```python
__author__ = 'Geoffrey Nyaga'

import sys
sys.path.append('../')
from API.db_API import write_to_db,read_from_db

import numpy as np
import matplotlib.pyplot as plt
import math


import API.TESTING_MAINAPI as tmapi


Range = read_from_db('Range')
propEff = read_from_db('propEff')
AR= read_from_db('AR')
pax= read_from_db('pax')
crew= read_from_db('crew')


wtoGuess = np.arange(1000,6500,1)
#Gudmundsson
# weWtoGud = 0.4074 + 0.0253 * np.log(wtoGuess)
# print(weWtoGud)
#use this when using Gudmundsson sizing and constants

paxWeight = read_from_db('paxWeight')
crewWeight = read_from_db('crewWeight')
payloadPax=read_from_db('payloadPax')

paxTotal=pax*paxWeight
payload = (payloadPax*pax)+paxTotal #total payload
crewTotal = crew*crewWeight


## also in input  main file, decide what to import and from which file
oswaldeff=1.78*(1-0.045*AR **0.68)-0.64 # e is oswalds span efficiency factor 0.7-0.95 #
k=1/(np.pi*oswaldeff*AR) # k is the induced drag factor k=1/(pi*e*AR) #

cdo=0.025  #zerolift drag coefficient cdo = 0.022 - 0.028#
ldmax1=2*np.sqrt(k*cdo)
ldMax=ldmax1 **(-1)


write_to_db('cdo',cdo)
write_to_db('ldMax',ldMax)
write_to_db('k',k)

Vc = read_from_db("maxSpeed")/1.2   #AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
cbhp = 0.4
fuelAllowance = 5 # in %

write_to_db('cbhp',cbhp)

w4w3 = math.exp((-Range*3280.8399*cbhp/3600)/(propEff*ldMax*550))
w2w1= 0.98
w3w2= 0.97
w5w4= 0.99
w6w5=0.997
w6w1=w2w1*w3w2*w4w3*w5w4*w6w5
# print(w4w3,"w4w3")
wfWto=((100+fuelAllowance)/100)*(1-w6w1)

wfWtoRoskam = (1+(fuelAllowance/100))*(1 - w4w3*0.992*0.992*0.996*0.99*0.992*0.992)
wfWtoRaymer = (1+(fuelAllowance/100))*(1 - w4w3*0.97*0.985*0.995)
wfWtoGud = (1+(fuelAllowance/100))*(1 - w4w3*0.994*0.985*0.996*0.995)
wfWtoSadraey = (1+(fuelAllowance/100))*(1 -w2w1*w3w2*w4w3*w5w4*w6w5 )

#empty weight constants
sizingConstantA= 1.51
sizingConstantB= -0.1

#Raymer
weWto = sizingConstantA*(wtoGuess**sizingConstantB)
wtoYaxisRaymer=(payload+crewTotal)/(1-wfWtoRaymer-weWto)

```

```python
#Roskam
wtoYaxisRoskam=(payload+crewTotal)/(1-wfWtoRoskam-weWto)

#Sadraey
wtoYaxisSadraey=(payload+crewTotal)/(1-wfWtoSadraey-weWto)

#Gudmundsson
wtoYaxisGud=(payload+crewTotal)/(1-wfWtoGud-weWto)


plt.plot(wtoGuess,wtoGuess)
plt.plot(wtoGuess,wtoYaxisRaymer, label ="Raymer")
plt.plot(wtoGuess,wtoYaxisGud, label = "Gudmundsson")
plt.plot(wtoGuess,wtoYaxisRoskam, label = "Roskam")
plt.plot(wtoGuess,wtoYaxisSadraey, label= "Sadraey")

idx = np.argwhere(np.diff(np.sign(wtoGuess-wtoYaxisRaymer))!=0).reshape(-1)+0
