```python
import numpy as np

# Weight Estimate Calculator Modified for Assignment A4

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)
```

```python
    return sfc_convert

def EngineWeight(P, prop_eta = 0.77):
    """Estimates the weight of a turboprop engine based on the engines power requirement and
propeller efficiency
        P: The required thrust power of the engine in horsepower

        returns: The estimated weight of the engine in lbs"""

    P_shaft = P/prop_eta
    W_eng = (P_shaft**0.9306) * (10**-0.1205)

    return W_eng

def TaxiFraction(P, W, prop_eta = 0.77, c_SL = 0.60):
    """Calculates W_1/W_o or the fuel burn from running the engine for 15 min @ 5% max
        P: The required thrust power of the engine in horsepower
        W: Initial gross weight of the aircraft
        prop_eta: The propeller efficiency
        c_sl: Predicted specific fuel consumption at sea level in lbf/hp*hr"""

    P_shaft = P/prop_eta
    frac = 1 - (c_SL * P_shaft * 0.05 * 15 / (W * 60))

    return frac

def TakeoffFraction(P, W, prop_eta = 0.77, c_SL = 0.60):
    """Calculates W_2/W_1 or the fuel burn from running the engine at max power for 1 min
        P: The required thrust power of the engine in horsepower
        W: Gross weight of aircraft after taxi
        prop_eta: The propeller efficiency
        c_sl: Predicted specific fuel consumption at sea level in lbf/hp*hr"""

    P_shaft = P/prop_eta
    frac = 1 - (c_SL * P_shaft / (W * 60))

    return frac

def ZeroLiftDrag(S_ref, W_o, cf = 0.0070):
    """Calculates the zero lift drag of the aircraft based on the takeoff weight and wing area
(applies for clean config only)
        S_ref: The wing area of the aircraft in ft^2
        W_o: The GTOW of the aircraft in lbs
        cf: The predicted skin friction factor of the aircraft"""

    # Roskam constants
    c = 1.0447
    d = 0.5326

    S_wet = (10**c) * (W_o**d)
    f = S_wet*cf

    Cdo = f/S_ref

    return Cdo

def LiftCoefficient(Cdo, AR = 5, e = 1.1):
    """Calculates the lift coefficient when the aircraft is in its clean configuration"""
    k = (np.pi * e* AR)**-1
    C_L = np.sqrt(Cdo/k)
    L_D = 0.94 * C_L / (Cdo + k*(C_L**2))

    return L_D


# Iterative Solver for Takeoff Weight
```

```python
def solve_takeoff_weight_2(S_o, P_i, S_design, P_design, tol = 1e-6, max_iter=10000):
    """
    Iteratively solve for the takeoff weight of the aircraft.

    Parameters:
    - S_o: Current wing area in ft^2
    - P_i: Thrust Power in hp for the current iteration
    - S_design: The wing area at the chosen design point in ft^2
    - P_design: The required power at the chosen design point in hp
    - tol: Convergence tolerance
    - max_iter: max number of iterations

    Returns:
    - takeoff_weight: Converged takeoff weight in lbf
    """
    crew_weight = 190  # lbf
    payload_weight = 4000  # lbf
    A = 0.74              # Raymer constant for "Agricultural aircraft" (Table 3.1)
    C = -0.03             # Raymer constant for "Agricultural aircraft" (Table 3.1)
    aerial_desnsity = 1.2 # lbf/ft^2 from Table 7.1 in Metabook

    # Define flight segments
    cruise_segments = [
        (151903, 0.6, 0.82),  # Range in ft, specific fuel consumption in lbm/hp/hr, propeller efficiency, L/D
        (1033065, 0.6, 0.82),
        (151903, 0.6, 0.82),
    ]

    loiter_segments = [
        (0.5, 180, 0.6, 0.72),  # Endurance in hrs, velocity in ft/s, specific fuel consumption lbm/hp/hr, propeller efficiency, L/D
    ]

    other_segments = [0.998,0.999,0.998,0.999,0.998]  # Estimated weight fractions for non-cruise/loiter phases

    # Initial guess for takeoff weight
    takeoff_weight_guess = crew_weight + payload_weight + 10000  # Arbitrary guess
    iterations = 0

    # 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)

        # Accounting for weight of engine (Roskam Eqn)
        empty_weight = empty_weight_frac * takeoff_weight_guess
        empty_weight = empty_weight + EngineWeight(P_i) - EngineWeight(P_design)

        # Accounting for weight of wings
        empty_weight = empty_weight + (aerial_desnsity*(S_o - S_design))
        empty_weight_frac = empty_weight/takeoff_weight_guess

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

        # Taxi and takeoff fractions
        taxi_frac = TaxiFraction(P_i, takeoff_weight_guess)
        W_1 = taxi_frac * takeoff_weight_guess
        takeoff_frac = TakeoffFraction(P_i, W_1)
        segment_frac = segment_frac * taxi_frac * takeoff_frac

        # Cruise segments
```

```
        # Calculating new L/D based on S_o
        Cdo = ZeroLiftDrag(S_o, takeoff_weight_guess)
        L_D = LiftCoefficient(Cdo)

        print(L_D)

        for R, c, eta in cruise_segments:
            segment_frac *= CruiseFraction(R, c, eta, L_D)

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


        # Custom segments (These values come directly metabook/textbook)
        for frac in other_segments:
            segment_frac *= frac

        # 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 3: Solve for the new takeoff weight
        new_takeoff_weight = NewWeight(crew_weight, payload_weight, fuel_weight_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 < tol:
            # Calculate the actual weights
            empty_weight = empty_weight_frac * new_takeoff_weight
            fuel_weight = fuel_weight_frac * new_takeoff_weight
            return (new_takeoff_weight, empty_weight_frac, fuel_weight_frac,
                    empty_weight, fuel_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}")

################################################ Example for solving for takeoff weight
######################################
W_design = 12214.12 # lbs
WP_design = 17.8
WS_design = 25.4

S_design = W_design * (WS_design)**-1
P_design = W_design * (WP_design)**-1

# Values that should come from iteration in Algorithm 4
S_o = 387.2 # ft^2
T_i = 620 # hp
```

```python
results = solve_takeoff_weight_2(S_o, T_i, S_design, P_design)

takeoff_weight, empty_weight_frac, fuel_weight_frac,\
empty_weight, fuel_weight, iterations = results

print(f"Takeoff Weight Estimate after {iterations} iterations:")
print(f"  Takeoff Weight (W_naught): {takeoff_weight:.2f} lbs")
print(f"  Empty Weight Fraction: {empty_weight_frac:.4f}")
print(f"  Fuel Weight Fraction: {fuel_weight_frac:.4f}")
print(f"  Empty Weight: {empty_weight:.2f} lbs")
print(f"  Fuel Weight: {fuel_weight:.2f} lbs")
```