GraviPy - tutorial

Coordinates and *MetricTensor*

To start working with the gravipy package you must load the package and to initialize a pretty-printing mode in IPython environment

In [1]:
```
from gravipy import * # import SymPy and GraviPy package
init_printing()
```

The next step is to choose coordinates and to define a metric tensor of a particular space. Let's take, for example, the Schwarzschild metric - vacuum solution to the Einstein's field equations which describes the gravitational field of a spherical mass distribution.

In [2]:
```
t, r, theta, phi, M = symbols('t, r, \\theta, \phi, M') # define some symbolic variables
x = Coordinates('\chi', [t, r, theta, phi]) # create a four-vector of coordinates object instantiating the Coordinates class
Metric = diag(-(1-2*M/r), 1/(1-2*M/r), r**2, r**2*sin(theta)**2) # define a matrix of a metric tensor components
g = MetricTensor('g', x, Metric) # create a metric tensor object instantiating the MetricTensor class
```

Each component of any tensor object, can be computed by calling the appropriate instance of the *GeneralTensor* subclass with indices as arguments. The covariant indices take positive integer values (1, 2, ..., dim). The contravariant indices take negative values (-dim, ..., -2, -1).

In [3]:
```
x(-1)
```

Out[3]: $$t$$

In [4]:
```
g(1, 1)
```

Out[4]: $$\frac{2 M}{r} - 1$$

In [5]:
```
x(1)
```

Out[5]: $$t \left(\frac{2 M}{r} - 1\right)$$

Matrix representation of a tensor can be obtained in the following way

In [6]:
```
x(-All)
```

Out[6]: $$\left[\begin{matrix}t & r & \theta & \phi\end{matrix}\right]$$

In [7]:
```
g(All, All)
```

Out[7]: $$\left[\begin{matrix}\frac{2 M}{r} - 1 & 0 & 0 & 0\\0 & \frac{1}{- \frac{2 M}{r} + 1} & 0 & 0\\0 & 0 & r^{2} & 0\\0 & 0 & 0 & r^{2} \sin^{2}{\left (\theta \right)}\end{matrix}\right]$$

In [8]:
```
g(All, 4)
```

Out[8]: $$\left[\begin{matrix}0 & 0 & 0 & r^{2} \sin^{2}{\left (\theta \right)}\end{matrix}\right]$$

Predefined *Tensor* Classes

The GraviPy package contains a number of the *Tensor* subclasses that can be used to calculate a tensor components. The *Tensor* subclasses available in the current version of GraviPy package are

In [9]:
```
print([cls.__name__ for cls in vars()['Tensor'].__subclasses__()])
```

```
['Christoffel', 'Ricci', 'Riemann', 'Einstein', 'Geodesic']
```

The *Christoffel* symbols

The first one is the *Christoffel* class that represents Christoffel symbols of the first and second kind. (Note that the Christoffel symbols are not tensors) Components of the *Christoffel* objects are computed from the below formula

\[\Gamma_{\rho \mu \nu} = g_{\rho \sigma}\Gamma^{\sigma}_{\ \mu \nu} = \frac{1}{2}(g_{\rho \mu, \nu} + g_{\rho \nu, \mu} - g_{\mu \nu, \rho})\]

Let's create an instance of the *Christoffel* class for the Schwarzschild metric g and compute some components of the object

In [10]:
```
Ga = Christoffel('Ga', g)
Ga(1, 2, 1)
```

Out[10]: $$- \frac{M}{r^{2}}$$

Each component of the *Tensor* object is computed only once due to memoization procedure implemented in the *Tensor* class. Computed value of a tensor component is stored in *components* dictionary (attribute of a *Tensor* instance) and returned by the next call to the instance.

In [11]:
```
Ga.components
```

Out[11]: $$\begin{Bmatrix}\begin{pmatrix}1, & 1, & 2\end{pmatrix} : - \frac{M}{r^{2}}, & \begin{pmatrix}1, & 2, & 1\end{pmatrix} : - \frac{M}{r^{2}}\end{Bmatrix}$$

The above dictionary consists of two elements because the symmetry of the Christoffel symbols is implemented in the *Christoffel* class.

