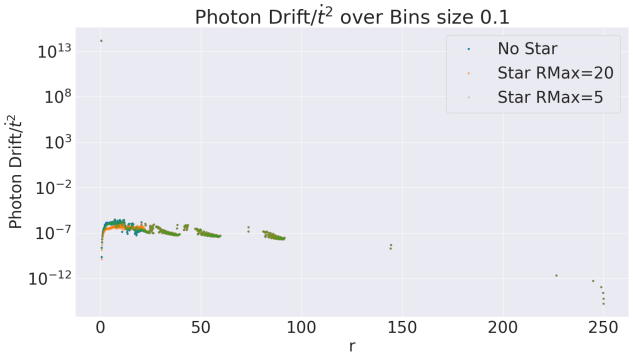
We can now look at some of the same plots, but using the test case: the converted analytical isotropic Schwarzschild metric we have been looking at before. As with the results for the LORENE conversion, this lets us test the second half of the pipeline.

We will make a test case of a Schwarzschild metric with the same spectral parameters as the LORENE metric above: mass 1, domain limits $[0.51, 1, 2, 4, 8, \infty]$ and resolution $n_r = 25$, $n_\theta = 7$ and $n_\phi = 4$. This lets us compare them directly.

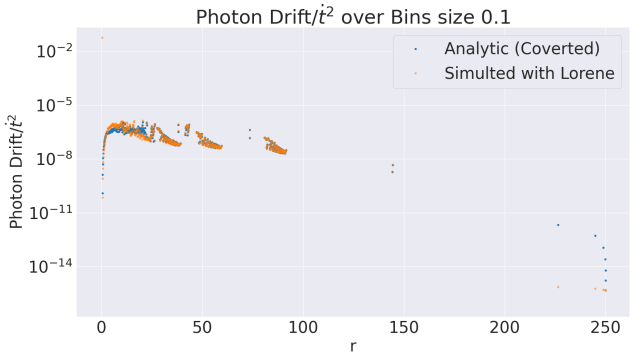
In fig. 14.3(a) we can see one intensity plot comparing $RMax = 5$ and $RMax = 20$,



((a)) Scatter plot showing the drift of the photon momentum. Here all the photons at all the time steps/radii are included.



((b)) Plot showing the drift of the photon momentum. Instead of plotting all the radii here we have made bins of 0.1. We then take the mean of all the values in side each bin.



((c)) The norm drift for without a star using both the test case and the LORENE metric

Figure 14.4: The drift of the norm of the photon momenta from the initial value of 10^{-16} plotted as for different radii.Here we have used the Schwarzschild test case with mass 1, domain limits $[0.51, 1, 2, 4, 8, \infty]$ and resolution $n_r = 25$, $n_\theta = 7$ and $n_\phi = 4$. The parameters for the ray tracing can be found in sec. 10.3 or in the appendix E.1.

where we can see that we get a difference at the order 10^{-3} to 10^{-6}, which is the same as we did for the LORENE metric. In fig. 14.3(b) we see the difference between the intensity for a fixed star with our test case and one made with the LORENE metric. We expect them to be the same, and with a difference on the order of 10^{-6} we have a very similar intensity. In table 14.1 we can see an overview of the absolute and relative errors we get when summing over all the pixels in the image. The test case has only a relative error of 1.6×10^{-4} between the two types $RMax$' and only a relative error of 1.34×10^{-6} from the LORENE metric.

In fig. 14.4 we can see the norm drift for the test case. We see that 14.4(a) and 14.4(b) as very similar to the same plots made using LORENE. There are some outliers

which reach 10^{13} close to the horizon. When looking at the printout of the norm, these will lie at or just passed the event horizon, meaning that they are included due to numerical errors in the integration steps. They should not have any effects on the results.

In fig. 14.4(c) we see norm drift without a star for the test case and the LORENE metric. From this we can see that the results are more or less the same. There are differences, especially at $r = 250$. But these are so small that we can conclude that we have managed to do ray tracing on our test case with the same accuracy as our standard (the LORENE metric). This means that the conversion to a spectral representation using LORENE works as intended.

14.3 Single Black Hole

We have now seen that our test case, a converted analytical isotropic Schwarzschild metric, gives results close to the LORENE metric. We can now move on to look at effects of the reading of data from Einstein Toolkit on the ray tracing results, i.e. create an actual ray tracing using Einstein Toolkit data.

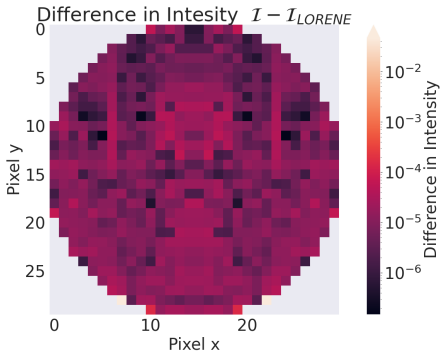
We will start by looking at the same type of plots as we need for the test case, and then we will look at some plots showing what happens to the norm when we have a star.

14.3.1 Comparison with LORENE

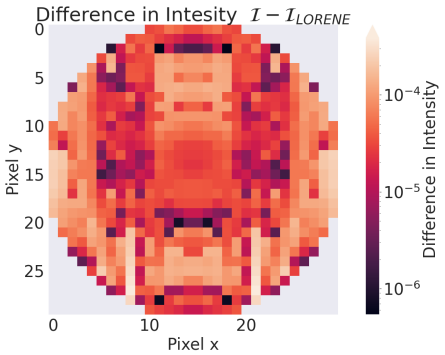
We will look at three different cases of single black holes. The difference is the spectral parameters. The parameters can be found in table 14.2. All the cases have the resolution $n_r = [25, 25, 25, 25, 17, 7, 7, 7, 7]$, $n_\theta = 11$ and $n_\phi = 8$. As we have discussed when we looked at what happens outside the simulated domain, there are two ways of handling this: One is to have a cutoff at some r, where we instead enforce Minkowski, and the other way is to ignore it, and have a finite domain covering further out than we are ray tracing and just ignore everything outside this. Here we will look at both cases. The first case will have the last final domain very close to the center, and at $r = 280$ the conversion code will no longer use the Einstein Toolkit data to get the function values, but instead enforce Minkowski. This will also be the case for case 2, but the last finite domain is further out than the start of the ray tracing (this starts at $r = 250$). The last case will have the same parameters, but without the cutoff. In this case the none-geometry will just give a default constant value of 0.

In fig. 14.5 we can see the comparisons between ray tracings of a fixed star with $RMax = 20$ using the three different cases and LORENE. We can see that they are more or less the same, differing only by a factor 10^{-5}. The results are not as good as the test case. If we look at table 14.1 we can see that the relative error is more or less a factor 100 worse than the difference we got using the test case.

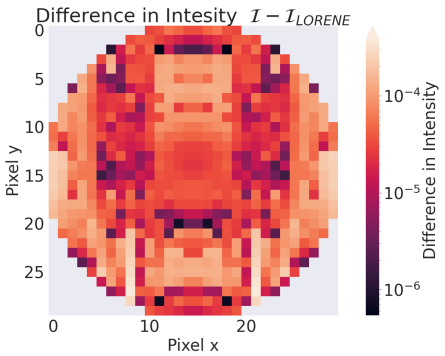
We can also see that the results of both fig. 14.5 and table 14.1 are more or less identical for case 2 and 3, meaning that since the last finite domain covers the start of the ray tracing, then what happens afterwards is unimportant. We will not conclude



((a)) Difference in intensity between case 1 and LORENE.

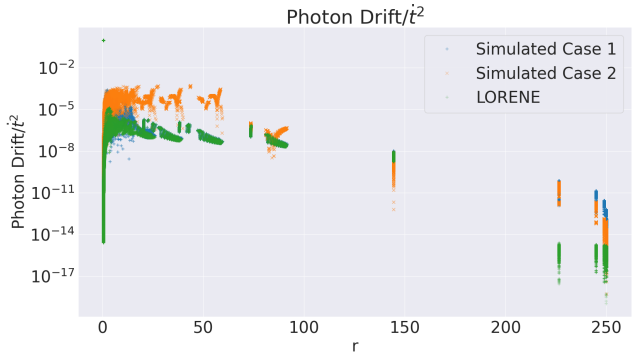


((b)) Difference in intensity between case 2 and LORENE.

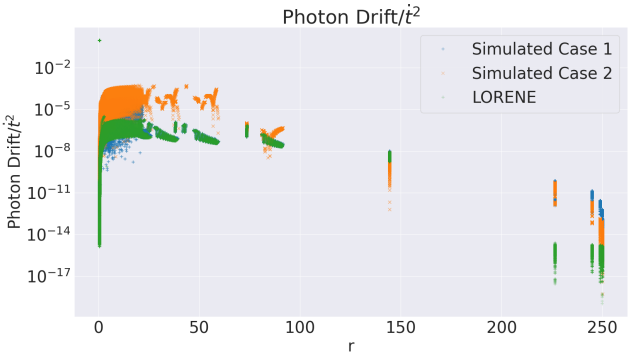


((c)) Difference in intensity between case 3 and LORENE.

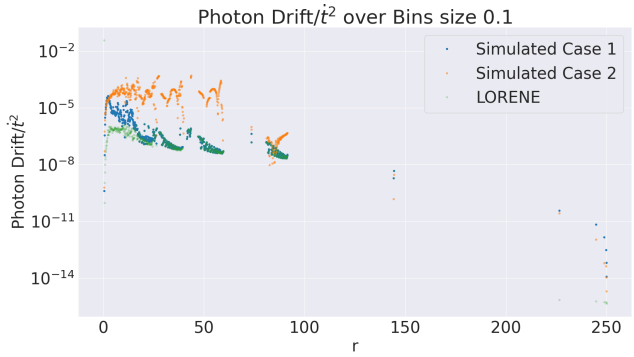
Figure 14.5: The difference in intensity between fixed stars with $RMax = 20$ using metrics simulated with Einstein Toolkit and a metric made by LORENE. The different spectral parameters for the different cases are found in table. 14.2.



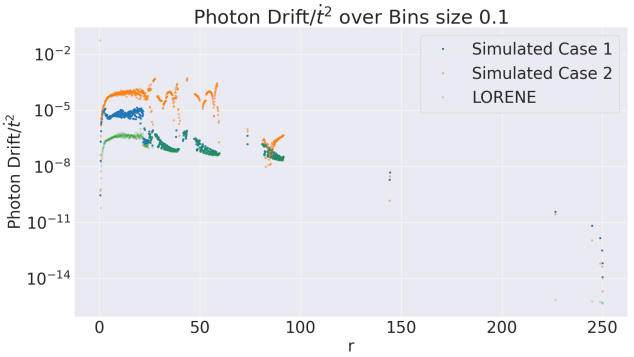
((a)) Scatter plot of drift in norm for different cases using no star.



((b)) Scatter plot of drift in norm for different cases using fixed star with $RMax = 20$.



((c)) Binned plot of drift in norm for different cases using no star.



((d)) Binned plot of drift in norm for different cases using fixed star with $RMax = 20$.

Figure 14.6: Photon norm drift for case 1 and 2 compared with LORENE, without a star and with a fixed star with $RMax = 20$.

that this always is the case, but for these spectral parameters, this is the case. We will therefore not compare these two results anymore, and instead focus on the difference between case 1 and 2.

We now come to what could be taken as the most important result yet. Fig. 14.6 we see the drift in photon norm for case 1(blue) and 2(orange) together with that of our LORENE metric(green). We can immediately see that there is huge difference between some of the cases. Case 1 seems to follow the LORENE metric quite nicely until some r, where it increases. If the plot uses a fixed star, this r is around 20, which should not surprise us. Where no star is used, this r is instead around 16. As we will discuss below, this makes sense.

If we look at the results for case 2, we have a larger error, and without the sudden increase we saw with case 1. This can be explained by the fact that the error is already larger for case 2 by the time it reaches the r at which the error of case 1 increased. This

means that the change in error is smaller than the already existing error. What we can see is that when we have a fixed star at $RMax = 20$ we instead of an increase in error at this points, we instead get a higher spread of error. We will look at the reason for this below.

So what is the reason for the increase in error at certain r's for case 1 and the all around high error of case 2? When we are using case 1, the last finite domain ends at $r = 16$. Meaning that the last, infinite, domain have only seven points to resolve $r = 16$ to $r = \infty$. This isn't that much of a trouble for LORENE, but what happens is instead that all the small errors from the reading is ironed out by the small sampling, giving us a result more or less the same as for the LORENE metric. When we hit $r = 16$ we now have many more points to resolve the error, increasing the error rapidly. For case 2 we have finite domain all the way out, with enough points to resolve the error in all the domains. The error will therefore have time to slowly build up.

We have seen that test case has more or less the same precision as the LORENE metric. This means that the error we see here comes from either Einstein Toolkit or, more likely, from the reading and interpolation of the data.

All this means that we are able to do ray tracing using Einstein Toolkit metrics(!), but we have to expect some errors. If, as is the case with a single black hole, we can use LORENE to iron out a lot of the error which comes from the reading of the data, but when using metric where more precision is needed we have to be ware of this error. The error does not in anyway invalidate the results, since we get an error from case 1 of, at most, 10^{-3} compared to 10^{-6} with LORENE.

There is most likely other spectral parameters which gives better results, and lies between case 1 and two, but there are no magic rules for a set of parameters we can give to achieve this, and instead the user must test different parameters to find some that suits his or hers simulated metric.

14.3.2 Reasons for Larger Norm Drift for Fixed Star

We will here look only at case 1. As we saw in fig. 14.6 we saw that when the photons cross $r = RMax$ we get an increase and spread in the photon norm. The reason for this is two-fold. The first reason is that when we are inside of $RMax$ GYOTO will use more integration steps. We can see this in fig. 14.7, where each integration point for each photon is plotted for both case 1 and the LORENE metric. The difference between the points representing case 1 and the LORENE metric is difficult to see, but the important part is the total number of points. We see that at some r the number of points increase. We can see from the two plots that we have $r = RMax$ when this begins to happen. Since we always have $30 \times 30 = 900$ photons, this means that it is the number of integration steps that must increase. This is therefore the reason for why we see more points in 14.6 when we have a fixed star.

We can also from 14.7 clearly see that we have a bending of the photons, meaning that we can conclude that we actually have successfully been able to ray trace!

The last question is now why we have a higher error when we have a fixed star. Fig. 14.7 begins to give us a hint. Since all the other plots we have seen of the photon

Type	Difference RMax [Absolute/Relative]	Difference from LORENE [Absolute/Relative]
LORENE	$5.47\times10^{-5}/1.6\times10^{-4}$	-
Test Case	$5.39\times10^{-5}/1.6\times10^{-4}$	$1.34\times10^{-6}/3.91\times10^{-6}$
Case 1	$1.6\times10^{-4}/4.8\times10^{-4}$	$1.1\times10^{-4}/3.3\times10^{-4}$
Case 2	$7.31\times10^{-5}/2.1\times10^{-4}$	$4.97\times10^{-5}/1.5\times10^{-4}$
Case 3	$7.31\times10^{-5}/2.1\times10^{-4}$	$4.97\times10^{-5}/1.5\times10^{-4}$

Table 14.1: An overview of the total absolute and relative differences between the intensity for a fixed star with $RMax = 5$ and one with $RMax = 20$, and the difference between the different cases and a metric from LORENE. For this, latter, difference a fixed star with $RMax = 20$ is used.

