```python
import numpy as np
import matplotlib.pyplot as plt

# Weight Estimate Calculator for Fueled Propeller Aircraft

def RaymerMethod(W_naught,A,C):

    """Uses existing regression analyses to calculate the empty weight of the aircraft
        values of needed constants can be found in the Raymer textbook

        W_naught: Initial gross weight input in lbf
        A: Constant from Raymer tables
        C: Constant from Raymer tables

        returns: empty weight fraction of aircraft"""

    empty_weight_frac = A*(W_naught**C)

    return empty_weight_frac

def CruiseFraction(R, c, eta, lift_to_drag):


    """ Calculates the weight fraction of cruise stage based on needed range
        R: The range the aircraft will travel during segment in ft
        c: Specific fuel consumption of engine in ft^-1
        eta: Estimated efficiency of propeller
        lift_to_drag: Estimated using wetted aspect ratio

        returns: weight fraction of cruise segment"""

    weight_frac = np.exp( (-R*SfcUnitConverter(c)) / (eta * lift_to_drag))

    return weight_frac

def LoiterFraction(E, V, c, eta, lift_to_drag):

    """ Calculates the weight fraction of the loiter stage based on needed endurance
        E: Time aircraft will loiter during segment in seconds
        V: Expected velocity of craft during loiter period in ft/s
        c: Specific fuel consumption of engine in ft^-1
        eta: Estimated efficiency of propeller
        lift_to_drag: Estimated using wetted aspect ratio

        returns: weight fraction of loiter segment"""

    E_seconds = E*3600

    weight_frac = np.exp( (-E_seconds*V*SfcUnitConverter(c)) / (eta*lift_to_drag))

    return weight_frac

def NewWeight(crew_weight, payload_weight, fuel_weight_frac, empty_weight_frac):

    """Calculates a new guess value for the takeoff weight of the aircraft.
        crew_weight: weight of the crew in lbf
        payload_weight: weight of the payload in lbf
        fuel_weight_frac: calculated total fuel weight fraction
        empty_weight_fraction: calculated empty weight fraction"""

    weight = (crew_weight + payload_weight) / (1 - fuel_weight_frac - empty_weight_frac)

    return weight

def SfcUnitConverter(sfc):
    """Converts a specific fuel consumption from lbm/hp/hr to ft^-1
        bsfc: Brake Specific fuel consumption in lbf/hp/hr"""
```

```
    sfc_convert = sfc/(1980000)

    return sfc_convert

def HybridFuelFrac(h, segment_frac):
    """Takes the fuel weight fraction for a mission segment and adjusts it for a hybrid
aircraft
        h: percent hybridization
        segment_frac: The calculated fuel weight fraction for the flight segment

        returns: The fuel weight fraction for a flight segment for a hybrid aircraft"""

    hybrid_frac = segment_frac + (1 - segment_frac)*h

    return hybrid_frac


def BatteryWeightFrac(h, ef, eb, eta_th, eta_elec, fuel_frac):
    """Estimates the wieght of the batteries needed for a hybrid aircraft
        h: The degree of hybridization of the aircraft as a fraction. This is assumed to be
constant through the entire mission
        ef: Fuel specific energy in Wh/kg
        eb: battery specific energy in Wh/kg
        eta_th: Efficiency of converting fuel energy into power
        eta_elec: Efficiency of converting battery energy into power
        fuel_frac: The previously calculated fuel fraction of the hybrid aircraft.

