PyOGRe-Private

March 11, 2022

1 PyOGRe: A Python Object-Oriented General Relativity Package

Documentation for v1.0.0 (January 2022)

By **Barak Shoshany** (website) and **Jared Wogan** (website)

GitHub repository

PyPi Project

2 Introduction

2.1 Summary

PyOGRe is the Python port of the Mathematica package OGRe, written by Professor Barak Shoshany at Brock University. OGRe is a tensor calculus package that is designed to be both powerful and user-friendly. OGRe can perform calculations involving tensors extremely quickly, which could often take hours to do by hand. It is extremely easy to pick up and use, with easy to learn syntax. Naturally, the package has applications in general relativity where tensors are used abundantly and was the focus point during the development of the package. There is no restriction preventing the package from being used for other applications, like electromagnetism, quantum mechanics, or other fields.

Tensors are abstract mathematical objects that can be represented by multi-dimensional arrays. A tensor may be represented by anything from a scalar, a vector, a matrix, to a N-dimensional array. Each array is a specific representation of a tensor, but this representation is not the only valid representation. By performing a change of coordinates, or changing the index configuration, we can produce another completely valid representation of the same tensor. It is thus crucial that we specify both a set of coordinates and an index configuration when calling a specific array a representation of a tensor. Take, for example, the (rank 2) tensor $T_{\mu\nu}$ below:

$$T_{\mu\nu}(t,x,y,z) = \begin{pmatrix} T_{tt} & T_{tx} & T_{ty} & T_{tz} \\ T_{xt} & T_{xx} & T_{xy} & T_{xz} \\ T_{yt} & T_{yx} & T_{yy} & T_{yz} \\ T_{zt} & T_{zx} & T_{zy} & T_{zz} \end{pmatrix} = \begin{pmatrix} T_{00} & T_{01} & T_{02} & T_{03} \\ T_{10} & T_{11} & T_{12} & T_{13} \\ T_{20} & T_{21} & T_{22} & T_{23} \\ T_{30} & T_{31} & T_{32} & T_{33} \end{pmatrix}$$

Above is a single representation of the tensor T, with two lower indices, written in standard Cartesian coordinates. We call T a rank two tensor because it requires two indices to specify a component of the tensor; this means a scalar can represent a rank zero tensor, a vector can represent a rank

one tensor, and so on (note how we do not say a scalar is a tensor, but instead that it can represent a tensor). If we wrote the same tensor in a different coordinate system, or with a different index configuration, the components would be completely different. Since we are working with representations of abstract objects, it is clear that we must be careful and ensure that all representations of a given tensor truly do represent the same tensor.

When working with tensors, it is natural to want to combine them in different ways. PyOGRe allows the user to perform calculations involving tensors in a straightforward manner. The user will only need to input a tensor once with a specified choice of coordinates and an index configuration, then behind the scenes, PyOGRe will automatically ensure the correct representation of the tensor is used. This makes the often complicated and/or confusing expressions involving tensors straightforward, taking away the unnecessary complexities of the calculation. Some of the possible operations that PyOGRe can perform are:

- Addition
- Subtraction
- Multiplication by a scalar
- Trace
- Contraction
- Partial and Covariant Derivatives

The documentation provided below will ensure that a proper theoretical understanding of the mathematics behind the operations is achieved, then, we will explain how each calculation can be performed using PyOGRe.

Behind the scenes, PyOGRe takes an object-oriented design and stores each tensor as a single object (we will explain object-oriented design in a later section). Each object stores all the representations of the tensor, where each representation is stored as an array. We ensure that all representations stored within the tensor object truly represents the same tensor by preventing the user from modifying the data once it has been created. Preventing the user from changing the data of a tensor is a fundamenetal part of the object-oriented design, as it allows a series of assumptions to be made about the tensor, greatly improving performance and preventing errors.

Instead of allowing the user to modify the data of a tensor, PyOGRe instead allows the user to request specific reprentations of any tensor. Handling the data of a tensor in this manner allows PyOGRe to be much more efficient; if a specific representation already exists, it will not be recalculated, while still giving the user the ability to request a tensor in different representations. Secondly, if a represntation of a tensor is calculated in some intermediate step of a calculation, all intermediate representations are stored as well, which can provide another increase in efficiency.

The remainder of this document will outline why we have decided to port OGRe to Python, a brief list of the features, the meaning of object-oriented design, the differences between the original OGRe package and PyOGRe, and finally documentation of each module of the package.

2.2 Motivation for the Python Port

Porting a piece of software is simply the act of converting a piece of software from one programming language to another. A port is meant to achieve the same level of functionality. PyOGRe is simply a port of the OGRe package written in Mathematica to Python. Both versions of the package will retain near identical syntax, as well as cross compatibility, so that work in one package may be freely moved to the other.

The decision to port the package from Mathematica to Python consists of several motivating factors. One of the primary reasons is because Mathematica requires a license which must be purchased. This makes accessing the package difficult for people who may not have the required funding for a Mathematica license, which is where Python can help. Python is **freely** available for anyone to install, so by porting the package to Python, we are removing the paywall that the Mathematica version currently imposes.

Additionally, while Mathematica is extremely powerful and popular within the mathematics or physics communities, Python is far more popular in general. Python has been seeing a surge in popularity over the past years and is now becoming one of the most common languages beginners start with, especially within physics. Porting the package to Python will thus vastly improve the number of potential users, as well as the **accessibility** over the existing package in Mathematica.

Finally, in order to port the package to another language, we needed to consider which languages had support for **symbolic computation**. Again, Python is a clear option over a language such as C/C++, as it is both popular for scientific computation already, and has support for symbolic computation through a package called SymPy. We have used SymPy extensively when developing PyOGRe and have found it to be both easy to use and powerful. SymPy is very easy to pick up and learn, and there is very little the end user will have to know about SymPy in order to use PyOGRe. Of course, since Python is also an object-oriented language, we were able to stay true to the object-oriented design philosophy, which was not entirely possible with Mathematica.

PyOGRe is not meant to be a replacement for OGRe in any way, but merely a complimentary package to the existing package, giving the user flexibility to perform computation in either Python or Mathematica. All of the same features found in the original package are available in PyOGRe or will be made available in the future. We have also made it possible for the user to export and import tensors created in one version of the package to the other version.

2.3 Features

PyOGRe is currently under active development and we are constantly adding new functionality. The following is a brief outline of the main features that can currently be found in PyOGRe:

- Define coordinate systems and any transformation rules between them. Tensor components in any representation can then be transformed automatically as needed.
- Define metrics which can then be used to define any arbitrary tensor. The metric associated with a tensor will be used to raise and lower indices as needed.
- Display any tensor object in any coordinate system as an array, or as a list of the unique non-zero components. Metrics can additionally be displayed as a line element.
  - When displaying a tensor, substitutions can be made for any variable or function present. Additionally, a function may be specified that will be mapped to each component of the tensor.
- Export tensors to a file so that they can later be imported into a new session or into the Mathematica version.
- A simple API for performing calculations on tensors, including addition, subtraction, multiplication by a scalar, trace, contraction, as well as partial and covariant derivatives.
- Built in tensors for commonly used coordinate systems and metrics.
- Built in functions for calculating the Christoffel symbols (the Levi-Cevita connection), Riemann tensor, Ricci tensor, Ricci scalar, Einstein Tensor, curve Lagrangian, and volume element from a metric, as well as the norm squared of any tensor.

- Calculate the geodesic equations in terms of a user defined curve parameter (affine parameter), using either the Christoffel symbols or the curve Lagrangian (for spacetime metrics, the geodesic equations can be calculated in terms of the time coordinate).
- Designed to be performant using optimized algortihms for common operations (these functions can be used on any SymPy array).
- Quick and easy to install straight from PyPI (supports Python 3.6 and above, previous versions untested).
- Command line and jupyter notebook support.
- Clear and well documented source code, complete with examples.
- Open source and available for all to use.
- Easily extendible and modifiable.
- Under active development and will be updated regularly.

2.4 Object-Oriented Design Philosophy

Object-oriented programming (OOP) simply refers to a programming paradigm in which objects are used to combine data and functionality. Each object contains some amount of data (often called **attributes**), as well as functions that can operate on that data (often called **methods**). Each **object** belongs to a **class** that the user defines and can be thought of as a blueprint. The class tells the end user what kind of data each object will have, and how that data should be stored. The class also tells the end user what kind of functions each object will have, and how those functions should be implemented.

An extremely important aspect of object-oriented design is the idea of **encapsulation**. Encapsulation is the process of hiding the implementation details of an object, and only exposing the interface to the user. What this means is that the user may only access the data of an object through methods defined by the class but is unable to change or modify the actual data of the object directly. This enables a new idea, **class invariants**, which are assumptions about the object that should always be preserved. The invariance of a class allows the assumption that all data stored in each object remains valid, leading to more efficient and simpler code.

As we discussed previously, tensors are abstract mathematical objects, which makes object-oriented programming a natural choice. Each tensor is stored as a single, self-contained object. The tensor object stores all the required data about each tensor, including the components for each known or calculated representation. Since all tensors in PyOGRe share a common class, we only need to define operations on tensor objects once in an abstract manner, and PyOGRe will automatically be able to apply these operations to any tensor object.

