Exploring $\mathbf{SO}(3)$ logarithmic map: degeneracies and derivatives

Zhakshylyk Nurlanov

Abstract

In this work, we investigate the SO(3) logarithmic map from various points of view. First, we propose two numerically stable solutions to the degenerate case of the logarithmic map (from SO(3) rotation matrices to so(3)-Lie algebra vectors) when the angle of rotation is close to or exactly pi. Second, we observe and theoretically prove that the Jacobian of the logarithmic map in a manifold sense (as the linear map between tangent spaces) can be found as a chain of Jacobians in a numerical sense of the logarithmic map and box plus operator via extending logarithmic map over the SO(3) manifold. Moreover, we show that the result of the chain rule does not depend on the extension of the log map, even though the numerical Jacobians of different extensions are different. These results ensure that we do not run into degenerate cases in practical implementations of optimization algorithms, such as with the Ceres solver library, for applications like pose graph optimization. We have proposed corresponding improvements to the implementation of SO3 in the popular Sophus library.

1 $\mathbf{SO}(3)$ exponential map

1.1 Axis-angle to Matrix

The map:

$$\begin{aligned} \exp : \mathfrak{so}(3) &\mapsto \mathbf{SO}(3) && \text{(1a)} \\ \boldsymbol{\omega} &\mapsto \mathbf{R}_{3\times 3} && \text{(1b)} \end{aligned}$$

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

$$e^{\boldsymbol{\omega}} \equiv \text{matexp}(\boldsymbol{\omega}^\wedge) = \mathbf{I_3} + \frac{\sin\theta}{\theta}\boldsymbol{\omega}^\wedge + \frac{1-\cos\theta}{\theta^2}(\boldsymbol{\omega}^\wedge)^2 \tag{2}$$

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 θ in the coordinate form can be written as:

$$\mathbf{R}(\mathbf{n},\theta) = \begin{pmatrix} \cos\theta + n_1^2(1-\cos\theta) & n_1n_2(1-\cos\theta) - n_3\sin\theta & n_1n_3(1-\cos\theta) + n_2\sin\theta \\ n_1n_2(1-\cos\theta) + n_3\sin\theta & \cos\theta + n_2^2(1-\cos\theta) & n_2n_3(1-\cos\theta) - n_1\sin\theta \\ n_1n_3(1-\cos\theta) - n_2\sin\theta & n_2n_3(1-\cos\theta) + n_1\sin\theta & \cos\theta + n_3^2(1-\cos\theta) \end{pmatrix} \tag{3}$$

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

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

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 θ:

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

1.2 Axis-angle to Quaternion

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

$$\exp : \mathfrak{so}(3) \mapsto \mathbf{SO}(3) \tag{6a}$$
$$\boldsymbol{\omega} \mapsto \mathbf{SU}(2) \tag{6b}$$

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

2 SO(3) logarithm map

2.1 Matrix to Axis-angle (baseline)

The map:

$$\log : \mathbf{SO}(3) \mapsto \mathfrak{so}(3) \tag{7a}$$
$$\mathbf{R}_{3\times3} \mapsto \boldsymbol{\omega} \tag{7b}$$

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:

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

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:

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

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

The above implementation is the most common, however, it diverges at a special case when the angle of rotation θ is close to π. 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 θ is derived from eq. 8, and inserting it to the Rodrigues' formula 2:

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

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:

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

2.2 Matrix to Angle-axis (around π)

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

$$S \equiv \mathbf{R} + \mathbf{R}^{\top} + (1 - tr\mathbf{R})\mathbf{I_3} \tag{12}$$

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

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

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.

