Cessna 182RG Drag and Performance Analysis

EAE 130A - Senior Design

Jack Comey

February 1, 2021

Contents

1 Engine Performance

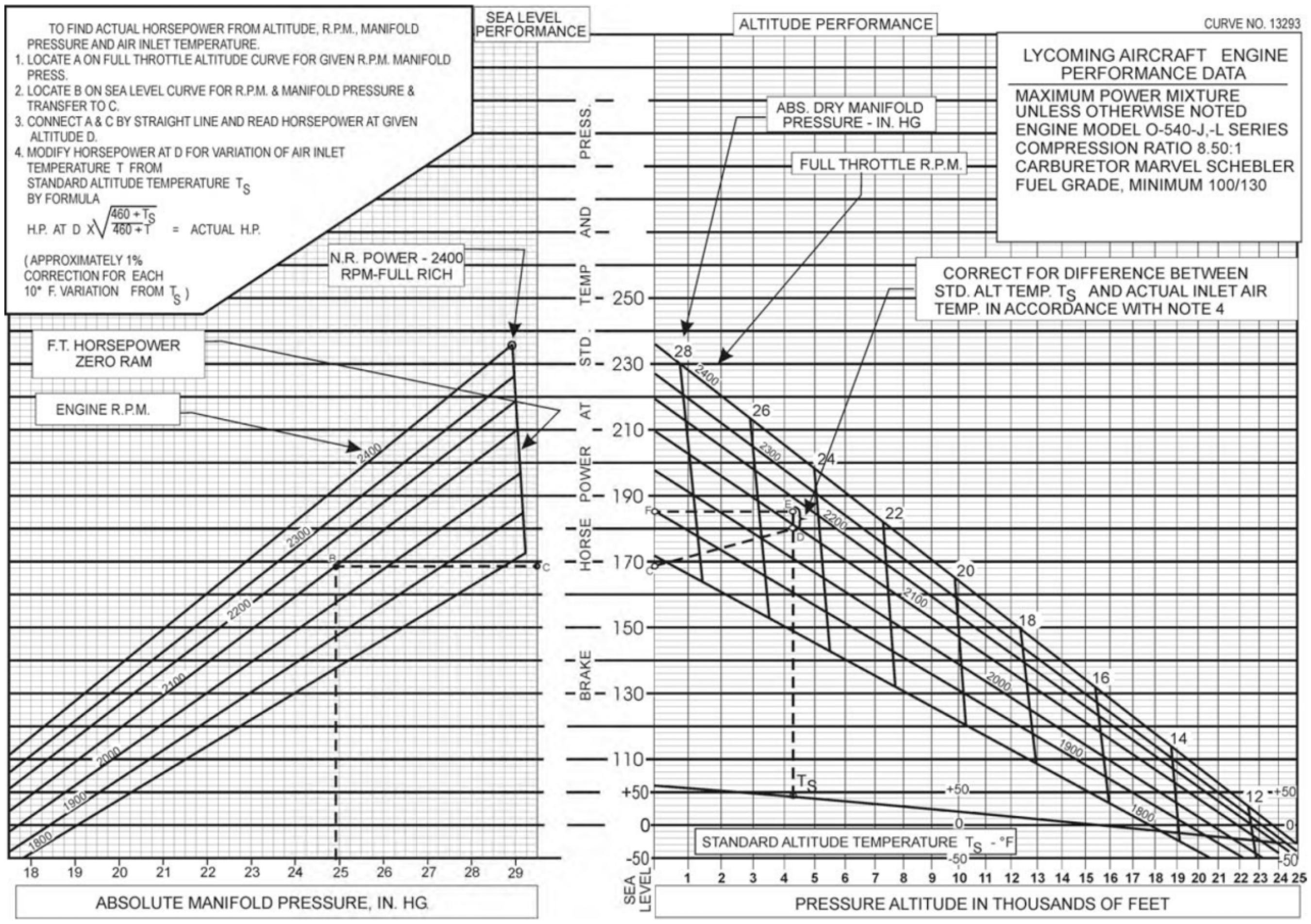

Figure 1: Lycoming O-540-J3C5D performance charts. Lycoming. "Operator's Manual Lycoming O-540, IO-540 Series". In: (2006)

Using the figure above, power available from the Lycoming O-540-J3C5D at 8,000 ft and full throttle (2400 RPM) is calculated and reported in the table below:

Table 1: Lycoming O-540-J3C5D Operating Properties

Property	Value
Altitude	8000 ft
RPM	2400
Engine Power	177 BHP
% Max. Rated Power	75.139%

2 Propeller Design

Using the BEM propeller design procedure as listed in the provided documents[1], design properties for a three-bladed propeller are calculated for the operating conditions and reported below:

Table 2: Propeller Design Properties

Property	Value	Units
B	3	—
D	6.75	ft
$\beta_{0.75R}$	10.08°	Degree
AF	759.84	—
$C_{L_{design}}$	0.4	—
J	0.9751	—
η_P	0.896	—
C_T	0.0533	—
C_Q	0.00924	—
C_P	0.0580	—

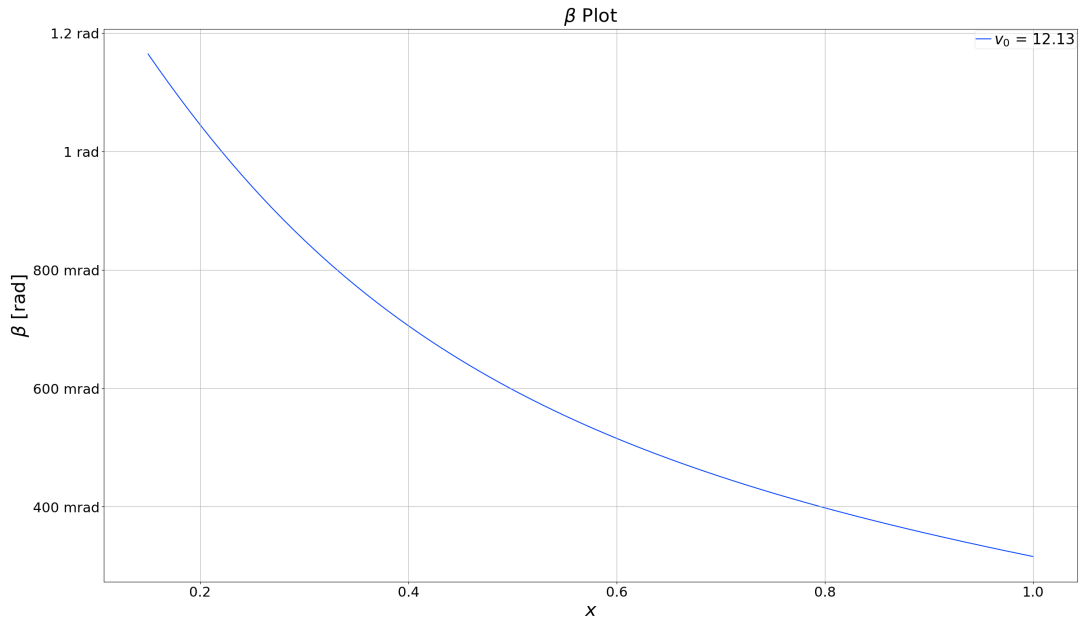

Figure 2: Propeller β distribution as a function of fraction of propeller length

[1]C.P. van Dam. “Simple Blade Element Momentum Theory”. In: (2021).

