WeightAdjustment

August 31, 2022

0.0.1 Assumptions

- Assumes a 4+ with 4 rowers of the same weight
- Adjusts from a reference weight of 270 lbs in order to match C2 calculator https://www.concept2.com/indoor-rowers/training/calculators/weight-adjustment-calculator
- Ignores air resistance, assumes that weight doesn't significantly change it
- Assumes turbulent boundary layer due to Reynold's number >5*10^5 (https://en.wikipedia.org/wiki/Reynolds_number#Laminar%E2%80%93turbulent_transition)
- Assumes laminar flow outside of boundary layer
- For drag calcs, assume boat profile is an equilateral triangular prism with skin friction along 2 flat surfaces

0.0.2 Define functions

[55]:
```
import math

def CalculateSubmergedAreas(rower_weight_kg, rho, boat_length):

    # Boat weight assumptions
    oar_weight = 2.5 # kg
    boat_rigged_weight = 112/2.2 # 112 lbs in kg, https://www.pocock.com/shells/
↪hypercarbon-k4/
    coxswain_weight = 125/2.2 # 125 lbs in kg
    # Not used boat_width = 0.47 # meters, https://www.empacher.com/en/products/
↪racing-boats/empacher-racing-four

    # Calculate submerged surface area & cross sectional area
    total_boat_weight =␣
↪4*rower_weight_kg+boat_rigged_weight+coxswain_weight+4*oar_weight
    displaced_water_vol = total_boat_weight/rho
    boat_depth = math.sqrt(math.sqrt(3)*displaced_water_vol/boat_length) #␣
↪volume of triangular prism = depth^2*length/sqrt(3)
    submerged_surface_area = 4*boat_length*boat_depth/math.sqrt(3) # side␣
↪length of equilateral triangle = 2*depth/sqrt(3), 2 sides
    submerged_cross_sectional_area = math.pow(boat_depth,2)/math.sqrt(3)
```

```
    # print(f"depth = {boat_depth}, SA = {submerged_surface_area}, XA =␣
↪{submerged_cross_sectional_area}")

    return submerged_surface_area, submerged_cross_sectional_area

def WeightAdjustment(distance_m, minutes, seconds, rower_weight_kg):

    # Assumptions
    reference_rower_weight_kg = 270/2.2 # 270 lbs in kg, use same reference␣
↪weight as C2
    drag_coeff_hydro = 0.04 # This value is a pretty sketchy guess based on␣
↪this forum: https://www.boatdesign.net/threads/
↪calculating-friction-resistance-drag-coefficient.62553/
    rho = 998 # kg/m^3, fresh water http://web.mit.edu/2.016/www/handouts/
↪2005intro_1x.pdf
    mu = 0.001 # pascal-seconds, dynamic viscosity of water
    boat_length = 13.58 # meters, https://www.empacher.com/en/products/
↪racing-boats/empacher-racing-four

    # Calculate boatspeed
    v = distance_m/(minutes*60+seconds)

    # Calculate Reynolds number
    Re = rho*v*boat_length/mu # treat boat as a foil in a flow of water: https:/
↪/en.wikipedia.org/wiki/Reynolds_number#Flow_around_airfoils

    # Calculate areas for reference weight rowers
    submerged_surface_area_ref, submerged_cross_sectional_area_ref =␣
↪CalculateSubmergedAreas(reference_rower_weight_kg, rho, boat_length)

    # Calculate skin friction drag. Approximate coefficient using Prandtl's 1/7␣
↪power law: https://en.wikipedia.org/wiki/
↪Skin_friction_drag#Prandtl's_one-seventh-power_law
    drag_coeff_skin_fric = 0.027/math.pow(Re,(1/7))
    D_fric = 1/2*drag_coeff_skin_fric*rho*math.
↪pow(v,2)*submerged_surface_area_ref # Should be an area integral but assume␣
↪nothing varies across the surface

    # Calculate hydrodynamic (pressure) drag (https://ocw.mit.edu/courses/
↪2-20-marine-hydrodynamics-13-021-spring-2005/pages/lecture-notes/, L15 & L16)
    D_hydro = 1/2*drag_coeff_hydro*rho*math.
↪pow(v,2)*submerged_cross_sectional_area_ref

    # Rowing force = drag force with reference rower weights
    F_rowing = D_fric + D_hydro
```

```
    # Calculate areas for actual rower weight
    submerged_surface_area, submerged_cross_sectional_area =␣
↪CalculateSubmergedAreas(rower_weight_kg, rho, boat_length)

    # Solve for v_adjusted (assume that Reynolds number doesn't change␣
↪significantly so skin friction drag coeff is constant)
    v_adjusted = math.sqrt(2*F_rowing/
↪(rho*(drag_coeff_skin_fric*submerged_surface_area+drag_coeff_hydro*submerged_cross_sectiona

    # print(f"v = {v}, v_adj = {v_adjusted}")

    # Calculate adjusted time & return values
    time_adjusted_sec = distance_m/v_adjusted
    minutes_adjusted = math.floor(time_adjusted_sec/60)
    seconds_adjusted = round(time_adjusted_sec - minutes_adjusted*60,1)
    return(minutes_adjusted, seconds_adjusted)
```

