

Project_2\Project2BackendCalcs.py

```
1 import numpy as np
2 import pandas as pd
3 ...
4 AEEM4063 Project 2 Backend Calculations
5
6 Computes compressor and turbine states, angles, and velocities for the multi-stage compressor and turbine design
7
8 Authors: Matt Boller, Pierce Elliott
9
10 University of Cincinnati FS23
11 ...
12 ...
13
14 # Begin the code
15 class TurboMachineryComputation:
16
17     def __init__(self):
18         self.BPR = 6.0 # Bypass ratio
19         self.mdot = 280.0 # Total mass flow, kg/s
20
21
22         # Sea level static conditions
23         self.pa = 1.01325 # Ambient pressure, bar
24         self.Ta = 288.15 # Ambient temperature, K
25         self.a = 340.3 # Speed of sound, m/s
26         self.rho_a = 1.2250 # Ambient density, kg/m^3
27         self.R = 287 # Gas constant, J/(kg*K)
28
29         # Compressor parameters
30         self.cpr = 15 # Compressor pressure ratio
31         self.n_inf_c = 0.9 # Lower bound of polytropic efficiency
32         self.pi_f = 1.4 # Fan pressure ratio
33         self.n_inf_f = 0.92 # Fan polytropic efficiency
34         self.haller = 0.65 # Lower bound on de Haller criteria
35         self.TMach_r = 1.2 # Upper bound on tip Mach number
36         self.y_c = 1.4 # Gamma in the cold section
37         self.cp_c = 1.005 # Specific heat cold section, kJ/(kg*K)
38         self.rr_rt_o = 0.5 # Root to tip ratio at entry to the compressor
39         self.lam = 0.98 # Loading coefficient for the first stage
40
41         # Turbine parameters
42         self.T_04 = 2275 # Turbine inlet temperature
43         self.n_inf_t = 0.92 # Turbine polytropic efficiency
44         self.phi = 0.78 # Lower bound on flow coefficient
45         self.psi = 3.3 # Upper bound on loading coefficient
46         self.y_h = 1.333 # Gamma in the hot section
47         self.cp_h = 1.148 # Specific heat hot section, kJ/(kg*K)
48
49         # Basic design parameters - constant tip radius assumption
50         self.U_t1 = 380 # Inlet Tip speed, m/s
51
52         self.C_a = 175 # Inlet axial velocity, m/s
53
54         # Other parameters
55         self.n_m = 0.99 # assumed mechanical efficiency of 99%
56         self.delP_b = 0.00 # assumed burner pressure loss of 0%
57         self.n_b = 1.0 # assumed burner eff. of 100%
58
59         # Calcs for inlet conditions to the compressor
60         # First, calculate fan outlet (assume intake has no loss)
61         self.P_02 = self.pi_f*self.pa
62         self.T_02 = self.Ta + self.Ta*((self.pi_f)**((self.y_c-1)/(self.y_c*self.n_inf_f)) - 1)
63
64         # Static temperature, pressure, and density at the inlet
65         T2 = self.T_02 - self.C_a**2/(2*self.cp_c*1000)
66         self.p2 = self.P_02*(T2/self.T_02)**(self.y_c/(self.y_c-1))
67         self.rho2 = self.p2*1e5/(self.R*T2)
68
69         # Tip radius and rotation rate
70         self.rt = np.sqrt((self.mdot/(self.BPR+1))/(np.pi*self.rho2*self.C_a*(1-self.rr_rt_o**2)))
71         self.N = self.U_t1/(2*np.pi*self.rt)
72
73         # Root and mean radius at the inlet
74         self.rr = self.rt*self.rr_rt_o
75         self.rm = (self.rr+self.rt)/2
76
77         # Velocity and Mach number relative to the inlet
78         V1_t = np.sqrt(self.U_t1**2+self.C_a**2)
79         self.M1_t = V1_t/(np.sqrt(self.y_c*self.R*T2))
```

```

78 # Outlet calculations for the compressor
79 self.P_03 = self.P_02*self.cpr
80 self.T_03 = self.T_02*(self.cpr)**((self.y_c-1)/(self.y_c*self.n_inf_c))
81 # Assume the exit velocity is axial and the same as the initial inlet velocity
82 T3 = self.T_03 - self.C_a**2/(2*self.cp_c*1000)
83 self.p3 = self.P_03*(T3/self.T_03)**(self.y_c/(self.y_c-1))
84 self.rho3 = self.p3*1e5/(self.R*T3)
85
86 # Calculates the fuel flow
87 FAR = (self.cp_h*(self.T_04) - self.cp_c*(self.T_03))/(43100 - self.cp_h*self.T_03)
88 m_fuel = FAR*(self.mdot / (self.BPR + 1))
89 self.mdot = (self.mdot / (self.BPR + 1)) + m_fuel # Hot Section Flow, kg/s
90
91 # Calculates inlet pressure to the turbine using burner efficiency
92 self.P_04 = self.P_03*self.n_b
93
94 A3 = self.mdot/(self.BPR+1)/(self.rho3*self.C_a)
95 h3 = A3/(2*np.pi*self.rm)
96 self.rt_3 = self.rm + (h3/2)
97 self.rr_3 = self.rm - (h3/2)
98
99 # Calculates the inlet a
100
101 return
102
103 def update(self):
104     return self.N, self.rt, self.rm, self.rr, self.M1_t, self.rt_3, self.rr_3, self.P_02, self.T_02, self.P_03, self.T_03
105
106 def fullcompressor(self):
107     # Temperature rise through the compressor
108     self.delt_Ts = self.T_03-self.T_02
109     # Assuming axial velocity remains constant throughout each stage
110     # Calculating mean blade speed
111     self.U_m = 2*np.pi*self.rm*self.N
112     # Calculating the estimated temperature rise per stage
113     beta1 = np.arctan(self.U_m/self.C_a)
114     V1 = self.C_a/np.cos(beta1)
115     # Using de Haller criteria to find V2
116     V2 = V1*self.haller
117     beta2 = np.arccos(self.C_a/V2)
118     # Estimated delta T0 per stage
119     T0s_est = self.lam*self.U_m*self.C_a*(np.tan(beta1)-np.tan(beta2))/(self.cp_c*1e3)
120     # Estimated number of stages
121     stage_est = np.ceil(self.delt_Ts/T0s_est)
122
123     alpha1 = 0.0 # radians, inlet blade angle