If necessary, you can clear the *components* dictionary

In [12]:
```
Ga.components = {}
Ga.components
```

Out[12]: $$\begin{Bmatrix}\end{Bmatrix}$$

The *Matrix* representation of the Christoffel symbols is the following

In [13]:
```
Ga(All, All, All)
```

Out[13]: $$\left[\begin{matrix}\left[\begin{matrix}0 & - \frac{M}{r^{2}} & 0 & 0\\- \frac{M}{r^{2}} & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{matrix}\right] & \left[\begin{matrix}\frac{M}{r^{2}} & 0 & 0 & 0\\0 & - \frac{M}{\left(2 M - r\right)^{2}} & 0 & 0\\0 & 0 & - r & 0\\0 & 0 & 0 & - r \sin^{2}{\left (\theta \right)}\end{matrix}\right] & \left[\begin{matrix}0 & 0 & 0 & 0\\0 & 0 & r & 0\\0 & r & 0 & 0\\0 & 0 & 0 & - \frac{r^{2}}{2} \sin{\left (2 \theta \right)}\end{matrix}\right] & \left[\begin{matrix}0 & 0 & 0 & 0\\0 & 0 & 0 & r \sin^{2}{\left (\theta \right)}\\0 & 0 & 0 & \frac{r^{2}}{2} \sin{\left (2 \theta \right)}\\0 & r \sin^{2}{\left (\theta \right)} & \frac{r^{2}}{2} \sin{\left (2 \theta \right)} & 0\end{matrix}\right]\end{matrix}\right]$$

You can get help on any of classes mentioned before by running the command

In [14]:
```
help(Christoffel)
```

```
Help on class Christoffel in module gravipy.tensorial:

class Christoffel(Tensor)
 |  Represents a class of Christoffel symbols of the first and second kind.
 |
 |  Parameters
 |  ==========
 |

 |  symbol : python string - name of the Christoffel symbol
 |  metric : GraviPy MtricTensor object
 |
 |  Examples
 |  ========
 |
 |  Define a Christoffel symbols for the Schwarzschild metric:
 |
 |  >>> from gravipy import *
 |  >>> t, r, theta, phi = symbols('t, r, \\theta, \phi')
 |  >>> chi = Coordinates('\chi', [t, r, theta, phi])
 |  >>> M = Symbol('M')
 |  >>> Metric = diag(-(1 - 2 * M / r), 1 / (1 - 2 * M / r), r ** 2,
 |  ...                  r ** 2 * sin(theta) ** 2)
 |  >>> g = MetricTensor('g', chi, Metric)
 |  >>> Ga = Christoffel('Ga', g)
 |  >>> Ga(-1, 2, 1)
```

```
 |  -M/(r*(2*M - r))
 |  >>> Ga(2, All, All)
 |  Matrix([
 |  [M/r**2,               0,  0,                0],
 |  [     0, -M/(2*M - r)**2,  0,                0],
 |  [     0,               0, -r,                0],
 |  [     0,               0,  0, -r*sin(\theta)**2]])
 |  >>> Ga(1, -1, 2) # doctest: +IGNORE_EXCEPTION_DETAIL
 |  Traceback (most recent call last):
 |  GraviPyError: "Tensor component Ga(1, -1, 2) doesn't  exist"
 |
 |  Method resolution order:
 |      Christoffel
 |      Tensor
 |      GeneralTensor
 |      __builtin__.object
 |
 |  Methods defined here:
 |
 |  __init__(self, symbol, metric, *args, **kwargs)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from Tensor:
 |
 |  TensorObjects = [<gravipy.tensorial.Christoffel object>]
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from GeneralTensor:
 |
 |  __call__(self, *idxs)
 |

 |  covariantD(self, *idxs)
 |
 |  partialD(self, *idxs)
 |
 |  ----------------------------------------------------------------------
 |  Static methods inherited from GeneralTensor:
 |
 |  get_nmatrixel(M, idxs)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from GeneralTensor:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

