UNIVERSITY OF TARTU

Faculty of Science and Technology

Institute of Physics

Anna Aader

THE COMPUTATION OF TENSORS IN GENERAL RELATIVITY USING THE PYTHON PACKAGE SYMPY

Bachelor's thesis (6 ECTS)

Physics

Supervisor:
Manuel Hohmann, PhD

Tartu 2020

The Computation of Tensors in General Relativity Using the Python Package SymPy

Additional functions are coded for the differential geometry module diffgeom of Python's symbolic computation package SymPy, in order to compute various tensors and derivatives used in general relativity. Ricci tensors and scalars, as well as the Kretschmann scalars are computed using these new functions for the Taub-NUT metric in hyperspherical coordinates and the Gödel type metric in cylindrical coordinates.

Keywords: differential geometry, tensor calculus, general relativity, computer algebra, Python
CERCS: P190 — Mathematical and general theoretical physics, classical mechanics, quantum mechanics, relativity, gravitation, statistical physics, thermodynamics

Üldrelatiivsusteooria tensorite arvutamine Python-i SymPy paketti kasutades

Python-i sümboolarvutus paketi SymPy diferentsiaalgeomeetria mooduli diffgeom jaoks programmeeriti uued funktsioonid üldrelatiivsusteoorias kasutatavate tensorite ja tuletiste arvutamiseks. Arvutati nii Ricci tensorid ja skalaarid, kui ka Kretschmanni skalaarid Taub-NUT meetrika jaoks hüpersfäärilistes koordinaatides ja Gödeli tüüpi meetrika jaoks silindrilistes koordinaatides.

Märksõnad: diferentsiaalgeomeetria, tensorite diferentsiaalarvutus, üldrelatiivsusteooria, arvuti algebra, Python
CERCS: P190 — Matemaatiline ja üldine teoreetiline füüsika, klassikaline mehhaanika, kvantmehhaanika, relatiivsus, gravitatsioon, statistiline füüsika, soojusteadus

Contents

Introduction

In order to extend the theory of electromagnetism for stationary frames to moving frames, Einstein raised a postulate, namely the principle of relativity, in which Maxwell's equations must have the same form regardless of the frame of reference. Einstein showed, that under this postulate the resulting "Lorentz invariance" implies, that the speed of light in a vacuum is definite and independent of the movement of the emitter.[1]

Einstein then went on to generalize his theory of relativity further to non-inertial reference frames. This new general theory of relativity makes use of fundamental concepts in differential geometry, the mathematics of which was already in development for quite some time before Einstein found a physical application. This leads to the notion of a curved spacetime, which is necessary to preserve Lorentz invariance in non-inertial frames.[2]

The importance of studying and working with general relativity is seen through its use in the Global Positioning System (GPS). While effects that come into play in general relativity are not notable in day to day life on earth, on the scale of satellites working together with objects on the surface of the planet the effects cease to be negligible. Not accounting for them can lead to positions being miscalculated by kilometers instead of meters.[3]

In order for the equations describing laws to maintain their form regardless of the coordinate system being used, tensors, which can simply be described as a collection of n dimensional arrays, were taken into use. While it is possible to compute tensors by hand, it is more time efficient to make use of programs.

Currently, Python is one of the most popular programming languages[4] and has packages such as SymPy, NumPy, and Matplotlib, which make Python a solid choice for scientific computation[5]. It is for this reason, as well as the possibility of creating new packages or extensions to existing packages, that Python is used by CERN (European Organization for Nuclear Research)[6] and many others.

The package SymPy allows for symbolic computation within Python and enables algebra, calculus, discrete mathematics, and so on. While it has a differential geometry module and can

handle the symbolic computation of tensors, it is lacking in many areas when it comes to finding results in general relativity.[7]

Chapter 1

Theory

Spacetime can be described as a 4 dimensional manifold equipped with a metric (distance between points), which is described by a symmetric rank-2 tensor. Such a manifold is called a Pseudo-Riemannian manifold.

The metric defines how the spacetime curves, and is therefore used to find the curvature and torsion of the space, as well as a guide for parallel transport on the given space (a connection).

Solving Einstein Field Equations (EFE), which equate the curvature of the spacetime to the energy and stress of the matter in it, leads to various metrics that depend on the properties of the spacetime (expanding, contracting, static, homogeneous, isotropic, *et cetera*).

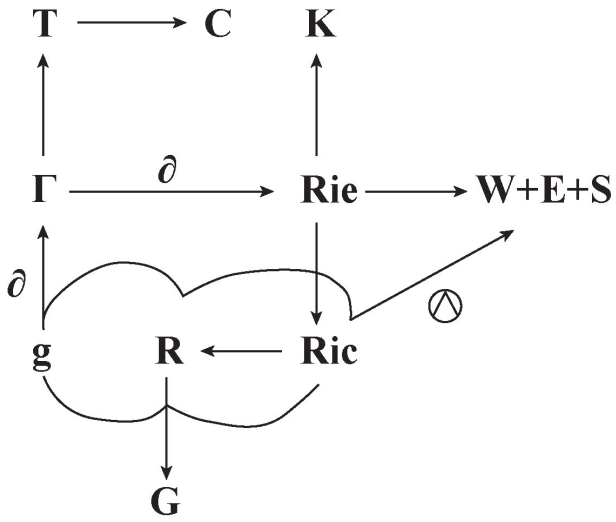

Figure 1.1: Relations between common tensors in general relativity

The metric tensor g, as seen in figure 1.1, is used to define the connection Γ using the partial derivative ∂, which defines both the torsion T and Riemann curvature tensor Rie. The contorsion

tensor C is a variant of the torsion tensor T.

The Riemann curvature tensor can be used to find two scalars describing the curvature of the spacetime: the Ricci scalar R and the Kretschmann scalar K. The Ricci scalar is the trace of the Ricci tensor Ric, which is the contraction of the first and third indices of the Riemann tensor.

The Einstein tensor G is a combination of the Ricci tensor, Ricci scalar, and the metric.

The Ricci decomposition $W + E + S$ is defined using the Kulkarni-Nomizu product $\owedge$ of the metric and itself, the metric and the Ricci tensor in conjunction with the Ricci scalar, and the Riemann curvature tensor.

1.1 Coordinate systems

The most well-known coordinate system is the Cartesian coordinate system, in which the *origin* is the intersection of n perpendicular lines. The location of a point is given by how far along the line its projection onto the line is from the origin. A 3 dimensional space with cartesian coordinates can be expressed as $\mathbb{R}^3$.

The spacetime $\mathbb{R} \times \mathbb{R}^3$ equipped with the metric

$$ds^2 = \mp dt^2 \pm d\Sigma^2 \,, \tag{1.1}$$

where $d\Sigma^2$ represents the sum of all spatial directions, is called the Minkowski spacetime $\mathbf{M}^4$. The notion of *distance* invariant under Lorentz transformation measured on the Minkowski spacetime treats the temporal direction differently. These flat spacetimes (R^3 and $\mathbf{M}^4$) serve as *local charts* of the curved spacetime manifold.

In general relativity the first (zeroth) dimension is reserved for time and can be seen as an add-on to the spatial coordinates ($\mathbb{R} \times \mathbb{R}^3$).

The spherical coordinate system has a radial component r, which can have values in $[0, \infty)$, along which the distance from the origin is defined. Rotating the radial axis about its origin covers a plane: the change in the position of the axis is measured by the polar angle θ.

The last dimension is created by rotating this 2 dimensional plane defined through r and θ about itself so that the normal of the plane is the normal of the radial axis at every position. This change is measured by the azimuthal angle ϕ. Conventionally, this is related to the Cartesian coordinate system by defining the azimuthal angle ϕ as the one between the x and y axis and the polar angle θ between the z axis and the xy plane.

The azimuth can have values in $[0, 2\pi)$ and the polar angle in $[0, \pi)$. Cartesian coordinates can be expressed through spherical coordinates as

$$\begin{cases} t = t \\ x = r\sin\theta\cos\phi \\ y = r\sin\theta\sin\phi \\ z = r\cos\theta \ , \end{cases}$$

where $t \in \mathbb{R}$ is time, which rarely undergoes a transformation during a change in the basis.

Cylindrical coordinates are similar to spherical coordinates, except there is only one angular coordinate ψ. The three non angular coordinates are time $t \in \mathbb{R}$, height $z \in \mathbb{R}$, and the radius $\rho \in [0, \infty)$

$$\begin{cases} t = t \\ x = \rho\cos\psi \\ y = \rho\sin\psi \\ z = z \ . \end{cases}$$

Though unconventional, in order to include the Taub-NUT metric which uses a $\mathbb{R} \times S^3$ coordinate system, 4 dimensional "hyperspherical" coordinates will be used as well. They are set up based on the definition for an n dimensional spherical coordinate system

$$\begin{cases} x_1 = r\cos\phi_1 \\ x_2 = r\sin\phi_1\cos\phi_2 \\ x_3 = r\sin\phi_1\sin\phi_2\cos\phi_3 \\ \vdots \\ x_n = r\sin\phi_1\cdots\sin\phi_{n-2}\sin\phi_{n-1} \ . \end{cases}$$

Unlike the first three coordinate systems, in hyperspherical coordinates the 'time' is represented by radial time t' which, when expressed using spherical coordinates, turns out to be the length of the vector (t, r), where t is time and r is the radius.

The cartesian coordinates, as expressed using the hyperspherical coordinate system, are

$$\begin{cases} t = t' \cos\psi \\ x = t' \sin\psi \cos\theta \\ y = t' \sin\psi \sin\theta \cos\phi \\ z = t' \sin\psi \sin\theta \sin\phi \ . \end{cases}$$

1.2 Covariance and contravariance

For now let us work in flat spacetimes, or locally in a curved spacetime. A basis is a minimal set of elements in a vector space over $\mathbb{R}$, through which all other elements can be expressed by means of some linear combination. The coefficients of the elements in the basis make up the *coordinates* of a given element (point). A transformation from one basis to another can be expressed using an invertible matrix

$$e = Ae' \ , \tag{1.2}$$

where

$$e = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}, \ e' = \begin{pmatrix} e'_1 \\ e'_2 \end{pmatrix}, \tag{1.3}$$

are the given bases in 2 dimensional vector space and

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \tag{1.4}$$

is the transformation matrix. If the bases have certain properties, such as orthonormality, these properties carry over to the transformation matrix reflecting the change in basis.