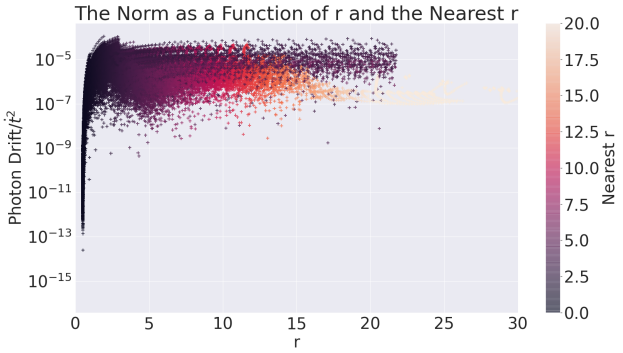
Case	Domain Limit	Cutoff r
1	0.5, 1, 2, 3, 4, 6, 8, 16, ∞	Yes, at $r = 280$
2	0.5,1,2, 8, 16, 64, 128, 256, ∞	Yes, at $r = 280$
3	0.5,1,2, 8, 16, 64, 128, 256, ∞	No

Table 14.2: Different spectral parameters used to make the different data files used to test the ray tracing on a single black hole simulated in Einstein Toolkit. All the cases have the resolution $n_r = [25, 25, 25, 25, 17, 7, 7, 7, 7]$, $n_\theta = 11$ and $n_\phi = 8$.

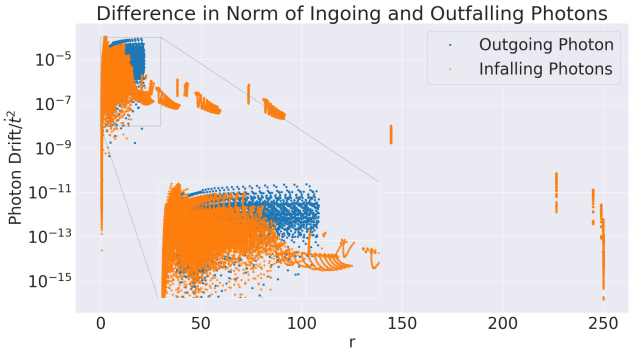
((a)) Fixed star with $RMax = 5$.

((b)) Fixed star with $RMax = 20$.

Figure 14.7: Here the photon norm drift is plotted in a polar plot. Case 1 and the LORENE metric are used. Each point is one integration step for one photon. So more points, thus more integration steps. The color of the point is the norm drift.



((a)) Fixed star with $RMax = 20$.



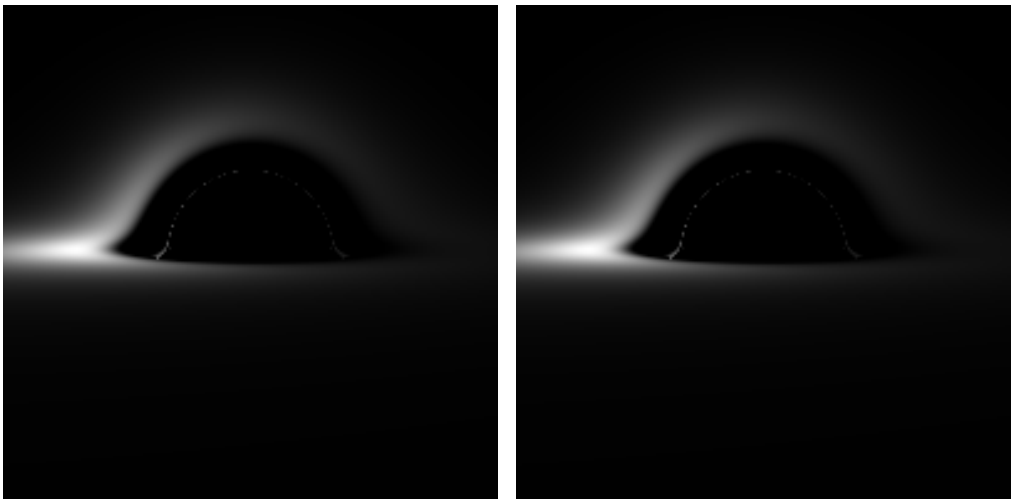
((b)) The orange points are photons falling into the black hole, and the blue are photons already having been close to the black hole, and are now leaving.

Figure 14.8: Here we see two different plots showing how the norm of the photons are affected by coming close to the center. Both are made with case 1 and $RMax = 20$.

norm are only dependent on r we do not know if they are falling into the black hole or on their way away from the black hole. Our original thought was that the increase in error is due to the photons having travelled close to the black hole, picked up a lot of error and then moved away. When we have no star, GYOTO will not do any more integration steps after passing the black hole, since there is nothing on the other side. But when we have a star, GYOTO will continue to integrate the photons that have passed the center but still are in the star. This also means that a large $RMax$ will lead to the photons being tracked longer. This is the reason we see an increase in error at $RMax$: Not only are there more integration steps to pick up error, but we are also seeing the photons travelling away from the center, but still have $r < RMax$.

In fig. 14.9 we see two different ways of looking at how the photons are affected by getting close to the center. 14.8(a) is the normal norm drift plot we have looked at before, but the points are colored according to closest point they have been to the center. So for a photon falling in the current r is the same as the closest r, but for an outgoing photon the current r is larger than the closest r. We can see from this plot that from the infalling photons have a smaller norm, while the outgoing photons make up most of the photons with a high norm. This is better from 14.8(b), where all the infalling photons are colored orange and all the outgoing photons are colored blue. We see that most of the spread we observed previously is due to the outgoing photons. The increase in the norm of the infalling photons do actually not start to increase before $r = 16$, which is the exact same behaviour as the ray tracing without any star.

We can thus conclude that the reason for the increase and the greater spread of the photon norm drift when using a fixed star is due the photons travelling close to the center, picking up a larger error and then moving away! If this effect is removed we are left with the same norm pattern as for the ray tracing without the star.



((a)) Page-Thorne disk using a Einstein Toolkit metric(case 3 in table 14.2).

((b)) Page-Thorne disk using the LORENE metric.

Figure 14.9: Two 200×200 images of a Page-Thorne disk, using case 3 and the LORENE metric. As we can see, they are almost identical. This proves that it is possible to ray trace using a metric from Einstein Toolkit.

14.4 Summary and a Final Showcase Result: Page-Thorne Disk

We have showed that we can get the same results using a Einstein Toolkit metric, as we did with a standard metric simulated with LORENE. There will be some errors, i.e. a photon norm drift away from zero, but they are small enough so that they don't get in the way of any physics. We have also seen that when using a fixed star the error increases inside of the radius where GYOTO takes the star into consideration when integrating. This error was demonstrated to be photons travelling away from the center with the error they had picked up close to the center. This does not change the images of the fixed stars we have been looking at, but if there were to be any objects around the fixed star we want to look at, the results could be affected by the picked up errors.

From the results of the photon norm drift and intensity of the fixed star we can conclude that we have successfully managed to bring an Einstein Toolkit metric to GYOTO and used it to do ray tracing!

Thus we have seen many diagnostic plots and boring stars with a low resolution. To end the results we have made a larger image using case 3 and the LORENE metric. This ray tracing is of a Page-Thorne Disk. We will not analyze the norm of the pixels of this result, but instead study its beauty! This is also the final proof we have succeeded in using an Einstein Toolkit metric in GYOTO!

Part IV

Conclusion

Chapter 15

Conclusion

15.1 Summary

Two types of metrics were considered: A single black hole and a binary black hole system. The single black hole metric worked as a base metric, easy to compare to analytical solution and therefore told us if the pipeline from Einstein Toolkit to GYOTO worked, and if any numerical errors were introduced. Using the lessons learned from this single black hole metric, we could use this pipeline on the binary black hole merger, without having any analytical expressions to compare with.

The first step consisted of simulating the two metrics. The single black hole used an analytic isotropic Schwarzschild metric to create the initial conditions. The binary black hole system used a two puncture method to initialize the simulation. Both simulations were then read into the conversion program using the library PostCactus. We had to choose between multiple ways of reading and interpolating the data from Einstein Toolkit. After rigorous testing comparing the single black hole metric to an analytic expression after reading and interpolation, we found that the *none-geometry*, where the reading is done only for the points using in the Einstein Toolkit simulation and a linear interpolation is applied, gave superior runtime and results. For the other geometries, where the result of the simulation is interpolated unto an uniform grid and a linear or spline interpolation might have its uses, we showed that the behaviour of these geometries were to volatile to be used.

The error of the interpolated single black hole data compared to the analytical expression was around 10^{-6}, meaning that some error would be carried from the reading and interpolation of simulation data to the final ray tracing.

We then looked at the resolution of the Einstein Toolkit simulation. We found that the numerical error after reading decreased linearly with an increase in the resolution. The computational resources did increase close to exponential with the resolution, meaning that we settled on a suitably good resolution without trying to push to up to get marginally better errors.

The next step was to use LORENE to do the spectral transformation. We ended up using a C-code separate from the main Python code to give the collocation points

needed to do the transformation. This meant running the C-code using a subprocess in the Python code, and then catching and formatting the output of the C-code. This lets us find the values at the collocation points using the data we read and interpolated from Einstein Toolkit in the Python code. Since the simulated data are finite and have a symmetry for faster simulation, the Python code handled this. The binary black hole metric broke the spherical topology of LORENE and GYOTO, meaning that we had to introduce a splitting function to split the black hole into separate, pseudo-spherical topologies to reduce error caused by the spectral transformation. The values of at the collocation points could then be passed back to the C-code where calls to LORENE could to the final transformation to a spectral transformation.

The results after the spectral transformation could once again be compared to analytical expressions (at least for the single black hole). Here we found that the results were highly dependent on the spectral parameters used (the domain limits and resolution). For test cases here analytical metrics went through the transformation process, we were able to keep the error down to 10^{-11}. For the simulated single black hole metric we were able to keep the error down to the level at which it was before the transformation with the correct limits and resolutions. For the binary metric we had to measure by eye the quality of the transformation. We found that the resolution needed to get good results were significantly higher than for the single black hole. We also saw that the splitting function was needed, and without that the results were had artifacts in them. But we were able to create good results for the binary metric using this splitting, and outputting one file per black hole, so that the final metric would be a sum of the files.

The last step was the ray tracing. Since GYOTO lacks the final modifications needed to use multiple files, all the ray tracings were done only for the single black hole. Here we used a metric simulated in LORENE by the creators of GYOTO as a standard to compare the results. Using the drift of the photon momentum norm, we found that the analytical test case gave more or less the same results as our LORENE metric, meaning that the spectral transformation itself worked. Using a ray traced image of a fixed star as measures of error, we also found that the two metrics were close enough that we concluded that the transformation was a success.

For the simulated single black hole metric we saw that, once again, the end results were quite dependent on the spectral parameters. When using a transformation where the last finite domain was close to the black hole and where we enforced Minkowski after $r = 280$, we found that we got results close to the LORENE and test metrics, with the errors starting to appear only after the photons entered the finite domains. When the last finite domain ended after the start of the photons, $r = 250$, we saw that the errors were larger throughout the ray tracing. We concluded that this was due to the fact that when the last finite domain ended early, LORENE used few points to interpolate $r = 16$ to ∞, smoothing out the errors, but when the entire path of the photons were inside finite domain, there were more points for LORENE in interpolate the errors. The errors were at an order 1000 higher than the errors from the LORENE and test metrics, but still low in an absolute term (10^{-3} compared to 10^{-6}), and we therefore concluded that we were successful in ray tracing a metric of a single black

hole simulated in Einstein Toolkit.

15.2 Conclusion

The goal of the thesis was to be able to use spacetimes simulated in Einstein Toolkit to do ray tracing on objects like black holes and black hole mergers using GYOTO. Since GYOTO is built upon LORENE, this meant we would have to transform the spacetimes from Einstein Toolkit from a finite difference system into a spectral representation. What we ultimately wished to find out, was whether it would be possible to use a binary black hole merger in the ray tracing, so that further studies of the gravitational waves emitted by merger could look at the effects of the waves on the photons.

We showed that this was possible for a isotropic Schwarzschild metric created in Einstein Toolkit. Reading the data from Einstein Toolkit resulted in some numerical errors, which persisted all the way to the ray tracing. These errors were small enough that they were comparable to the integration errors made by GYOTO. This leads us to conclude that for a single back hole metric, it is possible to do ray tracing using GYOTO, and to get physics about out of this, minding the errors created by the reading, interpolation and transformation of the data.

As for the binary black hole system, we were able to read, interpolate and transform the data. This required the splitting of the system into multiple systems with only one black hole, which we demonstrated made the results after the spectral transformation converge to the results before the transformation. We were not able to do the actual ray tracing of the binary black hole system, since GYOTO still needs to implement a way of using the sum of multiple files to create one single metric. This work is under way, and will be continued after this thesis.

This means that we have to wait to look at the physics of ray tracing through gravitational waves. We also saw that it takes a much greater resolution to remove noise from the binary black hole system. This means that we are not ensured that we will ever reach the same accuracy as with the single black hole, and how this affects gravitational waves is unknown. The massive metric files created by this increased resolution will also mean that the ray tracing will take much longer time. But this will have to be looked into in the future.

All in all we weren't able to be able to look at gravitational waves using a binary black hole merger and GYOTO. But we laid down the ground work, and made a pipeline connecting Einstein Toolkit and GYOTO – we even got at nice Page-Thorn disk image from it. This is not a trivial results in of itself, and managing to get this Einstein Toolkit simulation to GYOTO is a good result, and leaves the door open for more simulations done in Einstein Toolkit to be used in GYOTO.

15.3 Future Work

The most obvious work to be done is the implementation of the use of multiple metric files in GYOTO. As mentioned this work is in progress. I'm still working with the

creators of LORENE and GYOTO, Éric Gourgoulhon and Frederic Vincent, to get this feature to work.

When we manage to get GYOTO to work with the binary black hole system, we can finally begin to look the gravitational waves created by the merger.

The converter code runs on Python2.7 since PostCactus wasn't ported to Python3 when this project started. Since then some other users have ported PostCactus to Python3[46]. This means that the whole conversion code should be ported to Python3 as well.

A lot of the parameters used were made ad hoc and were found by trial and error, making them quite difficult to use for future users. The hope is to get a better understanding of which parameters works best for the different situations, so that the user only have to apply small changes depending on their specific spacetimes .

The use of the conversion code is as of now quite involved, with different parameters found in either the Python or the C code. The parameters found in the C-code must also be recompiled after changing them. The plan is to wrap everything in a user interface which handles all the parameters using a parameter file, and dynamically gives them to the C-code during runtime. A system for log files are also in the making, since too many hours were spent during this thesis trying to remember which parameters were used where. This wrapper is already in the process of being put together.

As stated earlier in the thesis, the code was ultimately meant to be made into a Thorn for Einstein Toolkit. The fact that future user might have to use trial and error when choosing parameters might make it difficult to have a tool that does the conversion run time, so having a Python tool that can do post-processing instead might be the way to go. This means that the Python tool might stay as a standalone tool.

All in all the goal is to publish the code as an open source research tool. This means that we need to both understand the error and how minimize them, as well as making the code easy to use. This work will continue after the thesis is done, and hopefully this dream will become a reality.