$$\begin{aligned}
\text{if } tr(\mathbf{R}) &> 0 \\
t &= \sqrt{1+tr(\mathbf{R})}, \quad q_r = 0.5 \cdot t, \quad \mathbf{q}_v = 0.5/t \cdot (\mathbf{R} - \mathbf{R}^\top)^\vee && (14a)\\
\text{else if } R_{11} &\geq R_{22}, \quad R_{11} \geq R_{33} \\
t &= \sqrt{1 + R_{11} - R_{22} - R_{33}}, \quad q_x = 0.5 \cdot t, \\
q_r &= 0.5/t \cdot (R_{3,2} - R_{2,3}), \quad q_y = 0.5/t \cdot (R_{2,1} + R_{1,2}), \quad q_z = 0.5/t \cdot (R_{3,1} + R_{1,3}) && (14b)\\
\text{else if } R_{22} &> R_{11}, \quad R_{22} \geq R_{33} \\
t &= \sqrt{1 + R_{22} - R_{33} - R_{11}}, \quad q_y = 0.5 \cdot t, \\
q_r &= 0.5/t \cdot (R_{1,3} - R_{3,1}), \quad q_z = 0.5/t \cdot (R_{3,2} + R_{2,3}), \quad q_x = 0.5/t \cdot (R_{1,2} + R_{2,1}) && (14c)\\
\text{else if } R_{33} &> R_{11}, \quad R_{33} > R_{22} \\
t &= \sqrt{1 + R_{33} - R_{11} - R_{22}}, \quad q_z = 0.5 \cdot t, \\
q_r &= 0.5/t \cdot (R_{2,1} - R_{1,2}), \quad q_x = 0.5/t \cdot (R_{1,3} + R_{3,1}), \quad q_y = 0.5/t \cdot (R_{2,3} + R_{3,2}) && (14d)
\end{aligned}$$

2.4 Quaternion to Axis-angle

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

$$\begin{aligned}
\log : \mathbf{SO}(3) &\mapsto \mathfrak{so}(3) && (15a)\\
\mathbf{SU}(2) &\mapsto \boldsymbol{\omega} && (15b)\\
\boldsymbol{\omega} &= 2\arccos(q_r)\ \frac{\mathbf{q}_v}{||\mathbf{q}_v||_2} && (15c)
\end{aligned}$$

Or the numerically more stable implementation:

$$\begin{aligned}
\mathbf{q} &= \text{sign}(q_r)\mathbf{q} && (16a)\\
\boldsymbol{\omega} &= \left(\frac{2}{q_r} - \frac{2}{3} \cdot \frac{||\mathbf{q}_v||_2^2}{q_r^3}\right)\mathbf{q}_v, \qquad \text{if } ||\mathbf{q}_v||_2 < \epsilon && (16b)\\
\boldsymbol{\omega} &= 4\arctan\left(\frac{||\mathbf{q}_v||_2}{q_r + \sqrt{q_r^2 + ||\mathbf{q}_v||_2^2}}\right) \cdot \frac{\mathbf{q}_v}{||\mathbf{q}_v||_2}, \qquad \text{else} && (16c)
\end{aligned}$$

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:

$$\frac{\partial \log(R \boxplus \mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\mathbf{0}} = \frac{\partial \log(R \exp(\mathbf{x}))}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\mathbf{0}} = J_r^{-1}\big|_{3\times3} \tag{17}$$

where (see cheatsheets[1,2]) J_r^{-1} is known as the inverse right Jacobian of exponential mapping:

$$\begin{aligned}
\boldsymbol{\omega} &= \log(R), \quad \theta = ||\boldsymbol{\omega}||_2 && \text{(18a)}\\
J_r^{-1} &= I + \frac{\boldsymbol{\omega}^\wedge}{2} + \left(\frac{1}{\theta^2} - \frac{1+\cos(\theta)}{2\theta\sin(\theta)}\right)(\boldsymbol{\omega}^\wedge)^2, \qquad \text{if } \theta \in [\epsilon, \pi - \epsilon], && \text{(18b)}\\
J_r^{-1} &= I + \frac{\boldsymbol{\omega}^\wedge}{2} + \left(\frac{1}{\pi^2} + \frac{(\pi^2-8)(\theta-\pi)}{4\pi^3}\right)(\boldsymbol{\omega}^\wedge)^2, \qquad \text{if } \theta \in (\pi - \epsilon, \pi], && \text{(18c)}\\
J_r^{-1} &= I + \frac{\boldsymbol{\omega}^\wedge}{2} + \left(\frac{1}{12} + \frac{\theta^2}{720}\right)(\boldsymbol{\omega}^\wedge)^2, \qquad \text{if } \theta \in [0, \epsilon) && \text{(18d)}
\end{aligned}$$

