

# SO(3) transformations and Jacobian of extended log map

Zhakshylyk Nurlanov

## 1 SO(3) exponential map

### 1.1 Axis-angle to Matrix

The map:

$$\exp : \mathfrak{so}(3) \mapsto \mathbf{SO}(3) \quad (1a)$$

$$\boldsymbol{\omega} \mapsto \mathbf{R}_{3 \times 3} \quad (1b)$$

is well-defined, surjective, and corresponds to the matrix exponentiation, which has the closed-form solution: the Rodrigues' formula from 1840, that is

$$e^{\boldsymbol{\omega}} \equiv \text{matexp}(\boldsymbol{\omega}^\wedge) = \mathbf{I}_3 + \frac{\sin \theta}{\theta} \boldsymbol{\omega}^\wedge + \frac{1 - \cos \theta}{\theta^2} (\boldsymbol{\omega}^\wedge)^2 \quad (2)$$

where the angle  $\theta = |\boldsymbol{\omega}|$  and  $\boldsymbol{\omega}^\wedge$  is the skew symmetric matrix generated by the 3-vector  $\boldsymbol{\omega}$ .

We can define a unit vector, representing the axis of rotation, as  $\mathbf{n} = \frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|} = (n_1, n_2, n_3)$  with respect to a fixed Cartesian coordinate system, and the angle of rotation  $\theta = |\boldsymbol{\omega}|$  around this axis. One can show that the Rodrigues' formula (eq. 2) for the rotation matrix  $\mathbf{R}(\mathbf{n}, \theta)$  representing rotation around axis  $\mathbf{n}$  about the angle  $\theta$  in the coordinate form can be written as:

$$\mathbf{R}(\mathbf{n}, \theta) = \begin{pmatrix} \cos \theta + n_1^2(1 - \cos \theta) & n_1 n_2(1 - \cos \theta) - n_3 \sin \theta & n_1 n_3(1 - \cos \theta) + n_2 \sin \theta \\ n_1 n_2(1 - \cos \theta) + n_3 \sin \theta & \cos \theta + n_2^2(1 - \cos \theta) & n_2 n_3(1 - \cos \theta) - n_1 \sin \theta \\ n_1 n_3(1 - \cos \theta) - n_2 \sin \theta & n_2 n_3(1 - \cos \theta) + n_1 \sin \theta & \cos \theta + n_3^2(1 - \cos \theta) \end{pmatrix} \quad (3)$$

It is also usefull to derive the following representation of rotation matrix:

$$\mathbf{R}(\mathbf{n}, \theta) = P \mathbf{R}(z, \theta) P^{-1} \quad (4)$$

where  $P$  is an orthogonal matrix, i.e.  $P^{-1} = P^\top$ , and  $\mathbf{R}(z, \theta)$  is a standart rotation matrix around  $z$ -axis about angle  $\theta$ :

$$P = \begin{pmatrix} \frac{n_3 n_1}{\sqrt{n_1^2 + n_2^2}} & \frac{-n_2}{\sqrt{n_1^2 + n_2^2}} & n_1 \\ \frac{n_3 n_2}{\sqrt{n_1^2 + n_2^2}} & \frac{n_1}{\sqrt{n_1^2 + n_2^2}} & n_2 \\ -\sqrt{n_1^2 + n_2^2} & 0 & n_3 \end{pmatrix}, \quad \mathbf{R}(z, \theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (5)$$

## 1.2 Axis-angle to Quaternion

The exponential map can be also directly mapped as a unit quaternion  $(q_r \ q_x \ q_y \ q_z)^\top$ :

$$\exp : \mathfrak{so}(3) \mapsto \mathbf{SO}(3) \quad (6a)$$

$$\boldsymbol{\omega} \mapsto \mathbf{SU}(2) \quad (6b)$$

$$e_q^\omega = \begin{cases} (1, 0, 0, 0)^\top & , \text{ if } \boldsymbol{\omega} = (0, 0, 0)^\top \\ \left( \cos \frac{|\boldsymbol{\omega}|}{2}, \frac{\sin \frac{|\boldsymbol{\omega}|}{2}}{|\boldsymbol{\omega}|} \boldsymbol{\omega} \right)^\top & , \text{ otherwise} \end{cases} \quad (6c)$$

