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 (W_c)	**97,000** [lbs]
Descent Weight (W_d)	**82,000** [lbs]
Engine Properties	
Specific Fuel Consumption (C_t)	**0.82** [$^{lb}/_{lb-h}$]
Sea-Level Thrust (T_{sl})	**14,500** [lbs]
Environment Properties	
Altitude (h)	**31,000** [ft]
Temperature (T)	**399.67** [$°R$]
Gas Constant (R)	**1718** [$^{ft*lbf}/_{slug*°R}$]
Heat Capacity Ratio (γ)	**1.4**
Sea Level Density (ρ_{sl})	**0.002377** [$^{slugs}/_{ft^3}$]
Altitude Density (ρ)	**0.000876** [$^{slugs}/_{ft^3}$]
Wing Properties	
Planform Area (S_w)	**1,000** [ft^2]

Span (b_w)	**93.2** [ft]
Root Chord (c_{r_w})	**17.8** [ft]
Sweep Angle (Λ_w)	**24.5** [deg]
Average Thickness Ratio (tc_w)	**0.106**
Taper Ratio (σ_w)	**0.2**
Wing Area Covered by Fuselage (S_{cov_w})	**17**%
Vertical Tail Properties	
Planform Area (S_v)	**161** [ft^2]
Root Chord (c_{r_v})	**15.5** [ft]
Sweep Angle (Λ_v)	**43.5** [deg]
Average Thickness Ratio (tc_v)	**0.09**
Taper Ratio (σ_v)	**0.80**
Horizontal Tail Properties	
Planform Area (S_h)	**261** [ft^2]
Root Chord (c_{r_h})	**11.1** [ft]
Sweep Angle (Λ_h)	**31.6** [deg]
Average Thickness Ratio (tc_h)	**0.09**
Taper Ratio (σ_h)	**0.35**
Fuselage Properties	
Wetted Area (S_{wet_f})	**3,280** [ft^2]
Diameter ($\emptyset$)	**11.5** [ft]
Length (l_f)	**107** [ft]
Nacelles Properties	
Wetted Area (S_{wet_n})	**455** [ft^2]
Finesse Ratio (fin)	**5.0**
Length (l_n)	**16.8** [ft]
Pylons Properties	
Wetted Area (S_{wet_p})	**117** [ft^2]
Chord (C_p)	**16.2** [ft]
Sweep Angle (Λ_p)	**0** [deg]
Average Thickness Ratio (tc_p)	**0.06**
Taper Ratio (σ_p)	**1.0**
Flap Hinge Fairings	
Flat Plate Drag Area (f_{FH})	**0.15** [ft^2]

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	**0.0217 – 0.0197** across airspeed range
Total Flat Plate Drag Area	**21.75 – 19.69** [ft²] across airspeed range
Maximum Range	**1988.46** [miles] @ **509.28** [kts]
Maximum Endurance	**1.21** [hrs] @ **383.06** [kts]