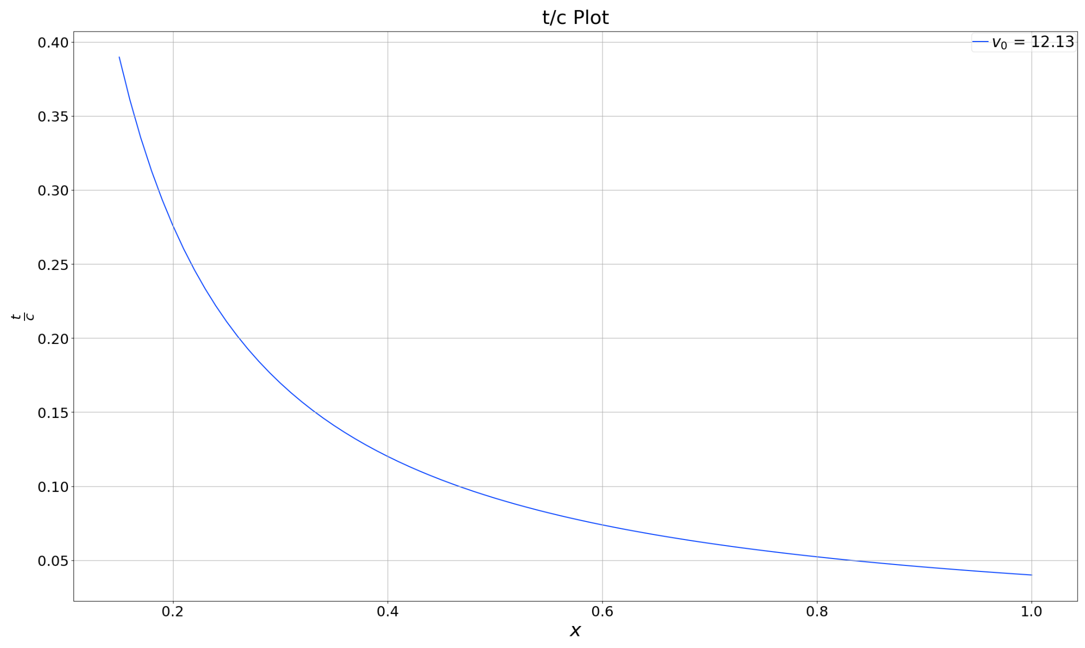

Figure 3: Propeller $\frac{t}{c}$ ratio as a function of fraction of propeller length

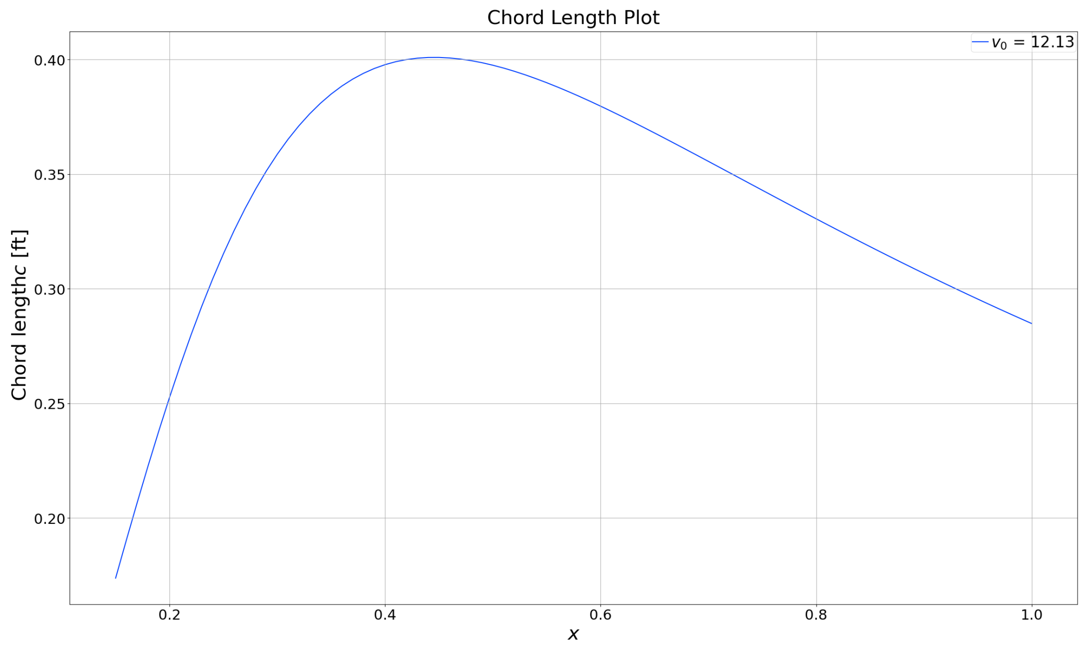

Figure 4: Propeller chord length as a function of fraction of propeller length

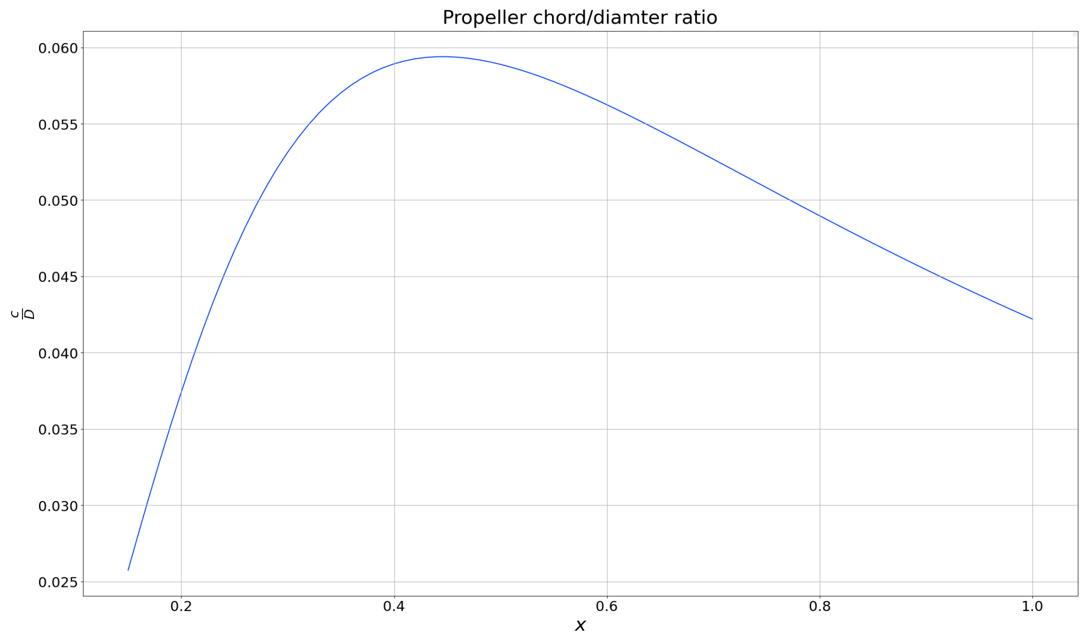

Figure 5: Propeller chord length over propeller diameter as a function of fraction of propeller length