We can also express the derivative of boxplus operator $\boxplus$: $\frac{\partial(R \boxplus \mathbf{x})}{\partial \mathbf{x}}\big|_{9\times3 \text{ or } 4\times3} = \frac{\partial(R\exp(\mathbf{x}))}{\partial \mathbf{x}}$ using the derivative of exponential map [Gallego and Yezzi, 2013] and the derivative of composition [Blanco, 2010]. These derivatives are well-defined and their form depend on the representation of exponential map.

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:

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

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:

$$\frac{\partial \log(R \boxplus \mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\mathbf{0}} = \frac{\partial \overline{\log}(R)}{R}\Big|_{3\times9 \text{ or } 3\times4} \cdot \frac{\partial(R \boxplus \mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\mathbf{0}}\Big|_{9\times3 \text{ or } 4\times3} \qquad (20)$$

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 π", 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[3] 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.

[1]`https://docs.leggedrobotics.com/kindr/cheatsheet_latest.pdf`
[2]`https://ingmec.ual.es/~jlblanco/papers/jlblanco2010geometry3D_techrep.pdf`
[3]`https://github.com/nurlanov-zh/so3_log_map/blob/main/so3_log_map_analysis.ipynb`

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:

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

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

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

3.2.2 Jacobian of extended log map (around π)

```python
def Dx_log_x_pi(R):
    trR = R[0, 0] + R[1, 1] + R[2, 2]
    cos_theta = max(min(0.5 * (trR - 1), 1), -1)
    sin_theta = np.sqrt(1 - cos_theta * cos_theta)
    theta = np.arctan2(sin_theta, cos_theta)
    R_minus_R_T_vee = np.array([R[2, 1] - R[1, 2],
                                R[0, 2] - R[2, 0], R[1, 0] - R[0, 1]])

    if abs(3 - trR) < 1e-8:
        # return Jacobian of log map at Theta = 0
        return np.array([[0, 0, 0, 0, 0, -0.5, 0, 0.5, 0],
                         [0, 0, 0.5, 0, 0, 0, -0.5, 0, 0],
                         [0, -0.5, 0, 0.5, 0, 0, 0, 0, 0]])

    S = R + R.transpose() + (1 - trR) * np.eye(3)
    rest_tr = 3 - trR
    n = np.ones(3)
    # Fix modules of n_i
    for i in range(3):
        n[i] = np.sqrt(max(0, S[i, i] / rest_tr))
    max_i = np.argmax(n)

    # Fix signs according to the sign of max element
    for i in range(3):
        if i != max_i:
            n[i] *= np.sign(S[max_i, i])

