RESEARCH IN PHYSICS COURSE REPORT

Evolve Schwarzschild geometry using ADM equations

Author:
Dongchan KIM

Supervisor:
Gungwon KANG

Department of Physics
CHUNG-ANG UNIVERSITY

December 9, 2022

Abstract

We have briefly reviewed the 3+1 formalism in general relativity which is essential for numerically solving Einstein's equations. Based on a simple form of this formalism, i.e., ADM formalism, preliminary results on the numerical solution for a spherically symmetric vacuum spacetime, e.g., the Schwarzschild metric, are presented. From this, we lay the groundwork for further simulations in the future such as stellar collapses and binary black hole coalescences.

Contents

List of Figures

List of Tables

List of Abbreviations

NR	**N**umerical **R**elativity
ADM	**A**rnowitt, **D**eser and **M**isner
BSSN	**B**aumgarte, **S**hapiro, **S**hibata and **N**akamura

List of Symbols

g_{ab}, g^{ab}	spacetime metric and its inverse
γ_{ij}, γ^{ij}	spatial metric and its inverse
$\boldsymbol{\partial}_t$	time vector
N	lapse function
$\boldsymbol{\beta}$	shift vector
Σ_t	hypersurface at time t
${}^{(4)}\boldsymbol{R}$, ${}^{(4)}R$	Ricci tensor associated with $\boldsymbol{g}$ and the corresponding Ricci scalar
$\boldsymbol{T}$	matter stress-energy tensor
E	matter energy density
$\boldsymbol{p}$	matter momentum density
$\boldsymbol{S}$	matter stress tensor
$\pounds_{\boldsymbol{m}}$	Lie derivative along vector field $\boldsymbol{m}$

Chapter 1

Introduction

1.1 Why we need numerical relativity?

General relativity is the theory that best describes modern cosmology, celestial bodies such as black holes, and gravitational waves. In general relativity, Einstein's field equation describes curved spacetime as the momentum and energy of matter.

However, these are ten nonlinear partial differential equations, whose complete solutions have not yet been solved analytically, except for the Friedmann–Lemaître–Robertson–Walker metric, etc. With advances in computer algorithms and physics, Einstein's equations began to be solved numerically, which is the beginning of numerical relativity.

1.2 Introduction to numerical relativity

In numerical relativity, Einstein's equations is separated into space part and time part, that is, 3+1 decomposition.

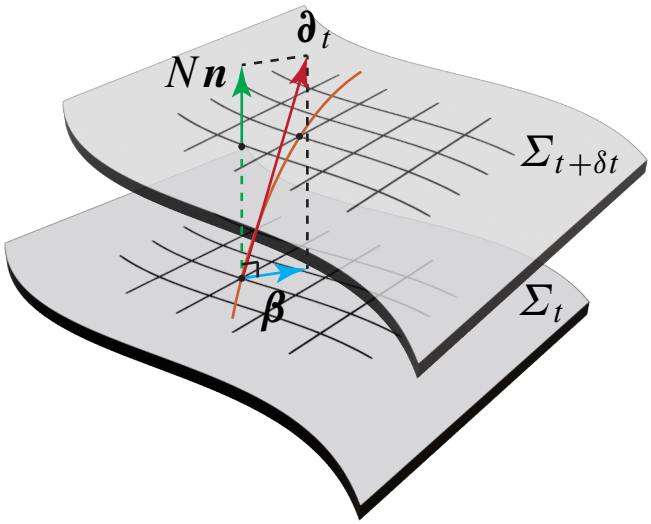

FIGURE 1.1: A foliation of the spacetime. There are time vector ∂_t, normal evolution vector $N\boldsymbol{n}$, and shift vector $\boldsymbol{\beta}$ satisfying $\partial_t = N\boldsymbol{n} + \boldsymbol{\beta}$ between the two hypersurfaces Σ_t, $\Sigma_{t+\delta t}$.

From this, it is possible to calculate the next spatial metric from the initial condition of the space at a specific time. This is done in a similar way as in Newtonian classical mechanics, given the initial position and velocity of an object, the position and velocity at the next time can be calculated using the acceleration due to the force acting on the object. However, unlike in Newtonian mechanics, the initial value must satisfy certain specific conditions. Therefore, the calculation is made by first constructing the initial data, selecting the appropriate coordinates, and then evolving it from the numerical method.

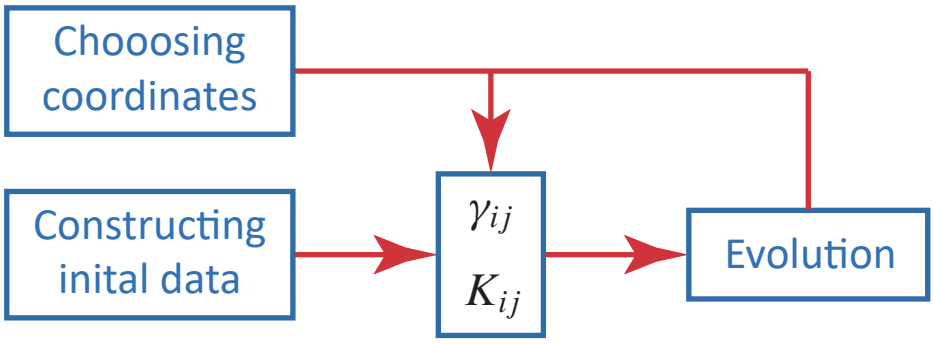

FIGURE 1.2: A brief schema of numerical relativity.

In this work, we try to numerically evolve the Schwarzschild metric from its initial conditions to understand the basic way of working with numerical relativity.

1.3 Notation and conventions

Throughout this report, we follows the "Landau-Lifshitz Spacelike Convention"$(-+++)$ as [1]. Also we will adopt a units for measurements in which both the gravitational constant G and the speed of light c are assigned the values of one.

We denote the dimension 4 spacetime metric by g_{ab}, the dimension 3 spatial metric by γ_{ij}. Also the dimension 4 objects associated with g_{ab} are denoted with a superscript $^{(4)}$ in front of the symbol, objects related to γ_{ij} carry no decorations.

Chapter 2

The ADM Equations

2.1 Introduction

In this chapter, we present how Einstein's equations can be decomposed into two constraints and two evolution equations. The development in this chapter follows [6].

2.2 The Einstein Equation

Einstein's equation is

$$^{(4)}\boldsymbol{R} - \frac{1}{2}{}^{(4)}R\boldsymbol{g} = 8\pi\boldsymbol{T}. \tag{2.1}$$

We limit cosmological constant $\Lambda = 0$.

2.3 Constraint equations

Let's start with the *Gauss relation*

$$\gamma^{\mu}_{\alpha}\gamma^{\nu}_{\beta}\gamma^{\gamma}_{\rho}\gamma^{\sigma}_{\delta}\,{}^{(4)}R^{\rho}_{\sigma\mu\nu} = R^{\gamma}_{\delta\alpha\beta} + K^{\gamma}_{\alpha}K_{\delta\beta} - K^{\gamma}_{\beta}K_{\alpha\delta}. \tag{2.2}$$

We can obtain *scalar Gauss relation*

$$^{(4)}R + 2\,{}^{(4)}R_{\mu\nu}n^{\mu}n^{\nu} = R + K^2 - K_{ij}K^{ij} \tag{2.3}$$

by contract the Gauss relation Eq. (2.2) on the indices γ and α and use $\gamma^{\mu}_{\alpha}\gamma^{\alpha}_{\rho} = \gamma^{\mu}_{\rho} = \delta^{\mu}_{\rho} + n^{\mu}n_{\rho}$, and take its trace with respect to $\boldsymbol{\gamma}$.