## 2 SO(3) logarithm map

### 2.1 Matrix to Axis-angle (baseline)

The map:

$$\log : \mathbf{SO}(3) \mapsto \mathfrak{so}(3) \quad (7a)$$

$$\mathbf{R}_{3 \times 3} \mapsto \boldsymbol{\omega} \quad (7b)$$

is well-defined for rotation angles  $\theta \in (0, \pi)$ , surjective, and is the inverse of the exp function defined above. From Rodrigues' formula (eq. 3) or from rotation matrix factorization and trace properties (eq. 4) it follows that:

$$\cos \theta = \frac{1}{2}(tr(\mathbf{R}) - 1) \quad (8a)$$

$$\sin \theta = (1 - \cos^2 \theta)^{1/2} = \frac{1}{2} \sqrt{(3 - tr(\mathbf{R}))(1 + tr(\mathbf{R}))} \quad (8b)$$

where  $\sin \theta \geq 0$  is a consequence of the convention for the range of the rotation angle,  $\theta \in [0, \pi]$ .

Then from Rodrigues' formula (eq. 3) and Taylor approximation it follows that logarithmic map can be implemented as follows:

$$\boldsymbol{\omega} = [\log(\mathbf{R})]^\vee = \frac{\theta}{2 \sin \theta} [\mathbf{R} - \mathbf{R}^\top]^\vee = \frac{\theta}{2 \sin \theta} (R_{32} - R_{23}, R_{13} - R_{31}, R_{21} - R_{12})^\top, \quad (9a)$$

$$\boldsymbol{\omega} = 0.5 \cdot \left(1 + \frac{1}{6}\theta^2 + \frac{7}{360}\theta^4\right) [\mathbf{R} - \mathbf{R}^\top]^\vee, \quad \text{if } \text{abs}(3 - tr \mathbf{R}) < 10^{-8} \quad (9b)$$

The above implementation is the most common, however, it diverges at a special case when the angle of rotation  $\theta$  is close to  $\pi$ . Nevertheless, it is possible to describe the log map at these edge cases when  $\sin \theta = 0$ , that is  $\theta = 0$  or  $\theta = \pi$ . In both cases (from eq. 3)  $R_{ij} = R_{ji}$ , and  $\boldsymbol{\omega}$  can not be determined by eq. 9. However, the angle  $\theta$  is derived from eq. 8, and inserting it to the Rodrigues' formula 2:

$$\frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|} = \mathbf{n} = \begin{cases} \boldsymbol{\omega} = (0, 0, 0) & \text{if } \theta = 0, \\ \left( \epsilon_1 \sqrt{\frac{1}{2}(1 + R_{11})}, \epsilon_2 \sqrt{\frac{1}{2}(1 + R_{22})}, \epsilon_3 \sqrt{\frac{1}{2}(1 + R_{33})} \right)^\top & \text{if } \theta = \pi \end{cases} \quad (10)$$

where the individual signs  $\epsilon_i = \pm 1$  (if  $n_i \neq 0$ ) are determined up to an overall sign (since  $\mathbf{R}(\mathbf{n}, \pi) = \mathbf{R}(\mathbf{n}, -\pi)$ ) via the following relation:

$$\epsilon_i \epsilon_j = \frac{R_{ij}}{\sqrt{(1 + R_{ii})(1 + R_{jj})}}, \text{ for } i \neq j, R_{ii} \neq -1, R_{jj} \neq -1 \quad (11)$$

## 2.2 Matrix to Angle-axis (around $\pi$ )

There is an alternative approach for working around the case  $\theta = \pi$ , which determines the axis of rotation  $\mathbf{n}$  for angles close to  $\pi$  without numerical issues. We define matrix:

$$S \equiv \mathbf{R} + \mathbf{R}^\top + (1 - \text{tr} \mathbf{R}) \mathbf{I}_3 \quad (12)$$

