project3

December 28, 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
from ipynb.fs.full.project3_functions import i_sr, di_s, K_inc, K_M, K_p, K_Re,␣
 ↪Y_p1, Y_p2
from ipynb.fs.full.project3_functions import Y_p, Y_s, Y_TE, Y_sh, Y_EX, Y_CL,␣
 ↪Y_LW
from ipynb.fs.full.project3_functions import dh0_leak, dh0_DF, dh0_adm, dh0_gap
```

1 DATA

1.1 Pressure & Temperature

[2]:
```
T01     = 955                 #k
T03     = 810                 #k
P01     = 2.12                #bar
P03     = 1.01                #bar
P01_P03 = P01/P03
```

1.2 Reaction

[3]:
```
R_h = 0.06
R_m = 0.5
R_t = 0.71
```

1.3 Radius

[4]:
```
r_m   = 0.216                 #meter
r_h_i = 0.177                 #meter
r_h_m = 0.160                 #meter
r_h_e = 0.144                 #meter
r_t_i = 0.256                 #meter
r_t_m = 0.272                 #meter
```

```
r_t_e = 0.289                  #meter
```

1.4 Height

[5]:
```
h_b1 = 0.079                 #meter
h_b2 = 0.112                 #meter
h_b3 = 0.145                 #meter
```

1.5 Omega

[6]:
```
omega = 1570                 #rad/sec
```

1.6 Flow Angles

Flow angles are provided relative to turbine axis so we're going to have to write them relative to tangential axis:

1.6.1 Absolute Angles

[7]:
```
alpha1_h = 90-18.56          #degree
alpha1_m = 90-15.38          #degree
alpha1_t = 90-13.07          #degree
alpha2_h = 90-64.09          #degree
alpha2_m = 90-56.74          #degree
alpha2_t = 90-50.44          #degree
alpha3_h = 90-22.44          #degree
alpha3_m = 90-15.38          #degree
alpha3_t = 90-11.62          #degree
```

1.6.2 Relative Angles

[8]:
```
Beta2_h = 90-48.63           #degree
Beta2_m = 90-15.38           #degree
Beta2_t = 19.77-90           #degree
Beta3_h = 90-51.2            #degree
Beta3_m = 90-56.74           #degree
Beta3_t = 90-61.91           #degree
```

1.7 Flow Tangential Velocities

1.7.1 Absolute Velocities

[9]:
```
C_theta1_h = 91.3            #meter/sec
C_theta1_m = 74.82           #meter/sec
C_theta1_t = 63.14           #meter/sec
C_theta2_h = 559.9           #meter/sec
```

```
C_theta2_m = 414.7          #meter/sec
C_theta2_t = 329.3          #meter/sec
C_theta3_h = 112.3          #meter/sec
C_theta3_m = 74.8           #meter/sec
C_theta3_t = 55.9           #meter/sec
```

1.7.2 Relative Velocities

[10]:
```
W_theta2_h = 308.8          #meter/sec
W_theta2_m = 74.82          #meter/sec
W_theta2_t = -97.8          #meter/sec
W_theta3_h = 338.3          #meter/sec
W_theta3_m = 414.7          #meter/sec
W_theta3_t = 509.6          #meter/sec
```

1.8 Stator

[11]:
```
s_stator = 0.048            #meter
c_stator = 0.064            #meter
s_c_stator = 0.74
NOB_stator = 28
alpha1_prime = 90-15.38     #degree
alpha2_prime = 90-56.74     #degree
stagger_stator = 13.0       #degree
t_max_over_c_stator = 0.15
s_Rc_stator = 0.2
```

1.9 Rotor

[12]:
```
s_rotor = 0.064             #meter
c_rotor = 0.086             #meter
s_c_rotor = 0.74
NOB_rotor = 21
Beta1_prime  = 90-15.38     #degree
Beta2_prime  = 90-56.74     #degree
stagger_rotor  = 13.0       #degree
t_max_over_c_rotor = 0.15
s_Rc_rotor = 0.2
```

1.10 Mass Flow & C_m

[13]:
```
m_dot = 20.36               #kg/sec
C_m = 272                   #meter/sec
```

2 Thermodynamic Specifications

2.1 Air Specifications

[14]:
```
R_u = 8.314           #kj/kmol-k
R = R_u/28.9647       #kj/kg-k
P_ref = 1             #bar
T_ref = 298.15        #k
a = 3.653
b = -1.334*10**(-3)
c = 3.291*10**(-6)
d = -1.91*10**(-9)
e = 0.275*10**(-12)
```

2.2 Point 01

[15]:
```
point01 = {'P' : P01 , 'T' : T01}         #P in bar and T in k
```

[16]:
```
Properties01 = properties(point01)
```

[17]:
```
Properties01
```

[17]:
```
{'P[bar]': 2.12,