    # Fix an overall sign
    if any(np.sign(n) * np.sign(R_minus_R_T_vee) < 0):
        n = -n

    dn_dR_11 = np.zeros(3)
    dn_dR_22 = np.zeros(3)
    dn_dR_33 = np.zeros(3)
    if abs(n[0]) > np.finfo(float).eps:
        dn_dR_11[0] = (S[0, 0] + rest_tr) / (2 * n[0] * rest_tr * rest_tr)
        dn_dR_22[0] = (S[0, 0] - rest_tr) / (2 * n[0] * rest_tr * rest_tr)
        dn_dR_33[0] = dn_dR_22[0]
    if abs(n[1]) > np.finfo(float).eps:
        dn_dR_11[1] = (S[1, 1] - rest_tr) / (2 * n[1] * rest_tr * rest_tr)
        dn_dR_22[1] = (S[1, 1] + rest_tr) / (2 * n[1] * rest_tr * rest_tr)
        dn_dR_33[1] = dn_dR_11[1]
    if abs(n[2]) > np.finfo(float).eps:
        dn_dR_11[2] = (S[2, 2] - rest_tr) / (2 * n[2] * rest_tr * rest_tr)
        dn_dR_22[2] = dn_dR_11[2]
        dn_dR_33[2] = (S[2, 2] + rest_tr) / (2 * n[2] * rest_tr * rest_tr)

    dtheta_d_R_11 = -sin_theta * 0.5 + cos_theta * 0.5 * (dn_dR_11.dot(R_minus_R_T_vee))
    dtheta_d_R_22 = -sin_theta * 0.5 + cos_theta * 0.5 * (dn_dR_22.dot(R_minus_R_T_vee))
    dtheta_d_R_33 = -sin_theta * 0.5 + cos_theta * 0.5 * (dn_dR_33.dot(R_minus_R_T_vee))

    dtheta_d_R_12 = cos_theta * 0.5 * (-n[2])
    dtheta_d_R_13 = cos_theta * 0.5 * (n[1])
    dtheta_d_R_23 = cos_theta * 0.5 * (-n[0])
    dtheta_d_R_21 = -dtheta_d_R_12
    dtheta_d_R_31 = -dtheta_d_R_13
    dtheta_d_R_32 = -dtheta_d_R_23

    J = np.zeros((3, 3, 3))

    J[0, 0, 0] = dtheta_d_R_11 * n[0] + theta * dn_dR_11[0]
    J[0, 1, 1] = dtheta_d_R_22 * n[0] + theta * dn_dR_22[0]
    J[0, 2, 2] = dtheta_d_R_33 * n[0] + theta * dn_dR_33[0]
    J[0, 0, 1] = dtheta_d_R_12 * n[0]
    J[0, 0, 2] = dtheta_d_R_13 * n[0]
    J[0, 1, 2] = dtheta_d_R_23 * n[0]
    J[0, 1, 0] = -J[0, 0, 1]
    J[0, 2, 0] = -J[0, 0, 2]
    J[0, 2, 1] = -J[0, 1, 2]

    J[1, 0, 0] = dtheta_d_R_11 * n[1] + theta * dn_dR_11[1]
    J[1, 1, 1] = dtheta_d_R_22 * n[1] + theta * dn_dR_22[1]
    J[1, 2, 2] = dtheta_d_R_33 * n[1] + theta * dn_dR_33[1]
    J[1, 0, 1] = dtheta_d_R_12 * n[1]
    J[1, 0, 2] = dtheta_d_R_13 * n[1]
    J[1, 1, 2] = dtheta_d_R_23 * n[1]
    J[1, 1, 0] = -J[1, 0, 1]
    J[1, 2, 0] = -J[1, 0, 2]
    J[1, 2, 1] = -J[1, 1, 2]

    J[2, 0, 0] = dtheta_d_R_11 * n[2] + theta * dn_dR_11[2]
    J[2, 1, 1] = dtheta_d_R_22 * n[2] + theta * dn_dR_22[2]
    J[2, 2, 2] = dtheta_d_R_33 * n[2] + theta * dn_dR_33[2]
    J[2, 0, 1] = dtheta_d_R_12 * n[2]
    J[2, 0, 2] = dtheta_d_R_13 * n[2]
    J[2, 1, 2] = dtheta_d_R_23 * n[2]
    J[2, 1, 0] = -J[2, 0, 1]
    J[2, 2, 0] = -J[2, 0, 2]
    J[2, 2, 1] = -J[2, 1, 2]

    return J.reshape(3, 9)
```