Then the Rodrigues' equation in coordinate form (eq. 3) yields:

$$n_j n_k = \frac{S_{jk}}{3 - \text{tr}(\mathbf{R})}, \quad \text{tr}(\mathbf{R}) \neq 3 \quad (13)$$

To determine  $\mathbf{n}$  up to an overall sign, we simply set  $j = k$  in eq. (13), which fixes the value of  $n_j^2$ . If  $\sin \theta \neq 0$ , the overall sign of  $\mathbf{n}$  is determined by eq. (9). If  $\sin \theta = 0$  then there are two cases. For  $\theta = 0$  (corresponding to the identity rotation),  $S = 0$  and the rotation axis  $\mathbf{n}$  is undefined, but the logarithmic map can be derived as in 9b. For  $\theta = \pi$ , the ambiguity in the overall sign of  $\mathbf{n}$  is immaterial, since  $\mathbf{R}(\mathbf{n}, \pi) = \mathbf{R}(\mathbf{n}, -\pi)$ , so we can fix the sign of maximal component  $n_i$  to be positive and derive the signs of all other components according to the signs of corresponding  $S_{ij}$ .

## 2.3 Matrix to Quaternion

The transformation from rotation matrix  $\mathbf{R}$  to quaternion  $\mathbf{q} = (q_r \ q_x \ q_y \ q_z)^\top = (q_r, \mathbf{q}_v^\top)^\top$  is well-defined for all angles due to resolving ambiguities through case differentiation.

$$\begin{aligned} & \text{if } \text{tr}(\mathbf{R}) > 0 \\ & \quad t = \sqrt{1 + \text{tr}(\mathbf{R})}, \quad q_r = 0.5 \cdot t, \quad \mathbf{q}_v = 0.5/t \cdot (\mathbf{R} - \mathbf{R}^\top)^\vee \end{aligned} \quad (14a)$$

$$\begin{aligned} & \text{else if } R_{11} \geq R_{22}, \quad R_{11} \geq R_{33} \\ & \quad t = \sqrt{1 + R_{11} - R_{22} - R_{33}}, \quad q_x = 0.5 \cdot t, \\ & \quad q_r = 0.5/t \cdot (R_{3,2} - R_{2,3}), \quad q_y = 0.5/t \cdot (R_{2,1} + R_{1,2}), \quad q_z = 0.5/t \cdot (R_{3,1} + R_{1,3}) \end{aligned} \quad (14b)$$

$$\begin{aligned} & \text{else if } R_{22} > R_{11}, \quad R_{22} \geq R_{33} \\ & \quad t = \sqrt{1 + R_{22} - R_{33} - R_{11}}, \quad q_y = 0.5 \cdot t, \\ & \quad q_r = 0.5/t \cdot (R_{1,3} - R_{3,1}), \quad q_z = 0.5/t \cdot (R_{3,2} + R_{2,3}), \quad q_x = 0.5/t \cdot (R_{1,2} + R_{2,1}) \end{aligned} \quad (14c)$$

$$\begin{aligned} & \text{else if } R_{33} > R_{11}, \quad R_{33} > R_{22} \\ & \quad t = \sqrt{1 + R_{33} - R_{11} - R_{22}}, \quad q_z = 0.5 \cdot t, \\ & \quad q_r = 0.5/t \cdot (R_{2,1} - R_{1,2}), \quad q_x = 0.5/t \cdot (R_{1,3} + R_{3,1}), \quad q_y = 0.5/t \cdot (R_{2,3} + R_{3,2}) \end{aligned} \quad (14d)$$

## 2.4 Quaternion to Axis-angle

The logarithm map can be directly given from a unit quaternion  $\mathbf{q} = (q_r \ q_x \ q_y \ q_z)^\top = (q_r, \mathbf{q}_v^\top)^\top$ :

$$\log : \mathbf{SO}(3) \mapsto \mathfrak{so}(3) \quad (15a)$$

$$\mathbf{SU}(2) \mapsto \boldsymbol{\omega} \quad (15b)$$

$$\boldsymbol{\omega} = 2 \arccos(q_r) \frac{\mathbf{q}_v}{\|\mathbf{q}_v\|_2} \quad (15c)$$

