Basic Computations in Differential Geometry with SageManifolds

A. Megías and J. J. Ruiz-Lorenzo

Departamento de Física, Universidad de Extremadura, E-06006 Badajoz, Spain

March 13, 2021

1 Differential Geometry Computations

This notebook is based on the original notebook: Schwarzschild spacetime.

In this notebook we will use the metric of the two-dimensional euclidean space in polar coordinates. The corresponding tools have been developed within the SageManifolds project.

For a given metric $g_{\mu\nu}$ we can compute:

- The inverse metric: $g^{\mu\nu}$.
- Christoffel Symbols: $\Gamma^{\lambda}{}_{\mu\nu} = \frac{1}{2} g^{\lambda\sigma} \left(\partial_\mu g_{\sigma\nu} + \partial_\nu g_{\sigma\mu} - \partial_\sigma g_{\mu\nu}\right)$.
- Riemann tensor: $R^{\lambda}{}_{\mu\nu\sigma} = \partial_\nu \Gamma^{\lambda}{}_{\mu\sigma} - \partial_\sigma \Gamma^{\lambda}{}_{\mu\nu} + \Gamma^{\eta}_{\mu\sigma}\Gamma^{\lambda}_{\eta\nu} - \Gamma^{\eta}_{\mu\nu}\Gamma^{\lambda}_{\eta\sigma}$.
- Ricci tensor: $R_{\mu\nu} = R^{\lambda}{}_{\mu\lambda\nu}$.
- Scalar curvature: $R = g^{\mu\nu} R_{\mu\nu}$.
- Einstein tensor: $G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R$.

Although the notebook is created for d dimensions, it can be generalized to whatever d dimensional non-Lorentzian metric.

To run it, you must start SageMath with the Jupyter interface, via the command `sage -n jupyter`

[1]:
```
version() #SageMath version
%display latex #To display LaTeX expressions in some outputs
```

1.1 Differentiable manifold

We define our differentiable manifold. The method `Manifold()` must receive the following arguments: d dimensions of the manifold, and name of the manifold.

[2]:
```
d=2 #space dimensions
M = Manifold(d, 'M')
print(M)
```

```
2-dimensional differentiable manifold M
```

1.2 List of coordinates

We must define our coordinates via the method `chart()` applied to the object `M` (our manifold). Note that the argument of `chart()` is a raw string (hence the prefix `r` in front of it), which defines the range of each coordinate, if different from $(-\infty, +\infty)$, as well as its LaTeX symbol, if different from the Python symbol to denote the coordinate. The Python variables for each coordinate are declared within the `<...>` operator on the left-hand side of the identity, `X` denoting the Python variable chosen for the coordinate chart.

As an example, the standard **polar coordinates** are introduced. The coordinates are the following:

$$r \in (0, +\infty), \quad \theta \in (0, 2\pi).$$

[3]:
```
X.<r,th> = M.chart(r"r:(0,+oo) th:(0,2*pi):\theta")
X
```

[3]: $(M, (r, \theta))$

[4]:
```
X[:]
```

[4]: (r, θ)

The coordinates follows the same indexing: $X^0 = r$, $X^1 = \theta$.

[5]:
```
X[0],X[1]
```

[5]: (r, θ)

1.3 Metric tensor $g_{\mu\nu}$.

If we want to introduce a constant parameter m as a symbolic positive variable, it must be done via the function `var()`:

[6]:
```
#m = var('m') #To uncomment delete #
#assume(m>=0) #To uncomment delete #
```

The metric tensor of the manifold `M` is returned by the method `metric()`; we initialize its components in the chart `X`, which is the default (unique) chart on `M`:

[7]:
```
g = M.metric('g')
g[0,0] = 1
g[1,1] = r^2
g.display()
```

[7]: $g = \mathrm{d}r \otimes \mathrm{d}r + r^2 \mathrm{d}\theta \otimes \mathrm{d}\theta$

To display the metric as a matrix:

[8]:
```
g[:]
```

[8]:

$$\begin{pmatrix} 1 & 0 \\ 0 & r^2 \end{pmatrix}$$

In order to access to a the component of the metric with components (μ, ν) we would write: `g[mu,nu]`. Where `mu` and `nu` are integer variables such that $\mu, \nu \in \{0, \ldots, d\}$, where $\{r, \theta\} \equiv \{0, 1\}$ for our case. Here, we display the component g_{rr}.

[9]:
```
g[0,0]
```

[9]:

1

The inverse metric can be computed via `g.inverse()`.

[10]:
```
ginv=g.inverse(); ginv
```

[10]:

g^{-1}

[11]:
```
ginv.display()
```

[11]:

$$g^{-1} = \frac{\partial}{\partial r} \otimes \frac{\partial}{\partial r} + \frac{1}{r^2} \frac{\partial}{\partial \theta} \otimes \frac{\partial}{\partial \theta}$$

[12]:
```
ginv[:]
```

[12]:

$$\begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{r^2} \end{pmatrix}$$

If we multiply both matrices, we should get the $d \times d$ identity matrix

[13]:
```
delta = g['_{ab}']*ginv['^{bc}']
```

[14]:
```
delta[:]
```

[14]:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

1.4 Christoffel symbols $\Gamma^{\lambda}{}_{\mu\nu}$.

The Christoffel symbols of g with respect to the given coordinates are printed by the method `christoffel_symbols_display()` applied to the metric object `g`. By default, only the nonzero symbols and the nonredundant ones (taking into account the symmetry of the last two indices) are displayed. Type `g.christoffel_symbols_display?` to see all possible options.