plt.plot(wtoGuess[idx], wtoYaxisRaymer[idx], 'ro')

idx = np.argwhere(np.diff(np.sign(wtoGuess-wtoYaxisGud))!=0).reshape(-1)+0
plt.plot(wtoGuess[idx], wtoYaxisGud[idx], 'ro')

idx = np.argwhere(np.diff(np.sign(wtoGuess-wtoYaxisRoskam))!=0).reshape(-1)+0
plt.plot(wtoGuess[idx], wtoYaxisRoskam[idx], 'ro')

idx = np.argwhere(np.diff(np.sign(wtoGuess-wtoYaxisSadraey))!=0).reshape(-1)+0
plt.plot(wtoGuess[idx], wtoYaxisSadraey[idx], 'ro')

plt.xlabel("Wto Guess")
plt.ylabel("Wto")
plt.title("WEIGHT SIZING CONSIDERING VARIOUS FUEL FRACTIONS \n But the sizing constants are Raymer's")
plt.legend()
if __name__ == '__main__':
    plt.show()

d = wtoYaxisRaymer[idx]
e = wtoYaxisGud[idx]
f = wtoYaxisRoskam[idx]
g = wtoYaxisSadraey[idx]

h= np.array([d,e,f,g])
finalMTOW = np.mean(h)
print("Raymer",d,"Gudmundsson",e,"Roskam",f,"Sadraey",g)
print(finalMTOW,"LBS <<-- final MTOW")

write_to_db('finalMTOW',finalMTOW)


initialWeight = tmapi.initialWeight(finalMTOW)
write_to_db('initialWeight',initialWeight)

finalWeight = initialWeight*w4w3
write_to_db('finalWeight',finalWeight)
# KENYA ONE PROJECT #
# Python code to solve for MTOW,other weights,design point plot,#
# Cl calculations and Vn diagram.#
        #----------------------------------------------------------------#
                 # done by Geoffrey Nyaga Kinyua
        #----------------------------------------------------------------#

##SEE IF OSWALDEFF CAN BE IMPORTED PRIOR TO THIS
# breguet range equation

# b=a/(n*ldMax)
# cfraction=np.exp(b)  # this is w4/w3 #
#d = 0.98*0.97*w4w3*0.99*0.997  # this will give W6/W1 #
#e=1.05*(1-w6w1)    # Wf/Wto = 1.05(1 - W6/W1) #
# f = A*(Wto **C)*Kvs     f is We/Wto  ......equation 1#
# f = d - (payload+crewTotal)/Wto  ......... ....................equation 2 #
# print(wfWto*100 , "% fuel of the total weight")


mtow=read_from_db('finalMTOW')
Wf=mtow*wfWto
We=mtow-(Wf+crewTotal+payload)
# print(' The value of MTOW is ' + str(mtow) + ' lbs')
```

```
# print(' The value of FUEL is ' + str(Wf)+' lbs')
# print(' The value of WE is ' + str(We)+' lbs')

write_to_db('emptyWeight', We)



# print( '____________________________________________________ ')
# print( '   NOW ENTER THE INITIAL PERFORMANCE DATA ESTIMATES  ')
# print( ' ___________________________________________________ ')

# h= float(input(' Ceiling (ft) ===>  ') )
# vmaxe=float(input(' Vmax (knots) ===>  ') )
# sto=float(input(' Take-Off Run (ft) ===>  ') )
# vstall=float(input(' Stall speed (61knots max) ===>  ') )
# rateOfClimb_estimate= float(input(' rate of climb (m/s) ===>  ') )

h= read_from_db('ceiling')
