MAE 158 Extra Credit Assignment

Thomas Slagle (72602272)

**Part 1**

**Problem Statement**

The Python script written for part one of the extra credit assignment evaluates performance characteristics of the DC – 9 – 30 commercial airliners over a range of flight speed values at a 31,000 ft cruising altitude. This aircraft was evaluated for speeds between 250 and 575 knots true airspeed. The aircraft parameters used in the script are shown in Table 1. The values obtained from the Python script are shown in Table 2 and plotted in Figures 1, 2, and 3. Lastly, the complete Python script is shown in Appendix A.

Table 1 presents the relevant parameters that were input in as variables in the Python script. The values listed in Table 1 were used to derive all other necessary inputs for the script such as the dynamic viscosity of air, the speed of sound, and the Reynolds number.

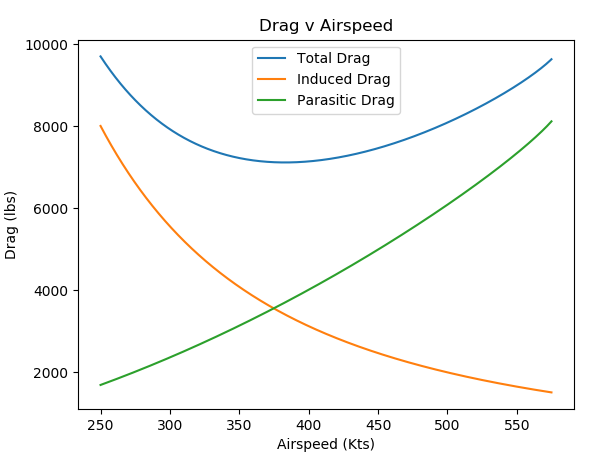
Table 1: DC – 9 – 30 Parameters

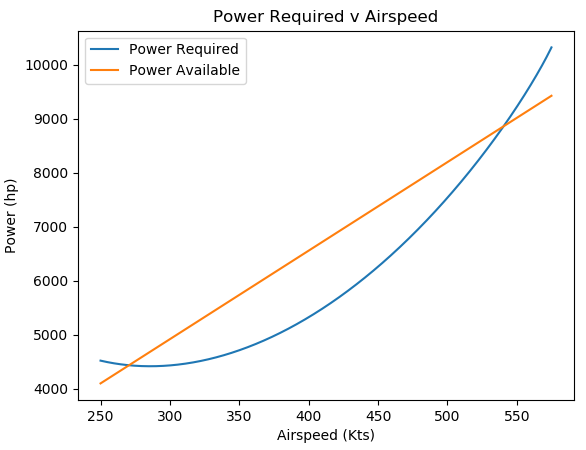
|  |  |
| --- | --- |
| **Parameter** | **Value** |
| **General Aircraft Properties** | |
| **Cruise Weight** () | **97,000** [] |
| **Descent Weight** () | **82,000** [] |
| **Engine Properties** | |
| **Specific Fuel Consumption** () | **0.82** [] |
| **Sea-Level Thrust** () | **14,500** [] |
| **Environment Properties** | |
| **Altitude** () | **31,000** [] |
| **Temperature** () | **399.67** [] |
| **Gas Constant** () | **1718** [] |
| **Heat Capacity Ratio** () | **1.4** |
| **Sea Level Density** () | **0.002377** [] |
| **Altitude Density** () | **0.000876** [] |
| **Wing Properties** | |
| **Planform Area** () | **1,000** [] |
| **Span** () | **93.2** [] |
| **Root Chord** () | **17.8** [] |
| **Sweep Angle** () | **24.5** [deg] |
| **Average Thickness Ratio** () | **0.106** |
| **Taper Ratio** () | **0.2** |
| **Wing Area Covered by Fuselage** () | **17**% |
| **Vertical Tail Properties** | |
| **Planform Area** () | **161** [] |
| **Root Chord** () | **15.5** [] |
| **Sweep Angle** () | **43.5** [deg] |
| **Average Thickness Ratio** () | **0.09** |
| **Taper Ratio** () | **0.80** |
| **Horizontal Tail Properties** | |
| **Planform Area** () | **261** [] |
| **Root Chord** () | **11.1** [] |
| **Sweep Angle** () | **31.6** [deg] |
| **Average Thickness Ratio** () | **0.09** |
| **Taper Ratio** () | **0.35** |
| **Fuselage Properties** | |
| **Wetted Area** () | **3,280** [] |
| **Diameter** (Ø) | **11.5** [] |
| **Length** () | **107** [] |
| **Nacelles Properties** | |
| **Wetted Area** () | **455** [] |
| **Finesse Ratio** () | **5.0** |
| **Length** () | **16.8** [] |
| **Pylons Properties** | |
| **Wetted Area** () | **117** [] |
| **Chord** () | **16.2** [] |
| **Sweep Angle** () | **0** [deg] |
| **Average Thickness Ratio** () | **0.06** |
| **Taper Ratio** () | **1.0** |
| **Flap Hinge Fairings** | |
| **Flat Plate Drag Area** () | **0.15** [] |