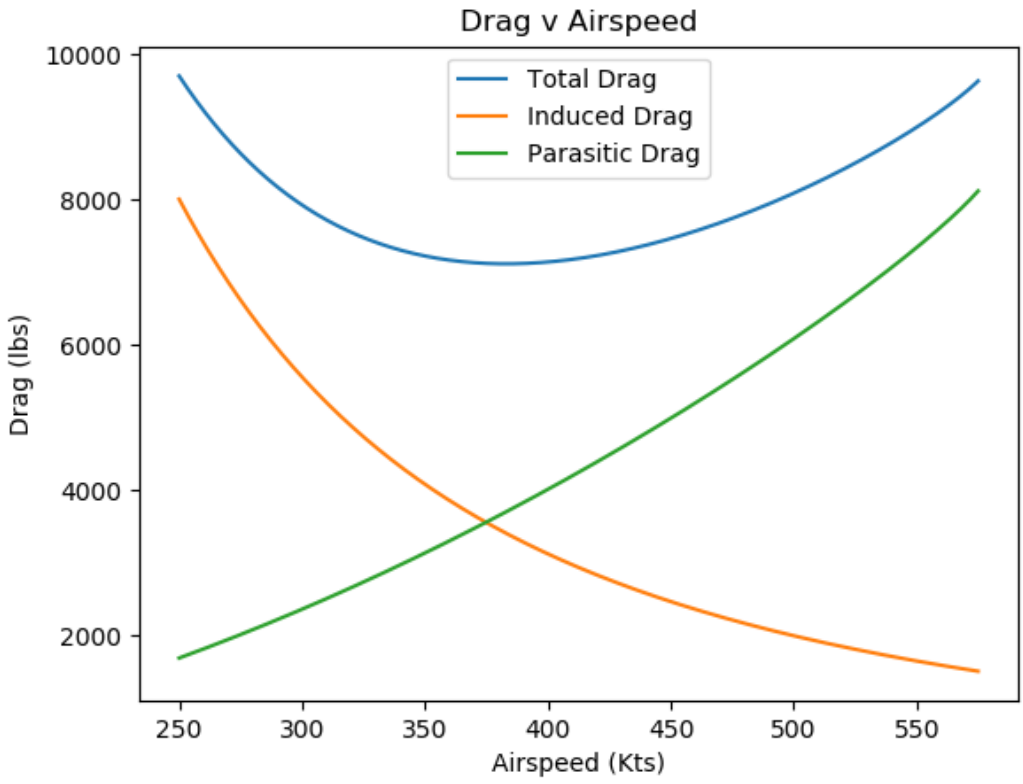

Figure 1. Plot showing total, induced, and parasitic drags versus airspeed.

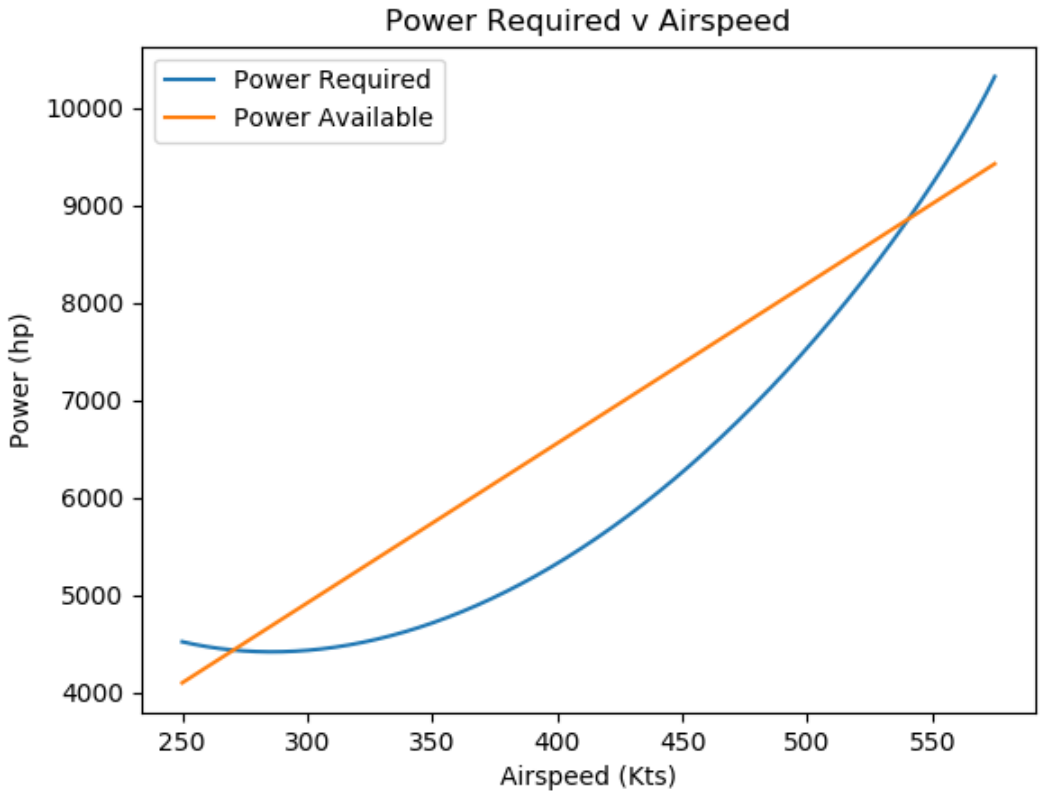

Figure 2. Plot showing power required and power available from the jet engines versus airspeed.