The most important class invariant of a tensor object in PyOGRe are naturally the components of the tensor in each representation. Since each tensor stores these components in the object representing the tensor, it is crucial that each representation does indeed represent the tensor. This is done through encapsulation; the user is **not** allowed to access these components of the tensor once it has been defined. The components may only be modified by **private methods** (methods the user does not have access to) that preserve the invariance of the class. This prevents the user from accidently violating the class invariance, and thus, the invariance of the tensor object. This again, allows us to work under the assumption that all the data stored inside a tensor object is valid, and that the data does in fact represent the tensor.

In PyOGRe, the user will initially define (or **construct**) a tensor in some specific representation once but will then never have to worry about coordinates or indices anymore. In fact, the user will

not even need to remember which coordinates or indices were used to construct the tensor in the first place. The user will simply be able to request the tensor in any representation they desire, or when performing calculations, PyOGRe will determine which representation is required in each context.

An important distinguishing feature between the Mathematica version (OGRe) and the Python version (PyOGRe) of the package is that in Python, we can truly define tensors as objects. In Mathematica, there is no notion of a class; all tensors are simply represented as an association. At heart, the Mathematica version operates as though it is object-oriented, and ensures that the "objects" are invariant, however the objects themselves are merely a list. This is because Mathematica is not an object-oriented programming language; in Mathematica, it is not possible to define class methods, as there are no classes to begin with. To ensure PyOGRe feels familiar to the user, we have also included functions alongside the class methods that maintain the same functionality.

2.5 Syntax Differences

To illustrate some of the differences between the two versions, and how Python allows us to adhere to the object-oriented paradigm more closely, consider the following. Suppose a user wanted to define the Minkowski metric. In OGRe, assuming the Cartesian coordinates were already defined, the user could define a new metric as follows:

```
TNewMetric["Minkowski", "Cartesian", DiagonalMatrix[{-1, 1, 1, 1}]]
```

while in Python, the user could instead do:

```
cartesian.new_metric("Minkowski", sympy.diagonal(-1, 1, 1, 1))
```

The difference is that in Python, defining a metric can be done by calling a method on a coordinate system. PyOGRe will then know automatically that the new metric is defined in terms of the cartesian coordinates and will not have to be told explicitly. Of course, one can also create a new tensor by calling a method on the metric. If instead the user wanted to calculate the norm squared of a tensor, the user could use the method `calc_norm_squared` on the tensor in PyOGRe, where in OGRe one would instead have to write `TCalcNormSquared`, with the tensor as an argument. It should be noted though that the functional equivalents of these methods are also available in PyOGRe.

Another minute difference between the two versions, is that in OGRe, function names use camel case (e.g., TCalcNormSquared). In PyOGRe, we use snake case to adhere to common Python conventions (e.g., calc_norm_squared). This does not in any way change the functionality of the functions or methods, it is merely a slightly different naming scheme.

3 Installing and Loading the Package

3.1 Installing

Installing PyOGRe is done using PIP. Simply open a terminal, and type:

```
pip install PyOGRe
```

PIP will automatically find the latest version and install it to the default interpreter, including all of the dependencies. If you would like to update a current installation of PyOGRe, you may instead run:

```
pip install --upgrade PyOGRe
```

If you are looking for a specific version of PyOGRe, you may specifiy the version using the following command:

```
pip install PyOGRe==x.y.z
```

where **x.y.z** is the targeted version.

3.2 Importing PyOGRe

Importing PyOGRe is done in the same manner as any other Python package. We recommend also importing `SymPy`, which is used extensively by the package, and the `display` function from `IPython.display` if you are working within a Jupyter Notebook.

NOTE: Although tempting, never write something such as:

```
from PyOGRe import *
```

This is known as a wildcard import and can clutter the namespace, therefore we recommend importing PyOGRe as we have done below.

[ ]:
```
%load_ext autoreload
%autoreload 2
```

[ ]:
```
import sympy as sym
from IPython.display import display

import PyOGRe as og
```

If you are working in a terminal, call the `command_line_support()` function to tell PyOGRe that it should display results using ASCII characters. The default behaviour assumes you are working in a Jupyter Notebook (this can be set manually using `jupyter_support()`), and all outputs are rendered using Markdown and LaTeX.

[ ]:
```
og.command_line_support()
og.jupyter_support()
```

If you would like to see the current options the package is using, call the `get_options()` function.

[ ]:
```
og.get_options()
```

```
LATEX: True
ALL_SYMBOLS: (mu, nu, rho, sigma, kappa, lambda, alpha, beta, gamma, delta,
epsilon, zeta, theta, iota, xi, pi, tau, phi, chi, psi, omega)
CURVE_PARAMETER: lambda
INFO_ORDER: ('Name', 'Symbol', 'Type', 'Rank', 'Metric', 'Default Coordinates',
'Default Indices', 'Default Coordinates For', 'Coordinate Transformations',
'Indices Symbols', 'Tensors Using This Metric')
LIST_PER_LINE: 6
PARALLEL: False
```

One may also use the `doc()` function to find the documentation for the package, which is this notebook (also available in PDF format).

[ ]:
```
og.doc()
```

```
PyOGRe Documentation

Version: 1.0.0

PyOGRe is an Object-Oriented General Relativity Package for Python.
The full documentation is available at:
```

4 Creating and Displaying Tensor Objects

4.1 A Quick Overview of SymPy

[ ]:
```
t, x, y, z = sym.symbols('t x y z')
r, theta, phi = sym.symbols('r theta phi')

cartesian_symbols = sym.Array([t, x, y, z])
spherical_symbols = sym.Array([t, r, theta, phi])

display(cartesian_symbols)
display(spherical_symbols)
```

$$\begin{bmatrix} t & x & y & z \end{bmatrix}$$

$$\begin{bmatrix} t & r & \theta & \phi \end{bmatrix}$$

4.2 Defining Coordinate Systems

[ ]:
```
cartesian = og.new_coordinates(
    name="Cartesian",
    components=cartesian_symbols
)

spherical = og.new_coordinates(
    name="Spherical",
    components=spherical_symbols
)
```

4.3 Displaying PyOGRe Objects

4.3.1 Displaying as an Array

[ ]:
```
cartesian.show()
```

Cartesian:

$$x^{\mu} = \begin{pmatrix} t \\ x \\ y \\ z \end{pmatrix}$$

4.3.2 Displaying as a List

[ ]:
```
spherical.list()
```

Spherical:

$$x^{t} = t$$

$$x^{r} = r$$

$$x^{\theta} = \theta$$

$$x^{\phi} = \phi$$

4.3.3 Retrieving Information about a Tensor

[ ]:
```
cartesian.info()
```

Cartesian:

Name: Cartesian

Symbol: x

Type: Coordinates

Rank: 1

Default Indices: (1,)

Default Coordinates For:

4.3.4 Getting the Rank of a Tensor

[ ]:
```
print(f"{cartesian.rank() = }")
print(f"{spherical.rank() = }")
```

```
cartesian.rank() = 1
spherical.rank() = 1
```

4.4 Defining Coordinate Transformations

[ ]:
```
# t -> t
# x -> r*sin(theta)*cos(phi)
# y -> r*sin(theta)*sin(phi)
# z -> r*cos(theta)
cartesian_to_spherical = [
    sym.Eq(x, r*sym.sin(theta)*sym.cos(phi)),
    sym.Eq(y, r*sym.sin(theta)*sym.sin(phi)),
    sym.Eq(z, r*sym.cos(theta))
]

cartesian.add_coord_transformation(
    coords=spherical,
    rules=cartesian_to_spherical
)
```

[ ]:
```
Cartesian
```

[ ]:
```
# t -> t
# r -> sqrt(x^2+y^2+z^2)
# theta -> acos(z/(sqrt(x^2+y^2+z^2)))
# phi -> atan(y/x)
spherical_to_cartesian = [
    sym.Eq(r, sym.sqrt(x**2+y**2+z**2)),
    sym.Eq(theta, sym.acos(z/(sym.sqrt(x**2+y**2+z**2)))),
    sym.Eq(phi, sym.atan(y/x)),
]

spherical.add_coord_transformation(
    coords=cartesian,
    rules=spherical_to_cartesian
)
```

[ ]:
```
Spherical
```

4.5 Defining Metrics

[ ]:
```
minkowski = og.new_metric(
    name="Minkowski",
    coords=cartesian,
    components=sym.diag(-1, 1, 1, 1)
)
minkowski.show()
```

Minkowski:

$$g_{\mu\nu}\left(t,x,y,z\right)=\begin{pmatrix}-1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$$

[ ]:
```
M = sym.symbols("M")

schwarschild = spherical.new_metric(
    name="Schwarzschild",
    components=sym.Array(
        [
            [-(1-2*M/r), 0, 0, 0],
            [0, 1/(1-2*M/r), 0, 0],
            [0, 0, r**2, 0],
            [0, 0, 0, r**2 * sym.sin(theta)**2]
        ]
    )
)
schwarschild.show()
```

Schwarzschild:

$$g_{\mu\nu}\left(t,r,\theta,\phi\right)=\begin{pmatrix}\frac{2M}{r}-1&0&0&0\\0&\frac{1}{-\frac{2M}{r}+1}&0&0\\0&0&r^2&0\\0&0&0&r^2\sin^2\left(\theta\right)\end{pmatrix}$$

4.5.1 Displaying Line Elements

[ ]:
```
minkowski.line_element()
minkowski.line_element(coords=spherical)
```

Minkowski:

$$\mathrm{d}s^2=-\mathrm{d}t^2+\mathrm{d}x^2+\mathrm{d}y^2+\mathrm{d}z^2$$

Minkowski:

$$\mathrm{d}s^2=-\mathrm{d}t^2+\mathrm{d}r^2+r^2\mathrm{d}\theta^2+r^2\sin^2\left(\theta\right)\mathrm{d}\phi^2$$