```
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from GeneralTensor:
 |
 |  GeneralTensorObjects = [<gravipy.tensorial.Coordinates object>, <gravi.
..
```

Try also "*Christoffel?*" and "*Christoffel??*"

The *Ricci* tensor

$$ R_{\mu \nu} = \frac{\partial \Gamma^{\sigma}_{\ \mu \nu}}{\partial x^{\sigma}} - \frac{\partial \Gamma^{\sigma}_{\ \mu \sigma}}{\partial x^{\nu}} + \Gamma^{\sigma}_{\ \mu \nu}\Gamma^{\rho}_{\ \sigma \rho} - \Gamma^{\rho}_{\ \mu \sigma}\Gamma^{\sigma}_{\ \nu \rho} $$

In [15]:
```
Ri = Ricci('Ri', g)
Ri(All, All)
```

Out[15]: $$\left[\begin{matrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{matrix}\right]$$

Contraction of the *Ricci* tensor $R = R_{\mu}^{\ \mu} = g^{\mu \nu}R_{\mu \nu}$

In [16]:
```
Ri.scalar()
```

Out[16]: $$0$$

The *Riemann* tensor

$$ R_{\mu \nu \rho \sigma} = \frac{\partial \Gamma_{\mu \nu \sigma}}{\partial x^{\rho}} - \frac{\partial \Gamma_{\mu \nu \rho}}{\partial x^{\sigma}} + \Gamma^{\alpha}_{\ \nu \sigma}\Gamma_{\mu \rho \alpha} - \Gamma^{\alpha}_{\ \nu \rho}\Gamma_{\mu \sigma \alpha} - \frac{\partial g_{\mu \alpha}}{\partial x^{\rho}}\Gamma^{\alpha}_{\ \nu \sigma} + \frac{\partial g_{\mu \alpha}}{\partial x^{\sigma}}\Gamma^{\alpha}_{\ \nu \rho} $$

In [17]:
```
Rm = Riemann('Rm', g)
```

Some nonzero components of the *Riemann* tensor are

In [18]:
```
from IPython.core.display import display, Math
for i, j, k, l in list(variations(range(1, 5), 4, True)):
    if Rm(i, j, k, l) != 0 and k<l and i<j:
        display(Math('R_{'+str(i)+str(j)+str(k)+str(l)+'} = '+latex(Rm(i, j
, k, l))))
```

$$R_{1212} = - \frac{2 M}{r^{3}}$$

$$R_{1313} = \frac{M}{r^{2}} \left(- 2 M + r\right)$$

$$R_{1414} = \frac{M}{r^{2}} \left(- 2 M + r\right) \sin^{2}{\left (\theta \right)}$$

$$R_{2323} = \frac{M}{2 M - r}$$

$$R_{2424} = \frac{M \sin^{2}{\left (\theta \right)}}{2 M - r}$$

$$R_{3434} = 2 M r \sin^{2}{\left (\theta \right)}$$

You can also display the matrix representation of the tensor

In [19]:
```
# Rm(All, All, All, All)
```

Contraction of the *Riemann* tensor $R_{\mu \nu} = R^{\rho}_{\ \mu \rho \nu}$

In [20]:
```
ricci = sum([Rm(i, All, k, All)*g(-i, -k) for i, k in list(variations(range
(1, 5), 2, True))], zeros(4))
ricci.simplify()
ricci
```

Out[20]: $$\left[\begin{matrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{matrix}\right]$$

The *Einstein* tensor

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

In [21]:
```
G = Einstein('G', Ri)
G(All, All)
```

Out[21]: $$\left[\begin{matrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{matrix}\right]$$

Geodesics

$$w_{\mu} = \frac{Du_{\mu}}{d\tau} = \frac{d^2x_{\mu}}{d\tau^2} - \frac{1}{2}g_{\rho \sigma, \mu} \frac{dx^{\rho}}{d\tau}\frac{dx^{\sigma}}{d\tau}$$

In [22]:
```
tau = Symbol('\\tau')