 'T[k]': 955,
 'C_p[kj/kg-k]': 1.1325605725778782,
 'h[kj/kg]': 999.5485812663288,
 's[kj/kg-k]': 6.9015215064805275}
```

2.3 Point 1

[18]:
```
C1 = C_theta1_m/math.cos(math.radians(alpha1_m))
```

[19]:
```
s1 = Properties01['s[kj/kg-k]']
```

[20]:
```
h01 = Properties01['h[kj/kg]']
h1  = h01-(C1**2)/2/1000
```

[21]:
```
point1 = {'h' : h1 , 's' : s1}
```

[22]:
```
point1
```

[22]:
```
{'h': 959.7567172608573, 's': 6.9015215064805275}
```

[23]:
```
Properties1 = properties(point1)
```

[24]:
```
Properties1
```

[24]:
```
{'P[bar]': 1.8285216664703146,
 'T[k]': 919.7545909849749,
 'C_p[kj/kg-k]': 1.12541055798866,
 'h[kj/kg]': 959.7567172608573,
 's[kj/kg-k]': 6.9015215064805275}
```

2.4 Point 03

[25]:
```
point03 = {'P' : P03 , 'T' : T03}          #P in bar and T in k
```

[26]:
```
Properties03 = properties(point03)
```

[27]:
```
Properties03
```

[27]:
```
{'P[bar]': 1.01,
 'T[k]': 810,
 'C_p[kj/kg-k]': 1.1007979962383003,
 'h[kj/kg]': 837.5602307343482,
 's[kj/kg-k]': 6.930452306287302}
```

But also We can use the relationship between Reactin Factor and delta_h0, so We can act like this:

[28]:
```
C2 = C_theta2_m/math.cos(math.radians(alpha2_m))
```

For Reaction Factor = 0.5, We have:

[29]:
```
dh0 = (C2**2-C1**2)/1000
```

[30]:
```
h03 = h01-dh0
```

[31]:
```
h03
```

[31]:
```
833.1761705336395
```

You can obviously see that both values computed for h03 are really close to each other and this is showing that the model created for real gas has an acceptable performance, but for this point it is really better to use the value resulted from Reaction Factor, so We have:

[32]:
```
Properties03['h[kj/kg]'] = h03
```

[33]:
```
Properties03
```

[33]:
```
{'P[bar]': 1.01,
 'T[k]': 810,
 'C_p[kj/kg-k]': 1.1007979962383003,
 'h[kj/kg]': 833.1761705336395,
 's[kj/kg-k]': 6.930452306287302}
```

2.5 Point 03_rel

[34]:
```
W3 = W_theta3_m/math.cos(math.radians(Beta3_m))
```

[35]:
```
C3 = C_theta3_m/math.cos(math.radians(alpha3_m))
```

[36]:
```
h03_rel = h03+0.5*(-C3**2+W3**2)/1000
```

[37]:
```
h03_rel
```

[37]:
```
916.3836464402993
```

[38]:
```
s03_rel = Properties03['s[kj/kg-k]']
```

[39]:
```
point03_rel = {'h' : h03_rel, 's' : s03_rel}
```

[40]:
```
point03_rel
```

[40]:
```
{'h': 916.3836464402993, 's': 6.930452306287302}
```

[41]:
```
Properties03_rel = properties(point03_rel)
```

[42]:
```
Properties03_rel
```

[42]:
```
{'P[bar]': 1.397783250463403,
 'T[k]': 881.0751252086811,
 'C_p[kj/kg-k]': 1.1170879306315786,
 'h[kj/kg]': 916.3836464402993,
 's[kj/kg-k]': 6.930452306287302}
```

2.6 Point 3

[43]:
```
s3 = Properties03['s[kj/kg-k]']
```

[44]:
```
h03 = Properties03['h[kj/kg]']
h3  = h03-(C3**2)/2/1000
```

[45]:
```
point3 = {'h' : h3, 's' : s3}
```

[46]:
```
point3
```

[46]:
```
{'h': 793.4055770684831, 's': 6.930452306287302}
```

[47]:
```
Properties3 = properties(point3)
```

[48]:
```
Properties3
```

[48]: {'P[bar]': 0.8312439131421313,
 'T[k]': 769.7145242070117,
 'C_p[kj/kg-k]': 1.091181020223647,
 'h[kj/kg]': 793.4055770684831,
 's[kj/kg-k]': 6.930452306287302}

2.7 rho2

[49]:
```
W2 = W_theta2_m/math.cos(math.radians(Beta2_m))
Cm2 = W2*math.sin(math.radians(Beta2_m))
A2 = math.pi*(r_t_m**2-r_h_m**2)
```

[50]:
```
rho2 = m_dot/(A2*Cm2)
```

[51]:
```
rho2
```

[51]: 0.4924387846534955

2.8 Point 2_approximate

From Reaction Factor We can approximately say:

[52]:
```
P1 = Properties1['P[bar]']
```

[53]:
```
P3 = Properties3['P[bar]']
```

[54]:
```
P2_app = P3+0.5*(P1-P3)
```

[55]:
```
P2_app
```

[55]: 1.3298827898062229

But this pressure is not true and for this pressure we will have s2 < s1 and that is obviously incorrect, so we need to use a correction factor and 0.91 is really acceptable:

[56]:
```
P2_app = ((0.91))*P2_app
```

[57]:
```
P2_app
```

[57]: 1.2101933387236627

[58]:
```
h02 = h01
```

[59]:
```
h2_app = h02-(0.5*C2**2)/1000
```

[60]:
```
point2_app = {'h' : h2_app, 'P' : P2_app}
```

[61]:
```
point2_app
```

[61]: {'h': 876.5705118945125, 'P': 1.2101933387236627}

[62]:
```
Properties2_app = properties(point2_app)
```

[63]:
```
Properties2_app
```

[63]:
```
{'P[bar]': 1.2101933387236627,
 'T[k]': 845.3055091819699,
 'C_p[kj/kg-k]': 1.1090252467857047,
 'h[kj/kg]': 876.5705118945125,
 's[kj/kg-k]': 6.925686459605529}
```

2.9 Point 02 & Point 02_rel

[64]:
```
h02_rel = h03_rel
```

[65]:
```
h02, h02_rel
```

[65]: (999.5485812663288, 916.3836464402993)

From the relationship between static enthalpy and total enthalpies in the above We can check the validation of computations:

[66]:
```
W2 = W_theta2_m/math.cos(math.radians(Beta2_m))
```

[67]:
```
abs((h02_rel-(0.5*W2**2)/1000)-(h02-(0.5*C2**2)/1000))
```

[67]: 0.021270540315299513

[68]:
```
s_app = Properties2_app['s[kj/kg-k]']
```

[69]:
```
dic = properties({'h' : h02_rel, 's' : s_app})
```

[70]:
```
P02_rel_app, T02_rel_app = dic['P[bar]'], dic['T[k]']
```

[71]:
```
P02_rel_app, T02_rel_app
```

[71]: (1.4211850525158858, 881.0751252086811)

[72]:
```
dic = properties({'h' : h02, 's' : s_app})
```

[73]:
```
P02_app, T02_app = dic['P[bar]'], dic['T[k]']
```

[74]:
```
P02_app, T02_app
```

[74]: (1.9487894954675762, 954.9949916527546)

[75]:
```
properties({'h' : h02_rel, 'P' : P02_rel_app})
```

[75]:
```
{'P[bar]': 1.4211850525158858,
 'T[k]': 881.0751252086811,
 'C_p[kj/kg-k]': 1.1170879306315786,
 'h[kj/kg]': 916.3836464402993,
 's[kj/kg-k]': 6.925686459605529}
```

[76]:
```
properties({'h' : h02, 'P' : P02_app})
```

[76]:
```
{'P[bar]': 1.9487894954675762,
 'T[k]': 954.9949916527546,
 'C_p[kj/kg-k]': 1.132559589276396,
 'h[kj/kg]': 999.5485812663288,
 's[kj/kg-k]': 6.925686459605529}
```

So You are able to see that by the approximation used in the previous section We managed to extraxt thermodynamic specifications at point 02 and point 02_rel, so We can say:

[77]:
```
Properties02 = properties({'h' : h02, 'P' : P02_app})
```

[78]:
```
Properties02
```

[78]:
```
{'P[bar]': 1.9487894954675762,
 'T[k]': 954.9949916527546,
 'C_p[kj/kg-k]': 1.132559589276396,
 'h[kj/kg]': 999.5485812663288,
 's[kj/kg-k]': 6.925686459605529}
```

[79]:
```
Properties02_rel = properties({'h' : h02_rel, 'P' : P02_rel_app})
```

[80]:
```
Properties02_rel
```

[80]:
```
{'P[bar]': 1.4211850525158858,
 'T[k]': 881.0751252086811,
 'C_p[kj/kg-k]': 1.1170879306315786,
 'h[kj/kg]': 916.3836464402993,
 's[kj/kg-k]': 6.925686459605529}
```

2.10 Point 2

[81]:
```
s2 = Properties02['s[kj/kg-k]']
```

[82]:
```
h2 = h02-(C2**2)/2/1000
```

[83]:
```
point2 = {'h' : h2, 's' : s2}
```

From Reaction Factor We have:

[84]:
```
h1 = Properties1['h[kj/kg]']
h3 = Properties3['h[kj/kg]']
```

[85]:
```
abs(0.5*(h1-h3)+h3-h2)
```

[85]:
```
0.010635270157763443
```

That shows Computations are valid.

[86]:
```
point2
```

[86]:
```
{'h': 876.5705118945125, 's': 6.925686459605529}
```

[87]:
```
Properties2 = properties(point2)
```

[88]:
```
Properties2
```

[88]:
```
{'P[bar]': 1.2101933387236636,
 'T[k]': 845.3055091819699,
 'C_p[kj/kg-k]': 1.1090252467857047,
 'h[kj/kg]': 876.5705118945125,
 's[kj/kg-k]': 6.925686459605529}
```

[89]:
```
P2 = Properties2['P[bar]']
T2 = Properties2['T[k]']
```

[90]:
```
P2*101.325/(R*T2)
```

[90]:
```
0.5053784110118777
```

[91]:
```
abs(rho2-P2*101.325/(R*T2))
```

[91]:
```
0.01293962635838225
```

[92]:
```
abs(rho2-P2*101.325/(R*T2))/rho2     #relative error
```

[92]:
```
0.026276619067458745
```

This shows that Our computations are valid.