Or the numerically more stable implementation:

$$\mathbf{q} = \text{sign}(q_r) \mathbf{q} \quad (16a)$$

$$\boldsymbol{\omega} = \left( \frac{2}{q_r} - \frac{2}{3} \cdot \frac{\|\mathbf{q}_v\|_2^2}{q_r^3} \right) \mathbf{q}_v, \quad \text{if } \|\mathbf{q}_v\|_2 < \epsilon \quad (16b)$$

$$\boldsymbol{\omega} = 4 \arctan \left( \frac{\|\mathbf{q}_v\|_2}{q_r + \sqrt{q_r^2 + \|\mathbf{q}_v\|_2^2}} \right) \cdot \frac{\mathbf{q}_v}{\|\mathbf{q}_v\|_2}, \quad \text{else} \quad (16c)$$

## 3 Jacobian of SO(3) logarithm map

### 3.1 Inverse Jacobian of exponential map

According to the definition of derivatives on manifold, the (right) Jacobian of logarithm map will be expressed as the linear mapping between two tangent spaces:

$$\frac{\partial \log(R \boxplus \mathbf{x})}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{0}} = \frac{\partial \log(R \exp(\mathbf{x}))}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{0}} = J_r^{-1} \Big|_{3 \times 3} \quad (17)$$

where (see cheatsheets<sup>1,2</sup>)  $J_r^{-1}$  is known as the inverse right Jacobian of exponential mapping:

$$\boldsymbol{\omega} = \log(R), \quad \theta = \|\boldsymbol{\omega}\|_2 \quad (18a)$$

$$J_r^{-1} = I + \frac{\boldsymbol{\omega}^\wedge}{2} + \left( \frac{1}{\theta^2} - \frac{1 + \cos(\theta)}{2\theta \sin(\theta)} \right) (\boldsymbol{\omega}^\wedge)^2, \quad \text{if } \theta \in [\epsilon, \pi - \epsilon], \quad (18b)$$

$$J_r^{-1} = I + \frac{\boldsymbol{\omega}^\wedge}{2} + \left( \frac{1}{\pi^2} + \frac{(\pi^2 - 8)(\theta - \pi)}{4\pi^3} \right) (\boldsymbol{\omega}^\wedge)^2, \quad \text{if } \theta \in (\pi - \epsilon, \pi], \quad (18c)$$

$$J_r^{-1} = I + \frac{\boldsymbol{\omega}^\wedge}{2} + \left( \frac{1}{12} + \frac{\theta^2}{720} \right) (\boldsymbol{\omega}^\wedge)^2, \quad \text{if } \theta \in [0, \epsilon) \quad (18d)$$

We can also express the derivative of boxplus operator  $\boxplus$ :  $\frac{\partial(R \boxplus \mathbf{x})}{\partial \mathbf{x}} \Big|_{9 \times 3 \text{ or } 4 \times 3} = \frac{\partial(R \exp(\mathbf{x}))}{\partial \mathbf{x}}$  using the derivative of exponential map  $\exp$  and the derivative of composition  $\circ$ . These derivatives are well-defined and their form depend on the representation of exponential map.

<sup>1</sup>[https://docs.leggedrobotics.com/kindr/cheatsheet\\_latest.pdf](https://docs.leggedrobotics.com/kindr/cheatsheet_latest.pdf)

<sup>2</sup>[https://ingmec.ual.es/~jlblanco/papers/jlblanco2010geometry3D\\_techrep.pdf](https://ingmec.ual.es/~jlblanco/papers/jlblanco2010geometry3D_techrep.pdf)

### 3.2 Jacobian of extended log map

However, the derivative  $\frac{\partial \log(R)}{\partial R}$  is not defined in a common sense, because an arbitrary perturbation of  $R \in \mathbf{SO}(3)$  or  $R \in \mathbf{SU}(2)$  (here,  $R$  can be either rotation matrix or unit quaternion) might take it out of the manifold. If we extend the logarithmic map  $\log(R)$  out of the manifold to the smallest open super set of the manifold  $O(\mathbf{SO}(3)) \supset \mathbf{SO}(3)$ , i.e. extended log map:

$$\overline{\log} : O(\mathbf{SO}(3)) \mapsto \mathfrak{so}(3) \subseteq \mathbb{R}^3 \quad (19)$$

then we would be able to define derivative of log map in a numerical sense.