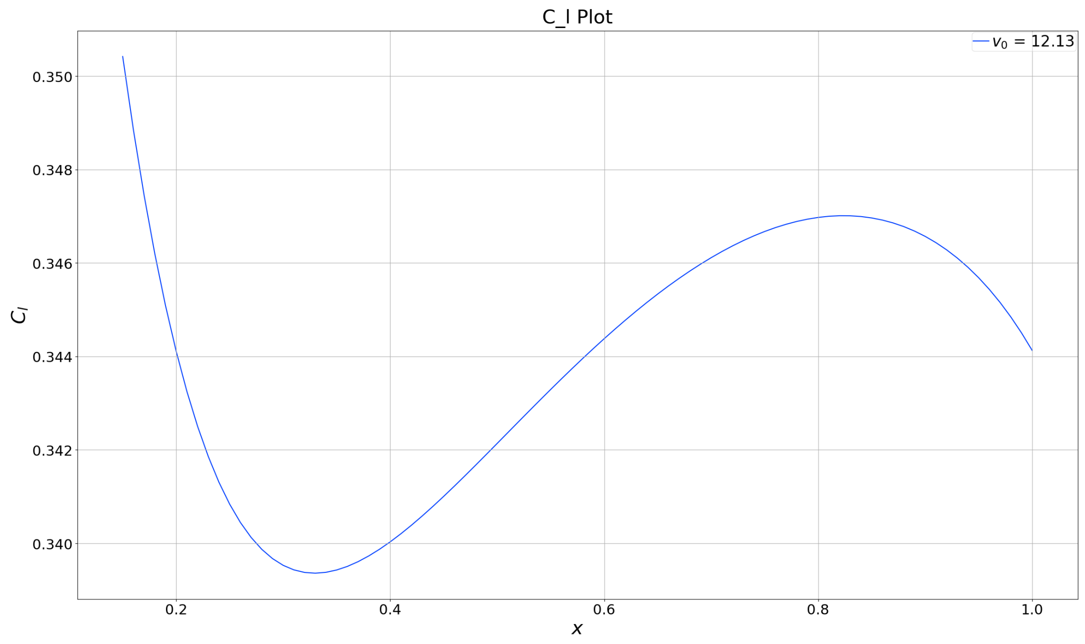

Figure 6: Propeller C_l as a function of fraction of propeller length

3 Propeller Analysis

Performance for the propeller designed in the previous section is to be analyzed for a variety of values for advance ratio J. Using a constant engine RPM value of 2400, airspeed is varied from 55-160 $KTAS$, plotted and reported below:

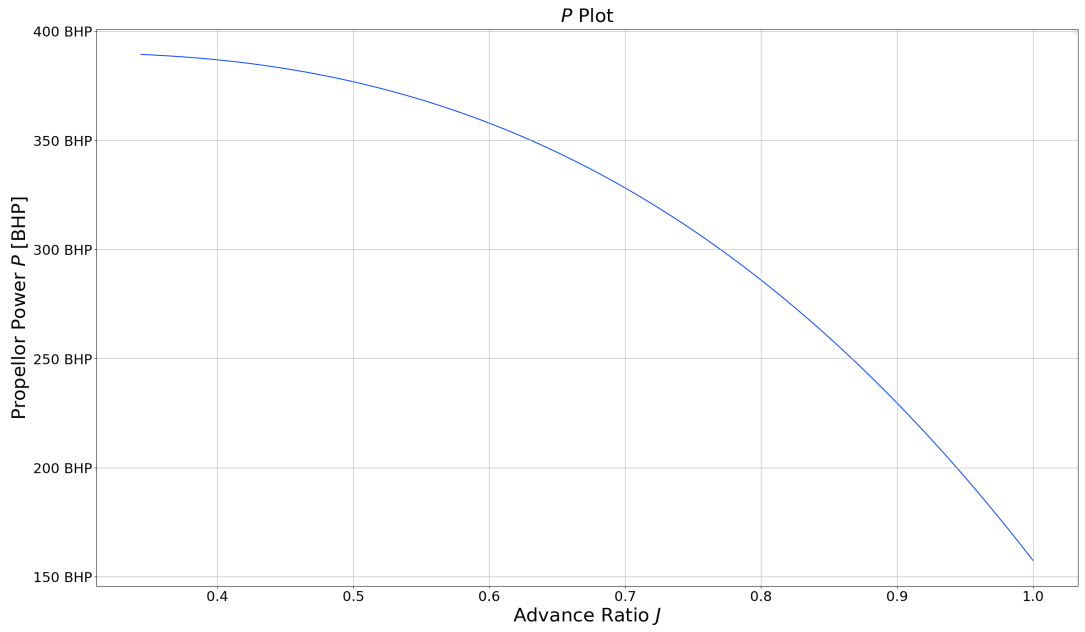

Figure 7: Propeller power P as a function of advance ratio J

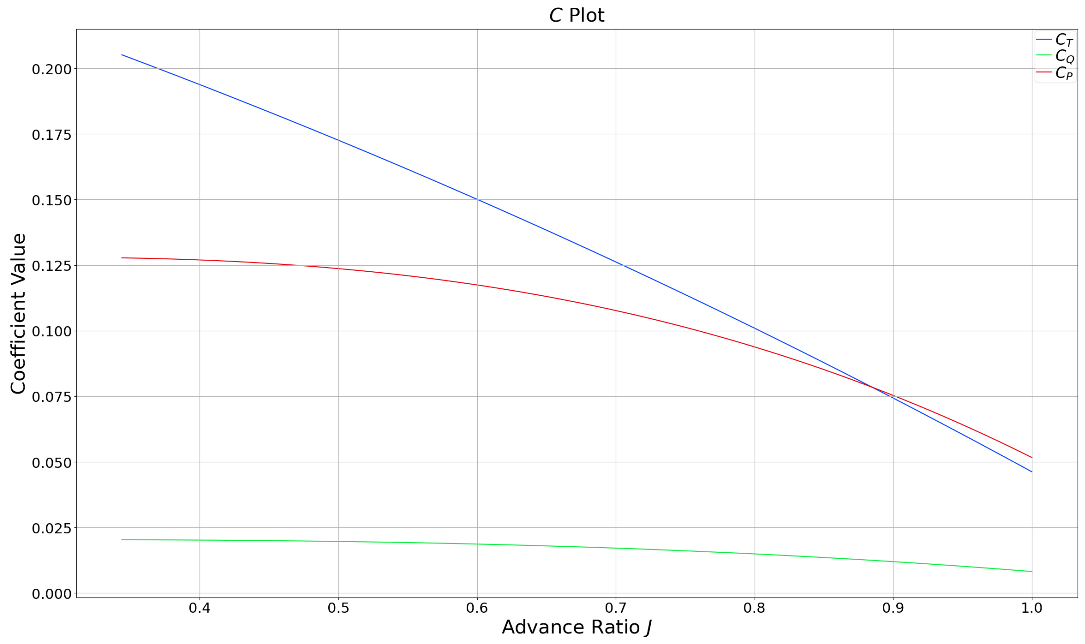

Figure 8: Propeller performance coefficients as a function of advance ratio J

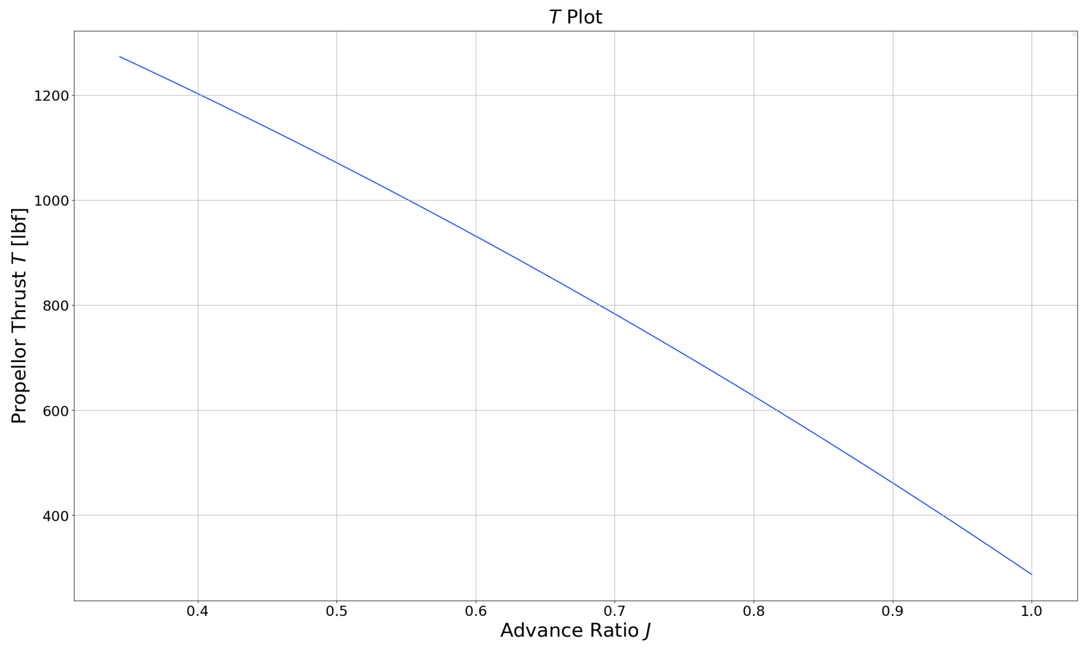

Figure 9: Propeller thrust T as a function of advance ratio J

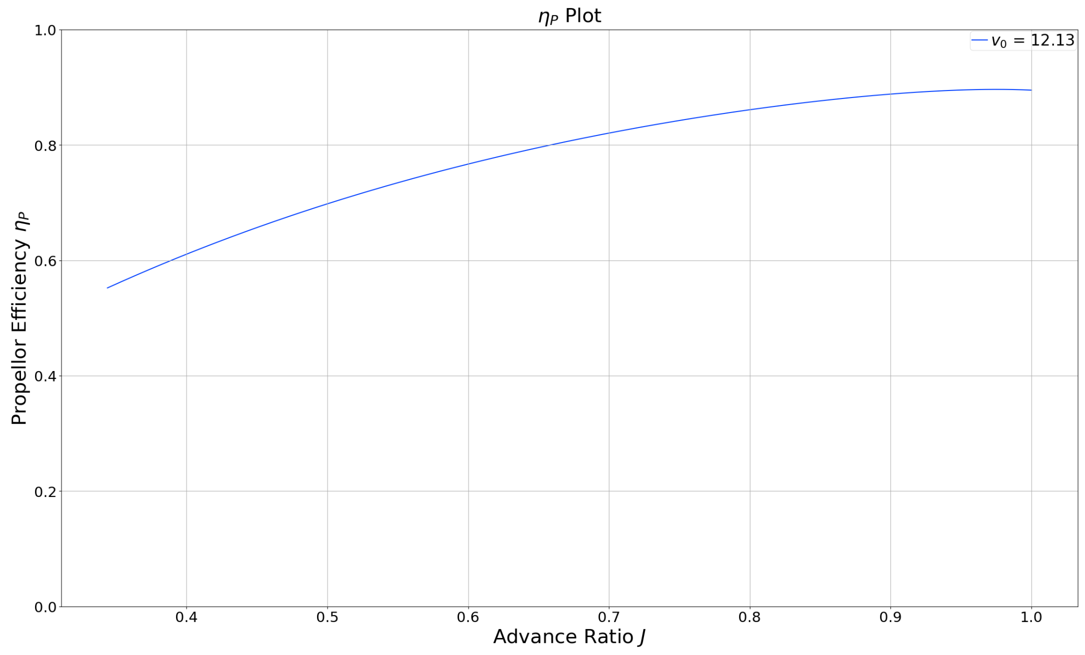

Figure 10: Propeller efficiency η_P as a function of advance ratio J

4 Propeller Variable Pitch

Assuming a variable pitch propeller, efficiency, power, and thrust values can be calculated for a varying $\Delta\beta$ values:

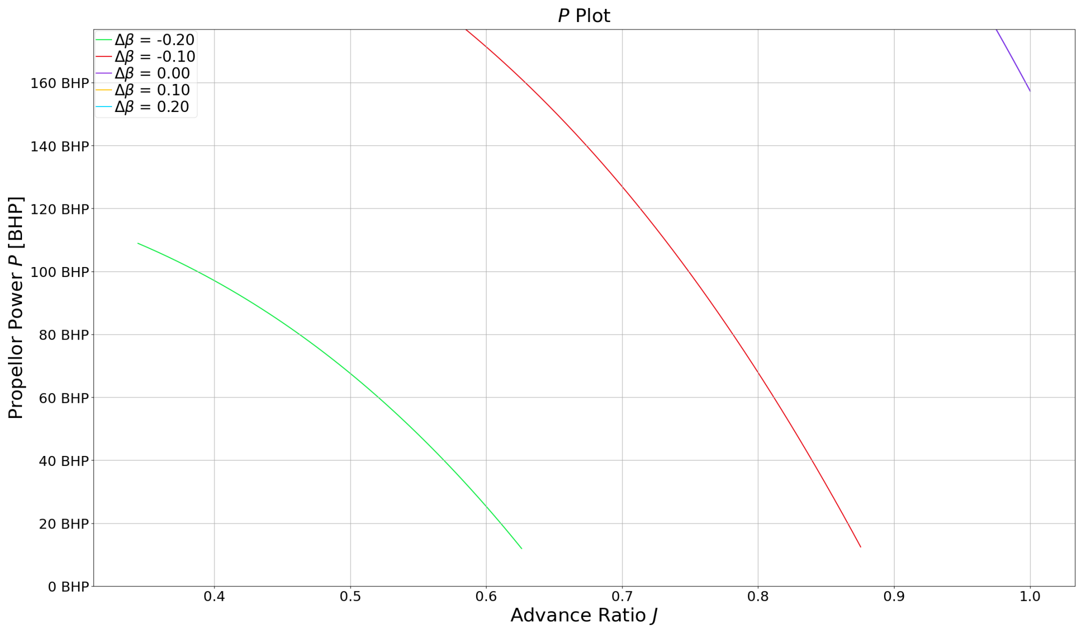

Figure 11: Propeller power P as a function of advance ratio J with varying $\Delta\beta$ values. Limited view to maximum engine power at operating conditions

