

[Understanding Riemannian Geometry] 6. Counting and Calculating Curvature (Python)

Su Jianlin

October 19, 2016

Independent Components of Curvature

The Riemann curvature tensor is a very important tensor; the space is flat if and only if all of its components are zero. It also appears in Einstein's field equations. In short, whenever Riemannian geometry is involved, the Riemann curvature tensor is inevitably the core content.

As we have seen, the Riemann curvature tensor has 4 indices, which means it has n^4 components, where n is the dimension of the space. Thus, in 2, 3, and 4-dimensional spaces, it has 16, 81, and 256 components, respectively. Clearly, calculating it is a quite painful task. Fortunately, this tensor possesses many symmetry properties that greatly reduce the number of independent components. Let us analyze this.

First, we derive some symmetry properties of the Riemann curvature tensor, which are consistent with classical textbooks. Define:

$$R_{\mu\alpha\beta\gamma} = g_{\mu\nu} R^{\nu}_{\alpha\beta\gamma} \quad (50)$$

The reason for defining this quantity involves the distinction between contravariant and covariant tensors. Here, we are primarily concerned with the geometric perspective, so we skip a detailed analysis of tensors. This quantity is called the fully covariant Riemann curvature tensor, or sometimes simply the Riemann curvature tensor; as long as it does not cause confusion, generally no distinction is made. Through somewhat tedious algebraic operations (found in general textbooks on differential geometry, Riemannian geometry, or general relativity), we can obtain:

$$\begin{aligned} R_{\mu\alpha\beta\gamma} &= -R_{\mu\alpha\gamma\beta} \\ R_{\mu\alpha\beta\gamma} &= -R_{\alpha\mu\beta\gamma} \\ R_{\mu\alpha\beta\gamma} &= R_{\beta\gamma\mu\alpha} \\ R_{\mu\alpha\beta\gamma} + R_{\mu\beta\gamma\alpha} + R_{\mu\gamma\alpha\beta} &= 0 \end{aligned} \quad (51)$$

The first two equations immediately tell us that if $\beta = \gamma$ or $\mu = \alpha$, then $R_{\mu\alpha\beta\gamma} = 0$, which is a result of antisymmetry. The following counting shows that the number of independent components can be further reduced. We can view each equation as a constraint; for every constraint, the number of independent components decreases by 1. We need to consider how many independent constraints there are to determine the number of independent components.

First, $R_{\mu\alpha\beta\gamma}$ can be viewed as an $n \times n$ matrix (the first two indices), where each element of this matrix is itself an $n \times n$ matrix (the last two indices). From $R_{\mu\alpha\beta\gamma} = -R_{\mu\alpha\gamma\beta}$, we know that the matrix acting as each element is an antisymmetric matrix, so it has only $n(n-1)/2$ independent components. From $R_{\mu\alpha\beta\gamma} = -R_{\alpha\mu\beta\gamma}$, we know that the "matrix of matrices" is also antisymmetric, so it also has only $n(n-1)/2$ independent components. That is to say, based on the first two equations, the total number of independent components does not exceed $[n(n-1)/2]^2$.

Next, the third and fourth constraints further reduce the number of components. $R_{\mu\alpha\beta\gamma} = R_{\beta\gamma\mu\alpha}$ indicates that if we arrange the remaining $[n(n-1)/2]^2$ components into an $n(n-1)/2 \times n(n-1)/2$ matrix, it is symmetric, so the number of independent components is further reduced to $[n(n-1)/2]^2 + [n(n-1)/2]$.

$1)/2 \times n(n-1)/2$ matrix, that matrix is symmetric. In this case, the number of independent components is:

$$\frac{1}{2} \frac{n(n-1)}{2} \left[\frac{n(n-1)}{2} + 1 \right] \quad (52)$$

The last constraint $R_{\mu\alpha\beta\gamma} + R_{\mu\beta\gamma\alpha} + R_{\mu\gamma\alpha\beta} = 0$ (the algebraic Bianchi identity) applies to four distinct indices, because if there is a pair of identical indices, it can be transformed into a linear combination of the first three constraints. Therefore, the number of independent components needs to be further reduced by $\binom{n}{4}$:

$$\frac{1}{2} \frac{n(n-1)}{2} \left[\frac{n(n-1)}{2} + 1 \right] - \binom{n}{4} = \frac{n^2(n^2-1)}{12} \quad (53)$$

Thus, in 2, 3, 4, and 5-dimensional spaces, the number of independent components of the Riemann curvature tensor is 1, 6, 20, and 50, respectively. In particular, in a two-dimensional space (i.e., when studying a 2D surface in 3D space), the curvature tensor has only one independent component.