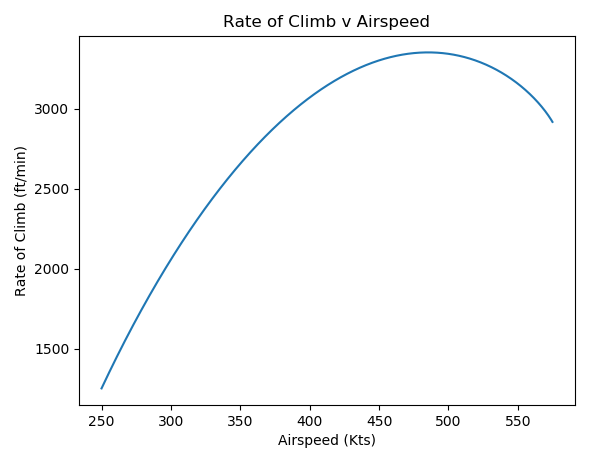
Table 2 and figures 1, 2, and 3 present the values and plots generated from the code. The values were all generated using the methods learned in class.

Table 2: Results obtained from the Python Script

|  |  |
| --- | --- |
| Parameter | Calculated Value |
| Total Parasitic Drag Coefficient |  |
| Total Flat Plate Drag Area |  |
| Maximum Range | 1988.46 miles @ 509.28 kts |
| Maximum Endurance | 1.21 hrs @ 383.06 kts |

  
Figure 1. Plot showing total, induced, and parasitic drags versus airspeed.

  
Figure 2. Plot showing power required and power available from the jet engines versus airspeed.

  
Figure 3. Plot showing rate of climb versus airspeed.

**Part 2**

**Problem Statement**

The Python script written for part one of the extra credit assignment evaluates performance characteristics of the Pipistrel Alpha Electro aircraft over a range of flight speed values at a 9,000 ft cruising altitude. This aircraft was evaluated for speeds between 25 and 200 knots true airspeed. The aircraft parameters used in the script are shown in Table 3. The values obtained from the Python script are shown in Table 4 and plotted in Figures 5, 6, and 7. Lastly, the complete Python script is shown in Appendix B.

Table 3 presents the relevant parameters that were input in as variables in the Python script. Values that were not available from Pipistrel directly were evaluated using the ImageJ program, which relates a known pixel length in an image to a known distance, the results are shown in Figure 4. The thickness ratio was assumed to be constant across the wing and tails, respectively. Taper ratio was difficult to estimate from the images available online and was thus assumed to be 1 for all geometries that needed it. The values listed in Table 3 were used to derive all other necessary inputs for the script such as the dynamic viscosity of air, the speed of sound, and the Reynolds number.

A small airplane on a runway

Description automatically generated with medium confidence

Figure 4. ImageJ results for aircraft parameters.