0.0.3 Inputs

[41]:
```
distance_m = 5000 # distance of piece in meters
minutes = 17 # minutes of time of piece
seconds = 37.0 # seconds of time of piece
rower_weight_lbs = 192.4 # weight of rower in lbs
rower_weight_kg = rower_weight_lbs/2.2
```

0.0.4 Calculate weight adjusted time

[68]:
```
minutes_adjusted, seconds_adjusted = WeightAdjustment(distance_m, minutes,␣
↪seconds, rower_weight_kg)
print(f"Weight-adjusted time for {distance_m} meter piece: {minutes_adjusted}:
↪{seconds_adjusted}")
```

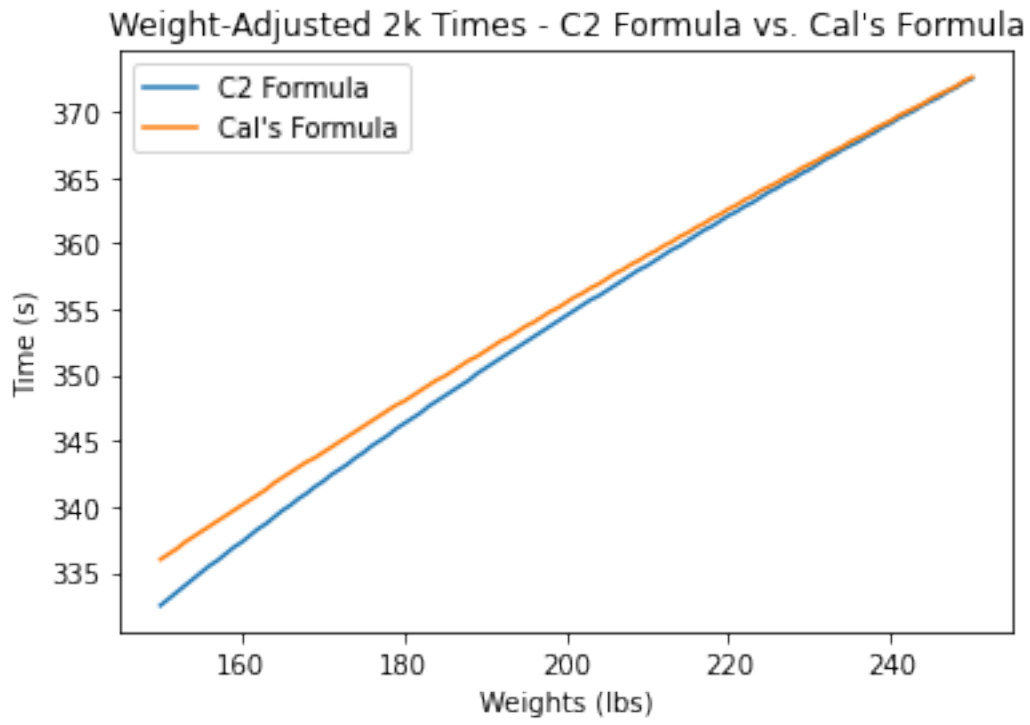
```
Weight-adjusted time for 5000 meter piece: 16:24.0
```

