```python
__author__ = 'Geoffrey Nyaga'

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

from math import sqrt,pi
import numpy as np
import matplotlib.pyplot as plt

grossWeight = read_from_db('finalMTOW')
cruiseSpeed = read_from_db('cruiseSpeed')
ROC = read_from_db('rateOfClimb')*3.28*60
vLof = read_from_db('stallSpeed')*1.1
AR = read_from_db('AR')
cdMin = read_from_db('cdMin')
wsfromsizing = read_from_db('WS')
rhoSL = read_from_db('rhoSL')
propEff = read_from_db('propEff')

cruiseAltitude = 10000 #ft
gForce = 2
V_ROC = 80
groundRun = 900
serviceCeiling = 18000
wsInitial = 22.6 #lb/f**2
g = 32.174
CDto = 0.04
CLto = 0.5
groundFriction = 0.04


def oswaldEff (AR):
    e= (1.78*(1-(0.045*AR**0.68)))-0.64
    return e

e = oswaldEff(AR)


k = 1/(pi * AR * e)


write_to_db('k', k)

#dynamic pressure at altitude
def rhoAlt(cruiseAltitude):
    rhoalt = rhoSL*(1-0.0000068756*cruiseAltitude)**4.2561
    return rhoalt

rhoCruise = rhoAlt(cruiseAltitude)
# print ('air density at cruise altitude, rho = ' +str(rhoCruise))

qAltitude = 0.5*rhoCruise*(1.688*cruiseSpeed)**2
# print('dynamic pressure at altitude = ' +str(qAltitude))

#Gag Ferrar Model
def gagFerrar(bhp):
    normBhp=bhp/(1.132*(rhoCruise/rhoSL)-0.132)
    return normBhp

WS = np.arange(10,30)

twTurn = qAltitude*( (cdMin/WS)+ k*(gForce/ qAltitude)**2 *(WS) )
qROC = 0.5*rhoSL*(V_ROC*1.688)**2
Vv = ROC/60
twROC = ( (Vv/(V_ROC*1.688)) + (qROC*cdMin/WS)+(k*WS/qROC))
qVlof = 0.5*rhoSL*(vLof*1.688/sqrt(2))**2
twVlof = ((vLof*1.688)**2/(2*g*groundRun))+(qVlof*CDto/WS)+(groundFriction*(1-(qVlof*CLto/WS)) )

rhoCeiling = rhoAlt(serviceCeiling)
# print(rhoCeiling)
twCruise = qAltitude*cdMin*(1/WS) + (k)

twCeiling = (1.667/(np.sqrt((2*WS/rhoCeiling)*sqrt(k/3*cdMin))))+((k*cdMin/3)*4)

plt.figure(1)
plt.subplot(121)
```

```

plt.plot(WS,twTurn, label = 'Rate of Turn')
plt.plot(WS,twROC, label = 'Rate of Climb')
plt.plot(WS,twVlof, label = 'Vlof')
plt.plot(WS,twCruise, label = 'Cruise')
plt.plot(WS,twCeiling, label = 'Ceiling')
plt.axvline(x=wsfromsizing)
plt.title(' Graph 1 \n HP/Weight ratio')
plt.legend()

# ax = plt.gca()
# ax.set_xticklabels([])

###NORMAlization
norm_twTurn = gagFerrar((grossWeight*twTurn*1.688*cruiseSpeed/(propEff*550)))
test=(grossWeight*twTurn*1.688*cruiseSpeed/(propEff*550))
norm_twROC = gagFerrar((grossWeight*twROC*1.688*V_ROC/(propEff*550)))
norm_twVlof = gagFerrar((grossWeight*twVlof*1.688*vLof/(propEff*550)))
norm_twCruise = gagFerrar((grossWeight*twCruise*1.688*cruiseSpeed/(propEff*550)))
norm_twCeiling = gagFerrar((grossWeight*twCeiling*1.688*cruiseSpeed/(propEff*550)))

plt.subplot(122)

plt.plot(WS,norm_twTurn, label = 'Rate of Turn')
plt.plot(WS,norm_twROC, label = 'Rate of Climb')
plt.plot(WS,norm_twVlof, label = 'Vlof')
plt.plot(WS,norm_twCruise, label = 'Cruise')
plt.plot(WS,norm_twCeiling, label = 'Ceiling')
plt.title('Graph 2 \n Normalised BHP')
plt.legend()
plt.axvline(x=wsfromsizing)

plt.tight_layout()
if __name__ == '__main__':
    plt.show()

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

# print(find_nearest(ws, plotWS))
plotWS = read_from_db('WS')
myidx = find_nearest(WS, plotWS)

def point():
    cruiseidx = (norm_twCruise[myidx])
    takeoffidx = norm_twVlof[myidx]
    climbidx = norm_twROC[myidx]
    turnidx = norm_twTurn[myidx]
    ceilingidx = norm_twCeiling[myidx]
    # print([cruiseidx,takeoffidx,climbidx,turnidx,ceilingidx])
    # print (cruiseidx,"cruiseidx")
    x = np.array([cruiseidx,takeoffidx,climbidx,turnidx,ceilingidx])
    return x[np.argmax(x)]

finalBHP= point()
write_to_db('finalBHP',finalBHP)
print ( finalBHP,"The Final normalised BHP")



# now switch back to figure 1 and make some changes

# plt.close()
```