```python
# Author: Ryan Wu (ID: weihuanw)
# Carnegie Mellon University
# 24-677 Special Topics: Modern Control - Theory and Design
# Project: Part 2 Exercise 1
# Description: determine controllability and observability of the given system and generate plots
# Due: 11/09/2023 11:59 PM

# import the required libraries
import numpy as np
import matplotlib.pyplot as plt
import control as ctrl

# declaring given variables
Ca = 20000  # Newton
m = 1888.6  # kg
Iz = 25854  # kgm^2
lr = 1.39  # m
lf = 1.55  # m

# given longitudinal velocities for analysis [m/s]
velocities = [2, 5, 8]

# --  Exercise 1.1: check controllability (P) and observability (Q) with velocities [2, 5, 8] m/s -- #

# iterate through each velocity
for velocity in velocities:
    xdot = velocity
    # define the state-space matrices
    A = np.array([[0, 1, 0, 0], [0, -4 * Ca / (m * xdot), 4 * Ca / m, (-2 * Ca * (lf - lr)) / (m * xdot)], [0, 0, 0, 1], [0, (-2 * Ca * (lf - lr)) / (Iz * xdot), (2 * Ca * (lf - lr)) / Iz, (-2 * Ca * (lf ** 2 + lr ** 2)) / (Iz * xdot)]])
    B = np.array([[0], [2 * Ca / m], [0], [2 * Ca * lf / Iz]])
    C = np.identity(4)
```

```
    D = 0

    # create the state-space model
    sys = ctrl.StateSpace(A, B, C, D)
    # print(sys) # for debugging

    # calculate the rank of the controllability and observability matrices
    P = np.linalg.matrix_rank(ctrl.ctrb(sys.A, sys.B))
    Q = np.linalg.matrix_rank(ctrl.obsv(sys.A, sys.C))
    # check the rank of the controllability and observability matrices
    controllable = P == sys.A.shape[0]
    observable = Q == sys.A.shape[0]

    # print and show the results
    print(f"At {velocity} m/s:")
    print(f"Rank of controllability matrix P: {P}, Controllable: {'Yes' if controllable else 'No'}")
    print(f"Rank of observability matrix Q: {Q}, Observable: {'Yes' if observable else 'No'}")
    print("=" * 55)

# -- Exercise 1.2: plot the log(sigma) vs velocity & real parts vs velocity -- #

# initialize sigma1_values abd real_parts
sigma1_values = []
real_parts = []

# iterate through each velocity
for velocity in range(1, 41):
    xdot = velocity
    # define the state-space matrices
    A = np.array([[0, 1, 0, 0],
                  [0, -4 * Ca / (m * xdot), 4 * Ca / m, (-2 * Ca * (lf - lr)) / (m * xdot)],
                  [0, 0, 0, 1],
                  [0, (-2 * Ca * (lf - lr)) / (Iz
```

```python
 * xdot), (2 * Ca * (lf - lr)) / Iz, (-2 * Ca * (lf ** 2 + lr ** 2)) / (Iz * xdot)]])

    B = np.array([[0], [2 * Ca / m], [0], [2 * Ca * lf / Iz]])

    C = np.identity(4)
    D = 0

    # create the state-space model
    sys = ctrl.StateSpace(A, B, C, D)

    # calculate the logarithm of the greatest singular value over the least singular value
    singular_values = np.linalg.svd(ctrl.ctrb(sys.A, sys.B), compute_uv=False)
    sigma1_values.append(np.log10(np.max(singular_values) / np.min(singular_values)))

    # calculate the poles (real parts)
    poles = np.linalg.eigvals(sys.A)
    real_parts.append([np.real(p) for p in poles])

# plot log(sigma) vs velocity
plt.figure(figsize=(12, 8))
plt.grid(True)
plt.plot(range(1, 41)[:len(sigma1_values)], sigma1_values, linewidth=2.5)
plt.title("$\log_{10}$ $\dfrac{\sigma_1}{\sigma_n}$ vs Longitudinal Velocity")
plt.xlabel("Longitudinal Velocity [m/s]")
plt.xlim(0, 40)
plt.ylabel("$\log_{10}$ $\dfrac{\sigma_1}{\sigma_n}$")
plt.ylim(0, 7 + 1)
plt.savefig("log(sigma) vs velocity.png")
plt.show()

# plot real parts vs velocity
plt.figure(figsize=(12, 8))
for i in range(4):
```

```
    plt.subplot(2, 2, i+1)
    plt.plot(range(1, 41), [p[i] for p in real_parts], linewidth=2.5)
    plt.grid(True)
    plt.title(f'Re(Pole {i+1}) vs Longitudinal Velocity')
    plt.xlabel('Longitudinal Velocity [m/s]')
    plt.xlim(0, 40)
    plt.ylabel(f'Re(Pole {i + 1})')

plt.tight_layout()
plt.savefig("real parts vs velocity.png")
plt.show()
```