        return: The battery weight fraction of the hybrid aircraft"""

    battery_frac = (ef/eb)*(eta_th/eta_elec)*(h /(1-h))*(fuel_frac)

    return battery_frac

def NewHybridWeight(crew_weight, payload_weight, fuel_weight_frac,
battery_weight_frac,empty_weight_frac):
    """Calculates a new guess value for the takeoff weight of the aircraft.
        crew_weight: weight of the crew in lbf
        payload_weight: weight of the payload in lbf
        fuel_weight_frac: calculated total fuel weight fraction
        battery_weight_frac: calculated weight fraction of the battery
        empty_weight_fraction: calculated empty weight fraction"""

    hybrid_weight = (crew_weight + payload_weight) / (1 - fuel_weight_frac - empty_weight_frac
- battery_weight_frac)

    return hybrid_weight



# Iterative Solver for Takeoff Weight
def solve_takeoff_weight(crew_weight, payload_weight, A, C, cruise_segments, loiter_segments,
custom_segments,
                         h, ef, eb, eta_th, eta_elec,
                         e = 1e-6, max_iter=1000):
    """
    Iteratively solve for the takeoff weight of the aircraft.

    Parameters:
    - crew_weight: Total crew weight (including baggage)
    - payload_weight: Total payload weight
    - A, C: Constants from Raymer tables for empty weight fraction
    - cruise_segments: List of tuples [(R, c, eta, lift_to_drag), ...] for cruise segments
    - loiter_segments: List of tuples [(E, V, c, eta, lift_to_drag), ...] for loiter segments
    - custom_segments: List of weight fractions [guess_1, guess_2, ...] for non-cruise/loiter
segments
    - h: degree of hybridization as a fraction maximum installed power of electric motor over
total power
```

```
    - ef: specific energy of hydrocarbon fuel in Wh/kg
    - eb: specific energy of battery in Wh/kg
    - eta_th: efficiency of converting fuel energy into power
    - eta_elec: efficiency of converting battery energy into power
    - tol: Convergence tolerance
    - max_iter: Maximum number of iterations

    Returns:
    - takeoff_weight: Converged takeoff weight
    - iterations: Number of iterations used
    """
    # Initial guess for takeoff weight
    takeoff_weight_guess = crew_weight + payload_weight + 1000  # Arbitrary guess
    iterations = 0

    while iterations < max_iter:
        # Step 1: Calculate the empty weight fraction
        empty_weight_frac = RaymerMethod(takeoff_weight_guess, A, C)

        # Step 2: Calculate the total fuel weight fraction
        segment_frac = 1.0  # Start with a neutral multiplier

        # Cruise segments
        for R, c, eta, lift_to_drag in cruise_segments:
            cruise_frac = CruiseFraction(R, c, eta, lift_to_drag)
            segment_frac *= HybridFuelFrac(h, cruise_frac)

        loiter_fractions = []
        # Loiter segments
        for E, V, c, eta, lift_to_drag in loiter_segments:
            loiter_seg = LoiterFraction(E, V, c, eta, lift_to_drag)
            loiter_seg_hyb = HybridFuelFrac(h, loiter_seg)
            loiter_fractions.append(1 - loiter_seg_hyb)
            segment_frac *= loiter_seg_hyb


        # Custom segments (These values come directly metabook/textbook)
        for frac in custom_segments:
            misc_segment = HybridFuelFrac(h,frac)
            segment_frac *= misc_segment

        ######## Trying to calculate loiter fuel weight fraction #########

        reserves_adjusted = 1.06 - sum(loiter_fractions)  # Subtract loiter fuel fractions
from reserves

        # Ensure reserves_adjusted does not go below 1.0 (no negative reserves)
        reserves_adjusted = max(reserves_adjusted, 1.0)

        fuel_weight_frac = (1 - segment_frac) * reserves_adjusted  # Adjust fuel weight
fraction

        # Step 2a: Solve for the battery weight fraction
        battery_frac = BatteryWeightFrac(h, ef, eb, eta_th, eta_elec, fuel_weight_frac)