Appendices

Appendix A

Installing Einstein Toolkit

The installation of Einstein Toolkit is quite easy and a good guide is found at their web page[50]. We will here go through the steps quickly and then see how to add small customizations to the installation.

1. Make and navigate to a folder where you want to install Einstein Toolkit
2. Run

```
curl -kLO https://raw.githubusercontent.com/gridaphobe/CRL/
ET_2020_11/GetComponents
chmod a+x GetComponents

```

   This will download a script for download the components of Einstein Toolkit, or more precisely Cactus. Since the different components use different version control solutions, like SVN and Git, this script will handle everything without the user having to keep track of which component uses what. The download link will change as new versions of Einstein Toolkit is released. The second line will makes the script an executable.

3. Then run

```
./GetComponents --parallel https://bitbucket.org/
einsteintoolkit/manifest/raw/ET_2020_11/einsteintoolkit.th

```

   This runs the script which downloads all the components. The second argument gives the so called thorn list. As we saw in sec. 4.2.5 Einstein Toolkit/Cactus consists of a collection of thorns. We need different thorns to do different simulations, and can therefore give a list of thorns which are needed in the current installation. The list given here is a general list, containing most of the thorns used by the community. Many of the thorns are not necessary, but we will still download them – it is easier and faster than going through the list and removing the ones we don't need. Note that this will also download the thorn list, since

we need it when we are compiling. We can also choose not to compile all the downloaded thorns if we change the thorn list.

4. We can then setup Cactus before compiling. We start by running

```
cd Cactus
./simfactory/bin/sim setup-silent

```

This will take us into the Cactus folder, where most of the simulations will happen. The next line make different config files for the compilation, guessing at the type of machine the user is running and the different type of compilers. Most of the config file will be correctly set up, but a file which is smart to change is `./simfactory/mdb/machines/<hostname >.ini`. Here we can change the number of cores and threads the simulations can use, as well as where the output of the simulations are stored. The output can be very large, so it is good to change this to a disk with a lot of space.

5. To compile run

```
./simfactory/bin/sim build -j16 --thornlist ../einsteintoolkit.
th

```

This will compile the thorns found in the thorn list. Here we use the same as above, which will compile all the thorns. This will, in my experience, take about 20-60 minutes to compile, so just grab some coffee. The argument `-j16` tells the compilers to use 16 cores/threads. The larger this number is, the faster the process should take.

6. Einstein Toolkit is now installed and ready for colliding black holes! Below we will go through some examples how to run the simulation. Here I will show the generic command to start the simulation

```
./simfactory/bin/sim create-run simulation_name --parfile
parameters.par

```

The first command tells Einstein Toolkit to create a simulation with the name *simulation_name* and then run it. The second parameter tells Einstein Toolkit how to setup the simulation and what thorns to use. More on that below. We can also choose to just create the simulation by running this with `create`. We can then later run the command

```
./simfactory/bin/sim submit simulation_name --cores=2 --num-
threads=1 --walltime=0:20:0

```

This is used to submit the simulation with a simulation script, if the user are running on a cluster or a supercomputer. The parameters `cores` and `num-threads`

are self-explanatory. `walltime` is the maximum runtime, which often is needed to set when running on clusters or supercomputers. If we want to see how out simulation are going, we can run

```
./simfactory/bin/sim list-simulations simulation_name

```

and

```
./simfactory/bin/sim show-output --follow simulation_name

```

The first will output the created simulation and their status (in most cases *RUNNING* or *FINISHED*). This will also give the PID of the simulation, so that the user can stop it if wanted. The second command will output the simulation output. This is very useful for monitoring how the simulation is going along.

We are now ready to run simulations. All the steps here are taken from the tutorial found at [50]. Here you can also find out to run small test simulations like a TOV star.

Appendix B

What Happens Inside the Code

We will here go through how to use the code, and a summery of what happens when in the code. This is the same as described in the method section, but in a short and chronological order, so that it is easier to understand what happens when. As we saw in the method section, there are a lot of code which does a lot of things. This is for the most part hidden from the user, so that they only have to run a couple of commands to get the code to work. Note that everything described below (and the whole code for that matter) depends on the simulation done by Einstein Toolkit being of n black holes with the corresponding 3+1 quantities being given as 3d output. The simulation must also give the position and horizon of the black holes. Other than this the user will only have to do the following to do the conversion:

First they have to define the domains and resolution of the spectral transformation. This is done in the C code. The user will then have to run the makefile to compile the program.

The user can then move on to the Python code. Here the user will first have to make an instance of the converter class. They can then use this class to make as many geometries as they want, depending on the complexity of the metric they want to convert. Thirdly they have to make an instance of the class which reads the Einstein Toolkit data and makes the interpolation. The user will also have to make a list over all the iterations that they want to convert. Lastly the user can call first instance to do the actual conversion of the metric. This function will know the number of bodies the spacetime holds, and will apply the splitting function accordingly. A typical use of the conversion code will look something like

```
inter = ETInterpolater(sim_folder, number_sim_bodies)
g = inter.make_positive_geometry(corner, grid_points)

it = [0, 128,256] #etc

et_q = ReadQuantites(g, it, folder, pickle_folder=pickle_folder, pickle=
    True)

inter.analyse_bbh(g, et_q, it, quantities=quantities, test=False,
    do_gyoto_converstion=False)
```

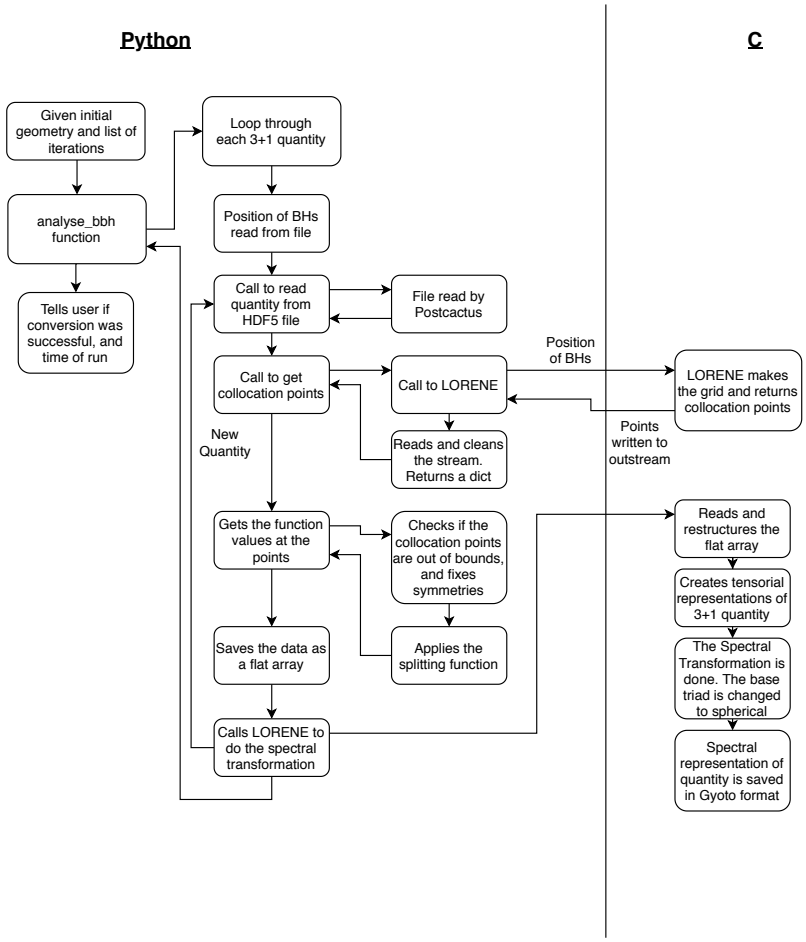


Figure B.1: Schematics over the flow of the converter. The left most column shows what is done by the user; the center python column shows the main loop doing conversion; and the right python column shows auxiliary python functions. The C column shows the LORENE functionalities called by the C code.

We see here that `do_gyoto_converstion` is false. In this case the user will have to run the line `./get_points x y z 1 nb_body it` afterwards to get LORENE to do the conversion. If `do_gyoto_converstion` is true, then the converter will do this by itself.

We will now go over what happens inside of the code In figure B.1 one can see a schematic representation of the conversion code.

1. For all the geometries the 3+1 quantites given by the users are read. They are then interpolated onto the grid points. From these values either a spline or a linear interpolation is made by the ReadQuantites instance. If pickled version of these interpolations are found, they will be used instead. This is given to the conversion class

2. analyse_bbh is now called, and does the rest of the work.

3. The code will read the position and radius of the black holes[1] from Einstein Toolkit. The position will be used to calculate the splitting function (6.1).

4. For each iteration given by the user, the code will loop though all the 3+1 quantities – the lapse function, and all the different indices of the shift vector, the spatial metric and the extrinsic curvature. The below is what will happen for each quantity at a given iteration.

5. Now a call is made to get the collocation points needed to do the spectral transformation. A C code is called with the information about the origin of one of the black holes – so that the grid can be centered correctly. The C code has defined the different domains, and how many points it should use for each domain. Using this, and the origin given by the Python code, LORENE will calculate the spectral grid and print it to outstream. The Python function then captures the stream the C code makes, and reads and formats lists of collocation points. The program now has the x, y and z coordinates of all the collocation points.

6. The code will now begin to calculate the function value on the collocation points. This is done in the following steps

    - The symmetries discussed above must be taken care of. For the most trivial case, with positive x, y and z-values, all negative coordinates are simply mirrored unto the positive side. For other symmetries, other measures are taken.
    - Care are taken at the bounds. The collocation points may be outside of the simulated geometry, so points outside the geometry must be either given a value from inside the geometry or extrapolated.
    - The function values are calculated for the desymmeterized and bounded values using a call to the instance of ReadQuantites.

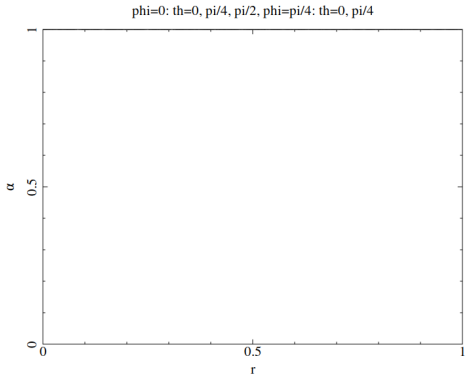
[1] It might be other objects than black holes, but I will use black holes as an umbrella term here

- The splitting function (6.1) is applied to the function values.

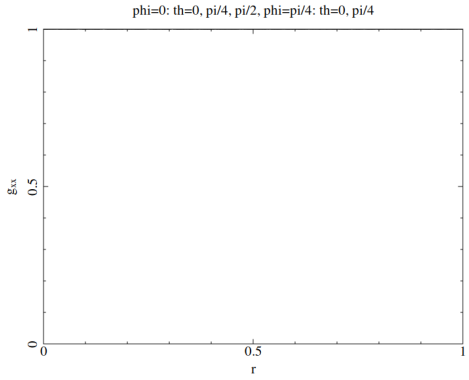
The end result is then returned as a flat array, with a mapping back to the formatting used by the collocation points. The mapping is $d[l][k][j][i] = (\sum_{0<=m<l} n_r[m]* n_t[m] * n_p[m]) + k * n_t[l] * n_r[l] + j * n_r[l] + i$, with l being the domain, k being the k'th ϕ-value, j being the j'th θ-value and i being the i'th r-value, and n_r, n_t and n_p being the arrays of points in the different domains. This array is written to file with a specific name.

7. Points 4 and 5 are done for the second black hole (if there are two black holes).
8. Points 3-6 are done for the next quantity, until all are processed.
9. Having all the collocation points filled with function values for each 3+1 quantity, the C code is again called. This code will then
   - The C code reads the files with the function values.
   - All the 3+1 quantities are defined as tensorial objects, and their values are read from the read arrays.
   - LORENE is then asked to make a spectral transformation on the quantities.
   - Since Einstein Toolkit is in Cartesian coordinates, the 3+1 quantities are currently in Cartesian coordinates as well. Since GYOTO used spherical coordinates, the base triads of the quantities are changed to spherical coordinates.
   - The quantities are then saved together with the map of the spectral grid to a file readable by GYOTO.

The program is now done, and GYOTO can be run.



((a)) Plot of the lapse function α after the conversion.



((b)) Plot of the spatial metric coefficient g_{xx} after the conversion.

Figure C.1: Here we can see plots of the lapse function α and the spatial metric coefficient $g_x x$ for a Minkowski metric after conversion. Three domains where used in this conversion, with the limits $[0.5, 8, \infty]$ and the resolution $n_r = 25$, $n_\theta = 7$ and $n_\phi = 4$. The results $\alpha = g_{xx} = 1$ are hidden but the automatic limits of the plotting class.

Appendix C

Additional Plots

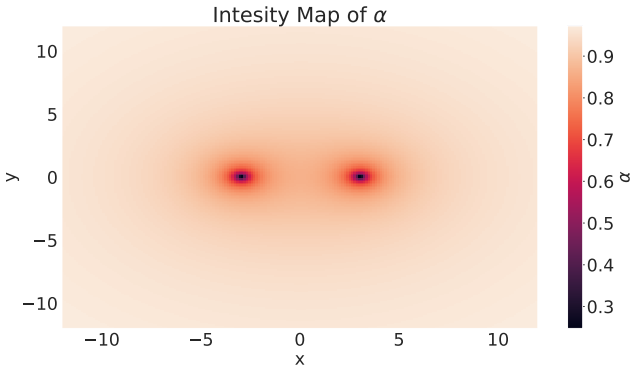


Figure C.2: An intensity map of α.