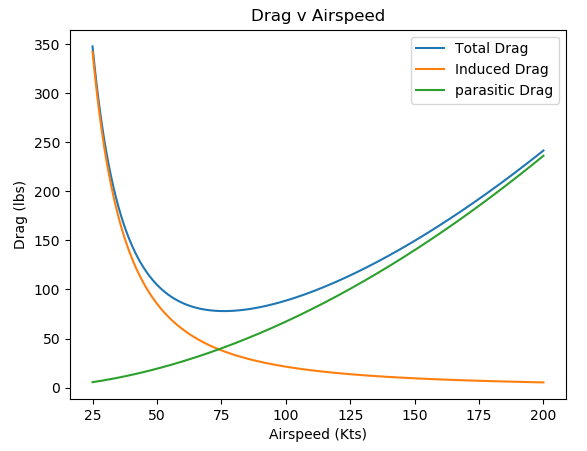
Table 3: Pipistrel Alpha Electro Parameters

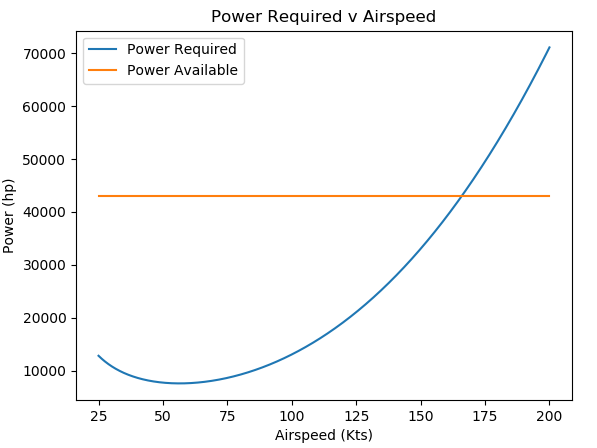
|  |  |
| --- | --- |
| **Parameter** | **Value** |
| **General Aircraft Properties** | |
| **Gross Weight** () | **1,214** [] |
| **Battery Weight** () | **277** [] |
| **Engine Properties** | |
| **Cell Energy Density** () | **130** [] |
| **Propellor-Motor Efficiency** () | **86**% |
| **Battery Efficiency** () | **180** [] |
| **Motor Shaft Power** () | **50,000** [] |
| **Environment Properties** | |
| **Altitude** () | **9000** [] |
| **Temperature** () | **486.61** [] |
| **Gas Constant** () | **1718** [] |
| **Heat Capacity Ratio** () | **1.4** |
| **Sea Level Density** () | **0.002377** [] |
| **Altitude Pressure** () | **1512.9** [] |
| **Wing Properties** | |
| **Planform Area** () | **102.4** [] |
| **Span** () | **34.5** [] |
| **Root Chord** () | **2.896** [] |
| **Sweep Angle** () | **0** [deg] |
| **Thickness Ratio** () | **0.1775** |
| **Taper Ratio** () | **1** |
| **Vertical Tail Properties** | |
| **Planform Area** () | **11.8** [] |
| **Root Chord** () | **4.109** [] |
| **Sweep Angle** () | **0** [deg] |
| **Thickness Ratio** () | **0.10** |
| **Taper Ratio** () | **1** |
| **Horizontal Tail Properties** | |
| **Planform Area** () | **11.6** [] |
| **Root Chord** () | **2.240** [] |
| **Sweep Angle** () | **0** [deg] |
| **Thickness Ratio** () | **0.10625** |
| **Taper Ratio** () | **1** |
| **Fuselage Properties** | |
| **Length** () | **21.4** [] |
| **Diameter** (Ø) | **3.5** [] |
| **Landing Gear** | |
| **Flat Plate Drag Area** () | **0.15** [] |

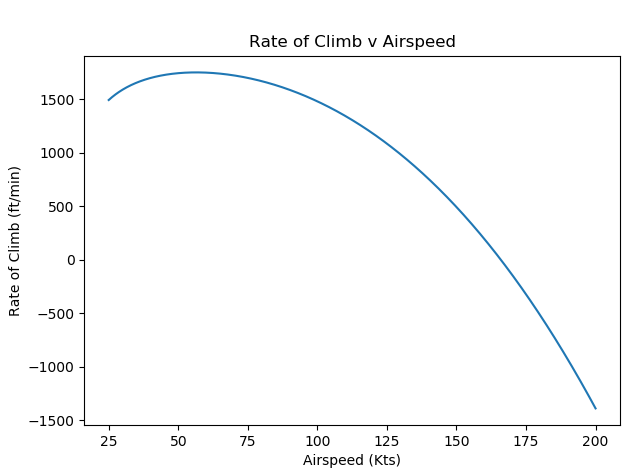
Table 2 and figures 1, 2, and 3 present the values and plots generated from the code. The values were all generated using the methods learned in class. The results obtained from the code were compared to the spec sheet for this aircraft from Pipistrel’s website. The results matched well for the flight speed range, the listed maximum fight speed was 135 kts with a nominal cruising speed of 85 kts; however, the max range and endurance values generated from the Python script were much higher than their listed values. This is likely due to the cell efficiency given in the assignment description being higher than what pipistrel offers.