Using the results in a previous subsection and a chain rule we can check the following relation:

$$\frac{\partial \log(R \boxplus \mathbf{x})}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{0}} = \frac{\partial \overline{\log}(R)}{R} \Big|_{3 \times 9 \text{ or } 3 \times 4} \cdot \frac{\partial (R \boxplus \mathbf{x})}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{0}} \Big|_{9 \times 3 \text{ or } 4 \times 3} \quad (20)$$

Depending on the implementation of extended logarithmic map  $\overline{\log}$  we can derive different Jacobians  $\frac{\partial \overline{\log}(R)}{R}$ . Above we described 3 different implementations of log map (matrix to angle-axis) denoted as: "baseline", "around  $\pi$ ", and through "quaternion" (as the composition of two maps). All of them can be naturally extended to  $O(\mathbf{SO}(3))$  with minor changes, i.e. ensuring that  $\cos$  and  $\sin$  are in  $[-1, 1]$  range, and that the argument of a square root is non-negative via clipping the invalid values into valid ranges. In the experimental notebook<sup>3</sup> we derive the analytical Jacobians for each of the extended log maps, compare them against numerical differentiation and test the chain rule (20).

Interestingly, **the result of the chain rule (20) does not depend on the implementation of an extended logarithmic map, although the Jacobians of extended log maps are different.** This experimental result needs further theoretical analysis together with deriving the conditions for extending the maps where the result holds.

#### 3.2.1 Jacobian of extended log map (baseline)

Given an input rotation matrix  $\mathbf{R}$ :

$$\mathbf{R} = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix}$$

it can be shown that:

$$\frac{\partial \log(\mathbf{R})^\vee}{\partial \mathbf{R}} \Big|_{3 \times 9} = \begin{cases} \left( \begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \end{array} \right), & \text{if } \cos \theta > 1 - \epsilon \\ \left( \begin{array}{ccc|ccc|ccc} a_1 & 0 & 0 & 0 & a_1 & -b & 0 & b & a_1 \\ a_2 & 0 & b & 0 & a_2 & 0 & -b & 0 & a_2 \\ a_3 & -b & 0 & b & a_3 & 0 & 0 & 0 & a_3 \end{array} \right), & \text{otherwise} \end{cases} \quad (21)$$

where the order of the 9 components is assumed to be row-major ( $R_{11}, R_{12}, \dots$ ) and:

<sup>3</sup>[https://github.com/nurlanov-zh/so3\\_log\\_map/blob/main/so3\\_log\\_map\\_analysis.ipynb](https://github.com/nurlanov-zh/so3_log_map/blob/main/so3_log_map_analysis.ipynb)

$$\begin{aligned}
\cos \theta &= \frac{\text{tr}(\mathbf{R}) - 1}{2} \\
\sin \theta &= \sqrt{1 - \cos^2 \theta} \\
\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} &= \left[ \mathbf{R} - \mathbf{R}^\top \right]^\vee \frac{\theta \cos \theta - \sin \theta}{4 \sin^3 \theta} = \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix} \frac{\theta \cos \theta - \sin \theta}{4 \sin^3 \theta} \\
b &= \frac{\theta}{2 \sin \theta}
\end{aligned}$$

