GR_curvature_cal

April 9, 2021

[1]:
```python
from IPython.display import Latex, display
from sympy import *
from numpy import zeros


init_printing()


def inverse_metric_calc(xmetric):
    """
    Taking the inverse of a metric

    Input:  xmetric: given metric, g_ij
    Output: inverse_metric: the inverse of the metric, g^ij
    """
    inverse_metric = MutableSparseNDimArray(zeros((ndim,)*2))
    for i in range(0, ndim):
            inverse_metric[i, i] = 1 / xmetric[i, i]
    return inverse_metric


def derivative_of_metric(xmetric, i, j, k):
    """
    Taking the derivative of a given metric: \partial_i(g_jk)
    where g_jk is the metric and the \partial_i is the partial derivative with␣
 ↪respect
    to i'th component

    Input:  xmetric: the given metric
            i,j,k: indices that runs from 0-ndim
    Output: The partial derivative of a given metric's [j][k] component with␣
 ↪respect to
    the [i]'th  component
    """
    expr = xmetric[j, k]
    return diff(expr, coord_sys[i])
```

```python
def derivative_of_chris(xchris_symb, i, j, k, l):
    """
    Taking the derivative of a given christoffel symbol; \partial_i␣
 ↪(\Gamma^j_kl)
    where \Gamma^j_kl is the christoffel symbol and the \partial_i is the␣
 ↪partial derivative
    with respect to i'th component

    Input:  xchris_symb: given Christoffel Symbol i,j,k,l: indices that runs␣
 ↪from 0-ndim
    Output: The partial derivative of a given Christoffel Symbols's ^[j]_[k][l]
    component with respect to the [i]'th component
    """
    expr = xchris_symb[j, k, l]
    return diff(expr, coord_sys[i])


def christoffel_sym_calctr(xmetric):
    """
    Input:  The Metric Tensor
    Output: Christoffel Symbols - \Gamma^m_ij
    """
    # creating an empty tensor to fill
    Chris_sym = MutableSparseNDimArray(zeros((ndim,)*3))
    for m in range(0, ndim):
        for i in range(0, ndim):
            for j in range(0, ndim):
                einstein_sum = 0
                for k in range (0, ndim):
                    I1 = derivative_of_metric(xmetric, j, k, i)
                    I2 = derivative_of_metric(xmetric, i, k, j)
                    I3 = derivative_of_metric(xmetric, k, i, j)
                    S = I1 + I2 - I3
                    einstein_sum += 1/2 * g_inverse[m, k] * S
                Chris_sym[m, i, j] = einstein_sum
    return Chris_sym


def riemann_tensor_calctr(xchris_symbol):
    """
    Input:  Christoffel Symbols - \Gamma^m_ij
    Output: The Riemann Curvature Tensor - R^(l)_(ijk)
    """
    # creating an empty tensor to fill
    riemann_curv_tensor = MutableSparseNDimArray(zeros((ndim,)*4))
    for i in range(0, ndim):
```

```python
        for j in range(0, ndim):
            for k in range(0, ndim):
                for l in range(0, ndim):
                    Q1 = derivative_of_chris(xchris_symbol, j, l, i, k)
                    Q2 = derivative_of_chris(xchris_symbol, i, l, j, k)
                    einstein_sum = 0
                    for p in range(0, ndim):
                        I1 = xchris_symbol[p, i, k] * xchris_symbol[l, j, p]
                        I2 = xchris_symbol[p, j, k] * xchris_symbol[l, i, p]
                        einstein_sum += (I1 - I2)
                    riemann_curv_tensor[l, i, j, k] = Q1 - Q2 + einstein_sum
    return riemann_curv_tensor


def ricci_tensor_calctr(xriemann_tensor):
    """
    Input:  The Riemann Curvature Tensor - R^(l)_(ijk)
    Output: The Ricci Tensor - R_(ik)
    """
    # creating an empty tensor to fill
    ricci_tensor = MutableSparseNDimArray(zeros((ndim,)*2))
    einstiem_sum = 0
    for i in range(0, ndim):
        for k in range(0, ndim):
            einstein_sum = 0
            for j in range(0, ndim):
                einstein_sum += xriemann_tensor[j, i, j, k]
            ricci_tensor[i, k] = einstein_sum
    return ricci_tensor


def ricci_scalar_calctr(xmetric, xricci_tensor):
    """
    Input:  The Ricci Tensor - R_(ik)
    Output: The Ricci Scalar - R
    """
    R = 0
    for i in range(0, ndim):
        for k in range(0, ndim):
            R += g_inverse[i, k] * xricci_tensor[i, k]
    return R
```