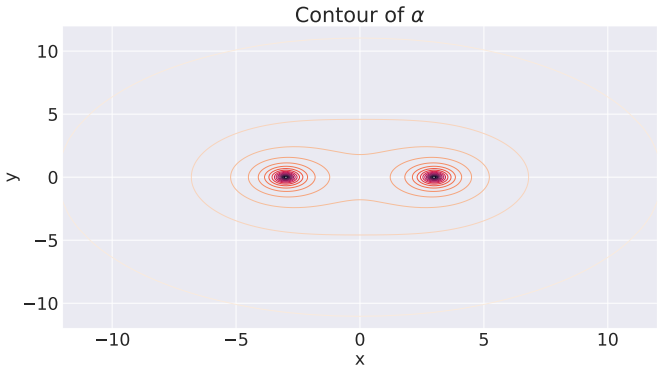


Figure C.3: A contour plot of α

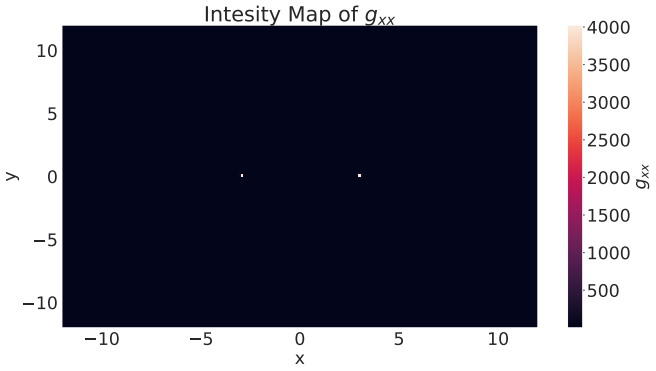


Figure C.4: An intensity map of g_{xx}

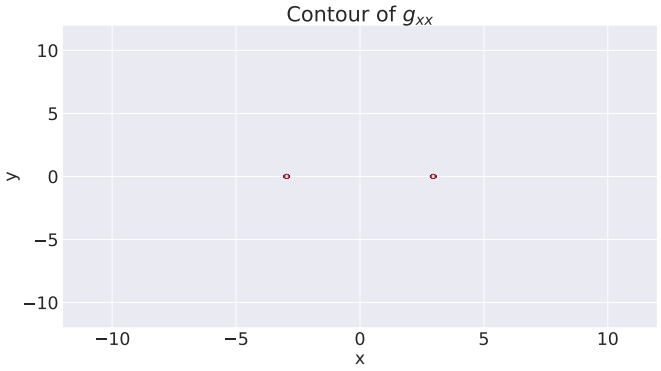


Figure C.5: A contour plot of g_{xx}

Figure C.6: The results of the reading and interpolation of the Einstein Toolkit data for a binary black hole system using a multigrid with a linear interpolation. We see that compared to a single black hole, we do not seem to get the errors at 10^{60}. These will be the results we use as a comparison to the data after conversion to LORENE

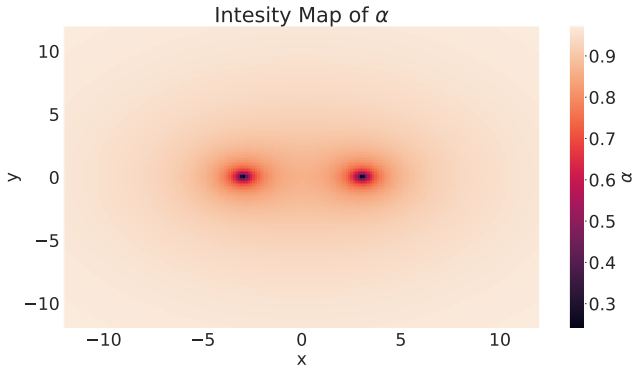


Figure C.7: An intensity map of α.

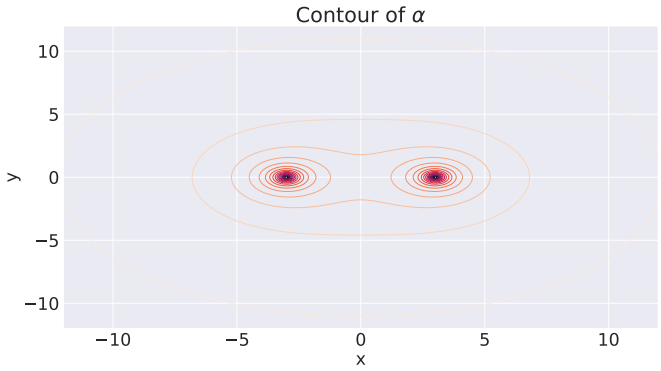


Figure C.8: A contour plot of α

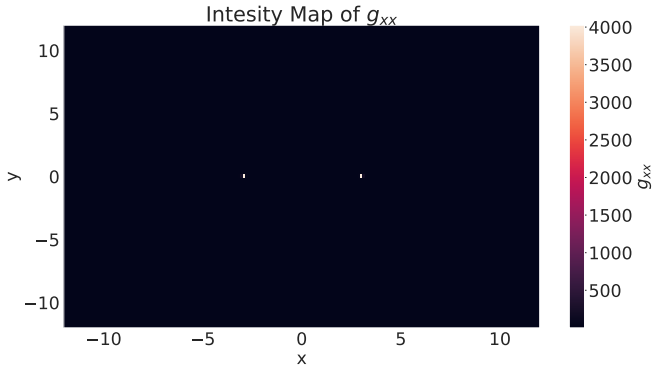


Figure C.9: An intensity map of g_{xx}

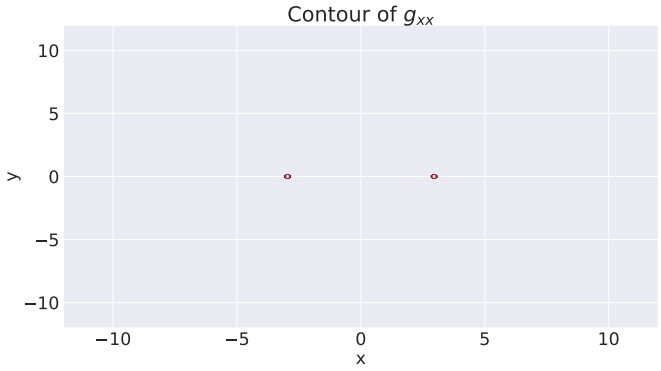
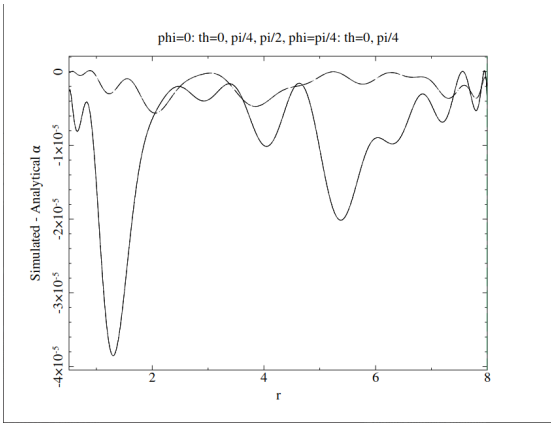
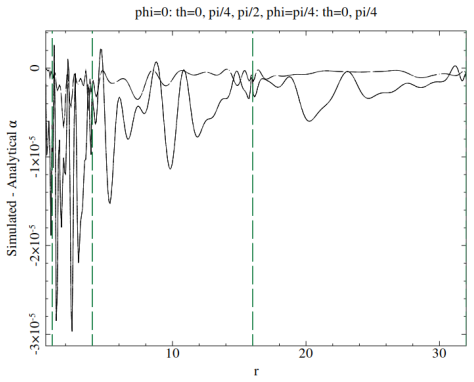


Figure C.10: A contour plot of g_{xx}

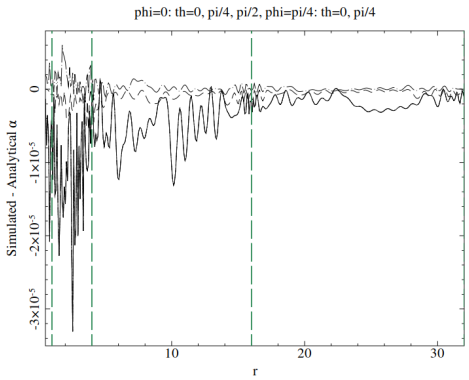
Figure C.11: The results of the reading and interpolation of the Einstein Toolkit data for a binary black hole system using a none-geometry. These will be the results we use as a comparison to the data after conversion to LORENE



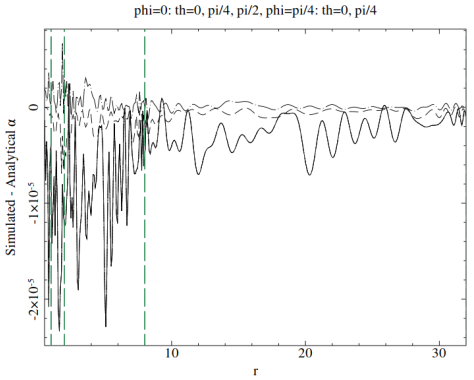
((a)) Difference between simulated and analytical α for case 1



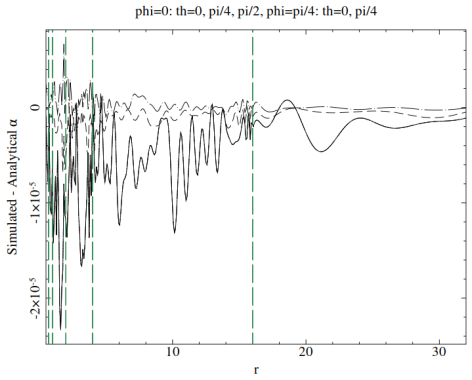
((b)) Difference between simulated and analytical α for case 2.



((c)) Difference between simulated and analytical α for case 3.

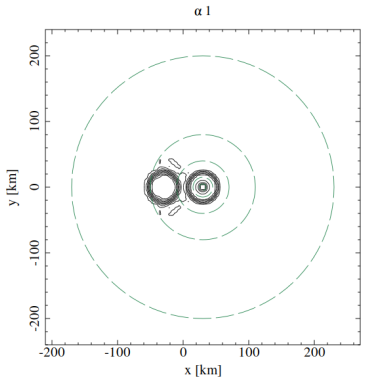


((d)) Difference between simulated and analytical α for case 4.

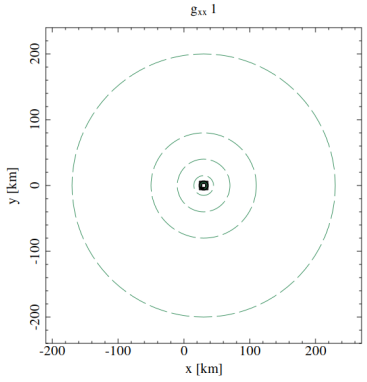


((e)) Difference between simulated and analytical α for case 5.

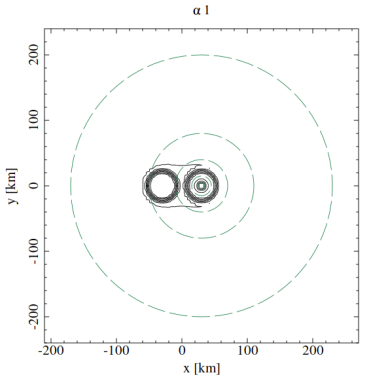
Figure C.12: Here we can see the difference between the simulated and analytical α after conversion. The domain parameters are taken from table 13.1. We can see that the differences are around 10^{-5} and dependent on domain limit and resolution.



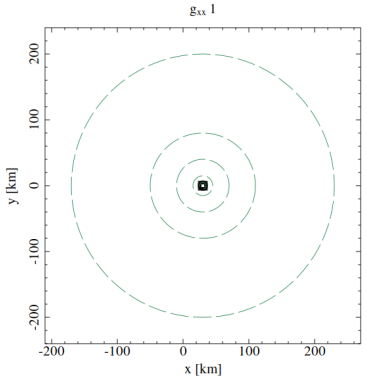
((a)) Contour plot of α for a converted binary black hole system. The conversion with the lowest resolution.



((b)) Contour plot of g_{xx} for a converted binary black hole system. The conversion with the lowest resolution.



((c)) Contour plot of α for a converted binary black hole system. The conversion with the highest resolution.



((d)) Contour plot of g_{xx} for a converted binary black hole system. The conversion with the highest resolution.

Figure C.13: Contour plots of α and g_{xx} for a binary black hole system after conversion. For all the plots we have the domain limits $[0.5, 1.5, 4, 8, 20, \infty]$, but for the top plots the resolution is $n_r = 51$, $n_\theta = 21$ and $n_\phi = 40$, while for the bottom plots the resolution is $n_r = [135, 135, 135, 135, 67, 57]$, $n_\theta = [51, 51, 51, 51, 51, 31]$ and $n_\phi = [142, 142, 142, 122, 102, 62]$. All the plots are of the black hole located at -3 at the x-axis, the results for the other black hole is located in fig. 13.6.

Appendix D

Einstein Toolkit Parameter Files

D.1 Schwarzschild Black Hole

```
        #AHFinderDirect
        #NaNChecker
ActiveThorns = "
        Boundary
        CartGrid3D
        CoordBase
        IOUtil
        InitBase
        MoL
  Time
        SymBase

        ADMBase
        ADMCoupling
        ADMMacros
        CoordGauge
        SpaceMask
        StaticConformal


        Carpet
        CarpetIOASCII
        CarpetIOBasic
        CarpetIOScalar
        CarpetLib
        CarpetReduce
  CarpetRegrid2
        CarpetInterp
        LoopControl

        ReflectionSymmetry
        RotatingSymmetry180

        TwoPunctures
        Formaline
```

```
        GSL

        Slab

        GenericFD

        ML_BSSN
        ML_BSSN_Helper
  SphericalSurface
  TmunuBase
  Dissipation

        CarpetIOHDF5


        AEILocalInterp LocalReduce

        QuasiLocalMeasures

        WeylScal4 MultiPole

  NoExcision

  IDAnalyticBH

  StaticConformal
"
#        NoExcision LocalInterp SummationByParts

# output

IO::out_dir = $parfile

IOBasic::outInfo_every = 1


Activethorns = "CarpetIOHDF5"

# 3D HDF5 Output
 CarpetIOHDF5::out3D_every = 2048
carpetIOHDF5::out3D_vars = "
         ADMBase::lapse
         ADMBase::shift
         ADMBase::metric
         ADMBase::curv
"
### Checkpointing

CarpetIOHDF5::checkpoint                = no


#--- driver

Cactus::cctk_itlast = 0
```

```
Cactus::terminate = iteration
Cactus::cctk_final_time = 1
Carpet::use_buffer_zones      = yes
Carpet::use_tapered_grids     = yes

CarpetLib::interleave_communications = yes
CarpetLib::combine_sends             = yes
CarpetLib::print_memstats_every      = 1024
#Carpet::regrid_in_level_mode         = no
Carpet::output_timers_every          = 1024
CarpetLib::print_timestats_every     = 1024
Carpet::print_timestats_every        = 1
#Carpet::init_each_timelevel          = yes
InitBase::initial_data_setup_method="init_all_levels"


carpet::verbose = no
carpet::veryverbose = no

#--- Carpet prolongation order settings

Carpet::prolongation_order_space               = 5
Carpet::prolongation_order_time                = 1


#--- grid and symmetries

driver::ghost_size            = 3
grid::type = "CoordBase"
CoordBase::domainsize = "minmax"
CoordBase::xmax = 300
CoordBase::ymax = 300
CoordBase::zmax = 300
CoordBase::xmin =   0.000
CoordBase::ymin =-300
CoordBase::zmin =   0.000
CoordBase::dx   =   2.0
CoordBase::dy   =   2.0
CoordBase::dz   =   2.0


ReflectionSymmetry::reflection_x   = no
ReflectionSymmetry::reflection_y   = no
ReflectionSymmetry::reflection_z   = yes
ReflectionSymmetry::avoid_origin_x = no
ReflectionSymmetry::avoid_origin_y = no
ReflectionSymmetry::avoid_origin_z = no
CarpetRegrid2::symmetry_rotating180     = yes

CoordBase::boundary_shiftout_x_lower = 1
CoordBase::boundary_shiftout_z_lower = 1

Carpet::domain_from_coordbase = yes