From Eq. (2.1), after full projection perpendicular to hypersurface Σ_t, we get

$$^{(4)}\boldsymbol{R}(\boldsymbol{n},\boldsymbol{n}) + \frac{1}{2}{}^{(4)}R = 8\pi \boldsymbol{T}(\boldsymbol{n},\boldsymbol{n}) =: 8\pi E \tag{2.4}$$

since $\boldsymbol{g}(\boldsymbol{n},\boldsymbol{n}) = -1$. By combining Eq. (2.3), (2.4), we get

$$R + K^2 - K^{ij}_{ij} = 16\pi E \tag{2.5}$$

which is called the *Hamiltonian constraint*.

Now let us project Eq. (2.1) onto Σ_t and normal $\boldsymbol{n}$,

$$^{(4)}\boldsymbol{R}(\boldsymbol{n},\vec{\boldsymbol{\gamma}}(.)) - \frac{1}{2}{}^{(4)}R\boldsymbol{g}(\boldsymbol{n},\vec{\boldsymbol{\gamma}}(.)) = 8\pi \boldsymbol{T}(\boldsymbol{n},\vec{\boldsymbol{\gamma}}(.)). \tag{2.6}$$

We can use *Codazzi relation*

$$\gamma^{\gamma}_{\rho} n^{\sigma} \gamma^{\mu}_{\alpha} \gamma^{\nu}_{\beta} {}^{(4)}R^{\rho}_{\sigma\mu\nu} = D_{\beta} K^{\gamma}_{\alpha} - D_{\alpha} K^{\gamma}_{\beta} \tag{2.7}$$

to get *contracted Codazzi relation*

$$\gamma^{\mu}_{\alpha} n^{\nu} {}^{(4)}R_{\mu\nu} = D_{\alpha} K - D_{\mu} K^{\mu}_{\alpha} \tag{2.8}$$

by contracting the Eq. (2.7) on the indices α and γ.

Since $\boldsymbol{g}(\boldsymbol{n},\vec{\boldsymbol{\gamma}}(.))$ in Eq. (2.6) is equal to 0 and by introducing *matter momentum density* $\boldsymbol{p} := -\boldsymbol{T}(\boldsymbol{n},\vec{\boldsymbol{\gamma}}(.))$, we get

$$\boldsymbol{D}\cdot\vec{\boldsymbol{K}} - \boldsymbol{D}K = 8\pi \boldsymbol{p}, \tag{2.9}$$

or, in components,

$$D_j K^j_i - D_i K = 8\pi p_i \tag{2.10}$$

which is called the *momentum constraint*.

2.4 Evolution equations

Since we can write time vector in to sum of the normal evolution vector $\boldsymbol{m} := N\boldsymbol{n}$ and the shift vector $\boldsymbol{\beta}$,

$$\boldsymbol{\partial}_t =: \boldsymbol{m} + \boldsymbol{\beta}, \tag{2.11}$$

we can write

$$\mathcal{L}_{\boldsymbol{m}}\boldsymbol{T} = \mathcal{L}_{\boldsymbol{\partial}_t}\boldsymbol{T} - \mathcal{L}\boldsymbol{T}, \tag{2.12}$$

for $\boldsymbol{T}$ be any tensor field tangent to Σ_t. Moreover, Lie derivative is simply obtained by taking the partial derivative of the vector components with respect to t. Therefore, Eq. (2.12) can be written as

$$\mathcal{L}_{\boldsymbol{m}} T^{i\cdots}_{j\cdots} = \left(\frac{\partial}{\partial t} - \mathcal{L}_{\boldsymbol{\beta}}\right) T^{i\cdots}_{j\cdots}. \tag{2.13}$$

By applying it to extrinsic curvature

$$\mathcal{L}_{\boldsymbol{m}}\boldsymbol{\gamma} = -2N\boldsymbol{K}, \tag{2.14}$$

it becomes

$$\left(\frac{\partial}{\partial t} - \mathcal{L}_{\boldsymbol{\beta}}\right)\gamma_{ij} = -2NK_{ij}, \tag{2.15}$$

which is called the *evolution equation for the spatial metric*.

If we applying the operator $\vec{\boldsymbol{\gamma}}^*$ to the Einstein equation,

$$\vec{\boldsymbol{\gamma}}^{*(4)}\boldsymbol{R} = 8\pi\left(\vec{\boldsymbol{\gamma}}^*\boldsymbol{T} - \frac{1}{2}T\vec{\boldsymbol{\gamma}}^*\boldsymbol{g}\right). \tag{2.16}$$

From the 3+1 decomposition of the Riemann tensor, we obtained

$$\vec{\boldsymbol{\gamma}}^{*(4)}\boldsymbol{R} = -\frac{1}{N}\mathcal{L}_{\boldsymbol{m}}\boldsymbol{K} - \frac{1}{N}\boldsymbol{D}\boldsymbol{D}N + \boldsymbol{R} + K\boldsymbol{K} - 2\boldsymbol{K}\cdot\vec{\boldsymbol{K}}. \tag{2.17}$$

Therefore

$$-\frac{1}{N}\mathcal{L}_{\boldsymbol{m}}\boldsymbol{K} - \frac{1}{N}\boldsymbol{D}\boldsymbol{D}N + \boldsymbol{R} + K\boldsymbol{K} - 2\boldsymbol{K}\cdot\vec{\boldsymbol{K}} = 8\pi\left[\boldsymbol{S} - \frac{1}{2}(S-E)\boldsymbol{\gamma}\right], \tag{2.18}$$

where *matter stress tensor* $\boldsymbol{S} := \vec{\boldsymbol{\gamma}}^*\boldsymbol{T}$.