2.11 Point 2_is

[93]:
```
P2_is = Properties2['P[bar]']
```

[94]:
```
s2_is = Properties01['s[kj/kg-k]']
```

[95]:
```
point2_is = {'P' : P2_is, 's' : s2_is}
```

[96]:
```
point2_is
```

[96]:
```
{'P': 1.2101933387236636, 's': 6.9015215064805275}
```

[97]:
```
Properties2_is = properties(point2_is)
```

[98]:
```
Properties2_is
```

[98]:
```
{'P[bar]': 1.2101933387236636,
 'T[k]': 827.0550918196996,
 'C_p[kj/kg-k]': 1.1048009831261285,
 'h[kj/kg]': 856.3686469837434,
 's[kj/kg-k]': 6.9015215064805275}
```

[99]:
```
h01 = Properties01['h[kj/kg]']
h2_is = Properties2_is['h[kj/kg]']
```

[100]:
```
C2_is = (2*(h01-h2_is)*1000)**0.5
```

Spouting Velocity is equal to:

[101]:
```
C2_is
```

[101]:
```
535.1260305434328
```

2.12 Point 3_is

[102]:
```
P3_is = Properties3['P[bar]']
```

[103]:
```
s3_is = Properties02['s[kj/kg-k]']
```

[104]:
```
point3_is = {'P' : P3_is, 's' : s3_is}
```

[105]:
```
point3_is
```

[105]:
```
{'P': 0.8312439131421313, 's': 6.925686459605529}
```

[106]:
```
Properties3_is = properties(point3_is)
```

[107]:
```
Properties3_is
```

[107]:
```
{'P[bar]': 0.8312439131421313,
 'T[k]': 766.3555926544241,
```

```
 'C_p[kj/kg-k]': 1.0903715676539332,
 'h[kj/kg]': 789.7432864682881,
 's[kj/kg-k]': 6.925686459605529}
```

2.13 Table of Specifications

[108]:
```python
Properties = {
    'Point 1'             : Properties1,
    'Point 01'            : Properties01,
    'Point 2'             : Properties2,
    'Point 02'            : Properties02,
    'Point 02_rel'        : Properties02_rel,
    'Point 2_approximate' : Properties2_app,
    'Point 2_is'          : Properties2_is,
    'Point 3'             : Properties3,
    'Point 03'            : Properties03,
    'Point 03_rel'        : Properties03_rel,
    'Point 3_is'          : Properties3_is,
}
```

[109]:
```python
df = pd.DataFrame(Properties).T
```

[110]:
```python
df
```

[110]:
```
                       P[bar]        T[k]  C_p[kj/kg-k]    h[kj/kg]  \
Point 1              1.828522  919.754591      1.125411  959.756717
Point 01             2.120000  955.000000      1.132561  999.548581
Point 2              1.210193  845.305509      1.109025  876.570512
Point 02             1.948789  954.994992      1.132560  999.548581
Point 02_rel         1.421185  881.075125      1.117088  916.383646
Point 2_approximate  1.210193  845.305509      1.109025  876.570512
Point 2_is           1.210193  827.055092      1.104801  856.368647
Point 3              0.831244  769.714524      1.091181  793.405577
Point 03             1.010000  810.000000      1.100798  833.176171
Point 03_rel         1.397783  881.075125      1.117088  916.383646
Point 3_is           0.831244  766.355593      1.090372  789.743286

                     s[kj/kg-k]
Point 1                6.901522
Point 01               6.901522
Point 2                6.925686
Point 02               6.925686
Point 02_rel           6.925686
Point 2_approximate    6.925686
Point 2_is             6.901522
Point 3                6.930452
Point 03               6.930452
```

```
Point 03_rel            6.930452
Point 3_is              6.925686
```

3 Aerodynamic Losses

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

