```
1  import numpy as np
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import math as mt
5  from os import system
6  from tabulate import tabulate
7
8  headers = ["Variable", "Value"]
9
10 flag = 0
11 flag2 = 0
12
13 CLmax = 0
14 Nlim = 0
15 g = 9.81
16 pi = 3.141592
17
18
19 uPos = []
20 CL = []
21
22 uGraph = np.arange(0.7, 1.4, 0.01)
23
24 ###################################
25 ###################################
26 ####### Type of aircraft #######
27 ###################################
28 ###################################
29
30
31 print("Choose the type of aircraft:")
32
33 while flag == 0:
34  print("""[1] Propeller --- [2] Turbojet""")
35  type = int(input())
36
```

```
37  if (type == 1):
38   print("You have chosen a propeller aircraft.")
39   flag = 1
40  elif (type == 2):
41   print("You have chosen a turbojet aircraft.")
42   flag = 2
43  else:
44   print("Please type only the digits [1] or [2].")
45
46
47 ##############################################################
48 ##############################################################
49 ####### Aerodynamical and Structural Restrictions #######
50 ##############################################################
51 ##############################################################
52
53 print("Are there aerodynamic and structural restrictions? [y/n]")
54
55 while flag2 == 0:
56  ans = input()
57
58  if (ans == "y"):
59   a = 0
60
61   while a == 0:
62    b = 0
63    print("Enter the value of CLmax: ")
64    CLmax = float(input())
65    print("Enter the value of Nlim: ")
66    Nlim = float(input())
67    print("You have chosen a CLmax = %f and Nlim = %f. Are these values correct
          ? [y/n]" %(CLmax, Nlim))
68
69    while b == 0:
70     typo = input()
71     if (typo == "y"):
```

```
72          a = 1
73          b = 1
74         elif (typo == "n"):
75          b = 1
76         else:
77          print("Please type only the characters [y] or [n].")
78
79       flag2 = 1
80
81      elif (ans == "n"):
82       print("There will be no aerodynamic and/or structural restrictions for the
             analysis of the aircraft.")
83       flag2 = 2
84      else:
85       print("Please type only the characters [y] or [n].")
86
87
88  ############################################################
89  ############################################################
90  ####### Equations and calculations for each case #######
91  ############################################################
92  ############################################################
93
94
95  if (flag == 1 and flag2 == 1):
96   # 1 1 = propeller with restrictions
97
98   flag3 = 0
99   while flag3 == 0:
100   c = 0
101   system('cls')
102
103   ########################
104   ### Preliminary data ###
105   ########################
106
```

```
107   print("Enter the weight of the aircraft (Newtons):")
108   W = float(input())
109   print("Enter the surface area of the wings (m^2):")
110   S = float(input())
111   print("Enter the zero lift drag coefficient (CD0):")
112   CD0 = float(input())
113   print("Enter the factor k:")
114   k = float(input())
115   print("Enter the air density (kg/m^3):")
116   rho = float(input())
117   print("Enter the efficiency of the motor (eta):")
118   eta = float(input())
119   print("Enter the power of the motor (kW):")
120   power = float(input())
121
122   table = [["W", W], ["S", S], ["CD0", CD0], ["k", k], ["rho", rho], ["eta",
          eta], ["power", power]]
123   print(tabulate(table, headers, tablefmt="grid"))
124   print("Are these values correct? [y/n]")
125
126   while c == 0:
127    secTypo = str(input())
128    if secTypo == 'y':
129     c = 1
130     flag3 = 1
131    elif secTypo == 'n':
132     c = 1
133    else:
134     print("Please type only the characters [y] or [n].")