Calculation of the Curvature Tensor

Note: The following code contains an error due to a bug in SymPy at the time of writing. It has been submitted to the official developers; please wait for a fix.

Regrettably, we must omit the manual calculation of this curvature tensor here, as there is no obviously simple method for manual computation. While variational methods can simplify the calculation of $\Gamma_{\alpha\beta}^{\mu}$ and thus indirectly simplify the curvature tensor, a top-down simplified calculation method requires knowledge of exterior calculus, which we cannot cover in this series yet.

However, in terms of computing, using the symbolic computation library SymPy in Python can help us quickly calculate connection coefficients and the curvature tensor. SymPy is a symbolic computation library for Python that is compact, open-source, and powerful. Of course, it cannot compare to commercial software like Mathematica in terms of breadth, but in certain specific areas, it is actually superior—it "wins not in being all-encompassing, but in being specialized." SymPy has built-in modules specifically for handling differential geometry and tensors, making it very convenient to calculate $\Gamma_{\alpha\beta}^{\mu}$ and $R_{\alpha\beta\gamma}^{\mu}$.

A simple example of the code is as follows:

```

1 from sympy.diffgeom import Manifold, Patch, CoordSystem
2 from sympy.diffgeom import TensorProduct as TP
3 from sympy.diffgeom import metric_to_Riemann_components as Riemann
4 from sympy.diffgeom import metric_to_Christoffel_2nd as Christoffel
5 from sympy import Symbol, Function, latex, sin
6
7 n = 2
8 M = Manifold('M', n)
9 P = Patch('P', M)
10
11 coord = CoordSystem('coord', P, ['x%s'%i for i in range(n)])
12 x = coord.coord_functions()
13 dx = coord.base_oneforms()
14 g = [[1, 0], [0, sin(x[0])**2]]
15 metric = sum([g[i][j]*TP(dx[i], dx[j]) for i in range(n) for j in range
16                 (n)])
17
18 C = Christoffel(metric)
19 R = Riemann(metric)

```

This uses 2D spherical coordinates as an example, where n defines the dimension and g defines the metric matrix. If the reader wants to modify it, they only need to change these two parameters. The x variable is the set of coordinates, and the dx variable is the set of differential elements. For consistency, the indices here start from 0, traversing $0 \sim n - 1$. The results obtained are:

$$\begin{aligned} &[[[0, 0], [0, -\sin(x_0)\cos(x_0)], [0, \cos(x_0)/\sin(x_0)], [\cos(x_0)/\sin(x_0), 0]]] \\ &[[[0, 0], [0, 0]], [[0, \sin(x_0)**2], [-\sin(x_0)**2, 0]], [[[0, -1], [1, 0]], [[0, 0], [0, 0]]]] \end{aligned}$$

As we can see, the program lists all components. If we only care about non-zero components, we can use the following code for output:

```

1 for i1 in range(n):
2     for i2 in range(n):
3         for i3 in range(n):
4             if C[i1, i2, i3]:
5                 print 'Gamma^%s_%s%s:'%(i1, i2, i3), C[i1, i2, i3]
6
7 for i1 in range(n):
8     for i2 in range(n):
9         for i3 in range(n):
10            for i4 in range(n):
11                if R[i1, i2, i3, i4]:
12                    print 'Riemann^%s_%s%s%s:'%(i1, i2, i3, i4), R[i1,
                    i2, i3, i4]
```

Resulting in:

```

Gamma^0_11: -sin(x0)*cos(x0)
Gamma^1_01: cos(x0)/sin(x0)
Gamma^1_10: cos(x0)/sin(x0)
Riemann^0_101: sin(x0)**2
Riemann^0_110: -sin(x0)**2
Riemann^1_001: -1
Riemann^1_010: 1
```