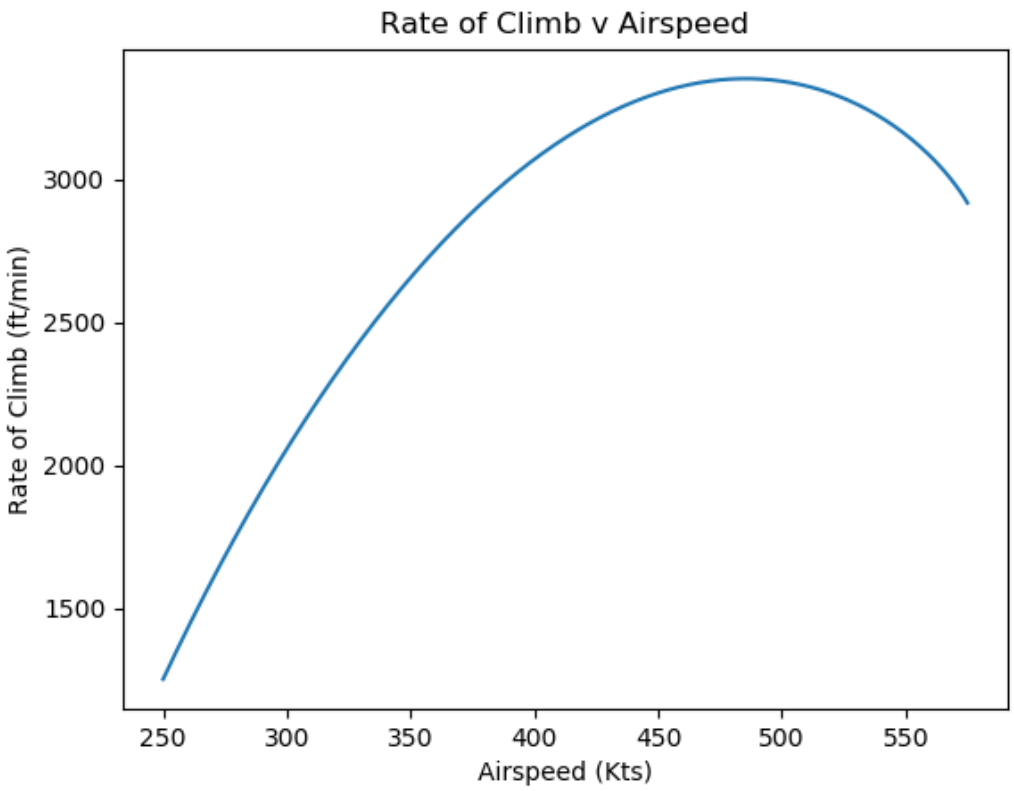

Figure 3. Plot showing rate of climb versus airspeed.

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.

Figure 4. ImageJ results for aircraft parameters.

Table 3: Pipistrel Alpha Electro Parameters

Parameter	Value
General Aircraft Properties	
Gross Weight (W_g)	**1,214** [lbs]
Battery Weight (W_b)	**277** [lbs]
Engine Properties	
Cell Energy Density (C_b)	**130** [$^{Wh}/_{lb}$]
Propellor-Motor Efficiency (η)	**86**%
Battery Efficiency (bat)	**180** [$^{W}/_{lb-T}$]
Motor Shaft Power (P_{shaft})	**50,000** [W]
Environment Properties	
Altitude (h)	**9000** [ft]
Temperature (T)	**486.61** [$°R$]
Gas Constant (R)	**1718** [$^{ft*lbf}/_{slug*°R}$]
Heat Capacity Ratio (γ)	**1.4**
Sea Level Density (ρ_{sl})	**0.002377** [$^{slugs}/_{ft^3}$]
Altitude Pressure (P)	**1512.9** [$^{lb}/_{ft\text{^}2}$]
Wing Properties	
Planform Area (S_w)	**102.4** [ft^2]
Span (b_w)	**34.5** [ft]
Root Chord (c_{r_w})	**2.896** [ft]
Sweep Angle (Λ_w)	**0** [deg]
Thickness Ratio (tc_w)	**0.1775**
Taper Ratio (σ_w)	**1**
Vertical Tail Properties	
Planform Area (S_v)	**11.8** [ft^2]
Root Chord (c_{r_v})	**4.109** [ft]
Sweep Angle (Λ_v)	**0** [deg]
Thickness Ratio (tc_v)	**0.10**
Taper Ratio (σ_v)	**1**

Horizontal Tail Properties	
Planform Area (S_h)	**11.6** [ft^2]
Root Chord (c_{r_h})	**2.240** [ft]
Sweep Angle (Λ_h)	**0** [deg]
Thickness Ratio (tc_h)	**0.10625**
Taper Ratio (σ_h)	**1**
Fuselage Properties	
Length (l_f)	**21.4** [ft]
Diameter (Ø)	**3.5** [ft]
Landing Gear	
Flat Plate Drag Area (f_{LG})	**0.15** [ft^2]