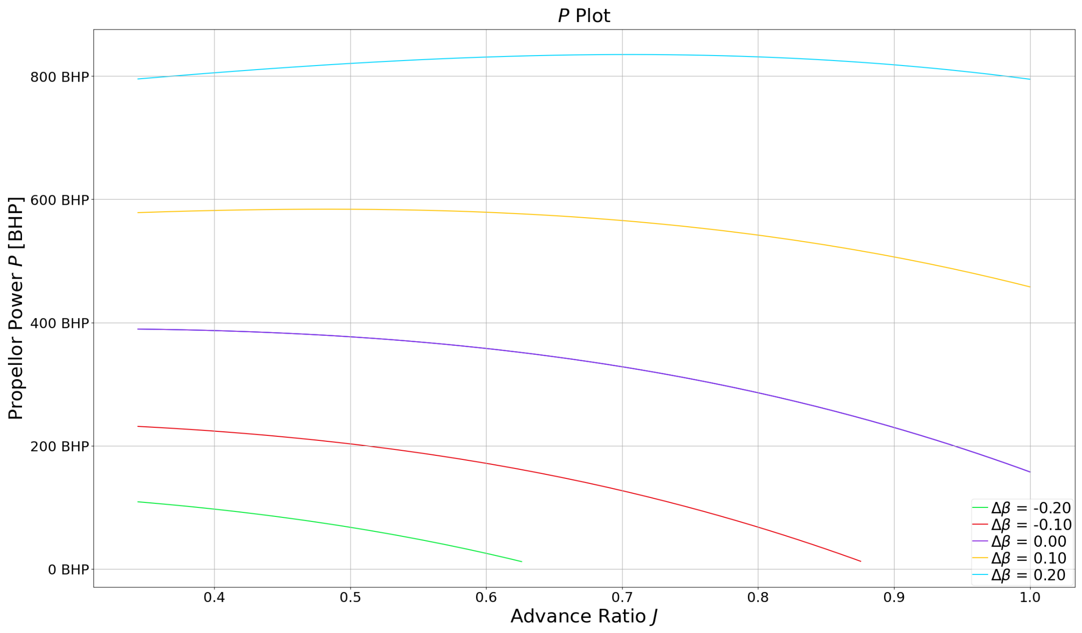

Figure 12: Propeller power P as a function of advance ratio J with varying $\Delta\beta$ values. Unlimited view of values

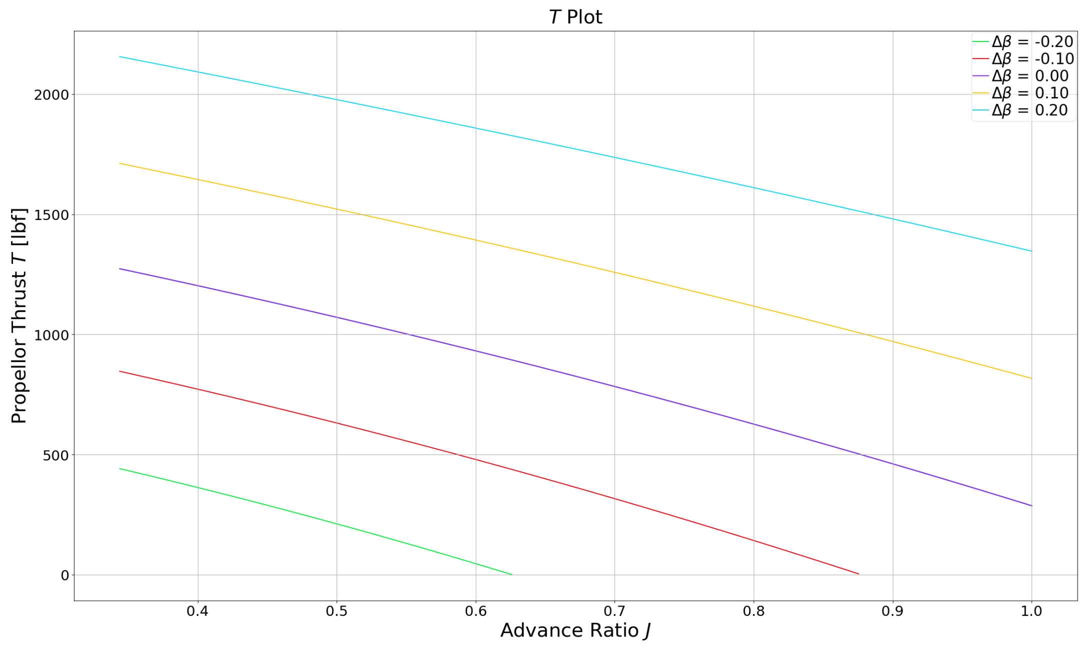

Figure 13: Propeller thrust T as a function of advance ratio J with varying $\Delta\beta$ values

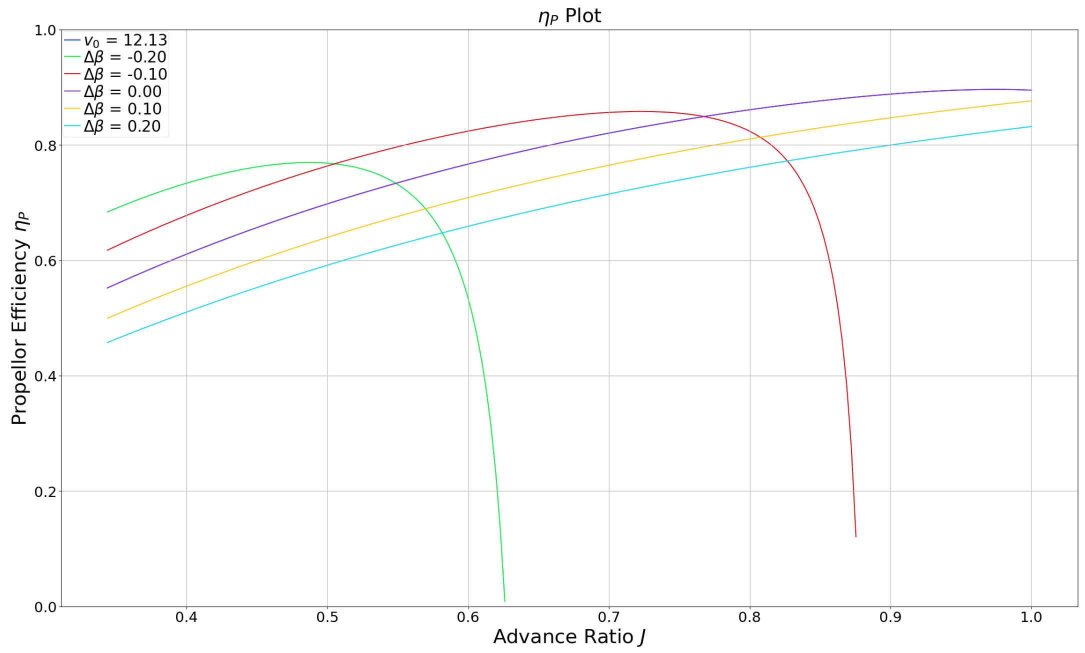

Figure 14: Propeller efficiency η_P as a function of advance ratio J with varying $\Delta\beta$ values