[15]:
```
g.christoffel_symbols_display()
```

[15]:

$$\begin{array}{lcl} \Gamma^{r}{}_{\theta\theta} & = & -r \\ \Gamma^{\theta}{}_{r\theta} & = & \frac{1}{r} \end{array}$$

Accessing to a Christoffel symbol specified by its indices (e.g. $\Gamma^{r}{}_{\theta\theta}$):

[16]:
```
g.christoffel_symbols()[0,1,1]
```

[16]:

$-r$

Checking the symmetry on the last two indices:

[17]:
```
g.christoffel_symbols()[0,0,1] == g.christoffel_symbols()[0,1,0]
```

[17]: True

1.5 Riemann curvature tensor

The Riemann curvature tensor is obtained by the method `riemann()`:

[18]:
```
Riem = g.riemann()
print(Riem)
```

```
Tensor field Riem(g) of type (1,3) on the 2-dimensional differentiable manifold
M
```

Displaying its nonredundant components:

[19]:
```
Riem.display_comp(only_nonredundant=True) #If there is no elements displayed␣
↪means that all of them are identically zero.
```

[19]:

We can **lower and raise all the indices** of the components $R^{\lambda}{}_{\mu\nu\sigma}$ of the Riemann tensor, via the metric g by the methods `down()` and `up()`.

[20]:
```
Riemdown = Riem.down(g);
Riemup = Riem.up(g);
```

1.6 Ricci tensor

We know that the Ricci tensor is computed via the Riemann curvature tensor: $R_{\mu\nu} = R^{\lambda}{}_{\mu\lambda\nu}$. However, SageMath can give us directly the Ricci tensor from the metric g with the method `g.ricci()`.

[21]:
```
Ric = g.ricci()
```

[22]:
```
Ric.display()
```

[22]: $\mathrm{Ric}\left(g\right) = 0$

[23]:
```
Ric[:]
```

[23]: $\left(\begin{array}{rr} 0 & 0 \\ 0 & 0 \end{array}\right)$

Let us check that the definition of the Ricci tensor via the contraction of the Riemann tensor and the one given by the SageMath method `g.ricci()` coincides.

[24]:
```
Ric == Riem.down(g)['_{abcd}']*ginv['^{ac}']
```

[24]: True

1.7 Calculating the Scalar Curvature

It is computed by the contraction of theinverse metric and the Ricci tensor, i.e., $R = g^{\mu\nu} R_{\mu\nu}$.

[25]:
```
ScalarCurvature=ginv['^{ab}']*Ric['_ab']; ScalarCurvature
```

[25]: 0

1.8 Kretschmann scalar

The Kretschmann scalar is the "square" of the Riemann tensor defined by

$$K = R_{\lambda\mu\nu\sigma} R^{\lambda\mu\nu\sigma}$$

To compute it, we must first form the tensor fields whose components are $R_{\lambda\mu\nu\sigma}$ and $R^{\lambda\mu\nu\sigma}$. They are obtained by respectively lowering and raising the indices of the components $R^{\lambda}{}_{\mu\nu\sigma}$ of the Riemann tensor, via the metric g. These two operations are performed by the methods `down()` and `up()`. The contraction is performed by summation on repeated indices:

[26]:
```
K = Riem.down(g)['_{abcd}'] * Riem.up(g)['^{abcd}']
K
```

[26]: 0

[27]:
```
K.display()
```

[27]:
$$\begin{array}{llcl} 0: & M & \longrightarrow & \mathbb{R} \\ & (r,\theta) & \longmapsto & 0 \end{array}$$

[28]:
```
K.expr()
```

[28]: 0

1.9 Levi-Civita Connection

The Levi-Civita Connection ∇ associated with the metric g.

[31]:
```
nab = g.connection() ; print(nab)
```

```
Levi-Civita connection nabla_g associated with the Riemannian metric g on the
2-dimensional differentiable manifold M
```

We check the compatibility of the connection with the metric (that is, $\nabla_g g = 0$).

[29]:
```
nab(g).display()
```

[29]: $\nabla_g g = 0$

[30]:
```
w = M.vector_field('w')
```

Compute the covariant derivative of the vector $w = (r, r\sin\theta)$, $\nabla_\nu w^\nu$.

[31]:
```
w[:] = [r,r*sin(th)]
```

[32]:
```
DW = (nab(w)['^a_b']*delta['_a^b'])
DW.expr()
```

[32]: $r\cos(\theta) + 2$

Check that $\nabla_\nu w^\nu = \partial_\nu w^\nu + w^\gamma \Gamma^\nu{}_{\gamma\nu}$.

[33]:
```
sum([w[i].diff(i)+w[i]*sum([g.christoffel_symbols()[j,i,j] for j in M.irange()])␣
 ↪for i in M.irange()])
```

[33]: $r\cos(\theta) + 2$