135
136  ######################
137  ### Calculations ###
138  ######################
139
140  vRef = mt.sqrt((2*W)/(rho*S)) * mt.pow((k/CD0), 1.0/4.0)
141  Em = 1/(2*mt.sqrt(CD0*k))
```

```
142  p = (eta*power*1000*Em)/(vRef*W)
143
144  ### MSTR ###
145  rootsU = np.roots([1, 0, 0, p, -1])
146  for i in range(len(rootsU)):
147   if str(rootsU[i]).find("0j") != -1:
148    if float(rootsU[i]) > 0:
149     uMSTR = float(rootsU[i])
150
151 NMSTR = mt.sqrt((2*p*uMSTR) - mt.pow(uMSTR, 4))
152 CLMSTR = mt.sqrt(CD0/k) * (NMSTR/mt.pow(uMSTR, 2))
153
154  if (CLMSTR > CLmax or NMSTR > Nlim):
155   CLMSTR = CLmax
156   uMSTR = mt.pow((2*p)/(1 + ((k/CD0) * mt.pow(CLmax, 2))), 1.0/3.0)
157   NMSTR = mt.sqrt((2*p*uMSTR) - mt.pow(uMSTR, 4))
158   if (NMSTR > Nlim):
159    NMSTR = Nlim
160    rootsU = np.roots([1, 0, 0, -2*p, mt.pow(Nlim, 2)])
161
162    for i in range(len(rootsU)):
163     if str(rootsU[i]).find("0j") != -1:
164      if float(rootsU[i]) > 0:
165       uPos.append(float(rootsU[i]))
166
167    for root in range(len(uPos)):
168     CL.append(mt.sqrt(CD0/k) * (NMSTR/mt.pow(uPos[root], 2)))
169
170    if (max(CL) <= CLmax):
171     CLMSTR = max(CL)
172     uMSTR = min(uPos)
173    else:
174     CLMSTR = min(CL)
175     uMSTR = max(uPos)
176
177    RMSTR = (mt.pow(vRef, 2) * mt.pow(uMSTR, 2)) / (g*mt.sqrt(mt.pow(NMSTR, 2)
```

```
        - 1))
178     omegaMSTR = (g*mt.sqrt(mt.pow(NMSTR, 2) - 1)) / (vRef * uMSTR)
179     EMSTR = Em*NMSTR*uMSTR / p
180     muMSTR = mt.atan(mt.sqrt(mt.pow(NMSTR, 2) - 1)) * 180 / pi
181
182    else:
183     RMSTR = (mt.pow(vRef, 2) * mt.pow(uMSTR, 2)) / (g*mt.sqrt(mt.pow(NMSTR, 2)
        - 1))
184     omegaMSTR = (g*mt.sqrt(mt.pow(NMSTR, 2) - 1)) / (vRef * uMSTR)
185     EMSTR = Em*NMSTR*uMSTR / p
186     muMSTR = mt.atan(mt.sqrt(mt.pow(NMSTR, 2) - 1)) * 180 / pi
187   else:
188    RMSTR = (mt.pow(vRef, 2) * mt.pow(uMSTR, 2)) / (g*mt.sqrt(mt.pow(NMSTR, 2) -
        1))
189    omegaMSTR = (g*mt.sqrt(mt.pow(NMSTR, 2) - 1)) / (vRef * uMSTR)
190    EMSTR = Em*NMSTR*uMSTR / p
191    muMSTR = mt.atan(mt.sqrt(mt.pow(NMSTR, 2) - 1)) * 180 / pi
192
193
194
195   # Resetting the arrays
196   uPos = []
197   CL = []
198
199   ### SST ###
200   uSST = 2/(3*p)
201   NSST = mt.sqrt((2*uSST*p) - mt.pow(uSST, 4))
202   CLSST = mt.sqrt(CD0/k) * (NSST/mt.pow(uSST, 2))
203
204   if (CLSST > CLmax or NSST > Nlim):
205    CLSST = CLmax
206    uSST = mt.pow((2*p)/(1 + ((k/CD0) * mt.pow(CLmax, 2))), 1.0/3.0)
207    NSST = mt.sqrt((2*uSST*p) - mt.pow(uSST, 4))
208    if (NMSTR > Nlim):
209     NSST = Nlim
210     rootsU = np.roots([1, 0, 0, -2*p, mt.pow(Nlim, 2)])
```

```
211
212     for i in range(len(rootsU)):
213      if str(rootsU[i]).find("0j") != -1:
214       if float(rootsU[i]) > 0:
215        uPos.append(float(rootsU[i]))
216
217     for root in range(len(uPos)):
218      CL.append(mt.sqrt(CD0/k) * (NSST/mt.pow(uPos[root], 2)))
219
220     if (max(CL) <= CLmax):
221      CLSST = max(CL)
222      uSST = min(uPos)
223     else:
224      CLSST = min(CL)
225      uSST = max(uPos)
226