3.1 Profile Loss

3.1.1 Stator

[111]:
```
Beta_1_prime_s = alpha1_prime
Beta1_s = alpha1_m
Beta2_s = alpha2_m
C_p1 = Properties1['C_p[kj/kg-k]']
T1 = Properties1['T[k]']
gama1 = C_p1/(C_p1-R)
M_w1_s = C1/((gama1*R*1000*T1)**0.5)
C_p2 = Properties2['C_p[kj/kg-k]']
gama2 = C_p2/(C_p2-R)
P2 = Properties2['P[bar]']
T2 = Properties2['T[k]']
M_w2_s = C2/((gama2*R*1000*T2)**0.5)
rho2_stator = P2*101.325/(R*T2)                                                    ␣
 ↪         #kg/m^3
Mu2 = (571.85-500)/(600-500)*(3.846-3.563)*10**(-5)+3.563*10**(-5)                 ␣
 ↪ #kg/m-sec for air at 571.85 centigrade
correctinon_fuel = 1.25
Mu2 *= correctinon_fuel                                                            ␣
 ↪ #kg/m-sec
chord = c_stator
Re_c_s = (rho2_stator*C2*chord)/Mu2
e = 0
Re_e_s = (rho2_stator*C2*e)/Mu2
t_max_over_c_s = t_max_over_c_stator
O_s = 0.03
s_s = s_stator
s_Rc_s = s_Rc_stator
```

[112]:
```
Y_profile_stator = Y_p(Beta_1_prime_s, Beta1_s, Beta2_s, t_max_over_c_s, O_s,␣
 ↪s_s, chord, M_w1_s, M_w2_s, Re_c_s, Re_e_s, s_Rc_s)
```

[113]:
```
Y_profile_stator
```

[113]:
```
{'K_mod': 0.67,
 'K_inc': 1.0,
```

```
 'K_M': 1.5925009578940743,
 'K_p': 0.7508799055553661,
 'K_Re': 1,
 'Y_p1': 0.024545048091145567,
 'Y_p2': 0.09234845537154945,
 'dY_TE': 0.0010928215285841132,
 'Y_p': 0.021005905209344478}
```

[114]:
```
Y_p_stator = Y_profile_stator['Y_p']
```

[115]:
```
Y_p_stator
```

[115]:
```
0.021005905209344478
```

3.1.2 Rotor

[116]:
```
Beta_1_prime_r = Beta1_prime
Beta1_r = Beta2_m
Beta2_r = Beta3_m
C_p1 = Properties2['C_p[kj/kg-k]']
T1 = Properties2['T[k]']
gama1 = C_p1/(C_p1-R)
M_w1_r = W2/((gama1*R*1000*T1)**0.5)
C_p2 = Properties3['C_p[kj/kg-k]']
gama2 = C_p2/(C_p2-R)
P2 = Properties3['P[bar]']
T2 = Properties3['T[k]']
M_w2_r = W3/((gama2*R*1000*T2)**0.5)
rho2_rotor = P2*101.325/(R*T2)                                                   
 ↪         #kg/m^3
Mu2 = (571.85-500)/(600-500)*(3.846-3.563)*10**(-5)+3.563*10**(-5)              
 ↪ #kg/m-sec for air at 571.85 centigrade
correctinon_fuel = 1.25
Mu2 *= correctinon_fuel                                                          
 ↪ #kg/m-sec
chord = c_rotor
Re_c_r = (rho2_rotor*W3*chord)/Mu2
e = 0
Re_e_r = (rho2_rotor*W3*e)/Mu2
t_max_over_c_r = t_max_over_c_rotor
O_r = 0.03
s_r = s_rotor
s_Rc_r = s_Rc_rotor
```

[117]:
```
Y_profile_rotor = Y_p(Beta_1_prime_r, Beta1_r, Beta2_r, t_max_over_c_r, O_r,
 ↪s_r, chord, M_w1_r, M_w2_r, Re_c_r, Re_e_r, s_Rc_r)
```

[118]:
```
Y_profile_rotor
```

[118]:
```
{'K_mod': 0.67,
 'K_inc': 1.0,
 'K_M': 1.6279172827426682,
 'K_p': 0.7385824956231877,
 'K_Re': 1,
 'Y_p1': 0.024710421488420578,
 'Y_p2': 0.09205641931398229,
 'dY_TE': 0.001986328473553123,
 'Y_p': 0.020499790440675212}
```

[119]:
```
Y_p_rotor = Y_profile_rotor['Y_p']
```

[120]:
```
Y_p_rotor
```

[120]:
```
0.020499790440675212
```

3.2 Secondary Loss

3.2.1 Stator

[121]:
```
K_Re_stator, K_p_stator = Y_profile_stator['K_Re'], Y_profile_stator['K_p']
chord = c_stator
bx_stator = chord*math.cos(math.radians(stagger_stator))
h_stator = h_b1
bx_h_stator = bx_stator/h_stator
h_c_stator = h_stator/chord
```

[122]:
```