Tangent vectors are one example of the above described elements which can be expressed as coordinates based on a basis. They are called contravariant, because in the case of a basis change they are multiplied by the inverse of the transformation matrix A.

Given a finite dimensional vector space V, there is a corresponding dual vector space V^* which consist of linear functionals from V to $\mathbb{R}$. The choice of a basis e_i in V defines a dual basis e_i^* in V^* for which $e_i^*(e_j) = \delta_{ij}$. The dual space of the tangent space is the cotangent space, which transforms covariantly.

Elements in the dual basis, also called cotangent vectors or covectors, transform from one basis

to another by being multiplied by the transformation matrix A. Covectors are therefore called covariant.

Using Einstein's notation, tangent vectors are expressed as

$$\boldsymbol{v} = v^i \boldsymbol{e}_i \,, \tag{1.5}$$

where v^i are the coordinates that change *contrary* to how the basis changes. This is brought to attention by the upper indice i, as opposed to the basis and covariant components which use lower indices

$$\boldsymbol{v} = v_i e^i \text{ , where } \; e^i = e^*_i \,. \tag{1.6}$$

In Einstein's notation whenever indices are repeated, with one index covariant and the other contravariant, there is an implied summation sign running through each dimension. Equation 1.5 can be expanded into

$$\boldsymbol{v} = \sum_i^n v^i \boldsymbol{e}_i = v^1 \boldsymbol{e}_1 + ... + v^n \boldsymbol{e}_n \,, \tag{1.7}$$

where n is the number of dimensions.

1.3 Manifolds

A Hausdorff topological space is an open set with an open, empty subset that conserves openness of subsets for their infinite intersections, while also being locally Euclidian. Any two points have disjoint open neighbourhoods.[8]

A topological manifold is a second-countable Hausdorff topological space equipped with a continuous atlas: all open sets map homeomorphically into flat space. Smooth manifolds are equipped with a differentiable atlas.[9]

In general relativity the most commonly used manifold is called the pseudo-Riemannian manifold, which is smooth and equipped with a non-degenerate metric. However, pseudo-Riemannian manifolds are not positive-definite like Riemannian manifolds, as either spacelike or timelike vectors must have negative form.

1.4 Metrics

The metric as a function shows the distance between two points, and therefore how the spacetime curves. The metric is often denoted by g in general relativity.

As an example, the metric of the Minkowski spacetime $\mathbb{R}^{1,3}$ with Cartesian coordinates is

$$g_{ij}dx^i dx^j = -dt^2 + dx^2 + dy^2 + dz^2 \ . \tag{1.8}$$

The coefficients of the various tensor products of the infinitesimal changes make up the metric tensor

$$g = g_{\mu\nu}dx^\mu \otimes dx^\nu \ . \tag{1.9}$$

The metric is often derived from the line element ds^2 or $-c^2d\tau^2$. When it comes to line elements, commutativity holds for infinitesimal changes: $dx^i dx^j = dx^j dx^i$. For this reason, when deriving the metric from the line element, the components where $j \neq i$ must be expanded into

$$2dx^i dx^j = \left(dx^i dx^j + dx^j dx^i\right) , \tag{1.10}$$

which forces the metric tensor $g_{\mu\nu}$ to be symmetric.

1.4.1 Friedmann–Lemaître–Robertson–Walker (FLRW) spacetime

One exact solution to EFE is the FLRW spacetime, which describes a universe that is isotropic (uniform in all directions), homogeneous (all points in this spacetime have the same properties), and either expanding, contracting, or static.[10] In other words, it describes a maximally symmetric space characterized by its mean curvature.

The line element is expressed as

$$-c^2d\tau^2 = -c^2dt^2 + a(t)^2 d\Sigma^2 \ , \tag{1.11}$$

in which $a(t)$ is known as the scale factor, and the non-time-dependent $d\Sigma^2$ represents the 3 spatial components of the spacetime. In a flat spacetime, $d\Sigma^2$ can be expressed using Cartesian coordinates as

$$d\Sigma^2 = dx^2 + dy^2 + dz^2 \ , \tag{1.12}$$

but in non-flat cases reduced circumference polar coordinates

$$d\Sigma^2 = \frac{dr^2}{1-kr^2} + r^2 d\theta^2 + r^2 \sin^2\theta d\phi^2 \ , \tag{1.13}$$

where k represents the curvature, can be used, though when $k > 0$ only half of the 3-sphere is covered, which is why hyperspherical coordinates are sometimes used

$$d\Sigma^2 = dr^2 + S_k(r)^2 d\theta^2 + S_k(r)^2 \sin^2\theta d\phi^2 \,, \tag{1.14}$$

where

$$S_k(r) = \begin{cases} \sqrt{k}^{-1} \sin r\sqrt{k} \,, & k > 0 \\ r \,, & k = 0 \\ \sqrt{|k|}^{-1} \sinh r\sqrt{|k|} \,, & k < 0 \end{cases} . \tag{1.15}$$

Using different coordinate systems can remove singularities borne of the coordinate system (as opposed to physical singularities such as the origin in a black hole metric).

1.4.2 Gödel spacetime

The Gödel metrics are a type of solution to the Einstein field equations, which describe incoherent matter with rotation.[11] The original form of the Gödel metric is

$$g = a^2(-dt^2 + dx^2 + dz^2 - \frac{e^{2x}}{2}dy^2 - e^x(dtdy + dydt) \,, \tag{1.16}$$

where a is a constant parameter defined later as $(\sqrt{2}\omega)^{-1}$ with ω being the angular velocity at which the Gödel spacetime rotates on an axis (often the y axis). The original metric can be expressed using cylindrical coordinates as

$$g = 4a^2(-dt^2 + d\rho^2 + dz^2 - \sqrt{2}\sinh\rho(dtd\psi + d\psi dt) - (\sinh^2\rho(\sinh^2\rho - 1))d\psi^2) \,. \tag{1.17}$$

The curve $\{(t = t_0,\ r = r_0\ , z = z_0, \phi \in [0, 2\pi])\}$ is closed and timelike when $r_0 > \ln(1+\sqrt{2})$, and closed null when $r_0 = \ln(1+\sqrt{2})$.[12]

Gödel type metrics are a generalization of the original Gödel metric. The line element

$$g = dt^2 - d\rho^2 - dz^2 + H(\rho)(dtd\psi + d\psi dt) + (H^2(\rho) - D^2(\rho))d\psi^2 \,, \tag{1.18}$$

is expressed using cylindrical coordinates. It was shown that the functions D and H of ρ satisfy

the following ordinary differential equations

$$\frac{D''}{D} = m^2 \,, \;\; \frac{H'}{D} = -2\omega \,, \tag{1.19}$$

for the Gödel type metric.[12]

While the parameter ω has a phyiscal significance as the angular velocity of the dust cloud, m remains undefined.

The Gödel type metrics can be classified by the solutions D, H of the ODEs in equation 1.19 based on the relation between the parameters and 0. There are therefore four classes of Gödel type metrics.

1. $m^2 > 0,\ \omega \neq 0$

$$H = \frac{2\omega(1-\cosh mr)}{m^2},\ D = \frac{\sinh mr}{m} \tag{1.20}$$

2. $m^2 = 0,\ \omega \neq 0$

$$H = -\omega r^2,\ D = r \tag{1.21}$$

3. $-\mu^2 \equiv m^2 < 0,\ \omega \neq 0$

$$H = \frac{2\omega(1-\cos \mu r)}{\mu^2},\ D = \frac{\sin \mu r}{\mu} \tag{1.22}$$

4. $m^2 \neq 0,\ \omega = 0$

$$H = 0,\ D = \frac{\sin \mu r}{\mu} \text{ or } D = \frac{\sinh mr}{m} \,. \tag{1.23}$$

The choice in the 4th type, referred to as the class of degenerated Gödel type manifolds, for the value of D depends on whether $m^2 > 0$ or $m^2 < 0$, analogous to types 1 and 3.

In the case of both parameters being zero, the metric as seen in equation 1.18 reduces to a Minkowskian metric. When $m^2 = 2\omega^2$, the metric becomes the original Gödel metric, which does not conserve the causality principle.[13]

It was found, that the amount of causal and noncausal regions depend on how the two parameters (m^2, ω) are defined. When $m^2 \geq 4\omega^2$, there are no closed timelike curves.[12][14]

1.4.3 Taub-NUT spacetime

The Taub-NUT spacetime is another exact solution to EFE, one seen as an generalization of the Schwarzschild solution (which can be used to estimate slowly rotating objects). Both have

a mass parameter m, however Taub-NUT has the additional parameter l, which is commonly referred to as the magnetic mass or the gravito-mangetic monopole moment.[15] The metric is expressed as

$$g = -\frac{dt'^2}{U(t')} + 4l^2 U(t')(d\psi + \cos\theta d\phi)^2 + (t'^2 + l^2)(d\theta^2 + \sin^2\theta d\phi)^2 \,, \tag{1.24}$$

where the function $U(t')$ depends on both mass parameters l, m and the radial time t'

$$U(t') = \frac{2mt' + l^2 - t'^2}{t'^2 + l^2} \,. \tag{1.25}$$

1.5 Covariant derivative

The covariant derivative was built on three principles: linearity, the Leibniz rule, and its reduction to a directional derivative in flat space. These therefore generalize the notion of parallel transport in curved spaces, called a connection.