The result from property that the Lie derivative along $\boldsymbol{m}$ of any tensor field $\boldsymbol{T}$ tangent to Σ_t is a tensor field tangent to Σ_t, Eq. (2.18) can be written as

$$\mathcal{L}_{\boldsymbol{m}}K_{ij} = -D_iD_jN + N\left\{R_{ij} + KK_{ij} - 2K_{ik}K^k_j + 4\pi\left[(S-E)\gamma_{ij} - 2S_{ij}\right]\right\}. \tag{2.19}$$

From Eq. (2.11), we get *evolution equation for the extrinsic curvature*

$$\left(\frac{\partial}{\partial t} - \mathcal{L}_{\boldsymbol{\beta}}\right) K_{ij} = -D_i D_j N + N\Big\{R_{ij} + K K_{ij} - 2K_{ik}K^k_j + 4\pi\big[(S-E)\gamma_{ij} - 2S_{ij}\big]\Big\}. \tag{2.20}$$

2.5 Summary

In this chapter, we obtained the *Hamiltonian constraint*

$$R + K^2 - K_{ij}K^{ij} = 16\pi E, \tag{2.21}$$

the *momentum constraint*

$$D_j K^j_i - D_i K = 8\pi p_i, \tag{2.22}$$

the *evolution equation for the spatial metric*

$$\left(\frac{\partial}{\partial t} - \mathcal{L}_{\boldsymbol{\beta}}\right)\gamma_{ij} = -2NK_{ij}, \tag{2.23}$$

and the *evolution equation for the extrinsic curvature*

$$\left(\frac{\partial}{\partial t} - \mathcal{L}_{\boldsymbol{\beta}}\right) K_{ij} = -D_i D_j N + N\Big\{R_{ij} + K K_{ij} - 2K_{ik}K^k_j + 4\pi\big[(S-E)\gamma_{ij} - 2S_{ij}\big]\Big\}. \tag{2.24}$$

Chapter 3

Numerical simulation

3.1 Introduction

In this chapter, we describe the initial data set required prior to simulation, the numerical method for evolution, and the results.

3.2 Schwarzschild black hole

3.2.1 Types of black holes

Accordingly to the No-Hair theorem, all black holes solutions of the Einstein-Maxwell equation of electromagnetism in general relativity can be completely characterized by their observable classical parameters mass, electric charge and angular momentum.

The schwarzschild metric describes the spacetime geometry exterior to any spherical collapsing body. Kerr metric describes the geometry of empty spacetime around a rotating uncharged axially-symmetric black hole with quasi-spherical event horizon. There are also the Reissner-Nordström metric and the Kerr-Newman metric that describe charged black holes[8, 5]. In Table 3.1, it can be seen that the types of black holes are classified according to angular momentum and charge.

TABLE 3.1: Classifications of black holes.

	Non-rotating ($J = 0$)	**Rotating** ($J > 0$)
Uncharged ($Q = 0$)	Schwarzschild	Kerr
Charged ($Q \neq 0$)	Reissner-Nordström	Kerr-Newman

3.2.2 Isotropic coordinate

In this report, we choose the simplest form, the Schwarzschild black hole. The original form of the Schwarzschild metric is

$$\mathrm{d}s^2 = -\left(1 - \frac{2M}{r}\right)\mathrm{d}t^2 + \left(1 - \frac{2M}{r}\right)^{-1}\mathrm{d}r^2 + r^2\left(\mathrm{d}\theta^2 + \sin^2\theta\,\mathrm{d}\phi^2\right). \tag{3.1}$$

When r goes $2M$, g_{rr} diverges. However, this is just a coordinate singularity, like the problem that occurs at the north and south poles in the spherical coordinate system[7]. This can be solved by choosing another coordinate system, such as the Kruskal coordinate system.

We can avoid coordinate singularity at $r = 2M$ by adopting an isotropic coordinate system by substituting $r = \bar{r}(1 + M/2\bar{r})^2$, we get

$$\mathrm{d}s^2 = -\left(\frac{1 - M/(2\bar{r})}{1 + M/(2\bar{r})}\right)^2\mathrm{d}t^2 + \left(1 + \frac{M}{2\bar{r}}\right)^4\left(\mathrm{d}\bar{r}^2 + \bar{r}^2\,\mathrm{d}\theta^2 + \bar{r}^2\sin^2\theta\,\mathrm{d}\phi^2\right). \tag{3.2}$$

This coordinate system describes the area outside the event horizon $\bar{r} = M/2$. The reason for using this coordinate system is as follows.

1. A spatial metric is numerically valid in any space where $r > M/2$.
2. Since spatial metrics are flat, they can be replaced with Cartesian coordinates, which is more useful for numerical calculations[2].

So, in practice we use metric:

$$\mathrm{d}s^2 = -\left(\frac{1 - M/(2r)}{1 + M/(2r)}\right)^2\mathrm{d}t^2 + \left(1 + \frac{M}{2r}\right)^4\left(\mathrm{d}x^2 + \mathrm{d}y^2 + \mathrm{d}z^2\right), \tag{3.3}$$

where $r = \sqrt{x^2 + y^2 + z^2}$.

3.3 Gauge conditions

Eq. (2.21)-(2.24) does not contain any time derivative of lapse function N nor of the shift vector $\boldsymbol{\beta}$. This means that N and $\boldsymbol{\beta}$ are not dynamical variables. Therefore, we may choose the lapse and shift freely, without changing the physical solution g of the Einstein equation[6].

In this simulation, we choose the lapse function and the shift vector

$$N = \frac{1 - M/(2r)}{1 + M/(2r)}, \qquad \boldsymbol{\beta} = 0. \tag{3.4}$$

3.4 Initial data

After 3+1 decomposition, we should evolve forward in time some initial data. Instead of solving Hamiltonian and momentum constraint, we use well-known initial data from isotropic coordinates of Schwarzshcild metric. So the initial spatial metric becomes $\gamma_{ij} = (1 + M/(2r))^4 \delta_{ij}$ and the initial extrinsic curvature becomes $K_{ij} = 0$ since there is no time-dependent and zero shift.

We will show that these data satisfies Eq. (2.21, 2.22). Since E and p_i are all 0 in vacuum space and $K_{ij} = 0$, the momentum constraint is naturally satisfied. Now, to satisfy the hamiltonian constraint, we need to show that the 3-metric Ricci scalar R is 0. Let's use a spherical coordinate system here. The non-vanishing Ricci tensor is:

$$R_{rr} = -\frac{8M}{r\,(M + 2r)^2}, \tag{3.5}$$

$$R_{\theta\theta} = \frac{4Mr}{(M + 2r)^2}, \tag{3.6}$$

$$R_{\phi\phi} = \frac{4Mr\sin^2(\theta)}{(M + 2r)^2}. \tag{3.7}$$

Therefore, it can be seen that the 3-metric Ricci scalar $R = \gamma^{ij} R_{ij} = 0$, and it can be confirmed that the given constraint condition is well satisfied.

3.5 Numerical methods

3.5.1 Finite difference method

This simulation uses the finite difference method. The Taylor expansion of the function $f(x)$ in x_0 is

$$f(x_0 + h) = f(x_0) + \frac{f'(x_0)}{1!}h + \frac{f^{(2)}(x_0)}{2!}h^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}h^n + \cdots . \tag{3.8}$$

Arranging this, we get

$$\begin{aligned} \frac{f(x_0+h)-f(x_0)}{h} &= f'(x_0)+\frac{f^{(2)}(x_0)}{2!}h+\cdots \\ &= f'(x_0)+\mathcal{O}(h). \end{aligned} \tag{3.9}$$

Accuracy is on the order of O(h).

The central difference method selects the function value from $x-h$ and $x+h$, respectively, and has a more accurate error of $\mathcal{O}(h^2)$.

$$f'(x_0)=\frac{f(x+h)-f(x-h)}{2h}+\mathcal{O}(h^2). \tag{3.10}$$

3.5.2 Boundary condition

The isotropic coordinates of the schwarzschild metric only describe the region outside the black hole horizon, i.e. $r=\frac{M}{2}$. Therefore, γ_{ij} and K_{ij} did not evolve in the region of $r\leq\frac{M}{2}$.

