project3_functions

December 27, 2022

[1]:
```
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
from ipynb.fs.full.project2 import properties
```

1 Thermodynamic Specifications

2 Aerodynamic Losses

In this section We are going to Calculate Aerodynamic losses. Aerodynamic losses are divided to below parts:

2.1 Profile Loss

[2]:
```
def i_sr(Beta2, landa):
    A = 61.8-(1.6-Beta2/165)*Beta2
    B = 71.9-1.69*Beta2
    C = 7.8-(0.28-Beta2/320)*Beta2
    D = 14.2-(0.16+Beta2/160)*Beta2
    i_s0 = 20-(landa+1)/0.11

    if Beta2 <= 40:
        i = i_s0+A-B*landa**2+C*landa**3+D*landa**4

    else:
        i = i_s0+((abs(i_sr(40, landa)-i_s0))*(abs(55-Beta2)))/15

    return i

def di_s(s_c, Beta2):
    X = s_c-0.75

    if s_c <= 0.8:
        di = -38*X-53.5*X**2-29*X**3
```

```
    else:
        di = 2.0374-(s_c-0.8)*(69.58-(Beta2/14.48)**3.1)

    return di

def K_inc(i, i_s):
    if (i/i_s) < -3:
        K = -1.39214-1.90738*(i/i_s)

    elif -3 <= (i/i_s) < 0:
        K = 1+0.52*(abs(i/i_s))**1.7

    elif 0 <= (i/i_s) < 1.7:
        K = 1+(i/i_s)**(2.3+0.5*(i/i_s))

    else:
        K = 6.23-9.8577*((i/i_s)-1.7)

    return K
```

[3]:

```
def K_M(s_Rc, M_w2):
    if M_w2 <= 0.6:
        K = 1

    elif 0.6 < M_w2 <= 1:
        K = 1+(1.65*(M_w2-0.6)+24*(M_w2-0.6)**2)*(s_Rc)**(3*(M_w2-0.6))

    return K
```

[4]:

```
def K_p(M_w1, M_w2):
    if M_w2 <= 0.2:
        K = 1

    else:
        K1 = 1-0.625*((M_w2-0.2)+abs(M_w2-0.2))
        K = 1-(1-K1)*(M_w1/M_w2)**2

    if K <= 0:
        M_tilda_w1 = ((M_w1+0.566)-abs(0.566-M_w1))/2
        M_tilda_w2 = ((M_w2+1)-abs(M_w2-1))/2
        X = 2*M_tilda_w1/(M_tilda_w1+M_tilda_w2+abs(M_tilda_w2-M_tilda_w1))
        K1 = 1-0.625*((M_w2-0.2)+abs(M_w2-0.2))
        K = 1-(1-K1)*X**2

    if K <= 0.5:
        print('K_p is not in the acceptable range: K_p > 0.5')
```

```
    return K
```

[5]:
```
def K_Re(Re_c, Re_e=0):
    if Re_e < 100:
        if 1*10**5 <= Re_c <= 5*10**5:
            K = 1

        elif Re_c < 1*10**5:
            K = ((1*10**5)/Re_c)**0.5

        elif Re_c > 5*10**5:
            K = (math.log(5*10**5, 10)/math.log(Re_c, 10))**2.58

    else:
        Re_r = 100*Chord/e

        if Re_c > Re_r:
            K = (math.log(5*10**5, 10)/math.log(Re_r, 10))**2.58

        else:
            K = K_Re(Re_c, 0)

    return K
```

[6]:
```
def Y_p1(Beta2, s_c):
    if Beta2 <= 27:
        A = 0.025+(27-Beta2)/530
    else:
        A = 0.025+(27-Beta2)/3085

    B = 0.1583-Beta2/1640
    C = 0.08*((Beta2/30)**2-1)

    if Beta2 <= 30:
        s_c_min = 0.46+Beta2/77
        X = (s_c)-(s_c_min)
        Y_p = A+B*X**2+C*X**3
    else:
        s_c_min = 0.614+Beta2/130
        X = (s_c)-(s_c_min)
        n = 1+Beta2/30
        Y_p = A+B*(abs(X))**(n)

    return Y_p
```

[7]:
```
def Y_p2(Beta2, s_c):
    A = 0.242-Beta2/151+(Beta2/127)**2
```

```python
    if Beta2 <= 30:
        B = 0.3+(30-Beta2)/50

    else:
        B = 0.3+(30-Beta2)/275

    C = 0.88-Beta2/42.4+(Beta2/72.8)**2
    s_c_min = 0.224+1.575*(Beta2/90)-(Beta2/90)**2
    X = (s_c)-(s_c_min)
    Y_p = A+B*X**2+C*X**3

    return Y_p
```