Y_s_stator = Y_s(K_Re_stator, K_p_stator, bx_h_stator, Beta_1_prime_s, Beta1_s,␣
 ↪Beta2_s, s_c_stator, h_c_stator)
```

[123]:
```
Y_s_stator
```

[123]:
```
0.05457892055902439
```

3.2.2 Rotor

[124]:
```
K_Re_rotor, K_p_rotor = Y_profile_rotor['K_Re'], Y_profile_rotor['K_p']
chord = c_rotor
bx_rotor = chord*math.cos(math.radians(stagger_rotor))
h_rotor = h_b2
bx_h_rotor = bx_rotor/h_rotor
h_c_rotor = h_rotor/chord
```

[125]:
```
Y_s_rotor = Y_s(K_Re_rotor, K_p_rotor, bx_h_rotor, Beta_1_prime_r, Beta1_r,␣
 ↪Beta2_r, s_c_rotor, h_c_rotor)
```

[126]:
```
Y_s_rotor
```

[126]: 0.052727303464646685

3.3 Trailing Edge Loss

3.3.1 Stator

[127]:
```
t2_stator = 0.006    #meter
```

[128]:
```
Y_TE_stator = Y_TE(O_s, s_s, t2_stator, rho2_stator, C2)
```

[129]:
```
Y_TE_stator
```

[129]: 0.0625

3.3.2 Rotor

[130]:
```
t2_rotor = 0.006    #meter
```

[131]:
```
Y_TE_rotor = Y_TE(O_r, s_r, t2_rotor, rho2_rotor, W3)
```

[132]:
```
Y_TE_rotor
```

[132]: 0.0625

3.4 Sock Loss

3.4.1 Stator

[133]:
```
Y_sh_stator = Y_sh(M_w1_s, M_w2_s)
```

[134]:
```
Y_sh_stator
```

[134]: 0.004366567999639051

3.4.2 Rotor

[135]:
```
Y_sh_rotor = Y_sh(M_w1_r, M_w2_r)
```

[136]:
```
Y_sh_rotor
```

[136]: 0.006927385098119088

3.5 Supersonic Expansion Loss

3.6 Stator

[137]:
```
Y_EX_stator = Y_EX(M_w2_s)
```

[138]:
```
Y_EX_stator
```

[138]:
```
0
```

3.6.1 Rotor

[139]:
```
Y_EX_rotor = Y_EX(M_w2_r)
```

[140]:
```
Y_EX_rotor
```

[140]:
```
0
```

3.7 Blade Clearance Loss

3.7.1 Stator

[141]:
```
delta_stator = 0.004
```

[142]:
```
Y_CL_stator = Y_CL(Beta1_s, Beta2_s, s_c_stator, h_c_stator, c_stator,␣
↪delta_stator)
```

[143]:
```
Y_CL_stator
```

[143]:
```
0.2012526171929543
```

3.7.2 Rotor

[144]:
```
delta_rotor = 0.004
```

[145]:
```
Y_CL_rotor = Y_CL(Beta1_r, Beta2_r, s_c_rotor, h_c_rotor, c_rotor, delta_rotor)
```

[146]:
```
Y_CL_rotor
```

[146]:
```
0.15148885061003062
```

3.8 Lashing Wire Loss

3.8.1 Stator

[147]:
```
N_LW_stator = 1
D_LW_stator = 0.01    #meter
W2_stator = C2
```

```
Cm2_stator = C2*math.sin(math.radians(Beta2_s))
```

We can compute Cm2 from C or W, the difference between these mothods show that the data and methods are valid:

[148]:
```
abs(W2*math.sin(math.radians(Beta2_m))-C2*math.sin(math.radians(Beta2_s)))
```

[148]: 0.010380349913532427

[149]:
```
Y_LW_stator = Y_LW(N_LW_stator, D_LW_stator, Cm2_stator, rho2_stator, h_stator,
    W2_stator, Mu2)
```

[150]:
```
Y_LW_stator
```

[150]: 0.03807411833646628

3.8.2 Rotor

[151]:
```
N_LW_rotor = 1
D_LW_rotor = 0.01    #meter
W2_rotor = W3
Cm2_rotor = W3*math.sin(math.radians(Beta2_r))
```

We can compute Cm2 from C or W, the difference between these mothods show that the data and methods are valid:

[152]:
```
abs(W3*math.sin(math.radians(Beta2_r))-C3*math.sin(math.radians(alpha3_m)))
```

[152]: 0.062328315393585854

[153]:
```
Y_LW_rotor = Y_LW(N_LW_rotor, D_LW_rotor, Cm2_rotor, rho2_rotor, h_rotor,
    W2_rotor, Mu2)
```

[154]:
```
Y_LW_rotor
```

[154]: 0.02685585132661461

3.9 Table of Aerodynamic Losses

[155]:
```
Aerodynamic_Losses_stator = {
    'Y_p'  : Y_p_stator,
    'Y_s'  : Y_s_stator,
    'Y_TE' : Y_TE_stator,