4.5.2 Displaying Volume Elements

[ ]:
```
minkowski.volume_element()
minkowski.volume_element(coords=spherical)
```

Minkowski:

$$\mathrm{d}V^2 = -1$$

Minkowski:

$$\mathrm{d}V^2 = -r^4 \sin^2(\theta)$$

4.6 Defining Tensors

4.6.1 Scalars

[ ]:
```
kretschmann = schwarschild.new_tensor(
    name="Kretschmann",
    components=sym.Array(48*M**2/r**6),
    indices=(),
    symbol="S"
)
kretschmann.show()
```

Kretschmann:

$$S(t, r, \theta, \phi) = \frac{48M^2}{r^6}$$

4.6.2 Generic Tensors

[ ]:
```
rho, p, v = sym.symbols("rho p v")

four_velocity = og.new_tensor(
    name="4-Velocity",
    components=1/sym.sqrt(1-v**2) * sym.Array([1, v, 0, 0]),
    indices=(1,),
    coords=cartesian,
    metric=minkowski,
    symbol="u"
)
four_velocity.show()
```

4-Velocity:

$$u^{\mu}\left(t,x,y,z\right)=\begin{pmatrix}\frac{1}{\sqrt{1-v^{2}}}\\ \frac{v}{\sqrt{1-v^{2}}}\\ 0\\ 0\end{pmatrix}$$

[ ]:
```
perfect_fluid = minkowski.new_tensor(
    name="Perfect Fluid",
    components=sym.diag(rho, p, p, p),
    indices=(1, 1),
    symbol="T"
)
perfect_fluid.show()
```

Perfect Fluid:

$$T^{\mu\nu}\left(t,x,y,z\right)=\begin{pmatrix}\rho & 0 & 0 & 0\\ 0 & p & 0 & 0\\ 0 & 0 & p & 0\\ 0 & 0 & 0 & p\end{pmatrix}$$

4.7 Transforming Tensors

[ ]:
```
minkowski.list(indices=(1, 1))
```

Minkowski:

$$g^{tt}=-g^{xx}=-g^{yy}=-g^{zz}=-1$$

[ ]:
```
minkowski.list(coords=spherical)
```

Minkowski:

$$g_{tt}=-g_{rr}=-1$$

$$g_{\theta\theta}=r^{2}$$

$$g_{\phi\phi}=r^{2}\sin^{2}\left(\theta\right)$$

[ ]:
```
minkowski.list(coords=spherical, indices=(1, 1))
```

Minkowski:

$$g^{tt} = -g^{rr} = -1$$

$$g^{\theta\theta} = \frac{1}{r^2}$$

$$g^{\phi\phi} = \frac{1}{r^2 \sin^2(\theta)}$$

4.8 Performing Substitutions and Mapping Functions

[ ]:
```
minkowski.list(coords=spherical, indices=(1, 1), replace={r: 2, theta: sym.pi/
↪4})
```

Minkowski:

$$g^{tt} = -g^{rr} = -1$$

$$g^{\theta\theta} = \frac{1}{4}$$

$$g^{\phi\phi} = \frac{1}{2}$$

[ ]:
```
def myfunc(x):
    return -4*x

minkowski.list(coords=spherical, indices=(1, 1), replace={r: 2, theta: sym.pi/
↪4}, function=myfunc)
```

Minkowski:

$$g^{tt} = -g^{rr} = 4$$

$$g^{\theta\theta} = -1$$

$$g^{\phi\phi} = -2$$

4.9 Retrieving Tensor Components

4.9.1 As a SymPy Array

[ ]:
```
sympy_coordinates = cartesian.get_components()
print(sympy_coordinates)

sympy_minkowski = minkowski.get_components(coords=spherical, indices=(-1, -1))
print(sympy_minkowski)
```

```
sympy_minkowski = minkowski.get_components(coords=spherical, indices=(-1, -1),␣
↪mode="sympy")
print(sympy_minkowski)
```

```
[t, x, y, z]
[[-1, 0, 0, 0], [0, 1, 0, 0], [0, 0, r**2, 0], [0, 0, 0, r**2*sin(theta)**2]]
[[-1, 0, 0, 0], [0, 1, 0, 0], [0, 0, r**2, 0], [0, 0, 0, r**2*sin(theta)**2]]
```

4.9.2 As a Mathematica List

[ ]:
```
mathematica_coordinates = cartesian.get_components(mode="mathematica")
print(mathematica_coordinates)

mathematica_minkowski = minkowski.get_components(coords=spherical, indices=(-1,␣
↪-1), mode="mathematica")
print(mathematica_minkowski)
```

```
{t, x, y, z}
{{-1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, r^2, 0}, {0, 0, 0, r^2*Sin[theta]^2}}
```

4.9.3 As a LaTex Expression

[ ]:
```
latex_coordinates = cartesian.get_components(mode="latex")
print(latex_coordinates)

latex_minkowski = minkowski.get_components(coords=spherical, indices=(-1, -1),␣
↪mode="latex")
print(latex_minkowski)
```

```
\left(\begin{matrix}t\\x\\y\\z\end{matrix}\right)
\left(\begin{matrix}-1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & r^{2} & 0\\0 & 0 & 0
& r^{2} \sin^{2}{\left(\theta \right)}\end{matrix}\right)
```

4.10 Changing Defaults

4.10.1 Setting the Index Letters

[ ]:
```
og.set_index_letters()
og.set_index_letters("a b c d e f g h i j k l m n o p")
og.set_index_letters("automatic")
```

$(\mu,\ \nu,\ \rho,\ \sigma,\ \kappa,\ \lambda,\ \alpha,\ \beta,\ \gamma,\ \delta,\ \epsilon,\ \zeta,\ \theta,\ \iota,\ \xi,\ \pi,\ \tau,\ \phi,\ \chi,\ \psi,\ \omega)$

$(a,\ b,\ c,\ d,\ e,\ f,\ g,\ h,\ i,\ j,\ k,\ l,\ m,\ n,\ o,\ p)$

$(\mu,\ \nu,\ \rho,\ \sigma,\ \kappa,\ \lambda,\ \alpha,\ \beta,\ \gamma,\ \delta,\ \epsilon,\ \zeta,\ \theta,\ \iota,\ \xi,\ \pi,\ \tau,\ \phi,\ \chi,\ \psi,\ \omega)$

4.10.2 Changing the Name of a Tensor

[ ]:
```
four_velocity.change_name("Four Velocity")
four_velocity.info()
```

Four Velocity:

Name: Four Velocity

Symbol: u

Type: Tensor

Rank: 1

Metric: Minkowski

Default Coordinates: Cartesian

Default Indices: (1,)

4.10.3 Changing the Tensor Symbol

[ ]:
```
kretschmann.change_symbol("K")
kretschmann.show()
```

Kretschmann:

$$K\left(t, r, \theta, \phi\right) = \frac{48M^2}{r^6}$$

4.10.4 Changing the Default Indices

[ ]:
```
four_velocity.change_default_indices(indices=(-1,))
four_velocity.list()
```

Four Velocity:

$$u_t = -\frac{1}{\sqrt{1 - v^2}}$$

$$u_x = \frac{v}{\sqrt{1 - v^2}}$$

4.10.5 Changing the Default Coordinates

[ ]:
```
four_velocity.change_default_coords(coords=spherical)
four_velocity.list()
```

Four Velocity:

$$u_t = -\frac{1}{\sqrt{1-v^2}}$$

$$u_r = \frac{v\sin(\theta)\cos(\phi)}{\sqrt{1-v^2}}$$

$$u_\theta = \frac{rv\cos(\phi)\cos(\theta)}{\sqrt{1-v^2}}$$

$$u_\phi = -\frac{rv\sin(\phi)\sin(\theta)}{\sqrt{1-v^2}}$$

[ ]:
```
four_velocity.change_default_coords(coords=cartesian).show()
```

Four Velocity:

$$u_\mu(t,x,y,z) = \begin{pmatrix} -\frac{1}{\sqrt{1-v^2}} \\ \frac{v}{\sqrt{1-v^2}} \\ 0 \\ 0 \end{pmatrix}$$

4.11 Deleting PyOGRe Objects

[ ]:
```
bad_coordinates = og.new_coordinates(
    name="I was a mistake",
    components=cartesian_symbols
)

bad_coordinates.delete()
```

I was a mistake:

Successfully Deleted

4.12 Retrieving All Tensor Objects

[ ]:
```
og.get_instances()
```

[ ]:
```
[Cartesian,
 Spherical,
 Minkowski,
 Schwarzschild,
 Kretschmann,
 Four Velocity,
 Perfect Fluid]
```

4.13 Cleaning Up Result Tensors

[ ]:
```
og.delete_results()
display(og.get_instances())
```

Deleted 0 tensor(s) named 'Result'.

```
[Cartesian,
 Spherical,
 Minkowski,
 Schwarzschild,
 Kretschmann,
 Four Velocity,
 Perfect Fluid]
```

