3.a) Lie Algebra Structure:

Question:
With $\xi = (v_x, v_y, \omega)^T = (\mathbf{v}^T, \omega)^T$, the wedge map is:

$$\xi^\wedge = \begin{bmatrix} \omega^\wedge & \mathbf{v} \\ \mathbf{0}^T & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\omega & v_x \\ \omega & 0 & v_y \\ 0 & 0 & 0 \end{bmatrix}$$

- Verify this is block upper-triangular (rotation block in upper-left, translation in upper-right).
- Implement `se2_wedge(xi)` and `se2_vee(Xi)`.

Solution:
Code snippet 1 shows the implementation of `se2_wedge(xi)` and `se2_wedge(Xi)`

3.b) Exponential Map:

Question:
The closed-form expression is:

$$\mathrm{Exp}(()\xi^\wedge) = \begin{bmatrix} R(\omega) & V(\omega)\mathbf{v} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

where $V(\omega) = \frac{\sin\omega}{\omega} I + \frac{1-\cos\omega}{\omega} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$.

- Derive by hand using the matrix exponential power series (show key steps).
- Implement `se2_exp(xi)` with Taylor series for $\omega \approx 0$ (use 5 terms).

Solution:
Code snippet 1 shows the implementation of `se2_exp(xi)`

3.c) Logarithm Map:

Question:
The inverse of $V(\omega)$ is:

$$V(\omega)^{-1} = \frac{\omega}{2} \cot\frac{\omega}{2} I + \frac{\omega}{2} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

- Implement `se2_log(X)`. Use Taylor series for $\omega \approx 0$ (use 5 terms).
- Verify numerically: $\mathrm{Exp}(()\mathrm{Log}(()X)) = X$ for random $X \in \mathrm{SE}(()2)$.

Solution:
Code snippet 1 shows the implementation of `se2_log(X)`.
Code snippet 2 shows the implementation of the unit test verifying $\mathrm{Exp}(()\mathrm{Log}(()X)) = X$ for random $X \in \mathrm{SE}(()2)$ named `test_se2_exp_log()` and Fig 1 shows the test passing.

3.d) Adjoint Representation Ad_X:

Question:
The adjoint $\mathrm{Ad}_X : \mathfrak{se}(2) \to \mathfrak{se}(2)$ is defined by $(\mathrm{Ad}_X \xi)^\wedge = X\xi^\wedge X^{-1}$. *Note: You don't need to multiply out 2×2 blocks explicitly. Showing block form (e.g., $R\omega^\wedge R^T$, $R\mathbf{v}$) is sufficient.*

Translation-first ordering $\xi = (v_x, v_y, \omega)^T$:

- Compute $X\xi^\wedge X^{-1}$ explicitly for $X = (\mathbf{t}, R)$.

- Extract the 3×3 matrix Ad_X such that $\mathrm{Ad}_X \xi$ gives the transformed twist.

  You should get: $\mathrm{Ad}_X = \begin{bmatrix} R & \mathbf{t}^{\odot} \\ \mathbf{0}^T & 1 \end{bmatrix}$ where $\mathbf{t}^{\odot} = \begin{bmatrix} t_y \\ -t_x \end{bmatrix} = -J\mathbf{t}$ with $J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$.

Rotation-first ordering $\xi = (\omega, v_x, v_y)^T$:

- Repeat the derivation with this ordering. The wedge map produces the same 3×3 matrix, but extracting Ad_X requires matching to the new vector ordering.
- You should get: $\mathrm{Ad}_X = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{t}^{\odot} & R \end{bmatrix}$ (block *lower*-triangular).

Verification:

- Verify numerically: $\mathrm{Ad}_{X_1 X_2} = \mathrm{Ad}_{X_1} \mathrm{Ad}_{X_2}$ for random $X_1, X_2 \in \mathrm{SE}(()2)$.
- Verify numerically: $\mathrm{Ad}_X^{-1} = \mathrm{Ad}_{X^{-1}}$ (the adjoint respects inverses).
- **Connection to Problem 2:** Confirm that R acts on velocity components, as you predicted from the normal subgroup structure.

Solution:
Code snippet 2 shows the implementation of the unit test verifying $\mathrm{Ad}_{X_1 X_2} = \mathrm{Ad}_{X_1} \mathrm{Ad}_{X_2}$ for random $X_1, X_2 \in \mathrm{SE}(()2)$ for random $X \in \mathrm{SE}(()2)$ named `test_se2_Ad_composition` and Fig 1 shows the test passing. Code snippet 2 shows the implementation of the unit test verifying $\mathrm{Ad}_X^{-1} = \mathrm{Ad}_{X^{-1}}$ for random $X \in \mathrm{SE}(()2)$ named `test_se2_Ad_inv()` and Fig 1 shows the test passing.