Table 2: Results obtained from the Python Script

|  |  |
| --- | --- |
| Parameter | Calculated Value |
| Total Parasitic Drag Coefficient |  |
| Total Flat Plate Drag Area |  |
| Maximum Range | 529.18 miles @ 75.98 kts |
| Maximum Endurance | 4.76 hrs @ 56.36 kts |

  
Figure 5. Plot showing total, induced, and parasitic drags versus airspeed.

  
Figure 6. Plot showing power required and power available from the propellor motor versus airspeed.

  
Figure 7. Plot showing rate of climb versus airspeed; the graph becomes negative one the power required is greater than the power available (shown in Figure 6).

**Appendix A**

import numpy as np

from matplotlib import pyplot as plt

*#Functions*

def Re(R,L):

    return (R\*L)

def C\_th(C\_r, sigma):

    return C\_r \* sigma

def MAC\_exp(Y,C\_r,sigma,B):

    C\_t = C\_th(C\_r, sigma)

    C\_r\_fuse = C\_r - (C\_r - C\_t)\*Y/(B/2)

    sigma\_exp = C\_t/C\_r\_fuse

    return (2/3\*C\_r\_fuse\*(1+sigma\_exp-sigma\_exp/(1+sigma\_exp)))

def MAC(C\_r,sigma):

    return (2/3\*C\_r\*(1+sigma-sigma/(1+sigma)))

*#Parasitic Drag Equation (Calc from Nabil Discussion)*

def c\_f(R):

    return 0.208\*R\*\*-0.27845 + 0.00101

*#0.1944\*R\*\*-0.2734 + 0.0009782*

*#e\_lambda\_zero Equation (calc from Nabil Discussion)*

def e(Ar, C\_d\_p, \_lambda):

    e\_straight = np.abs(1.78\*(1 - 0.045\*Ar\*\*0.68) - 0.64)

    e\_30 = np.abs(4.61\*(1 - 0.045\*Ar\*\*0.68)\*(np.cos(np.radians(\_lambda)))\*\*0.15 - 3.1)

    e\_mid = np.abs((e\_30 - e\_straight)/(np.radians(\_lambda) - 0))

    if (\_lambda == 0):

        return e\_straight

    elif (\_lambda >= 30):

        return e\_30

    else:

        return e\_mid

*#0.9423 + 0.00041\*Ar + 4.591\*C\_d\_p - 6.878e-06\*Ar\*\*2 - 1.348\*Ar\*C\_d\_p - 242.9 \*C\_d\_p\*\*2 + 0.01989\*Ar\*\*2\*C\_d\_p + 11.86\*Ar\*C\_d\_p\*\*2 + 3483\*C\_d\_p\*\*3*

def kappa(M, Lambda, TC):

    z = ((2-M\*\*2)\*np.cos(np.radians(Lambda)))/(np.sqrt(1 - M\*\*2\*np.cos(np.radians(Lambda))))