At the boundary outside the grid, the physical quantity at the corresponding point was calculated using linear extrapolation. For example, when we need to find the derivative at f_i,

$$(\partial f)_{i-2}=\frac{f_{i-1}-f_{i-3}}{2h}, \tag{3.11}$$

$$(\partial f)_{i-1}=\frac{f_i-f_{i-2}}{2h}. \tag{3.12}$$

Therefore, we get

$$\begin{aligned} (\partial f)_i &= \frac{f_{i-1}-f_{i-3}}{2h} \\ &= \frac{2f_i-f_{i-1}-2f_{i-2}+f_{i-3}}{2h}. \end{aligned} \tag{3.13}$$

Alternatively, there is a way to use a fixed value at the grid boundary, as well as at the black hole horizon.

3.5.3 Inverse matrix

Cofactors were used to find the inverse matrix. See Appendix A.

3.6 Grid setting

The size of the grid is 100^3, the mass of the black hole is $0.2M$, and the grid distance is $0.01M$. Thus, the horizon of a black hole corresponds to $0.1M$, i.e. the surface of a sphere with a radius of 10 grids.

3.7 Result

The results obtained at first do not change with time as shown in Fig. 3.1.

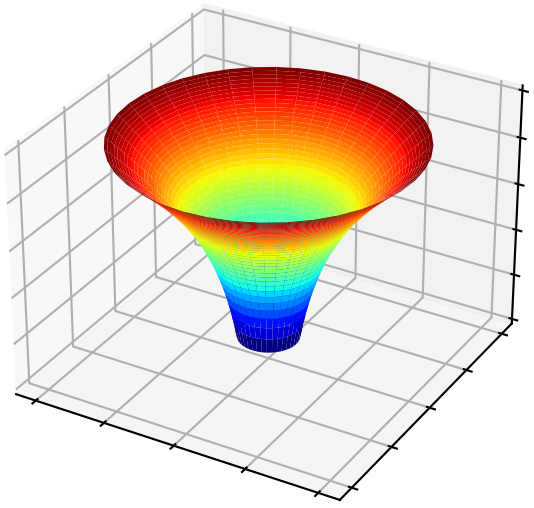

FIGURE 3.1: Flamm's paraboloid from the first simulation. It represents γ_{xx} in the equatorial plane where $\theta = \frac{\pi}{2}$. The slope at each point represents the magnitude of γ_{xx}.

However, when the errors in the coordinate system of the lapse function and minor errors were corrected, the metric could no longer be considered static as shown in Figure 1.

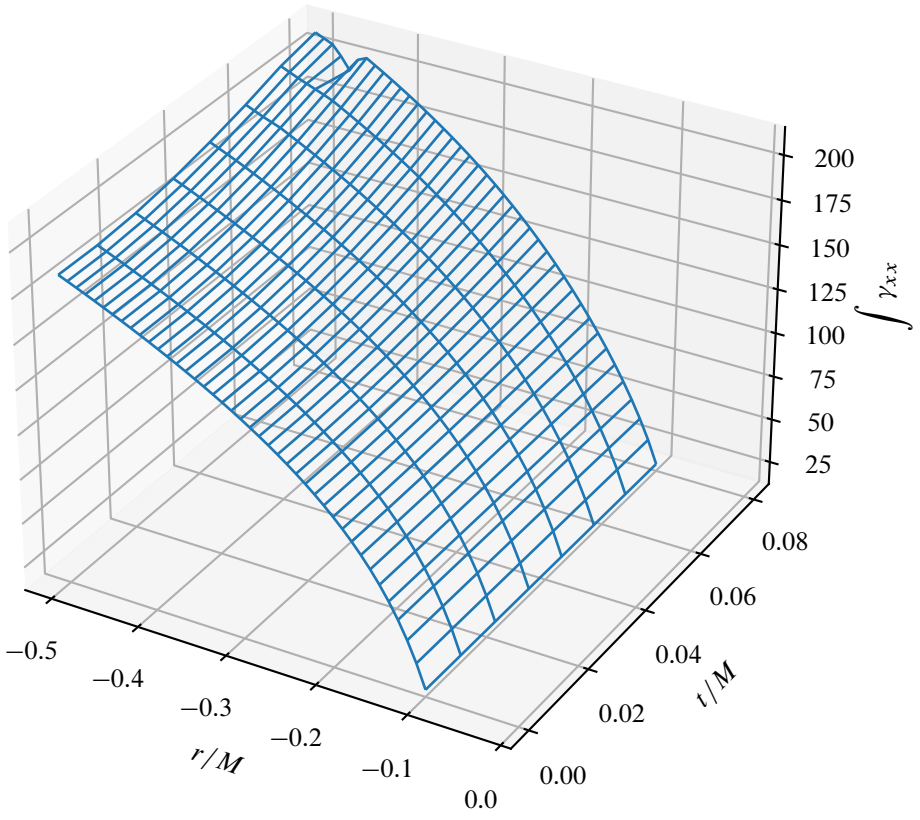

FIGURE 3.2: The unidirectional component of Flamm's paraboloid as a function of time. Even in a very short time interval, errors gradually accumulate over time, especially distortions can be seen.

Chapter 4

Discussion and Conclusion

4.1 Analytical calculation of evolution values

In order for the metric to be static according to time, $K_{ij} = 0$ must always be present. Also, to keep $K_{ij} = 0$, in Eq. (2.24), $-D_i D_j N + N R_{ij} = 0$. Let's check if this is true by calculating K_{rr} in the spherical coordinate system.

The lapse function is

$$N = \frac{1 - M/(2r)}{1 + M/(2r)}, \tag{4.1}$$

And the covariant derivative of lapse function is

$$D_i D_j N = \frac{\partial^2 N}{\partial x^i \partial x^j} - \Gamma^k_{ij} \frac{\partial N}{\partial x^k}. \tag{4.2}$$

When $i = j = r$,

$$\frac{\partial^2 N}{\partial x^r \partial x^r} = -\frac{16M}{(M + 2r)^3}. \tag{4.3}$$

Since N is a function of r only, the significant term in the Christoffel symbols is

$$\Gamma^r_{rr} = -\frac{8M}{r\,(M + 2r)^2}. \tag{4.4}$$

Therefore, we get

$$D_r D_r N = -\frac{16M\left(Mr + 2M + 2r^2\right)}{r\,(M + 2r)^4}. \tag{4.5}$$

On the other hand, the Ricci tensor is

$$R_{rr} = -\frac{8M}{r\,(M + 2r)^2}, \tag{4.6}$$

we get

$$NR_{rr} = -\frac{8M\,(-M+2r)}{r\,(M+2r)^3}. \tag{4.7}$$

Therefore,

$$-D_r D_r N + NR_{rr} = \frac{8M^2\,(M+2r+4)}{r\,(M+2r)^4} \tag{4.8}$$

Contrary to expectations, it does not become 0.

4.2 How about initial data?

Let's check that the simulation calculates the Hamiltonian constraints well, at least for the initial data. In the case of the momentum constraint, since $K_{ij} = 0$, the condition is obviously satisfied.