3.2.3 Jacobian of extended log map (quaternion)

Jacobian of Matrix to Quaternion map:

```
def Dquaternion_DR(R):
    """Computes d quaternion(R) / d R , 4 x 9 Jacobian."""
    J_quat = np.zeros((4, 3, 3))
    t = R[0, 0] + R[1, 1] + R[2, 2]
    if t > 0:
        # case 1
        isqrt_t = 1 / np.sqrt(1 + t)
        J_quat[0, :, :] = 0.25 * isqrt_t * np.eye(3)
        J_quat[1, :, :] = -0.25 * isqrt_t / (1 + t) * (R[2, 1] - R[1, 2]) * np.eye(3)
        J_quat[1, 2, 1] = 0.5 * isqrt_t
        J_quat[1, 1, 2] = -0.5 * isqrt_t
        J_quat[2, :, :] = -0.25 * isqrt_t / (1 + t) * (R[0, 2] - R[2, 0]) * np.eye(3)
        J_quat[2, 0, 2] = 0.5 * isqrt_t
        J_quat[2, 2, 0] = -0.5 * isqrt_t
        J_quat[3, :, :] = -0.25 * isqrt_t / (1 + t) * (R[1, 0] - R[0, 1]) * np.eye(3)
        J_quat[3, 1, 0] = 0.5 * isqrt_t
        J_quat[3, 0, 1] = -0.5 * isqrt_t
    else:
        i = 0
        if R[1, 1] > R[0, 0]:
            i = 1
        if R[2, 2] > R[i, i]:
            i = 2
        j = (i + 1) % 3
        k = (j + 1) % 3
        r = np.sqrt(R[i, i] - R[j, j] - R[k, k] + 1)
        i_r = 1 / r
        i_r_cube = 1 / ((R[i, i] - R[j, j] - R[k, k] + 1) * r)
        r_eye = np.eye(3)
        r_eye[j, j] = -1
        r_eye[k, k] = -1
        J_quat[1 + i, :, :] = 0.25 * i_r * r_eye
```

```python

        J_quat[0, :, :] = -0.25 * (R[k, j] - R[j, k]) * i_r_cube * r_eye
        J_quat[0, k, j] = 0.5 * i_r
        J_quat[0, j, k] = -0.5 * i_r
        J_quat[1+j, :, :] = -0.25 * (R[j, i] + R[i, j]) * i_r_cube * r_eye
        J_quat[1+j, j, i] = 0.5 * i_r
        J_quat[1+j, i, j] = 0.5 * i_r
        J_quat[1+k, :, :] = -0.25 * (R[k, i] + R[i, k]) * i_r_cube * r_eye
        J_quat[1+k, k, i] = 0.5 * i_r
        J_quat[1+k, i, k] = 0.5 * i_r
    return J_quat.reshape((4, 9))
```

Jacobian of Quaternion to Axis-angle map:

```python
def Dlog_Dquaternion2(q):
    """Computes d log(q) / d q , 3 x 4 Jacobian."""
    J_vec = np.zeros((3, 4))
    squared_n = q[1]*q[1] + q[2]*q[2] + q[3]*q[3]
    n = np.sqrt(squared_n)
    w = q[0]
    squared_w = w*w
    sign = 1
    if w < 0:
        sign = -1
        w = -w
    if squared_n < np.finfo(float).eps * np.finfo(float).eps:
        # n (~Theta) close to 0
        two_atan_nbyw_by_n = 2 / w - 2.0 / 3 * squared_n / (w * squared_w)
        d_q0 = -2 / squared_w + 2 * squared_n / (squared_w * squared_w)
        d_q1 = -4.0 / 3 * q[1] / (w * squared_w)
        d_q2 = -4.0 / 3 * q[2] / (w * squared_w)
        d_q3 = -4.0 / 3 * q[3] / (w * squared_w)
    else:
        # Regular case
        two_atan_nbyw_by_n = 4 * np.arctan(n / (w + np.sqrt(squared_w + squared_n)))
 / n
        d_q0 = -2 / (squared_w + squared_n)
        c0 = (2 * w - two_atan_nbyw_by_n) / squared_n
        d_q1 = c0 * q[1]
        d_q2 = c0 * q[2]
        d_q3 = c0 * q[3]

    J_vec[:, 0] = sign * d_q0 * q[1:]
    J_vec[:, 1] = d_q1 * q[1:]
    J_vec[0, 1] += two_atan_nbyw_by_n
    J_vec[:, 2] = d_q2 * q[1:]
    J_vec[1, 2] += two_atan_nbyw_by_n
    J_vec[:, 3] = d_q3 * q[1:]
    J_vec[2, 3] += two_atan_nbyw_by_n
    J_vec = J_vec * sign
    return J_vec
```