3.e) Lie Bracket and ad_ξ:

Question:
The Lie bracket on vectors is defined by $[\xi_1, \xi_2]^\wedge := \xi_1^\wedge \xi_2^\wedge - \xi_2^\wedge \xi_1^\wedge$ (i.e., compute the matrix commutator, then apply vee). The small adjoint ad_ξ is the 3×3 matrix such that $[\xi_1, \xi_2] = \mathrm{ad}_{\xi_1} \xi_2$.

Translation-first ordering $\xi = (v_x, v_y, \omega)^T$:

- Compute $[\xi_1^\wedge, \xi_2^\wedge]$ for $\xi_i = (v_{ix}, v_{iy}, \omega_i)^T$. Extract the result as a vector.
- Derive by hand the 3×3 matrix ad_ξ.

  You should get: $\mathrm{ad}_\xi = \begin{bmatrix} \omega^\wedge & \mathbf{v}^{\odot} \\ \mathbf{0}^T & 0 \end{bmatrix}$ where $\mathbf{v}^{\odot} = \begin{bmatrix} v_y \\ -v_x \end{bmatrix} = -J\mathbf{v}$.

Solution:

Implementation and Unit Tests

Code Snippet 1: SE(2) and $\mathfrak{se}(2)$ Function Implementations

```python
import numpy as np
from Code.SO2.SO2_maps import so2_wedge, so2_vee, so2_exp, so2_log

def se2_compose(X1, X2):
    """
    Composition of elements of SE(2)

    :param X1: element of SE(2)
    :param X2: element of SE(2)
    """
    X1_t = X1[0:2, 2].reshape((2, 1))
    X1_R = X1[0:2, 0:2]

    X2_t = X2[0:2, 2].reshape((2, 1))
```

```
    X2_R = X2[0:2, 0:2]

    X12_R = X1_R @ X2_R
    X12_t = X1_R @ X2_t + X1_t

    X12 = np.block([[X12_R, X12_t],
                    [np.zeros((1, 2)), 1]])

    return X12

def se2_inverse(X):
    """
    Inverse of SE(2) element

    :param X: element of SE(2)
    """
    X_t = X[0:2, 2].reshape((2, 1))
    X_R = X[0:2, 0:2]

    X_inv_t = -X_R.T @ X_t
    X_inv_R = X_R.T

    X_inv = np.block([[X_inv_R, X_inv_t],
                      [np.zeros((1, 2)), 1]])

    return X_inv

def se2_wedge(xi):
    """
    Maps from se2 real number parametrization to se2 Lie algebra

    :param xi: twist vector (v_x, v_y, omega) in R3
    """
    xiwedge = np.block([[so2_wedge(xi[2]), xi[0:2].reshape((2, 1))],
                        [np.zeros((1, 2)),        0]])

    return xiwedge

def se2_vee(xiwedge):
    """
    Maps from se2 Lie algebra to its real number parametrization

    :param xiwedge: 3x3 se2 Lie algebra element
    """

    xi = np.block([xiwedge[[0, 1], [2, 2]], so2_vee(xiwedge[0:2, 0:2])]);

    return xi

def se2_exp(xi):
    """
    Maps from parametrization of se2 Lie algebra to SE2 Lie group

    :param xi: twist vector (v_x, v_y, omega) in R3
    """
    # Extract elements
    v = xi[0:2]
    omega = xi[2]

    # SO2 exponential map
    R = so2_exp(omega)

    # Translation-orientation coupling
    if abs(omega) > 1e-8:
        V = np.sin(omega) / omega * np.eye(2) + (1 - np.cos(omega)) / omega * np.array([[0,
    -1], [1, 0]])
    else:
        V = ((1 - omega ** 2 / 6 + omega ** 4 / 120) * np.eye(2)
```

```
           + (omega / 2 - omega ** 3 / 24) * np.array([[0, -1], [1, 0]]))

    X = np.block([[R,    V @ v.reshape((2, 1))],
                  [np.zeros((1, 2)),     1]])

    return X

def se2_log(xiwedge):
   """
   Maps from SE2 Lie group to the parametrization of its Lie algebra se2

   :param xiwedge: 3x3 se2 Lie algebra element
   """
   omega = so2_log(xiwedge[0:2, 0:2])

   if abs(omega) > 1e-8:
       V_inv = omega / 2 * (np.cos(omega / 2) / np.sin(omega / 2) * np.eye(2) + np.array([[0, 1], [-1, 0]]))
   else:
       V_inv = 1 / 2 * ((2 - omega ** 2 / 6 - omega ** 4 / 360 - 2 * omega ** 6 / 30240 + omega ** 8 / 604800) * np.eye(2) + omega * np.array([[0, -1], [1, 0]]))

   v = V_inv @ xiwedge[0:2, 2]

   xi = np.vstack((v.reshape((2, 1)), np.reshape(omega, (1,1))))

   return xi

def se2_Ad(X):
    """
    Maps from se2 to se2 defined by (Ad_X xi)**wedge = X xi**wedge X**-1
    Shifts tangent spaces

    :param X: SE2 Lie group element
    """
    Ad = np.copy(X)
    Ad[0:2, 2] = -np.array([[0, -1], [1, 0]]) @ X[0:2, 2]

    return Ad

def se2_ad(xi):
    """
    Derivative of Adjoint at identity

    :param xi: se(2) Lie algebra element parametrization
    """
    ad = se2_wedge(xi)
    ad[0:2, 2] = -np.array([[0, -1], [1, 0]]) @ ad[0:2, 2]

    return ad
```