Note that the curvature tensor calculated here is $R_{\alpha\beta\gamma}^{\mu}$. If the reader is interested in $R_{\mu\alpha\beta\gamma}$, and since SymPy does not have a ready-made function for it, we can calculate it ourselves using the tensor operation library:

```

1 from sympy.tensor.array import tensorproduct, tensorcontraction
2 tensorcontraction(tensorproduct(g, R), (1, 2))
```

Where `tensorproduct` is used for tensor products and `tensorcontraction` is used for contraction. This yields:

$$\begin{aligned} &[[[0, 0], [0, 0]], [[0, \sin(x_0)**2], [-\sin(x_0)**2, 0]], [[[0, -\sin(x_0)**2], [\sin(x_0)**2, 0]], \\ &[[0, 0], [0, 0]]]] \end{aligned}$$

Finally, if you want to output LaTeX code, just use the `latex` command. For example:

```

1 for i1 in range(n):
2     for i2 in range(n):
3         for i3 in range(n):
4             if C[i1, i2, i3]:
5                 print 'Gamma^%s_%s%s:'%(i1, i2, i3), \
6                     latex(C[i1, i2, i3]).replace('\mathrm{x}_', '{x_'}
                     ).replace('\boldsymbol{x}_', '{\mathbf{x}_'})
```

The final `replace` calls perform some simple fine-tuning to make the results more consistent with our usual notation. The result is:

$$\begin{aligned}\Gamma_{0,11} &: -\sin\{\left(x_0\right)\}\cos\{\left(x_0\right)\} \\ \Gamma_{1,01} &: \frac{\cos\{\left(x_0\right)\}\sin\{\left(x_0\right)\}}{\sin\{\left(x_0\right)\}} \\ \Gamma_{1,10} &: \frac{\cos\{\left(x_0\right)\}}{\sin\{\left(x_0\right)\}}\end{aligned}$$

Lastly, we might need to find the curvature for a metric containing undetermined functions, such as $ds^2 = f(x, y)(dx^2 + dy^2)$. The code would look something like this:

```

1 f = Function('f')
2 g = [[f(x[0], x[1]), 0], [0, f(x[0], x[1])]]
3 metric = sum([g[i][j]*TP(dx[i], dx[j]) for i in range(n) for j in range
4             (n)])
5 C = Christoffel(metric)
6 R = Riemann(metric)

```

In this case, terms like the following will appear in $\Gamma_{\alpha\beta}^{\mu}$:

$$\text{Subs}(\text{Derivative}(f(_xi_1, x1), _xi_1), (_xi_1), (x0,)) / (2 * f(x0, x1))$$

Converting this to LaTeX (with the same simple processing as before to replace `\boldsymbol` and `\mathrm` tags) gives:

$$\left. \frac{\left(\frac{d}{d\xi_1} f(\xi_1, x_1) \right) \right|_{\substack{\xi_1 = x_0 \\ x_1 = x_0}} = 2 f(x_0, x_1)$$

The output effect is:

$$\frac{\frac{d}{d\xi_1} f(\xi_1, x_1) \Big|_{\xi_1=x_0}}{2f(x_0, x_1)}$$

Which is actually:

$$\frac{\frac{\partial}{\partial x_0} f(x_0, x_1)}{2f(x_0, x_1)}$$

Evidently, SymPy’s output still has room for improvement; many parts still require manual optimization. Even worse, terms like this appear in $R_{\alpha\beta\gamma}^{\mu}$:

$$\begin{aligned} & -\text{Subs}(\text{Derivative}(f(_\#_1(_\text{Dummy_38}), x1), _\#_1(_\text{Dummy_38}), _\#_1(_\text{Dummy_38})), \\ & (_\#_1(_\text{Dummy_38}),), (x0,)) / f(x0, x1) - \text{Subs}(\text{Derivative}(f(x0, _\#_0(_\text{Dummy_38})), \\ & _\#_0(_\text{Dummy_38}), _\#_0(_\text{Dummy_38})), (_\#_0(_\text{Dummy_38}),), (x1,)) / f(x0, \\ & x1) + \text{Subs}(\text{Derivative}(f(_\text{xi_1}, x1), _\text{xi_1}), (_\text{xi_1}), (x0,))^{**2} / (2^{*}f(x0, x1)^{**2}) \\ & + \text{Subs}(\text{Derivative}(f(x0, _\text{xi_2}), _\text{xi_2}), (_\text{xi_2}), (x1,))^{**2} / (2^{*}f(x0, x1)^{**2}) \end{aligned}$$