227     RSST = (mt.pow(vRef, 2) * mt.pow(uSST, 2)) / (g * mt.sqrt(mt.pow(NSST, 2) -
            1))
228     omegaSST = (g * mt.sqrt(mt.pow(NSST, 2) - 1)) / (vRef*uSST)
229     ESST = (uSST * Em * NSST) / p
230     muSST = mt.atan(mt.sqrt(mt.pow(NSST, 2) - 1)) * 180 / pi
231
232    else:
233     RSST = (mt.pow(vRef, 2) * mt.pow(uSST, 2)) / (g * mt.sqrt(mt.pow(NSST, 2) -
            1))
234     omegaSST = (g * mt.sqrt(mt.pow(NSST, 2) - 1)) / (vRef*uSST)
235     ESST = (uSST * Em * NSST) / p
236     muSST = mt.atan(mt.sqrt(mt.pow(NSST, 2) - 1)) * 180 / pi
237
238   else:
239    RSST = (mt.pow(vRef, 2) * mt.pow(uSST, 2)) / (g * mt.sqrt(mt.pow(NSST, 2) -
           1))
240    omegaSST = (g * mt.sqrt(mt.pow(NSST, 2) - 1)) / (vRef*uSST)
241    ESST = (uSST * Em * NSST) / p
242    muSST = mt.atan(mt.sqrt(mt.pow(NSST, 2) - 1)) * 180 / pi
243
```

```
244  # Resetting the arrays
245  uPos = []
246  CL = []
247
248  ### Nmax ###
249  uNmax = mt.pow((p/2), 1.0/3.0)
250  NNmax = mt.sqrt((2*p*uNmax) - mt.pow(uNmax, 4))
251  CLNmax = mt.sqrt(CD0/k) * (NNmax/mt.pow(uNmax, 2))
252
253  if (CLNmax > CLmax or NNmax > Nlim):
254   CLNmax = CLmax
255   uNmax = mt.pow((2*p)/(1 + ((k/CD0) * mt.pow(CLmax, 2))), 1.0/3.0)
256   NNmax = mt.sqrt((2*uNmax*p) - mt.pow(uNmax, 4))
257   if (NNmax > Nlim):
258    NNmax = Nlim
259    rootsU = np.roots([1, 0, 0, -2*p, mt.pow(Nlim, 2)])
260
261    for i in range(len(rootsU)):
262     if str(rootsU[i]).find("0j") != -1:
263      if float(rootsU[i]) > 0:
264       uPos.append(float(rootsU[i]))
265
266    for root in range(len(uPos)):
267     CL.append(mt.sqrt(CD0/k) * (NSST/mt.pow(uPos[root], 2)))
268
269    if (max(CL) <= CLmax):
270     CLNmax = max(CL)
271     uNmax = min(uPos)
272    else:
273     CLNmax = min(CL)
274     uNmax = max(uPos)
275
276    RNmax = (mt.pow(vRef, 2) * mt.pow(uNmax, 2)) / (g * mt.sqrt(mt.pow(NNmax,
         2) - 1))
277    omegaNmax = (g * mt.sqrt(mt.pow(NNmax, 2) - 1)) / (vRef*uNmax)
278    ENmax = Em * (mt.sqrt(3)/2)
```

```
279     muNmax = mt.atan(mt.sqrt(mt.pow(NNmax, 2) - 1)) * 180 / pi
280
281    else:
282     RNmax = (mt.pow(vRef, 2) * mt.pow(uNmax, 2)) / (g * mt.sqrt(mt.pow(NNmax,
            2) - 1))
283     omegaNmax = (g * mt.sqrt(mt.pow(NNmax, 2) - 1)) / (vRef*uNmax)
284     ENmax = Em * (mt.sqrt(3)/2)
285     muNmax = mt.atan(mt.sqrt(mt.pow(NNmax, 2) - 1)) * 180 / pi
286
287   else:
288    RNmax = (mt.pow(vRef, 2) * mt.pow(uNmax, 2)) / (g * mt.sqrt(mt.pow(NNmax, 2)
         - 1))
289    omegaNmax = (g * mt.sqrt(mt.pow(NNmax, 2) - 1)) / (vRef*uNmax)
290    ENmax = Em * (mt.sqrt(3)/2)
291    muNmax = mt.atan(mt.sqrt(mt.pow(NNmax, 2) - 1)) * 180 / pi
292
293
294   ########################
295   ## Arrays for graphs ##
296   ########################
297
298   CLGraph = []
299   omegaGraph = []
300   RGraph = []
301   NGraph = []
302
303   for val in range(len(uGraph)):
304    NGraph.append(mt.sqrt((2*p*uGraph[val]) - mt.pow(uGraph[val], 4)))
305    CLGraph.append(mt.sqrt(CD0/k) * (mt.sqrt((2*p*uGraph[val]) - mt.pow(uGraph[
         val], 4))/mt.pow(uGraph[val], 2)))