124
125 ##### Stage 1 #####
126     # First stage calculation
127     delta_T0 = 30 ## Desired temperature rise per stage for the first stage
128     delta_C_w = self.cp_c*1e3*delta_T0/(self.lam*self.U_m)
129     self.lam = self.workdone(1) ## Update work done factor
130     # Whirl velocities
131     C_w1 = self.C_a*np.tan(alpha1)
132     C_w2 = delta_C_w + C_w1
133     # Relative angles
134     beta1 = np.arctan((self.U_m - C_w1)/self.C_a)
135     beta2 = np.arctan((self.U_m - C_w2)/self.C_a)
136     # Outlet angle of rotor
137     alpha2 = np.arctan(C_w2/self.C_a)
138     diffusion = np.cos(beta1)/np.cos(beta2)
139     # Organizing data for tables
140     data = np.concatenate((np.rad2deg(np.array([alpha1,alpha2,beta1,beta2])), np.array([0.0,diffusion,0.0])))
141     # Outlet stagnation pressure calculation of the first stage
142     p031 = self.P_02*(1 + (self.n_inf_c*delta_T0)/self.T_02)**(self.y_c/(self.y_c-1))
143     # Inlet tip Mach number
144     T2 = self.T_02 - self.C_a**2/(2*self.cp_c*1000)
145     # M1t = self.C_a/np.cos(beta1)/np.sqrt(self.y_c*self.R*T2)
146     # Outlet tip Mach number
147     C21 = self.C_a/np.cos(alpha2)
148     T021 = self.T_02 + delta_T0
149     T21 = T021 - C21**2/(2*self.cp_c*1e3)
150     # M21t = self.C_a/np.cos(beta2)/np.sqrt(self.y_c*self.R*T21)
151     # Approximate Degree of Reaction at the mean radius
152     React = 1 - (C_w1+C_w2)/(2*self.U_m)
153     # Radius at the inlet to the rotor
154     T11 = self.T_02 - self.C_a**2/(2*self.cp_c*1e3)
155     p11 = self.P_02*(T11/self.T_02)**(self.y_c/(self.y_c-1))
156     rho11 = p11*1e5/(T11*self.R)
157     A11 = self.mdot/(self.BPR+1)/(rho11*self.C_a)

```

```

158     h11 = A11/(2*np.pi*self.rm)
159     rt11 = self.rm + (h11/2)
160     rr11 = self.rm - (h11/2)
161     rr_rt1 = rr11/rt11
162     # Radius in between rotor and stator
163     p21 = p031*(T21/T021)**(self.y_c/(self.y_c-1))
164     rho21 = p21*1e5/(T21*self.R)
165     A21 = self.mdot/(self.BPR+1)/(rho21*self.C_a)
166     h21 = A21/(2*np.pi*self.rm)
167     rt21 = self.rm + (h21/2)
168     rr21 = self.rm - (h21/2)
169     rr_rt = rr21/rt21
170
171 sizingtable = pd.DataFrame(np.round([np.array([rr_rt1,rr_rt,0.0, h11,h21,0.0])],5),columns=['r_t 1','r_t 2','r_t 3','h1','h2','
172 h3'])
173
174 meantable = pd.DataFrame(np.round([np.concatenate((data,np.array([T021]),np.array([p031/self.P_02]),np.array([p031]),
175 np.array([0.0]),np.array([0.0,C_w1,C_w2]),np.array([React]),np.array([self.lam]))),3], columns=['alpha1','alpha2','beta1','beta2','
176 alpha3','V2/V1','C3/C2','T02','P03/P01','P03','M1t','M2t','Cw1','Cw2','Reaction','Work Done'])
177 # meantable['C3/C2'][0] = 2.0
178
179 ##### Stage 2 #####
180 delta_T0 = 29 ## Desired temperature rise for the second stage
181 React = 0.65 ## Degree of reaction for the second stage
182 self.lam = self.workdone(2) ## Update work done factor
183 # Calculate relative blade angles by solving system of eqs
184 B1 = delta_T0*self.cp_c*1e3/(self.lam*self.U_m*self.C_a)
185 B2 = React*2*self.U_m/self.C_a
186 beta2 = np.arctan((B2-B1)/2)
187 beta1 = np.arctan(B1+np.tan(beta2))
188 # Calculate absolute blade angles
189 alpha1 = np.arctan(self.U_m/self.C_a - np.tan(beta1))
190 alpha2 = np.arctan(self.U_m/self.C_a - np.tan(beta2))
191 # Calculate whirl velocities
192 C_w1 = self.C_a*np.tan(alpha1)
193 C_w2 = self.C_a*np.tan(alpha2)
194 # Calculation diffusion for de Haller criteria
195 diffusion = np.cos(beta1)/np.cos(beta2)
196 # Calculate outlet pressure
197 p032 = p031*(1 + (self.n_inf_c*delta_T0)/T021)**(self.y_c/(self.y_c-1))
198 # Calculate outlet temperature
199 T022 = T021 + delta_T0
200 # Calculate inlet Mach number
201 C21 = self.C_a/np.cos(alpha1)
202 T21 = T021 - C21**2/(2*self.cp_c*1e3)
203 # M12t = self.C_a/np.cos(beta1)/np.sqrt(self.y_c*self.R*T21)
204 # Calculate outlet Mach number
205 C22 = self.C_a/np.cos(alpha2)
206 T22 = T022 - C22**2/(2*self.cp_c*1e3)
207 # M22t = self.C_a/np.cos(beta2)/np.sqrt(self.y_c*self.R*T22)
208 # Data organization
209 data = np.round(np.concatenate((np.rad2deg(np.array([alpha1,alpha2,beta1,beta2])), np.array([0.0,diffusion, 0.0,T022, p032/p031,
210 p032, 0.0, 0.0,C_w1,C_w2, React, self.lam])),3)
211 # Radius at the inlet to the rotor
212 T12 = T021 - C21**2/(2*self.cp_c*1e3)
213 p12 = p031*(T12/T021)**(self.y_c/(self.y_c-1))
214 rho12 = p12*1e5/(T12*self.R)
215 A12 = self.mdot/(self.BPR+1)/(rho12*self.C_a)
216 h12 = A12/(2*np.pi*self.rm)
217 rt12 = self.rm + (h12/2)
218 rr12 = self.rm - (h12/2)
219 rr_rt1 = rr12/rt12
220 # Radius in between rotor and stator
221 p22 = p032*(T22/T022)**(self.y_c/(self.y_c-1))
222 rho22 = p22*1e5/(T22*self.R)
223 A22 = self.mdot/(self.BPR+1)/(rho22*self.C_a)
224 h22 = A22/(2*np.pi*self.rm)
225 rt22 = self.rm + (h22/2)
226 rr22 = self.rm - (h22/2)
227 rr_rt = rr22/rt22
228
229 s_data = np.round(np.array([rr_rt1,rr_rt,0.0, h12,h22,0.0]),5)
230
231 # Update table
232 meantable.loc[len(meantable)] = data
233 sizingtable.loc[len(sizingtable)] = s_data
234
235 ##### Stage 3 #####
236 delta_T0 = 29 ## Desired temperature rise for the second stage
237 React = 0.5 ## Degree of reaction for the second stage
238 self.lam = self.workdone(3) ## Update work done factor