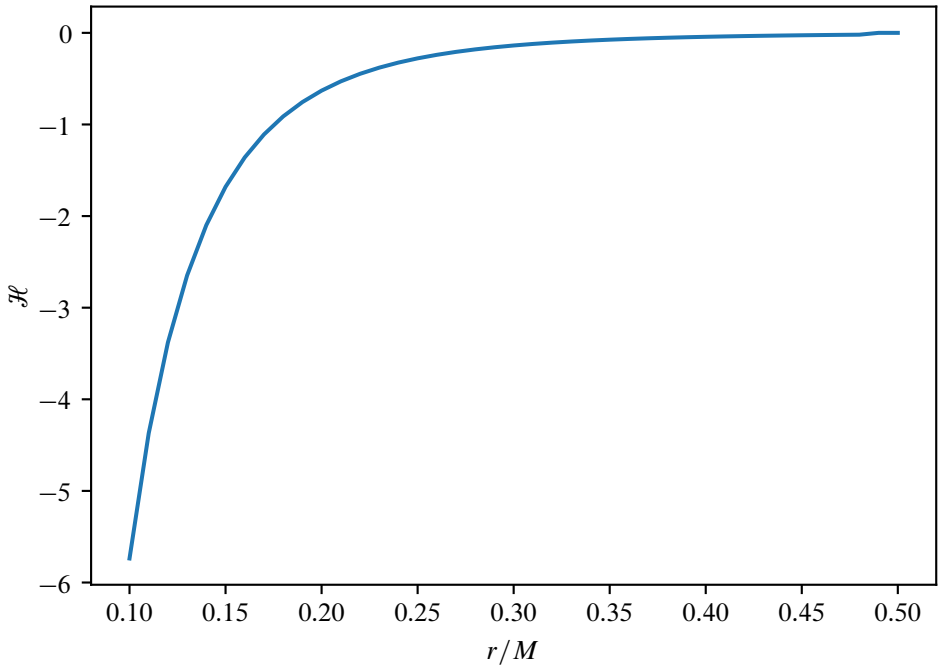

FIGURE 4.1: It shows the value of $\mathcal{H} \equiv R + K^2 - K_{ij}K^{ij}$, the left side of the Hamiltonian constraint, according to r.

It is close to the expected value of 0, but it can be seen that the error gets worse as you get closer to the black hole horizon. In this simulation, since the numerical derivative is calculated on the order of $\mathcal{O}(h^2)$, if the grid spacing is reduced from 0.01 to 0.005, that is, by a factor of 2, the error should be reduced by a factor of 4, which is shown in Figure 4.2.

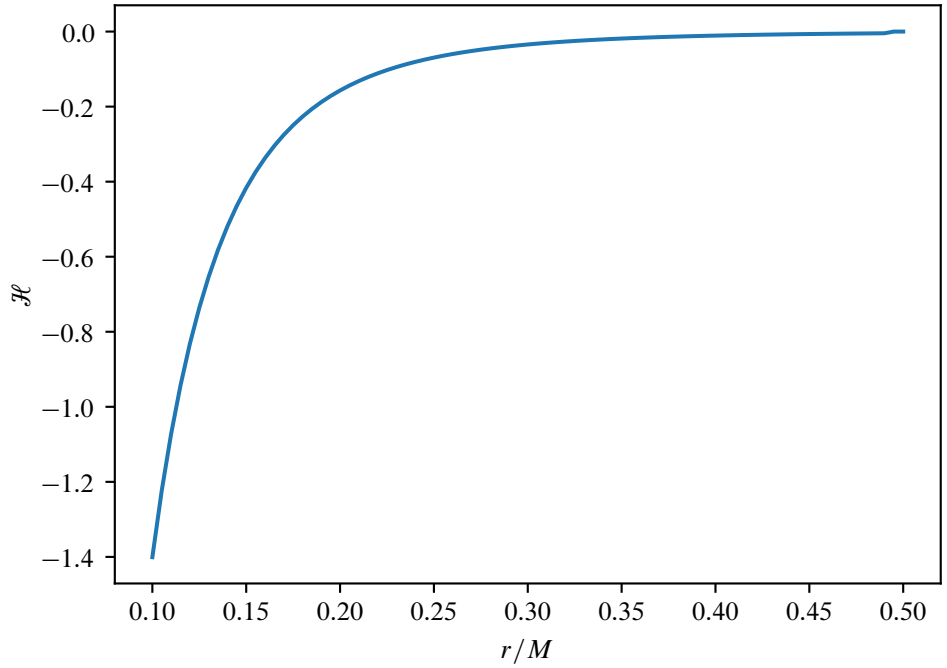

FIGURE 4.2: It shows the value of $\mathcal{H} \equiv R + K^2 - K_{ij}\,K^{ij}$, the left side of the Hamiltonian constraint, according to r. Compared to Fig. 4.1, the resolution has doubled.

4.3 Conclusion

The off-diagonal terms of γ_{ij} have a slight relative error of the order of 10^{-7} in comparison with diagonal terms. This accumulates over time. Ignoring these small errors, we were able to confirm that the spatial metric was static and still spherically symmetric. We should either accept a variation form such as the BSSN equation, or try to do more accurate numerical calculations for better simulations.

Appendix A

Frequently Asked Questions

A.1 Used code

This is part of the code used for simulation.

```
import pickle

SIZE = 101
CENTER = SIZE // 2
dx = 0.01
dt = 0.1
M = 0.2

to_ij = [[0, 0], [0, 1], [0, 2], [1, 1], [1, 2], [2, 2]]
_dx = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]

def sqrt(x):
return x ** .5

def to_ind(x, y):
return to_ij.index(sorted([x, y]))


def dist(x, y, z):
return (x * x + y * y + z * z) ** .5