5 Discussion

The propeller design procedure used in this report is the blade element momentum theory (BEM), which discretizes a propeller into smaller segments. Flight conditions and some basic propeller properties, such the number of blades, $\frac{t}{c}(x)$, and $\alpha_0(x)$, were given as part of the problem statement. An arbitrary propeller diameter D was selected at 6.75 ft, matching an example given in the summary document. An arbitrary, constant $C_{l_{des}}$ distribution was selected for the propeller as an initial design condition. Using these input properties, all other propeller properties were derived. A propeller chord length distribution, $c(x)$, was derived from the $C_{l_{des}}$, as well as propeller pitch distribution $\beta(x)$. These derived geometric properties wholly define the propeller, and can be used to analyze propeller performance under various flight conditions.

At the design condition,the propeller is at peak performance. The efficiency of the propeller η_P is at its peak at the design advance ratio J. Operating at a lower flight speeds significantly increases the propeller power P and thrust T, but does so well beyond what the engine is able to provide. A lower engine RPM will allow the propeller to continue to operate, but doing so would place the propeller back into the optimal J value range. The propeller therefore operates poorly at off-design conditions, but is highly efficient at J values close to the design condition.

Poor performance at off-design conditions may be compensated for by using a variable pitch mechanism. Note the plot of propeller efficiency in Figure 14. The effect of a constant varied $\Delta\beta$ value shifts peak propeller efficiency to higher or lower J values. While the propeller trends to be more efficient at higher J values as $\Delta\beta$ increases, a negative $\Delta\beta$ value would allow the Cessna 182-RG to operate at a lower speeds. Given the potential use, it is recommended that the Cessna 182-RG be fitted with a variable pitch mechanism.

6 Appendix

6.1 Bibliography

References

[1] C.P. van Dam. “Simple Blade Element Momentum Theory”. In: (2021).

[2] Lycoming. “Operator’s Manual Lycoming O-540, IO-540 Series”. In: (2006).

6.2 Python Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 27 15:16:47 2021

@author: jack
"""

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import copy
import time
import pandas as pd
plt.style.use("default")
plt.style.use("seaborn-bright")


params={#FONT SIZES
    'axes.labelsize':30,#Axis Labels
    'axes.titlesize':30,#Title
    'font.size':28,#Textbox
    'xtick.labelsize':22,#Axis tick labels
    'ytick.labelsize':22,#Axis tick labels
    'legend.fontsize':24,#Legend font size
    'font.family':'sans-serif',
    'font.fantasy':'xkcd',
    'font.sans-serif':'Helvetica',
    'font.monospace':'Courier',
    #AXIS PROPERTIES
    'axes.titlepad':2*6.0,#title spacing from axis
    'axes.grid':True,#grid on plot
    'figure.figsize':(28,16),#square plots
    # 'savefig.bbox':'tight',#reduce whitespace in saved figures#LEGEND PROPERTIES
    'legend.framealpha':0.5,
    'legend.fancybox':True,
    'legend.frameon':True,
    'legend.numpoints':1,
    'legend.scatterpoints':1,
    'legend.borderpad':0.1,
    'legend.borderaxespad':0.1,
    'legend.handletextpad':0.2,
    'legend.handlelength':1.0,
    'legend.labelspacing':0,}
mpl.rcParams.update(params)

def plothusly(ax, x, y, *, xtitle='', ytitle='',
              datalabel='', title='', linestyle='-',
              marker=''):
    """
    A little function to make graphing less of a pain.
    Creates a plot with titles and axis labels.
    Adds a new line to a blank figure and labels it.

    Parameters
    ----------
    ax : The graph object
    x : X axis data
    y : Y axis data
    xtitle : Optional x axis data title. The default is ''.
    ytitle : Optional y axis data title. The default is ''.
    datalabel : Optional label for data. The default is ''.
    title : Graph Title. The default is ''.

    Returns
    -------
    out : Resultant graph.

    """

    ax.set_xlabel(xtitle)
    ax.set_ylabel(ytitle)
    ax.set_title(title)
    out = ax.plot(x, y, zorder=1, label=datalabel, linestyle = linestyle,
                  marker = marker)
    ax.grid(True)
    ax.legend(loc='best')
    return out


def plothus(ax, x, y, *, datalabel='', linestyle = '-',
            marker = ''):
    """
    A little function to make graphing less of a pain

    Adds a new line to a blank figure and labels it
    """
    out = ax.plot(x, y, zorder=1, label=datalabel, linestyle = linestyle,
                  marker = marker)
    ax.legend(loc='best')

    return out

class Propeller:

    def __init__(self, B, D, c_x, beta_x, alpha_0_x, P_eng, x):
        self.B = B
        self.D = D
        self.R = D/2
        self.c_x = c_x
        self.P_eng = P_eng


        # FIX ME
        self.beta = np.zeros_like(x)
        self.alpha_0 = np.zeros_like(x)

    def GetSigma(self, x):
        return self.B * self.c_x(x) / (np.pi * self.R)

    # def beta(self, i):
    #     return self.beta[i]

    # def alpha_0(self, i):
    #     return self.alpha_0[i]


class FlightConditions:

    def __init__(self, V, rho, mu, a, RPM):
        self.V = KTAS2FPS(V)
        self.rho = rho
        self.mu = mu
        self.a = a
        self.RPM = RPM
        self.n = RPM2n(RPM)


#%%##########################

def SectionalLift(M):
    m_0 = np.zeros_like(M)
    for n, m in enumerate(M):
        if m >= 0.9:
            m_0[n] = (2 * np.pi)
            m_0[n] /= (1 - 0.9**2)**0.5
        else:
            m_0[n] = (2 * np.pi)
            m_0[n] /= (1 - m**2)**0.5
    return m_0

def GetSigma(atmos, prop):
    return prop.B * prop.c / (np.pi * prop.R)

def LocalM(atmos, prop):
    M = prop.V_R/atmos.a
    return M

def LocalV(atmos, omega, r):
    V = (atmos.V**2 + (omega*r)**2)**0.5
    return V

def KTAS2FPS(v):
    return 1.68781 * v

def RPM2n(RPM):
    RPS = RPM / 60
    return RPS

def GetV_R(prop, phi, x):
    r = x * prop.R
    V_R = 2 * np.pi * atmos.n * r
    V_R /= np.cos(phi)
    return V_R

def GetJ(atmos, prop):
    J = atmos.V / (atmos.n * prop.D)
    return J

def GetPhi(J, x):
    phi = np.arctan2(J, np.pi*x)
    return phi

def GetTheta(atmos, prop):
    A_0 = -1 * (prop.beta - prop.phi - prop.alpha_0) * prop.sigma * prop.m_0 / (8 * prop.x)