After converting to LaTeX, it becomes:

$$- \frac{1}{f} \left(f \left(\left(x_0 \right), \left(x_1 \right) \right) \right) \left| \right. \frac{d^2}{d} \operatorname{operatorname}_{\#1} \left(\left(\left(\operatorname{Dummy}_{38} \right) \right)^2 \right) f \left(\left(\operatorname{operatorname}_{\#1} \right) \left(\left(\operatorname{Dummy}_{38} \right) \right), \left(x_1 \right) \right) \right| \left| \right| \substack{\operatorname{operatorname}_{\#1} \left(\left(\operatorname{Dummy}_{38} \right) \right) = \left(x_0 \right)} \left| \right| - \frac{1}{f} \left(f \left(\left(x_0 \right), \left(x_1 \right) \right) \right) \left| \right. \frac{d^2}{d} \operatorname{operatorname}_{\#0} \left(\left(\operatorname{Dummy}_{38} \right) \right)^2 \left| \right. f \left(\left(x_0 \right), \operatorname{operatorname}_{\#0} \left(\left(\operatorname{Dummy}_{38} \right) \right) \right) \left| \right. \left| \right| \substack{\operatorname{operatorname}_{\#0} \left(\left(\operatorname{Dummy}_{38} \right) \right) = x_1} \left| \right| \left| \right| + \frac{d}{d} \xi_1 \left(\xi_1, \left(x_1 \right) \right)$$

$$\left. \frac{1}{f(x_0, x_1)} \frac{d^2}{d\xi^2} f(\xi, x_1) \right|_{\xi=x_0} - \frac{1}{f(x_0, x_1)} \frac{d^2}{d\xi^2} f(x_0, \xi) \Big|_{\xi=x_1} \\ + \frac{\left. \frac{d}{d\xi_1} f(\xi_1, x_1) \right|_{\xi_1=x_0}^2}{2f^2(x_0, x_1)} + \frac{\left. \frac{d}{d\xi_2} f(x_0, \xi_2) \right|_{\xi_2=x_1}^2}{2f^2(x_0, x_1)}$$

In fact, this will even throw an error! At this point, we also need to replace content like `\operatorname{_{#_0}}{\left (Dummy_{38} \right )}` with ordinary symbols. For example:

```

1 print latex(R[0,1,0,1]).replace('\mathrm{x_','{x_}').replace('\boldsymbol{{x_','{{x_}}}\
2 .replace('\operatorname{_{#_0}}{\left (Dummy_{38} \right )}','\xi')\
3 .replace('\operatorname{_{#_1}}{\left (Dummy_{38} \right )}','\xi')

```

This finally gives an output that doesn't error and looks somewhat reasonable:

$$\begin{aligned}
& - \frac{1}{f(x_0, x_1)} \frac{d^2}{d\xi^2} f(\xi, x_1) \Big|_{\xi=x_0} - \frac{1}{f(x_0, x_1)} \frac{d^2}{d\xi^2} f(x_0, \xi) \Big|_{\xi=x_1} \\
& + \frac{\left. \frac{d}{d\xi_1} f(\xi_1, x_1) \right|_{\xi_1=x_0}^2}{2f^2(x_0, x_1)} + \frac{\left. \frac{d}{d\xi_2} f(x_0, \xi_2) \right|_{\xi_2=x_1}^2}{2f^2(x_0, x_1)}
\end{aligned}$$

Which can finally be organized as:

$$-\frac{\frac{\partial^2 f(x_0, x_1)}{\partial x_0^2}}{f(x_0, x_1)} - \frac{\frac{\partial^2 f(x_0, x_1)}{\partial x_1^2}}{f(x_0, x_1)} + \frac{\left(\frac{\partial f(x_0, x_1)}{\partial x_0} \right)^2}{2f^2(x_0, x_1)} + \frac{\left(\frac{\partial f(x_0, x_1)}{\partial x_1} \right)^2}{2f^2(x_0, x_1)}$$

Or:

$$-\frac{\nabla^2 f}{f} + \frac{1}{2} \left| \frac{\nabla f}{f} \right|^2$$

Original Address: <https://kexue.fm/archives/4026>

For more details on reprinting, please refer to: Scientific Space FAQ