        # Step 3: Solve for the new takeoff weight
        new_takeoff_weight = NewHybridWeight(crew_weight, payload_weight, fuel_weight_frac,
battery_frac, empty_weight_frac)

        # Step 4: Calculate error
        error = abs(new_takeoff_weight - takeoff_weight_guess) / new_takeoff_weight

        # Step 5: Check convergence
        if 2*error < e:
            # Calculate the actual weights
            empty_weight = empty_weight_frac * new_takeoff_weight
            fuel_weight = fuel_weight_frac * new_takeoff_weight
```

```
            battery_weight = battery_frac * new_takeoff_weight
            return (new_takeoff_weight, empty_weight_frac, fuel_weight_frac, battery_frac,
                    empty_weight, fuel_weight, battery_weight, iterations)

        # Update guess and iterate
        takeoff_weight_guess = new_takeoff_weight
        iterations += 1

    # If the loop completes without convergence
    raise ValueError(f"Did not converge within {max_iter} iterations. Last error:
{error:.6f}")

############################### USE CODE STARTING HERE ###########################
crew_weight = 190  # lbf
payload_weight = 2000  # lbf
A = 0.74              # Raymer constant for "Agricultural aircraft" (Table 3.1)
C = -0.03             # Raymer constant for "Agricultural aircraft" (Table 3.1)
H = 0             # Degree hybridization in terms of maximum power of electric motor over
total aircraft power
e_f = 12800          # Fuel specific energy in Wh/kg (using Jet-A)
e_b = 260            # battery specific energy in Wh/kg
eta_prop = 0.8        # efficiency of propeller
eta_th = 0.30         # Thermal efficiency of converting fuel into power
eta_elec = 0.85       # Efficiency of converting battery power into power for the electric
motor
tol = 1e-6            # Convergence tolerance

# Define flight segments
cruise_segments = [
    (151903, 0.6, 0.82, 7.56),  # Range in ft, specific fuel consumption in lbm/hp/hr,
propeller efficiency, L/D
    (1033065, 0.6, 0.82, 7.56),
    (151903, 0.6, 0.82, 7.56),
]
loiter_segments = [
    (0.5, 180, 0.6, 0.72, 10.58),  # Endurance in hrs, velocity in ft/s, specific fuel
consumption lbm/hp/hr, propeller efficiency, L/D
]
other_segments = [0.996,0.995,0.996,0.998,0.999,0.998,0.999,0.998]  # Estimated weight
fractions for non-cruise/loiter phases


# Define the range of hybridization ratios (0 to 1)
hybrid_ratios = np.linspace(0, 1, 100)  # 100 points from 0 to 1

# Define battery specific energy values from 0.20 kWh/kg to 1.20 kWh/kg
battery_energies = np.arange(250, 1300, 150)  # Convert kWh/kg to Wh/kg

# Store results for plotting
fuel_weights = {e_b: [] for e_b in battery_energies}
valid_hybrid_ratios = {e_b: [] for e_b in battery_energies}

# Loop over battery specific energy values
for e_b in battery_energies:
    # Loop over hybridization ratios
    for H in hybrid_ratios:
        try:
            results = solve_takeoff_weight(
                crew_weight, payload_weight, A, C, cruise_segments, loiter_segments,
other_segments,
                H, e_f, e_b, eta_th, eta_elec,
                tol
            )
            _, _, fuel_weight_frac, _, _, fuel_weight, _, _ = results

            # Stop adding values if fuel weight becomes negative
            if fuel_weight <= 0:
```

```python
                break

            fuel_weights[e_b].append(fuel_weight)
            valid_hybrid_ratios[e_b].append(H)
        except ValueError:
            break  # Stop processing this hybridization series if an error occurs

# Plot results
plt.figure(figsize=(10, 6))
for e_b in battery_energies:
    plt.plot(valid_hybrid_ratios[e_b], fuel_weights[e_b], label=f"$H_b$ = {e_b :.2f} Wh/kg")

plt.xlabel("Hybridization Ratio ($\gamma^H$)")
plt.ylabel("Fuel Weight (lbs)")
plt.title("Fuel Weight vs Hybridization Ratio for Different Battery Specific Energies")
plt.legend()
plt.grid()
plt.show()
```