306    RGraph.append(((mt.pow(vRef, 2)*mt.pow(uGraph[val], 2))/ (g * (mt.sqrt( mt.
         pow(mt.sqrt((2*p*uGraph[val]) - mt.pow(uGraph[val], 4)), 2) - 1))) ))
307    omegaGraph.append(( (g * (mt.sqrt( mt.pow(mt.sqrt((2*p*uGraph[val]) - mt.pow
         (uGraph[val], 4)), 2) - 1))) / (uGraph[val] * vRef)))
308
309
```

```
310 elif (flag == 1 and flag2 == 2):
311  # 1 2 propeller without restrictions
312
313  flag3 = 0
314  while flag3 == 0:
315   c = 0
316   system('cls')
317
318   ##########################
319   ### Preliminary data ###
320   ##########################
321
322   print("Enter the weight of the aircraft (Newtons):")
323   W = float(input())
324   print("Enter the surface area of the wings (m^2):")
325   S = float(input())
326   print("Enter the zero lift drag coefficient (CD0):")
327   CD0 = float(input())
328   print("Enter the factor k:")
329   k = float(input())
330   print("Enter the air density (kg/m^3):")
331   rho = float(input())
332   print("Enter the efficiency of the motor (eta):")
333   eta = float(input())
334   print("Enter the power of the motor (kW):")
335   power = float(input())
336
337   table = [["W", W], ["S", S], ["CD0", CD0], ["k", k], ["rho", rho], ["eta",
          eta], ["power", power]]
338   print(tabulate(table, headers, tablefmt="grid"))
339   print("Are these values correct? [y/n]")
340
341   while c == 0:
342    secTypo = str(input())
343    if secTypo == 'y':
344     c = 1
```

```
345         flag3 = 1
346       elif secTypo == 'n':
347         c = 1
348       else:
349         print("Please type only the characters [y] or [n].")
350
351
352   ######################
353   ### Calculations ###
354   ######################
355
356   vRef = mt.sqrt((2*W)/(rho*S)) * mt.pow((k/CD0), 1.0/4.0)
357   Em = 1/(2*mt.sqrt(CD0*k))
358   p = (eta*power*1000*Em)/(vRef*W)
359
360   ### MSTR ###
361   rootsU = np.roots([1, 0, 0, p, -1])
362   for i in range(len(rootsU)):
363     if str(rootsU[i]).find("0j") != -1:
364       if float(rootsU[i]) > 0:
365         uMSTR = float(rootsU[i])
366
367   NMSTR = mt.sqrt((2*p*uMSTR) - mt.pow(uMSTR, 4))
368   CLMSTR = mt.sqrt(CD0/k) * (NMSTR/mt.pow(uMSTR, 2))
369   RMSTR = (mt.pow(vRef, 2) * mt.pow(uMSTR, 2)) / (g*mt.sqrt(mt.pow(NMSTR, 2) -
          1))
370   omegaMSTR = (g*mt.sqrt(mt.pow(NMSTR, 2) - 1)) / (vRef * uMSTR)
371   EMSTR = Em*NMSTR*uMSTR / p
372   muMSTR = mt.atan(mt.sqrt(mt.pow(NMSTR, 2) - 1)) * 180 / pi
373
374
375   ### SST ###
376   uSST = 2/(3*p)
377   NSST = mt.sqrt((2*uSST*p) - mt.pow(uSST, 4))
378   CLSST = mt.sqrt(CD0/k) * (NSST/mt.pow(uSST, 2))
379   RSST = (mt.pow(vRef, 2) * mt.pow(uSST, 2)) / (g * mt.sqrt(mt.pow(NSST, 2) -
```

```
        1))
380   omegaSST = (g * mt.sqrt(mt.pow(NSST, 2) - 1)) / (vRef*uSST)
381   ESST = (uSST * Em * NSST) / p
382   muSST = mt.atan(mt.sqrt(mt.pow(NSST, 2) - 1)) * 180 / pi
383
384   ### Nmax ###
385   uNmax = mt.pow((p/2), 1.0/3.0)
386   NNmax = mt.sqrt((2*p*uNmax) - mt.pow(uNmax, 4))
387   CLNmax = mt.sqrt(CD0/k) * (NNmax/mt.pow(uNmax, 2))
388   RNmax = (mt.pow(vRef, 2) * mt.pow(uNmax, 2)) / (g * mt.sqrt(mt.pow(NNmax, 2)
        - 1))
389   omegaNmax = (g * mt.sqrt(mt.pow(NNmax, 2) - 1)) / (vRef*uNmax)
390   ENmax = Em * (mt.sqrt(3)/2)
391   muNmax = mt.atan(mt.sqrt(mt.pow(NNmax, 2) - 1)) * 180 / pi
392
393   ########################
394   ## Arrays for graphs ##
395   ########################
396
397   CLGraph = []
398   omegaGraph = []
399   RGraph = []
400   NGraph = []
401
402   for val in range(len(uGraph)):
403     NGraph.append(mt.sqrt((2*p*uGraph[val]) - mt.pow(uGraph[val], 4)))
404     CLGraph.append(mt.sqrt(CD0/k) * (mt.sqrt((2*p*uGraph[val]) - mt.pow(uGraph[
          val], 4))/mt.pow(uGraph[val], 2)))