It is possible to locally derive the form of the covariant derivative

$$\nabla_\mu V^\nu = \partial_\mu V^\nu + \Gamma^\nu_{\lambda\mu} V^\lambda \,. \tag{1.26}$$

The covariant derivative of a tensor with a covariant index can be derived from the following two, new properties

$$\text{commutativity of contractions } \nabla_\mu (T^\lambda{}_{\lambda\rho}) = (\nabla T)_\mu{}^\lambda{}_{\lambda\rho} \,, \tag{1.27}$$

$$\text{reduction to the partial derivative when used with scalars } \nabla_\mu \phi = \partial_\mu \phi \,, \tag{1.28}$$

which shows, that the covariant tensor is also parallel transported by the Levi-Civita connection, except with the opposite sign. From here the formula for the covariant derivative follows

$$\nabla_\rho T^{\nu_1 \ldots \nu_n}_{\mu_1 \ldots \mu_m} = \partial_\rho T^{\nu_1 \ldots \nu_n}_{\mu_1 \ldots \mu_m} \tag{1.29}$$

$$+\Gamma^{\nu_1}_{\rho\lambda} T^{\lambda \ldots \nu_n}_{\mu_1 \ldots \mu_m} + \ldots + \Gamma^{\nu_1}_{\rho\lambda} T^{\nu_1 \ldots \lambda}_{\mu_1 \ldots \mu_m} \tag{1.30}$$

$$-\Gamma^{\nu_1}_{\rho\lambda} T^{\nu_1 \ldots \nu_n}_{\lambda \ldots \mu_m} - \ldots - \Gamma^{\nu_1}_{\rho\lambda} T^{\nu_1 \ldots \nu_m}_{\mu_1 \ldots \lambda} \,. \tag{1.31}$$

Solving the following system of equations, while assuming the symmetry in the covariant indices of the Levi-Civita connection as well as torsion-freeness,

$$\nabla_\rho g_{\mu\nu} = \partial_\rho g_{\mu\nu} - \Gamma^\lambda_{\rho\mu} g_{\lambda\nu} - \Gamma^\lambda_{\rho\nu} g_{\mu\lambda} = 0\,, \tag{1.32}$$

$$\nabla_\mu g_{\nu\rho} = \partial_\mu g_{\nu\rho} - \Gamma^\lambda_{\mu\nu} g_{\lambda\rho} - \Gamma^\lambda_{\mu\rho} g_{\nu\lambda} = 0\,, \tag{1.33}$$

$$\nabla_\nu g_{\rho\mu} = \partial_\nu g_{\rho\mu} - \Gamma^\lambda_{\nu\rho} g_{\lambda\mu} - \Gamma^\lambda_{\nu\mu} g_{\rho\lambda} = 0\,, \tag{1.34}$$

leads to the formula

$$\Gamma^\sigma_{\mu\nu} = \frac{1}{2} g^{\sigma\rho} (\partial_\mu g_{\nu\rho} + \partial_\nu g_{\rho\mu} - \partial_\rho g_{\mu\nu})\,, \tag{1.35}$$

for the coefficients of the connection $\Gamma^\sigma_{\mu\nu}$, which are called Christoffel coefficients (of the second kind, the first kind is the completely covariant form of the Christoffel coefficients second kind).[16]

The coefficients of a connection are not those of a tensor; they are made in a way such that 1.26 transforms as a tensor, but the connection itself should not be treated as one.

1.6 Curvature tensors

Parallel transporting a vector in flat space through some closed path will not change the direction of the vector, but this does not hold in curved space, which is why the amount the direction of a vector changes given an infinitesimal loop can be used to describe the curvature of a manifold.

Let $p_{\xi_t} : T_{\xi_0} M \to T_{\xi_t} M$ be a parallel transport map along a curve ξ_t on a Riemannian manifold M. For any vector field Y defined on the curve, the covariant derivative can be expressed through the map p_{ξ_t} by

$$\nabla_{\dot{\xi}_0} = \lim_{k\to 0} \frac{1}{k} \left(Y_{\xi_0} - p^{-1}_{\xi_k}(Y_{\xi_k})\right) = \frac{d}{dt}\left(p_{\xi_t} Y\right)\Big|_{t=0}\,. \tag{1.36}$$

Given two commuting vector fields X and Y, which both generate groups of diffeomorphisms with single parameters for some neighbourhood of ξ_0. The parallel transports along the flows of X and Y for some time t can then be denoted by p_{tX} and p_{tY}.

Taking some vector $Z \in T_{\xi_0} M$ and performing parallel transport on it around a quadrilateral with sides yY and xX as seen in figure 1.2, the vector at the origin after it has been parallel transported

around a closed loop is given by

$$Z_{xy} = p_{xX}^{-1} p_{yY}^{-1} p_{xX} p_{yY} Z \,. \tag{1.37}$$

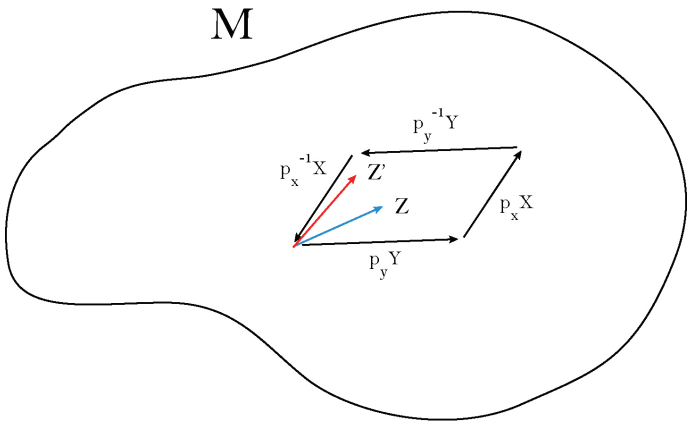

Figure 1.2: A vector Z parallel transported along a quadrilateral with sides yY and xX on a manifold M

The Riemann curvature tensor is the coefficient $p_{xX}^{-1} p_{yY}^{-1} p_{xX} p_{yY}$ for the change in the vector Z in case of an infinitesimal loop $(x, y \to 0)$

$$\frac{d}{dx}\frac{d}{dy} p_{xX}^{-1} p_{yY}^{-1} p_{xX} p_{yY}\Big|_{x=y=0} = [\nabla_X, \nabla_Y] - \nabla_{[X,Y]} = R(X,Y) \,. \tag{1.38}$$

By using the definition 1.26 for the covariant derivative and inserting it into the equation describing the curvature tensor, it can be derived that the components of the Riemann curvature tensor are defined by

$$R^{\rho}_{\sigma\mu\nu} = \partial_\mu \Gamma^{\rho}_{\nu\sigma} - \partial_\nu \Gamma^{\rho}_{\mu\sigma} + \Gamma^{\rho}_{\mu\lambda}\Gamma^{\lambda}_{\nu\sigma} - \Gamma^{\rho}_{\nu\lambda}\Gamma^{\lambda}_{\mu\sigma} \,. \tag{1.39}$$

At times, the fully covariant form of the Riemann curvature tensor is more convenient to use

$$R_{\rho\sigma\mu\nu} = g_{\rho\zeta} R^{\zeta}_{\sigma\mu\nu} \,. \tag{1.40}$$

Contracting the the first and third indices of the (1,3) Riemann tensor gives the (0,2) Ricci curvature tensor

$$R_{\sigma\nu} = R^{\mu}_{\sigma\mu\nu} \,, \tag{1.41}$$

which is characterized by how much the volume of a ball is deformed while moving along geodesics in a given space compared to the same ball moving along geodesics in flat space.[16][17]

1.7 Scalar curvature

The Ricci scalar is the trace of the Ricci curvature tensor

$$R = \mathrm{tr}_g Ric = R^{\sigma}_{\sigma} = g^{\sigma\mu} R_{\sigma\mu} \ , \tag{1.42}$$

which in 2 dimensions gives twice the Gaussian curvature. More generally the Ricci scalar shows how much the volume of a small geodesic ball deviates from the volume of a standard ball in Euclidean space. Given positive scalar curvature at a point the volume of the ball would be smaller than that of a ball with the same radius in Euclidean space; the reverse applies when the curvature is negative.

The Kretschmann scalar is a complete contraction of the Riemann curvature tensor on itself

$$K = R_{\rho\sigma\mu\nu} R^{\rho\sigma\mu\nu} \ , \tag{1.43}$$

and can be interpreted as the magnitude of curvature. At times the Ricci scalar is zero, such as in the case of the Schwarzschild solution, while the Kretschmann scalar is not, making it vital in showing how spacetime curves in that solution.[18][19]

Scalars are useful as they are invariant under coordinate transformations and therefore give information about a metric that does not depend on the chosen coordinate system, such as concerning singularities.

1.8 Torsion

As with curvature, the definition of torsion starts from the infinitesimal parallel transport of a vector: a point $P \in M$ is parallel transported by infinitesimal vectors $X, Y \in T_p M$, the parallel transports along those two vectors, as before, are given by p_{xX} and p_{yY}.

In a non-flat space $p_{xX} p_{yY} P$ and $p_{yY} p_{xX} P$ are not necessarily equal: the displacement between them is defined as the torsion[17]

$$\frac{d}{dx}\frac{d}{dy}\left(p_{xX} p_{yY} P - p_{yY} p_{xX} P\right)\big|_{x=y=0} = \nabla_X Y - \nabla_Y X - [X,Y] \ . \tag{1.44}$$

The torsion tensor can be expressed as

$$T^{\sigma}_{\mu\nu} = \Gamma^{\sigma}_{\mu\nu} - \Gamma^{\sigma}_{\nu\mu} - \gamma^{\sigma}_{\mu\nu} \,, \tag{1.45}$$

where the last component can generally be left out, leaving the definition of the torsion tensor as the asymmetry in the two lower indices of the connection coefficients.[20]

The contorsion $C_{\sigma\mu\nu}$ is found by antisymmetrically cycling through the three indices of the torsion tensor

$$C_{\sigma\mu\nu} = \frac{1}{2}\big(T_{\sigma\mu\nu} - T_{\mu\nu\sigma} + T_{\nu\sigma\mu}\big) \,, \tag{1.46}$$

and is used to calculate the torsion-free connection coefficients

$$\overline{\Gamma}^{\sigma}_{\mu\nu} = \Gamma^{\sigma}_{\mu\nu} - C^{\sigma}_{\mu\nu} \,. \tag{1.47}$$

If Γ is the Levi-Civita connection, the torsion T has no nonzero components due to the property of symmetry in the second and third indices. It follows, that the contorsion C is null as well, meaning that the torsion-free connection coefficients are the same as the Levi-Civita connection coefficients. Unlike with curvature, there is no well known scalar that expresses the torsion of a connection.