vmaxe=read_from_db('maxSpeed')
sto=read_from_db('takeOffRun')
vstall=read_from_db('stallSpeed')
rateOfClimb_estimate= read_from_db('rateOfClimb')

# print( '                          ')
# print( 'A GRAPH OF POWER LOADING VS WING LOADING IS SHOWN HERE' )
# print( '                          ')
# print( '- - - - - - - - - - - - - - - - - - - - - - - - - - ' )
# print( 'PLEASE READ THE GRAPH AND FILL IN THE VALUES BELOW')
# print( '- - - - - - - - - - - - - - - - - - - - - - - - - - ' )
# print( '                          ')

# WS = W/S WING LOADING lb/ft **2 #
# WP = W/P POWER LOADING lb/hp #

                   #Vmax calculations#

# start = 5
# end = 30
# interval = 1

# ws = np.arange(start,end,interval)


propEff = 0.7 # AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
rhoSL = read_from_db('rhoSL')
vmax = vmaxe*1.688 # we have assumed it is 150 knots

# AR=7.5 AR is the aspect ratio 5-9 for GA aircraft #
# e=0.8 e is oswalds span efficiency factor 0.7-0.95 #

#k=1/(np.pi*oswaldeff*AR)
altitude = read_from_db('cruise_altitude')

# write_to_db('altitude',altitude)


# altitudeDensity=0.001756    #USE THE ALTITUDE DENSITY FUNCTION ##CHECK
altitudeDensity = tmapi.altitudeDensity(altitude,rhoSL)
write_to_db('altitudeDensity',altitudeDensity)

# rade=(1- (h*6.873*10 **(-6))) ** 4.26
# d1=0.5*rhoSL*(vmax **3)*cdo
# d4a=d1/ws
# d2=(2 * k) /(altitudeDensity*rade*vmax)
# nume=0.7*550
# wpvmax =nume/(d4a+(d2*ws))


# TAKE-OFF RUN CALCULATION#
vto=1.1*vstall*1.688


U=0.04
CLC=.28 # MATLAB??? AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CLFLAP=0.8
CLTO=CLC+CLFLAP

```

```
write_to_db('CLTO',CLTO)

CDOLG=0.009
CDOHLD=0.007
CDOTO=cdo+CDOLG+CDOHLD
CDTO=CDOTO+k*CLTO **2
CLR=1.8/1.1 **2
CDG=CDTO-U*CLTO
# mf4=0.6*rhoSL*32.17*CDG*sto
# mf2=ws/(mf4)
# mf3=mf2 **-1
# mf=2.71828183**(mf3)
# mf1=U+(CDG/CLR)
# mf5=(1-mf)
# mf6=mf1*mf
# mff=(mf5)/(U-(mf6))
# wptor= (mff*0.6*550)/(vto)

# SERVICE CEILING CALCULATION #
# e1= math.sqrt (3*cdo/k)
# f=2/(altitudeDensity*e1)
# h=np.sqrt(f*ws)
# g=1.155/(ldMax*0.7*550)
# i=g*h
# j=i/rade
# l=j**-1
# wpc=l

#RATE OF CLIMB CALCULATION#

# rateOfClimb_estimate1=rateOfClimb_estimate*3.28084
# f1=2/(rhoSL*e1)
# h1=np.sqrt(f1*ws)
# i=g*h1
# h2=rateOfClimb_estimate1/(0.7*550)
# i1=h2+i
# l=i1**-1
# wprateOfClimb=l

#Vstall calculations#
# WP=np.arange(1,51)
clmax=1.8 #CAN THIS BE IMPORTED LATER???