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

```python
def Dx_log_x_quaternion(R):
    """Computes d log(R) / d R , 3 x 9 Jacobian."""
    J_quat = Dquaternion_DR(R)
    q = SO3.matrix2quaternion(R)
    J_vec = Dlog_Dquaternion2(q)
    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 even for rotations with non-edge angles.

2. However, after matrix multiplication with Jacobian of the box plus at zero the values of the product become the same.

3. The observation from previous points is violated when the extension is non-differentiable. For example, the Jacobian around π nullifies rows corresponding to zero components of the rotation axis, i.e. $n_i = 0 \rightsquigarrow J^{\pi}[i,:] = 0$), because the extension around π contains $\text{sign}(x)$ function which is non-differentiable at zero.

4. For small angles θ close to 0: the Jacobian around π diverges (tends to infinity).

5. For large angles θ close to π: the baseline Jacobian diverges.

6. The Jacobian through intermediate quaternion representation does not diverge anywhere. Overall, the map through quaternion is the only one that we should use to get correct results for log map value, autodiff jacobians, and analytical jacobians.

4 Theoretical result

Proposition 1: Given the functions

$$h : X \to Z, g : X \to Y, f : Y \to Z, \quad h(x) = f(g(x)), \quad \forall x \in X$$

$$X \subset \mathbb{R}^n, Y \subset \mathbb{R}^m, Z \subset \mathbb{R}^k$$

$$h(x) = f(g(x)), \quad \forall x \in X \subset \mathbb{R}^n$$

Assume that functions g and h are differentiable on X, i.e.:

$$\exists J_g(a) \in \mathbb{R}^{m\times n} : g(a+\epsilon) - g(a) = J_g(a)\epsilon + \eta(\epsilon)\cdot\epsilon, \quad \lim_{||\epsilon||\to 0} ||\eta(\epsilon)|| = 0, \epsilon \in \mathbb{R}^n, \eta(\epsilon) \in \mathbb{R}^{m\times n}$$

$$\exists J_h(a) \in \mathbb{R}^{k\times n} : h(a+\epsilon) - h(a) = J_h(a)\epsilon + \mu(\epsilon)\cdot\epsilon, \quad \lim_{||\epsilon||\to 0} ||\mu(\epsilon)|| = 0, \epsilon \in \mathbb{R}^n, \mu(\epsilon) \in \mathbb{R}^{k\times n}$$

$$\text{for all } a \in X \subset \mathbb{R}^n, \text{ and } \epsilon \text{ close to the origin in } \mathbb{R}^n$$

The second function of composition $f : Y \to Z$ is defined on Y - manifold (curve or sphere). We can extend f onto open superset of Y, i.e. $\overline{Y} \supset Y, \overline{Y} \subset \mathbb{R}^m$, such that extension $\overline{f}$ is differentiable at every point of Y. Let's assume we have 2 differentiable extentions with different Jacobians:

$$\exists f_1 : \overline{Y} \to Z, f_1|_Y = f,$$