```

```

235 # Calculate relative blade angles by solving system of eqs
236 B1 = delta_T0*self.cp_c*1e3/(self.lam*self.U_m*self.C_a)
237 B2 = React*2*self.U_m/self.C_a
238 beta2 = np.arctan((B2-B1)/2)
239 beta1 = np.arctan(B1+np.tan(beta2))
240 # Calculate absolute blade angles
241 alpha1 = np.arctan(self.U_m/self.C_a - np.tan(beta1))
242 alpha2 = np.arctan(self.U_m/self.C_a - np.tan(beta2))
243 # Calculate whirl velocities
244 C_w1 = self.C_a*np.tan(alpha1)
245 C_w2 = self.C_a*np.tan(alpha2)
246 # Calculation diffusion for de Haller criteria
247 diffusion = np.cos(beta1)/np.cos(beta2)
248 # Calculate outlet pressure
249 p033 = p032*(1 + (self.n_inf_c*delta_T0)/T022)**(self.y_c/(self.y_c-1))
250 # Calculate outlet temperature
251 T023 = T022 + delta_T0
252 # Calculate inlet Mach number
253 C22 = self.C_a/np.cos(alpha1)
254 T22 = T022 - C22**2/(2*self.cp_c*1e3)
255 # M13t = self.C_a/np.cos(beta1)/np.sqrt(self.y_c*self.R*T22)
256 # Calculate outlet Mach number
257 C23 = self.C_a/np.cos(alpha2)
258 T23 = T023 - C23**2/(2*self.cp_c*1e3)
259 # M23t = self.C_a/np.cos(beta2)/np.sqrt(self.y_c*self.R*T23)
260 # Data organization
261 data = np.round(np.concatenate((np.rad2deg(np.array([alpha1,alpha2,beta1,beta2])), np.array([0.0,diffusion, 0.0,T023, p033/p032,
p033, 0.0, 0.0,C_w1,C_w2, React, self.lam])),3)
262 # Radius at the inlet to the rotor
263 p13 = p032*(T22/T022)**(self.y_c/(self.y_c-1))
264 rho13 = p13*1e5/(T22*self.R)
265 A13 = self.mdot/(self.BPR+1)/(rho13*self.C_a)
266 h13 = A13/(2*np.pi*self.rm)
267 rt13 = self.rm + (h13/2)
268 rr13 = self.rm - (h13/2)
269 rr_rt1 = rr13/rt13
270 # Radius in between rotor and stator
271 p23 = p033*(T23/T023)**(self.y_c/(self.y_c-1))
272 rho23 = p23*1e5/(T23*self.R)
273 A23 = self.mdot/(self.BPR+1)/(rho23*self.C_a)
274 h23 = A23/(2*np.pi*self.rm)
275 rt23 = self.rm + (h23/2)
276 rr23 = self.rm - (h23/2)
277 rr_rt = rr23/rt23
278
279 s_data = np.round(np.array([rr_rt1,rr_rt,0.0,h13,h23,0.0]),5)
280
281 # Update table
282 meantable.loc[len(meantable)] = data
283 sizingtable.loc[len(sizingtable)] = s_data
284
285 test_p03 = p033
286 test_T02 = T023
287
288 test3_p03 = test_p03
289
290 final_P03 = self.cpr*self.P_02
291 ns = 4
292
293 ##### Intermediate Stages #####
294 while test3_p03 < final_P03:
295     self.lam = self.workdone(ns) ## Update work done factor
296     ns += 1
297
298     delta_T0 = 100.0
299     React = 0.5
300     data, s_data, test2_p03, test2_T02, diffusion = self.compressorstage(delta_T0, React, test_p03, test_T02)
301     while diffusion < self.haller+0.07:
302         delta_T0 -= 0.01
303         data, s_data, test2_p03, test2_T02, diffusion = self.compressorstage(delta_T0, React, test_p03, test_T02)
304
305     test_p03 = test2_p03
306     test_T02 = test2_T02
307     # Update table
308     meantable.loc[len(meantable)] = data
309     sizingtable.loc[len(sizingtable)] = s_data
310
311     # Calculate next estimated max
312     delta_T0 = 100.0
313     data, s_data, test2_p03, test2_T02, diffusion = self.compressorstage(delta_T0, React, test_p03, test_T02)
314     while diffusion < self.haller+0.01:

```

```

315         delta_T0 -= 0.01
316         data, s_data, test2_p03, test2_T02, diffusion = self.compressorstage(delta_T0, React, test_p03, test_T02)
317
318     test3_p03 = test2_p03
319
320     p01 = test_p03
321     T01 = test_T02
322
323 ##### Last Stage #####
324     delta_T0 = ((15.0*self.P_02/p01)**((self.y_c-1)/self.y_c-1))*T01/self.n_inf_c # Desired temperature rise for the second stage
325     React = 0.5 ## Degree of reaction for the second stage
326     self.lam = self.workdone(15) ## Update work done factor
327     # Calculate relative blade angles by solving system of eqs
328     B1 = delta_T0*self.cp_c*1e3/(self.lam*self.U_m*self.C_a)
329     B2 = React*2*self.U_m/self.C_a
330     beta2 = np.arctan((B2-B1)/2)
331     beta1 = np.arctan(B1+np.tan(beta2))
332     # Calculate absolute blade angles
333     alpha1 = np.arctan(self.U_m/self.C_a - np.tan(beta1))
334     alpha2 = np.arctan(self.U_m/self.C_a - np.tan(beta2))
335     # Calculate whirl velocities
336     C_w1 = self.C_a*np.tan(alpha1)
337     C_w2 = self.C_a*np.tan(alpha2)
338     # Calculation diffusion for de Haller criteria
339     diffusion = np.cos(beta1)/np.cos(beta2)
340     # Calculate outlet pressure
341     p03 = p01*(1 + (self.n_inf_c*delta_T0)/T01)**(self.y_c/(self.y_c-1))
342     # Calculate outlet temperature
343     T02 = T01 + delta_T0
344     # Calculate inlet Mach number
345     C2 = self.C_a*np.cos(alpha1)
346     T1 = T01 - C2**2/(2*self.cp_c*1e3)
347     # M1t = self.C_a*np.cos(beta1)/np.sqrt(self.y_c*self.R*T1)
348     # Calculate outlet Mach number
349     C23 = self.C_a*np.cos(alpha2)
350     T2 = T02 - C23**2/(2*self.cp_c*1e3)
351     # M2t = self.C_a*np.cos(beta2)/np.sqrt(self.y_c*self.R*T2)
352     # Data organization
353     data = np.round(np.concatenate((np.rad2deg(np.array([alpha1,alpha2,beta1,beta2])), np.array([0.0,diffusion, 0.0,T02, p03/p01,
p03, 0.0, 0.0,C_w1,C_w2, React, self.lam])),3)
354     # Radius at the inlet to the rotor
355     C1 = self.C_a*np.cos(alpha1)
356     T1 = T01 - C1**2/(2*self.cp_c*1e3)