5 Importing and Exporting Tensors

[ ]:
```
minkowski.export()
```

[ ]:
```
'<|"Minkowski"-><|"Components"-><|{{-1,-1},"Cartesian"}->{{-1, 0, 0, 0}, {0, 1,
0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}\\\\ReleaseHold,{{1,-1},"Cartesian"}->{{1, 0,
0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0,
1}}\\\\ReleaseHold,{{-1,1},"Cartesian"}->{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1,
0}, {0, 0, 0, 1}}\\\\ReleaseHold,{{1,1},"Cartesian"}->{{-1, 0, 0, 0}, {0, 1, 0,
0}, {0, 0, 1, 0}, {0, 0, 0, 1}}\\\\ReleaseHold,{{-1,-1},"Spherical"}->{{-1, 0,
0, 0}, {0, 1, 0, 0}, {0, 0, r^2, 0}, {0, 0, 0,
r^2*Sin[theta]^2}}\\\\ReleaseHold,{{1,1},"Spherical"}->{{-1, 0, 0, 0}, {0, 1, 0,
0}, {0, 0, r^(-2), 0}, {0, 0, 0,
1/(r^2*Sin[theta]^2)}}\\\\ReleaseHold,{{-1,1},"Spherical"}->{{1, 0, 0, 0}, {0,
1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}\\\\ReleaseHold,{{1,-1},"Spherical"}->{{1,
0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0,
1}}\\\\ReleaseHold|>,"DefaultCoords"->"Cartesian","DefaultIndices"->{-1, -1}\\\\
ReleaseHold,"Role"->"Metric","Symbol"->g\\\\ReleaseHold,"Metric"->"Minkowski","O
GReVersion"->"PyOGRe v1.0.0"|>|>'
```

[ ]:
```
og.export_all()
```

```
<|"Options"-><|"OGReVersion"->"PyOGRe
v1.0.0"|>,"Cartesian"-><|"Components"-><|{{1},"Cartesian"}->{t, x, y, z}\\Releas
eHold|>,"DefaultCoords"->"Cartesian","DefaultIndices"->{1}\\ReleaseHold,"Role"->
"Coordinates","Symbol"->x\\ReleaseHold,"Jacobians"-><|"Spherical"-><|"Jacobian"-
>{{1, 0, 0, 0}, {0, Sin[theta]*Cos[phi], r*Cos[phi]*Cos[theta],
-r*Sin[phi]*Sin[theta]}, {0, Sin[phi]*Sin[theta], r*Sin[phi]*Cos[theta],
r*Sin[theta]*Cos[phi]}, {0, Cos[theta], -r*Sin[theta],
0}}\\ReleaseHold,"InverseJacobian"->{{1, 0, 0, 0}, {0, Sin[theta]*Cos[phi],
Sin[phi]*Sin[theta], Cos[theta]}, {0, Cos[phi]*Cos[theta]/r,
Sin[phi]*Cos[theta]/r, -Sin[theta]/r}, {0, -Sin[phi]/(r*Sin[theta]),
Cos[phi]/(r*Sin[theta]), 0}}\\ReleaseHold,"ChristoffelJacobian"->{{{0, 0, 0, 0},
```

```
{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0,
Cos[phi]*Cos[theta], -Sin[phi]*Sin[theta]}, {0, Cos[phi]*Cos[theta],
-r*Sin[theta]*Cos[phi], -r*Sin[phi]*Cos[theta]}, {0, -Sin[phi]*Sin[theta],
-r*Sin[phi]*Cos[theta], -r*Sin[theta]*Cos[phi]}}, {{0, 0, 0, 0}, {0, 0,
Sin[phi]*Cos[theta], Sin[theta]*Cos[phi]}, {0, Sin[phi]*Cos[theta],
-r*Sin[phi]*Sin[theta], r*Cos[phi]*Cos[theta]}, {0, Sin[theta]*Cos[phi],
r*Cos[phi]*Cos[theta], -r*Sin[phi]*Sin[theta]}}, {{0, 0, 0, 0}, {0, 0,
-Sin[theta], 0}, {0, -Sin[theta], -r*Cos[theta], 0}, {0, 0, 0, 0}}}\\ReleaseHold
|>|>,"CoordTransformations"-><|"Spherical"->{x->r*Sin[theta]*Cos[phi]\\ReleaseHo
ld,y->r*Sin[phi]*Sin[theta]\\ReleaseHold,z->r*Cos[theta]\\ReleaseHold}|>|>,"Sphe
rical"-><|"Components"-><|{{1},"Spherical"}->{t, r, theta, phi}\\ReleaseHold|>,"
DefaultCoords"->"Spherical","DefaultIndices"->{1}\\ReleaseHold,"Role"->"Coordina
tes","Symbol"->x\\ReleaseHold,"CoordTransformations"-><|"Cartesian"->{r->(x^2 +
y^2 + z^2)^(1/2)\\ReleaseHold,theta->ArcCos[z/(x^2 + y^2 + z^2)^(1/2)]\\ReleaseH
old,phi->ArcTan[y/x]\\ReleaseHold}|>|>,"Minkowski"-><|"Components"-><|{{-1,-1},"
Cartesian"}->{{-1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0,
1}}\\ReleaseHold,{{1,-1},"Cartesian"}->{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1,
0}, {0, 0, 0, 1}}\\ReleaseHold,{{-1,1},"Cartesian"}->{{1, 0, 0, 0}, {0, 1, 0,
0}, {0, 0, 1, 0}, {0, 0, 0, 1}}\\ReleaseHold,{{1,1},"Cartesian"}->{{-1, 0, 0,
0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0,
1}}\\ReleaseHold,{{-1,-1},"Spherical"}->{{-1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0,
r^2, 0}, {0, 0, 0, r^2*Sin[theta]^2}}\\ReleaseHold,{{1,1},"Spherical"}->{{-1, 0,
0, 0}, {0, 1, 0, 0}, {0, 0, r^(-2), 0}, {0, 0, 0,
1/(r^2*Sin[theta]^2)}}\\ReleaseHold,{{-1,1},"Spherical"}->{{1, 0, 0, 0}, {0, 1,
0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}\\ReleaseHold,{{1,-1},"Spherical"}->{{1, 0, 0,
0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0,
1}}\\ReleaseHold|>,"DefaultCoords"->"Cartesian","DefaultIndices"->{-1, -1}\\Rele
aseHold,"Role"->"Metric","Symbol"->g\\ReleaseHold,"Metric"->"Minkowski"|>,"Schwa
rzschild"-><|"Components"-><|{{-1,-1},"Spherical"}->{{2*M/r - 1, 0, 0, 0}, {0,
(-2*M/r + 1)^(-1), 0, 0}, {0, 0, r^2, 0}, {0, 0, 0,
r^2*Sin[theta]^2}}\\ReleaseHold,{{1,-1},"Spherical"}->{{1, 0, 0, 0}, {0, 1, 0,
0}, {0, 0, 1, 0}, {0, 0, 0, 1}}\\ReleaseHold,{{-1,1},"Spherical"}->{{1, 0, 0,
0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0,
1}}\\ReleaseHold,{{1,1},"Spherical"}->{{(2*M/r - 1)^(-1), 0, 0, 0}, {0, -2*M/r +
1, 0, 0}, {0, 0, r^(-2), 0}, {0, 0, 0, 1/(r^2*Sin[theta]^2)}}\\ReleaseHold|>,"De
faultCoords"->"Spherical","DefaultIndices"->{-1, -1}\\ReleaseHold,"Role"->"Metri
c","Symbol"->g\\ReleaseHold,"Metric"->"Schwarzschild"|>,"Kretschmann"-><|"Compon
ents"-><|{{},"Spherical"}->{48*M^2/r^6}\\ReleaseHold|>,"DefaultCoords"->"Spheric
al","DefaultIndices"->{}\\ReleaseHold,"Role"->"Tensor","Symbol"->K\\ReleaseHold,
"Metric"->"Schwarzschild"|>,"Four
Velocity"-><|"Components"-><|{{1},"Cartesian"}->{(1 - v^2)^(-1/2), v/(1 -
v^2)^(1/2), 0, 0}\\ReleaseHold,{{-1},"Cartesian"}->{-1/(1 - v^2)^(1/2), v/(1 -
v^2)^(1/2), 0, 0}\\ReleaseHold,{{-1},"Spherical"}->{-1/(1 - v^2)^(1/2),
v*Sin[theta]*Cos[phi]/(1 - v^2)^(1/2), r*v*Cos[phi]*Cos[theta]/(1 - v^2)^(1/2),
-r*v*Sin[phi]*Sin[theta]/(1 - v^2)^(1/2)}\\ReleaseHold|>,"DefaultCoords"->"Carte
sian","DefaultIndices"->{-1}\\ReleaseHold,"Role"->"Tensor","Symbol"->u\\ReleaseH
old,"Metric"->"Minkowski"|>,"Perfect
Fluid"-><|"Components"-><|{{1,1},"Cartesian"}->{{rho, 0, 0, 0}, {0, p, 0, 0},
```

```
{0, 0, p, 0}, {0, 0, 0,
p}}\\ReleaseHold|>,"DefaultCoords"->"Cartesian","DefaultIndices"->{1, 1}\\Releas
eHold,"Role"->"Tensor","Symbol"->T\\ReleaseHold,"Metric"->"Minkowski"|>|>
```