405     RGraph.append(((mt.pow(vRef, 2)*mt.pow(uGraph[val], 2))/ (g * (mt.sqrt( mt.
          pow(mt.sqrt((2*p*uGraph[val]) - mt.pow(uGraph[val], 4)), 2) - 1))) ))
406     omegaGraph.append(( (g * (mt.sqrt( mt.pow(mt.sqrt((2*p*uGraph[val]) - mt.pow
          (uGraph[val], 4)), 2) - 1))) / (uGraph[val] * vRef)))
407
408 elif (flag == 2 and flag2 == 1):
409   # 2 1 turbojet with restrictions
410
```

```
411  flag3 = 0
412  while flag3 == 0:
413   c = 0
414   system('cls')
415
416   ##########################
417   ### Preliminary data ###
418   ##########################
419
420   print("Enter the weight of the aircraft (Newtons):")
421   W = float(input())
422   print("Enter the surface area of the wings (m^2):")
423   S = float(input())
424   print("Enter the zero lift drag coefficient (CD0):")
425   CD0 = float(input())
426   print("Enter the factor k:")
427   k = float(input())
428   print("Enter the air density (kg/m^3):")
429   rho = float(input())
430   print("Enter the thrust of the motor (Newtons):")
431   thrust = float(input())
432
433   table = [["W", W], ["S", S], ["CD0", CD0], ["k", k], ["rho", rho], ["thrust"
          , thrust]]
434   print(tabulate(table, headers, tablefmt="grid"))
435   print("Are these values correct? [y/n]")
436
437   while c == 0:
438    secTypo = str(input())
439    if secTypo == 'y':
440     c = 1
441     flag3 = 1
442    elif secTypo == 'n':
443     c = 1
444    else:
445     print("Please type only the characters [y] or [n].")
```

```
446
447
448  ######################
449  ### Calculations ###
450  ######################
451
452  vRef = mt.sqrt((2*W)/(rho*S)) * mt.pow((k/CD0), 1.0/4.0)
453  Em = 1/(2*mt.sqrt(CD0*k))
454  z = (thrust*Em)/W
455
456  # Resetting the arrays
457  uPos = []
458  CL = []
459
460  ### MSTR ###
461  uMSTR = 1
462  NMSTR = mt.sqrt((2*z) - 1)
463  CLMSTR = mt.sqrt((CD0/k) * ((2*z) - 1))
464
465  if (CLMSTR > CLmax or NMSTR > Nlim):
466   CLMSTR = CLmax
467   uMSTR = mt.sqrt((2*z)/ ( 1 + ( (k * mt.pow(CLmax, 2)) /CD0 ) ) )
468   NMSTR = mt.sqrt( (2 * z * mt.pow(uMSTR, 2)) - mt.pow(uMSTR, 4))
469   if (NMSTR > Nlim):
470    NMSTR = Nlim
471    rootsU = np.roots([1, 0, (2*z), 0, mt.pow(Nlim, 2)])
472
473    for i in range(len(rootsU)):
474     if str(rootsU[i]).find("0j") != -1:
475      if float(rootsU[i]) > 0:
476       uPos.append(float(rootsU[i]))
477
478    for root in range(len(uPos)):
479     CL.append(mt.sqrt((mt.pow(Nlim, 2) * CD0)/ (k * mt.pow(uPos[root], 4))))
480
481    if (max(CL) <= CLmax):
```

```
482         CLMSTR = max(CL)
483         uMSTR = min(uPos)
484       else:
485         CLMSTR = min(CL)
486         uMSTR = max(uPos)
487
488       omegaMSTR = (g/vRef) * mt.sqrt(2 * (z - 1))
489       RMSTR = (mt.pow(vRef, 2) * mt.pow(uMSTR, 2)) / (g * mt.sqrt(2*(z - 1)))
490       EMSTR = (Em / z) * mt.sqrt((2*z) - 1)
491       muMSTR = mt.acos(1/(mt.sqrt((2*z) - 1))) * (180 / pi)
492
493
494     else:
495       omegaMSTR = (g/vRef) * mt.sqrt(2 * (z - 1))
496       RMSTR = (mt.pow(vRef, 2) * mt.pow(uMSTR, 2)) / (g * mt.sqrt(2*(z - 1)))
497       EMSTR = (Em / z) * mt.sqrt((2*z) - 1)
498       muMSTR = mt.acos(1/(mt.sqrt((2*z) - 1))) * (180 / pi)
499
500   else:
501     omegaMSTR = (g/vRef) * mt.sqrt(2 * (z - 1))
502     RMSTR = (mt.pow(vRef, 2) * mt.pow(uMSTR, 2)) / (g * mt.sqrt(2*(z - 1)))
503     EMSTR = (Em / z) * mt.sqrt((2*z) - 1)
504     muMSTR = mt.acos(1/(mt.sqrt((2*z) - 1))) * (180 / pi)
505
506
507   # Resetting the arrays
508   uPos = []
509   CL = []
510
511
512   ### SST ###
513   uSST = 1 / mt.sqrt(z)