    return 1 + z\*TC + 100\*TC\*\*4

*#Given Parameters*

*#General Plane Properties*

W\_i = 97000 *#lbs*

W\_d = 82000 *#lbs*

C\_t = 0.82 *#lb/lb-h*

W\_ker = 6.7 *#lb/gal*

*#Wing Geometry*

Lambda\_w = 24.5 *#deg*

tc\_w = 0.106

b\_w = 93.2 *#ft*

S\_w = 1000 *#ft2*

sigma\_w = 0.2

c\_r\_w = 17.8 *#ft*

S\_covered\_w = 0.17

*#Vertical Tail*

Lambda\_v = 43.5 *#deg*

tc\_v = 0.09

S\_v = 161 *#ft2*

sigma\_v = 0.80

c\_r\_v = 15.5

*#Horizontal Tail*

Lambda\_h = 31.6 *#deg*

tc\_h = 0.09

S\_h = 261 *#ft2*

sigma\_h = 0.35

c\_r\_h = 11.1 *#ft*

*#Fuselage Geometry*

l = 107 *#ft*

dia = 11.5 *#ft*

S\_wet = 3280 *#ft2*

*#Pylons Geometry*

S\_wet\_p = 117 *#ft2*

tc\_p = 0.06

Lambda\_p = 0 *#deg*

sigma\_p = 1.0

c\_p = 16.2 *#ft*

*#Nacelles Geometry*

S\_wet\_n = 455 *#ft2*

fin\_ratio = 5.0

L\_n = 16.8 *#ft*

*#Flap hinge Fairing*

delta\_f\_FH = 0.15 *#ft2*

*#Environmental Variables*

h = 31000 *#ft*

*#M = 0.78*

T\_f = -60 *#F*

T = -60 + 459.67

rho\_sl = 0.002377

rho = 0.000876 *#slugs/ft3*

mu = 0.3170\*(T\*\*(3/2))\*(734.7/(T+216))\*(1/10\*\*10) *#3.04e-07*

*#print(mu)*

gamma = 1.4

R = 1718

a = np.sqrt(gamma\*R\*(T))

*#print(a)*

*#V\_0 = M\*a*

T\_sl = 14500

T = T\_sl\*(rho/rho\_sl)

print(T)

*#Arrays*

V\_array\_kts = np.linspace(250,575,1000)

D\_i\_array = np.array([])

D\_p\_array = np.array([])

D\_tot\_array = np.array([])

ld\_array = np.array([])

R\_array = np.array([])

E\_array = np.array([])

P\_req\_array = np.array([])

P\_ava\_array = np.array([])

P\_exc\_array = np.array([])

roc\_array = np.array([])

i = 0

for i in range (len(V\_array\_kts)):

    V = V\_array\_kts[i]\*1.68781

    M = V/a

    q = (rho\*V\*\*2)/2

*#q = gamma/2\*601.61\*M\*\*2*

    Re\_l = rho\*V/mu *#Rn/ft*

*#Wing*

    mac\_w = MAC\_exp(dia, c\_r\_w, sigma\_w, b\_w)

    re\_w = Re(Re\_l, mac\_w)

    cf\_w = c\_f(re\_w)

    kappa\_w = kappa(M, Lambda\_w, tc\_w)

    delta\_f\_w = (1.02\*2\*S\_w\*(1-S\_covered\_w)\*cf\_w\*kappa\_w)

*#Horizontal Tail*

    mac\_h = MAC(c\_r\_h, sigma\_h)

    re\_h = Re(Re\_l, mac\_h)

    cf\_h = c\_f(re\_h)

    kappa\_h = kappa(M, Lambda\_h, tc\_h)

    delta\_f\_h = (1.02\*2\*S\_h\*cf\_w\*kappa\_h)

*#Vertical Tail*

    mac\_v = MAC(c\_r\_v, sigma\_v)

    re\_v = Re(Re\_l, mac\_v)

    cf\_v = c\_f(re\_v)

    kappa\_v = kappa(M, Lambda\_v, tc\_v)

    delta\_f\_v = (1.02\*2\*S\_v\*cf\_v\*kappa\_v)

*#Pylons*

    re\_p = Re(Re\_l, c\_p)

    cf\_p = c\_f(re\_p)

    kappa\_p = kappa(M, Lambda\_p, tc\_p)

    delta\_f\_p = (S\_wet\_p\*cf\_p\*kappa\_p)

*#Fuselage*

    re\_f = Re(Re\_l, l)

    fin\_F = l/dia

    kappa\_f = 1.11

    cf\_f = c\_f(re\_f)

    delta\_f\_f = S\_wet\*cf\_f\*kappa\_f

*#Nacelles*

    re\_n = Re(Re\_l, L\_n)

    cf\_n = c\_f(re\_n)

    kappa\_n = 1.29

    delta\_f\_n = S\_wet\_n\*cf\_n\*kappa\_n

*#Total Parasite drag*

    f = 1.10\*(delta\_f\_w + delta\_f\_f + delta\_f\_h + delta\_f\_v + delta\_f\_n + delta\_f\_p + delta\_f\_FH)

    C\_d\_p = f/S\_w

    AsR = b\_w\*\*2/S\_w

    C\_l = W\_i/(q\*S\_w)