[ ]:
```
cartesian.export("cartesian.txt")
```

Exported Cartesian to cartesian.txt

[ ]:
```
og.export_all("tensors.txt")
```

Exported all tensors to tensors.txt

[ ]:
```
og.import_from_string(
'<|"Imported Minkowski"-><|"Components"-><|{{-1,-1},"Cartesian"}->{{-1, 0, 0,␣
↪0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}},{{1,-1},"Cartesian"}->{{1, 0,␣
↪0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}},{{-1,1},"Cartesian"}->{{1,␣
↪0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0,␣
↪1}},{{1,1},"Cartesian"}->{{-1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0,␣
↪0, 1}},{{-1,-1},"Spherical"}->{{-1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, r^2, 0},␣
↪{0, 0, 0, r^2*Sin[theta]^2}},{{1,1},"Spherical"}->{{-1, 0, 0, 0}, {0, 1, 0,␣
↪0}, {0, 0, r^(-2), 0}, {0, 0, 0, 1/
↪(r^2*Sin[theta]^2)}},{{-1,1},"Spherical"}->{{1, 0, 0, 0}, {0, 1, 0, 0}, {0,␣
↪0, 1, 0}, {0, 0, 0, 1}},{{1,-1},"Spherical"}->{{1, 0, 0, 0}, {0, 1, 0, 0},␣
↪{0, 0, 1, 0}, {0, 0, 0,␣
↪1}}|>,"DefaultCoords"->"Cartesian","DefaultIndices"->{-1,␣
↪-1},"Role"->"Metric","Symbol"->g,"Metric"->"Minkowski","OGReVersion"->"PyOGRe␣
↪v0.0.1"|>|>'
).info()
```

Imported Minkowski:

Name: Imported Minkowski

Symbol: g

Type: Metric

Rank: 2

Default Coordinates: Cartesian

Default Indices: (-1, -1)

Tensors Using This Metric: Imported Minkowski

[ ]:
```
og.import_from_file("cartesian.txt").change_name("Imported Cartesian").show()
```

Imported Cartesian:

$$x^{\mu} = \begin{pmatrix} t \\ x \\ y \\ z \end{pmatrix}$$

[ ]:
```
# tensors_list = og.import_all_from_string("TensorData")
# tensors_list = og.import_all_from_file("tensors.txt")
```

6 Built-in Tensors

[ ]:
```
from PyOGRe.Defaults import cartesian as default_cartesian
from PyOGRe.Defaults import spherical as default_spherical
from PyOGRe.Defaults import minkowski as default_minkowski
from PyOGRe.Defaults import schwarschild as default_schwarschild
from PyOGRe.Defaults import flrw as default_flrw
```

[ ]:
```
default_cartesian.show()
default_spherical.show()
default_minkowski.show()
default_schwarschild.show()
default_flrw.show()
```

4D Cartesian:

$$x^{\mu} = \begin{pmatrix} t \\ x \\ y \\ z \end{pmatrix}$$

4D Spherical:

$$x^{\mu} = \begin{pmatrix} t \\ r \\ \theta \\ \phi \end{pmatrix}$$

4D Minkowski:

$$g_{\mu\nu}\left(t, x, y, z\right) = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Schwarzschild:

$$g_{\mu\nu}\left(t, r, \theta, \phi\right) = \begin{pmatrix} \frac{2M}{r} - 1 & 0 & 0 & 0 \\ 0 & \frac{1}{-\frac{2M}{r}+1} & 0 & 0 \\ 0 & 0 & r^2 & 0 \\ 0 & 0 & 0 & r^2 \sin^2\left(\theta\right) \end{pmatrix}$$

FLRW:

$$g_{\mu\nu}\left(t, r, \theta, \phi\right) = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & \frac{a^2(t)}{-kr^2+1} & 0 & 0 \\ 0 & 0 & r^2a^2(t) & 0 \\ 0 & 0 & 0 & r^2a^2(t)\sin^2\left(\theta\right) \end{pmatrix}$$

7 Performing Tensor Calculations

7.1 Calc Module

[ ]:
```
print(og.Calc.__doc__)
```

```
    Tensor Calculation Function.
```

7.2 Simplifying Tensors

[ ]:
```
minkowski.simplify().show()
```

```
Simplifying:   0%|          | 0/128 [00:00<?, ?it/s]
```

Minkowski:

$$g_{\mu\nu}\left(t, x, y, z\right) = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

7.3 Addition and Subtraction of Tensors

[ ]:
```
og.Calc(
    minkowski("mu nu") + perfect_fluid("mu nu"),
).show(replace={v: 0})
```

Result:

$$\square_{\mu\nu}\left(t, x, y, z\right) = \begin{pmatrix} \rho - 1 & 0 & 0 & 0 \\ 0 & p + 1 & 0 & 0 \\ 0 & 0 & p + 1 & 0 \\ 0 & 0 & 0 & p + 1 \end{pmatrix}$$

[ ]:
```
og.Calc(
    minkowski("mu nu") + perfect_fluid("mu nu"),
    symbol="S",
```

```
    name="Sum Result"
).show(replace={v: 0})
```

Sum Result:

$$S_{\mu\nu}\left(t,x,y,z\right) = \begin{pmatrix} \rho - 1 & 0 & 0 & 0 \\ 0 & p + 1 & 0 & 0 \\ 0 & 0 & p + 1 & 0 \\ 0 & 0 & 0 & p + 1 \end{pmatrix}$$

[ ]:
```
non_symmetric = minkowski.new_tensor(
    name="Non-Symmetric",
    components=sym.Array([[0, 0, 0, 1], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0,␣
 ↪0]]),
    symbol="N",
    indices=(-1, -1)
)
non_symmetric.show()
```

Non-Symmetric:

$$N_{\mu\nu}\left(t,x,y,z\right) = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

[ ]:
```
og.Calc(
    minkowski("mu nu") + non_symmetric("mu nu")
).show()
```

Result:

$$\square_{\mu\nu}\left(t,x,y,z\right) = \begin{pmatrix} -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

[ ]:
```
og.Calc(
    minkowski("mu nu") + non_symmetric("nu mu")
).show()
```

Result:

$$\square_{\mu\nu}\left(t,x,y,z\right) = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

[ ]:
```
og.Calc(
    minkowski("mu nu") + perfect_fluid("mu nu") + non_symmetric("mu nu") +␣
 ↪non_symmetric("nu mu")
).show(replace={v: 0})
```

Result:

$$\square_{\mu\nu}\left(t,x,y,z\right) = \begin{pmatrix} \rho - 1 & 0 & 0 & 1 \\ 0 & p + 1 & 0 & 0 \\ 0 & 0 & p + 1 & 0 \\ 1 & 0 & 0 & p + 1 \end{pmatrix}$$

7.4 Multiplication of Tensor by a Scalar

[ ]:
```
og.Calc(
    2 * minkowski("a b")
).show()
```

Result:

$$\square_{\mu\nu}\left(t,x,y,z\right) = \begin{pmatrix} -2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

[ ]:
```
og.Calc(
    kretschmann() * minkowski("mu nu")
).show(coords=spherical)
```

Result:

$$\square_{\mu\nu}\left(t,r,\theta,\phi\right) = \begin{pmatrix} -\frac{48M^2}{r^6} & 0 & 0 & 0 \\ 0 & \frac{48M^2}{r^6} & 0 & 0 \\ 0 & 0 & \frac{48M^2}{r^4} & 0 \\ 0 & 0 & 0 & \frac{48M^2\sin^2\left(\theta\right)}{r^4} \end{pmatrix}$$

[ ]:
```
og.Calc(
    2 * t * minkowski("mu nu") - 3 * x * perfect_fluid("mu nu") + 4 * y *␣
 ↪non_symmetric("mu nu") - 5 * z * non_symmetric("nu mu")
).show(replace={v: 0})
```

Result:

$$\Box_{\mu\nu}(t,x,y,z) = \begin{pmatrix} -3\rho x - 2t & 0 & 0 & 4y \\ 0 & -3px + 2t & 0 & 0 \\ 0 & 0 & -3px + 2t & 0 \\ -5z & 0 & 0 & -3px + 2t \end{pmatrix}$$

7.5 Traces and Contractions

7.5.1 Theoretical Review

7.5.2 PyOGRe Syntax

[ ]:
```
perfect_fluid_from_velocity = og.Calc(
    (rho + p) * four_velocity("mu") @ four_velocity("nu") + p * minkowski("mu␣
 ↪nu"),
    name="Perfect fluid",
    symbol="T",
    indices="mu nu"
)
perfect_fluid_from_velocity.show()
```

Perfect fluid:

$$T_{\mu\nu}(t,x,y,z) = \begin{pmatrix} \frac{-pv^2-\rho}{v^2-1} & \frac{v(p+\rho)}{v^2-1} & 0 & 0 \\ \frac{v(p+\rho)}{v^2-1} & \frac{-p-\rho v^2}{v^2-1} & 0 & 0 \\ 0 & 0 & p & 0 \\ 0 & 0 & 0 & p \end{pmatrix}$$

[ ]:
```
perfect_fluid_from_velocity.show(indices=(1,1), replace={v: 0})
```

Perfect fluid:

$$T^{\mu\nu}(t,x,y,z) = \begin{pmatrix} \rho & 0 & 0 & 0 \\ 0 & p & 0 & 0 \\ 0 & 0 & p & 0 \\ 0 & 0 & 0 & p \end{pmatrix}$$

[ ]:
```
og.Calc(
    kretschmann() @ minkowski("mu nu")
).show()
```

Result:

$$\square_{\mu\nu}(t,x,y,z) = \begin{pmatrix} -\frac{48M^2}{(x^2+y^2+z^2)^3} & 0 & 0 & 0 \\ 0 & \frac{48M^2}{(x^2+y^2+z^2)^3} & 0 & 0 \\ 0 & 0 & \frac{48M^2}{(x^2+y^2+z^2)^3} & 0 \\ 0 & 0 & 0 & \frac{48M^2}{(x^2+y^2+z^2)^3} \end{pmatrix}$$