vs=vstall*1.688  #vs is stall speed and the minimun by law is 61knots#
WS3 = 0.5*rhoSL*clmax*vs **2
#clmax is between 1.6-2.2 so we take 1.6 #
#plot(WS3 , WP)#
# shadeline = np.full(int((end-start)/interval), 40)


# plt.plot(ws , wpc , label = 'ceiling')
# plt.plot(ws , wptor, label = 'Take-Off Run')
# plt.plot(ws , wpvmax, label = 'Max Speed')
# plt.plot(ws , wprateOfClimb, label = 'Rate of Climb')
# # plt.plot(ws , shadeline)


# plt.fill_between(ws, wpc,shadeline,color='k',alpha=.5)
# plt.fill_between(ws, wptor,shadeline,color='y',alpha=.5)

# # plt.plot(x, y, marker='.', lw=1)
# # d = scipy.zeros(len(y))
# plt.fill_between(ws,wpvmax, where = wpvmax>=0)
# plt.fill_between(ws,wprateOfClimb, where = wprateOfClimb>=0)
# # ax.fill_between(xs, ys, where=ys<=d, interpolate=True, color='red')

# plt.axvline(x=WS3)
# plt.legend()
# plt.show()


# def find_nearest(array,value):
#     idx = (np.abs(array-value)).argmin()
#     return idx

# myidx = find_nearest(ws, WS3)


```

```
# def design_point():

#     squares = []

#     maxspeedidx = wpvmax[myidx]
#     takeoffidx = wptor[myidx]
#     climbidx = wprateOfClimb[myidx]
#     ceilingidx = wpc[myidx]

#     myarray = np.array([maxspeedidx,takeoffidx,climbidx,ceilingidx])

#     for x in myarray:
#         mynum = maxspeedidx

#         if x >= mynum:
#             squares.append(x)
#             # print (squares)
#         else:
#             # print ('oops')
#             # return maxspeedidx
#             pass

#             for j in squares:
#                 mynum = takeoffidx
#                 if j > mynum:
#                     squares.remove(j)

#                 else:
#                     # print('whats the problem here')
#                     pass

#                 for k in squares:
#                     mynum = climbidx
#                     if k < mynum:
#                         squares.remove(k)

#                     else:
#                         pass

#                     for l in squares:
#                         mynum = ceilingidx
#                         if l > mynum:
#                             squares.remove(k)

#                         else:
#                             pass

#     # print(squares,"2nd")
#     return squares[0]

# WP = design_point()

# print (WP,"This is the computer generated WP")



#*******************************************PART TWO
 *********************************************************** #
#*******************************************AIRFOIL PARAMETERS
 ************************************************* #

# x = float(input (' enter the value of w/p -> ') )
# x1 = float(input (' enter the value of w/s -> ') )

write_to_db('WS',WS3)

import constraint

WP = (mtow)/(read_from_db('finalBHP'))

write_to_db('WP',WP)
x = read_from_db('WP')

x1 = WS3

# write_to_db('WP',x)
S=mtow/(x1*10.57)
```

```python
write_to_db('S',S)
# P=mtow/x
# write_to_db('P',P)

# print( ' Wing Surface Area =   ' + str(S)+ ' sq. m' )
# print( ' Engine Power =   ' + str(P)+' horsepower' )

                     # RESOLVING FOR FINAL VALUES #
# Vs RESOLVE#
vs1= (x1)/(0.5*rhoSL*clmax)
Vs2= math.sqrt (vs1)
stallSpeed= Vs2/1.688
write_to_db('stallSpeed',stallSpeed)

# Vmax RESOLVE #
x2=(0.7*550)/x
c=6.873*10 **-6
rade=(1-c*10000)**4.26
y=(0.5*rhoSL*cdo)/x1

z=(2*k*x1)/(altitudeDensity*rade)
z1= [ y, 0, 0, -x2, z]
s = np.roots (z1)
(print(s,"-----------------"))
#z=s[3]
z=np.max(s)
z1=abs(z)
# maxSpeed=z1/1.688
# write_to_db('maxSpeed',maxSpeed)

# Take-off Run Resolve
a= ((x*vto)/(0.6*550))