w = Geodesic('w', g, tau)
w(All).transpose()
```

Out[22]: $$\left[\begin{matrix}- \frac{2 M \frac{d}{d \tau} r{\left (\tau \right)}}{r^{2}{\left (\tau \right)}} \frac{d}{d \tau} t{\left (\tau \right)} + \left(\frac{2 M}{r{\left (\tau \right)}} - 1\right) \frac{d^{2}}{d \tau^{2}} t{\left (\tau \right)}\\\frac{M \left(\frac{d}{d \tau} t{\left (\tau \right)}\right)^{2}}{r^{2}{\left (\tau \right)}} - \frac{M \left(\frac{d}{d \tau} r{\left (\tau \right)}\right)^{2}}{\left(- \frac{2 M}{r{\left (\tau \right)}} + 1\right)^{2} r^{2}{\left (\tau \right)}} - r{\left (\tau \right)} \sin^{2}{\left (\theta{\left (\tau \right)} \right)} \left(\frac{d}{d \tau} \phi{\left (\tau \right)}\right)^{2} - r{\left (\tau \right)} \left(\frac{d}{d \tau} \theta{\left (\tau \right)}\right)^{2} + \frac{\frac{d^{2}}{d \tau^{2}} r{\left (\tau \right)}}{- \frac{2 M} {r{\left (\tau \right)}} + 1}\\- r^{2}{\left (\tau \right)} \sin{\left (\theta{\left (\tau \right)} \right)} \cos{\left (\theta{\left (\tau \right)} \right)} \left(\frac{d}{d \tau} \phi{\left (\tau \right)}\right)^{2} + r^{2}{\left (\tau \right)} \frac{d^{2}}{d \tau^{2}} \theta{\left (\tau \right)} + 2 r{\left (\tau \right)} \frac{d}{d \tau} \theta{\left (\tau \right)} \frac{d}{d \tau} r{\left (\tau \right)}\\r^{2}{\left (\tau \right)} \sin^{2}{\left (\theta{\left (\tau \right)} \right)} \frac{d^{2}}{d \tau^{2}} \phi{\left (\tau \right)} + 2 r^{2}{\left (\tau \right)} \sin{\left (\theta{\left (\tau \right)} \right)} \cos{\left (\theta{\left (\tau \right)} \right)} \frac{d}{d \tau} \phi{\left (\tau \right)} \frac{d}{d \tau} \theta{\left (\tau \right)} + 2 r{\left (\tau \right)} \sin^{2}{\left (\theta{\left (\tau \right)} \right)} \frac{d}{d \tau} \phi{\left (\tau \right)} \frac{d}{d \tau} r{\left (\tau \right)}\end{matrix}\right]$$

Please note that instantiation of a *Geodesic* class for the metric \(g\) automatically turns on a *Parametrization* mode for the metric \(g\).
Then all coordinates are functions of a world line parameter \(\tau\)

In [23]:
```
Parametrization.info()
```

Out[23]: $$\begin{bmatrix}\begin{bmatrix}\chi, & \tau\end{bmatrix}\end{bmatrix}$$

In [24]:
```
x(-All)
```

Out[24]: $$\left[\begin{matrix}t{\left (\tau \right)} & r{\left (\tau \right)} & \theta{\left (\tau \right)} & \phi{\left (\tau \right)}\end{matrix}\right]$$

In [25]:
```
g(All, All)
```

Out[25]: $$\left[\begin{matrix}\frac{2 M}{r{\left (\tau \right)}} - 1 & 0 & 0 & 0\\0 & \frac{1}{- \frac{2 M}{r{\left (\tau \right)}} + 1} & 0 & 0\\0 & 0 & r^{2}{\left (\tau \right)} & 0\\0 & 0 & 0 & r^{2}{\left (\tau \right)} \sin^{2}{\left (\theta{\left (\tau \right)} \right)}\end{matrix}\right]$$

Parametrization mode can be deactivated by typing

In [26]:
```
Parametrization.deactivate(x)
Parametrization.info()
```