[ ]:
```
og.Calc(
    kretschmann() * kretschmann()
).show()
```

Result:

$$\square(t,r,\theta,\phi) = \frac{2304M^4}{r^{12}}$$

[ ]:
```
og.Calc(
    minkowski("mu mu")
).show()
```

Result:

$$\square(t,x,y,z) = 4$$

[ ]:
```
og.Calc(
    perfect_fluid_from_velocity("mu mu")
).show()
```

Result:

$$\square(t,x,y,z) = 3p - \rho$$

[ ]:
```
display(
    og.str_symbols("x0:10")
)
```

```
'x0 x1 x2 x3 x4 x5 x6 x7 x8 x9'
```

[ ]:
```
og.Calc(
    perfect_fluid_from_velocity(og.str_symbols("x0:2")) @␣
 ↪perfect_fluid_from_velocity(og.str_symbols("x0:2"))
).show()
```

Result:

$$\square(t,x,y,z) = 3p^2 + \rho^2$$

7.6 Norm Squared

[ ]:
```
minkowski.calc_norm_squared().show()
```

Minkowski Norm Squared:

$$|g|^2 (t, x, y, z) = 4$$

[ ]:
```
four_velocity.calc_norm_squared().show()
```

Four Velocity Norm Squared:

$$|u|^2 (t, x, y, z) = -1$$

7.7 Derivatives and Christoffel Symbols

7.7.1 Partial Derivative

[ ]:
```
from PyOGRe import PartialD
```

[ ]:
```
og.Calc(
    PartialD("mu") @ kretschmann()
).show(coords=spherical)
```

Result:

$$\Box_\mu (t, r, \theta, \phi) = \begin{pmatrix} 0 \\ -\frac{288M^2}{r^7} \\ 0 \\ 0 \end{pmatrix}$$

[ ]:
```
og.Calc(
    PartialD("mu") @ schwarschild("a b")
).list()
```

Result:

$$\square_{rtt} = -\frac{2M}{r^2}$$

$$\square_{rrr} = -\frac{2M}{(2M - r)^2}$$

$$\square_{r\theta\theta} = 2r$$

$$\square_{r\phi\phi} = 2r \sin^2(\theta)$$

$$\square_{\theta\phi\phi} = r^2 \sin(2\theta)$$

7.7.2 Christoffel Symbols

[ ]:
```
og.Calc(
    (1/2) * schwarschild("lambda sigma") @ (
        PartialD("mu") @ schwarschild("nu sigma") +
        PartialD("nu") @ schwarschild("sigma mu") -
        PartialD("sigma") @ schwarschild("mu nu")
    ),
    name="Schwarzschild Christoffel Manual",
    symbol="Gamma",
).change_default_indices(indices=(1, -1, -1)).list()
```

Schwarzschild Christoffel Manual:

$$\Gamma^{t}{}_{tr} = \Gamma^{t}{}_{rt} = \frac{M}{r(-2M + r)}$$

$$\Gamma^{r}{}_{tt} = \frac{M(-2M + r)}{r^3}$$

$$\Gamma^{r}{}_{rr} = \frac{M}{r(2M - r)}$$

$$\Gamma^{r}{}_{\theta\theta} = 2M - r$$

$$\Gamma^{r}{}_{\phi\phi} = (2M - r)\sin^2(\theta)$$

$$\Gamma^{\theta}{}_{r\theta} = \Gamma^{\theta}{}_{\theta r} = \Gamma^{\phi}{}_{r\phi} = \Gamma^{\phi}{}_{\phi r} = \frac{1}{r}$$

$$\Gamma^{\theta}{}_{\phi\phi} = -\frac{\sin(2\theta)}{2}$$

$$\Gamma^{\phi}{}_{\theta\phi} = \Gamma^{\phi}{}_{\phi\theta} = \frac{1}{\tan(\theta)}$$

[ ]:
```
schwarschild_cs = schwarschild.calc_christoffel()
schwarschild_cs.list()
```

Schwarzschild Christoffel:

$$\Gamma^{t}{}_{tr} = \Gamma^{t}{}_{rt} = \frac{M}{r\left(-2M + r\right)}$$

$$\Gamma^{r}{}_{tt} = \frac{M\left(-2M + r\right)}{r^{3}}$$

$$\Gamma^{r}{}_{rr} = \frac{M}{r\left(2M - r\right)}$$

$$\Gamma^{r}{}_{\theta\theta} = 2M - r$$

$$\Gamma^{r}{}_{\phi\phi} = \left(2M - r\right)\sin^{2}\left(\theta\right)$$

$$\Gamma^{\theta}{}_{r\theta} = \Gamma^{\theta}{}_{\theta r} = \Gamma^{\phi}{}_{r\phi} = \Gamma^{\phi}{}_{\phi r} = \frac{1}{r}$$

$$\Gamma^{\theta}{}_{\phi\phi} = -\frac{\sin\left(2\theta\right)}{2}$$

$$\Gamma^{\phi}{}_{\theta\phi} = \Gamma^{\phi}{}_{\phi\theta} = \frac{1}{\tan\left(\theta\right)}$$

[ ]:
```
simple_metric = cartesian.new_metric(
    name="Simple Metric",
    components=sym.Array(
        [
            [-x, 0, 0, 0],
            [0, 1, 0, 0],
            [0, 0, 1, 0],
            [0, 0, 0, 1]
        ]
    ),
)
simple_metric.show()
```

Simple Metric:

$$g_{\mu\nu}\left(t, x, y, z\right) = \begin{pmatrix} -x & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

[ ]:
```
christoffel_manual = og.Calc(
    (1/2) * simple_metric("lambda sigma") @ (
        PartialD("mu") @ simple_metric("nu sigma") +
        PartialD("nu") @ simple_metric("sigma mu") -
        PartialD("sigma") @ simple_metric("mu nu")
    ),
    name="Simple Metric Christoffel Manual",
    symbol="Gamma"
)
christoffel_manual.change_default_indices(indices=(1,-1,-1))
christoffel_manual.list()
christoffel_manual.list(coords=spherical)
```

Simple Metric Christoffel Manual:

$$\Gamma^{t}{}_{tx} = \Gamma^{t}{}_{xt} = \frac{1}{2x}$$

$$\Gamma^{x}{}_{tt} = \frac{1}{2}$$

Simple Metric Christoffel Manual:

$$\Gamma^{t}{}_{tr} = \Gamma^{t}{}_{rt} = \frac{1}{2r}$$

$$\Gamma^{t}{}_{t\theta} = \Gamma^{t}{}_{\theta t} = \frac{1}{2\tan(\theta)}$$

$$\Gamma^{t}{}_{t\phi} = \Gamma^{t}{}_{\phi t} = -\frac{\tan(\phi)}{2}$$

$$\Gamma^{r}{}_{tt} = \frac{\sin(\theta)\cos(\phi)}{2}$$

$$\Gamma^{\theta}{}_{tt} = \frac{\cos(\phi)\cos(\theta)}{2r}$$

$$\Gamma^{\phi}{}_{tt} = -\frac{\sin(\phi)}{2r\sin(\theta)}$$

[ ]:
```
christoffel_auto = simple_metric.calc_christoffel()
christoffel_auto.list()
christoffel_auto.list(coords=spherical)
```

Simple Metric Christoffel:

$$\Gamma^{t}{}_{tx} = \Gamma^{t}{}_{xt} = \frac{1}{2x}$$

$$\Gamma^{x}{}_{tt} = \frac{1}{2}$$

Simple Metric Christoffel:

$$\Gamma^{t}{}_{tr} = \Gamma^{t}{}_{rt} = \frac{1}{2r}$$

$$\Gamma^{t}{}_{t\theta} = \Gamma^{t}{}_{\theta t} = \frac{1}{2\tan\left(\theta\right)}$$

$$\Gamma^{t}{}_{t\phi} = \Gamma^{t}{}_{\phi t} = -\frac{\tan\left(\phi\right)}{2}$$

$$\Gamma^{r}{}_{tt} = \frac{\sin\left(\theta\right)\cos\left(\phi\right)}{2}$$

$$\Gamma^{r}{}_{\theta\theta} = -r$$

$$\Gamma^{r}{}_{\phi\phi} = -r\sin^{2}\left(\theta\right)$$

$$\Gamma^{\theta}{}_{tt} = \frac{\cos\left(\phi\right)\cos\left(\theta\right)}{2r}$$

$$\Gamma^{\theta}{}_{r\theta} = \Gamma^{\theta}{}_{\theta r} = \Gamma^{\phi}{}_{r\phi} = \Gamma^{\phi}{}_{\phi r} = \frac{1}{r}$$

$$\Gamma^{\theta}{}_{\phi\phi} = -\frac{\sin\left(2\theta\right)}{2}$$

$$\Gamma^{\phi}{}_{tt} = -\frac{\sin\left(\phi\right)}{2r\sin\left(\theta\right)}$$

$$\Gamma^{\phi}{}_{\theta\phi} = \Gamma^{\phi}{}_{\phi\theta} = \frac{1}{\tan\left(\theta\right)}$$

