```
Exercise 2:
class RK4(Integrator): # Runge-Kutta 4th order
    def step(self, t, x, u):
        k1 = self.f(t, x, u)
        k2 = self.f(t + 0.5*self.dt, x + 0.5*self.dt*k1, u)
        k3 = self.f(t + 0.5*self.dt, x + 0.5*self.dt*k2, u)
        k4 = self.f(t + self.dt, x + self.dt*k3, u)
        return x + self.dt/6 * (k1 + 2*k2 + 2*k3 + k4)
```

```
Exercise 4:

# -*- coding: utf-8 -*-
"""
Created on Thu Feb  6 14:45:37 2025

@author: Adin Sacho-Tanzer & Gabriel Kret
The Propeller Heads
"""

import numpy as np
import matplotlib.pyplot as plt
import integrators_HW2 as intg
import Parameters_HW2 as p

#U is the input vector [f_x, f_y, f_z, l, m, n]
#u,v,w is the velocity in the body frame
#p,q,r is the angular velocity in the body frame
#p_n, p_e, p_d is the position in the NED frame
#phi, theta, psi is the euler angles


# define matrices and sub-equations


class rigid_body:
    def __init__(self, mass, J_xx, J_yy, J_zz, J_xz, S, b, c, S_prop, rho,
k_motor, k_T_p, k_Omega, e):
        self.mass = mass
        self.J_xx = J_xx
        self.J_yy = J_yy
        self.J_zz = J_zz
        self.J_xz = J_xz
        self.S = S
        self.b = b
        self.c = c
```

```python
        self.S_prop = S_prop
        self.rho = rho
        self.k_motor = k_motor
        self.k_T_p = k_T_p
        self.k_Omega = k_Omega
        self.e = e

    def x_dot(self, t, x, U):
        #state: x = [p_n, p_e, p_d, u, v, w, phi, theta, psi, p, q, r]
        #inputs: U = [f_x, f_y, f_z, l, m, n]
        f_x, f_y, f_z, l, m, n = U

        p_n = x[0]
        p_e = x[1]
        p_d = x[2]
        u = x[3]
        v = x[4]
        w = x[5]
        phi = x[6]
        theta = x[7]
        psi = x[8]
        p = x[9]
        q = x[10]
        r = x[11]


        #forces and moments
        A = np.array([[np.cos(theta)*np.cos(psi),
np.sin(phi)*np.sin(theta)*np.cos(psi)-np.cos(phi)*np.sin(psi),
np.cos(phi)*np.sin(theta)*np.cos(psi)+np.sin(phi)*np.sin(psi)],
             [np.cos(theta)*np.sin(psi),
np.sin(phi)*np.sin(theta)*np.sin(psi)+np.cos(phi)*np.cos(psi),
np.cos(phi)*np.sin(theta)*np.sin(psi)-np.sin(phi)*np.cos(psi)],
             [-np.sin(theta), np.sin(phi)*np.cos(theta),
np.cos(phi)*np.cos(theta)]])

        B = np.array([[r*v - q*w],
                    [p*w - r*u],
                    [q*u - p*v]])

        C = 1/self.mass * np.array([[f_x],
                                   [f_y],
                                   [f_z]])

        D = np.array([[1, np.sin(phi)*np.tan(theta), np.cos(phi)*np.tan(theta)],
```

```
                        [0, np.cos(phi), -np.sin(phi)],
                        [0, np.sin(phi)/np.cos(theta),
np.cos(phi)/np.cos(theta)]])

        T = self.J_xx*self.J_zz - self.J_xz**2
        T1 = self.J_xz*(self.J_xx - self.J_yy + self.J_zz)/T
        T2 = (self.J_zz*(self.J_zz - self.J_yy) + self.J_xz**2)/T
        T3 = self.J_zz/T
        T4 = (self.J_zz - self.J_xx)/T
        T5 = (self.J_xx - self.J_yy)/T
        T6 = self.J_xz/T
        T7 = (self.J_zz - self.J_xx)/T
        T8 = self.J_xx*(self.J_xx - self.J_yy + self.J_zz)/T

        E = np.array([[T1*p*q - T2*q*r],
                    [T5*p*r - T6*(p**2 - r**2)],
                    [T7*p*q - T1*q*r]])


        F = np.array([[T3*l + T4*n],
                    [1/self.J_yy*m],
                    [T4*l + T8*n]])

        p_dot = A @ np.array([u,v,w])
        V_dot = B + C
        Angle_dot = D @ np.array([p,q,r])
        Omega_dot = E + F

        p_dot = p_dot.flatten()
        V_dot = V_dot.flatten()
        Angle_dot = Angle_dot.flatten()
        Omega_dot = Omega_dot.flatten()

        return np.concatenate((p_dot, V_dot, Angle_dot, Omega_dot), axis = 0)

    def simulate(self, x0, U, t_start, t_stop, dt=0.1):
        # maybe make U a function later
        # maybe have gravity later
        rk4_integrator = intg.RK4(dt, self.x_dot)

        t_history = [t_start]
        x_rk4_history = [x0]

        x_rk4 = x0
        t = t_start
```

```
        while t < t_stop:
            x_rk4 = rk4_integrator.step(t, x_rk4, U)
            t += dt
            t_history.append(t)
            x_rk4_history.append(x_rk4)

        return t_history, x_rk4_history

myPlane = rigid_body(p.mass, p.J_xx, p.J_yy, p.J_zz, p.J_xz, p.S, p.b, p.c,
p.S_prop, p.rho, p.k_motor, p.k_T_p, p.k_Omega, p.e)
t, x = myPlane.simulate(np.array([0,0,0,0,0,0,0,0,0,0,0,0]),
np.array([p.mass*2,0,0,0,0,0]), 0, 1, 0.1)
        #state: x = [p_n, p_e, p_d, u, v, w, phi, theta, psi, p, q, r]
        #inputs: U = [f_x, f_y, f_z, l, m, n]
plt.figure(figsize=(10, 5))
plt.plot(t, x)
plt.legend(["p_n", "p_e", "p_d", "u", "v", "w", "phi", "theta", "psi", "p", "q",
"r"])
plt.title("State Variables vs Time -- Taxiing and Takeoff")
plt.show()
```