    A_1 = (atmos.V / prop.V_R)
    A_1 += ((prop.beta - prop.phi - prop.alpha_0) * np.tan(prop.phi) + 1) * prop.sigma * prop.m_0

    A_2 = np.cos(prop.phi) - prop.sigma * np.tan(prop.phi) * prop.m_0 / (8 * prop.x)

    theta = -A_1 + (A_1**2 - 4*A_2*A_0)**0.5
    theta /= 2*A_2
    return theta

def GetLambda_T(c_l, c_d, phi, theta):
    lambda_t = c_l * np.cos(phi+theta) - c_d * np.sin(phi+theta)
    lambda_t /= (np.cos(phi))**2
    return lambda_t

def GetLambda_Q(c_l, c_d, phi, theta):
    lambda_q = c_l * np.sin(phi+theta) + c_d * np.cos(phi+theta)
    lambda_q /= (np.cos(phi))**2
    return lambda_q

def GetAlpha(beta, phi, theta):
    return beta - phi - theta

def GetAlphaDesign(atmos, prop):
    return (prop.c_l / prop.m_0) + prop.alpha_0

def Getc_l(atmos, prop):
    c_l = prop.m_0 * (prop.alpha - prop.alpha_0)
    return c_l

def Getc_d(c_l):
    c_d_min = 0.0095
    k = 0.0040
    c_l_min = 0.2

    return c_d_min + k*(c_l - c_l_min)**2

def GetThetaBetz(atmos, prop):
    a = atmos.V + atmos.v_0
    b = 2 * np.pi * atmos.n *prop.r
    theta = np.arctan2(a, b) - prop.phi
    return theta

def Getc_lDesign(phi, theta, sigma, x):
    c_l_des = 8*x*np.cos(phi) * np.tan(theta+phi)
    c_l_des /= sigma
    return c_l_des

def GetsigmaDesign(atmos, prop):
    sigma = 8 * prop.x * prop.theta * np.cos(prop.phi) * np.tan(prop.theta + prop.phi)
    sigma /= prop.c_l
    return sigma

def PropellorAnalysis(atmos, prop):
    prop.J = GetJ(atmos, prop)
    prop.omega = 2 * np.pi * atmos.n
    prop.phi = GetPhi(prop.J, prop.x)
    prop.V_R = LocalV(atmos, prop.omega, prop.r)
    prop.M = LocalM(atmos, prop)
    prop.m_0 = SectionalLift(prop.M)
    prop.sigma = GetSigma(atmos, prop)
    prop.theta = GetTheta(atmos, prop)
    prop.alpha = GetAlpha(prop.beta, prop.phi, prop.theta)
    prop.c_l = Getc_l(atmos, prop)
    prop.c_d = Getc_d(prop.c_l)
    prop.lambda_t = GetLambda_T(prop.c_l, prop.c_d, prop.phi, prop.theta)
    prop.lambda_q = GetLambda_Q(prop.c_l, prop.c_d, prop.phi, prop.theta)
    prop["dCTdx"] = prop.sigma * (np.pi**3 * prop.x**2) * prop.lambda_t / 8
    prop["dCQdx"] = prop.sigma * (np.pi**3 * prop.x**3) * prop.lambda_q / 16

    C_T = np.trapz(prop.dCTdx, prop.x)
    C_Q = np.trapz(prop.dCQdx, prop.x)
    C_P = 2 * np.pi * C_Q


    T = C_T * atmos.rho * atmos.n**2 * prop.D[0]**4
    Q = C_Q * atmos.rho * atmos.n**2 * prop.D[0]**5
    P = C_P * atmos.rho * atmos.n**3 * prop.D[0]**5 / 550

    eta_P = prop.J[0] * C_T / C_P

    J = prop.J[0]

    AF = (10E5 / 16) * np.trapz(prop.c * prop.x**3 / prop.D, prop.x)
    C_L = 4 * np.trapz(prop.c_l*prop.x**3, prop.x)
    prop["p"]= 2 * np.pi * np.tan(prop.beta)
    p75 = np.interp(0.75, prop.x, prop.p)
    beta75 = np.arctan2(4*p75, 3*np.pi*prop.D[0])

    new_row = {"C_T": C_T,
               "C_Q": C_Q,
               "C_P": C_P,
               "J": J,
               "T": T,
               "Q": Q,
               "P": P,
               "eta_P": eta_P,
               "AF": AF,
               "C_L": C_L,
               "beta75": beta75}

    return new_row
```

```
def PropellorDesign(atmos, prop):
    repeat = True

    while repeat==True:
        prop["J"] = GetJ(atmos, prop)
        prop["omega"] = 2 * np.pi * atmos.n
        prop["phi"] = GetPhi(prop.J, prop.x)
        prop["V_R"] = LocalV(atmos, prop.omega, prop.r)
        prop["M"] = LocalM(atmos, prop)
        prop["m_0"] = SectionalLift(prop.M)
        prop["theta"] = GetThetaBetz(atmos, prop)
        prop["sigma"] = GetsigmaDesign(atmos, prop)
        prop["c"] = prop.sigma * np.pi * prop.R / prop.B
        prop["alpha"] = GetAlphaDesign(atmos, prop)
        prop["beta"] = prop.alpha + prop.phi + prop.theta
        prop["c_d"] = Getc_d(prop.c_l)
        prop["lambda_t"] = GetLambda_T(prop.c_l, prop.c_d, prop.phi, prop.theta)
        prop["lambda_q"] = GetLambda_Q(prop.c_l, prop.c_d, prop.phi, prop.theta)
        prop["dCTdx"] = prop.sigma * (np.pi**3 * prop.x**2) * prop.lambda_t / 8
        prop["dCQdx"] = prop.sigma * (np.pi**3 * prop.x**3) * prop.lambda_q / 16

        # prop, C_T, C_Q, C_P, eta_P = PropellorAnalysis(atmos, prop)

        C_T = np.trapz(prop.dCTdx, prop.x)
        C_Q = np.trapz(prop.dCQdx, prop.x)
        C_P = 2 * np.pi * C_Q

        T = C_T * atmos.rho * atmos.n**2 * prop.D[0]**4
        Q = C_Q * atmos.rho * atmos.n**2 * prop.D[0]**5
        P = C_P * atmos.rho * atmos.n**3 * prop.D[0]**5 / 550

        eta_P = prop.J[0] * C_T / C_P
        # print(eta_P)


        if abs(atmos.P_eng - P) <= 0.1:
            print("break")
            repeat = False
        elif atmos.P_eng >= P:
            atmos.v_0 += 0.001
        elif atmos.P_eng < P:
            atmos.v_0 -= 0.001
        else:
            pass