```

```
CoordBase::boundary_size_x_lower = 3
CoordBase::boundary_size_y_lower = 3
CoordBase::boundary_size_z_lower = 3
CoordBase::boundary_size_x_upper = 3
CoordBase::boundary_size_y_upper = 3
CoordBase::boundary_size_z_upper = 3


carpet::max_refinement_levels             = 10
CarpetRegrid2::num_centres              = 1
CarpetRegrid2::num_levels_1             = 10
CarpetRegrid2::position_x_1             = +0.0
CarpetRegrid2::radius_1             [1] = 128.0 # 1.536
CarpetRegrid2::radius_1             [2] =  64.0 # 0.768
CarpetRegrid2::radius_1             [3] =  16.0 # 0.384
CarpetRegrid2::radius_1             [4] =   8.0 # 0.192
CarpetRegrid2::radius_1             [5] =   4.0 # 0.096
CarpetRegrid2::radius_1             [6] =   2.0 # 0.048
CarpetRegrid2::radius_1             [7] =   1.0 # 0.048
CarpetRegrid2::radius_1             [8] =   0.7 # 0.024
CarpetRegrid2::radius_1             [9] =   0.5 # 0.024


#--- initial data
#admbase::metric_type = "static conformal"
#admbase::metric_type  = "yes"
ADMBase::initial_data      = "schwarzschild"
ADMBase::initial_lapse     = "schwarz"
ADMBase::initial_shift     = "zero"
ADMBase::initial_dtlapse   = "zero"
ADMBase::initial_dtshift   = "zero"
ADMBase::lapse_timelevels  = 2
ADMBase::shift_timelevels  = 2
ADMBase::metric_timelevels = 2
idanalyticbh::mass = 1.0


ADMMacros::spatial_order = 4
```

D.2 Binary Black Hole

```
Cactus::cctk_run_title = "BBH"

Cactus::cctk_full_warnings         = yes
Cactus::highlight_warning_messages = no

Cactus::terminate       = "time"
Cactus::cctk_final_time = 4.0



ActiveThorns = "IOUtil"

```

```
13 IO::out_dir = $parfile
14
15
16
17 ActiveThorns = "AEILocalInterp"
18
19 #ActiveThorns = "BLAS LAPACK"
20
21 ActiveThorns = "Fortran"
22
23 ActiveThorns = "GSL"
24
25 ActiveThorns = "GenericFD"
26
27 ActiveThorns = "HDF5"
28
29 ActiveThorns = "LocalInterp"
30
31 ActiveThorns = "LoopControl"
32
33 ActiveThorns = "Slab"
34
35
36
37 ActiveThorns = "SummationByParts"
38
39 SummationByParts::order = 4
40
41
42
43 ActiveThorns = "InitBase"
44
45
46
47 ActiveThorns = "Carpet CarpetLib CarpetInterp CarpetReduce CarpetSlab"
48
49 Carpet::verbose           = no
50 Carpet::veryverbose       = no
51 Carpet::schedule_barriers = no
52 Carpet::storage_verbose   = no
53 #Carpet::timers_verbose    = no
54 CarpetLib::output_bboxes  = no
55
56 Carpet::domain_from_coordbase = yes
57 Carpet::max_refinement_levels = 7
58
59 driver::ghost_size       = 3
60 Carpet::use_buffer_zones = yes
61
62 Carpet::prolongation_order_space = 5
63 Carpet::prolongation_order_time  = 2
64
65 Carpet::convergence_level = 0
66
```

```
67 Carpet::init_fill_timelevels = yes
68 #Carpet::init_3_timelevels = yes
69
70 Carpet::poison_new_timelevels = yes
71 CarpetLib::poison_new_memory  = yes
72
73 Carpet::output_timers_every      = 640
74 CarpetLib::print_timestats_every = 640
75 CarpetLib::print_memstats_every  = 640
76
77
78
79 ActiveThorns = "NaNChecker"
80
81 NaNChecker::check_every     = 1 # 64
82 #NaNChecker::verbose         = "all"
83 #NaNChecker::action_if_found = "just warn"
84 NaNChecker::action_if_found = "terminate"
85 NaNChecker::check_vars      = "
86         ML_BSSN::ML_log_confac
87         ML_BSSN::ML_metric
88         ML_BSSN::ML_trace_curv
89         ML_BSSN::ML_curv
90         ML_BSSN::ML_Gamma
91         ML_BSSN::ML_lapse
92         ML_BSSN::ML_shift
93         ML_BSSN::ML_dtlapse
94         ML_BSSN::ML_dtshift
95         ADMBase::metric
96         ADMBase::curv
97         ADMBase::lapse
98         ADMBase::shift
99         ADMBase::dtlapse
100        ADMBase::dtshift
101 "
102
103
104
105 ActiveThorns = "Boundary CartGrid3D CoordBase ReflectionSymmetry
        RotatingSymmetry180 SymBase"
106
107 CoordBase::domainsize = "minmax"
108
109 CoordBase::xmin =    0.00
110 CoordBase::ymin = -120.00
111 CoordBase::zmin =    0.00
112 CoordBase::xmax = +120.00
113 CoordBase::ymax = +120.00
114 CoordBase::zmax = +120.00
115 CoordBase::dx   =    2.00
116 CoordBase::dy   =    2.00
117 CoordBase::dz   =    2.00
118
119 CoordBase::boundary_size_x_lower     = 3
```

```
CoordBase::boundary_size_y_lower     = 3
CoordBase::boundary_size_z_lower     = 3
CoordBase::boundary_size_x_upper     = 3
CoordBase::boundary_size_y_upper     = 3
CoordBase::boundary_size_z_upper     = 3

CoordBase::boundary_shiftout_x_lower = 1
CoordBase::boundary_shiftout_z_lower = 1

CartGrid3D::type = "coordbase"

ReflectionSymmetry::reflection_z   = yes
ReflectionSymmetry::avoid_origin_z = no



ActiveThorns = "SphericalSurface"

SphericalSurface::nsurfaces = 5
SphericalSurface::maxntheta = 39
SphericalSurface::maxnphi   = 76

SphericalSurface::ntheta      [0] = 39
SphericalSurface::nphi        [0] = 76
SphericalSurface::nghoststheta[0] = 2
SphericalSurface::nghostsphi  [0] = 2

SphericalSurface::ntheta      [1] = 39
SphericalSurface::nphi        [1] = 76
SphericalSurface::nghoststheta[1] = 2
SphericalSurface::nghostsphi  [1] = 2

SphericalSurface::ntheta      [2] = 39
SphericalSurface::nphi        [2] = 76
SphericalSurface::nghoststheta[2] = 2
SphericalSurface::nghostsphi  [2] = 2

SphericalSurface::ntheta      [3] = 39
SphericalSurface::nphi        [3] = 76
SphericalSurface::nghoststheta[3] = 2
SphericalSurface::nghostsphi  [3] = 2

SphericalSurface::ntheta      [4] = 39
SphericalSurface::nphi        [4] = 76
SphericalSurface::nghoststheta[4] = 2
SphericalSurface::nghostsphi  [4] = 2



ActiveThorns = "CarpetMask"

CarpetMask::verbose = no

CarpetMask::excluded_surface       [0] = 3
```

```
174 CarpetMask::excluded_surface_factor[0] = 1.0
175
176 CarpetMask::excluded_surface       [1] = 4
177 CarpetMask::excluded_surface_factor[1] = 1.0
178
179
180
181 ActiveThorns = "ADMBase ADMCoupling ADMMacros CoordGauge SpaceMask
        StaticConformal TmunuBase"
182 ActiveThorns = "CarpetRegrid2 PunctureTracker CarpetTracker"
183
184 CarpetTracker::surface[0] = 0
185 CarpetTracker::surface[1] = 1
186 PunctureTracker::track                      [0] = yes
187 PunctureTracker::initial_x                  [0] = 3.0
188 PunctureTracker::which_surface_to_store_info[0] = 0
189 PunctureTracker::track                      [1] = yes
190 PunctureTracker::initial_x                  [1] = -3.0
191 PunctureTracker::which_surface_to_store_info[1] = 1
192
193
194 CarpetRegrid2::regrid_every            = 16
195 CarpetRegrid2::freeze_unaligned_levels = yes
196 CarpetRegrid2::symmetry_rotating180    = yes
197 CarpetRegrid2::verbose                 = yes
198
199 CarpetRegrid2::num_centres = 2
200
201 CarpetRegrid2::num_levels_1         =  7
202 CarpetRegrid2::position_x_1         = +3.0
203 CarpetRegrid2::radius_1[ 1]         =  64.0
204 CarpetRegrid2::radius_1[ 2]         =  16.0
205 CarpetRegrid2::radius_1[ 3]         =   8.0
206 CarpetRegrid2::radius_1[ 4]         =   4.0
207 CarpetRegrid2::radius_1[ 5]         =   2.0
208 CarpetRegrid2::radius_1[ 6]         =   1.0
209 CarpetRegrid2::movement_threshold_1 =   0.16
210
211 CarpetRegrid2::num_levels_2         =  7
212 CarpetRegrid2::position_x_2         = -3.0
213 CarpetRegrid2::radius_2[ 1]         =  64.0
214 CarpetRegrid2::radius_2[ 2]         =  16.0
215 CarpetRegrid2::radius_2[ 3]         =   8.0
216 CarpetRegrid2::radius_2[ 4]         =   4.0
217 CarpetRegrid2::radius_2[ 5]         =   2.0
218 CarpetRegrid2::radius_2[ 6]         =   1.0
219 CarpetRegrid2::movement_threshold_2 =   0.16
220
221
222 ActiveThorns = "MoL Time"
223
224 MoL::ODE_Method             = "RK4"
225 MoL::MoL_Intermediate_Steps = 4
226 MoL::MoL_Num_Scratch_Levels = 1
```

```
227
228 Carpet::time_refinement_factors = "[1, 1, 2, 4, 8, 16, 32, 64, 128,
    256]"
229
230 Time::dtfac = 0.25
231
232
233 ADMMacros::spatial_order = 4
234
235
236
237 ActiveThorns = "TwoPunctures"
238
239 ADMBase::metric_type = "physical"
240
241 ADMBase::initial_data    = "twopunctures"
242 ADMBase::initial_lapse   = "twopunctures-averaged"
243 ADMBase::initial_shift   = "zero"
244 ADMBase::initial_dtlapse = "zero"
245 ADMBase::initial_dtshift = "zero"
246
247 TwoPunctures::par_b          =  3.0
248 TwoPunctures::par_m_plus     =  0.47656
249 TwoPunctures::par_m_minus    =  0.47656
250 TwoPunctures::par_P_plus [1] = +0.13808
251 TwoPunctures::par_P_minus[1] = -0.13808
252
253 #TODO# TwoPunctures::grid_setup_method = "evaluation"
254
255 TwoPunctures::TP_epsilon = 1.0e-6
256 TwoPunctures::TP_Tiny    = 1.0e-2
257
258 TwoPunctures::verbose = yes
259
260
261
262 ActiveThorns = "ML_BSSN ML_BSSN_Helper NewRad"
263
264 ADMBase::evolution_method         = "ML_BSSN"
265 ADMBase::lapse_evolution_method   = "ML_BSSN"
266 ADMBase::shift_evolution_method   = "ML_BSSN"
267 ADMBase::dtlapse_evolution_method = "ML_BSSN"
268 ADMBase::dtshift_evolution_method = "ML_BSSN"
269
270 ML_BSSN::harmonicN           = 1      # 1+log
271 ML_BSSN::harmonicF           = 2.0    # 1+log
272 ML_BSSN::ShiftGammaCoeff     = 0.75
273 ML_BSSN::BetaDriver          = 1.0
274 ML_BSSN::LapseAdvectionCoeff = 1.0
275 ML_BSSN::ShiftAdvectionCoeff = 1.0
276
277 ML_BSSN::MinimumLapse        = 1.0e-8
278
279 ML_BSSN::my_initial_boundary_condition = "extrapolate-gammas"
```

```
ML_BSSN::my_rhs_boundary_condition     = "NewRad"
Boundary::radpower                     = 2

ML_BSSN::ML_log_confac_bound = "none"
ML_BSSN::ML_metric_bound     = "none"
ML_BSSN::ML_Gamma_bound      = "none"
ML_BSSN::ML_trace_curv_bound = "none"
ML_BSSN::ML_curv_bound       = "none"
ML_BSSN::ML_lapse_bound      = "none"
ML_BSSN::ML_dtlapse_bound    = "none"
ML_BSSN::ML_shift_bound      = "none"
ML_BSSN::ML_dtshift_bound    = "none"



ActiveThorns = "Dissipation"

Dissipation::order = 5
Dissipation::vars  = "
        ML_BSSN::ML_log_confac
        ML_BSSN::ML_metric
        ML_BSSN::ML_trace_curv
        ML_BSSN::ML_curv
        ML_BSSN::ML_Gamma
        ML_BSSN::ML_lapse
        ML_BSSN::ML_shift
        ML_BSSN::ML_dtlapse
        ML_BSSN::ML_dtshift
"



ActiveThorns = "ML_ADMConstraints"



ActiveThorns = "AHFinderDirect"

AHFinderDirect::find_every = 16

AHFinderDirect::run_at_CCTK_ANALYSIS = yes
AHFinderDirect::run_at_CCTK_POSTSTEP = false

AHFinderDirect::move_origins            = yes

AHFinderDirect::geometry_interpolator_name = "Lagrange polynomial
    interpolation"
AHFinderDirect::geometry_interpolator_pars = "order=4"
AHFinderDirect::surface_interpolator_name  = "Lagrange polynomial
    interpolation"
AHFinderDirect::surface_interpolator_pars  = "order=4"

AHFinderDirect::output_h_every = 0