```

```
23 def psi(r):
24 return 1 + M / 2 / r
25
26
27 def lapse(r):
28 return (2 * r - M) / (2 * r + M)
29
30
31 def dNx(ind, x, y, z):
32 if ind == 1:
33 x, y = y, x
34 elif ind == 2:
35 x, z = z, x
36 return 4 * M * x / ((M + 2 * sqrt(x ** 2 + y ** 2 + z ** 2)) ** 2 *
   ↪  sqrt(x ** 2 + y ** 2 + z ** 2))
37 def dNxx(ind, x, y, z):
38 if ind == 3:
39 x, y = y, x
40 elif ind == 5:
41 x, z = z, x
42 return 4 * M * (-x ** 2 * (M + 2 * sqrt(x ** 2 + y ** 2 + z ** 2)) * (x
   ↪  ** 2 + y ** 2 + z ** 2) ** (3 / 2) - 4 * x ** 2 * (
43 x ** 2 + y ** 2 + z ** 2) ** 2 + (M + 2 * sqrt(x ** 2 + y ** 2 + z **
   ↪  2)) * (
44 x ** 2 + y ** 2 + z ** 2) ** (5 / 2)) / (
45 (M + 2 * sqrt(x ** 2 + y ** 2 + z ** 2)) ** 3 * (x ** 2 + y ** 2 + z **
   ↪  2) ** 3)
46 def dNxy(ind, x, y, z):
47 if ind == 2:
48 y, z = z, y
49 elif ind == 4:
50 x, z = z, x
51 return 4 * M * x * y * (-(M + 2 * sqrt(x ** 2 + y ** 2 + z ** 2)) * (x
   ↪  ** 2 + y ** 2 + z ** 2) - 4 * (
```

```
x ** 2 + y ** 2 + z ** 2) ** (3 / 2)) / (
(M + 2 * sqrt(x ** 2 + y ** 2 + z ** 2)) ** 3 * (x ** 2 + y ** 2 + z **
↪  2) ** (5 / 2))
def ddlapse(ind, r, i, j, k):
_i, _j = to_ij[ind]
x, y, z = (i - CENTER) * dx, (j - CENTER) * dx, (k - CENTER) * dx
if _i == _j:
return dNxx(ind, x, y, z) - sum(Christoffel[_k][ind][i][j][k] * dNx(_k,
↪  x, y, z) for _k in range(3))
else:
return dNxy(ind, x, y, z) - sum(Christoffel[_k][ind][i][j][k] * dNx(_k,
↪  x, y, z) for _k in range(3))
# if ind == 0:
#     return -16 * M / (2 * r + M) ** 3 - 4 * M / (2 * r + M) ** 2 *
↪  Christoffel[0][0][i][j][k]


def transposeMatrix(m):
return list(map(list, zip(*m)))


def getMatrixMinor(m, i, j):
return [row[:j] + row[j + 1:] for row in (m[:i] + m[i + 1:])]


def getMatrixDeternminant(m):
if len(m) == 2:
return m[0][0] * m[1][1] - m[0][1] * m[1][0]
determinant = 0
for c in range(len(m)):
determinant += ((-1) ** c) * m[0][c] *
↪  getMatrixDeternminant(getMatrixMinor(m, 0, c))
return determinant

```

```

def getMatrixInverse(m):
determinant = getMatrixDeternminant(m)
if determinant == 0:
print('Warning : determinant is zero.')
return [[0] * 3 for _ in range(3)]
if len(m) == 2:
return [[m[1][1] / determinant, -1 * m[0][1] / determinant],
[-1 * m[1][0] / determinant, m[0][0] / determinant]]
cofactors = []
for r in range(len(m)):
cofactorRow = []
for c in range(len(m)):
minor = getMatrixMinor(m, r, c)
cofactorRow.append(((-1) ** (r + c)) * getMatrixDeternminant(minor))
cofactors.append(cofactorRow)
cofactors = transposeMatrix(cofactors)
for r in range(len(cofactors)):
for c in range(len(cofactors)):
cofactors[r][c] = cofactors[r][c] / determinant
return cofactors

initial = 0
tc = 0
errest = 1
evolve = 0
for _tt in range(1):
print('Step :', tc)
# 0rr, 1rtheta, 2rphi, 3thetatheta, 4thetaphi, 5phiphi

if initial:
gamma = [[[[0] * SIZE for _i in range(SIZE)] for _j in range(SIZE)] for
↪  _k in range(6)]
```

```
113 K = [[[[0] * SIZE for _i in range(SIZE)] for _j in range(SIZE)] for _k
    ↪  in range(6)]
114 else:
115 with open("gamma_%d.txt" % tc, "rb") as f:
116 gamma = pickle.load(f)
117 with open("K_%d.txt" % tc, "rb") as f:
118 K = pickle.load(f)
119
120 gamma_inv = [[[[0] * SIZE for _i in range(SIZE)] for _j in range(SIZE)]
    ↪  for _k in range(6)]
121 pgamma = [[[[[0] * SIZE for _i in range(SIZE)] for _j in range(SIZE)]
    ↪  for _k in range(6)] for _ in range(3)]
122 meanK = [[[0] * SIZE for _i in range(SIZE)] for _j in range(SIZE)]
123 R = [[[[0] * SIZE for _i in range(SIZE)] for _j in range(SIZE)] for _k
    ↪  in range(6)]
124
125 r0 = M / 2
126 if initial:
127 print('Set Inital Gamma')
128 for i in range(SIZE):
129 for j in range(SIZE):
130 for k in range(SIZE):
131 if i == j == k == CENTER:
132 continue
133 else:
134 r = dist(i - CENTER, j - CENTER, k - CENTER) * dx
135 gamma[0][i][j][k] = psi(r) ** 4
136 gamma[3][i][j][k] = psi(r) ** 4
137 gamma[5][i][j][k] = psi(r) ** 4
138 with open('gamma_%d.txt' % (0), 'wb') as f:
139 pickle.dump(gamma, f)
140 with open('K_%d.txt' % (0), 'wb') as f:
141 pickle.dump(K, f)
142
```

```
print('Get Inverse Gamma')
for i in range(SIZE):
for j in range(SIZE):
for k in range(SIZE):
if i == j == k == CENTER:
continue

_subgamma = [[gamma[0][i][j][k], gamma[1][i][j][k], gamma[2][i][j][k]],
[gamma[1][i][j][k], gamma[3][i][j][k], gamma[4][i][j][k]],
[gamma[2][i][j][k], gamma[4][i][j][k], gamma[5][i][j][k]]]
_gammainv = getMatrixInverse(_subgamma)
for _ind in range(6):
gamma_inv[_ind][i][j][k] = _gammainv[to_ij[_ind][0]][to_ij[_ind][1]]

print('Get Mean K')
for i in range(SIZE):
for j in range(SIZE):
for k in range(SIZE):
if i == j == k == CENTER:
continue
meanK[i][j][k] = sum(
gamma_inv[to_ind(_i, _j)][i][j][k] * K[to_ind(_i, _j)][i][j][k] for _i
↪  in range(3) for _j in range(3))

print('Get Partial Gamma')
for i in range(SIZE):
for j in range(SIZE):
for k in range(SIZE):
if abs(i - CENTER) + abs(j - CENTER) + abs(k - CENTER) <= 1:
continue
if i == 0 or i == SIZE - 1 or j == 0 or j == SIZE - 1 or k == 0 or k ==
↪  SIZE - 1:
continue