357     p1 = p01*(T1/T01)**(self.y_c/(self.y_c-1))
358     rho1 = p1*1e5/(T1*self.R)
359     A1 = self.mdot/(self.BPR+1)/(rho1*self.C_a)
360     h1 = A1/(2*np.pi*self.rm)
361     rt1 = self.rm + (h1/2)
362     rr1 = self.rm - (h1/2)
363     rr_rt1 = rr1/rt1
364     # Radius in between rotor and stator
365     p2 = p03*(T2/T02)**(self.y_c/(self.y_c-1))
366     rho2 = p2*1e5/(T2*self.R)
367     A2 = self.mdot/(self.BPR+1)/(rho2*self.C_a)
368     h2 = A2/(2*np.pi*self.rm)
369     rt2 = self.rm + (h2/2)
370     rr2 = self.rm - (h2/2)
371     rr_rt2 = rr2/rt2
372     # Radius outlet of the stator
373     T3 = T02 - self.C_a**2/(2*self.cp_c*1e3)
374     p3 = p03*(T3/T02)**(self.y_c/(self.y_c-1))
375     rho3 = p3*1e5/(T3*self.R)
376     A3 = self.mdot/(self.BPR+1)/(rho3*self.C_a)
377     h3 = A3/(2*np.pi*self.rm)
378     rt3 = self.rm + (h3/2)
379     rr3 = self.rm - (h3/2)
380     rr_rt3 = rr3/rt3
381
382     s_data = np.round(np.array([rr_rt1,rr_rt,rr_rt3,h1,h2,h3]),5)
383
384     # Update table
385     meantable.loc[len(meantable)] = data
386     sizingtable.loc[len(sizingtable)] = s_data
387
388     # Iterate through table to organize stator outlet angles
389     for i in range(0,len(meantable)-1):
390         meantable['alpha3'][i] = meantable['alpha1'][i+1]
391         sizingtable['r_t 3'][i] = sizingtable['r_t 1'][i+1]
392         sizingtable['h3'][i] = sizingtable['h1'][i+1]
393
394     # Iterate through table to calculate de Haller values for the stators

```

```

395     for i in range(0,len(meantable)):
396         alpha2 = np.deg2rad(meantable['alpha2'][i])
397         alpha3 = np.deg2rad(meantable['alpha3'][i])
398         s_diffusion = np.cos(alpha2)/np.cos(alpha3)
399         meantable['C3/C2'][i] = np.round(s_diffusion,3)
400
401     tiproot_table, whirl_table, vel_table, meantable, dif_table = self.comp_root_tip(meantable=sizingtable,sizingtable=sizingtable,rm=
self.rm)
402
403     sizingtable.index = np.arange(1, len(sizingtable)+1)
404     meantable.index = np.arange(1, len(meantable)+1)
405     tiproot_table.index = np.arange(1, len(tiproot_table)+1)
406     whirl_table.index = np.arange(1, len(whirl_table)+1)
407     vel_table.index = np.arange(1, len(vel_table)+1)
408     dif_table.index = np.arange(1, len(dif_table)+1)
409     # meantable.index.name = 'Stage'
410     # meantable.reset_index().to_string(index=False)
411     return meantable, sizingtable, tiproot_table, whirl_table, vel_table, dif_table
412
413
414
415
416 def fullturbine(self):
417
418     ##### Values to Toggle #####
419     Um = 340 #assumed mean blade speed based on experience, m/s
420     psi_max = [3.1, 3.2, 3.3] #stage max temp drop coeff. values
421     phi_vals = [0.78, 0.78, 0.78] #stage flow coeff. values
422     lambda_vals = [0.6, 0.5, 0.5] #desired deg. of reaction values
423
424     ##### Preliminary Sizing #####
425     # Sets the rotational speed and mean blade speed
426     N = self.N
427     lamdaN = 0.05 # assumed standard value
428
429     # Calculates the total temperature drop based on a work balance from the compressor (using assumed mech. eff.)
430     dT0_turb = (self.cp_c*(self.T_03-self.T_02))/(self.cp_h*self.n_m)
431
432     ##### Stage 1 #####
433     # Uses stage estimation based on constant drop (initial guess) over the stages
434     T0s_est = 1000 # K
435
436     # Calculates the temp drop coeff. for the first stage
437     psi_turb1 = (2*self.cp_h*1e3*T0s_est)/(Um**2)
438     # iteration to find the stage drop below the loading coefficient
439     T0s_rev1 = T0s_est
440     if np.round(psi_turb1, 3) > psi_max[0]:
441         while np.round(psi_turb1, 3) > psi_max[0]:
442             T0s_rev1 -= 0.01
443             psi_turb1 = (2*self.cp_h*1e3*T0s_rev1)/(Um**2)
444
445     stage_est = (dT0_turb/T0s_rev1)
446
447
448     # Assumptions for first stage
449     alpha1 = 0.0
450     alpha3 = 0.0
451     P_011 = self.P_04
452     T_011 = self.T_04
453
454     gasParamsStg1, measurementsStg1, axialV1 = self.turbinestage(alpha1, alpha3, Um, T_011, P_011, T0s_rev1, psi_turb1, phi_vals[0],
lambda_vals[0])
455
456     rootVals1, tipVals1, rtMeasure1 = self.turb_root_tip(gasParamsStg1, measurementsStg1, Um, axialV1)
457
458     '''# Calculates the B3 and deg. of reaction
459     beta3 = np.arctan(np.tan(alpha3) + (1/self.phi))
460     Lambda = (2*self.phi*np.tan(beta3)- (psi_turb/2))/2
461     # iterates to find a suitable degree of reaction
462     if np.round(Lambda, 3) > 0.420:
463         while np.round(Lambda, 3) > 0.420:
464             alpha3 += np.deg2rad(5)
465             beta3 = np.arctan(np.tan(alpha3) + (1/self.phi))
466             Lambda = (2*self.phi*np.tan(beta3)- (psi_turb/2))/2
467
468     # Calculates B2 and a2
469     beta2 = np.arctan((1/(2*self.phi))*(psi_turb/2 - 2*Lambda))
470     alpha2 = np.arctan(np.tan(beta2) + (1/self.phi))
471
472     # Calculates Ca2 and C2
473     Ca2 = Um*self.phi

```

```

474 C2 = Ca2 / np.cos(alpha2)
475
476 # Calculates the isentropic T2', T2 and p2
477 T2 = self.T_04 - (C2**2/(2*self.cp_h*1e3))
478 T2prime = T2- lamdaN*(C2**2/(2*self.cp_h*1e3))
479 stagStaticRat = (self.T_04/T2prime)**(self.y_h/(self.y_h-1))
480 CritPrat = 1.853
481 if stagStaticRat > CritPrat:
482     print('ask the child if he or she is choking')
483
484 P2 = self.P_04/stagStaticRat
485
486 # Calculate the rho2 and A2
487 rho2 = (P2*100)/(0.287*T2)
488 A2 = self.mdot/(rho2*Ca2)
489
490 # Calculates the Ca1, using Ca2 = Ca3 and C1 = C3
491 Ca3 = Ca2
492 C3 = Ca3/np.cos(alpha3)
493 C1 = C3
494 Ca1 = C1 #since alpha1 = 0.0
495
496 # Calculates rho1 and A1
497 T1 = self.T_04 - (C1**2/(2*self.cp_h*1e3))
498 P1 = self.P_04*(T1/self.T_04)**(self.y_h/(self.y_h-1))
499 rho1 = (P1*100)/(0.287*T1)
500 A1 = self.mdot/(rho1*Ca1)
501
502 # Calculates the outlet (station 3) conditions
503 T_03 = self.T_04 - T0s_rev
504 T3 = T_03 - (C3**2)/(2*self.cp_h*1e3)
505
506 # Calculates the P03 from the isentropic eff. (using P04 = P01 and T04 = T01) and P3 from isentropic
507 P_03 = self.P_04*(1 - (T0s_rev/(self.n_inf_t)))**(self.y_h/(self.y_h-1))
508 P3 = P_03*(T3/T_03)**(self.y_h/(self.y_h-1))
509
510 # Calculates the rho3, A3
511 rho3 = (P3*100)/(0.287*T3)
512 A3 = self.mdot/(rho3*Ca3)
513
514 # Rework to put at beginning for sizing
515 # Size at inlet and outlet of turbine
516 # Calculate the mean radius, use for calcs to get Um
517 # Calculate rm
518 rm = Um/(2*np.pi*N)
519
520 # Calculates the h1-h3
521 h1 = (N/Um)*A1
522 h2 = (N/Um)*A2
523 h3 = (N/Um)*A3
524
525 # Calculates the rt/rr
526 rtRat1 = (rm + (h1/2))/(rm - (h1/2))
527 rtRat2 = (rm + (h2/2))/(rm - (h2/2))
528 rtRat3 = (rm + (h3/2))/(rm - (h3/2))
529
530 # # Calculates the Yn (T02 = T04)
531 # P_02 = P2 / ((T2/self.T_04)**(self.y_h/(self.y_h - 1)))
532 # print(self.P_04)
533 # print(P_02)
534 # print(P2)
535 # Yn = (self.P_04 - P_02)/(P_02 - P2) ''
536
537 ##### Stage 2 #####
538 # Uses stage estimation based on constant drop (initial guess) over the stages
539 T0_remaining = dT0_turb - T0s_rev1 # K
540 T0s_est = T0_remaining
541
542 # Calculates the temp drop coeff. for the first stage
543 psi_turb2 = (2*self.cp_h*1e3*T0s_est)/(Um**2)
544 # iteration to find the stage drop below the loading coefficient
545 T0s_rev2 = T0s_est
546 if np.round(psi_turb2, 3) > psi_max[1]:
547     while np.round(psi_turb2, 3) > psi_max[1]:
548         T0s_rev2 -= 0.01
549         psi_turb2 = (2*self.cp_h*1e3*T0s_rev2)/(Um**2)
550
551 # Assumptions for second stage
552 alpha1 = np.deg2rad(gasParamsStg1[2]) # new alpha1 is the previous stage alpha3
553 alpha3 = 0.0

```

```

554     T_012 = self.T_04 - T0s_rev1
555     P_012 = self.P_04*gasParamsStg1[6]
556
557     gasParamsStg2, measurementsStg2, axialV2 = self.turbinestage(alpha1, alpha3, Um, T_012, P_012, T0s_rev2, psi_turb2, phi_vals[1], lambda_vals[1])
558
559     rootVals2, tipVals2, rtMeasure2 = self.turb_root_tip(gasParamsStg2, measurementsStg2, Um, axialV2)
560
561     ##### Stage 3 #####
562     # Uses stage estimation based on constant drop (initial guess) over the stages
563     T0_remaining -= T0s_rev2 # K
564     T0s_est = T0_remaining
565
566     # Calculates the temp drop coeff. for the first stage
567     psi_turb3 = (2*self.cp_h*1e3*T0s_est)/(Um**2)
568     # iteration to find the stage drop below the loading coefficient
569     T0s_rev3 = T0s_est
570     if np.round(psi_turb3, 3) > psi_max[2]:
571         while np.round(psi_turb3, 3) > psi_max[2]:
572             T0s_rev3 -= 0.01
573             psi_turb3 = (2*self.cp_h*1e3*T0s_rev3)/(Um**2)
574
575     # Assumptions for the third stage
576     alpha1 = np.deg2rad(gasParamsStg2[2])
577     alpha3 = 0.0
578     T_013 = self.T_04 - T0s_rev1 - T0s_rev2
579     P_013 = self.P_04*gasParamsStg1[6]*gasParamsStg2[6]
580
581
582     gasParamsStg3, measurementsStg3, axialV3 = self.turbinestage(alpha1, alpha3, Um, T_013, P_013, T0s_rev3, psi_turb3, phi_vals[2], lambda_vals[2])
583
584     rootVals3, tipVals3, rtMeasure3 = self.turb_root_tip(gasParamsStg3, measurementsStg3, Um, axialV3)
585
586     # Adds the data to Pandas DFs
587     gasParamData = np.round(np.array([gasParamsStg1,gasParamsStg2,gasParamsStg3]),3)
588     measurementsData = np.round(np.array([measurementsStg1,measurementsStg2,measurementsStg3]),3)
589     rootData = np.round(np.array([rootVals1, rootVals2, rootVals3]),3)
590     tipData = np.round(np.array([tipVals1, tipVals2, tipVals3]),3)
591     rtMeasurements = np.round(np.array([rtMeasure1[0],rtMeasure1[1], rtMeasure1[2], rtMeasure2[3], rtMeasure2[4]]),3) #[rootInlet, tipInlet, rm, 2ndRoot(outlet), 2ndTip(outlet)]
592
593     gasParamDF = pd.DataFrame(gasParamData, index=[1,2,3], columns=[' $\alpha_1$ ', ' $\alpha_2$ ', ' $\alpha_3$ ', ' $\beta_2$ ', ' $\beta_3$ ', ' $\Delta T_{0s}$ ', 'P02/P01', 'Cw3', 'M3t', ' $\Phi$ ', ' $\Psi$ ', ' $A'$ ])
594     measurementsDF = pd.DataFrame(measurementsData, index=[1,2,3], columns=['r_t 1', 'r_t 2', 'r_t 3', 'h1', 'h2', 'h3', 'rm'])
595     rootDF = pd.DataFrame(rootData, index=[1,2,3], columns=['Ur2', 'Ur3', 'alpha2r', 'alpha3r', 'beta2r', 'beta3r', 'Cw1r', 'V2r', 'C2r', 'Cw2r', 'V3r', 'C3r', 'Cw3r', 'phiRoot', 'psiRoot', 'lambdaRoot'])
596     tipDF = pd.DataFrame(tipData, index=[1,2,3], columns=['Ut2', 'Ut3', 'alpha2t', 'alpha3t', 'beta2t', 'beta3t', 'Cw1t', 'V2t', 'C2t', 'Cw2t', 'V3t', 'C3t', 'Cw3t', 'phiTip', 'psiTip', 'lambdaTip'])
597
598     ## START HERE WITH CHECKING VALUES AFTER