```

```
AHFinderDirect::N_horizons = 3

AHFinderDirect::origin_x                                [1] = +3.0
AHFinderDirect::initial_guess__coord_sphere__x_center[1] = +3.0
AHFinderDirect::initial_guess__coord_sphere__radius  [1] =  0.25
AHFinderDirect::which_surface_to_store_info          [1] = 2
AHFinderDirect::set_mask_for_individual_horizon      [1] = no
AHFinderDirect::reset_horizon_after_not_finding      [1] = no
AHFinderDirect::track_origin_from_grid_scalar        [1] = yes
AHFinderDirect::track_origin_source_x                [1] = "
    PunctureTracker::pt_loc_x[0]"
AHFinderDirect::track_origin_source_y                [1] = "
    PunctureTracker::pt_loc_y[0]"
AHFinderDirect::track_origin_source_z                [1] = "
    PunctureTracker::pt_loc_z[0]"
AHFinderDirect::max_allowable_horizon_radius         [1] = 3

AHFinderDirect::origin_x                                [2] = -3.0
AHFinderDirect::initial_guess__coord_sphere__x_center[2] = -3.0
AHFinderDirect::initial_guess__coord_sphere__radius  [2] =  0.25
AHFinderDirect::which_surface_to_store_info          [2] = 3
AHFinderDirect::set_mask_for_individual_horizon      [2] = no
AHFinderDirect::reset_horizon_after_not_finding      [2] = no
AHFinderDirect::track_origin_from_grid_scalar        [2] = yes
AHFinderDirect::track_origin_source_x                [2] = "
    PunctureTracker::pt_loc_x[1]"
AHFinderDirect::track_origin_source_y                [2] = "
    PunctureTracker::pt_loc_y[1]"
AHFinderDirect::track_origin_source_z                [2] = "
    PunctureTracker::pt_loc_z[1]"
AHFinderDirect::max_allowable_horizon_radius         [2] = 3

AHFinderDirect::origin_x                                [3] = 0
AHFinderDirect::find_after_individual_time           [3] = 100.0
AHFinderDirect::initial_guess__coord_sphere__x_center[3] = 0
AHFinderDirect::initial_guess__coord_sphere__radius  [3] = 1.0
AHFinderDirect::which_surface_to_store_info          [3] = 4
AHFinderDirect::reset_horizon_after_not_finding      [3] = no
AHFinderDirect::max_allowable_horizon_radius         [3] = 6


ActiveThorns = "WeylScal4 Multipole"

Multipole::nradii       = 4
Multipole::radius[0]    = 30
Multipole::radius[1]    = 40
Multipole::radius[2]    = 50
Multipole::radius[3]    = 60
Multipole::ntheta       = 60
Multipole::nphi         = 120
Multipole::variables    = "WeylScal4::Psi4r{sw=-2 cmplx='WeylScal4::
    Psi4i' name='psi4'}"
Multipole::out_every    = 4
Multipole::l_max       = 8
```

```
379
380
381 ActiveThorns = "CarpetIOBasic"
382
383 IOBasic::outInfo_every      = 1
384 IOBasic::outInfo_reductions = "norm2"
385 IOBasic::outInfo_vars       = "
386         Carpet::physical_time_per_hour
387         ML_BSSN::ML_trace_curv
388 "
389
390
391
392 ActiveThorns = "CarpetIOScalar"
393
394 IOScalar::one_file_per_group = yes
395
396 IOScalar::outScalar_every = 16
397 IOScalar::outScalar_vars  = "
398         CarpetReduce::weight
399         ADMBase::metric
400         ADMBase::curv
401         ADMBase::lapse
402         ADMBase::shift
403         ADMBase::dtlapse
404         ADMBase::dtshift
405         ML_ADMConstraints::ML_Ham
406         ML_ADMConstraints::ML_mom
407         SphericalSurface::sf_radius
408 "
409
410
411
412 ActiveThorns = "CarpetIOASCII"
413
414 IOASCII::one_file_per_group = yes
415
416 IOASCII::output_symmetry_points = no
417 IOASCII::out3D_ghosts           = no
418
419 IOASCII::out0D_every = 16
420 IOASCII::out0D_vars  = "
421         Carpet::timing
422         CarpetReduce::weight
423         ADMBase::metric
424         ADMBase::curv
425         ADMBase::lapse
426         ADMBase::shift
427         ADMBase::dtlapse
428         ADMBase::dtshift
429         ML_ADMConstraints::ML_Ham
430         ML_ADMConstraints::ML_mom
431         SphericalSurface::sf_active
432         SphericalSurface::sf_valid
```

```
        SphericalSurface::sf_info
        SphericalSurface::sf_radius
        SphericalSurface::sf_origin
        SphericalSurface::sf_coordinate_descriptors
        PunctureTracker::pt_loc
"

IOASCII::out1D_every = 16
IOASCII::out1D_vars  = "
        CarpetReduce::weight
        ADMBase::metric
        ADMBase::curv
        ADMBase::lapse
        ADMBase::shift
        ADMBase::dtlapse
        ADMBase::dtshift
        ML_ADMConstraints::ML_Ham
        ML_ADMConstraints::ML_mom
        SphericalSurface::sf_radius
"

IOASCII::out2D_every = 16
IOASCII::out2D_vars  = "
        SphericalSurface::sf_radius
"



Activethorns = "CarpetIOHDF5"

# 3D HDF5 Output
CarpetIOHDF5::out3D_every = 2048
carpetIOHDF5::out3D_vars = "
        ADMBase::lapse
        ADMBase::shift
        ADMBase::metric
        ADMBase::curv
"


#IOHDF5::out_every              = 64
#IOHDF5::one_file_per_group     = yes
#IOHDF5::output_symmetry_points = no
#IOHDF5::out3D_ghosts           = no
#IOHDF5::compression_level      = 1
#IOHDF5::use_checksums          = yes
#IOHDF5::out_vars               = "
#        CarpetReduce::weight
#        ADMBase::metric
#        ADMBase::curv
#        ADMBase::lapse
#        ADMBase::shift
#        ML_ADMConstraints::ML_Ham
#        ML_ADMConstraints::ML_mom
```

```
#         WeylScal4::Psi4r
#         WeylScal4::Psi4i
#"

IOHDF5::checkpoint                  = yes
IO::checkpoint_dir                  = $parfile
IO::checkpoint_ID                   = yes
IO::checkpoint_every_walltime_hours = 6.0
IO::checkpoint_on_terminate         = yes

IO::recover     = "autoprobe"
IO::recover_dir = $parfile



ActiveThorns = "Formaline"



ActiveThorns = "TimerReport"

TimerReport::out_every                  = 640
TimerReport::out_filename               = "TimerReport"
TimerReport::output_all_timers_together = yes
TimerReport::output_all_timers_readable = yes
TimerReport::n_top_timers               = 20
```

D.3 Single Black Hole Two Puncture

```
ActiveThorns = "
        Boundary
        CartGrid3D
        CoordBase
        IOUtil
        InitBase
        MoL
  Time
        SymBase

        ADMBase
        ADMCoupling
        ADMMacros
        CoordGauge
        SpaceMask
        StaticConformal

        NaNChecker

        Carpet
        CarpetIOASCII
        CarpetIOBasic
        CarpetIOScalar
        CarpetLib
```

```
        CarpetReduce
  CarpetRegrid2
        CarpetInterp
        LoopControl

        ReflectionSymmetry
        RotatingSymmetry180

        TwoPunctures
        Formaline
        GSL

        Slab

        GenericFD

        ML_BSSN
        ML_BSSN_Helper
  SphericalSurface
  TmunuBase
  Dissipation

        CarpetIOHDF5

        AHFinderDirect

        AEILocalInterp LocalReduce

        QuasiLocalMeasures

        WeylScal4 MultiPole
"
#        NoExcision LocalInterp SummationByParts

# output

IO::out_dir = $parfile

IOBasic::outInfo_every = 1
IOBasic::outInfo_vars = "
        ML_BSSN::H
        ML_BSSN::trK
"

IOScalar::one_file_per_group = yes
IOScalar::outScalar_every    = 64
IOScalar::outScalar_vars     = "
        ML_BSSN::ML_Ham
        ML_BSSN::ML_mom
        ML_BSSN::ML_cons_detg
        ML_BSSN::ML_cons_Gamma
        ML_BSSN::ML_cons_traceA
"

```

```
IOASCII::one_file_per_group = yes
IOASCII::out0D_every        = 60
IOASCII::out0D_vars         = "
        QuasiLocalMeasures::qlm_scalars
        QuasiLocalMeasures::qlm_multipole_moments
"

IOASCII::out1D_every        = 64
IOASCII::out1D_vars         = "
        ADMBase::metric
        ADMBase::curv
        ADMBase::lapse
        ADMBase::shift
        ML_BSSN::ML_log_confac
        ML_BSSN::ML_metric
        ML_BSSN::ML_trace_curv
        ML_BSSN::ML_curv
        ML_BSSN::ML_Gamma
        ML_BSSN::ML_lapse
        ML_BSSN::ML_shift
        ML_BSSN::ML_Ham
        ML_BSSN::ML_mom
        ML_BSSN::ML_cons_detg
        ML_BSSN::ML_cons_Gamma
        ML_BSSN::ML_cons_traceA
"
Activethorns = "CarpetIOHDF5"

# 3D HDF5 Output
 CarpetIOHDF5::out3D_every = 2048
carpetIOHDF5::out3D_vars = "
         ADMBase::lapse
         ADMBase::shift
         ADMBase::metric
         ADMBase::curv
"
### Checkpointing

CarpetIOHDF5::checkpoint                = yes
IO::checkpoint_ID                       = no
IO::recover                             = "autoprobe"
IO::checkpoint_every                    = 512
IO::out_proc_every                      = 1
IO::checkpoint_keep                     = 2
IO::checkpoint_dir                      = $parfile
IO::recover_dir                         = $parfile
Carpet::regrid_during_recovery          = no
CarpetIOHDF5::use_grid_structure_from_checkpoint = yes

#--- driver

Cactus::cctk_itlast = 0
Cactus::terminate = time
Cactus::cctk_final_time = 1
```

```
133 Carpet::use_buffer_zones      = yes
134 Carpet::use_tapered_grids     = yes
135
136 CarpetLib::interleave_communications = yes
137 CarpetLib::combine_sends              = yes
138 CarpetLib::print_memstats_every      = 1024
139 #Carpet::regrid_in_level_mode         = no
140 Carpet::output_timers_every          = 1024
141 CarpetLib::print_timestats_every     = 1024
142 Carpet::print_timestats_every        = 1
143 #Carpet::init_each_timelevel          = yes
144
145
146 carpet::verbose = no
147 carpet::veryverbose = no
148
149 #--- Carpet prolongation order settings
150
151 Carpet::prolongation_order_space                = 5
152 Carpet::prolongation_order_time                 = 1
153
154
155 # MoL time integration
156
157 MoL::ODE_Method = RK4
158 Carpet::num_integrator_substeps = 4
159 MoL::MoL_Intermediate_Steps       =   4
160 MoL::MoL_Num_Scratch_Levels     = 1
161 time::dtfac                                =   0.25
162
163 #--- grid and symmetries
164
165 driver::ghost_size            = 3
166 grid::type = "CoordBase"
167 CoordBase::domainsize = "minmax"
168 CoordBase::xmax = 258.048
169 CoordBase::ymax = 258.048
170 CoordBase::zmax = 258.048
171 CoordBase::xmin =   0.000
172 CoordBase::ymin =-258.048
173 CoordBase::zmin =   0.000
174 CoordBase::dx   =   3.072
175 CoordBase::dy   =   3.072
176 CoordBase::dz   =   3.072
177
178
179 ReflectionSymmetry::reflection_x   = no
180 ReflectionSymmetry::reflection_y   = no
181 ReflectionSymmetry::reflection_z   = yes
182 ReflectionSymmetry::avoid_origin_x = no
183 ReflectionSymmetry::avoid_origin_y = no
184 ReflectionSymmetry::avoid_origin_z = no
185 CarpetRegrid2::symmetry_rotating180     = yes
186
```

```
CoordBase::boundary_shiftout_x_lower = 1
CoordBase::boundary_shiftout_z_lower = 1

Carpet::domain_from_coordbase = yes

CoordBase::boundary_size_x_lower = 3
CoordBase::boundary_size_y_lower = 3
CoordBase::boundary_size_z_lower = 3
CoordBase::boundary_size_x_upper = 3
CoordBase::boundary_size_y_upper = 3
CoordBase::boundary_size_z_upper = 3


carpet::max_refinement_levels             = 8
CarpetRegrid2::num_centres              = 1
CarpetRegrid2::num_levels_1             = 8
CarpetRegrid2::position_x_1             = +0.0
CarpetRegrid2::radius_1             [1] = 128.0 # 1.536
CarpetRegrid2::radius_1             [2] =  64.0 # 0.768
CarpetRegrid2::radius_1             [3] =  16.0 # 0.384
CarpetRegrid2::radius_1             [4] =   8.0 # 0.192
CarpetRegrid2::radius_1             [5] =   4.0 # 0.096
CarpetRegrid2::radius_1             [6] =   2.0 # 0.048
CarpetRegrid2::radius_1             [7] =   1.0 # 0.024


#--- initial data

ADMBase::initial_data      = "twopunctures"
ADMBase::initial_lapse     = "twopunctures-averaged"
ADMBase::initial_shift     = "zero"
ADMBase::initial_dtlapse   = "zero"
ADMBase::initial_dtshift   = "zero"
ADMBase::lapse_timelevels  = 2
ADMBase::shift_timelevels  = 2
ADMBase::metric_timelevels = 2
InitBase::initial_data_setup_method     = init_some_levels
TwoPunctures::par_b                     = 1.0
TwoPunctures::center_offset[0]          = -1.0
TwoPunctures::par_m_plus                = 0.75174408
TwoPunctures::par_m_minus               = 0.0
TwoPunctures::par_S_plus[2]             = 0.7
TwoPunctures::TP_epsilon                = 1e-6
TwoPunctures::grid_setup_method         = evaluation
TwoPunctures::verbose                   = yes
TwoPunctures::do_residuum_debug_output  = yes
TwoPunctures::do_initial_debug_output   = yes
TwoPunctures::npoints_A                 = 40
TwoPunctures::npoints_B                 = 40
TwoPunctures::npoints_phi               = 16
#Carpet::init_each_timelevel             = yes
#Carpet::init_3_timelevels               = yes
Carpet::init_fill_timelevels             = yes