```

```
175 for _sup in range(6):
176 # _ipgamma = [(gamma[_sup][i + 1][j][k] - gamma[_sup][i - 1][j][k]) /
    ↪ dx / 2,
177 #             (gamma[_sup][i][j + 1][k] - gamma[_sup][i][j - 1][k]) /
    ↪ dx / 2,
178 #             (gamma[_sup][i][j][k + 1] - gamma[_sup][i][j][k - 1]) /
    ↪ dx / 2]
179 for _sub in range(3):
180 # pgamma[_sub][_sup][i][j][k] = _ipgamma[_sub]
181 # _case = 0
182 # _coord = [i, j, k]
183 # _fixed = _coord[_sub]
184 # _coordp = _coord[:]
185 # _coordp[_sub] += 1
186 # _coordm = _coord[:]
187 # _coordm[_sub] -= 1
188 # if _fixed == SIZE - 1 or _coordp == [CENTER, CENTER, CENTER]:
189 #     _case = 2
190 # elif _fixed == 0 or _coordm == [CENTER, CENTER, CENTER]:
191 #     _case = 1
192 pgamma[_sub][_sup][i][j][k] = (gamma[_sup][i + _dx[_sub][0]][j +
    ↪ _dx[_sub][1]][k + _dx[_sub][2]]
193 - gamma[_sup][i - _dx[_sub][0]][j - _dx[_sub][1]][k - _dx[_sub][2]]) /
    ↪ dx / 2
194 # elif _case == 1:
195 #     pgamma[_sub][_sup][i][j][k] = (2 * gamma[_sup][i + 2 *
    ↪ _dx[_sub][0]][j + 2 * _dx[_sub][1]][k + 2 * _dx[_sub][2]]
196 #                                    - gamma[_sup][i + 3 *
    ↪ _dx[_sub][0]][j + 3 * _dx[_sub][1]][k + 3 * _dx[_sub][2]]
197 #                                    - 2 * gamma[_sup][i][j][k]
198 #                                    + gamma[_sup][i +
    ↪ _dx[_sub][0]][j + _dx[_sub][1]][k + _dx[_sub][2]]) / dx / 2
199 # else:
```

```
200 #     pgamma[_sub][_sup][i][j][k] = (- 2 * gamma[_sup][i - 2 *
    ↪ _dx[_sub][0]][j - 2 * _dx[_sub][1]][
201 #                                         k - 2 * _dx[_sub][2]]
202 #                                     + gamma[_sup][i - 3 *
    ↪ _dx[_sub][0]][j - 3 * _dx[_sub][1]][
203 #                                         k - 3 * _dx[_sub][2]]
204 #                                     + 2 * gamma[_sup][i][j][k]
205 #                                     - gamma[_sup][i -
    ↪ _dx[_sub][0]][j - _dx[_sub][1]][
206 #                                         k - _dx[_sub][2]]) / dx /
    ↪ 2
207
208 if initial:
209 with open('Pgamma.txt', 'wb') as f:
210 pickle.dump(pgamma, f)
211
212 print('Get Christoffel')
213 Christoffel = [[[[[0] * SIZE for _i in range(SIZE)] for _j in
    ↪ range(SIZE)] for _k in range(6)] for _l in range(3)]
214 for i in range(SIZE):
215 print('Christoffel for : ', i)
216 for j in range(SIZE):
217 for k in range(SIZE):
218 if abs(i - CENTER) + abs(j - CENTER) + abs(k - CENTER) <= 1:
219 continue
220 if i == 0 or i == SIZE - 1 or j == 0 or j == SIZE -1 or k == 0 or k ==
    ↪ SIZE - 1:
221 continue
222
223 for _sup in range(3):
224 for _sub in range(6):
225 # _subgamma = [[gamma[0][i][j][k], gamma[1][i][j][k],
    ↪ gamma[2][i][j][k]],
```

```
226 #                [gamma[1][i][j][k], gamma[3][i][j][k],
    ↪  gamma[4][i][j][k]],
227 #                [gamma[2][i][j][k], gamma[4][i][j][k],
    ↪  gamma[5][i][j][k]]]
228 # _gammainv = getMatrixInverse(_subgamma)
229 _subi, _subj = to_ij[_sub]
230 Christoffel[_sup][_sub][i][j][k] = sum(
231 gamma_inv[to_ind(l, _sup)][i][j][k] * (pgamma[_subi][to_ind(l,
    ↪  _subj)][i][j][k] +
232 pgamma[_subj][to_ind(_subi, l)][i][j][k] -
233 pgamma[l][to_ind(_subi, _subj)][i][j][k]) for l in
234 range(3)) / 2
235 if initial:
236 with open('Christoffel.txt', 'wb') as f:
237 pickle.dump(Christoffel, f)
238
239 print('Get Ricci Tensor')
240
241 def pChristoffel(sup, sub, i, j, k, x):
242 # _case = 0
243 # _coord = [i, j, k]
244 # _fixed = _coord[x]
245 # _coordp = _coord[:]
246 # _coordp[x] += 1
247 # _coordm = _coord[:]
248 # _coordm[x] -= 1
249 # if _fixed == SIZE - 1 or _coordp == [CENTER, CENTER, CENTER]:
250 #     _case = 2
251 # elif _fixed == 0 or _coordm == [CENTER, CENTER, CENTER]:
252 #     _case = 1
253
254 # if _case == 0:
```

```
255 return (Christoffel[sup][sub][i + _dx[x][0]][j + _dx[x][1]][k +
    ↪ _dx[x][2]] - Christoffel[sup][sub][i - _dx[x][0]][j - _dx[x][1]][k
    ↪ - _dx[x][2]]) / dx / 2
256 # elif _case == 1:
257 #     return (2 * Christoffel[sup][sub][i + 2 * _dx[x][0]][j + 2 *
    ↪ _dx[x][1]][k + 2 * _dx[x][2]]
258 #             - Christoffel[sup][sub][i + 3 * _dx[x][0]][j + 3 *
    ↪ _dx[x][1]][k + 3 * _dx[x][2]]
259 #             - 2 * Christoffel[sup][sub][i][j][k]
260 #             + Christoffel[sup][sub][i + _dx[x][0]][j + _dx[x][1]][k +
    ↪ _dx[x][2]])
261 # else:
262 #     return (- 2 * Christoffel[sup][sub][i - 2 * _dx[x][0]][j - 2 *
    ↪ _dx[x][1]][k - 2 * _dx[x][2]]
263 #             + Christoffel[sup][sub][i - 3 * _dx[x][0]][j - 3 *
    ↪ _dx[x][1]][k - 3 * _dx[x][2]]
264 #             + 2 * Christoffel[sup][sub][i][j][k]
265 #             - Christoffel[sup][sub][i - _dx[x][0]][j - _dx[x][1]][k -
    ↪ _dx[x][2]])
266
267
268
269 for i in range(SIZE):
270 print('Ricci Tensor for :', i)
271 for j in range(SIZE):
272 for k in range(SIZE):
273 if abs(i - CENTER) + abs(j - CENTER) + abs(k - CENTER) <= 2:
274 continue
275 # if i == j == k == CENTER or i == 0 or i == SIZE - 1 or j == 0 or j ==
    ↪ SIZE - 1 or k == 0 or k == SIZE - 1:
276 #     continue
277 if i <= 1 or i >= SIZE - 2 or j <= 1 or j >= SIZE - 2 or k <= 1 or k >=
    ↪ SIZE - 2:
278 continue
```

```
279
280 for _ind in range(6):
281 _i, _j = to_ij[_ind]
282 R[_ind][i][j][k] = sum(pChristoffel(_k, _ind, i, j, k, _k) for _k in
    ↪ range(3)) \
283 - sum(pChristoffel(_k, to_ind(_i, _k), i, j, k, _j) for _k in
    ↪ range(3))\
284 + sum(
285 Christoffel[_k][to_ind(_i, _j)][i][j][k] * Christoffel[_l][to_ind(_k,