*#print(C\_l)*

    C\_d\_i = C\_l\*\*2/(np.pi\*AsR\*e(AsR, C\_d\_p, Lambda\_w))

    D\_i = (W\_i/b\_w)\*\*2/(np.pi\*q\*e(AsR, C\_d\_p, Lambda\_w))

    D\_i\_array = np.append(D\_i\_array,[D\_i])

    D\_p = f\*q

    D\_p\_array = np.append(D\_p\_array,[D\_p])

    D\_tot = D\_p + D\_i

    C\_d = D\_tot/(q\*S\_w)

    D\_tot\_array = np.append(D\_tot\_array,[D\_tot])

    ld = D\_tot/W\_i

    ld\_array = np.append(ld\_array, [ld])

    P\_req = (1/550)\*np.sqrt(2\*W\_i\*\*3/(rho\_sl\*S\_w))\*1/(C\_l\*\*(3/2)/C\_d) *#D\_tot\*V/550#*

    P\_req\_array = np.append(P\_req\_array, [P\_req])

    R\_jet = 2/C\_t \* np.sqrt(2/(rho\*S\_w))\*np.sqrt(C\_l)/(C\_d\_p+C\_d\_i)\*(np.sqrt(W\_i)-np.sqrt(W\_d))

    R\_array = np.append(R\_array, [R\_jet])

    E\_jet = 1/C\_t\*(C\_l/C\_d)\*np.log10(W\_i/W\_d)

    E\_array = np.append(E\_array, [E\_jet])

    P\_ava = T\*V/550

    P\_ava\_array = np.append(P\_ava\_array, [P\_ava])

    P\_exc\_array = np.append(P\_exc\_array, [(P\_ava-P\_req)\*4])

    roc = (T\_sl/W\_i - 1/(C\_l/C\_d))\*V\*60

    roc\_array = np.append(roc\_array,[roc])

print('Your max range is: \n' +  str(np.max(R\_array)) + ' miles\nThis value occurs at:\n' + str(V\_array\_kts[np.argmax(R\_array)]) + ' kts\n\n')

print('Your max endurance is:\n' + str(np.max(E\_array)) + ' hr\n' + str(V\_array\_kts[np.argmax(E\_array)]) + ' kts')

plt.plot(V\_array\_kts, D\_tot\_array, label = 'Total Drag')

plt.plot(V\_array\_kts, D\_i\_array, label = 'Induced Drag')

plt.plot(V\_array\_kts, D\_p\_array, label = 'Parasitic Drag')

plt.xlabel('Airspeed (Kts)')

plt.ylabel('Drag (lbs)')

plt.title('Drag v Airspeed')

plt.legend()

plt.show()

plt.close()

plt.plot(V\_array\_kts, P\_req\_array, label = 'Power Required')

plt.plot(V\_array\_kts, P\_ava\_array, label = 'Power Available')

*#plt.plot(V\_array\_kts, P\_exc\_array, label = 'Excess Power')*

*#ax2 = plt.secondary\_yaxis("right", functions=(V\_array\_kts, P\_exc\_array))*

plt.xlabel('Airspeed (Kts)')

plt.ylabel('Power (hp)')

plt.title('Power Required v Airspeed')

plt.legend()

plt.show()

plt.close()

plt.plot(V\_array\_kts, roc\_array)

plt.xlabel('Airspeed (Kts)')

plt.ylabel('Rate of Climb (ft/min)')

plt.title('Rate of Climb v Airspeed')

plt.show()

**Appendix B**

*###### PART 2 #####*

import numpy as np

from matplotlib import pyplot as plt

*#Functions*

def Re(R,L):

    return (R\*L)

def C\_th(C\_r, sigma):

    return C\_r \* sigma

def MAC\_exp(Y,C\_r,sigma,B):

    C\_t = C\_th(C\_r, sigma)

    C\_r\_fuse = C\_r - (C\_r - C\_t)\*Y/(B/2)

    sigma\_exp = C\_t/C\_r\_fuse

    return (2/3\*C\_r\_fuse\*(1+sigma\_exp-sigma\_exp/(1+sigma\_exp)))

def MAC(C\_r,sigma):

    return (2/3\*C\_r\*(1+sigma-sigma/(1+sigma)))