514   NSST = (1/z) * mt.sqrt((2*z*z) - 1)
515   CLSST = mt.sqrt((CD0/k) * ((2*z*z) - 1))
516
517   if (CLSST > CLmax or NSST > Nlim):
```

```
518    CLSST = CLmax
519    uSST = mt.sqrt((2*z)/ ( 1 + ( (k * mt.pow(CLmax, 2)) /CD0 ) ) )
520    NSST = mt.sqrt( ((2 * z * z) - 1) / z)
521    if (NSST > Nlim):
522     NSST = Nlim
523     rootsU = np.roots([1, 0, 2*z, 0, mt.pow(Nlim, 2)])
524
525     for i in range(len(rootsU)):
526      if str(rootsU[i]).find("0j") != -1:
527       if float(rootsU[i]) > 0:
528        uPos.append(float(rootsU[i]))
529
530     for root in range(len(uPos)):
531      CL.append(mt.sqrt((mt.pow(Nlim, 2) * CD0)/ (k * mt.pow(uPos[root], 4))))
532
533     if (max(CL) <= CLmax):
534      CLSST = max(CL)
535      uSST = min(uPos)
536     else:
537      CLSST = min(CL)
538      uSST = max(uPos)
539
540     omegaSST = mt.sqrt(((z*z) - 1) / z) * (g/vRef)
541     RSST = (mt.pow(vRef, 2)/g) * (1 / mt.sqrt((z*z) - 1))
542     ESST = (Em / (z*z)) * mt.sqrt((2*z*z) - 1)
543     muSST = mt.acos(z/(mt.sqrt((2*z*z) - 1))) * (180 / pi)
544
545
546    else:
547     omegaSST = mt.sqrt(((z*z) - 1) / z) * (g/vRef)
548     RSST = (mt.pow(vRef, 2)/g) * (1 / mt.sqrt((z*z) - 1))
549     ESST = (Em / (z*z)) * mt.sqrt((2*z*z) - 1)
550     muSST = mt.acos(z/(mt.sqrt((2*z*z) - 1))) * (180 / pi)
551
552   else:
553    omegaSST = mt.sqrt(((z*z) - 1) / z) * (g/vRef)
```

```
554     RSST = (mt.pow(vRef, 2)/g) * (1 / mt.sqrt((z*z) - 1))
555     ESST = (Em / (z*z)) * mt.sqrt((2*z*z) - 1)
556     muSST = mt.acos(z/(mt.sqrt((2*z*z) - 1))) * (180 / pi)
557
558   # Resetting the arrays
559   uPos = []
560   CL = []
561
562   ### NMax ###
563   uNmax = mt.sqrt(z)
564   NNmax = z
565   CLNmax = mt.sqrt(CD0/k)
566
567   if (CLNmax > CLmax or NNmax > Nlim):
568     CLNmax = CLmax
569     uNmax = mt.sqrt((2*z)/ ( 1 + ( (k * mt.pow(CLmax, 2)) /CD0 ) ) )
570     NNmax = mt.sqrt( (2 * z * mt.pow(uNmax, 2)) - mt.pow(uNmax, 4))
571     if (NNmax > Nlim):
572       NNmax = Nlim
573       rootsU = np.roots([1, 0, 2*z, 0, mt.pow(Nlim, 2)])
574
575       for i in range(len(rootsU)):
576         if str(rootsU[i]).find("0j") != -1:
577           if float(rootsU[i]) > 0:
578             uPos.append(float(rootsU[i]))
579
580       for root in range(len(uPos)):
581         CL.append(mt.sqrt((mt.pow(Nlim, 2) * CD0)/ (k * mt.pow(uPos[root], 4))))
582
583       if (max(CL) <= CLmax):
584         CLNmax = max(CL)
585         uNmax = min(uPos)
586       else:
587         CLNmax = min(CL)
588         uNmax = max(uPos)
589
```

```
590     omegaNmax = mt.sqrt(((z*z) - 1) / z) * (g/vRef)
591     RNmax = (mt.pow(vRef, 2) * z)/(g * mt.sqrt((z*z) - 1))
592     ENmax = Em
593     muNmax = mt.acos(1/z) * (180 / pi)
594
595    else:
596     omegaNmax = mt.sqrt(((z*z) - 1) / z) * (g/vRef)
597     RNmax = (mt.pow(vRef, 2) * z)/(g * mt.sqrt((z*z) - 1))
598     ENmax = Em
599     muNmax = mt.acos(1/z) * (180 / pi)
600
601   else:
602    omegaNmax = mt.sqrt(((z*z) - 1) / z) * (g/vRef)
603    RNmax = (mt.pow(vRef, 2) * z)/(g * mt.sqrt((z*z) - 1))
604    ENmax = Em
605    muNmax = mt.acos(1/z) * (180 / pi)
606
607
608   ##########################
609   ## Arrays for graphs ##
610   ##########################
611
612   CLGraph = []
613   omegaGraph = []
614   RGraph = []
615   NGraph = []
616
617   for val in range(len(uGraph)):
618    NGraph.append(mt.sqrt((2*z*mt.pow(uGraph[val], 2)) - mt.pow(uGraph[val], 4))
         )
619    CLGraph.append((mt.sqrt(CD0/k) * ((mt.sqrt((2*z*mt.pow(uGraph[val], 2)) - mt
         .pow(uGraph[val], 4)))/mt.pow(uGraph[val], 2))))