```

```
241
242
243 ##--- NoExcision
244 #
245 #NoExcision::num_regions = 1
246 #NoExcision::method = new
247 #NoExcision::smooth_regions = yes
248 #NoExcision::use_user_regions = yes
249 #NoExcision::centre_x            [0] = +0.0
250 #NoExcision::radius              [0] =  0.001
251 #
252 #NoExcision::smoothing_order         = 6
253 #NoExcision::smoothing_eps           = 1e-5
254 #NoExcision::verbose                 = yes
255
256 #--- ML_BSSN
257
258 ADMBase::evolution_method = "ML_BSSN"
259 ADMBase::lapse_evolution_method = "ML_BSSN"
260 ADMBase::shift_evolution_method = "ML_BSSN"
261
262 ML_BSSN::timelevels = 2
263
264 ML_BSSN::harmonicN       = 1      # 1+log
265 ML_BSSN::harmonicF       = 2.0    # 1+log
266 ML_BSSN::ShiftGammaCoeff = 0.75
267 ML_BSSN::BetaDriver      = 1.0
268 ML_BSSN::LapseAdvectionCoeff = 1.0
269 ML_BSSN::ShiftAdvectionCoeff = 1.0
270
271 #ML_BSSN::ML_log_confac_bound = "radiative"
272 #ML_BSSN::ML_metric_bound     = "radiative"
273 #ML_BSSN::ML_Gamma_bound      = "radiative"
274 #ML_BSSN::ML_trace_curv_bound = "radiative"
275 #ML_BSSN::ML_curv_bound       = "radiative"
276 #ML_BSSN::ML_lapse_bound      = "radiative"
277 #ML_BSSN::ML_dtlapse_bound    = "radiative"
278 #ML_BSSN::ML_shift_bound      = "radiative"
279 #ML_BSSN::ML_dtshift_bound    = "radiative"
280
281 ML_BSSN::my_boundary_condition = "Minkowski"
282
283 ML_BSSN::ML_log_confac_bound = "none"
284 ML_BSSN::ML_metric_bound     = "none"
285 ML_BSSN::ML_Gamma_bound      = "none"
286 ML_BSSN::ML_trace_curv_bound = "none"
287 ML_BSSN::ML_curv_bound       = "none"
288 ML_BSSN::ML_lapse_bound      = "none"
289 ML_BSSN::ML_dtlapse_bound    = "none"
290 ML_BSSN::ML_shift_bound      = "none"
291 ML_BSSN::ML_dtshift_bound    = "none"
292
293 ADMMacros::spatial_order = 4
294
```

```
#--- Dissipation

Dissipation::vars = "
  ML_BSSN::ML_Gamma
    ML_BSSN::ML_lapse
    ML_BSSN::ML_shift
    ML_BSSN::ML_log_confac
    ML_BSSN::ML_metric
    ML_BSSN::ML_trace_curv
    ML_BSSN::ML_curv
"
Dissipation::order = 5

### Horizons

AHFinderDirect::N_horizons                                = 1
AHFinderDirect::find_every                                = 12
AHFinderDirect::output_h_every                            = 0
AHFinderDirect::max_Newton_iterations__initial            = 50
AHFinderDirect::max_Newton_iterations__subsequent         = 50
AHFinderDirect::max_allowable_Theta_growth_iterations     = 10
AHFinderDirect::max_allowable_Theta_nonshrink_iterations = 10
#AHFinderDirect::max_N_zones_per_right_angle               = 36
#AHFinderDirect::N_zones_per_right_angle[1]                = 36
AHFinderDirect::geometry_interpolator_name                = "Lagrange
    polynomial interpolation"
AHFinderDirect::geometry_interpolator_pars                = "order=4"
AHFinderDirect::surface_interpolator_name                 = "Lagrange
    polynomial interpolation"
AHFinderDirect::surface_interpolator_pars                 = "order=4"
#AHFinderDirect::verbose_level                             = "physics
    details"
AHFinderDirect::verbose_level                             = "algorithm
    highlights"
AHFinderDirect::Jacobian_store_solve_method               = "row-oriented
     sparse matrix/ILUCG"
AHFinderDirect::move_origins                              = yes

AHFinderDirect::origin_x                                  [1] = 0.0
AHFinderDirect::initial_guess__coord_sphere__x_center     [1] = 0.0
AHFinderDirect::initial_guess__coord_sphere__radius       [1] = 0.5
AHFinderDirect::which_surface_to_store_info               [1] = 0
AHFinderDirect::set_mask_for_individual_horizon           [1] = no
AHFinderDirect::reset_horizon_after_not_finding           [1] = no
#AHFinderDirect::dont_find_after_individual                [1] = 3328


# Horizon surfaces

SphericalSurface::nsurfaces             = 1
SphericalSurface::maxntheta             = 73
SphericalSurface::maxnphi               = 144

SphericalSurface::ntheta            [0] = 73
```

```
SphericalSurface::nphi               [0] = 144
SphericalSurface::nghoststheta       [0] = 2
SphericalSurface::nghostsphi         [0] = 2

WeylScal4::fd_order = "4th"

Multipole::integration_method = "Simpson"
Multipole::interpolator_name = "Lagrange polynomial interpolation"
Multipole::interpolator_pars = "order=4 boundary_off_centering_tolerance
    ={0.0 0.0 0.0 0.0 0.0 0.0} boundary_extrapolation_tolerance={0.0 0.0
    0.0 0.0 0.0 0.0}"
Multipole::nradii       = 4
Multipole::radius[0]    = 30
Multipole::radius[1]    = 40
Multipole::radius[2]    = 50
Multipole::radius[3]    = 60
Multipole::ntheta       = 72
Multipole::nphi         = 144
Multipole::variables    = "WeylScal4::Psi4r{sw=-2 cmplx='WeylScal4::
    Psi4i' name='psi4'}"
Multipole::out_every    = 64
Multipole::l_max       = 8


### QuasiLocalMeasures

QuasiLocalMeasures::verbose                = yes
#IsolatedHorizon::veryverbose            = no
QuasiLocalMeasures::interpolator           = "Lagrange polynomial
    interpolation"
QuasiLocalMeasures::interpolator_options   = "order=4"
QuasiLocalMeasures::spatial_order          = 4
QuasiLocalMeasures::num_surfaces           = 1
QuasiLocalMeasures::surface_index      [0] = 0

#--- Analysis

nanchecker::check_every                          =       1
nanchecker::check_vars                           =       "
        ML_BSSN::ML_Gamma
        ML_BSSN::ML_lapse
        ML_BSSN::ML_shift
        ML_BSSN::ML_log_confac
        ML_BSSN::ML_metric
        ML_BSSN::ML_trace_curv
        ML_BSSN::ML_curv
"
nanchecker::action_if_found                      =       "terminate"
```

Appendix E

GYOTO Scripts

E.1 Fixed Star

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Scenery>

  <Metric kind = "NumericalMetricLorene">
    <MapAf/>
    <Horizon>0.51</Horizon>
    <File>/home/dulte/Documents/Skole/Master/Gyoto_files/Metrics/</File>
    <AxisymCirc/>
  </Metric>


  <!-- Uncomment this if you want to use an analytical Kerr metric -->
  <!-- <Metric kind = "KerrBL"> -->
  <!--   <Spin>0.</Spin> -->
  <!-- </Metric> -->

  <Integ31/>

  <Screen>
    <Position>1000. 250. 1.483 0.</Position>
    <Time unit="geometrical_time">1000.</Time>
    <FieldOfView> 0.05 </FieldOfView>
    <Resolution>30</Resolution>
  </Screen>

  <Quantities>Intensity</Quantities>

  <Astrobj kind = "FixedStar">
    <Radius> 3.972 </Radius> 10. 8.972
    <Position> 0 0 0 </Position>
    <Spectrum kind="PowerLaw">
      <Exponent> 0 </Exponent>
      <Constant> 1. </Constant>
    </Spectrum>
    <OpticallyThin/>
```

```
    <RMax>0.</RMax>
  </Astrobj>

  <MinimumTime> -1e4 </MinimumTime>
  <NThreads> 1 </NThreads>

</Scenery>
```

E.2 Page-Thorne Disk

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Scenery>
  <Metric kind = "NumericalMetricLorene">
    <MapAf/>
    <Horizon>0.51</Horizon>
    <File>/home/dulte/Documents/Skole/Master/Gyoto_files/Metrics/</File>
    <!--<File>/data/fvincent/BinData/Kerr/Metric/</File>-->
    <!--<AxisymCirc/>-->
    <BosonStarCircular/>
  </Metric>



  <Screen>
    <Position>1000. 100. 1.483 0.</Position>
    <Time unit="geometrical_time">1000.</Time>
    <FieldOfView>
      0.31415926535897932384626433832795028841971 6
    </FieldOfView>
    <Resolution>
      200
    </Resolution>
  </Screen>



  <Astrobj kind = "PageThorneDisk">
    <Bolometric/>
  </Astrobj>
  <MinimumTime> -1000. </MinimumTime>
  <!--Quantities> User4 </Quantities-->
  Bolometric emission, this is the default (and only intensity) for
  PageThorneDisk.
</Scenery>
```

Appendix F

Metric for Use in GYOTO with no Spherical Symmetry

Using Lorene data with Gyoto

1 Metric coefficients in quasi-isotropic coordinates in a circular spacetime

Lorene is using quasi-isotropic spherical coordinates (t, r, θ, φ) in a circular spacetime. This means that we assume (1) $g_{tr} = g_{t\theta} = g_{\varphi r} = g_{\varphi\theta} = 0$ and (2) $g_{r\theta} = 0$ as well as $g_{\theta\theta} = r^2 g_{rr}$. The line element is then

$$\mathrm{d}s^2 = -N^2\mathrm{d}t^2 + A^2\left(\mathrm{d}r^2 + r^2\mathrm{d}\theta^2\right) + B^2r^2\sin^2\theta\left(\mathrm{d}\varphi + \beta^\varphi\mathrm{d}t\right)^2 \tag{1}$$

where N is the lapse, A, B and $\omega = -g_{t\varphi}/g_{\varphi\varphi}$ are scalar functions of r and θ. The shift vector is

$$\boldsymbol{\beta} = (0, 0, -\omega). \tag{2}$$

The 4D metric and inverse metric are

$$g_{\alpha\beta} = \begin{pmatrix} -N^2 + \omega^2B^2r^2\sin^2\theta & 0 & 0 & -\omega B^2r^2\sin^2\theta \\ 0 & A^2 & 0 & 0 \\ 0 & 0 & A^2r^2 & 0 \\ -\omega B^2r^2\sin^2\theta & 0 & 0 & B^2r^2\sin^2\theta \end{pmatrix} \tag{3}$$

$$g^{\alpha\beta} = \begin{pmatrix} -N^{-2} & 0 & 0 & -\omega/N^2 \\ 0 & A^{-2} & 0 & 0 \\ 0 & 0 & A^{-2}r^{-2} & 0 \\ -\omega/N^2 & 0 & 0 & -\omega^2/N^2 + (Br\sin\theta)^{-2} \end{pmatrix} \tag{4}$$

and the induced 3-metric reads

$$\begin{aligned} \gamma_{ij} &= \mathrm{diag}(\mathrm{A}^2, \mathrm{A}^2\mathrm{r}^2, \mathrm{B}^2\mathrm{r}^2\sin^2\theta), \\ \gamma^{ij} &= \mathrm{diag}(\frac{1}{\mathrm{A}^2}, \frac{1}{\mathrm{A}^2\mathrm{r}^2}, \frac{1}{\mathrm{B}^2\mathrm{r}^2\sin^2\theta}) \end{aligned} \tag{5}$$

with thus $g_{ij} = \gamma_{ij}$ but $g^{33} \neq \gamma^{33}$. These expressions are valid in the "Gyoto basis", $(\boldsymbol{\partial_r}, \boldsymbol{\partial_\theta}, \boldsymbol{\partial_\varphi})$. The corresponding expressions in the "Lorene basis" $(\boldsymbol{e_r}, \boldsymbol{e_\theta}, \boldsymbol{e_\varphi})$ are given later.

A note on dimensions here. We have $\mathrm{d}s^2 = g_{\mu\nu}\mathrm{d}x^\mu\mathrm{d}x^\nu$ which is sufficient to determine the dimensions of $g_{\mu\nu}$ and thus of the 3+1 metric quantities. For instance, from Eq. 1, $L^2 \approx N^2c^2\mathrm{d}t^2$, where c is used here explicitly and L is a length dimension. Thus N is dimensionless. We also have $L^2 \approx B^2r^2\left(\beta^\varphi\right)^2c^2\mathrm{d}t^2$ which

makes sense if B is dimensionless (just as A) and if β^φ is homogeneous to L^{-1} (actually to an inverse time, which is the same as L^{-1} in $c = 1$ units). To recap

$$\begin{aligned} N \approx A \approx B \approx 1, \\ \beta^\varphi \approx \text{length}^{-1} \end{aligned} \tag{6}$$

so β^φ should be given in units of M^{-1}.

Due to the symmetries only two components of the extrinsic curvature are non-zero, $K_{r\varphi}$ and $K_{\theta\varphi}$.

The 4D Christoffels are, for a *general 3+1 spacetime*

$$\begin{aligned}
{}^4\Gamma^0{}_{00} &= \frac{1}{N}\left(\frac{\partial N}{\partial t} + \beta^j\partial_j N - K_{jk}\beta^j\beta^k\right) \qquad (7)\\
{}^4\Gamma^0{}_{0j} &= \frac{1}{N}\left(\partial_j N - K_{jk}\beta^k\right)\\
{}^4\Gamma^0{}_{jk} &= -\frac{1}{N}K_{jk}\\
{}^4\Gamma^i{}_{00} &= N\gamma^{ij}\partial_j N - 2NK^i{}_j\beta^j + \frac{\beta^i}{N}\left(K_{jk}\beta^j\beta^k - \frac{\partial N}{\partial t} - \beta^j\partial_j N\right) + \frac{\partial\beta^i}{\partial t} + \beta^j D_j\beta^i\\
{}^4\Gamma^i{}_{0j} &= D_j\beta^i - NK^i{}_j + \frac{\beta^i}{N}\left(K_{jk}\beta^k - \partial_j N\right)\\
{}^4\Gamma^i{}_{jk} &= {}^3\Gamma^i{}_{jk} + \frac{\beta^i}{N}K_{jk}
\end{aligned}$$

where

$${}^3\Gamma^i{}_{jk} = \frac{1}{2}\gamma^{is}\left(\gamma_{sk,j} + \gamma_{sj,k} - \gamma_{jk,s}\right) \tag{8}$$

and

$$D_j\beta^i = \partial_j\beta^i + {}^3\Gamma^i{}_{jk}\beta^k. \tag{9}$$

For a *circular spacetime* the non-zero Christoffels are

$$
\begin{aligned}
{}^4\Gamma^t{}_{tt} &= \frac{1}{N}\frac{\partial N}{\partial t} \qquad (10)\\
{}^4\Gamma^t{}_{tr} &= \frac{1}{N}\left(\partial_r N - K_{r\varphi}\beta^\varphi\right)\\
{}^4\Gamma^t{}_{t\theta} &= \frac{1}{N}\left(\partial_\theta N - K_{\theta\varphi}\beta^\varphi\right)\\
{}^4\Gamma^t{}_{r\varphi} &= -\frac{1}{N}K_{r\varphi}\\
{}^4\Gamma^t{}_{\theta\varphi} &= -\frac{1}{N}K_{\theta\varphi}\\
{}^4\Gamma^r{}_{tt} &= N\gamma^{rr}\left(\partial_r N - 2K_{r\varphi}\beta^\varphi - \frac{\left(\beta^\varphi\right)^2}{2N}\partial_r\gamma_{\varphi\varphi}\right)\\
{}^4\Gamma^\theta{}_{tt} &= N\gamma^{\theta\theta}\left(\partial_\theta N - 2K_{\theta\varphi}\beta^\varphi - \frac{\left(\beta^\varphi\right)^2}{2N}\partial_\theta\gamma_{\varphi\varphi}\right)\\
{}^4\Gamma^\varphi{}_{tt} &= -\frac{\beta^\varphi}{N}\frac{\partial N}{\partial t} + \frac{\partial\beta^\varphi}{\partial t}\\
{}^4\Gamma^r{}_{t\varphi} &= -\gamma^{rr}\left(NK_{r\varphi} + \frac{1}{2}\beta^\varphi\partial_r\gamma_{\varphi\varphi}\right)\\
{}^4\Gamma^\theta{}_{t\varphi} &= -\gamma^{\theta\theta}\left(NK_{\theta\varphi} + \frac{1}{2}\beta^\varphi\partial_\theta\gamma_{\varphi\varphi}\right)\\
{}^4\Gamma^\varphi{}_{tr} &= \partial_r\beta^\varphi + \frac{1}{2}\gamma^{\varphi\varphi}\partial_r\gamma_{\varphi\varphi}\beta^\varphi - N\gamma^{\varphi\varphi}K_{r\varphi} + \frac{\beta^\varphi}{N}\left(K_{r\varphi}\beta^\varphi - \partial_r N\right)\\
{}^4\Gamma^\varphi{}_{t\theta} &= \partial_\theta\beta^\varphi + \frac{1}{2}\gamma^{\varphi\varphi}\partial_\theta\gamma_{\varphi\varphi}\beta^\varphi - N\gamma^{\varphi\varphi}K_{\theta\varphi} + \frac{\beta^\varphi}{N}\left(K_{\theta\varphi}\beta^\varphi - \partial_\theta N\right)\\
{}^4\Gamma^r{}_{rr} &= \frac{1}{2}\gamma^{rr}\partial_r\gamma_{rr}\\
{}^4\Gamma^r{}_{r\theta} &= \frac{1}{2}\gamma^{rr}\partial_\theta\gamma_{rr}\\
{}^4\Gamma^r{}_{\theta\theta} &= -\frac{1}{2}\gamma^{rr}\partial_r\gamma_{\theta\theta}\\
{}^4\Gamma^r{}_{\varphi\varphi} &= -\frac{1}{2}\gamma^{rr}\partial_r\gamma_{\varphi\varphi}\\
{}^4\Gamma^\theta{}_{rr} &= -\frac{1}{2}\gamma^{\theta\theta}\partial_\theta\gamma_{rr}\\
{}^4\Gamma^\theta{}_{r\theta} &= \frac{1}{2}\gamma^{\theta\theta}\partial_r\gamma_{\theta\theta}\\
{}^4\Gamma^\theta{}_{\theta\theta} &= \frac{1}{2}\gamma^{\theta\theta}\partial_\theta\gamma_{\theta\theta}\\
{}^4\Gamma^\theta{}_{\varphi\varphi} &= -\frac{1}{2}\gamma^{\theta\theta}\partial_\theta\gamma_{\varphi\varphi}\\
{}^4\Gamma^\varphi{}_{\varphi r} &= \frac{1}{2}\gamma^{\varphi\varphi}\partial_r\gamma_{\varphi\varphi} + \frac{\beta^\varphi}{N}K_{r\varphi}\\
{}^4\Gamma^\varphi{}_{\varphi\theta} &= \frac{1}{2}\gamma^{\varphi\varphi}\partial_\theta\gamma_{\varphi\varphi} + \frac{\beta^\varphi}{N}K_{\theta\varphi}
\end{aligned}
$$

Bibliography

[1] James W. York Jr. "Kinematics and Dynamics of General Relativity". In: *Proceedings, Sources of Gravitational Radiation: Seattle, WA, USA, July 24 - August 4, 1978*, pp. 83–126.