*#Parasitic Drag Equation (Calc from Nabil Discussion)*

def c\_f(R):

    return 0.208\*R\*\*-0.27845 + 0.00101

*#0.1944\*R\*\*-0.2734 + 0.0009782*

*#e\_lambda\_zero Equation (calc from Nabil Discussion)*

def e(Ar, C\_d\_p, \_lambda):

    e\_straight = np.abs(1.78\*(1 - 0.045\*Ar\*\*0.68) - 0.64)

    return e\_straight

def kappa(M, Lambda, TC):

    z = ((2-M\*\*2)\*np.cos(np.radians(Lambda)))/(np.sqrt(1 - M\*\*2\*np.cos(np.radians(Lambda))))

    return 1 + z\*TC + 100\*TC\*\*4

*#Given Parameters*

C\_bat = 130 *#Wh/lb*

bat = 180 *#W/lb-T*

eta\_p = 0.86 *#https://en.wikipedia.org/wiki/Propulsive\_efficiency*

*#Basing my design on the Alpha-Electro Plane*

*#Many basic dimensions: https://www.pipistrel-aircraft.com/aircraft/electric-flight/alpha-electro/*

*#Measurements taken using imageJ*

W\_bat = 277 *# lbs*

Wh\_bat = W\_bat \* C\_bat *#Wh*

W\_gross = 1214 *#lb*

P\_shaft = 50000 *#Watts*

*#Wing Geometry*

Lambda\_w = 0 *#deg*

tc\_w = 0.1775

b\_w = 34.5 *#ft*

S\_w = 102.4 *#ft2*

sigma\_w = 1

c\_r\_w = 2.896 *#ft*

*#Vertical Tail*

Lambda\_v = 0 *#deg*

tc\_v = 0.10

S\_v = 11.8 *#ft2*

sigma\_v = 0.654

c\_r\_v = 4.109

*#Horizontal Tail*

Lambda\_h = 0 *#deg*

tc\_h = 0.10625

S\_h = 11.6 *#ft2*

sigma\_h = 1

c\_r\_h = 2.240 *#ft*

*#Fuselage Geometry*

l = 21.4 *#ft*

dia = 3.5 *#ft*

S\_wet = np.pi \* dia \* (l - 1.3\*dia) *#ft2 #calculated from https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118568101.app1*

*#Landing Gear*

delta\_f\_LG = 0.15 *#ft2*

*#Environmental Variables*

gamma = 1.4

R = 1718

h = 9000 *#ft*

T = 486.61 *#R*

rho\_sl = 0.002377

P = 1512.9

rho = P/(R\*T) *#slugs/ft3*

mu = 0.3170\*(T\*\*(3/2))\*(734.7/(T+216))\*(1/10\*\*10) *#3.04e-07*

a = np.sqrt(gamma\*R\*(T))

*#print(T)*

*#Arrays*

V\_array\_kts = np.linspace(25,200,1000)

D\_i\_array = np.array([])

D\_p\_array = np.array([])

D\_tot\_array = np.array([])

ld\_array = np.array([])

R\_array = np.array([])

E\_array = np.array([])

P\_req\_array = np.array([])

P\_ava\_array = np.array([])

P\_exc\_array = np.array([])

roc\_array = np.array([])

i = 0

for i in range (len(V\_array\_kts)):

    V = V\_array\_kts[i]\*1.68781

    M = V/a

    q = (rho\*V\*\*2)/2

*#q = gamma/2\*601.61\*M\*\*2*

    Re\_l = rho\*V/mu *#Rn/ft*

*#Wing*

    mac\_w = MAC\_exp(dia, c\_r\_w, sigma\_w, b\_w)

    re\_w = Re(Re\_l, mac\_w)

    cf\_w = c\_f(re\_w)

    kappa\_w = kappa(M, Lambda\_w, tc\_w)

    delta\_f\_w = (1.02\*2\*S\_w\*cf\_w\*kappa\_w)

*#print(delta\_f\_w)*

*#Horizontal Tail*

    mac\_h = MAC(c\_r\_h, sigma\_h)

    re\_h = Re(Re\_l, mac\_h)

    cf\_h = c\_f(re\_h)

    kappa\_h = kappa(M, Lambda\_h, tc\_h)

    delta\_f\_h = (1.02\*2\*S\_h\*cf\_w\*kappa\_h)