599     return gasParamDF, measurementsDF, rootDF, tipDF, rtMeasurements, stage_est, Um
600     # return gasParamDF, measurementsDF, stage_est, Um
601
602
603
604 def compressorstage(self, delta_T0, React, p01, T01):
605     # Calculate relative blade angles by solving system of eqs
606     B1 = delta_T0*self.cp_c*1e3/(self.lam*self.U_m*self.C_a)
607     B2 = React**2*self.U_m/self.C_a
608     beta2 = np.arctan((B2-B1)/2)
609     beta1 = np.arctan(B1+np.tan(beta2))
610     # Calculate absolute blade angles
611     alpha1 = np.arctan(self.U_m/self.C_a - np.tan(beta1))
612     alpha2 = np.arctan(self.U_m/self.C_a - np.tan(beta2))
613     # Calculate whirl velocities
614     C_w1 = self.C_a*np.tan(alpha1)
615     C_w2 = self.C_a*np.tan(alpha2)
616     # Calculation diffusion for de Haller criteria
617     diffusion = np.cos(beta1)/np.cos(beta2)
618     # Calculate outlet pressure
619     p03 = p01*(1 + (self.n_inf_c*delta_T0)/T01)**(self.y_c/(self.y_c-1))
620     # Calculate outlet temperature
621     T02 = T01 + delta_T0
622     # Calculate inlet Mach number
623     C2 = self.C_a/np.cos(alpha1)
624     T1 = T01 - C2**2/(2*self.cp_c*1e3)
625     # M1 = self.C_a/np.cos(beta1)/np.sqrt(self.y_c*self.R*T1)
626     # Calculate outlet Mach number
627     C23 = self.C_a/np.cos(alpha2)
628     T2 = T02 - C23**2/(2*self.cp_c*1e3)
629     # M2t = self.C_a/np.cos(beta2)/np.sqrt(self.y_c*self.R*T2)

```

```

630 # Data organization
631 data = np.round(np.concatenate((np.rad2deg(np.array([alpha1,alpha2,beta1,beta2])), np.array([0.0,diffusion, 0.0,T02, p03/p01,
632 p03, 0.0, 0.0,C_w1,C_w2, React, self.lam])),3)
633 # Radius at the inlet to the rotor
634 C1 = self.C_a/np.cos(alpha1)
635 T1 = T01 - C1**2/(2*self.cp_c*1e3)
636 p1 = p01*(T1/T01)**(self.y_c/(self.y_c-1))
637 rho1 = p1*1e5/(T1*self.R)
638 A1 = self.mdot/(self.BPR+1)/(rho1*self.C_a)
639 h1 = A1/(2*np.pi*self.rm)
640 rt1 = self.rm + (h1/2)
641 rr1 = self.rm - (h1/2)
642 rr_rt1 = rr1/rt1
643 # Radius in between rotor and stator
644 p2 = p03*(T2/T02)**(self.y_c/(self.y_c-1))
645 rho2 = p2*1e5/(T2*self.R)
646 A2 = self.mdot/(self.BPR+1)/(rho2*self.C_a)
647 h2 = A2/(2*np.pi*self.rm)
648 rt2 = self.rm + (h2/2)
649 rr2 = self.rm - (h2/2)
650 rr_rt = rr2/rt2
651
652 s_data = np.round(np.array([rr_rt1,rr_rt,0.0,h1,h2,0.0]),5)
653
654 return data, s_data, p03, T02, diffusion
655
656 def turbinestage(self, alpha1, alpha3, Um, T_01, P_01, T0s_rev, psi_turb, phi, desired_lambda):
657     """
658         Inputs: alpha1 = stage inlet angle
659                 alpha3 = stage exit angle
660                 Um = mean blade speed
661                 T01 = Inlet stag. temp
662                 P01 = Inlet stag. pressure
663                 T0s_rev = revised stage temp drop
664                 psi_turb = turbine temp drop coeff.
665                 phi = flow coefficient (>=0.78)
666                 desired_lambda = desired deg. of reaction (~0.5)
667         Outputs:
668             gasParams = [alpha1,alpha2,alpha3,beta2,beta3,T0s_rev,Pr,Cw3,M3t,phi,psi_turb,Lambda]
669             measurements = [rtRat1,rtRat2,rtRat3,h1,h2,h3,rm]
670             axialVelocities = [Ca1,Cw1,Ca2,Cw2,Cw3]
671     """
672
673 lambdaN = 0.05 # nozzle loss coefficient based on experience
674
675 # Calculates the B3 and deg. of reaction
676 beta3 = np.arctan(np.tan(alpha3) + (1/phi))
677 Lambda = (2*phi*np.tan(beta3)- (psi_turb/2))/2
678 # iterates to find a suitable degree of reaction
679 if np.round(Lambda, 3) < desired_lambda:
680     while np.round(Lambda, 3) < desired_lambda:
681         alpha3 += np.deg2rad(0.01)
682         beta3 = np.arctan(np.tan(alpha3) + (1/phi))
683         Lambda = (2*phi*np.tan(beta3)- (psi_turb/2))/2
684
685 # Calculates B2 and a2
686 beta2 = np.arctan((1/(2*phi))*(psi_turb/2 - 2*Lambda))
687 alpha2 = np.arctan(np.tan(beta2) + (1/phi))
688
689 # Calculates Ca2 and C2
690 Ca2 = Um*phi
691 C2 = Ca2 / np.cos(alpha2)
692
693 # Calculates the isentropic T2', T2 and p2
694 T2 = T_01 - (C2**2/(2*self.cp_h*1e3))
695 T2prime = T2- lambdaN*(C2**2/(2*self.cp_h*1e3))
696 stagStaticRat = (T_01/T2prime)**(self.y_h/(self.y_h-1))
697 CritPrat = 1.853
698 if stagStaticRat > CritPrat:
699     print('ask the child if he or she is choking')
700
701 P2 = P_01/stagStaticRat
702
703 # Calculate the rho2 and A2
704 rho2 = (P2*100)/(0.287*T2)
705 A2 = self.mdot/(rho2*Ca2)
706
707 # Calculates the Ca1, using Ca2 = Ca3 and C1 = C3
708 Ca3 = Ca2
709 C3 = Ca3/np.cos(alpha3)
710 C1 = C3

```

```

710 Ca1 = C1*np.cos(alpha1)
711
712 # Calculates rho1 and A1
713 T1 = T_01 - (C1**2)/(2*self.cp_h*1e3))
714 P1 = P_01*(T1/T_01)**(self.y_h/(self.y_h-1))
715 rho1 = (P1*100)/(0.287*T1)
716 A1 = self.mdot/(rho1*Ca1)
717
718 # Calculates the outlet (station 3) conditions
719 T_03 = T_01 - T0s_rev
720 T3 = T_03 - (C3**2)/(2*self.cp_h*1e3)
721
722 # Calculates the P03 from the isentropic eff. and P3 from isentropic
723 P_03 = P_01*(1 - (T0s_rev/(self.n_inf_t*T_01)))**(self.y_h/(self.y_h-1))
724 P3 = P_03*(T3/T_03)**(self.y_h/(self.y_h-1))
725
726 # Calculates the rho3, A3
727 rho3 = (P3*100)/(0.287*T3)
728 A3 = self.mdot/(rho3*Ca3)
729
730
731 # Mean radius calculation
732 rm = Um/(2*np.pi*self.N)
733
734 # Calculates the h1-h3
735 h1 = (self.N/Um)*A1