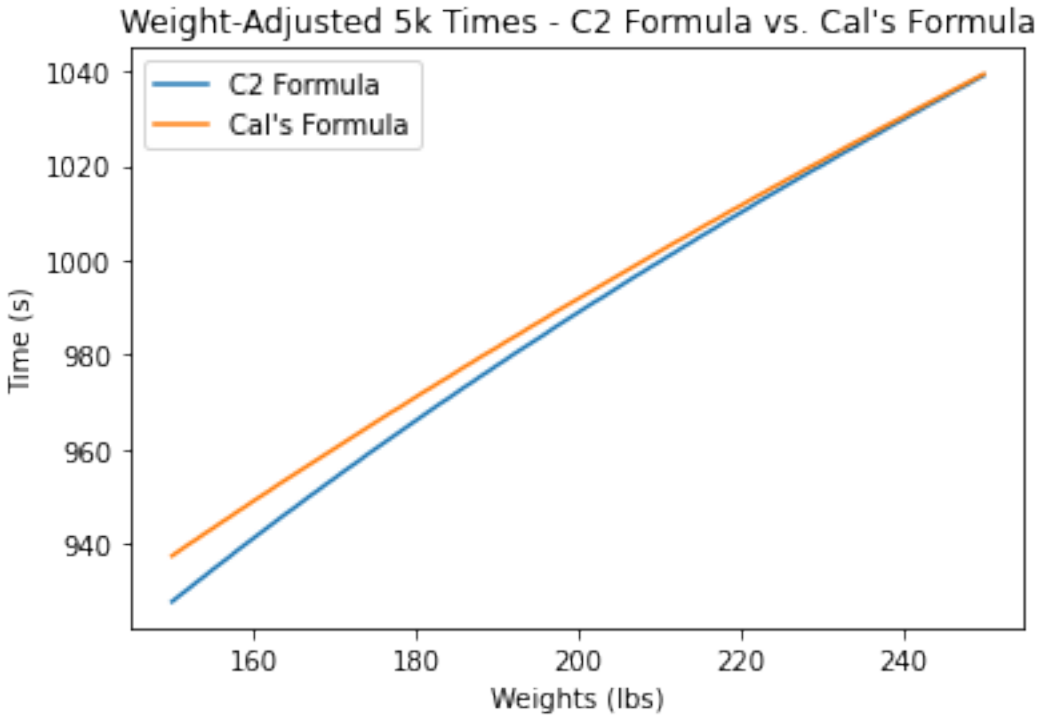
0.0.5 Plot sweep of weights at different distances to compare to C2 formula