1.9 Einstein tensor and the stress-energy tensor

The Einstein tensor is defined as

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R \,, \tag{1.48}$$

and is symmetric and divergenceless

$$\nabla^{\mu} G_{\mu\nu} = 0 \,. \tag{1.49}$$

The Einstein tensor allows for the EFE to be written in a shorter form

$$G_{\mu\nu} + \Lambda g_{\mu\nu} = \kappa T_{\mu\nu}, \;\; \kappa = \frac{8\pi G}{c^4} \,, \tag{1.50}$$

where Λ is the cosmological constant, which Einstein introduced to make a static universe possible. The solutions to equation 1.50 are metrics which describe a spacetime.

The stress-energy tensor $T_{\mu\nu}$, just like the Einstein tensor, is divergenceless and symmetric.

Equation 1.50 shows how the curvature of spacetime affects matter and how the matter, in turn, changes the curvature. The Ricci curvature tensor was chosen to define gravity due to its definition: when a ball moves along a geodesic and the volume decreases compared to a standard ball in Euclidian space, it is comparable to the ball being pulled together due to gravity.

1.10 Kulkarni-Nomizu product and Ricci decomposition

The Kulkarni-Nomizu product takes two symmetric (0,2) type tensors as the arguments and returns a (0,4) type tensor. It is defined as

$$h_{ab} \owedge k_{cd} = h_{ac}k_{bd} + h_{bd}k_{ac} - h_{ad}k_{bc} - h_{bc}k_{ad} \, . \tag{1.51}$$

One use for the Kulkarni-Nomizu product is checking whether or not a Riemannian manifold has constant sectional curvature. In the affermative case the equation

$$R_{\alpha\beta\mu\nu} = \frac{k}{2} g_{\alpha\beta} \owedge g_{\mu\nu} \, , \tag{1.52}$$

holds and the given spacetime has constant sectional curvature k.

Ricci decomposition is one way to split the Riemann curvature tensor, so that each part has specific properties, the most well known being the Weyl tensor $W_{\alpha\beta\mu\nu}$, which is the fully traceless part of the decomposition. The tensor $E_{\rho\sigma\mu\nu}$ is the semi-traceless part, and $S_{\rho\sigma\mu\nu}$ the scalar part

$$R_{\alpha\beta\mu\nu} = S_{\alpha\beta\mu\nu} + E_{\alpha\beta\mu\nu} + W_{\alpha\beta\mu\nu} \, . \tag{1.53}$$

While the decomposition makes up various parts of the Riemann tensor, it is called the Ricci decomposition due to the formulae for each part depending on the Ricci tensor and scalar.

The Weyl tensor is therefore defined as the difference between the Riemann curvature tensor, and the semi-traceless and the fully traceless parts of the decomposition

$$W_{\alpha\beta\mu\nu} = R_{\alpha\beta\mu\nu} - E_{\alpha\beta\mu\nu} - S_{\alpha\beta\mu\nu} \, , \tag{1.54}$$

which leads to the formula for the Weyl tensor

$$W_{\alpha\beta\mu\nu} = R_{\alpha\beta\mu\nu} - \frac{1}{n-2}\left(R_{\alpha\beta} - \frac{R}{n} g_{\alpha\beta}\right) \owedge g_{\mu\nu} - \frac{R}{2n(n-1)} g_{\alpha\beta} \owedge g_{\mu\nu} \, , \tag{1.55}$$

where n is the number of dimensions. The expanded and simplified form of the formula is

$$W_{\alpha\beta\mu\nu} = R_{\alpha\beta\mu\nu} - \frac{2}{n-2}\left(g_{\alpha[\mu}S_{\nu]\beta} - g_{\beta[\mu}S_{\nu]\alpha}\right) + \frac{2}{(n-1)(n-2)}Rg_{\alpha[\mu}S_{\nu]\beta}\,. \tag{1.56}$$

Contraction of the Weyl tensor over any two indices yields zero. Hermann Weyl showed, that the Weyl tensor measures the deviation of a manifold from conformal flatness.

A manifold is called conformally flat, if there exists a smooth function $f : \mathbb{R}^n \to M$, where the manifold M is n dimensional, so that f is a conformal transformation (the angles are preserved, but not necessarily the lengths).

1.11 Lie derivative

Let there be a vector field X that describes the velocity field of the flow on a manifold: for some velocity field u, X is locally $u \cdot \nabla$. The movement of quantities along the flow over 'time', like the movement of leaves on a river, is lie dragging. [9]

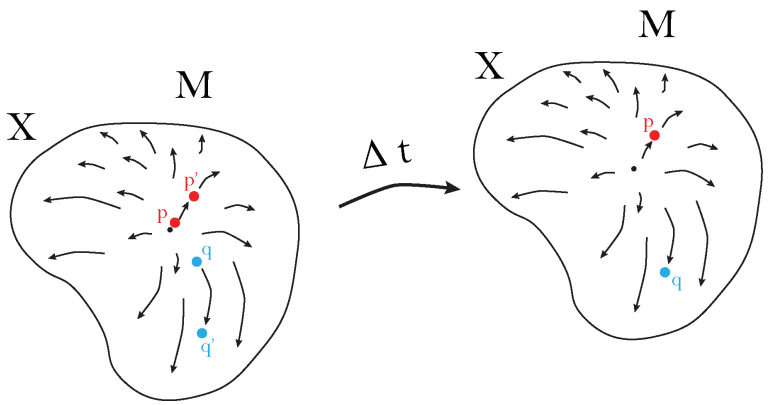

Figure 1.3: The movement of points p and q over some time Δt due to the vector field X

The Lie derivative is the difference between the original value at some point on the manifold and the new value given an infinitesimal change Δt in time

$$\mathscr{L}_X Y = [X,Y]\,. \tag{1.57}$$

For tensors in general relativity this becomes

$$\mathscr{L}_V T^{a_1\ldots a_n}_{b_1\ldots b_m} = V^c(\nabla_c T^{a_1\ldots a_n}_{b_1\ldots b_m}) \tag{1.58}$$

$$-(\nabla_c V^{a_1})T^{c\ldots a_n}_{b_1\ldots b_m} - \ldots - (\nabla_c V^{a_n})T^{a_1\ldots c}_{b_1\ldots b_m} \tag{1.59}$$

$$+(\nabla_{b_1} V^c)T^{a_1\ldots a_n}_{c\ldots b_m} + \ldots + (\nabla_{b_m} V^c)T^{a_1\ldots a_n}_{b_1\ldots c} \ . \tag{1.60}$$

At times the Lie derivative is calculated using the partial derivative in place of the covariant derivative. Vectors X that result in a trivial Lie derivative of the metric

$$\mathscr{L}_X g_{\mu\nu} = 0 \ , \tag{1.61}$$

are known as Killing vectors. Killing vectors can help simplify the matter of solving the partial differential equations when finding exact solutions to Einstein's field equations.[21]

Chapter 2

Methods

Functions that compute various tensors and derivatives in Python were made using the package for symbolic computation in Python (SymPy), specifically its modules for arrays (array) and differential geometry (diffgeom).

The differential geometry module already has methods to define manifolds (as well as two predefined manifolds in 2 and 3 dimensional space) and compute the metric based on the metric function (as opposed to the tensor), the Christoffel connections (both forms), the Riemann curvature tensor, as well as the Ricci curvature tensor. While functions exist to compute covariant and Lie derivatives, they do not work with tensors.

All tensors belong to the class `ImmutableDenseNDimArray`, chosen for its previous use in defining tensors, which stores all elements in the memory (dense) and cannot be changed once it has been defined (immutable).

Basic functions include the

- GR4 manifold
- Raising and lowering of indices: `raise_indices`, `lower_indices`
- Kulkarni-Nomizu product: `Kulkarni_Nomizu_product`
- Array simplifier: `simplify_array`
- Component printer: `print_components`

Added tensors that can be calculated with the metric as the input are

- Ricci scalar: `metric_to_Ricci_scalar`
- Kretschmann scalar: `metric_to_Kretschmann_scalar`

- Torsion tensor: `metric_to_torsion_components`
- Contorsion tensor: `metric_to_contorsion_components`
- Einstein tensor: `metric_to_Einstein_components`

Added derivatives

- Partial derivative: `partial_derivative`
- Covariant derivative: `covariant_derivative`
- Lie derivative: `Lie_derivative`

The code written can be seen in full in appendix B.

The FLRW metric was used to check the functions for mistakes, as the values of various tensors in FLRW spacetime are well known.