### 3.2.2 Jacobian of extended log map (around $\pi$ )

```

1 def Dx_log_x_pi(R):
2     trR = R[0, 0] + R[1, 1] + R[2, 2]
3     cos_theta = max(min(0.5 * (trR - 1), 1), -1)
4     sin_theta = np.sqrt(1 - cos_theta * cos_theta)
5     theta = np.arctan2(sin_theta, cos_theta)
6     R_minus_R_T_vee = np.array([R[2, 1] - R[1, 2],
7                                   R[0, 2] - R[2, 0], R[1, 0] - R[0, 1]])
8
9     if abs(3 - trR) < 1e-8:
10         # return Jacobian of log map at Theta = 0
11         return np.array([[0, 0, 0, 0, 0, -0.5, 0, 0.5, 0],
12                           [0, 0, 0.5, 0, 0, 0, -0.5, 0, 0],
13                           [0, -0.5, 0, 0.5, 0, 0, 0, 0, 0]])
14
15     S = R + R.transpose() + (1 - trR) * np.eye(3)
16     rest_tr = 3 - trR
17     n = np.ones(3)
18     # Fix modules of n_i
19     for i in range(3):
20         n[i] = np.sqrt(max(0, S[i, i] / rest_tr))
21     max_i = np.argmax(n)
22
23     # Fix signs according to the sign of max element
24     for i in range(3):
25         if i != max_i:
26             n[i] *= np.sign(S[max_i, i])
27
28     # Fix an overall sign
29     if any(np.sign(n) * np.sign(R_minus_R_T_vee) < 0):
30         n = -n
31
32     dn_dR_11 = np.zeros(3)
33     dn_dR_22 = np.zeros(3)
34     dn_dR_33 = np.zeros(3)
35     if abs(n[0]) > np.finfo(float).eps:
36         dn_dR_11[0] = (S[0, 0] + rest_tr) / (2 * n[0] * rest_tr * rest_tr)
37         dn_dR_22[0] = (S[0, 0] - rest_tr) / (2 * n[0] * rest_tr * rest_tr)
38         dn_dR_33[0] = dn_dR_22[0]
39     if abs(n[1]) > np.finfo(float).eps:
40         dn_dR_11[1] = (S[1, 1] - rest_tr) / (2 * n[1] * rest_tr * rest_tr)
41         dn_dR_22[1] = (S[1, 1] + rest_tr) / (2 * n[1] * rest_tr * rest_tr)