620    RGraph.append(((mt.pow(vRef, 2)*mt.pow(uGraph[val], 2))/ (g * (mt.sqrt( mt.
         pow(mt.sqrt((2*z*mt.pow(uGraph[val], 2)) - mt.pow(uGraph[val], 4)), 2) -
         1))) ))
621    omegaGraph.append(( (g * (mt.sqrt( mt.pow(mt.sqrt((2*z*mt.pow(uGraph[val],
```

```
        2)) - mt.pow(uGraph[val], 4)), 2) - 1))) / (uGraph[val] * vRef)))
622
623
624 elif (flag == 2 and flag2 == 2):
625  # 2 2 turbojet without restrictions
626
627  flag3 = 0
628  while flag3 == 0:
629   c = 0
630   system('cls')
631   ##########################
632   ### Preliminary data ###
633   ##########################
634   print("Enter the weight of the aircraft (Newtons):")
635   W = float(input())
636   print("Enter the surface area of the wings (m^2):")
637   S = float(input())
638   print("Enter the zero lift drag coefficient (CD0):")
639   CD0 = float(input())
640   print("Enter the factor k:")
641   k = float(input())
642   print("Enter the air density (kg/m^3):")
643   rho = float(input())
644   print("Enter the thrust of the motor (Newtons):")
645   thrust = float(input())
646
647   table = [["W", W], ["S", S], ["CD0", CD0], ["k", k], ["rho", rho], ["thrust"
        , thrust]]
648   print(tabulate(table, headers, tablefmt="grid"))
649   print("Are these values correct? [y/n]")
650
651   while c == 0:
652    secTypo = str(input())
653    if secTypo == 'y':
654     c = 1
655     flag3 = 1
```

```
656     elif secTypo == 'n':
657       c = 1
658     else:
659       print("Please type only the characters [y] or [n].")
660
661   ####################
662   ### Calculations ###
663   ####################
664
665   vRef = mt.sqrt((2*W)/(rho*S)) * mt.pow((k/CD0), 1.0/4.0)
666   Em = 1/(2*mt.sqrt(CD0*k))
667   z = (thrust*Em)/W
668
669   ### MSTR ###
670   uMSTR = 1
671   NMSTR = mt.sqrt((2*z) - 1)
672   CLMSTR = mt.sqrt((CD0/k) * ((2*z) - 1))
673   omegaMSTR = (g/vRef) * mt.sqrt(2 * (z - 1))
674   RMSTR = (mt.pow(vRef, 2) * mt.pow(uMSTR, 2)) / (g * mt.sqrt(2*(z - 1)))
675   EMSTR = (Em / z) * mt.sqrt((2*z) - 1)
676   muMSTR = mt.acos(1/(mt.sqrt((2*z) - 1))) * (180 / pi)
677
678   ### SST ###
679   uSST = 1 / mt.sqrt(z)
680   CLSST = mt.sqrt((CD0/k) * ((2*z*z) - 1))
681   NSST = (1/z) * mt.sqrt((2*z*z) - 1)
682   omegaSST = mt.sqrt(((z*z) - 1) / z) * (g/vRef)
683   RSST = (mt.pow(vRef, 2)/g) * (1 / mt.sqrt((z*z) - 1))
684   ESST = (Em / (z*z)) * mt.sqrt((2*z*z) - 1)
685   muSST = mt.acos(z/(mt.sqrt((2*z*z) - 1))) * (180 / pi)
686
687   ### NMax ###
688   uNmax = mt.sqrt(z)
689   CLNmax = mt.sqrt(CD0/k)
690   NNmax = z
691   omegaNmax = mt.sqrt(((z*z) - 1) / z) * (g/vRef)
```

```
692   RNmax = (mt.pow(vRef, 2) * z)/(g * mt.sqrt((z*z) - 1))
693   ENmax = Em
694   muNmax = mt.acos(1/z) * (180 / pi)
695
696   ##########################
697   ## Arrays for graphs ##
698   ##########################
699
700   CLGraph = []
701   omegaGraph = []
702   RGraph = []
703   NGraph = []
704
705   for val in range(len(uGraph)):
706     NGraph.append(mt.sqrt((2*z*mt.pow(uGraph[val], 2)) - mt.pow(uGraph[val], 4))
          )
707     CLGraph.append((mt.sqrt(CD0/k) * ((mt.sqrt((2*z*mt.pow(uGraph[val], 2)) - mt
          .pow(uGraph[val], 4)))/mt.pow(uGraph[val], 2))))