[8]:
```python
def Y_p(Beta1_prime, Beta1, Beta2, t_max_over_c, O, s, chord, M_w1, M_w2, Re_c,␣
↪Re_e, s_Rc):
    K_mod = 0.67       #modern engines
    landa = (90-Beta1_prime)/(90-Beta2)
    i = Beta1_prime-Beta1
    s_c = s/chord
    i_s = i_sr(Beta2, landa)+di_s(s_c, landa)

    t2_s = 0.02
    Beta_g = math.degrees(math.asin(O/s))
    dY_TE = ((t2_s)/(math.sin(math.radians(Beta_g))-t2_s))**2

    Y = K_mod*K_inc(i, i_s)*K_M(s_Rc, M_w2)*K_p(M_w1, M_w2)*K_Re(Re_c, Re_e)*\
    ((Y_p1(Beta2, s_c)+landa**2*(Y_p2(Beta2, s_c)-Y_p1(Beta2,␣
↪s_c)))*(t_max_over_c/0.2)**(landa)-dY_TE)

    dic = {
        'K_mod' : K_mod,
        'K_inc' : K_inc(i, i_s),
        'K_M'   : K_M(s_Rc, M_w2),
        'K_p'   : K_p(M_w1, M_w2),
        'K_Re'  : K_Re(Re_c, Re_e),
        'Y_p1'  : Y_p1(Beta2, s_c),
        'Y_p2'  : Y_p2(Beta2, s_c),
        'dY_TE' : dY_TE,
        'Y_p'   : Y
    }

    return dic
```

2.2 Secondary Loss

[9]:
```
def Y_s(K_Re, K_p, bx_h, Beta_1_prime, Beta1, Beta2, s_c, h_c):
    K_s = 1-((1-K_p)*(bx_h)**2)/(1+(bx_h)**2)

    if h_c >= 2:
        F_AR = 1/h_c

    else:
        F_AR = 0.5*((2/h_c)**0.7)

    Beta_m = 90-math.degrees(math.atan((1/math.tan(math.radians(Beta1))-1/math.
↪tan(math.radians(Beta2)))/2))
    C_L = 2*(s_c)*math.sin(math.radians(Beta_m))*(1/math.tan(math.
↪radians(Beta1))+1/math.tan(math.radians(Beta2)))
    Z = ((C_L/s_c)**2)*((math.sin(math.radians(Beta2)))**2)/((math.sin(math.
↪radians(Beta_m)))**3)
    Y_tilda_s = 0.0334*F_AR*Z*math.sin(math.radians(Beta2))/math.sin(math.
↪radians(Beta_1_prime))

    if Y_tilda_s > 0.365:
        print('Y_tilda_s is not in the acceptable range: Y_tilda_s <= 0.365 ')

    Y = K_Re*K_s*(((Y_tilda_s**2)/(1+7.5*Y_tilda_s**2))**0.5)

    return Y
```

2.3 Trailing Edge Loss

[10]:
```
def Y_TE(O, s, t2, rho, W2):
    Beta_g = math.degrees(math.asin(O/s))
    dP0 = 0.5*rho*(W2**2)*((t2)/(s*math.sin(math.radians(Beta_g))-t2))**2
    Y = dP0/(0.5*rho*(W2**2))

    return Y
```

2.4 Shock Loss

[11]:
```
def Y_sh(M_w1, M_w2):

    if M_w1 <= 0.4:
        X1 = 0

    else:
        X1 = M_w1-0.4
```

```python
    if M_w1 <= M_w2:
        X2 = 0

    else:
        X2 = M_w1/M_w2-1

    Y_tilda_sh = 0.8*X1**2+X2**2
    Y = ((Y_tilda_sh**2)/(1+Y_tilda_sh**2))**0.5

    if Y > 1:
        print('Y_sh is not in the acceptable range: Y <= 1')

    return Y
```

2.5 Supersonic Expansion Loss

[12]:
```python
def Y_EX(M_w2):

    if M_w2 > 1:
        Y = ((M_w2-1)/(M_w2))**2

    else:
        Y = 0

    return Y
```

2.6 Clearance Loss

[13]:
```python
def Y_CL(Beta1, Beta2, s_c, h_c, chord, delta):
    Beta_m = 90-math.degrees(math.atan((1/math.tan(math.radians(Beta1))-1/math.
 ↪tan(math.radians(Beta2)))/2))
    C_L = 2*(s_c)*math.sin(math.radians(Beta_m))*(1/math.tan(math.
 ↪radians(Beta1))+1/math.tan(math.radians(Beta2)))
    Z = ((C_L/s_c)**2)*((math.sin(math.radians(Beta2)))**2)/((math.sin(math.
 ↪radians(Beta_m)))**3)
    Y = 0.47*Z*(1/h_c)*(delta/chord)**0.78

    return Y
```