```

```

42     dn_dR_33[1] = dn_dR_11[1]
43     if abs(n[2]) > np.finfo(float).eps:
44         dn_dR_11[2] = (S[2, 2] - rest_tr) / (2 * n[2] * rest_tr * rest_tr)
45         dn_dR_22[2] = dn_dR_11[2]
46         dn_dR_33[2] = (S[2, 2] + rest_tr) / (2 * n[2] * rest_tr * rest_tr)
47
48     dtheta_d_R_11 = -sin_theta * 0.5 + cos_theta * 0.5 * (dn_dR_11.dot(
49 R_minus_R_T_vee))
50     dtheta_d_R_22 = -sin_theta * 0.5 + cos_theta * 0.5 * (dn_dR_22.dot(
51 R_minus_R_T_vee))
52     dtheta_d_R_33 = -sin_theta * 0.5 + cos_theta * 0.5 * (dn_dR_33.dot(
53 R_minus_R_T_vee))
54
55     dtheta_d_R_12 = cos_theta * 0.5 * (-n[2])
56     dtheta_d_R_13 = cos_theta * 0.5 * (n[1])
57     dtheta_d_R_23 = cos_theta * 0.5 * (-n[0])
58     dtheta_d_R_21 = -dtheta_d_R_12
59     dtheta_d_R_31 = -dtheta_d_R_13
60     dtheta_d_R_32 = -dtheta_d_R_23
61
62     J = np.zeros((3, 3, 3))
63
64     J[0, 0, 0] = dtheta_d_R_11 * n[0] + theta * dn_dR_11[0]
65     J[0, 1, 1] = dtheta_d_R_22 * n[0] + theta * dn_dR_22[0]
66     J[0, 2, 2] = dtheta_d_R_33 * n[0] + theta * dn_dR_33[0]
67     J[0, 0, 1] = dtheta_d_R_12 * n[0]
68     J[0, 0, 2] = dtheta_d_R_13 * n[0]
69     J[0, 1, 2] = dtheta_d_R_23 * n[0]
70     J[0, 1, 0] = -J[0, 0, 1]
71     J[0, 2, 0] = -J[0, 0, 2]
72     J[0, 2, 1] = -J[0, 1, 2]
73
74     J[1, 0, 0] = dtheta_d_R_11 * n[1] + theta * dn_dR_11[1]
75     J[1, 1, 1] = dtheta_d_R_22 * n[1] + theta * dn_dR_22[1]
76     J[1, 2, 2] = dtheta_d_R_33 * n[1] + theta * dn_dR_33[1]
77     J[1, 0, 1] = dtheta_d_R_12 * n[1]
78     J[1, 0, 2] = dtheta_d_R_13 * n[1]
79     J[1, 1, 2] = dtheta_d_R_23 * n[1]
80     J[1, 1, 0] = -J[1, 0, 1]
81     J[1, 2, 0] = -J[1, 0, 2]
82     J[1, 2, 1] = -J[1, 1, 2]
83
84     J[2, 0, 0] = dtheta_d_R_11 * n[2] + theta * dn_dR_11[2]
85     J[2, 1, 1] = dtheta_d_R_22 * n[2] + theta * dn_dR_22[2]
86     J[2, 2, 2] = dtheta_d_R_33 * n[2] + theta * dn_dR_33[2]
87     J[2, 0, 1] = dtheta_d_R_12 * n[2]
88     J[2, 0, 2] = dtheta_d_R_13 * n[2]
89     J[2, 1, 2] = dtheta_d_R_23 * n[2]
90     J[2, 1, 0] = -J[2, 0, 1]
91     J[2, 2, 0] = -J[2, 0, 2]
92     J[2, 2, 1] = -J[2, 1, 2]
93
94     return J.reshape(3, 9)

```

### 3.2.3 Jacobian of extended log map (quaternion)

Jacobian of Matrix to Quaternion map:

```
1 def Dquaternion_DR(R):
2     """Computes d quaternion(R) / d R , 4 x 9 Jacobian."""
3     J_quat = np.zeros((4, 3, 3))
4     t = R[0, 0] + R[1, 1] + R[2, 2]
5     if t > 0:
6         # case 1
7         isqrt_t = 1 / np.sqrt(1 + t)
8         J_quat[0, :, :] = 0.25 * isqrt_t * np.eye(3)
9         J_quat[1, :, :] = -0.25 * isqrt_t / (1 + t) * (R[2, 1] - R[1, 2]) * np.eye(3)
10        J_quat[1, 2, 1] = 0.5 * isqrt_t
11        J_quat[1, 1, 2] = -0.5 * isqrt_t
12        J_quat[2, :, :] = -0.25 * isqrt_t / (1 + t) * (R[0, 2] - R[2, 0]) * np.eye(3)
13        J_quat[2, 0, 2] = 0.5 * isqrt_t
14        J_quat[2, 2, 0] = -0.5 * isqrt_t
15        J_quat[3, :, :] = -0.25 * isqrt_t / (1 + t) * (R[1, 0] - R[0, 1]) * np.eye(3)
16        J_quat[3, 1, 0] = 0.5 * isqrt_t
17        J_quat[3, 0, 1] = -0.5 * isqrt_t
18    else:
19        i = 0
20        if R[1, 1] > R[0, 0]:
21            i = 1
22        if R[2, 2] > R[i, i]:
23            i = 2
24        j = (i + 1) % 3
25        k = (j + 1) % 3
26        r = np.sqrt(R[i, i] - R[j, j] - R[k, k] + 1)
27        i_r = 1 / r
28        i_r_cube = 1 / ((R[i, i] - R[j, j] - R[k, k] + 1) * r)
29        r_eye = np.eye(3)
30        r_eye[j, j] = -1
31        r_eye[k, k] = -1
32        J_quat[1 + i, :, :] = 0.25 * i_r * r_eye
33
34        J_quat[0, :, :] = -0.25 * (R[k, j] - R[j, k]) * i_r_cube * r_eye
35        J_quat[0, k, j] = 0.5 * i_r
36        J_quat[0, j, k] = -0.5 * i_r
37        J_quat[1 + j, :, :] = -0.25 * (R[j, i] + R[i, j]) * i_r_cube * r_eye
38        J_quat[1 + j, j, i] = 0.5 * i_r
39        J_quat[1 + j, i, j] = 0.5 * i_r
40        J_quat[1 + k, :, :] = -0.25 * (R[k, i] + R[i, k]) * i_r_cube * r_eye
41        J_quat[1 + k, k, i] = 0.5 * i_r
42        J_quat[1 + k, i, k] = 0.5 * i_r
43    return J_quat.reshape((4, 9))
```