Table 4 and figures 5, 6, and 7 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 4: Results obtained from the Python Script

Parameter	Calculated Value
Total Parasitic Drag Coefficient	**0.0338 – 0.0224** across airspeed range
Total Flat Plate Drag Area	**3.46 – 2.29** [ft²] across airspeed range
Maximum Range	**529.18** [miles] @ **75.98** [kts]
Maximum Endurance	**4.76** [hrs] @ **56.36** [kts]

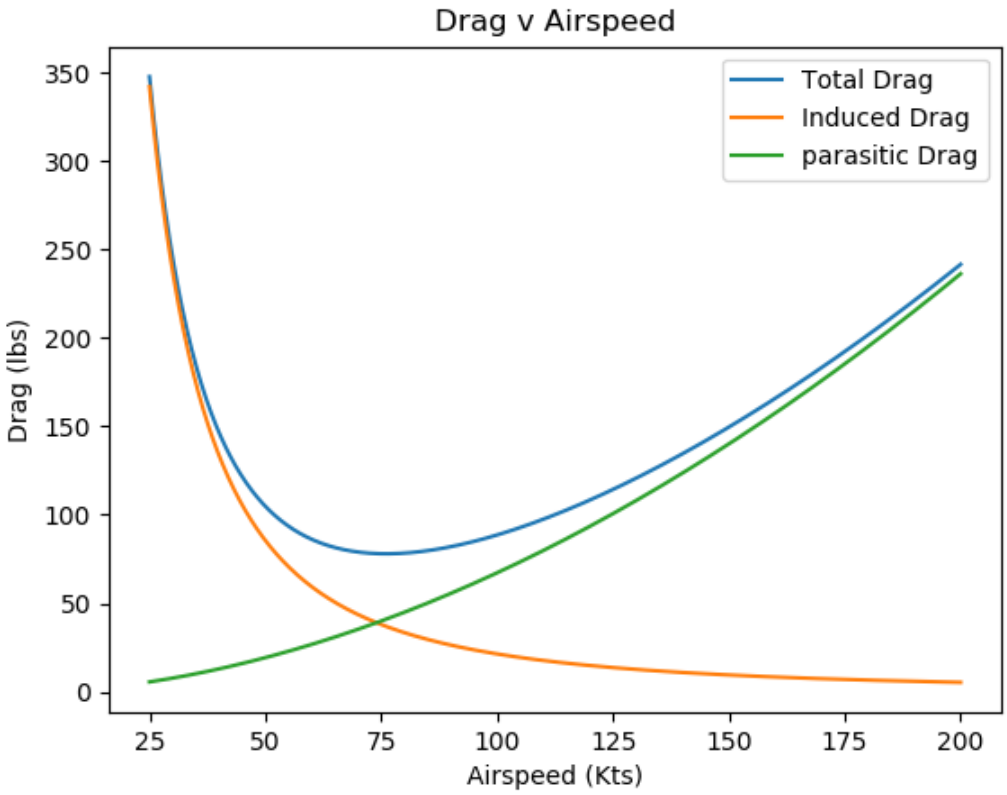

Figure 5. Plot showing total, induced, and parasitic drags versus airspeed.