    ↪ _l)][i][j][k] for _k in
286 range(3) for _l in range(3)) \
287 - sum(
288 Christoffel[_l][to_ind(_i, _k)][i][j][k] * Christoffel[_k][to_ind(_l,
    ↪ _j)][i][j][k] for _k in
289 range(3) for _l in range(3))
290
291 if errest:
292 print('Error est')
293 print('mean R')
294 meanR = [[[0] * SIZE for _i in range(SIZE)] for _j in range(SIZE)]
295 for i in range(SIZE):
296 for j in range(SIZE):
297 for k in range(SIZE):
298 if i == j == k == CENTER:
299 continue
300 meanR[i][j][k] = sum(
301 gamma_inv[to_ind(_i, _j)][i][j][k] * R[to_ind(_i, _j)][i][j][k] for _i
    ↪ in range(3) for _j in range(3))
302 print('KK')
303 KK = [[[0] * SIZE for _i in range(SIZE)] for _j in range(SIZE)]
304 for i in range(SIZE):
305 for j in range(SIZE):
306 for k in range(SIZE):
307 if i == j == k == CENTER:
```

```
308 continue
309 KK[i][j][k] = sum(sum(gamma_inv[to_ind(_i, _mu)][i][j][k] *
    ↪ gamma_inv[to_ind(_j, _nu)][i][j][k] *
310 K[to_ind(_mu, _nu)][i][j][k] for _mu in range(3) for _nu in range(3))
311 * K[to_ind(_i, _j)][i][j][k] for _i in range(3) for _j in range(3))
312 err = [[[0] * SIZE for _i in range(SIZE)] for _j in range(SIZE)]
313 for i in range(SIZE):
314 for j in range(SIZE):
315 for k in range(SIZE):
316 if i == j == k == CENTER:
317 continue
318 err[i][j][k] = meanR[i][j][k] + (meanK[i][j][k]) ** 2 - KK[i][j][k]
319 with open('error_%d.txt' % (tc), 'wb') as f:
320 pickle.dump(err, f)
321
322
323
324
325 # R[_ind][i][j][k] = sum((Christoffel[_k][_ind][i + _dx[_k][0]][j +
    ↪ _dx[_k][1]][k + _dx[_k][2]] -
326 #                         Christoffel[_k][_ind][i][j][k]) / dx for _k
    ↪ in range(3)) \
327 #                     - sum((Christoffel[_k][to_ind(_i, _k)][i +
    ↪ _dx[_j][0]][j + _dx[_j][1]][
328 #                                k + _dx[_j][2]] -
    ↪ Christoffel[_k][to_ind(_i, _k)][i][j][k]) / dx for _k in
329 #                           range(3)) \
330 #                     + sum(
331 #     Christoffel[_k][to_ind(_i, _j)][i][j][k] *
    ↪ Christoffel[_l][to_ind(_k, _l)][i][j][k] for _k in
332 #     range(3) for _l in range(3)) \
333 #                     - sum(
334 #     Christoffel[_l][to_ind(_i, _k)][i][j][k] *
    ↪ Christoffel[_k][to_ind(_l, _j)][i][j][k] for _k in
```

```
335 #     range(3) for _l in range(3))
336
337 # with open("gamma.txt", 'wb') as f:
338 #     pickle.dump(gamma, f)
339 # with open('gamma_inv.txt', 'wb') as f:
340 #     pickle.dump(gamma_inv, f)
341 # with open('pgamma.txt', 'wb') as f:
342 #     pickle.dump(pgamma, f)
343 # with open('K.txt', 'wb') as f:
344 #     pickle.dump(K, f)
345 # with open('meanK.txt', 'wb') as f:
346 #     pickle.dump(meanK, f)
347 # with open('R.txt', 'wb') as f:
348 #     pickle.dump(R, f)
349 # with open('Christoffel.txt', 'wb') as f:
350 #     pickle.dump(Christoffel, f)
351 #
352 if evolve:
353 print('Evolve')
354 _gamma = [[[[0] * SIZE for _i in range(SIZE)] for _j in range(SIZE)]
    ↪  for _k in range(6)]
355 _K = [[[[0] * SIZE for _i in range(SIZE)] for _j in range(SIZE)] for _k
    ↪  in range(6)]
356
357 for i in range(SIZE):
358 print('Evolve for :', i)
359 for j in range(SIZE):
360 for k in range(SIZE):
361 # if i == j == k == CENTER:
362 #     continue
363 # if i == 0 or i == SIZE - 1 or j == 0 or j == SIZE - 1 or k == 0 or k
    ↪  == SIZE - 1:
364 #     continue
```

```
365 if i <= 1 or i >= SIZE - 2 or j <= 1 or j >= SIZE - 2 or k <= 1 or k >=
    ↪ SIZE - 2:
366 for ind in range(6):
367 _gamma[ind][i][j][k] = gamma[ind][i][j][k]
368 _K[ind][i][j][k] = K[ind][i][j][k]
369 continue
370 r = dist(i - CENTER, j - CENTER, k - CENTER) * dx
371 if r <= r0:
372 for ind in range(6):
373 _gamma[ind][i][j][k] = gamma[ind][i][j][k]
374 _K[ind][i][j][k] = K[ind][i][j][k]
375 continue
376 for ind in range(6):
377 _i, _j = to_ij[ind]
378 _gamma[ind][i][j][k] = gamma[ind][i][j][k] - 2 * lapse(r) *
    ↪ K[ind][i][j][k] * dt
379 _K[ind][i][j][k] = K[ind][i][j][k] - dt * (-ddlapse(ind, r, i, j, k) +
    ↪ lapse(r) * (R[ind][i][j][k]
380 + meanK[i][j][k] *
381 K[ind][i][j][
382 k] - 2 * sum(
383 K[to_ind(_i, _k)][i][j][k] * sum(
384 gamma_inv[to_ind(_k, _u)][i][j][k] * K[to_ind(_u, _j)][i][j][k] for _u
    ↪ in range(3)) for
385 _k in range(3))))
386
387 with open('gamma_%d.txt' % (tc + 1), 'wb') as f:
388 pickle.dump(_gamma, f)
389 with open('K_%d.txt' % (tc + 1), 'wb') as f:
390 pickle.dump(_K, f)
391 tc += 1
```

Bibliography

[1] Charles W Misner, Kip S Thorne, and John Archibald Wheeler. *Gravitation*. Macmillan, 1973.

[2] Bernd Brügmann. "Adaptive mesh and geodesically sliced Schwarzschild spacetime in 3+ 1 dimensions". In: *Physical Review D* 54.12 (1996), p. 7361.

[3] Miguel Alcubierre. *Introduction to 3+ 1 numerical relativity*. Vol. 140. OUP Oxford, 2008.

[4] Thomas W Baumgarte and Stuart L Shapiro. *Numerical relativity: solving Einstein's equations on the computer*. Cambridge University Press, 2010.

[5] Robert M Wald. *General relativity*. University of Chicago press, 2010.

[6] Eric Gourgoulhon. *3+ 1 formalism in general relativity: bases of numerical relativity*. Vol. 846. Springer Science & Business Media, 2012.

[7] Anthony Zee. *Einstein gravity in a nutshell*. Vol. 14. Princeton University Press, 2013.

[8] WMHP Vindana and KAIL Wijewardena Gamalath. "Simulation of rotating black holes". In: *World Scientific News* 114 (2018), pp. 106–125.

[9] Bernard Schutz. *A first course in general relativity*. Cambridge university press, 2022.