*#Vertical Tail*

    mac\_v = MAC(c\_r\_v, sigma\_v)

    re\_v = Re(Re\_l, mac\_v)

    cf\_v = c\_f(re\_v)

    kappa\_v = kappa(M, Lambda\_v, tc\_v)

    delta\_f\_v = (1.02\*2\*S\_v\*cf\_v\*kappa\_v)

*#Fuselage*

    re\_f = Re(Re\_l, l)

    fin\_F = l/dia

    kappa\_f = 1.11

    cf\_f = c\_f(re\_f)

    delta\_f\_f = S\_wet\*cf\_f\*kappa\_f

*#Total Parasite drag*

    f = 1.10\*(delta\_f\_w + delta\_f\_f + delta\_f\_h + delta\_f\_v + delta\_f\_LG)

    C\_d\_p = f/S\_w

    AsR = b\_w\*\*2/S\_w

    C\_l = W\_gross/(q\*S\_w)

*#print(C\_l)*

    C\_d\_i = C\_l\*\*2/(np.pi\*AsR\*e(AsR, C\_d\_p, Lambda\_w))

    D\_i = (W\_gross/b\_w)\*\*2/(np.pi\*q\*e(AsR, C\_d\_p, Lambda\_w))

    D\_i\_array = np.append(D\_i\_array,[D\_i])

    D\_p = f\*q

    D\_p\_array = np.append(D\_p\_array,[D\_p])

    D\_tot = D\_p + D\_i

    C\_d = D\_tot/(q\*S\_w)

    D\_tot\_array = np.append(D\_tot\_array,[D\_tot])

    ld = D\_tot/W\_gross

    ld\_array = np.append(ld\_array, [ld])

    P\_req = np.sqrt(2\*W\_gross\*\*3/(rho\_sl\*S\_w))\*1/(C\_l\*\*(3/2)/C\_d) *#(1/550)\**

    P\_req\_array = np.append(P\_req\_array, [P\_req])

    E\_prop = Wh\_bat/P\_req

    E\_array = np.append(E\_array, [E\_prop])

    R\_prop = E\_prop \* V

    R\_array = np.append(R\_array, [R\_prop])

    P\_ava = P\_shaft \* eta\_p

    P\_ava\_array = np.append(P\_ava\_array, [P\_ava])

    P\_exc\_array = np.append(P\_exc\_array, [(P\_ava-P\_req)\*4])

    roc = (P\_ava-P\_req)/W\_gross\*60

    roc\_array = np.append(roc\_array,[roc])

print('Your max range is: \n' +  str(np.max(R\_array)) + ' miles\nThis value occurs at:\n' + str(V\_array\_kts[np.argmax(R\_array)]) + ' kts\n\n')

print('Your max endurance is:\n' + str(np.max(E\_array)) + ' hr\n' + str(V\_array\_kts[np.argmax(E\_array)]) + ' kts')

*#PLOTS*

plt.plot(V\_array\_kts, D\_tot\_array, label = 'Total Drag')

plt.plot(V\_array\_kts, D\_i\_array, label = 'Induced Drag')

plt.plot(V\_array\_kts, D\_p\_array, label = 'parasitic Drag')

plt.xlabel('Airspeed (Kts)')

plt.ylabel('Drag (lbs)')

plt.title('Drag v Airspeed')

plt.legend()

plt.show()

plt.close()

plt.plot(V\_array\_kts, P\_req\_array, label = 'Power Required')

plt.plot(V\_array\_kts, P\_ava\_array, label = 'Power Available')

*#plt.plot(V\_array\_kts, P\_exc\_array, label = 'Excess Power')*

*#ax2 = plt.secondary\_yaxis("right", functions=(V\_array\_kts, P\_exc\_array))*

plt.xlabel('Airspeed (Kts)')

plt.ylabel('Power (hp)')

plt.title('Power Required v Airspeed')

plt.legend()

plt.show()

plt.close()

plt.plot(V\_array\_kts, roc\_array)

plt.xlabel('Airspeed (Kts)')

plt.ylabel('Rate of Climb (ft/min)')

plt.title('Rate of Climb v Airspeed')

plt.show()