2.1 Basic functions

The first addition to diffgeom is a 4 dimensional manifold for general relativity (GR4), which has four coordinate systems defined on it: Cartesian (t,x,y,z), cylindrical (t,ρ,ψ,z), spherical (t,r,θ,ϕ), and hyperspherical (t',ψ,θ,ϕ), as outlined in chapter 1.1. Any of these can be used to define the metric, which will be the input for all functions. All components of a coordinate system are named in accordance to said coordinate system: `GR4_r.` for Cartesian (rectangular), `GR4_s.` for spherical, `GR4_c.` for cylindrical, and `GR4_h.` for hyperspherical.

```
TP = TensorProduct
k = symbols('k')
a = Function('a')(GR4_s.t)
FLRW = -TP(GR4_s.dt, GR4_s.dt) + \
        a^2*(TP(GR4_s.dr, GR4_s.dr)/(1-k*GR4_s.r^2) + \
        GR4_s.r^(2)*TP(GR4_s.dtheta, GR4_s.dtheta) + \
        sin(GR4_s.theta)^(2)*GR4_s.r^(2)*TP(GR4_s.dphi, GR4_s.dphi))
```

Listing 2.1: FLRW metric

Defining constants and functions used in the metric (such as the constant spatial curvature k and the scale factor $a(t)$) is done as in listing 2.1, using the core SymPy functions `symbols` and `Function`.

The `TensorProduct` function is a part of the diffgeom module and is used solely to define the metric. As for finding the product of two tensors, `tensorproduct` from the array module is used

instead.

Functions `lower_indices` and `raise_indices` lower and raise the indices of a tensor. Not using these functions may give outputs with the indices jumbled up due to how SymPy handles multiplying a tensor by the metric and then contracting it

$$g_{a_{i_1}c_{i_1}}\dots g_{a_{i_k}c_{i_k}}T^{a_1\dots a_s}{}_{b_1\dots b_r} = T_{c_{i_1}\dots c_{i_k}}{}^{a_1\dots a_{i_1-1}a_{i_1+1}\dots a_s}{}_{b_1\dots b_r}\,. \tag{2.1}$$

The functions for multiplying and contracting tensors, and therefore `raise_indices` and `lower_indices`, use tuples as the variable for the indices to be changed: in the case of a single index the tuple must be expressed as $(i,)$, as opposed to (i).

The Kulkarni-Nomizu product uses the function `permutedims` of the array module to switch around the "slots" of the (0,4) type tensor made up of the chosen two (0,2) type tensors. The `permutedims` function is also used in the raising and lowering of indices when unjumbling the order.

In SymPy it is possible to simplify expressions with the function `simplify(expr, ratio)`, where the ratio determines if the level of simplification is acceptable. In most cases it is not necessary to define the ratio, as the output is indeed a simpler form. This, however, does not work well with arrays. The function `simplify_array()` goes through the elements of an array one by one and simplifies the expressions within them, by first flattening the array into a vector and after simplification resetting the dimensions.

The function `print_components` eases the process of comparing components or locating a specific component: instead of having to count elements or seperately print specific elements of the array, this function prints the index of each element followed by the element itself. In order to print all possible permutations with the possibility of repeated elements, the permutations are based on a list with n of each dimension. After this, the list of indices is left with repeated indices, which is remedied by turning it into a set, then back into a list.

2.2 Derivatives

The partial derivative of a tensor $\partial_i T^{a_1\dots a_r}_{b_1\dots b_s}$ becomes a rank (r+s+1) object. When applying the partial derivative in every *direction* for each element the tensor does not automatically gain a rank: instead the dimension of the tensor is squared. This is solved by making the new tensor an array with the shape determined by the dimension of the manifold and the previous rank increased by one.

In the covariant derivative $\nabla_i T^{a_1\dots a_r}_{b_1\dots b_s}$ for each index in the tensor T the derivative gains an element,

with the sign depending on whether the index is contra- or covariant. Each additional element is a contraction of the tensor itself and the connection coefficients as seen in equation 1.29.

The positive elements are contractions of one of the covariant indices of the connection and each of the contravariant indices of the tensor T. Which one of the covariant indices of the Levi-Civita connection is contracted does not matter, as it is symmetric with respect to its last two indices but, in order to make the result accurate in case of a connection with torsion, the second lower index will be the one contracted. The order of indices in the tensor product becomes

$$(\Gamma T)^{i\ a_1\ldots a_r}_{\ jk\ \ \ b_1\ldots b_s}\,, \tag{2.2}$$

From which it is clear, that the contraction must take 2 as its first index and $\{3,4,...,2+r\}$ as the other. The indices for the negative elements are found to be 0 and $\{3+r,...,2+r+s\}$. The Lie derivative uses the same logic.

Chapter 3

Results

3.1 Computation speed

The computation of the tensors themselves does not take long: for the Gödel and FLRW metrics 3 seconds was the longest computation time. For other, more complex metrics, this took up to 4 minutes.

The component causing some metrics to take more than 12 hours to fully compute was the simplification process: SymPy has its own function for the simplification of symbolic equations, which is used in the new `simplify_array` function in order to apply it to every element. For the FLRW and Gödel metrics the simplification time did not exceed 10 minutes. On the other hand, finding, for example, the Kretschmann scalar of a black hole metric with a third parameter alongside the familiar mass m and magnetic mass l exceeded 24 hours.

3.2 Gödel spacetime

The metric used is the general Gödel type metric as seen in equation 1.18, expressed through SymPy as

```
H = Function('H')(GR4_c.rho)
D = Function('D')(GR4_c.rho)
G = TP(GR4_c.dt,GR4_c.dt) -\
    TP(GR4_c.drho,GR4_c.drho) +\
    H*(TP(GR4_c.dt,GR4_c.dpsi)+TP(GR4_c.dpsi,GR4_c.dt)) +\
    (H^2-D^2)*TP(GR4_c.dpsi,GR4_c.dpsi) -\
```

```
TP(GR4_c.dz,GR4_c.dz)
```

Listing 3.1: The form of the Gödel metric used to compute the results

The Ricci scalar was computed using the function `metric_to_Ricci_scalar`

$$R = \frac{4DD'' - H'^2}{2D^2} \,, \tag{3.1}$$

which, using equation 1.19, can be expressed in terms of ω and m as

$$R = 2m^2 - 4\omega^2 \,. \tag{3.2}$$

When finding the Kretschmann scalar, using the analogous `metric_to_Kretschmann_scalar` function, in terms of the two parameters

$$K = \frac{16D^2D''^2 - 8D^2H''^2 + 16DD'H'H'' - 24DD''H'^2 - 8D'^2H'^2 + 11H'^4}{4D^4} \,, \tag{3.3}$$

the components $-8D^2H''^2 + 16DD'H'H'' - 8D'^2H'^2$ can not be simplified immediately using equation 1.19. Instead one can use

$$H' = -2\omega D \iff H'' = -2\omega D' \,, \tag{3.4}$$

to find that the components cancel out

$$-2\frac{D^2H''^2}{D^4} + 4\frac{DD'H'H''}{D^4} - 2\frac{D'^2H'^2}{D^4} \tag{3.5}$$

$$= -2\frac{(-2\omega D')^2}{D^2} + 4\frac{(-2\omega)D'(-2\omega D')}{D^2} - 2\frac{D'^2(-2\omega)^2}{D^2} \tag{3.6}$$

$$= 0 \,. \tag{3.7}$$

This leads to the Kretschmann scalar being expressed as

$$K = 4(m^4 - 6m^2\omega^2 + 11\omega^4) \,. \tag{3.8}$$

When $m^2 = 2\omega^2$ the scalars simplify to

$$K = 4(4\omega^4 - 12\omega^4 + 11\omega^4) = 12\omega^4 \,, \tag{3.9}$$

$$R = 2\omega^2 - 4\omega^2 = -2\omega^2 \,, \tag{3.10}$$

which match the results found when deriving the scalars from the original Gödel metric.

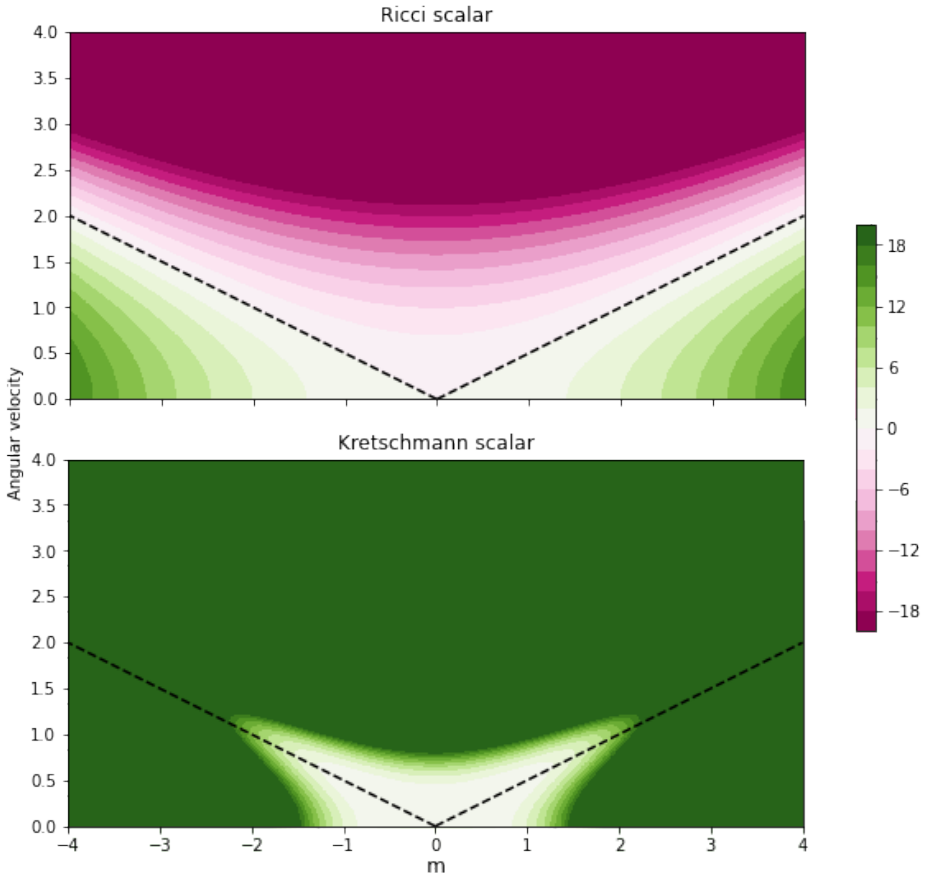

Figure 3.1: The dependence of the Ricci and Kretschmann curvature scalars on the parameters m and ω (angular velocity), with a black dotted line representing the limiting case $m^2 = 4\omega^2$ where the spacetime is completely homogenous

In the same way the non-zero Ricci tensor components take the form

$$R_{00} = 2\omega^2 \tag{3.11}$$
$$R_{11} = 2\omega^2 - m^2 \tag{3.12}$$
$$R_{02} = R_{20} = H(3\omega^2 - m^2) \tag{3.13}$$
$$R_{22} = 2\omega^2 H^2 + H'^2\left(\frac{1}{2} - m^2\right), \tag{3.14}$$

and the Einstein components take the form

$$G_{00} = 3\omega^2 - m^2 \tag{3.15}$$
$$G_{11} = \omega^2 \tag{3.16}$$
$$G_{02} = G_{20} = H(3\omega^2 - m^2) \tag{3.17}$$
$$G_{22} = H^2(3\omega^2 - m^2) + \frac{H'^2}{4} \tag{3.18}$$
$$G_{33} = m^2 - \omega^2 . \tag{3.19}$$

The original outputs of the functions are given in appendix A.