#b= exp (0.6*rhoSL*32.17*CDG*sto/x1)
c=U+(CDG/CLR)
d=(1-a*U)/(1-a*c)
sto1=np.log(d)
takeOffRun=sto1/(0.6*rhoSL*32.17*CDG/x1)
write_to_db('takeOffRun',takeOffRun)

#Rate Of Climb Resolve
k=1/(np.pi*oswaldeff*AR)
e1= np.sqrt (3*cdo/k)
f1=2/(rhoSL*e1)
h1=np.sqrt(f1*x1)
g=1.155/(ldMax*0.7*550)
i=g*h1
rateOfClimb1=(1-(x*i))*0.7*550/x
rateOfClimb=rateOfClimb1*0.3048
# write_to_db('rateOfClimb',rateOfClimb)
cruiseSpeed = read_from_db("maxSpeed")/1.2

print(cruiseSpeed,"cruiseSpeed")

vc=z1/1.2
write_to_db('cruiseSpeed',cruiseSpeed)
vs=Vs2
S1=S*10.76
#Wave=0.5*(mtow*2)
wbc=0.98*0.97*mtow
wec=wbc*w4w3
Wave=(wbc+wec)/2
altitudeDensity=0.001756 #AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

clc=(2*Wave)/(altitudeDensity*S1*(cruiseSpeed*1.688)**2)
print("altitudeDensity",altitudeDensity)
print("Wave",Wave)
print ("S1",S1)
print("clc",clc)
write_to_db('clc',clc/1.688)
clcw=clc/0.95
cli=clcw/0.9
write_to_db('cli',cli)

clmaxn=(2*mtow)/(rhoSL*S1*vs **2)
clmaxw=clmaxn/0.95
clmaxgross=clmaxw/0.9
write_to_db('clmaxgross',clmaxgross)
```

```
netclmax=clmaxgross-0.6  #cfc = cf/c #
write_to_db('netclmax',netclmax)

# print( '- - - - - - - - - - - - - - - - - - - - - - - - - - - - ' )
# print( '                CALCULATED PERFORMANCE VALUES                          ')
# print( '- - - - - - - - - - - - - - - - - - - - - - - - - - - - ' )
# print( '                          ')
# print( '    a) WING SURFACE AREA (in sq. meter) is ' + str(S) )
# # print( '    b) POWER REQUIRED (in horsepower)  is '  + str(P) )
# print( '    c) AIRCRAFT STALL SPEED (in knots) is '  + str(stallSpeed) )
# print( '    d) AIRCRAFT MAX. SPEED (in knots) is '  + str(maxSpeed) )
# print( '    e) AIRCRAFT Take-Off RUN  (in ft ) is '  + str(takeOffRun) )
# print( '    f) AIRCRAFT Rate Of Climb   (in m/s) is '  + str(rateOfClimb) )



# print( '*********USE THE BELOW VALUES FOR AEROFOIL SELECTION PROCESS *****************')
# print( '*********USE THE cli_clmax.doc document provided to match ****************')
# print( '                          ')

# print( '   a) IDEAL LIFT COEFFICIENT (cli) is  '  + str(cli) )
# print( '   b) NET MAX. LIFT COEFFICIENT (Clmax) is '  + str(netclmax) )

#WING PARAMETERS#

wingspan= np.sqrt (AR*S)
wmeanchord= wingspan/AR
wtaper=1#AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
wcroot=(wmeanchord*3) / (2*((1+wtaper+wtaper **2)/(1+wtaper)))
wctip=wtaper*wcroot

write_to_db('wingSpan',wingspan*3.2808)
write_to_db('averageChord',wmeanchord*3.2808)
write_to_db('taper',wtaper)
write_to_db('tipChord',wctip*3.2808)
write_to_db('rootChord',wcroot*3.2808)
#re=(stallSpeed*1.688*wmeanchord/3.28084)/(1.460*10 **-5)

# print( '   i) WINGSPAN     '  + str(wingspan) + ' m')
# print( '   ii) MEAN CHORD LENGTH     '  + str(wmeanchord) + ' m')
# print( '   iii) WING ROOT LENGTH   '  + str(wcroot) +' m' )
# print( '   iv) WING TIP LENGTH   '  + str(wctip) +' m' )
#print( ['   v) AIRFOIL REYNOLDS NUMBER   is  ' , + str(re) ])

#tire sizing
ww=0.9*mtow*0.5
mwdiameter=1.51*ww **0.349
mwwidth=0.7150*ww **0.312

# print( '  TIRE SIZING')
# print( '          ')

# print( '   main wheel diameter     '  + str(mwdiameter) + ' in')
# print( '   main wheel width     '  + str(mwwidth) + ' in')

write_to_db('mainWheelDiameter',mwdiameter)
write_to_db('mainWheelWidth',mwwidth)


#fuselage sizing
lfus=(0.86*(mtow) **0.42)/3.28084
write_to_db('fuselageLength',lfus*3.2808)

print(payload+crewTotal)
```