        # print(f"P = {P:.2f} hp")
    # print(f"eta_p = {eta_P:.2f}")
    # print(f"C_T = {C_T:.2f}")
    # print(f"C_Q = {C_Q:.2f}")
    # print(f"C_P = {C_P:.2f}")


    return prop


def RK4(Fdot, y, t, delta_t):
    k1 = delta_t * Fdot(y, t)
    k2 = delta_t * Fdot(y + k1/2, t + 0.5*delta_t)
    k3 = delta_t * Fdot(y + k2/2, t + 0.5*delta_t)
    k4 = delta_t * Fdot(y + k3, t+delta_t)

    y_nplusone = y + (k1 + 2*k2 + 2*k3 + k4)/6

    return y_nplusone

def GetThrust(atmos, prop, C_T_list, x_list):
    C_T = np.trapz(C_T_list, x_list)
    T = C_T * atmos.rho * atmos.n**2 * prop.D**4
    return T

def GetTorque(atmos, prop, C_Q_list, x_list):
    C_Q = np.trapz(C_Q_list, x_list)
    Q = C_Q * atmos.rho * atmos.n**2 * prop.D**5
    return Q

def GetPower(atmos, prop, C_P):
    return C_P * atmos.rho * atmos.n**3 * prop.D**5




T_inf = 490.141  # R
Pres = 1571.90  # lbs/ft**2
rho = 0.00186850  # slug/ft**3
a = 1085.31  # ft/s
mu = 3.61710e-7  # lbf s /ft**2
V_knots = 156

RPM = 2400
atmos = FlightConditions(V_knots, rho, mu, a, RPM)

B = 3
```

```
D = 6.75  # ft
c_l = 0.4
atmos.P_eng =  177
num_el = 86
alpha_0 = np.deg2rad(-2)

x_list = np.linspace(0.15, 1, num_el)

prop = pd.DataFrame()

prop["x"] = x_list
prop["c_l"] = c_l * np.ones_like(x_list)
prop["t_over_c"] = 0.04 / (prop.x**1.2)
prop["D"] =  D * np.ones_like(x_list)
prop["R"] =  0.5 * prop.D
prop["r"] = prop.R * prop.x
prop["B"] = B * np.ones_like(x_list)
prop["alpha_0"] = alpha_0 * np.ones_like(x_list)



atmos.v_0 = 0.045*atmos.V

prop = PropellorDesign(atmos, prop)


column_names = ["C_T",
                "C_Q",
                "C_P",
                "J",
                "T",
                "Q",
                "P",
                "eta_P",
                "AF",
                "C_L",
                "beta75"]

prob2results = pd.DataFrame(columns = column_names)
prob2results = prob2results.append(PropellorAnalysis(atmos, prop), ignore_index=True)

print(prob2results.to_markdown())

mass_test = pd.DataFrame(columns=column_names)


V_list = KTAS2FPS(np.linspace(55, 160, 201))
for i, atmos.V in enumerate(V_list):
    mass_test = mass_test.append(PropellorAnalysis(atmos, prop), ignore_index=True)


ft_format = mpl.ticker.EngFormatter(unit="ft")
rad_format = mpl.ticker.EngFormatter(unit="rad")
pow_format = mpl.ticker.EngFormatter(unit="BHP")
thr_format = mpl.ticker.EngFormatter(unit="lbf")



# print(prop.head())
fig, cplot = plt.subplots()
fig, betaplot = plt.subplots()
fig, tcplot = plt.subplots()
fig, clplot = plt.subplots()
fig, cDplot = plt.subplots()
fig, etaplot = plt.subplots()
plt.ylim([0, 1])
fig, powerplot = plt.subplots()
# plt.ylim([0, 177])
fig, coeffplot = plt.subplots()
fig, thrustplot = plt.subplots()

# cplot.yaxis.set_major_formatter(ft_format)
betaplot.yaxis.set_major_formatter(rad_format)
powerplot.yaxis.set_major_formatter(pow_format)
# thrustplot.yaxis.set_major_formatter(thr_format)


plothusly(cplot, prop.x, prop.c,
        xtitle=r"$x$",
        ytitle=r" Chord length$c$ [ft]",
        title="Chord Length Plot",
        datalabel=fr"$v_0$ = {atmos.v_0:.2f}")
plothusly(betaplot, prop.x, prop.beta,
        xtitle=r"$x$",
        ytitle=r"$\beta$ [rad]",
        title=r"$\beta$ Plot",
        # datalabel=fr"£v_0£ = {atmos.v_0:.2f}"
        )
plothusly(tcplot, prop.x, prop.t_over_c,
        xtitle=r"$x$",
        ytitle=r"$\frac{t}{c}$",
        title="t/c Plot",
        datalabel=fr"$v_0$ = {atmos.v_0:.2f}")
plothusly(clplot, prop.x, prop.c_l,
        xtitle=r"$x$",
        ytitle=r"$C_l$",
        title="C_l Plot",
        datalabel=fr"$v_0$ = {atmos.v_0:.2f}")
plothusly(cDplot, prop.x, prop.c/prop.D,
          xtitle=r"$x$",
          ytitle=r"$\frac{c}{D}$",
          title=r"Propeller chord/diamter ratio",)



plothusly(etaplot,
          mass_test.J,
          mass_test.eta_P,
          xtitle=r"Advance Ratio $J$",
          ytitle=r"Propellor Efficiency $\eta_P$",
          title=r"$\eta_P$ Plot",
          datalabel=fr"$v_0$ = {atmos.v_0:.2f}")
plothusly(powerplot,
          mass_test.J,
          mass_test.P,
          xtitle=r"Advance Ratio $J$",
          ytitle=r"Propellor Power $P$",
          title=r"$P$ Plot",
          # datalabel=fr"£v_0£ = {atmos.v_0:.2f}"
          )
plothusly(coeffplot,
          mass_test.J,
          mass_test.C_T,
          xtitle=r"Advance Ratio $J$",
          ytitle=r"Coefficient Value",
          title=r"$C$ Plot",
          datalabel=fr"$C_T$")
plothus(coeffplot,
        mass_test.J,
        mass_test.C_Q,
        datalabel=r"$C_Q$")
plothus(coeffplot,
        mass_test.J,
        mass_test.C_P,
        datalabel=r"$C_P$")

plothusly(thrustplot,
          mass_test.J,
          mass_test["T"],
          xtitle=r"Advance Ratio $J$",
          ytitle=r"Propellor Thrust $T$",
          title=r"$T$ Plot",
          # datalabel=fr"£v_0£ = {atmos.v_0:.2f}"
          )
```

```
#%%##########################


delta_beta_list = np.linspace(-0.2, 0.2, 5)

for delta_beta in delta_beta_list:
    prop2 = copy.deepcopy(prop)
    prop2.beta += delta_beta
    mass_test = pd.DataFrame(columns=column_names)
    for i, atmos.V in enumerate(V_list):
        new_row = PropellorAnalysis(atmos, prop2)
        if new_row["eta_P"] <= 0:
            break
        else:
            mass_test = mass_test.append(PropellorAnalysis(atmos, prop2), ignore_index=True)

    plothus(etaplot, mass_test.J, mass_test.eta_P, datalabel=fr"$\Delta\beta$ = {delta_beta:.2f}")
    plothus(thrustplot, mass_test.J, mass_test["T"], datalabel=fr"$\Delta\beta$ = {delta_beta:.2f
    plothus(powerplot, mass_test.J, mass_test.P, datalabel=fr"$\Delta\beta$ = {delta_beta:.2f}")
```