3.3 Taub-NUT spacetime

The Taub-NUT metric as, defined in equations 1.14 and 1.15, can be expressed in SymPy as

```
l,m = symbols('l m')
U =  (2*m*GR4_h.t+l**2-GR4_h.t**2)/(GR4_h.t**2 + l**2)
exr2 = - TP(GR4_h.dt,GR4_h.dt)/U + \
            4*l**2*U*(TP(GR4_h.dpsi,GR4_h.dpsi)+ \
            cos(GR4_h.theta)*TP(GR4_h.dpsi,GR4_h.dphi)+ \
                   cos(GR4_h.theta)*TP(GR4_h.dphi,GR4_h.dpsi)+ \
                   cos(GR4_h.theta)**2*TP(GR4_h.dphi,GR4_h.dphi)) + \
            (GR4_h.t**2+l**2)*(TP(GR4_h.dtheta,GR4_h.dtheta)+ \
            sin(GR4_h.theta)**2*TP(GR4_h.dphi,GR4_h.dphi))
```

Listing 3.2: The form of the Taub-NUT metric used to compute the results

Using listing 3.2 as the input, the result for the Ricci tensor has no nonzero components. The

Ricci scalar, the trace of the Ricci tensor, therefore takes the form

$$R = 0\,. \tag{3.20}$$

It follows, that the Einstein tensor has no nonzero components.

This can be compared to the case of the Schwarzchild metric, where the Ricci scalar cannot be used to truly see the shape of the black hole.[18]

The Kretschmann scalar is expressed through the two parameters, mass m and the magnetic mass l, while also depending on the radial time $t' = \sqrt{t^2 + r^2}$

$$K = -48\frac{(-l^4 + l^3 m - 3l^3 t' - 3l^2 m t' + 3l^{t'2} - 3lmt'^2 + lt'^3 + mt'^3)}{(l^2 + t'^2)^6} \tag{3.21}$$

$$\times \frac{(l^4 + l^3 m - 3l^3 t' + 3l^2 m t' - 3l^2 t'^2 - 3lmt'^2 + lt'^3 - mt'^3)}{(l^2 + t'^2)^6}\,. \tag{3.22}$$

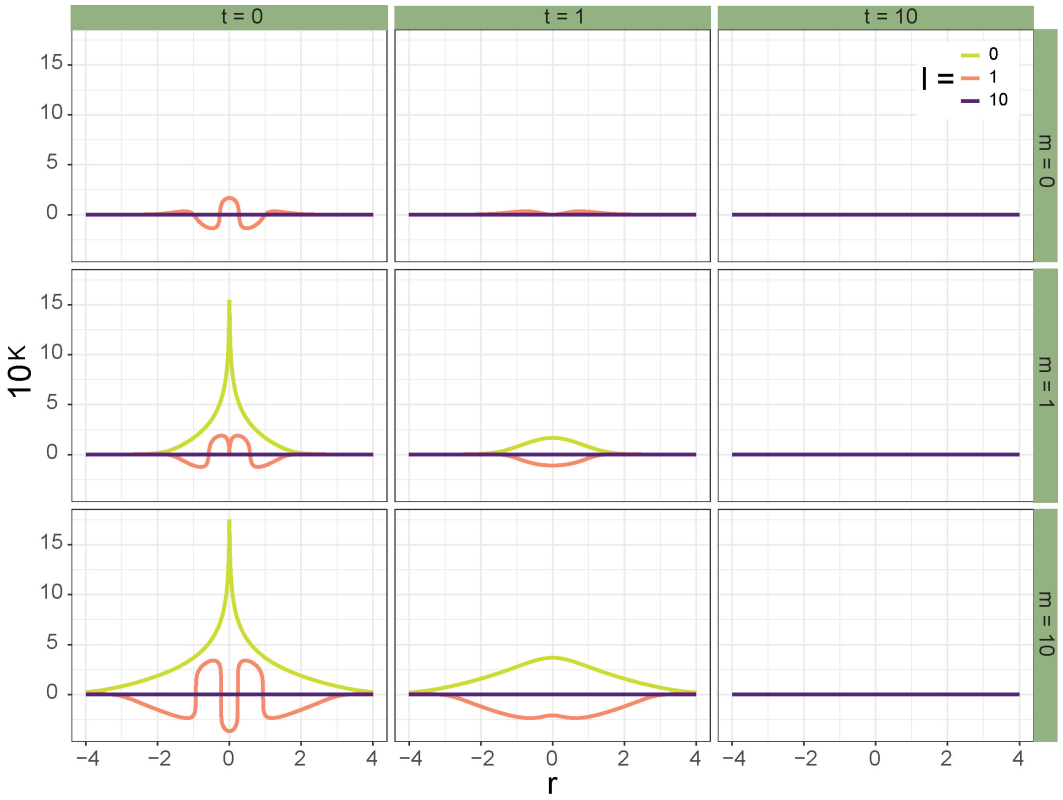

Figure 3.2: The dependence of the Kretschmann curvature scalar on the parameters m (mass), l (magnetic mass), t (time) and r (radius) on a logarithmic scale. The radius is extended to include negative values for the purpose of better demonstrating the peak at $r = 0$.

Chapter 4

Discussion

4.1 Gödel spacetime

The results for the Ricci tensor and scalar, Kretschmann scalar, as well as the Einstein tensor, match those found in earlier papers.[13]

As seen in figure 3.1, in cases where $m^2 \leq 4\omega^2$, which have one noncausal region, the Ricci scalar and Kretschmann scalar are both positive: a geodesic ball has a smaller volume than it would in Euclidean space at all points, however given the various, distinct elements in the Ricci and Riemann curvature tensors, the change in shape that gives rise to the lessened volume is not constant at all points in this Gödel space.

In the reversed case of $m^2 \geq 2\omega^2$, where there are no closed timelike curves, and therefore no noncausal regions, the Ricci scalar is negative while the Kretschamnn scalar retains its positive sign.

While in 2 dimensional space both negative and a positive Ricci scalar have strong topological implications, in 3 or more dimensional space, especially given a non-positive-definite metric, a positive Ricci scalar gives a better insight into the shape of the space than a negative Ricci scalar, as shown particularly by Hamilton's first convergence theorem for Ricci flow and Myers theorem for Riemannian manifolds, and the positive energy theorem for pseudo-Riemannian manifolds.

Myers theorem states, that if the Ricci scalar is, at every point on the Riemannian manifold, bounded below by $(n-1)k > 0$, then the diameter of the manifold does not exceed $\frac{\pi}{\sqrt{k}}$. This is equivalent to the manifold being compact.[22]

The positive energy theorem is analogous to the Myers theorem, stating that for a pseudo-Riemannian manifold with a Ricci scalar bounded below by $k > 0$, the ADM mass

m must be positive and nonzero. The ADM mass is the total energy of a spacetime as 'seen' by an observer at spatial infinity. The Minkowski space is the baseline case where the ADM mass is zero.

As seen in figure 3.1, the Ricci scalar is positive at all points in a Gödel space, given its parameters obey the inequality $m^2 < 4\omega^2$, due to $m^2 = 4\omega^2$ reducing the Ricci curvature scalar to zero. This indicates, that given parameters (m^2, ω^2), $m^2 < 4\omega^2$ the ADM mass is positive.[23]

As shown, taking $m^2 = 2\omega^2$ does reduce the scalars into the forms given by the Gödel metric defined in equation 1.16. Though the two have different coordinate systems, the curvature scalars do not depend on coordinates.

4.2 Taub-NUT spacetime

The Taub-NUT cannot be fully visualized using just the Ricci curvature tensor, as it is null like the Schwarzschild metric.

Due to the Kretschmann scalar depending on radial time $t' = \sqrt{t^2 + r^2}$, as time t increases while the radius r is constant, the black hole seems to *dissolve* into the space and the change in curvature near $r = 0$ lessens until the curvature is close to zero at all points on the manifold, as seen in figure 3.2.

Increases in the magnetic mass l flatten the curvature: even as the mass m increases, at $l = 10$ the curvature is flattened at all points.

The Kretschmann scalar, given parameters $m > 0$, $l = 0$, $t = 0$, strongly mirrors that of the Schwarzchild metric, which is to be expected, due to the Schwarzschild metric being a special case of the Taub-NUT spacetime.[24]

Chapter 5

Conclusion

There are few options for computing results in general relativity. Mathematica and Maplesoft are two good options that have packages allowing for the computation of tensors, but neither are free: The student license for Mathematica costs 159 euros[25] and for Maplesoft 90 euros[26]. Having an easy-to-use open source option for tensor computation is preferable to many. SageMath (another Python package, one which uses SymPy) and the SymPy package diffgeom both have uses when it comes to differential geometry in general relativity, however they are not fully equipped for every task a physicist might give them.