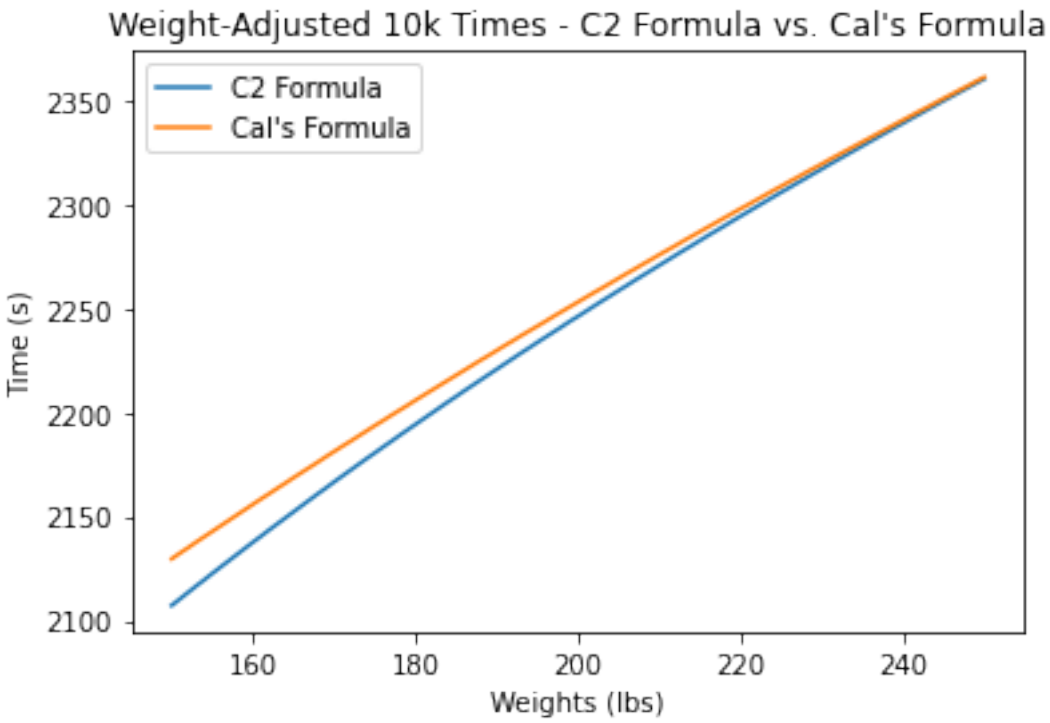
[77]:
```
import matplotlib.pyplot as plt

def C2WeightAdjustment(distance,minutes,seconds,weight_lbs):
    Wf = math.pow(weight/270,.222)
    time_adjusted_sec = (minutes*60+seconds)*Wf
    minutes_adjusted = math.floor(time_adjusted_sec/60)
    seconds_adjusted = round(time_adjusted_sec - minutes_adjusted*60,1)
    return(minutes_adjusted, seconds_adjusted)

# Initialize
```

```
C2_times_sec = {"2k times": [], "5k times": [], "10k times": []}
cal_times_sec = {"2k times": [], "5k times": [], "10k times": []}
distances = [2000, 5000, 10000]
times_min = [6,17,40]
times_sec = [18.9,37.0,1.0]
weights = range(150,251)

# Weight sweep
for weight in weights:
    for distance_ind in range(0,3):
        distance_key = list(cal_times_sec)[distance_ind]
        C2_min,C2_sec = C2WeightAdjustment(distances[distance_ind],times_min[distance_ind],times_sec[distance_ind],
        #print(f"Distance: {distances[distance_ind]}, Weight: {weight}, Time:{C2_min}:{C2_sec}")
        C2_times_sec[distance_key].append(C2_min*60+C2_sec)
        cal_min,cal_sec = WeightAdjustment(distances[distance_ind],times_min[distance_ind],times_sec[distance_ind],we
        cal_times_sec[distance_key].append(cal_min*60+cal_sec)

# Plot
plt.figure(0)
plt.plot(weights,C2_times_sec["2k times"])
plt.plot(weights,cal_times_sec["2k times"])
plt.xlabel("Weights (lbs)")
plt.ylabel("Time (s)")
plt.title("Weight-Adjusted 2k Times - C2 Formula vs. Cal's Formula")
plt.legend(["C2 Formula","Cal's Formula"])

plt.figure(1)
plt.plot(weights,C2_times_sec["5k times"])
plt.plot(weights,cal_times_sec["5k times"])
plt.xlabel("Weights (lbs)")
plt.ylabel("Time (s)")
plt.title("Weight-Adjusted 5k Times - C2 Formula vs. Cal's Formula")
plt.legend(["C2 Formula","Cal's Formula"])

plt.figure(2)
plt.plot(weights,C2_times_sec["10k times"])
plt.plot(weights,cal_times_sec["10k times"])
plt.xlabel("Weights (lbs)")
plt.ylabel("Time (s)")
plt.title("Weight-Adjusted 10k Times - C2 Formula vs. Cal's Formula")
plt.legend(["C2 Formula","Cal's Formula"])
```

[77]: <matplotlib.legend.Legend at 0x7f7a80a95970>


Weight-Adjusted 2k Times - C2 Formula vs. Cal's Formula
C2 Formula
Cal's Formula
370
365
360
355
350
345
340
335
Time (s)
160
180
200
220
240
Weights (lbs)


Weight-Adjusted 5k Times - C2 Formula vs. Cal's Formula
C2 Formula
Cal's Formula
1040
1020
1000
980
960
940
Time (s)
160
180
200
220
240
Weights (lbs)


Weight-Adjusted 10k Times - C2 Formula vs. Cal's Formula
C2 Formula
Cal's Formula
Time (s)
2350
2300
2250
2200
2150
2100
160
180
200
220
240
Weights (lbs)