[2]:
```python
# Defining the coordinate system of the metric that we will work on
coord_sys = symbols("t r theta phi")

# Defining some extra symbols
a = symbols("a")
```

```
# Defining the dimension of the space
ndim = 4

# Defining the components of the metric
metric = MutableSparseNDimArray(zeros((ndim,)*2))
metric[0, 0] = -1
metric[1, 1] = 1
metric[2, 2] = coord_sys[1]**2
metric[3, 3] = coord_sys[1]**2 * sin(coord_sys[2])**2
```

[3]:

```
# The metric tensor
g = metric

# Inverse of the metric tensor
g_inverse = inverse_metric_calc(metric)

# Christoffel symbols
Christoffel_symbol = christoffel_sym_calctr(g)

# Riemann Curvature Tensor
Riemann_curv_tensor = riemann_tensor_calctr(Christoffel_symbol)

# Ricci Curvature Tensor
Ricci_curv = ricci_tensor_calctr(Riemann_curv_tensor)

# Ricci Scalar
R = ricci_scalar_calctr(g, Ricci_curv)
```

[4]:

```
# The Metric Tensor
g.tolist()
g
```

[4]:

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & r^2 & 0 \\ 0 & 0 & 0 & r^2 \sin^2(\theta) \end{bmatrix}$$

[5]:

```
# Christoffel Symbols
Christoffel_symbol.tolist()
Christoffel_symbol
```

[5]:

$$\left[\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1.0r & 0 \\ 0 & 0 & 0 & -1.0r \sin^2(\theta) \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1.0}{r} & 0 \\ 0 & \frac{1.0}{r} & 0 & 0 \\ 0 & 0 & 0 & -1.0 \sin(\theta)\cos(\theta) \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1.0}{r} \\ 0 & 0 & 0 & \frac{1.0\cos(\theta)}{\sin(\theta)} \\ 0 & \frac{1.0}{r} & \frac{1.0\cos(\theta)}{\sin(\theta)} & 0 \end{bmatrix} \right.$$

[6]:
```
# Non-zero Components of the Christoffel Symbols
for i in range(0, ndim):
    for j in range(0, ndim):
        for k in range(0, ndim):
            if Christoffel_symbol[i, j, k] != 0:
                display(Latex('$\\Gamma^{{{0}}}{{}}_{{{1} {2}}} = {3}$'.
 ↪format(latex(coord_sys[i]), latex(coord_sys[j]), latex(coord_sys[k]),␣
 ↪latex(Christoffel_symbol[i,j,k]))))
```

$\Gamma^{r}{}_{\theta \theta} = -1.0r$

$\Gamma^{r}{}_{\phi \phi} = -1.0r \sin^{2}\left(\theta\right)$

$\Gamma^{\theta}{}_{r \theta} = \frac{1.0}{r}$

$\Gamma^{\theta}{}_{\theta r} = \frac{1.0}{r}$

$\Gamma^{\theta}{}_{\phi \phi} = -1.0 \sin\left(\theta\right) \cos\left(\theta\right)$

$\Gamma^{\phi}{}_{r \phi} = \frac{1.0}{r}$

$\Gamma^{\phi}{}_{\theta \phi} = \frac{1.0 \cos\left(\theta\right)}{\sin\left(\theta\right)}$

$\Gamma^{\phi}{}_{\phi r} = \frac{1.0}{r}$

$\Gamma^{\phi}{}_{\phi \theta} = \frac{1.0 \cos\left(\theta\right)}{\sin\left(\theta\right)}$

[7]:
```
# Riemann Curvature Tensor
Riemann_curv_tensor.tolist()
Riemann_curv_tensor
```

[7]:

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

[8]:
```
# Ricci Curvature Tensor
Ricci_curv.tolist()
Ricci_curv
```

[8]:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

[9]:
```
# Ricci Scalar
R
```

[9]: 0