[ ]:
```
a = sym.Function("a")(t)
k = sym.symbols("k")
flrw = spherical.new_metric(
    name="FLRW",
    components=sym.Array(
        [
            [-1, 0, 0, 0],
            [0, a**2/(1-k*r**2), 0, 0],
            [0, 0, a**2*r**2, 0],
            [0, 0, 0, a**2*r**2*sym.sin(theta)**2]
```

```
        ]
    )
)
flrw.show()
```

FLRW:

$$g_{\mu\nu}\left(t,r,\theta,\phi\right)=\begin{pmatrix}-1 & 0 & 0 & 0\\ 0 & \frac{a^2(t)}{-kr^2+1} & 0 & 0\\ 0 & 0 & r^2a^2(t) & 0\\ 0 & 0 & 0 & r^2a^2(t)\sin^2\left(\theta\right)\end{pmatrix}$$

[ ]:
```
flrw.calc_christoffel().list()
```

FLRW Christoffel:

$$\Gamma^{t}{}_{rr}=-\frac{a(t)\frac{d}{dt}a(t)}{kr^2-1}$$

$$\Gamma^{t}{}_{\theta\theta}=r^2a(t)\frac{d}{dt}a(t)$$

$$\Gamma^{t}{}_{\phi\phi}=r^2a(t)\sin^2\left(\theta\right)\frac{d}{dt}a(t)$$

$$\Gamma^{r}{}_{tr}=\Gamma^{r}{}_{rt}=\Gamma^{\theta}{}_{t\theta}=\Gamma^{\theta}{}_{\theta t}=\Gamma^{\phi}{}_{t\phi}=\Gamma^{\phi}{}_{\phi t}=\frac{\frac{d}{dt}a(t)}{a(t)}$$

$$\Gamma^{r}{}_{rr}=-\frac{kr}{kr^2-1}$$

$$\Gamma^{r}{}_{\theta\theta}=kr^3-r$$

$$\Gamma^{r}{}_{\phi\phi}=r\left(kr^2-1\right)\sin^2\left(\theta\right)$$

$$\Gamma^{\theta}{}_{r\theta}=\Gamma^{\theta}{}_{\theta r}=\Gamma^{\phi}{}_{r\phi}=\Gamma^{\phi}{}_{\phi r}=\frac{1}{r}$$

$$\Gamma^{\theta}{}_{\phi\phi}=-\frac{\sin\left(2\theta\right)}{2}$$

$$\Gamma^{\phi}{}_{\theta\phi}=\Gamma^{\phi}{}_{\phi\theta}=\frac{1}{\tan\left(\theta\right)}$$

7.7.3 Covariant Derivative

[ ]:
```
from PyOGRe import CovariantD
```

[ ]:
```
og.Calc(
    PartialD("mu") @ schwarschild("alpha beta") -
    schwarschild_cs("lambda mu alpha") @ schwarschild("lambda beta") -
    schwarschild_cs("lambda mu beta") @ schwarschild("alpha lambda"),
).list()
```

Result:

No Non-Zero Elements

[ ]:
```
og.Calc(
    CovariantD("mu") @ schwarschild("alpha beta")
).list()
```

Result:

No Non-Zero Elements

[ ]:
```
og.Calc(
    CovariantD("mu") @ flrw("alpha beta")
).list()
```

Result:

No Non-Zero Elements

7.8 Curvature Tensors

7.8.1 Riemann Tensor

[ ]:
```
flrw.calc_riemann_tensor().list()
```

FLRW Riemann:

$$R^{t}{}_{rtr} = -R^{t}{}_{rrt} = -\frac{a(t)\frac{d^2}{dt^2}a(t)}{kr^2 - 1}$$

$$R^{t}{}_{\theta t\theta} = -R^{t}{}_{\theta\theta t} = r^2 a(t)\frac{d^2}{dt^2}a(t)$$

$$R^{t}{}_{\phi t\phi} = -R^{t}{}_{\phi\phi t} = r^2 a(t)\sin^2(\theta)\frac{d^2}{dt^2}a(t)$$

$$R^{r}{}_{ttr} = -R^{r}{}_{trt} = R^{\theta}{}_{tt\theta} = -R^{\theta}{}_{t\theta t} = R^{\phi}{}_{tt\phi} = -R^{\phi}{}_{t\phi t} = \frac{\frac{d^2}{dt^2}a(t)}{a(t)}$$

$$R^{r}{}_{\theta r\theta} = R^{\phi}{}_{\theta\phi\theta} = r^2\left(k + \left(\frac{d}{dt}a(t)\right)^2\right)$$

$$R^{r}{}_{\theta\theta r} = R^{\phi}{}_{\theta\theta\phi} = r^2\left(-k - \left(\frac{d}{dt}a(t)\right)^2\right)$$

$$R^{r}{}_{\phi r\phi} = R^{\theta}{}_{\phi\theta\phi} = r^2\left(k + \left(\frac{d}{dt}a(t)\right)^2\right)\sin^2(\theta)$$

$$R^{r}{}_{\phi\phi r} = R^{\theta}{}_{\phi\phi\theta} = r^2\left(-k - \left(\frac{d}{dt}a(t)\right)^2\right)\sin^2(\theta)$$

$$R^{\theta}{}_{rr\theta} = R^{\phi}{}_{rr\phi} = \frac{k + \left(\frac{d}{dt}a(t)\right)^2}{kr^2 - 1}$$

$$R^{\theta}{}_{r\theta r} = R^{\phi}{}_{r\phi r} = \frac{-k - \left(\frac{d}{dt}a(t)\right)^2}{kr^2 - 1}$$

7.8.2 Ricci Tensor

[ ]:
```
flrw.calc_ricci_tensor().list()
```

FLRW Ricci Tensor:

$$R_{tt} = -\frac{3\frac{d^2}{dt^2}a(t)}{a(t)}$$

$$R_{rr} = \frac{-2k - a(t)\frac{d^2}{dt^2}a(t) - 2\left(\frac{d}{dt}a(t)\right)^2}{kr^2 - 1}$$

$$R_{\theta\theta} = r^2 \cdot \left(2k + a(t)\frac{d^2}{dt^2}a(t) + 2\left(\frac{d}{dt}a(t)\right)^2\right)$$

$$R_{\phi\phi} = r^2 \cdot \left(2k + a(t)\frac{d^2}{dt^2}a(t) + 2\left(\frac{d}{dt}a(t)\right)^2\right)\sin^2(\theta)$$

7.8.3 Ricci Scalar

[ ]:
```
flrw.calc_ricci_scalar().list()
```

FLRW Ricci Scalar:

$$R = \frac{6\left(k + a(t)\frac{d^2}{dt^2}a(t) + \left(\frac{d}{dt}a(t)\right)^2\right)}{a^2(t)}$$

7.8.4 Einstein Tensor

[ ]:
```
flrw.calc_einstein_tensor().list()
```

FLRW Einstein:

$$G_{tt} = \frac{3\left(k + \left(\frac{d}{dt}a(t)\right)^2\right)}{a^2(t)}$$

$$G_{rr} = \frac{k + 2a(t)\frac{d^2}{dt^2}a(t) + \left(\frac{d}{dt}a(t)\right)^2}{kr^2 - 1}$$

$$G_{\theta\theta} = r^2\left(-k - 2a(t)\frac{d^2}{dt^2}a(t) - \left(\frac{d}{dt}a(t)\right)^2\right)$$

$$G_{\phi\phi} = r^2\left(-k - 2a(t)\frac{d^2}{dt^2}a(t) - \left(\frac{d}{dt}a(t)\right)^2\right)\sin^2(\theta)$$

[ ]:
```
flrw.calc_einstein_tensor().list()
```

FLRW Einstein:

$$G_{tt} = \frac{3\left(k + \left(\frac{d}{dt}a(t)\right)^2\right)}{a^2(t)}$$

$$G_{rr} = \frac{k + 2a(t)\frac{d^2}{dt^2}a(t) + \left(\frac{d}{dt}a(t)\right)^2}{kr^2 - 1}$$

$$G_{\theta\theta} = r^2\left(-k - 2a(t)\frac{d^2}{dt^2}a(t) - \left(\frac{d}{dt}a(t)\right)^2\right)$$

$$G_{\phi\phi} = r^2\left(-k - 2a(t)\frac{d^2}{dt^2}a(t) - \left(\frac{d}{dt}a(t)\right)^2\right)\sin^2(\theta)$$

7.8.5 Weyl Tensor

[ ]:
```
schwarschild.calc_weyl_tensor().list()
flrw.calc_weyl_tensor().list()
```

Schwarzschild Weyl Tensor:

$$C_{trtr} = -C_{trrt} = -C_{rttr} = C_{rtrt} = -\frac{2M}{r^3}$$

$$C_{t\theta t\theta} = C_{\theta t\theta t} = \frac{M(-2M + r)}{r^2}$$

$$C_{t\theta\theta t} = C_{\theta tt\theta} = \frac{M(2M - r)}{r^2}$$

$$C_{t\phi t\phi} = C_{\phi t\phi t} = \frac{M(-2M + r)\sin^2(\theta)}{r^2}$$

$$C_{t\phi\phi t} = C_{\phi tt\phi} = \frac{M(2M - r)\sin^2(\theta)}{r^2}$$

$$C_{r\theta r\theta} = C_{\theta r\theta r} = \frac{M}{2M - r}$$

$$C_{r\theta\theta r} = C_{\theta rr\theta} = \frac{M}{-2M + r}$$

$$C_{r\phi r\phi} = C_{\phi r\phi r} = \frac{M\sin^2(\theta)}{2M - r}$$

$$C_{r\phi\phi r} = C_{\phi rr\phi} = \frac{M\sin^2(\theta)}{-2M + r}$$

$$C_{\theta\phi\theta\phi} = -C_{\theta\phi\phi\theta} = -C_{\phi\theta\theta\phi} = C_{\phi\theta\phi\theta} = 2Mr\sin^2(\theta)$$