```
No parametrization activated
```

In [27]:
```
x(-All)
```

Out[27]: $$\left[\begin{matrix}t & r & \theta & \phi\end{matrix}\right]$$

In [28]:
```
g(All, All)
```

Out[28]: $$\left[\begin{matrix}\frac{2 M}{r} - 1 & 0 & 0 & 0\\0 & \frac{1}{- \frac{2 M}{r} + 1} & 0 & 0\\0 & 0 & r^{2} & 0\\0 & 0 & 0 & r^{2} \sin^{2}{\left (\theta \right)}\end{matrix}\right]$$

Derivatives

Partial derivative

All instances of a *GeneralTensor* subclasses inherits *partialD* method which works exactly the same way as SymPy *diff* method.

In [29]:
```
T = Tensor('T', 2, g)
T(1, 2)
```

Out[29]: $$\operatorname{T(1, 2)}{\left (t,r,\theta,\phi \right)}$$

In [30]:
```
T.partialD(1, 2, 1, 3) # The first two indices belongs to second rank tensor T
```

Out[30]: $$\frac{\partial^{2}}{\partial \theta\partial t} \operatorname{T(1, 2)}{\left (t,r,\theta,\phi \right)}$$

In [31]:
```
T(1, 2).diff(x(-1), x(-3))
```

Out[31]: $$\frac{\partial^{2}}{\partial \theta\partial t} \operatorname{T(1, 2)}{\left (t,r,\theta,\phi \right)}$$

The only difference is that computed value of *partialD* is saved in "*partial_derivative_components*" dictionary an then returned by the next call to the *partialD* method.

In [32]:
```
T.partial_derivative_components
```

Out[32]: $$\begin{Bmatrix}\begin{pmatrix}1, & 2, & 1, & 3\end{pmatrix} : \frac{\partial^{2}}{\partial \theta\partial t} \operatorname{T(1, 2)}{\left (t,r,\theta,\phi \right)}\end{Bmatrix}$$

Covariant derivative

Covariant derivative components of the tensor ***T*** can be computed by the covariantD method from the formula

$$\nabla_{\sigma} T_{\mu}^{\ \nu} = T_{\mu \ ;\sigma}^{\ \nu} = \frac{\partial T_{\mu}^{\ \nu}}{\partial x^{\sigma}} - \Gamma^{\rho}_{\ \mu \sigma}T_{\rho}^{\ \nu} + \Gamma^{\nu}_{\ \rho \sigma}T_{\mu}^{\ \rho}$$

Let's compute some covariant derivatives of a scalar field C

In [33]:
```
C = Tensor('C', 0, g)
C()
```

Out[33]: $$C{\left (t,r,\theta,\phi \right)}$$

In [34]:
```
C.covariantD(1)
```

Out[34]: $$\frac{\partial}{\partial t} C{\left (t,r,\theta,\phi \right)}$$

In [35]:
```
C.covariantD(2, 3)
```

Out[35]: $$\frac{1}{r} \left(r \frac{\partial^{2}}{\partial \theta\partial r} C{\left (t,r,\theta,\phi \right)} - \frac{\partial}{\partial \theta} C{\left (t,r,\theta,\phi \right)}\right)$$

All *covariantD* components of every *Tensor* object are also memoized

In [36]:
```
C.covariant_derivative_components
```

Out[36]: $$\begin{Bmatrix}\begin{pmatrix}1\end{pmatrix} : \frac{\partial}{\partial t} C{\left (t,r,\theta,\phi \right)}, & \begin{pmatrix}2\end{pmatrix} : \frac{\partial}{\partial r} C{\left (t,r,\theta,\phi \right)}, & \begin{pmatrix}3\end{pmatrix} : \frac{\partial}{\partial \theta} C{\left (t,r,\theta,\phi \right)}, & \begin{pmatrix}4\end{pmatrix} : \frac{\partial}{\partial \phi} C{\left (t,r,\theta,\phi \right)}, & \begin{pmatrix}2, & 3\end{pmatrix} : \frac{1}{r} \left(r \frac{\partial^{2}}{\partial \theta\partial r} C{\left (t,r,\theta,\phi \right)} - \frac{\partial}{\partial \theta} C{\left (t,r,\theta,\phi \right)}\right)\end{Bmatrix}$$

In [37]:
```
C.covariantD(1, 2, 3)
```

Out[37]: $$\frac{1}{r \left(2 M - r\right)} \left(M \frac{\partial^{2}}{\partial \theta\partial t} C{\left (t,r,\theta,\phi \right)} + r \left(2 M - r\right) \frac{\partial^{3}}{\partial \theta\partial r\partial t} C{\left (t,r,\theta,\phi \right)} - \left(2 M - r\right) \frac{\partial^{2}}{\partial \theta\partial t} C{\left (t,r,\theta,\phi \right)}\right)$$

Proof that the covariant derivative of the metric tensor g is zero

In [38]:
```
not any([g.covariantD(i, j, k).simplify() for i, j, k in list(variations(range(1, 5), 3, True))])
```

Out[38]:
```
True
```

Bianchi identity in the Schwarzschild spacetime

$$R_{\mu \nu \sigma \rho ;\gamma} + R_{\mu \nu \gamma \sigma ;\rho} + R_{\mu \nu \rho \gamma ;\sigma} = 0$$

In [39]:
```
not any([(Rm.covariantD(i, j, k, l, m) + Rm.covariantD(i, j, m, k, l) + Rm.covariantD(i, j, l, m, k)).simplify() \
 for i, j, k, l, m in list(variations(range(1, 5), 5, True))])
```

Out[39]:
```
True
```

User-defined tensors

To define a new scalar/vector/tensor field in some space you should **extend** the *Tensor* class or **create an instance** of the *Tensor* class.

Tensor class instantiation

Let's create a third-rank tensor field living in the Schwarzshild spacetime as an instance of the *Tensor* class

In [40]:
```
S = Tensor('S', 3, g)
```

Until you define (override) the *_compute_covariant_component* method of the **S** object, all of \(4^3\) components are arbitrary functions of coordinates

In [41]:
```
S(1, 2, 3)
```

Out[41]: $$\operatorname{S(1, 2, 3)}{\left (t,r,\theta,\phi \right)}$$

In [42]:
```
source(T._compute_covariant_component)
```