736 h2 = (self.N/Um)*A2
737 h3 = (self.N/Um)*A3
738
739 # Calculates the rt/rr
740 rtRat1 = (rm + (h1/2))/(rm - (h1/2))
741 rtRat2 = (rm + (h2/2))/(rm - (h2/2))
742 rtRat3 = (rm + (h3/2))/(rm - (h3/2))
743
744 # Calculates the V2 and V3
745 V2 = Ca2*np.cos(beta2)
746 V3 = ((Um+(C3*np.cos(alpha3)))**2 + Ca3**2)**0.5
747
748 # Calculates the Cw1, Cw2 and Cw3
749 Cw1 = C1*np.sin(alpha1)
750 Cw2 = Um + V2*np.sin(beta2)
751 Cw3 = C3*np.sin(alpha3)
752
753 # Calculates the M3t to check for shocks
754 alpha3tip = np.arctan((rm/(rm + h3/2))*np.tan(alpha3))
755 C3t = Ca3*np.cos(alpha3tip)
756 Ut = self.N*2*np.pi*(rm + h3/2)
757 V3t = ((Ut+C3t)**2 + Ca3**2)**0.5
758 T3t = T_03 - (C3t**2)/(2*self.cp_h*1e3)
759 M3t = V3t/(self.y_h*self.R*T3t)**0.5
760
761 # Organizes return statements
762 Pr = P_03/P_01
763
764 gasParams = np.array([np.rad2deg(alpha1),np.rad2deg(alpha2),np.rad2deg(alpha3),np.rad2deg(beta2),np.rad2deg(beta3),T0s_rev,Pr,
Cw3,M3t,phi,psi_turb,Lambda])
765 measurements = np.array([rtRat1,rtRat2,rtRat3,h1,h2,h3,rm])
766
767 return gasParams, measurements, np.array([Ca1,Cw1,Ca2,Cw2,Cw3])
768
769 def turb_root_tip(self, meanParams, meanMeasurements, Um, axialVelocities):
770     # Pulls the mean values
771     [alpha1m, alpha2m, alpha3m, beta2m, beta3m, phiMean, psiMean] = np.deg2rad([meanParams[0],meanParams[1],meanParams[2],
meanParams[3],meanParams[4],meanParams[9],meanParams[10]])
772     [h1, h2, h3, rm] = [meanMeasurements[3], meanMeasurements[4], meanMeasurements[5], meanMeasurements[6]]
773     [Ca1,Cw1,Ca2,Cw2,Cw3] = [axialVelocities[0],axialVelocities[1], axialVelocities[2], axialVelocities[3], axialVelocities[4]]
774     Ca3 = Ca2
775     # Calculate radii
776     rt1 = rm + (h1/2); rr1 = rm - (h1/2)
777     rt2 = rm + (h2/2); rr2 = rm - (h2/2)
778     rt3 = rm + (h3/2); rr3 = rm - (h3/2)
779     # Calculate blade speed
780     Ut2 = rt2/rm * Um; Ur2 = rr2/rm * Um
781     Ut3 = rt3/rm * Um; Ur3 = rr3/rm * Um
782     # Calculates whirl velocities
783     # Cw2t = rt1/rm * Cw2; Cw2r = rr1/rm * Cw2
784     # Cw3t = rt2/rm * Cw3; Cw3r = rr2/rm * Cw3
785
786     # Calculates the angles
787 ##### Root #####
788     alpha2r = np.arctan((rm/rr2)*np.tan(alpha2m))

```

```

789     alpha3r = np.arctan((rm/rr3)*np.tan(alpha3m))
790     beta2r = np.arctan(np.tan(alpha2r) - Ur2/Ca2)
791     beta3r = np.arctan(np.tan(alpha3r) + Ur3/Ca3)
792
793     ##### Tip #####
794     alpha2t = np.arctan((rm/rt2)*np.tan(alpha2m))
795     alpha3t = np.arctan((rm/rt3)*np.tan(alpha3m))
796     beta2t = np.arctan(np.tan(alpha2t) - Ut2/Ca2)
797     beta3t = np.arctan(np.tan(alpha3t) + Ut3/Ca3)
798
799     # Calculates the velocities
800     ##### Root #####
801     V2r = Ca2/np.cos(beta2r)
802     Cw2r = Ur2 + V2r*np.sin(beta2r)
803     C2r = (Cw2r**2 + Ca2**2)**0.5
804     C3r = Ca3/np.cos(alpha3r)
805     Cw3r = C3r*np.sin(alpha3r)
806     V3r = ((Cw3r+Ur3)**2 + Ca3**2)**0.5
807     Cw1r = (rm/rr1)*Cw1
808
809     ##### Tip #####
810     V2t = Ca2/np.cos(beta2t)
811     Cw2t = Ut2 + V2t*np.sin(beta2t)
812     C2t = (Cw2t**2 + Ca2**2)**0.5
813     C3t = Ca3/np.cos(alpha3t)
814     Cw3t = C3t*np.sin(alpha3t)
815     V3t = ((Cw3t+Ut3)**2 + Ca3**2)**0.5
816     Cw1t = (rm/rt1)*Cw1
817
818     # Calculates the phi, psi, and the lambda for the tip and the root
819     phiTip = np.max(np.array([Ca2/Ut2, Ca3/Ut3]))
820     phiRoot = np.max(np.array([Ca2/Ur2, Ca3/Ur3]))
821     psiTip = 2*phiTip*(np.tan(beta2t)+ np.tan(beta3t))
822     psiRoot = 2*phiRoot*(np.tan(beta2r)+ np.tan(beta3r))
823     lambdaTip = (phiTip/2)*(np.tan(beta3t) - np.tan(beta2t))
824     lambdaRoot = (phiRoot/2)*(np.tan(beta3r) - np.tan(beta2r))
825
826     # Organizes return
827     rootValues = np.array([Ur2, Ur3, np.rad2deg(alpha2r), np.rad2deg(alpha3r), np.rad2deg(beta2r), np.rad2deg(beta3r), Cw1r, V2r,
828     C2r, Cw2r, V3r, C3r, Cw3r, phiRoot, psiRoot, lambdaRoot])
829     tipValues = np.array([Ut2, Ut3, np.rad2deg(alpha2t), np.rad2deg(alpha3t), np.rad2deg(beta2t), np.rad2deg(beta3t), Cw1t, V2t,
830     C2t, Cw2t, V3t, C3t, Cw3t, phiTip, psiTip, lambdaTip])
831     rtMeasurements = np.array([rr1, rt1, rm, rr3, rt3])
832
833     return rootValues, tipValues, rtMeasurements
834
835 def comp_root_tip(self, meantable,sizingtable,rm):
836     for i in range(0,len(meantable)):
837         # Pull blade heights
838         h1 = sizingtable['h1'][i]
839         h2 = sizingtable['h2'][i]
840         h3 = sizingtable['h3'][i]
841         # Pull mean blade angles
842         alpha1m = meantable['alpha1'][i]
843         alpha2m = meantable['alpha2'][i]
844         beta1m = meantable['beta1'][i]
845         beta2m = meantable['beta2'][i]
846         # Calculate radii
847         rt1 = rm + (h1/2); rr1 = rm - (h1/2)
848         rt2 = rm + (h2/2); rr2 = rm - (h2/2)
849         rt3 = rm + (h3/2); rr3 = rm - (h3/2)
850         # Calculate blade speed
851         Ut1 = rt1/rm * self.U_m; Ur1 = rr1/rm * self.U_m
852         Ut2 = rt2/rm * self.U_m; Ur2 = rr2/rm * self.U_m
853         # Calculate whirl velocities
854         Cw1 = meantable['Cw1'][i]; Cw2 = meantable['Cw2'][i]
855         Cw1t = rm/rt1 * Cw1; Cw1r = rm/rr1 * Cw1
856         Cw2t = rm/rt2 * Cw2; Cw2r = rm/rr2 * Cw2