    'Y_sh' : Y_sh_stator,
    'Y_EX' : Y_EX_stator,
    'Y_CL' : Y_CL_stator,
    'Y_LW' : Y_LW_stator,
```

```
}
```

[156]:
```
Aerodynamic_Losses_rotor = {
    'Y_p'  : Y_p_rotor,
    'Y_s'  : Y_s_rotor,
    'Y_TE' : Y_TE_rotor,
    'Y_sh' : Y_sh_rotor,
    'Y_EX' : Y_EX_rotor,
    'Y_CL' : Y_CL_rotor,
    'Y_LW' : Y_LW_rotor,
}
```

[157]:
```
Aerodynamic_Losses = {
    'Stator' : Aerodynamic_Losses_stator,
    'Rotor'  : Aerodynamic_Losses_rotor,
}
```

[158]:
```
df1 = pd.DataFrame(Aerodynamic_Losses).T
```

[159]:
```
df1['Y_total'] =␣
 ↪df1['Y_p']+df1['Y_s']+df1['Y_TE']+df1['Y_sh']+df1['Y_EX']+df1['Y_CL']+df1['Y_LW']
```

[160]:
```
df1
```

[160]:
```
             Y_p       Y_s    Y_TE      Y_sh  Y_EX      Y_CL      Y_LW  \
Stator  0.021006  0.054579  0.0625  0.004367   0.0  0.201253  0.038074
Rotor   0.020500  0.052727  0.0625  0.006927   0.0  0.151489  0.026856

         Y_total
Stator  0.381778
Rotor   0.320999
```

4 Parasitic Losses

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

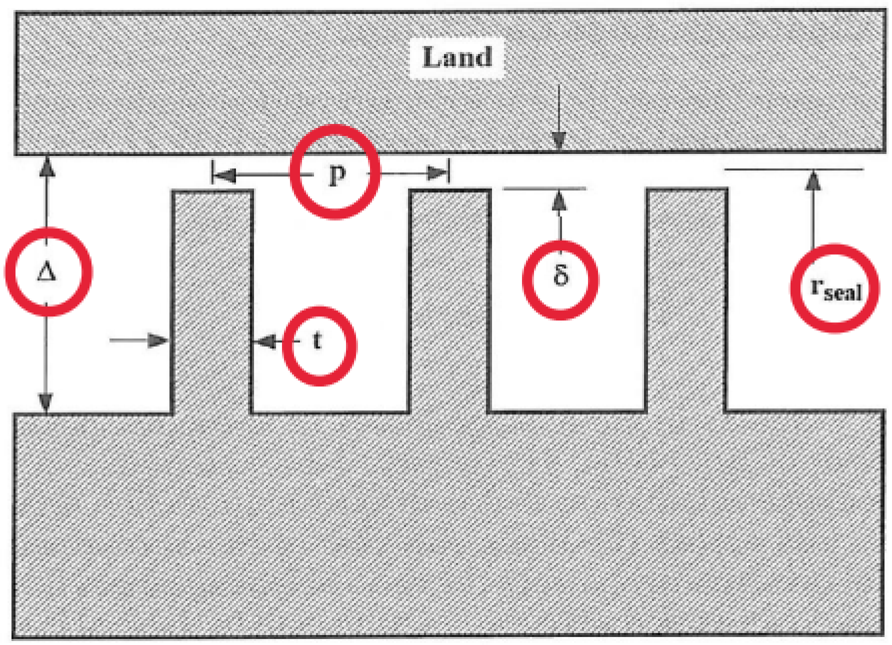
4.1 Leakage Bypass Loss

4.1.1 Stator

[161]:
```
dh0_leak_stator = 0
```

4.1.2 Rotor

in the below picture You can see the geometrical properties of the Labyrinth on the shroud:



But We need to consider this point that labyrinth in this case has been located on the Diaphragm-Disk and We don't have any shroud on the rotor blades. But We can say:

[162]:
```
r_seal_rotor = r_t_m+delta_rotor/2
```

But in this case We have:

[163]:
```
r_seal_is_constant = 'r_seal is not constant'
```

by paying attention to the previous picture, We can approximately say that:

[164]:
```
Delta_rotor = 6.5*delta_rotor
N_seal = 2
t_seal = 2*delta_rotor
p_seal = 0.04               #meter
```

As for Balance Hole We have:

[165]:
```
N_BH =2
D_BH = 0.005                #meter
```

[166]:
```
rho = rho2_stator
T = Properties2['T[k]']
P1 = Properties2['P[bar]']
P2 = Properties3['P[bar]']
PR = Properties3['P[bar]']/Properties2['P[bar]']
h01 = Properties02['h[kj/kg]']
h02 = Properties03['h[kj/kg]']
```

[167]:
```
dic = dh0_leak(N_seal,t_seal, p_seal, r_seal_rotor, PR, rho, T, P1, P2, h01,␣
↪h02, delta_rotor, R, N_BH, D_BH, m_dot, r_seal_is_constant)
```

[168]:
```
dic
```

[168]:
```
{'m_dot_seal[kg/sec]': 0.6911198970178687,
 'm_dot_BH[kg/sec]': 0.00765394747964283,
 'dh0_leak[kj/kg]': 5.710053490471531}
```

[169]:
```
dh0_leak_rotor = dic['dh0_leak[kj/kg]']
```

[170]:
```
dh0_leak_rotor
```

[170]:
```
5.710053490471531
```

4.1.3 Rotor Partial Admission Work

4.1.4 Stator

[171]:
```
dh0_adm_stator = 0
```

4.1.5 Rotor

[172]:
```
epsilon = 1
psi = 0
rho2, h = rho2_rotor, h_rotor
D_m = 2*r_m
U_m = r_m*omega
b_x = bx_rotor
N_active = 1
psi_noz = C2/C2_is
```

[173]:
```
dic = dh0_adm(epsilon, psi, rho2, h, D_m, U_m, b_x, N_active, psi_noz, C2,␣
↪m_dot)
```

[174]:
```
dic
```

[174]:
```
{'dh0_w[kj/kg]': 0.0,
 'dh0_sec[kj/kg]': 4.535079337775052,
 'dh0[kj/kg]': 4.535079337775052}
```

[175]:
```
dh0_adm_rotor = dic['dh0[kj/kg]']
```

[176]:
```
dh0_adm_rotor
```

[176]:
```
4.535079337775052
```

4.1.6 Rotor Diaphragm-Disk Friction Work

4.1.7 Stator

[177]:
```
dh0_DF_stator = 0
```

4.1.8 Rotor

[178]:
```