```
In file: /usr/local/lib/python2.7/dist-packages/gravipy/tensorial.py
    def _compute_covariant_component(self, idxs):
        if len(idxs) == 0:
            component = Function(str(self.symbol))(*self.coords.c)
        elif len(idxs) == 1:
            component = Function(str(self.symbol) +
                                 '(' + str(idxs[0]) + ')')(*self.coords.c)
        else:
            component = Function(str(self.symbol) + str(idxs))(*self.coords
.c)
        return component
```

Let's assume that tensor **S** is the commutator of the covariant derivatives of some arbitrary vector field **V** and create a new *_compute_covariant_component* method for the object **S**

In [43]:
```
V = Tensor('V', 1, g)
V(All)
```

Out[43]: $$\left[\begin{matrix}\operatorname{V(1)}{\left (t,r,\theta,\phi \right)} & \operatorname{V(2)}{\left (t,r,\theta,\phi \right)} & \operatorname{V(3)}{\left (t,r,\theta,\phi \right)} & \operatorname{V(4)}{\left (t,r,\theta,\phi \right)}\end{matrix}\right]$$

In [44]:
```
def S_new_method(idxs):
    component = (V.covariantD(idxs[0], idxs[1], idxs[2]) - V.covariantD(idx
s[0], idxs[2], idxs[1])).simplify() # definition
    S.components.update({idxs: component}) # memoization
    return component
S._compute_covariant_component = S_new_method # _compute_covariant_componen
t method was overriden
```

In [45]:
```
S(1, 1, 3)
```

Out[45]: $$\frac{M}{r^{4}} \left(2 M - r\right) \operatorname{V(3)}{\left (t,r,\theta,\phi \right)}$$

One can check that the well known formula is correct

\[V_{\mu ;\nu \rho} - V_{\mu ;\rho \nu} = R^{\sigma}_{\ \mu \nu \rho}V_{\sigma} \]

In [46]:
```
zeros = reduce(Matrix.add, [Rm(-i, All, All, All)*V(i) for i  in range(1, 5
)]) - S(All, All, All)
zeros.simplify()
zeros
```

Out[46]: $$\left[\begin{matrix}\left[\begin{matrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{matrix}\right] & \left[\begin{matrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{matrix}\right] & \left[\begin{matrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{matrix}\right] & \left[\begin{matrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{matrix}\right]\end{matrix}\right]$$

Another way of tensor creation is to make an instance of the *Tensor* class with components option. Tensor components stored in *Matrix* object are writen to the *components* dictionary of the instance by this method.

In [47]:
```
Z = Tensor('Z', 3, g, components=zeros, components_type=(1, 1, 1))
```

In [48]:
```
not any(Z.components.values())
```

Out[48]:
```
True
```

Tensor class extension

As an example of the *Tensor* class extension you can get the source code of any of the predefined *Tensor* subclasses

In [49]:
```
print([cls.__name__ for cls in vars()['Tensor'].__subclasses__()])
```