SymPy's inbuilt simplification function `simplify()` works well. However, it can take a fair amount of time to compute and may not return the simplest form for a human. The latter could be rectified by a combination of `simplify()` and `factor()` (which factorizes the expression), but this will add to the total computation time.

As outlined in the results chapter most of the computation time is spent simplifying the tensor or scalar, which can take 15 times longer than the actual computation. In the case of simpler metrics (such as the FLRW metric) the situation is reversed and the total computation time is rarely more than a minute, but as the complexity of the metric increases the total computation time exceeds 12 hours.

In general relativity, metrics have a symmetric form, meaning that the metric of an n dimensional space has $\frac{1}{2}n(n-1)$ unique elements instead of n^2. This lessening of unique elements can carry over to various tensors, and when taken into account, can greatly reduce the time needed to compute all elements of a given tensor. The code written within the bounds of this thesis does not take this into account and computes each element of a tensor separately which, for more complex metrics such as the Taub-NUT metric can, lengthen the computation time up to 20 minutes for the Kretschmann scalar.

In the future a more object oriented approach could be taken, starting with the defining of the `Tensor` class. The raising and lowering, as well as the contraction, of indices along with element-wise simplification would become methods of the `Tensor` class. With this approach a user would not have to wonder if the tensor is a (0,4) or (1,3) type, as, for example, `tensor.type()` would return the number of contravariant and covariant indices.

A system similar to that of Mathematica would be beneficial when it comes to returning the results: instead of printing each one, Mathematica only prints the unique elements, along with a list of dependencies.

The use of computers in symbolic calculations is still in development, especially in the public domain. Current open source packages that aid in the mathematical aspect of science are still lacking in some fields, but as demonstrated in this thesis it is possible to help create these necessary tools for those with no access to (or want to use) programs that require a license.

Acknowledgements

I'd like to thank Manuel Hohmann for the help choosing metrics to analyse and finding mistakes in my code, Hankel for answering questions related to math late at night, Anneli Poska for help making the figures look prim and proper, and Stewart Jackson for proofreading.

Bibliography

[1] A. Einstein and H. A. Lorentz. *The Principle of Relativity*. Methuen and Company, LTD, 1923.

[2] Klein Martin J. Schulmann Robert Kox, A. J. *Volume 6: The Berlin Years: Writings, 1914-1917 (English translation supplement)*. Princeton University Press, 1997.

[3] Neil Ashby. Relativity in the global positioning system. *Living Reviews in Relativity*, 55, 01 2003.

[4] TIOBE The Software Quality Company. Tiobe index. http://www.tiobe.com/tiobe-index/. Accessed: 2019-12-01.

[5] K. Millman and Michael Aivazis. Python for scientists and engineers. *Computing in Science & Engineering*, 13:9 – 12, 05 2011.

[6] CERN Bulletin. Python: The holy grail of programming. http://cdsweb.cern.ch/journal/CERNBulletin/2006/31/News%20Articles/974627?ln=en, 2006. Accessed: 2019-12-01.

[7] Aaron Meurer, Christopher P. Smith, and Paprocki. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[8] Jacques Dixmier. *General Topology*. Springer-Verlag, 1984.

[9] Bernard F. Schutz. *Geometrial methods of mathematical physics*. Cambridge University Press, 1982.

[10] George F.R. Ellis and Henk van Elst. Cosmological models: Cargese lectures 1998. *NATO Sci. Ser. C*, 541:1–116, 1999.

[11] Kurt Gödel. An example of a new type of cosmological solutions of einstein's field equations of gravitation. *Rev. Mod. Phys.*, 21:447–450, Jul 1949.

[12] M. J. Rebouças and J. E. Åman. Computer-aided study of a class of riemannian space-times. *Journal of Mathematical Physics*, 28(4):888–892, 1987.

[13] U. Camci and M. Sharif. Matter collineations of space-time homogeneous Godel type metrics. *Class. Quant. Grav.*, 20:2169, 2003.

[14] A.F.F. Teixeira W.M. Silva Jr M.O. Calvaõ, M.J. Rebouças. Notes on a class of homogeneous space-times. *Journal of Mathematical Physics*, 29:1127–1129, 1988.

[15] Haryanto M. Siahaan. Kerr-Sen-Taub-NUT spacetime and circular geodesics. 5 2019.

[16] Sean M. Carroll. Lecture notes on general relativity. 12 1997.

[17] Shoshichi Kobayashi and Katsumi Nomizu. *Foundations of Differential Geometry Volume 1*, volume 1. Interscience Publishers, 1963.

[18] Richard C. Henry. Kretschmann scalar for a kerr-newman black hole. *Astrophys. J.*, 535:350, 2000.

[19] Matthias Blau. Lecture notes on general relativity. http://www.blau.itp.unibe.ch/newlecturesGR.pdf, 2020. Accessed: 2020-05-10.

[20] Mikio Nakahara. *Geometry, Topology and Physics*. Taylor & Francis, 3rd edition, 2003.

[21] Hans Stephani. *Exact Solutions to Einstein's Field Equations*. Cambridge University Press, 2nd edition, 2003.

[22] S. B. Myers. Riemannian manifolds with positive mean curvature. *Duke Math. J.*, 8(2):401–404, 06 1941.

[23] Edward Witten. A new proof of the positive energy theorem. *Comm. Math. Phys.*, 80(3):381–402, 1981.

[24] C.P. Burgess, Robert C. Myers, and F. Quevedo. Duality and four-dimensional black holes. *Nucl. Phys. B*, 442:97–125, 1995.

[25] Mathematica for students. https://www.wolfram.com/mathematica/pricing/students/, 2020. Accessed: 2020-05-10.

[26] Maple for student: Pricing & purchase. https://www.maplesoft.com/products/maple/pricing/, 2020. Accessed: 2020-05-10.

Appendix A

Gödel type metric results

The Ricci components are

$$R_{00} = \frac{H'^2}{2D^2} \tag{A.1}$$

$$R_{11} = -\frac{2DD'' - H'^2}{2D^2} \tag{A.2}$$

$$R_{02} = R_{20} = \frac{D^2D'' - DD'H' + HH'^2}{2D^2} \tag{A.3}$$

$$R_{22} = -\frac{2D^3D'' - 2D^2HH'' - D^2H'^2 + 2DHD'H' - H^2H'^2}{2D^2}\,. \tag{A.4}$$

The Einstein components are

$$G_{00} = -\frac{4DD'' - 3H'^2}{4D^2} \tag{A.5}$$

$$G_{11} = \frac{H'^2}{4D^2} \tag{A.6}$$

$$G_{02} = G_{20} = \frac{2D^2H'' - 4DHD'' - 2DD'H' + 3HH'^2}{4D^2} \tag{A.7}$$

$$G_{22} = \frac{4D^2HH'' + D^2H'^2 - 4DH^2D'' - 4DHD'H' + 3H^2H'^2}{4D^2} \tag{A.8}$$

$$G_{33} = \frac{4DD'' - H'^2}{4D^2}\,. \tag{A.9}$$

The Ricci scalar is

$$R = \frac{4DD'' - H'^2}{2D^2}\,. \tag{A.10}$$

The Kretschmann scalar is

$$K = \frac{16D^2D''^2 - 8D^2H''^2 + 16DD'H'H'' - 24DD''H'^2 - 8D'^2H'^2 + 11H'^4}{4D^4} \,. \tag{A.11}$$

Appendix B

New functions

The import list for packages:

```
from sympy import *
from sympy.diffgeom import *
from sympy.diffgeom.rn import *
import numpy as np
from itertools import permutations
TP = TensorProduct
tp = tensorproduct
tc = tensorcontraction
```