rho = rho2_rotor
r = r_h_m
Delta_r = Delta_rotor/r
```

[179]:
```
dic = dh0_DF(rho, r, Delta_r, e, Mu2, omega, m_dot)
```

[180]:
```
dic
```

[180]:
```
{'dh01[kj/kg]': 0.05103686902569655, 'dh02[kj/kg]': 0.0}
```

[181]:
```
dh0_DF_rotor = max(list(dic.values()))
```

[182]:
```
dh0_DF_rotor
```

[182]:
```
0.05103686902569655
```

4.1.9 Clearance Gap Windage Loss

4.1.10 Stator

[183]:
```
dh0_gap_stator = 0
```

4.1.11 Rotor

[184]:
```
r_seal_rotor = r_t_m+delta_rotor/2
Delta_m = ((r_t_e-r_t_m)**2+(b_x)**2)**0.5
r, rho, Delta, b_x = r_seal_rotor, rho2_stator, Delta_rotor, bx_rotor
```

[185]:
```
dh0_gap_rotor = dh0_gap(r, rho, Delta, b_x, Delta_m, Mu2, omega, m_dot)
```

[186]:
```
dh0_gap_rotor
```

[186]:
```
0.1837349844868593
```

4.1.12 Moisture or Wet Steam Work Loss

4.2 Stator

[187]:
```
dh0_ML_stator = 0
```

4.2.1 Rotor

[188]:
```python
dh0_ML_rotor = 0
```

4.3 Table of Parasitic Losses

[189]:
```python
Parasitic_Losses_stator = {
    'dh0_leak[kj/kg]' : dh0_leak_stator,
    'dh0_adm[kj/kg]'  : dh0_adm_stator,
    'dh0_DF[kj/kg]'   : dh0_DF_stator,
    'dh0_gap[kj/kg]'  : dh0_gap_stator ,
    'dh0_ML[kj/kg]'   : dh0_ML_stator,
}
```

[190]:
```python
Parasitic_Losses_rotor = {
    'dh0_leak[kj/kg]' : dh0_leak_rotor,
    'dh0_adm[kj/kg]'  : dh0_adm_rotor,
    'dh0_DF[kj/kg]'   : dh0_DF_rotor,
    'dh0_gap[kj/kg]'  : dh0_gap_rotor,
    'dh0_ML[kj/kg]'   : dh0_ML_rotor,
}
```

[191]:
```python
Parasitic_Losses = {
    'Stator' : Parasitic_Losses_stator,
    'Rotor'  : Parasitic_Losses_rotor,
}
```

[192]:
```python
df2 = pd.DataFrame(Parasitic_Losses).T
```

[193]:
```python
df2['dh0_total[kj/kg]'] = df2['dh0_leak[kj/kg]']+df2['dh0_adm[kj/
 ↪kg]']+df2['dh0_DF[kj/kg]']+df2['dh0_gap[kj/kg]']+df2['dh0_ML[kj/kg]']
```

[194]:
```python
df2
```

[194]:
```
        dh0_leak[kj/kg]  dh0_adm[kj/kg]  dh0_DF[kj/kg]  dh0_gap[kj/kg]  \
Stator         0.000000        0.000000       0.000000        0.000000
Rotor          5.710053        4.535079       0.051037        0.183735

        dh0_ML[kj/kg]  dh0_total[kj/kg]
Stator            0.0          0.000000
Rotor             0.0         10.479905
```

5 Turbine Efficiency

5.0.1 Stage

[206]:
```
h1 = Properties1['h[kj/kg]']
h2 = Properties2['h[kj/kg]']
h2_is = Properties2_is['h[kj/kg]']
```

[207]:
```
psi_nozzle = (h2-h2_is)*1000/(0.5*C2**2)
```

[208]:
```
h3 = Properties3['h[kj/kg]']
h3_is = Properties3_is['h[kj/kg]']
```

[209]:
```
psi_rotor = (h3-h3_is)*1000/(0.5*W3**2)
```

[210]:
```
T03 = Properties03['T[k]']
T3 = Properties3['T[k]']
T2 = Properties2['T[k]']
h01 = Properties01['h[kj/kg]']
h03 = Properties03['h[kj/kg]']
```

[211]:
```
etta_tt = (1+(T03/T3)*(psi_nozzle*(C2**2)*(T3/T2)+(W3**2)*psi_rotor)/
↪(2000*(h01-h03)))**(-1)
```

[212]:
```
etta_tt
```

[212]: 0.8775634455605741

5.1 Stator

[213]:
```
etta_stator = (h1-h2)/(h1-h2_is)
```

[214]:
```
etta_stator
```

[214]: 0.8046015864632978

5.2 Rotor

[215]:
```
etta_rotor = (h2-h3)/(h2-h3_is)
```

[218]:
```
etta_rotor
```

[218]: 0.9578209417355301

From Reaction Factor We have:

[219]:
```
0.5*(etta_stator+etta_rotor)
```

[219]: 0.881211264099414

this is showing that Our Computations are valid