Jacobian of Quaternion to Axis-angle map:

```
1 def Dlog_Dquaternion2(q):
2     """Computes d log(q) / d q , 3 x 4 Jacobian."""
3     J_vec = np.zeros((3, 4))
4     squared_n = q[1]*q[1] + q[2]*q[2] + q[3]*q[3]
5     n = np.sqrt(squared_n)
6     w = q[0]
7     squared_w = w*w
8     sign = 1
```



```

9     if w < 0:
10         sign = -1
11         w = -w
12     if squared_n < np.finfo(float).eps * np.finfo(float).eps:
13         # n (~Theta) close to 0
14         two_atan_nbyw_by_n = 2 / w - 2.0 / 3 * squared_n / (w * squared_w)
15         d_q0 = -2 / squared_w + 2 * squared_n / (squared_w * squared_w)
16         d_q1 = -4.0 / 3 * q[1] / (w * squared_w)
17         d_q2 = -4.0 / 3 * q[2] / (w * squared_w)
18         d_q3 = -4.0 / 3 * q[3] / (w * squared_w)
19     else:
20         # Regular case
21         two_atan_nbyw_by_n = 4 * np.arctan(n / (w + np.sqrt(squared_w + squared_n)))
22     / n
23     d_q0 = -2 / (squared_w + squared_n)
24     c0 = (2 * w - two_atan_nbyw_by_n) / squared_n
25     d_q1 = c0 * q[1]
26     d_q2 = c0 * q[2]
27     d_q3 = c0 * q[3]
28     J_vec[:, 0] = sign * d_q0 * q[1:]
29     J_vec[:, 1] = d_q1 * q[1:]
30     J_vec[0, 1] += two_atan_nbyw_by_n
31     J_vec[:, 2] = d_q2 * q[1:]
32     J_vec[1, 2] += two_atan_nbyw_by_n
33     J_vec[:, 3] = d_q3 * q[1:]
34     J_vec[2, 3] += two_atan_nbyw_by_n
35     J_vec = J_vec * sign
36     return J_vec

```

Jacobian of Matrix to Axis-angle (through quaternion) map:

```

1 def Dx_log_x_quaternion(R):
2     """Computes d log(R) / d R , 3 x 9 Jacobian."""
3     J_quat = Dquaternion_DR(R)
4     q = SO3.matrix2quaternion(R)
5     J_vec = Dlog_Dquaternion2(q)
6     return J_vec @ J_quat

```

### 3.2.4 Comparison of Jacobians of extended log map

1. The Jacobian of extended log map depends on the implementation of log map. All of the implementations produce different values.
2. However, after matrix multiplication with Jacobian of the box plus at zero the values of the product become the same (except the case when the Jacobian around  $\pi$  nullifies rows corresponding to zero components of the rotation axis, i.e.  $n_i = 0 \rightsquigarrow J^\pi[i, :] = 0$ ).
3. For small angles  $\theta$  close to 0: the Jacobian around  $\pi$  diverges (tends to infinity).
4. For large angles  $\theta$  close to  $\pi$ : the baseline Jacobian diverges.
5. The Jacobian through intermediate quaternion representation does not diverge anywhere.