[2] B. P. Abbott et al. "Observation of Gravitational Waves from a Binary Black Hole Merger". In: *Physical Review Letters* 116.6 (Feb. 2016). ISSN: 1079-7114. DOI: 10.1103/physrevlett.116.061102. URL: http://dx.doi.org/10.1103/PhysRevLett.116.061102.

[3] Miguel Alcubierre. "Introduction to 3+1 Numerical Relativity". In: *Introduction to 3+1 Numerical Relativity* (Apr. 2006). DOI: 10.1093/acprof:oso/9780199205677.001.0001.

[4] Marcus Ansorg, Bernd Brügmann and Wolfgang Tichy. "Single-domain spectral method for black hole puncture data". In: *Physical Review D* 70.6 (Sept. 2004). ISSN: 1550-2368. DOI: 10.1103/physrevd.70.064011. URL: http://dx.doi.org/10.1103/PhysRevD.70.064011.

[5] Astropy Collaboration et al. "The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package". In: 156.3, 123 (Sept. 2018), p. 123. DOI: 10.3847/1538-3881/aabc4f. arXiv: 1801.02634 [astro-ph.IM].

[6] Thomas W. Baumgarte and Stuart L. Shapiro. *Numerical Relativity: Solving Einstein's Equations on the Computer*. Cambridge University Press, 2010. DOI: 10.1017/CBO9781139193344.

[7] Thomas W. Baumgarte and Stuart L. Shapiro. *Numerical Relativity: Starting from Scratch*. Cambridge University Press, 2021. DOI: 10.1017/9781108933445.

[8] Marsha J Berger and Joseph Oliger. "Adaptive mesh refinement for hyperbolic partial differential equations". In: *Journal of Computational Physics* 53.3 (1984), pp. 484–512. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(84)90073-1. URL: https://www.sciencedirect.com/science/article/pii/0021999184900731.

[9] Luc Blanchet and Jerome Novak. *Testing MOND in the Solar System*. 2011. arXiv: 1105.5815 [astro-ph.CO].

[10] C. Bona et al. "General-covariant evolution formalism for numerical relativity". In: *Physical Review D* 67.10 (May 2003). ISSN: 1089-4918. DOI: 10.1103/physrevd.67.104005. URL: http://dx.doi.org/10.1103/PhysRevD.67.104005.

[11] Silvano Bonazzola, Eric Gourgoulhon and Jean-Alain Marck. "Numerical approach for high precision 3D relativistic star models". In: *Phys. Rev. D* 58 (10 Oct. 1998), p. 104020. DOI: 10.1103/PhysRevD.58.104020. URL: https://link.aps.org/doi/10.1103/PhysRevD.58.104020.

[12] S. Bowyer et al. "Cosmic X-ray Sources". In: *Science* 147.3656 (Jan. 1965), pp. 394–398. DOI: 10.1126/science.147.3656.394.

[13] John Boyd, To Marilyn and Paraphrasing Eliot. "Chebyshev and Fourier Spectral Methods". In: (Oct. 2000).

[14] Avery E. Broderick, Abraham Loeb and Ramesh Narayan. "THE EVENT HORIZON OF SAGITTARIUS A*". In: *The Astrophysical Journal* 701.2 (July 2009), pp. 1357–1366. ISSN: 1538-4357. DOI: 10.1088/0004-637x/701/2/1357. URL: http://dx.doi.org/10.1088/0004-637X/701/2/1357.

[15] J. David Brown et al. "Turduckening black holes: an analytical and computational study". In: *Phys. Rev. D* 79 (2009), p. 044023. DOI: 10.1103/PhysRevD.79.044023. eprint: arXiv:0809.3533[gr-qc].

[16] Vitor Cardoso and Paolo Pani. "Testing the nature of dark compact objects: a status report". In: *Living Reviews in Relativity* 22.1 (July 2019). ISSN: 1433-8351. DOI: 10.1007/s41114-019-0020-4. URL: http://dx.doi.org/10.1007/s41114-019-0020-4.

[17] Carpet: Adaptive Mesh Refinement for the Cactus Framework. URL: https://bitbucket.org/eschnett/carpet.git.

[18] Sean M. Carroll. *Spacetime and Geometry: An Introduction to General Relativity*. Cambridge University Press, 2019.

[19] Gregory B. Cook. "Initial Data for Numerical Relativity". In: *Living Reviews in Relativity* 3.1 (Nov. 2000). ISSN: 1433-8351. DOI: 10.12942/lrr-2000-5. URL: http://dx.doi.org/10.12942/lrr-2000-5.

[20] Thibault Damour and Gilles Esposito-Farèse. "Light deflection by gravitational waves from localized sources". In: *Physical Review D* 58.4 (June 1998). ISSN: 1089-4918. DOI: 10.1103/physrevd.58.044003. URL: http://dx.doi.org/10.1103/PhysRevD.58.044003.

[21] "Dynamical Structure and Definition of Energy in General Relativity". en. In: *Physical Review* 116.5 (Dec. 1959), pp. 1322–1330. ISSN: 0031-899X. DOI: 10.1103/PhysRev.116.1322. URL: https://link.aps.org/doi/10.1103/PhysRev.116.1322 (visited on 21/02/2020).

[22] Albert Einstein. "The Foundation of the General Theory of Relativity". In: *Annalen Phys.* 49.7 (1916). [Annalen Phys.14,517(2005); ,65(1916); Annalen Phys.354,no.7,769(1916)], pp. 769–822. DOI: 10.1002/andp.200590044,10.1002/andp.19163540702.

[23] Albert Einstein and N. Rosen. "On Gravitational waves". In: *J. Franklin Inst.* 223 (1937), pp. 43–54. DOI: 10.1016/S0016-0032(37)90583-0.

[24] D. Gondek-Rosińska, E. Gourgoulhon and P. Haensel. "Are rotating strange quark stars good sources of gravitational waves?" In: *Astronomy Astrophysics* 412.3 (Dec. 2003), pp. 777–790. ISSN: 1432-0746. DOI: 10.1051/0004-6361:20031431. URL: http://dx.doi.org/10.1051/0004-6361:20031431.

[25] Tom Goodale et al. "The Cactus Framework and Toolkit: Design and Applications". In: *Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science*. Berlin: Springer, 2003. URL: http://edoc.mpg.de/3341.

[26] Gourgoulhon, E and Grandclément, P and Marck, J and Novak, J and Taniguchi, K. *LORENE*. [Online; accessed 10-May-2021]. URL: https://lorene.obspm.fr.

[27] Eric Gourgoulhon. *31 Formalism in General Relativity*. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-24525-1. URL: https://doi.org/10.1007%2F978-3-642-24525-1.

[28] Eric Gourgoulhon, Philippe Grandclement and Silvano Bonazzola. "Binary black holes in circular orbits. I. A global spacetime approach". In: *Phys. Rev. D* 65 (2002), p. 044020. DOI: 10.1103/PhysRevD.65.044020. eprint: arXiv:gr-qc/0106015.

[29] Eric Gourgoulhon et al. "Quasiequilibrium sequences of synchronized and irrotational binary neutron stars in general relativity. I. Method and tests". In: *Phys. Rev. D* 63 (2001), p. 064029. DOI: 10.1103/PhysRevD.63.064029. eprint: arXiv:gr-qc/0007028.

[30] Philippe Grandclement, Eric Gourgoulhon and Silvano Bonazzola. "Binary black holes in circular orbits. II. Numerical methods and first results". In: *Phys. Rev. D* 65 (2002), p. 044021. DOI: 10.1103/PhysRevD.65.044021. eprint: arXiv:gr-qc/0106016.

[31] Philippe Grandclément and Jérôme Novak. "Spectral Methods for Numerical Relativity". In: *Living Reviews in Relativity* 12.1 (Jan. 2009). ISSN: 1433-8351. DOI: 10.12942/lrr-2009-1. URL: http://dx.doi.org/10.12942/lrr-2009-1.

[32] Roland Haas et al. *The Einstein Toolkit*. Version The "DeWitt-Morette" release, ET_2020_11. To find out more, visit http://einsteintoolkit.org. Nov. 2020. DOI: 10.5281/zenodo.4298887. URL: https://doi.org/10.5281/zenodo.4298887.

[33] W. A. Joye and E. Mandel. "New Features of SAOImage DS9". In: *Astronomical Data Analysis Software and Systems XII*. Ed. by H. E. Payne, R. I. Jedrzejewski and R. N. Hook. Vol. 295. Astronomical Society of the Pacific Conference Series. Jan. 2003, p. 489.

[34] Wolfgang Kastaun. *PostCacuts/PyCactusET*. URL: https://github.com/wokast/PyCactus/tree/master/PostCactus.

[35] Janna Levin and Gabe Perez-Giz. "A periodic table for black hole orbits". In: *Physical Review D* 77.10 (May 2008). ISSN: 1550-2368. DOI: 10.1103/physrevd.77.103005. URL: http://dx.doi.org/10.1103/PhysRevD.77.103005.

[36] Frank Löffler et al. "The Einstein Toolkit: A Community Computational Infrastructure for Relativistic Astrophysics". In: *Classical and Quantum Gravity - CLASS QUANTUM GRAVITY* 29 (Nov. 2011). DOI: 10.1088/0264-9381/29/11/115001.

[37] Michele Maggiore. *Gravitational Waves. Vol. 1: Theory and Experiments*. Oxford Master Series in Physics. Oxford University Press, 2007. ISBN: 978-0-19-857074-5, 978-0-19-852074-0.

[38] Michele Maggiore. *Gravitational Waves. Vol. 2: Astrophysics and Cosmology*. Oxford University Press, Mar. 2018. ISBN: 978-0-19-857089-9.

[39] *McLachlan, a Public BSSN Code*. URL: http://www.cct.lsu.edu/~eschnett/McLachlan/.

[40] Charles W. Misner, K. S. Thorne and J. A. Wheeler. *Gravitation*. San Francisco: W. H. Freeman, 1973. ISBN: 9780716703440, 9780691177793.

[41] Yen Chin Ong. "Space–time singularities and cosmic censorship conjecture: A Review with some thoughts". In: *International Journal of Modern Physics A* 35.14 (May 2020), p. 2030007. ISSN: 1793-656X. DOI: 10.1142/s0217751x20300070. URL: http://dx.doi.org/10.1142/S0217751X20300070.

[42] Don N. Page and Kip S. Thorne. "Disk-Accretion onto a Black Hole. Time-Averaged Structure of Accretion Disk". In: 191 (July 1974), pp. 499–506. DOI: 10.1086/152990.

[43] Denis Pollney. *Using IDAnalyticBH*. URL: https://einsteintoolkit.org/thornguide/EinsteinInitialData/IDAnalyticBH/documentation.html (visited on 05/05/2021).

[44] Dennis Pollney. *Lectures on Numerical modelling of gravitational wave sources*. Jan. 2020. URL: http://www.chrisengelbrecht2020.com/lecture_notes.html.

[45] Frans Pretorius. "Evolution of Binary Black-Hole Spacetimes". In: *Phys. Rev. Lett.* 95 (12 Sept. 2005), p. 121101. DOI: 10.1103/PhysRevLett.95.121101. URL: https://link.aps.org/doi/10.1103/PhysRevLett.95.121101.

[46] Sbozzolo. *kuibit*. URL: https://github.com/Sbozzolo/kuibit/.

[47] H. L. Shipman. "The Implausible History of Triple Star Models for Cygnus X-1: Evidence for a Black Hole". In: 16 (Feb. 1975), p. 9.

[48] O. Straub et al. "Modelling the black hole silhouette in Sagittarius A* with ion tori". In: *Astronomy Astrophysics* 543 (July 2012), A83. ISSN: 1432-0746. DOI: 10.1051/0004-6361/201219209. URL: http://dx.doi.org/10.1051/0004-6361/201219209.

[49] Jonathan Thornburg. "A Fast Apparent-Horizon Finder for 3-Dimensional Cartesian Grids in Numerical Relativity". In: *Class. Quantum Grav.* 21 (2004), pp. 743–766. DOI: 10.1088/0264-9381/21/2/026. eprint: arXiv:gr-qc/0306056.

[50] Einstein Toolkit. *Einstein Toolkit Installation Guide*. URL: https://nbviewer.jupyter.org/github/nds-org/jupyter-et/blob/master/CactusTutorial.ipynb.

[51] A. Tveito and R. Winther. "Introduction to Partial Differential Equations: A Computational Approach". In: 1998.

[52] F H Vincent, E Gourgoulhon and J Novak. "3+1 geodesic equation and images in numerical spacetimes". In: *Classical and Quantum Gravity* 29.24 (Nov. 2012), p. 245005. ISSN: 1361-6382. DOI: 10.1088/0264-9381/29/24/245005. URL: http://dx.doi.org/10.1088/0264-9381/29/24/245005.

[53] F H Vincent et al. "GYOTO: a new general relativistic ray-tracing code". In: *Classical and Quantum Gravity* 28.22 (Oct. 2011), p. 225011. ISSN: 1361-6382. DOI: 10.1088/0264-9381/28/22/225011. URL: http://dx.doi.org/10.1088/0264-9381/28/22/225011.

[54] Frederic H. Vincent et al. "Accurate Ray-tracing of Realistic Neutron Star Atmospheres for Constraining Their Parameters". In: *The Astrophysical Journal* 855.2 (Mar. 2018), p. 116. ISSN: 1538-4357. DOI: 10.3847/1538-4357/aab0a3. URL: http://dx.doi.org/10.3847/1538-4357/aab0a3.

[55] Wikipedia contributors. *Albrecht Dürer — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-March-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Albrecht_D%C3%BCrer&oldid=1011812958.

[56] Wikipedia contributors. *Ray tracing (graphics) — Wikipedia, The Free Encyclopedia*. [Online; accessed 18-March-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Ray_tracing_(graphics)&oldid=1011256911.

[57] MIGUEL ZILHÃO and FRANK LÖFFLER. "AN INTRODUCTION TO THE EINSTEIN TOOLKIT". In: *International Journal of Modern Physics A* 28.22n23 (Sept. 2013), p. 1340014. ISSN: 1793-656X. DOI: 10.1142/s0217751x13400149. URL: http://dx.doi.org/10.1142/S0217751X13400149.