B.1 The manifold GR4

```
GR4 = Manifold('R^4', 4)
# Patch and coordinate systems.
GR4_origin = Patch('origin', GR4)
GR4_r = CoordSystem('rectangular',
            GR4_origin, ['t', 'x', 'y', 'z'])
GR4_c = CoordSystem('cylindrical',
            GR4_origin, ['t', 'rho', 'psi', 'z'])
GR4_s = CoordSystem('spherical',
            GR4_origin, ['t', 'r', 'theta', 'phi'])
GR4_h = CoordSystem('hyperspherical',
            GR4_origin, ['t', 'psi', 'theta', 'phi'])

# Connecting the coordinate charts.
t, x, y, z, rho, psi, r, theta, phi = [Dummy(s) for s in
```

```
                    ['t', 'x', 'y', 'z', 'rho',
                    'psi', 'r', 'theta', 'phi']]
## rectangular <-> cylindrical
GR4_r.connect_to(GR4_c, [t, x, y, z],
                      [t, sqrt(x**2 + y**2), atan2(y, x), z],
                inverse=False, fill_in_gaps=False)
GR4_c.connect_to(GR4_r, [t, rho, psi, z],
                      [t, rho*cos(psi), rho*sin(psi), z],
                inverse=False, fill_in_gaps=False)
## rectangular <-> spherical
GR4_r.connect_to(GR4_s, [t, x, y, z],
                      [t, sqrt(x**2 + y**2 + z**2), acos(z/
                            sqrt(x**2 + y**2 + z**2)), atan2(y, x)],
                inverse=False, fill_in_gaps=False)
GR4_s.connect_to(GR4_r, [t, r, theta, phi],
                      [t, r*sin(theta)*cos(phi), r*sin(
                          theta)*sin(phi), r*cos(theta)],
                inverse=False, fill_in_gaps=False)
## cylindrical <-> spherical
GR4_c.connect_to(GR4_s, [t, rho, psi, z],
                      [t, sqrt(rho**2 + z**2),
                      acos(z/sqrt(rho**2 + z**2)), psi],
                inverse=False, fill_in_gaps=False)
GR4_s.connect_to(GR4_c, [t, r, theta, phi],
                      [t, r*sin(theta), phi, r*cos(theta)],
                inverse=False, fill_in_gaps=False)

#hypserspherical <-> spherical
GR4_h.connect_to(GR4_s, [t, psi, theta, phi],
                      [t*cos(psi), t*sin(psi),
                      acos(sin(theta)*sin(phi)),
                      atan(tan(theta)*cos(phi))],
                inverse=False, fill_in_gaps=False)
GR4_s.connect_to(GR4_h, [t, r, theta, phi],
                      [sqrt(t**2+r**2), acos(t/sqrt(t**2+r**2)),
                      acos(sin(theta)*cos(phi)),
                      acos(sin(theta)*sin(phi)/sqrt(cos(theta)**2
                      +sin(theta)**2*sin(phi)**2))],
                inverse=False, fill_in_gaps=False)
## rectangular <-> hyperspherical
GR4_r.connect_to(GR4_c, [t, x, y, z],
                      [sqrt(t**2+x**2+y**2+z**2),
                      acos(t/sqrt(z**2+y**2+x**2+t**2)),
                      acos(x/sqrt(z**2+y**2+x**2)),
                      acos(y/sqrt(z**2+y**2))],
```

```
                inverse=False, fill_in_gaps=False)
GR4_c.connect_to(GR4_r, [t, psi, theta, phi],
                      [t*cos(psi), t*sin(psi)*cos(theta),
                      t*sin(psi)*sin(theta)*cos(phi),
                      t*sin(psi)*sin(theta)*sin(phi)],
                inverse=False, fill_in_gaps=False)
## cylindrical <-> hyperspherical
GR4_c.connect_to(GR4_s, [t, rho, psi, z],
                      [sqrt(t**2+rho**2+z**2),
                      acos(t/sqrt(t**2+rho**2+z**2)),
                      acos(rho*cos(phi)/sqrt(z**2+rho**2)),
                      acos(rho*sin(phi)/sqrt(z**2+rho**2*sin(phi)**2))],
                inverse=False, fill_in_gaps=False)
GR4_s.connect_to(GR4_c, [t, psi, theta, phi],
                      [t*cos(psi),
                      t*sin(psi)*sqrt(cos(theta)**2
                      +sin(theta)**2*cos(phi)**2),
                      atan(tan(theta)*cos(phi)),
                      t*sin(psi)*sin(theta)*sin(phi)],
                inverse=False, fill_in_gaps=False)

del t, x, y, z, rho, psi, r, theta, phi

# Defining the basis coordinate functions.
GR4_r.t, GR4_r.x, GR4_r.y, GR4_r.z = GR4_r.coord_functions()
GR4_c.t, GR4_c.rho, GR4_c.psi, GR4_c.z = GR4_c.coord_functions()
GR4_s.t, GR4_s.r, GR4_s.theta, GR4_s.phi = GR4_s.coord_functions()
GR4_h.t, GR4_h.psi, GR4_h.theta, GR4_h.phi = GR4_h.coord_functions()

# Defining the basis vector fields.
GR4_r.e_t, GR4_r.e_x, GR4_r.e_y, GR4_r.e_z = GR4_r.base_vectors()
GR4_c.e_t, GR4_c.e_rho, GR4_c.e_psi, GR4_c.e_z = GR4_c.base_vectors()
GR4_s.e_t, GR4_s.e_r, GR4_s.e_theta, GR4_s.e_phi = GR4_s.base_vectors()
GR4_h.e_t, GR4_h.e_psi, GR4_h.e_theta, GR4_h.e_phi = GR4_h.base_vectors()

# Defining the basis oneform fields.
GR4_r.dt, GR4_r.dx, GR4_r.dy, GR4_r.dz = GR4_r.base_oneforms()
GR4_c.dt, GR4_c.drho, GR4_c.dpsi, GR4_c.dz = GR4_c.base_oneforms()
GR4_s.dt, GR4_s.dr, GR4_s.dtheta, GR4_s.dphi = GR4_s.base_oneforms()
GR4_h.dt, GR4_h.dpsi, GR4_h.dtheta, GR4_h.dphi = GR4_h.base_oneforms()
```

B.2 Base functions

```
def lower_indices(T, expr, changes):
    metric = twoform_to_matrix(expr)

    N = T.rank()
    lis = list(range(len(changes), N))
    em = []
    k = len(changes)-1
    l = 0

    for j in range(N):
        if j in changes:
            em.append(k)
            k -= 1
        else:
            em.append(lis[l])
            l += 1

    new_order = tuple(em)

    lowered = T

    for i in changes:
        lowered = tc(tp(metric, lowered), (0, i+2))

    lowered = permutedims(lowered, new_order)

    return lowered

def raise_indices(T, expr, changes):
    coord_sys = expr.atoms(CoordSystem).pop()
    indices = list(range(coord_sys.dim))
    #inverting a matrix does not work if it contains nonsymbols
    #metric = twoform_to_matrix(expr).inv()
    matrix = twoform_to_matrix(expr)
    s_fields = set()
    for e in matrix:
         s_fields.update(e.atoms(BaseScalarField))
    s_fields = list(s_fields)
    dums = coord_sys._dummies
    invmetric = matrix.subs(list(zip(s_fields,
            dums))).inv().subs(list(zip(dums, s_fields)))
    #end of workaround

    N = T.rank()
    lis = list(range(len(changes), N))
```

```
    em = []
    k = len(changes)-1
    l = 0

    for j in range(N):
        if j in changes:
            em.append(k)
            k -= 1
        else:
            em.append(lis[l])
            l += 1

    new_order = tuple(em)

    raised = T

    for i in changes:
        raised = tc(tp(invmetric, raised), (0, i+2))

    raised = permutedims(raised, new_order)

    return raised


def simplify_array(A, d):
    R = A.rank()
    F = flatten(A)
    for i in range(len(F)):
        F[i] = simplify(F[i])
    t = tuple([d]*R)
    R = (Array(F, (d,)*R))

    return R


def Kulkarni_Nomizu_product(A,B):
    AB = tp(A,B)
    out = permutedims(AB, (0,2,1,3)) + permutedims(AB, (1,3,0,2)) -\
        permutedims(AB, (0,3,1,2)) - permutedims(AB, (1,2,0,3))
    return out


def print_components(T, d):
    perms = list(set(permutations(np.repeat(range(d),T.rank()),T.rank())))

```

```
    for perm in perms:
        print(perm, factor(T[perm]))
```

B.3 Derivatives

```
def partial_derivative(T, expr):
    coord_sys = expr.atoms(CoordSystem).pop()
    indices = list(range(coord_sys.dim))

    diffd = [T.applyfunc(lambda a: d(a))
                for d in coord_sys.base_vectors()]
    diffd = (Array(diffd, (coord_sys.dim,)*(T.rank()+1)))

    return ImmutableDenseNDimArray(diffd)


def covariant_derivative(T, expr, n_up):
    Ch = metric_to_Christoffel_2nd(expr)
    PD = partial_derivative(T, expr)

    for i in range(n_up):
        PD += tc(tp(Ch, T), (2, (3+i)))


    for i in range(T.rank()-n_up):
        PD -= tc(tp(Ch, T), (0, (i+3+n_up)))

    return ImmutableDenseNDimArray((PD))


def Lie_derivative_covariant(X, T, expr, n_up):
    LD = tc(tp(X, covariant_derivative(T, expr, n_up)), (0,1))

    for j in range(n_up):
        LD -= tc(tp(covariant_derivative(X, expr, 1), T),(0,2+j))

    for j in range(T.rank()-n_up):
        LD += tc(tp(covariant_derivative(X, expr, 1), T), (1,3+n_up+j))

    return LD


def Lie_derivative_partial(X, T, expr, n_up):
```

```
    LD = tc(tp(X, partial_derivative(T, expr)), (0,1))
    print(LD)

    for j in range(n_up):
        LD -= tc(tp(partial_derivative(X, expr), T),(0,2+j))

    for j in range(T.rank()-n_up):
        LD += tc(tp(partial_derivative(X, expr), T), (1,3+n_up+j))

    return LD
```

B.4 Tensors

```
def metric_to_Ricci_scalar(expr):
    ricci = metric_to_Ricci_components2(expr)

    scalar = tc(raise_indices(ricci, expr, (0,)), (0,1))

    return scalar


def metric_to_torsion_components(expr):
    Ch2 = metric_to_Christoffel_2nd(expr)

    torsion = Ch2 - permutedims(Ch2, (0,2,1))

    return ImmutableDenseNDimArray(torsion)


def metric_to_contorsion_components(expr):
    tor = lower_indices(metric_to_torsion_components(expr), expr, (0,))

    cont2 = tor-permutedims(tor,(1,2,0))+permutedims(tor,(2,0,1))
    cont = tp(Rational(1,2),cont2)

    return ImmutableDenseNDimArray(cont)


def metric_to_Einstein_components(expr):
    ricci = metric_to_Ricci_components(expr)
    metric = twoform_to_matrix(expr)
    scalar = metric_to_Ricci_scalar(expr)

```

```
    Rg = tp(Rational(1, 2), metric, scalar)
    einstein = ricci - Rg

    return ImmutableDenseNDimArray(einstein)


def metric_to_Kretschmann_scalar(expr):
    riemann = metric_to_Riemann_components(expr)

    rie_low = lower_indices(riemann, expr, (0,))
    rie_up = raise_indices(riemann, expr, (1,2,3))

    kretschmann = tc(tc(tc(tc(tp(rie_low, rie_up),
            (0,4)), (0,3)), (0,2)), (0,1))

    return kretschmann
```

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Anna Aader,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

   The Computation of Tensors in General Relativity Using the Python Package SymPy,

   mille juhendaja on Manuel Hohmann, PhD, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. olen teadlik, et punktis 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Anna Aader,
Tartu, 29. mai 2020. a.