708     RGraph.append(((mt.pow(vRef, 2)*mt.pow(uGraph[val], 2))/ (g * (mt.sqrt( mt.
          pow(mt.sqrt((2*z*mt.pow(uGraph[val], 2)) - mt.pow(uGraph[val], 4)), 2) -
          1))) ))
709     omegaGraph.append(( (g * (mt.sqrt( mt.pow(mt.sqrt((2*z*mt.pow(uGraph[val],
          2)) - mt.pow(uGraph[val], 4)), 2) - 1))) / (uGraph[val] * vRef)))
710
711
712 ################
713 #### TABLE ####
714 ################
715
716 x = np.linspace(0, 2 * np.pi, 400)
717 y = np.sin(x ** 2)
718
719
720
721
722 fig, axs = plt.subplots(2, 2)
```

```
723
724
725 plt.figure(1)
726
727 axs[0, 0].plot(uMSTR, CLMSTR, marker='x', color='r', label='CL MSTR')
728 axs[0, 0].plot(uSST, CLSST, marker='x', color='b', label='CL SST')
729 axs[0, 0].plot(uNmax, CLNmax, marker='x', color='g', label='CL Nmax')
730 axs[0, 0].plot(uGraph, CLGraph)
731 axs[0, 0].axhline(y=CLmax, color='c', linestyle='-', label='CLmax')
732 axs[0, 0].set_title("Cl vs. u")
733 axs[1, 0].plot(uMSTR, omegaMSTR, marker='x', color='r', label='omega MSTR')
734 axs[1, 0].plot(uSST, omegaSST, marker='x', color='b', label='omega SST')
735 axs[1, 0].plot(uNmax, omegaNmax, marker='x', color='g', label='omega Nmax')
736 axs[1, 0].plot(uGraph, omegaGraph)
737 axs[1, 0].set_title("omega vs. u")
738 axs[0, 1].plot(uMSTR, RMSTR, marker='x', color='r', label='R MSTR')
739 axs[0, 1].plot(uSST, RSST, marker='x', color='b', label='R SST')
740 axs[0, 1].plot(uNmax, RNmax, marker='x', color='g', label='R Nmax')
741 axs[0, 1].plot(uGraph, RGraph)
742 axs[0, 1].set_title("R vs u")
743 axs[1, 1].plot(uMSTR, NMSTR, marker='x', color='r', label='N MSTR')
744 axs[1, 1].plot(uSST, NSST, marker='x', color='b', label='N SST')
745 axs[1, 1].plot(uNmax, NNmax, marker='x', color='g', label='N Nmax')
746 axs[1, 1].plot(uGraph, NGraph)
747 axs[1, 1].axhline(y=Nlim, color='c', linestyle='-', label='Nlim')
748 axs[1, 1].set_title("N vs u")
749 axs[0, 0].legend()
750 axs[1, 0].legend()
751 axs[0, 1].legend()
752 axs[1, 1].legend()
753
754 fig.tight_layout()
755
756
757
758
```

```
759 fig, ax = plt.subplots(dpi=100)
760
761 # Hide axes
762 plt.figure(2)
763 fig.patch.set_visible(False)
764 ax.axis('off')
765 ax.axis('tight')
766
767 tableVariables=np.array(["u","omega (rad/s)","R (m)","N","Cl","E","Mu (deg)"])
768
769 mstrData=np.around(np.array([uMSTR, omegaMSTR, RMSTR, NMSTR, CLMSTR, EMSTR,
        muMSTR]),6)
770 string_magnitudes=str(mstrData)
771
772 sstData=np.around(np.array([uSST, omegaSST, RSST, NSST, CLSST, ESST, muSST])
        ,6)
773 string_magnitudes=str(sstData)
774
775 nmaxData=np.around(np.array([uNmax, omegaNmax, RNmax, NNmax, CLNmax, ENmax,
        muNmax]),6)
776 string_magnitudes=str(nmaxData)
777
778
779 df = pd.DataFrame({'MSTR' : mstrData.tolist(), 'SST' : sstData.tolist(), 'Nmax
        ' : nmaxData.tolist()})
780 ax.table(cellText=df.values, rowLabels=tableVariables, colLabels=df.columns,
        cellLoc='center', loc='center')
781 fig.tight_layout()
782 ####
783
784
785 plt.show()
```