2.7 Lashing Wire Loss

[14]:
```
def Y_LW(N_LW, D_LW, Cm2, rho2, h, W2, Mu2):
    Re = (rho2*Cm2*D_LW)/Mu2

    if Re <= 5*10**5:
        C_D = 1

    else:
        C_D = 0.35

    Y = (N_LW*C_D*D_LW*(Cm2**2))/(h*W2**2)

    return Y
```

3 Parasistic Losses

3.0.1 Leakage Bypass Loss

[15]:
```
def dh0_leak(N_seal,t_seal, p_seal, r_seal, PR, rho, T, P1, P2, h01, h02,␣
 ↪delta_rotor, R, N_BH, D_BH, m_dot, r_seal_is_constant):
    C_t = (2.143*(math.log(N_seal)-1.464))/(N_seal-4.322)*((1-PR)**(0.375*PR))

    if N_seal <= 12:
        X1 = 15.1-0.05255*(math.e**(0.507*(12-N_seal)))
        X2 = 1.058+0.0218*N_seal

    else:
        X1 = 13.15+0.1625*N_seal
        X2 = 1.32

    delta_p = delta_rotor/p_seal

    C_c = 1+X1*(delta_p-X2*math.log(1+delta_p))/(1-X2)

    if delta_p > X2-1:
        print('delta_p is not in the acceptable range: delta_p <= X2-1')

    if r_seal_is_constant == 'r_seal is not constant':
        C_c = 1

    delta_t = delta_rotor/t_seal

    C_r = 1-1/(3+(54.3/(1+100*delta_t))**3.45)

    m_dot_seal = 2*math.pi*r_seal*delta_rotor*C_t*C_c*C_r*rho*((R*1000*T)**0.5)

```

```
    m_dot_BH = 1/8*N_BH*math.pi*(D_BH**2)*((2*(P1-P2)*101325/rho)**0.5)

    dh0 = (m_dot_seal+m_dot_BH)*(h01-h02)/m_dot

    dic = {'m_dot_seal[kg/sec]' : m_dot_seal, 'm_dot_BH[kg/sec]' : m_dot_BH,␣
↪'dh0_leak[kj/kg]' : dh0}

    return dic
```

3.1 Rotor Partial Admission Work

[16]:
```
def dh0_adm(epsilon, psi, rho2, h, D_m, U_m, b_x, N_active, psi_noz, C2, m_dot):
    dh0_w = 0.05*math.pi*rho2*D_m*(U_m**3)*h*(1-epsilon-psi/2)/m_dot/1000
    dh0_sec = 0.15*N_active*b_x*U_m*C2*psi_noz/(epsilon*D_m)/1000
    dh0 = dh0_w+dh0_sec

    return {'dh0_w[kj/kg]':dh0_w, 'dh0_sec[kj/kg]':dh0_sec, 'dh0[kj/kg]':dh0,}
```

3.2 Rotor Diaphragm-Disk Friction Work

[17]:
```
def dh0_DF(rho, r, Delta_r, e, Mu2, omega, m_dot):
    Re = (rho*omega*r**2)/Mu2

    C_M1 = 2*math.pi/(Delta_r*Re)
    C_M2 = 3.7*(Delta_r**0.1)/(Re**0.5)
    C_M3 = 0.08/((Delta_r**(1/6))*(Re**0.25))
    C_M4 = 0.102*(Delta_r**0.1)/(Re**0.2)
    C_M = max(C_M1, C_M2, C_M3, C_M4)

    A = np.array([C_M1, C_M2, C_M3, C_M4])

    dh01 = C_M*rho*(omega**3)*(r**5)/m_dot
    dh02 = 0

    if (e != 0) & (A.argmax() != 0) & (A.argmax() != 1) : # the flow must be␣
↪turbolent and e != 0
        C_Mr = (3.8*math.log(r/e,10)-2.4*(Delta_r**0.25))**(-2)
        Re_s = 1100*((e/r)**(-0.4))/(C_M**0.5)
        Re_r = 1100*(r/e)-6*10**6
        C_M = C_M+(C_Mr-C_M)*(math.log(Re/Re_s,10))/(math.log(Re_r/Re_s,10))
        dh02 = (C_M*rho*(omega**3)*(r**5)/m_dot)

    return {'dh01[kj/kg]' : dh01/1000, 'dh02[kj/kg]' : dh02/1000}
```

3.3 Clearance Gap Windage Loss

[18]:
```
def dh0_gap(r, rho, Delta, b_x, Delta_m, Mu2, omega, m_dot):

    Re = rho*r*omega*Delta/(2*Mu2)

    if Re <= 2000:
        C_f = 16/Re

    else:
        C_f = 0.0791/(Re**0.25)

    dh0 = math.pi*rho*C_f*(r**4)*(omega**3)*Delta_m/(4*m_dot)/1000

    return dh0
```