```
['Christoffel', 'Ricci', 'Riemann', 'Einstein', 'Geodesic']
```

In [50]:
```
source(Christoffel)
```

```
In file: /usr/local/lib/python2.7/dist-packages/gravipy/tensorial.py
class Christoffel(Tensor):
    r"""
    Represents a class of Christoffel symbols of the first and second kind.

    Parameters
    ==========

    symbol : python string - name of the Christoffel symbol
    metric : GraviPy MtricTensor object

    Examples
```

```
    ========

    Define a Christoffel symbols for the Schwarzschild metric:

    >>> from gravipy import *
    >>> t, r, theta, phi = symbols('t, r, \\theta, \phi')
    >>> chi = Coordinates('\chi', [t, r, theta, phi])
    >>> M = Symbol('M')
    >>> Metric = diag(-(1 - 2 * M / r), 1 / (1 - 2 * M / r), r ** 2,
    ...                      r ** 2 * sin(theta) ** 2)
    >>> g = MetricTensor('g', chi, Metric)
    >>> Ga = Christoffel('Ga', g)
    >>> Ga(-1, 2, 1)
    -M/(r*(2*M - r))
    >>> Ga(2, All, All)
    Matrix([
    [M/r**2,                0,  0,                 0],
    [     0, -M/(2*M - r)**2,  0,                 0],
    [     0,                0, -r,                 0],
    [     0,                0,  0, -r*sin(\theta)**2]])
    >>> Ga(1, -1, 2) # doctest: +IGNORE_EXCEPTION_DETAIL
    Traceback (most recent call last):
    GraviPyError: "Tensor component Ga(1, -1, 2) doesn't  exist"
    """

    def __init__(self, symbol, metric, *args, **kwargs):
        super(Christoffel, self).__init__(
            symbol, 3, metric, index_types=(0, 1, 1), *args, **kwargs)
        self.is_connection = True
        self.conn = self
        self.metric.conn = self

    def _compute_covariant_component(self, idxs):
        component = Rational(1, 2) * (
            self.metric(idxs[0], idxs[1]).diff(self.coords(-idxs[2])) +
            self.metric(idxs[0], idxs[2]).diff(self.coords(-idxs[1])) -
            self.metric(idxs[1], idxs[2]).diff(self.coords(-idxs[0]))) \
            .together().simplify()
        self.components.update({idxs: component})
        if self.apply_tensor_symmetry:
            self.components.update({(idxs[0], idxs[2], idxs[1]): component}
)
        return component
```