857         alpha1t = np.arctan(Cw1t/self.C_a); alpha1r = np.arctan(Cw1r/self.C_a)
858         beta1t = np.arctan((Ut1-Cw1t)/self.C_a); beta1r = np.arctan((Ur1-Cw1r)/self.C_a)
859         alpha2t = np.arctan(Cw2t/self.C_a); alpha2r = np.arctan(Cw2r/self.C_a)
860         beta2t = np.arctan((Ut2-Cw2t)/self.C_a); beta2r = np.arctan((Ur2-Cw2r)/self.C_a)
861         # Calculate absolute velocities
862         C1m = self.C_a/np.cos(np.deg2rad(alpha1m)); C2m = self.C_a/np.cos(np.deg2rad(alpha2m))
863         C1t = self.C_a/np.cos(alpha1t); C2t = self.C_a/np.cos(alpha2t)
864         C1r = self.C_a/np.cos(alpha1r); C2r = self.C_a/np.cos(alpha2r)
865         # Calculate relative velocities
866         V1m = self.C_a/np.cos(np.deg2rad(beta1m)); V2m = self.C_a/np.cos(np.deg2rad(beta2m))
867         V1t = self.C_a/np.cos(beta1t); V2t = self.C_a/np.cos(beta2t)
868         V1r = self.C_a/np.cos(beta1r); V2r = self.C_a/np.cos(beta2r)
869         # Get temperatures

```

```

868     if i == 0:
869         T01 = self.T_02
870         T02 = meantable['T02'][i]
871     else:
872         T01 = meantable['T02'][i-1]
873         T02 = meantable['T02'][i]
874     # Calculate Mach number
875     T1 = T01 - C1t**2/(2*self.cp_c*1e3)
876     T2 = T02 - C2t**2/(2*self.cp_c*1e3)
877     M1t = V1t/np.sqrt(self.y_c*self.R*T1)
878     M2t = V2t/np.sqrt(self.y_c*self.R*T2)
879     # Update tables
880     if i == 0:
881         tiproot_table = pd.DataFrame(np.round(np.rad2deg([np.array([alpha1t,np.deg2rad(alpha1m),alpha1r,beta1t,
882 np.deg2rad(beta1m),beta1r,alpha2t,np.deg2rad(alpha2m),alpha2r,beta2t,np.deg2rad(beta2m),beta2r,0.0,0.0,0.0,0.0])]),2),columns=['alpha1_t','
883 alpha1_m','alpha1_r','beta1_t','beta1_m','beta1_r','alpha2_t','alpha2_m','alpha2_r','beta2_t','beta2_m','beta2_r','alpha3_t','alpha3_m',
884 'alpha3_r']))
885         whirl_table = pd.DataFrame(np.round([np.array([Cw1t,Cw1,Cw1r,Cw2t,Cw2,Cw2r,0.0,0.0,0.0,0.0])],2),columns=['Cw1_t','Cw1_m',
886 Cw1_r','Cw2_t','Cw2_m','Cw2_r','Cw3_t','Cw3_m','Cw3_r'])
887         vel_table = pd.DataFrame(np.round([np.array([C1t,C1m,C1r,C2t,C2m,C2r,0.0,0.0,0.0,V1t,V1m,V1r,V2t,V2m,V2r])],3),columns=
888 ['C1_t','C1_m','C1_r','C2_t','C2_m','C2_r','C3_t','C3_m','C3_r','V1_t','V1_m','V1_r','V2_t','V2_m','V2_r'])
889         dif_table = pd.DataFrame(np.round([np.array([V2t/V1t,V2m/V1m,V2r/V1r,0.0,0.0,0.0])],3),columns=['V2/V1_t','V2/V1_m',
890 'V2/V1_r','C3/C2_t','C3/C2_m','C3/C2_r'])
891     else:
892         data = np.round(np.rad2deg(np.array([alpha1t,np.deg2rad(alpha1m),alpha1r,beta1t,np.deg2rad(beta1m),beta1r,alpha2t,
893 np.deg2rad(alpha2m),alpha2r,beta2t,np.deg2rad(beta2m),beta2r,0.0,0.0,0.0,0.0])),2)
894         data2 = np.round(np.array([Cw1t,Cw1,Cw1r,Cw2t,Cw2,Cw2r,0.0,0.0,0.0,0.0]),2)
895         data3 = np.round(np.array([C1t,C1m,C1r,C2t,C2m,C2r,0.0,0.0,0.0,V1t,V1m,V1r,V2t,V2m,V2r]),3)
896         data4 = np.round(np.array([V2t/V1t,V2m/V1m,V2r/V1r,0.0,0.0,0.0]),3)
897         tiproot_table.loc[len(tiproot_table)] = data
898         whirl_table.loc[len(whirl_table)] = data2
899         vel_table.loc[len(vel_table)] = data3
900         dif_table.loc[len(dif_table)] = data4
901
902     meantable['M1t'][i] = np.round(M1t,3)
903     meantable['M2t'][i] = np.round(M2t,3)
904
905     for i in range(0,len(tiproot_table)-1):
906         tiproot_table['alpha3_t'][i] = tiproot_table['alpha1_t'][i+1]
907         tiproot_table['alpha3_m'][i] = tiproot_table['alpha1_m'][i+1]
908         tiproot_table['alpha3_r'][i] = tiproot_table['alpha1_r'][i+1]
909
910     for i in range(0,len(tiproot_table)):
911         whirl_table['Cw3_t'][i] = np.round(np.tan(np.deg2rad(tiproot_table['alpha3_t'][i]))*self.C_a,2)
912         whirl_table['Cw3_m'][i] = np.round(np.tan(np.deg2rad(tiproot_table['alpha3_m'][i]))*self.C_a,2)
913         whirl_table['Cw3_r'][i] = np.round(np.tan(np.deg2rad(tiproot_table['alpha3_r'][i]))*self.C_a,2)
914         vel_table['C3_t'][i] = np.round(self.C_a/np.cos(np.deg2rad(tiproot_table['alpha3_t'][i])),3)
915         vel_table['C3_m'][i] = np.round(self.C_a/np.cos(np.deg2rad(tiproot_table['alpha3_m'][i])),3)
916         vel_table['C3_r'][i] = np.round(self.C_a/np.cos(np.deg2rad(tiproot_table['alpha3_r'][i])),3)
917         dif_table['C3/C2_t'][i] = np.round(vel_table['C3_t'][i]/vel_table['C2_t'][i],3)
918         dif_table['C3/C2_m'][i] = np.round(vel_table['C3_m'][i]/vel_table['C2_m'][i],3)
919         dif_table['C3/C2_r'][i] = np.round(vel_table['C3_r'][i]/vel_table['C2_r'][i],3)
920
921     return tiproot_table, whirl_table, vel_table, meantable, dif_table
922
923 def workdone(self,stage):
924     # Loading factor calculation curve fit
925     a=0.847936849639078; b=0.15697659830655492; c=-0.2412700053204237
926     return a + b*np.exp(c*stage)
927
928 backend = TurboMachineryComputation()
929 gasParamDF, measurementsDF, rootDF, tipDF, rtMeasurements, stage_est, Um = backend.fullturbine()
930 # print('\nTurbine $\Delta$T: {}'.format(dT0_turb))
931 # print('\nStage $\Delta$T: {} K'.format(T0s))
932 # print('\nTip Mach Numbers: {}'.format(M))
933 # print('No. Turb Stages = {}'.format(stages))
934 # stuff = TurboMachineryComputation()
935 # morestuff = stuff.fullcompressor()

```