

Copilot

Absolutely, you can certainly decouple the effects of the different forces in the function `f`. This can make the code more modular and easier to understand. Here's how you can modify the function `f` to use the `gravity_effect`, `drag_effect`, and `drag_coef` functions:

```
import numpy as np

def gravity_effect(m: float, g: float) -> float:
    return m * g

def drag_effect(m: float, rho: float, S: float, velocity: np.array) -> np.array:
    return (1 / (2*m)) * rho * S * drag_coef(velocity) * np.linalg.norm(velocity) * velocity

def drag_coef(velocity: np.array) -> float:
    speed = np.linalg.norm(velocity)
    if speed <= 1.0:
        return 2.3
    else:
        return (5.6 / (speed ** 1.35))

def f(t, y, m, g, rho, S):
    x, y, vx, vy = y
    v = np.array([vx, vy])
    ax = -drag_effect(m, rho, S, v)[0]
    ay = gravity_effect(m, g) - drag_effect(m, rho, S, v)[1]
    return np.array([vx, vy, ax, ay])
```

In this Python function, `t` is the current time, `y` is the current state vector, `m` is the mass of the projectile, `g` is the acceleration due to gravity, `rho` is the air density, and `S` is the cross-sectional area of the projectile. The function returns the time derivatives of the state variables, which are used in the Runge-Kutta method to update the state vector.

Please note that this is a simplified model and actual projectile motion can be more complex due to factors like wind, spin, and variable air density. Also, the step size `h` may need to be adjusted based on the specifics of the problem to ensure the accuracy of the solution. Let me know if you have any other questions! I'm here to help. 😊