Code Snippet 2: Unit Tests for SE(2) Using Pytest

```
import pytest as pt
import numpy as np
from Code.SE2.SE2_maps import se2_compose, se2_inverse, se2_exp, se2_log, se2_vee, se2_wedge, se2_Ad, se2_ad

def test_se2_exp_log():
    """
    Test that exp(log(X)) = X for random SE2 elements
    """
    n_samples = 100

    rng = np.random.default_rng()
    xi_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))

    checks = np.zeros([n_samples, 1])
```

```
    for i in range(n_samples):
        X_random = se2_exp(xi_random[:, i])

        X_log = se2_log(X_random)
        X_tilde = se2_exp(X_log)

        checks[i] = np.all(X_tilde == pt.approx(X_random))

    assert np.all(checks)

def test_se2_Ad():
    """
    Test that Ad_X(xi) = (X xi^wedge X^-1)^vee for random SE2, se2 elements
    """
    n_samples = 100

    rng = np.random.default_rng()
    xi_X_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))
    xi_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))

    checks = np.zeros([n_samples, 1])
    for i in range(n_samples):
        X_random = se2_exp(xi_X_random[:, i])

        xi_prime = se2_vee(X_random @ se2_wedge(xi_random[:, i]) @ se2_inverse(X_random))
        xi_prime_ad = se2_Ad(X_random) @ xi_random[:, i]

        checks[i] = np.all(xi_prime_ad == pt.approx(xi_prime))

    assert np.all(checks)

def test_se2_Ad_composition():
    """
    Test that Ad_{X_1 X_2} = Ad_{X_1}Ad_{X_2} for random SE2 elements
    """
    n_samples = 100

    rng = np.random.default_rng()
    xi1_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))
    xi2_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))

    checks = np.zeros([n_samples, 1])
    for i in range(n_samples):
        X1_random = se2_exp(xi1_random[:, i])
        X2_random = se2_exp(xi2_random[:, i])

        Ad_X1X2 = se2_Ad(se2_compose(X1_random, X2_random))
        Ad_X1_Ad_X2 = se2_Ad(X1_random) @ se2_Ad(X2_random)

        checks[i] = np.all(Ad_X1X2 == pt.approx(Ad_X1_Ad_X2))

    assert np.all(checks)

def test_se2_Ad_inv():
    """
    Test that (Ad_X)^-1 = Ad_{X^-1} for random SE2 elements
    """
    n_samples = 100

    rng = np.random.default_rng()
    xi_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))

    checks = np.zeros([n_samples, 1])
    for i in range(n_samples):
        X_random = se2_exp(xi_random[:, i])

        Ad_X_inv = np.linalg.inv(se2_Ad(X_random))
        Ad_Xinv = se2_Ad(se2_inverse(X_random))
```

```python

        a = se2_Ad(X_random) @ Ad_X_inv
        b = se2_Ad(X_random) @ Ad_Xinv

        checks[i] = np.all(Ad_X_inv == pt.approx(Ad_Xinv))

    assert np.all(checks)

def test_se2_ad():
    """
    Test that ad_{xi_1}(xi_2) = [xi_1, xi_2] for random se2 elements
    """
    n_samples = 100

    rng = np.random.default_rng()
    xi1_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))
    xi2_random = np.reshape(rng.uniform(-10, 10, 3 * n_samples), (3, n_samples))

    checks = np.zeros([n_samples, 1])
    for i in range(n_samples):
        Lie_bracket = se2_vee(se2_wedge(xi1_random[:, i]) @ se2_wedge(xi2_random[:, i])
                              - se2_wedge(xi2_random[:, i]) @ se2_wedge(xi1_random[:, i]))
        Lie_bracket_ad = se2_ad(xi1_random[:, i]) @ xi2_random[:, i]

        checks[i] = np.all(Lie_bracket_ad == pt.approx(Lie_bracket))

    assert np.all(checks)

test_se2_Ad_inv()
```

Figure 1: Pytest unit test output showing tests passing

```
=================================== test session starts ===================================
platform win32 -- Python 3.10.11, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\thatf\OneDrive\Documents\Purdue Classes\AAE 590LGM\Lie Group Methods
collected 5 items

Tests\SE2\test_SE2.py .....                                                         [100%]

=================================== 5 passed in 0.56s ====================================
```