File - C:\Users\geoff\Desktop\FlyOx Concept\Python\API\wingAPI.py

```
from math import sqrt,cos,sin,pi,log,tan
import numpy as np
import matplotlib.pyplot as plt
# import sys
# import os


class aspectRatio:
    def __init__ (self, initialWeight, finalWeight,cruiseSpeed,S,rhoSL,altitude,cbhp,propEff,Range,cdMin,endurance,ldMax):
        self.initialWeight = initialWeight
        self.finalWeight = finalWeight
        self.cruiseSpeed = cruiseSpeed
        self.S = S
        self.rhoSL = rhoSL
        self.altitude = altitude
        self.averageWeight = (initialWeight+finalWeight)/2
        self.cbhp = cbhp
        self.propEff = propEff
        self.Range = Range
        self.cdMin = cdMin
        self.endurance = endurance
        self.ldMax = ldMax

    def altitudeDensity(self):
        ### fetch this function in other script
        altitudeDensity = self.rhoSL*(1-0.0000068756*self.altitude)**4.2561
        return altitudeDensity

    def cruiseCL (self):
        #### make a function to calculate cl
        cruiseCL = (2* self.averageWeight) / (aspectRatio.altitudeDensity(self)*self.S* self.cruiseSpeed**2)
        return cruiseCL

    def ct (self):
        ct = (self.cbhp * self.cruiseSpeed) / (1980000 * self.propEff)
        return ct

    def rangeAR (self):
        ###include range in the init function
        rangeAR = ((aspectRatio.cruiseCL(self))**2 ) / (pi*(( (self.cruiseSpeed*aspectRatio.cruiseCL(self)* log(self.initialWeight/self.finalWeight))/(self.Range * aspectRatio.ct(self))) - self.cdMin) )

        return rangeAR

    def enduranceAR(self):
        enduranceAR = ((aspectRatio.cruiseCL(self))**2 ) / (pi*(( (aspectRatio.cruiseCL(self)* log(self.initialWeight/self.finalWeight))/(self.endurance*3600 * aspectRatio.ct(self))) - self.cdMin) )

        return enduranceAR

    ## put some if-else statements for the sailplane category and remember to check the automatic AR<36 rule

    def unPoweredSailplaneAR(self):
        unPoweredSailplaneAR = 44.482 - sqrt(1672.2-28.41*self.ldMax)
        return unPoweredSailplaneAR

    def poweredSailplaneAR(self):
        poweredSailplaneAR = (self.ldMax + 0.443)/1.7405
        return poweredSailplaneAR


class wingDimensions:

    def __init__ (self, S , AR, taper):
        self.AR = AR
        self.S = S
        self.taper = taper

    def wingSpan (self):
        wingspan = sqrt (self.AR *self.S)
        return wingspan

    def Cavg (self):
        Cavg = wingDimensions.wingSpan(self)/self.AR
```

```
 71         return Cavg
 72
 73     def rootChord(self):
 74         rootChord = 2* wingDimensions.Cavg(self)/ (1+self.taper)
 75         return rootChord
 76
 77     def tipChord (self):
 78         tipChord = self.taper*wingDimensions.rootChord(self)
 79         return tipChord
 80
 81     def meanGeometricChord (self):
 82         meanGeometricChord = (2/3)*(wingDimensions.rootChord(self))*((1+self.taper+self.taper**2)/(1+
    self.taper))
 83         return meanGeometricChord
 84
 85     def chordAtY (self,y):
 86         chordAtY = wingDimensions.rootChord(self)*(1+(2*(self.taper - 1)*y/wingDimensions.wingSpan(
    self)))
 87         return chordAtY
 88
 89     def yMGC (self):
 90         yMGC = (wingDimensions.wingSpan(self)/6)*((1+2*self.taper)/(1+self.taper))
 91         return yMGC
 92
 93
 94
 95 class classOswaldEff:
 96
 97     def __init__ (self, rangeAR , sweepLeadingEdge, sweepTmax,fuselageWidth,wingSpan,cdMin):
 98         self.rangeAR = rangeAR
 99         self.sweepLeadingEdge = sweepLeadingEdge
100         self.sweepTmax = sweepTmax
101         self.fuselageWidth = fuselageWidth
102         self.wingSpan = wingSpan
103         self.cdMin = cdMin
104
105     def straightWingOswaldEff(self):
106         straightWingOswaldEff = 1.78 * (1 - 0.045 * self.rangeAR ** 0.68) - 0.64
107         return straightWingOswaldEff
108
109     def sweptWingOswaldEff(self):
110         sweptWingOswaldEff = 4.61 * (1 - 0.045 * self.rangeAR ** 0.68) * (cos(self.sweepLeadingEdge /
     57.3)) ** 0.15 - 3.1
111         return sweptWingOswaldEff
112
113     def brandtOswaldEff(self):
114         brandtOswaldEff = 2 / (2 - self.rangeAR + sqrt(4 + (self.rangeAR ** 2) * (1 + (tan(self.
    sweepTmax  / 57.3)) ** 2)))
115         return brandtOswaldEff
116
117     def douglasOswalfEff(self):
118         r = 0.38 - (self.sweepLeadingEdge / 3000) + (self.sweepLeadingEdge ** 2 / 15000)
119         u = 0.99
120         t = self.fuselageWidth / self.wingSpan
121         douglasOswalfEff = 1 / ((pi * self.rangeAR * r * self.cdMin) + ((1 + 0.03 * t - (2 * t ** 2))
     * u))
122         return douglasOswalfEff
123
124         ## read Gudmundsson example 9-12 and compute USAF DATCOM method u lazy bastard
125
```