FLRW Weyl Tensor:

No Non-Zero Elements

7.8.6 Non-trivial Covariant Derivative Example

[ ]:
```
flrw_einstein = flrw.calc_einstein_tensor()
og.Calc(
    CovariantD("mu") @ flrw_einstein("mu nu")
).list()
```

Result:

No Non-Zero Elements

[ ]:
```
f_rho = sym.Function("rho")(t, r, theta, phi)
f_p = sym.Function("p")(t, r, theta, phi)

flrw_rest_velocity = flrw.new_tensor(
    name="FLRW Rest Velocity",
    components=sym.Array(
        [1, 0, 0, 0]
    ),
    indices=(1,),
    symbol="u"
)
flrw_rest_velocity.show()

flrw_perfect_fluid = og.Calc(
    (f_rho + f_p) * flrw_rest_velocity("mu") @ flrw_rest_velocity("nu") + f_p *␣
 ↪flrw("mu nu"),
    name="FLRW Perfect Fluid",
    symbol="T"
)
flrw_perfect_fluid.show()
```

FLRW Rest Velocity:

$$u^{\mu}\left(t, r, \theta, \phi\right) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

FLRW Perfect Fluid:

$$T^{\mu\nu}\left(t,r,\theta,\phi\right) = \begin{pmatrix} \rho(t,r,\theta,\phi) & 0 & 0 & 0 \\ 0 & \frac{\left(-kr^2+1\right)p(t,r,\theta,\phi)}{a^2(t)} & 0 & 0 \\ 0 & 0 & \frac{p(t,r,\theta,\phi)}{r^2a^2(t)} & 0 \\ 0 & 0 & 0 & \frac{p(t,r,\theta,\phi)}{r^2a^2(t)\sin^2(\theta)} \end{pmatrix}$$

[ ]:
```
flrw_conservation = og.Calc(
    CovariantD("mu") @ flrw_perfect_fluid("mu nu"),
    name="FLRW Conservation",
    symbol="C"
)
flrw_conservation.list()
```

FLRW Conservation:

$$C^{t} = \frac{3\left(p(t,r,\theta,\phi)+\rho(t,r,\theta,\phi)\right)\frac{d}{dt}a(t)+a(t)\frac{\partial}{\partial t}\rho(t,r,\theta,\phi)}{a(t)}$$

$$C^{r} = \frac{\left(-kr^2+1\right)\frac{\partial}{\partial r}p(t,r,\theta,\phi)}{a^2(t)}$$

$$C^{\theta} = \frac{\frac{\partial}{\partial \theta}p(t,r,\theta,\phi)}{r^2a^2(t)}$$

$$C^{\phi} = \frac{\frac{\partial}{\partial \phi}p(t,r,\theta,\phi)}{r^2a^2(t)\sin^2(\theta)}$$

8 Curves and Geodesics

8.1 Setting the Curve Parameter

[ ]:
```
og.set_curve_parameter()
og.set_curve_parameter("kappa")
og.set_curve_parameter("automatic")
```

λ

κ

λ

8.2 The Curve Lagrangian

[ ]:
```
minkowski.calc_lagrangian().show()
```

Minkowski Lagrangian:

$$\mathcal{L}\left(t, x, y, z\right) = -\dot{t}^2 + \dot{x}^2 + \dot{y}^2 + \dot{z}^2$$

[ ]:
```
minkowski.calc_lagrangian().show(coords=spherical)
minkowski.calc_lagrangian(coords=spherical).show()
```

Minkowski Lagrangian:

$$\mathcal{L}\left(t, r, \theta, \phi\right) = \dot{\phi}^2 r^2 \sin^2\left(\theta\right) + r^2\dot{\theta}^2 + \dot{r}^2 - \dot{t}^2$$

Minkowski Lagrangian:

$$\mathcal{L}\left(t, r, \theta, \phi\right) = \dot{\phi}^2 r^2 \sin^2\left(\theta\right) + r^2\dot{\theta}^2 + \dot{r}^2 - \dot{t}^2$$

[ ]:
```
schwarschild.calc_lagrangian().list()
```

Schwarzschild Lagrangian:

$$\mathcal{L} = \frac{r^3 \cdot \left(2M - r\right)\left(\dot{\phi}^2 \sin^2\left(\theta\right) + \dot{\theta}^2\right) - r^2\dot{r}^2 + \dot{t}^2\left(2M - r\right)^2}{r\left(2M - r\right)}$$

[ ]:
```
flrw.calc_lagrangian().show()
```

FLRW Lagrangian:

$$\mathcal{L}\left(t, r, \theta, \phi\right) = \frac{-\dot{r}^2 a^2(t) + \left(kr^2 - 1\right)\left(\dot{\phi}^2 r^2 a^2(t)\sin^2\left(\theta\right) + r^2\dot{\theta}^2 a^2(t) - \dot{t}^2\right)}{kr^2 - 1}$$

8.3 Geodesics From the Lagrangian

[ ]:
```
minkowski.calc_geodesic_from_lagrangian().list()
```

Minkowski Geodesic From Lagrangian:

$$0^t = -\ddot{t}$$

$$0^x = \ddot{x}$$

$$0^y = \ddot{y}$$

$$0^z = \ddot{z}$$

[ ]:
```
schwarschild.calc_geodesic_from_lagrangian().list(replace={M: 1, theta: 0, phi:␣
↪0})
```

Schwarzschild Geodesic From Lagrangian:

$$0^t = -\ddot{t} + \frac{2\ddot{t}}{r} - \frac{2\dot{r}\dot{t}}{r^2}$$

$$0^r = r^4\ddot{r} - 2r^3\ddot{r} - r^2\dot{r}^2 + r^2\dot{t}^2 - 4r\dot{t}^2 + 4\dot{t}^2$$

[ ]:
```
flrw.calc_geodesic_from_lagrangian().list(replace={k: 0, theta: 0, phi: 0})
```

FLRW Geodesic From Lagrangian:

$$0^t = \dot{r}^2 a(t)\frac{d}{dt}a(t) + \ddot{t}$$

$$0^r = \left(\ddot{r}a(t) + 2\dot{r}\dot{t}\frac{d}{dt}a(t)\right)a(t)$$

8.4 Geodesics From the Christoffel Symbols

[ ]:
```
minkowski.calc_geodesic_from_christoffel().list()
```

Minkowski Geodesic From Christoffel:

$$0^t = \ddot{t}$$

$$0^x = \ddot{x}$$

$$0^y = \ddot{y}$$

$$0^z = \ddot{z}$$

[ ]:
```
omega = sym.symbols("omega")
c = og.get_curve_parameter()
schwarschild.calc_geodesic_from_christoffel().list(replace={M: 1, theta: 0, phi:
↪ 0})
schwarschild.calc_geodesic_from_christoffel().list(replace={t: c, r: 4*M, phi:␣
↪0, theta: c/(8*M)})
```

Schwarzschild Geodesic From Christoffel:

$$0^t = r^2\ddot{t} - 2r\ddot{t} + 2\dot{r}\dot{t}$$

$$0^r = r^4\ddot{r} - 2r^3\ddot{r} - r^2\dot{r}^2 + r^2\dot{t}^2 - 4r\dot{t}^2 + 4\dot{t}^2$$

Schwarzschild Geodesic From Christoffel:

No Non-Zero Elements

[ ]:
```
flrw.calc_geodesic_from_christoffel().list(replace={k: 0, theta:0, phi: 0})
```

FLRW Geodesic From Christoffel:

$$0^t = -\dot{r}^2 a(t) \frac{d}{dt} a(t) - \ddot{t}$$

$$0^r = -\ddot{r} a(t) - 2\dot{r}\dot{t} \frac{d}{dt} a(t)$$

8.5 Geodesics in Terms of the Time Coordinate

[ ]:
```
minkowski.calc_geodesic_with_time_parameter().list()
```

Minkowski Geodesic With Time Parameter:

$$0^x = \ddot{x}$$

$$0^y = \ddot{y}$$

$$0^z = \ddot{z}$$

[ ]:
```
schwarschild.calc_geodesic_with_time_parameter().list(replace={M: 1, theta: 0,␣
↪phi: 0})
```

Schwarzschild Geodesic With Time Parameter:

$$0^r = r^4 \ddot{r} - 2r^3 \ddot{r} - 3r^2 \dot{r}^2 + r^2 - 4r + 4$$

[ ]:
```
flrw.calc_geodesic_with_time_parameter().list(replace={k: 0, theta: 0, phi: 0})
```

FLRW Geodesic With Time Parameter:

$$0^r = -\ddot{r} a(t) + \dot{r}^3 a^2(t) \frac{d}{dt} a(t) - 2\dot{r} \frac{d}{dt} a(t)$$

9 Additional Information

9.1 Version History

The full version history and change log is available on the GitHub repository, in the CHANGELOG.md file.

9.2 Feature Requests and Bug Reports

The package is under continuous development. If you have any feature requests or bug reports, please feel free to open a new issue on GitHub. If you would like to provide any new functionality, please feel free to submit a pull request.

9.3 Copyright and Citing

No citing information available at the moment.