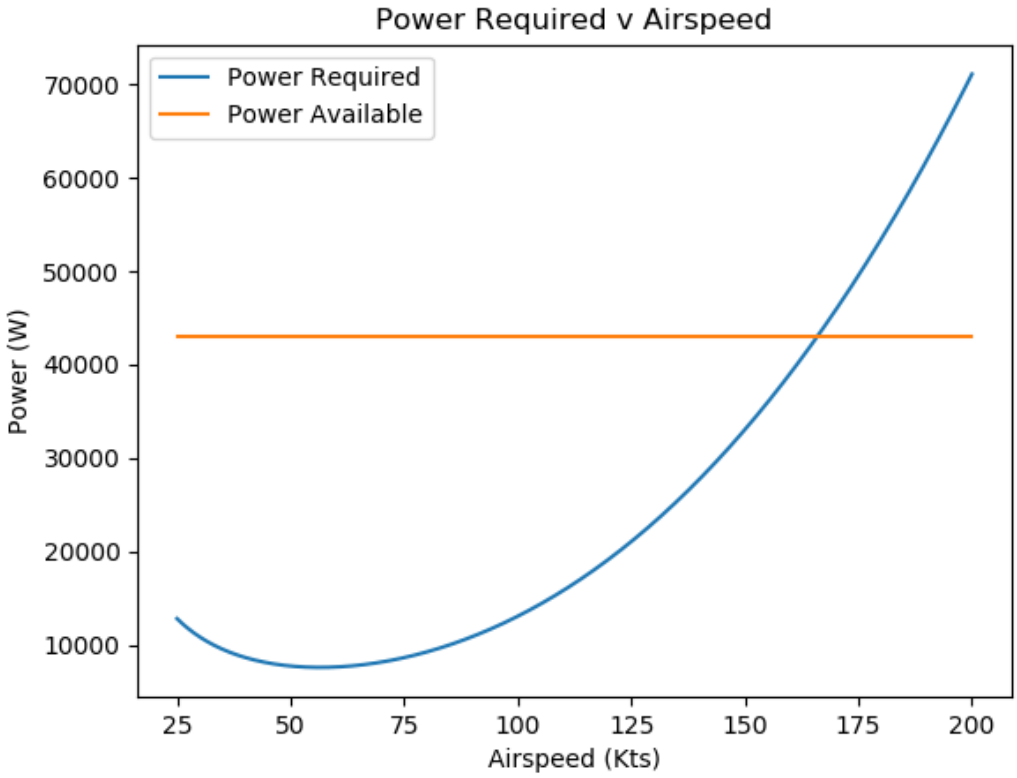

Figure 6. Plot showing power required and power available from the propellor motor versus airspeed.

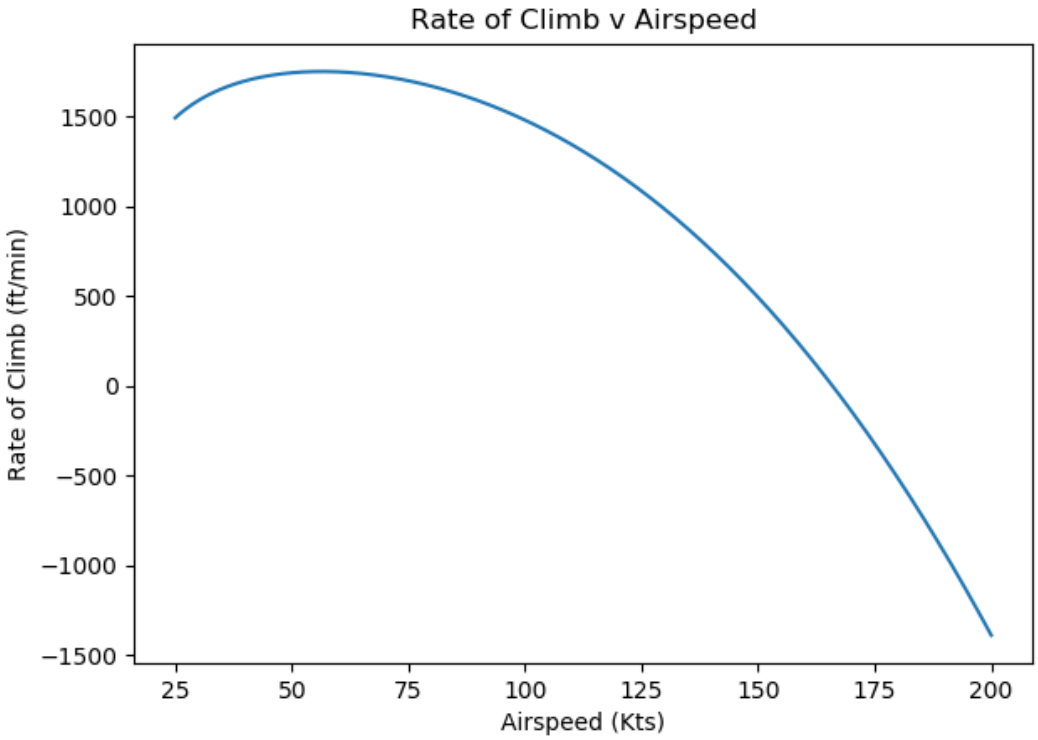

Figure 7. Plot showing rate of climb versus airspeed; the graph becomes negative one the power required is greater than the power available (shown in Figure 6).

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/55
0#
    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 occur
s 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_k
ts[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.w
iley.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 occur
s 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_k
ts[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()
```