$$f_1(g(a)+\delta) - f(g(a)) = J_{f_1}(g(a))\delta + \nu_1(\delta)\delta, \quad \lim_{||\delta||\to 0} ||\nu_1(\delta)|| = 0$$

$$\exists f_2 : \overline{Y} \to Z, f_2|_Y = f,$$
$$f_2(g(a)+\delta) - f(g(a)) = J_{f_2}(g(a))\delta + \nu_2(\delta)\delta, \qquad \lim_{||\delta||\to 0} ||\nu_2(\delta)|| = 0$$
$$J_{f_1}(g(a)) \neq J_{f_2}(g(a))$$

Then, the result of the chain rule does not depend on the extension as long as the extension is differentiable, i.e.:

$$J_{f_1}(g(a))J_g(a) = J_{f_2}(g(a))J_g(a) = J_h(a)$$

Proof:

$$\begin{aligned}
h(a+\epsilon) - h(a) &\overset{\text{diff. of } h}{=} J_h(a)\epsilon + \mu(\epsilon)\epsilon = \\
&\overset{\text{def composite}}{=} f(g(a+\epsilon)) - f(g(a)) = \\
&\overset{\text{diff. of } g}{=} f(g(a) + J_g(a)\epsilon + \eta(\epsilon)\epsilon) - f(g(a)) = (*) \\
(*) &\overset{\text{first extension}}{=} \overline{f}_1(g(a)+\delta_\epsilon) - f(g(a)) = \\
&= J_{f_1}(g(a))\delta_\epsilon + \nu_1(\delta_\epsilon)\delta_\epsilon = \\
&\overset{\delta_\epsilon = J_g(a)\epsilon + \eta(\epsilon)\epsilon}{=} J_{f_1}(g(a))J_g(a)\epsilon + [J_{f_1}(g(a))\eta(\epsilon) + \nu_1(\delta_\epsilon)J_g(a) + \nu_1(\delta_\epsilon)\eta(\epsilon)]\epsilon = \\
&\overset{[\ldots]=o_1(\epsilon)\to 0}{=} J_{f_1}(g(a))J_g(a)\epsilon + o_1(\epsilon)\epsilon, \\
(*) &\overset{\text{second extension}}{=} \overline{f}_2(g(a)+\delta_\epsilon) - f(g(a)) = \\
&= J_{f_2}(g(a))\delta_\epsilon + \nu_2(\delta_\epsilon)\delta_\epsilon = \\
&\overset{\delta_\epsilon = J_g(a)\epsilon + \eta(\epsilon)\epsilon}{=} J_{f_2}(g(a))J_g(a)\epsilon + [J_{f_2}(g(a))\eta(\epsilon) + \nu_2(\delta_\epsilon)J_g(a) + \nu_2(\delta_\epsilon)\eta(\epsilon)]\epsilon = \\
&\overset{[\ldots]=o_2(\epsilon)\to 0}{=} J_{f_2}(g(a))J_g(a)\epsilon + o_2(\epsilon)\epsilon.
\end{aligned}$$

So at the end we have:

$$J_h(a)\epsilon + \mu(\epsilon)\epsilon = J_{f_1}(g(a))J_g(a)\epsilon + o_1(\epsilon)\epsilon = J_{f_2}(g(a))J_g(a)\epsilon + o_2(\epsilon)\epsilon,$$
$$\text{where } \lim_{||\epsilon||\to 0} ||o_1(\epsilon)|| = 0, \lim_{||\epsilon||\to 0} ||o_2(\epsilon)|| = 0$$

Since the equality holds for arbitrary ϵ around origin in $\mathbb{R}^n$, then the linear coefficients are equal, i.e.

$$J_{f_1}(g(a))J_g(a) = J_{f_2}(g(a))J_g(a) = J_h(a)$$

It ends the proof.

References

[Blanco, 2010] Blanco, J.-L. (2010). A tutorial on se (3) transformation parameterizations and on-manifold optimization. *University of Malaga, Tech. Rep*, 3:6.

[Gallego and Yezzi, 2013] Gallego, G. and Yezzi, A. J. (2013). A compact formula for the derivative of a 3-d rotation in exponential coordinates. *CoRR*, abs/1312.0788.