Project1BackendCalcs.py

```python


'''
AEEM4063 Project 1 Backend Calculations

Computes the F, mdot, fuel flow f, and efficiencies for a real or ideal cycle HBPR Turbofan

Authors: Matt Boller, Pierce Elliott

University of Cincinnati FS23

'''


# Begin imports
import numpy as np

# Begin computation class

class TFComputation:

    def __init__(self):
        self.M = 0.85 # Mach number
        self.h = 11e3 # Altitude, m
        self.W_TO = 370000 # Max takeoff weight, lbf
        self.W = 0.8*self.W_TO # Cruise weight, lbf
        self.BPR = 10 # bypass ratio
        self.y_c = 1.4 # Gamma cold section
        self.y_h = 1.333 # Gamma hot section (post combustion)
        self.T_04 = 1560 # Turbine inlet temperature, K
        self.Q = 43100 # Enthalpy of formation for fuel, kJ/kg
        '''self.pi_f = 1.5 # Fan pressure ratio
        self.pi_c = 36/self.pi_f # Compressor pressure ratio
        self.pi_b = 1#0.96 # Pressure loss across the combustor
        self.n_i = 1#0.98 # Inlet efficiency
        self.n_inf_f = 1#0.89 # Fan polytropic efficiency
        self.n_inf_c = 1#0.90 # Compressor polytropic efficiency
        self.n_b = 1#0.99 # Combustor isentropic efficiency
        self.n_inf_t = 1#0.90 # Turbine polytropic efficiency
        self.n_m = 1#0.99 # Mechanical efficiency
        self.n_j = 1#0.99 # Nozzle efficiency'''
        self.pa = 0.22701e5 # Ambient pressure - ISA table, Pa
        # self.Ta = 216.8 # Ambient temperature - ISA table, K
        self.Ta = 216.92
        self.S_W = 285 # Wing area, m**2
        self.R = 287.1 # Ideal gas constant for air, J/kg*K
        self.cpa = 1.005 # Specific heat cold section, kJ/kg*K
        # self.cpa = 1.0048
        self.cpg = 1.148 # Specific heat hot section, kJ/kg*K

        # Lift and drag calculations
        L = self.W*4.44822162 # lbf to N
        q = self.y_c/2 * self.pa*self.M**2
        CL = L/(q*self.S_W)

        CD = 0.056*CL**2 - 0.004*CL + 0.014
        D = CD*q*self.S_W

        self.T_r = D # Thrust required
        # self.T_r = 150

        return

    def fullCycleCalc(self,B=10,pi_f=1.5,pi_c=36,isentropic='T'):

        if isentropic == 'T':
            self.pi_b = 1 # Pressure loss across the combustor
            self.n_i = 1 # Inlet efficiency
            self.n_inf_f = 1 # Fan polytropic efficiency
            self.n_inf_c = 1 # Compressor polytropic efficiency
            self.n_b = 1 # Combustor isentropic efficiency
            self.n_inf_t = 1 # Turbine polytropic efficiency
            self.n_m = 1 # Mechanical efficiency
            self.n_j = 1  # Nozzle efficiency
            self.usefuel = 0
        elif isentropic == 'F':
            self.pi_b = 0.96 # Pressure loss across the combustor
            self.n_i = 0.98 # Inlet efficiency
            self.n_inf_f = 0.89 # Fan polytropic efficiency
            self.n_inf_c = 0.90 # Compressor polytropic efficiency
            self.n_b = 0.99 # Combustor isentropic efficiency
            self.n_inf_t = 0.90 # Turbine polytropic efficiency
            self.n_m = 0.99 # Mechanical efficiency
            self.n_j = 0.99 # Nozzle efficiency
            self.usefuel = 1
        else:
            print('Error')

        self.BPR = B
        self.pi_f = pi_f
        self.pi_c = pi_c
```

```python

        # Steps through each stage of the engine
        self.intakeCalc()
        self.fanCalc()
        self.compressCalc()
        self.combustorCalc()
        self.turbCalc()
        self.nozzleCalc()
        self.mdotCalc()
        self.diaCalc()
        self.effCalc()

        # Propogates thrust, overall mdot, and fan diameter
        F = self.T_r
        mdot = self.mdot
        dia = self.D

        # Propogates fuel-air ratio and TSFC
        f = self.f
        TSFC = self.TSFC

        # Propogates efficiencies
        thermoEff = self.n_e
        propEff = self.n_p
        overEff = self.n_o

        # Calculates temperature ratios
        self.tau_f = self.T_02_5/self.T_02
        self.tau_cH = self.T_03/self.T_02_5
        self.tau_tH = self.T_04_5/self.T_04
        self.tau_tL = self.T_05/self.T_04_5

        return [mdot, dia, (F/mdot), TSFC, f, thermoEff, propEff, overEff], [self.tau_f,self.tau_cH,self.tau_tH,self.tau_tL], [self.T_02,self.T_02_5,self.T_03,
self.T_04, self.T_04_5, self.T_05, self.P_02, self.P_02_5, self.P_03, self.P_04,self.P_04_5,self.P_05], [self.M9,self.M19,self.C0,self.C9,self.C19]

    def intakeCalc(self):
        y = self.y_c
        M = self.M
        ni = self.n_i

        # Calculate temp and pressure after intake
        self.T_02 = self.Ta*(1+ (y-1)/2 * M**2)
        self.P_02 = self.pa*(1 + ni*(y-1)/2*M**2)**(y/(y-1))
        return True

    def fanCalc(self):
        y = self.y_c
        nf = self.n_inf_f

        # Calculate temp and pressure after fan
        self.T_02_5 = self.T_02*(self.pi_f)**((y-1)/(nf*y))
        self.P_02_5 = self.P_02*self.pi_f
        return True

    def compressCalc(self):
        nc = self.n_inf_c
        y = self.y_c

        # Calculate temp and pressure after compressor
        self.T_03 = self.T_02_5*(self.pi_c)**((y-1)/(nc*y))
        self.P_03 = self.P_02_5*self.pi_c
        return True

    def combustorCalc(self):
        # Calculate pressure after combustor and fuel flow
        self.P_04 = self.P_03*self.pi_b
        self.f = (self.cpg*self.T_04 - self.cpa*self.T_03)/(self.n_b*(self.Q - self.cpg*self.T_04))
        return True

    def turbCalc(self):
        nm = self.n_m
        nt = self.n_inf_t
        y = self.y_h
        cpa = self.cpa
        cpg = self.cpg
        B = self.BPR

        # Calculate temp after HPT
        if self.usefuel == 0:
            delta_T_HPT = cpa/(nm*cpg) * (self.T_03 - self.T_02_5)
        else:
            delta_T_HPT = cpa/((1+self.f)*nm*cpg) * (self.T_03 - self.T_02_5)
        self.T_04_5 = self.T_04 - delta_T_HPT

        # Calculate temp after LPT
        if self.usefuel == 0:
            delta_T_LPT = (B+1)*cpa/(nm*cpg) * (self.T_02_5 - self.T_02)
        else:
            delta_T_LPT = (B+1)*cpa/((1+self.f)*nm*cpg) * (self.T_02_5 - self.T_02)
        self.T_05 = self.T_04_5 - delta_T_LPT

        # Pressure after HPT
        self.P_04_5 = self.P_04/(self.T_04/self.T_04_5)**(y/(nt*(y-1)))

```

```python
        # Pressure after LPT
        self.P_05 = self.P_04_5/(self.T_04_5/self.T_05)**(y/(nt*(y-1)))

        return True

    def nozzleCalc(self):
        yc = self.y_c
        yh = self.y_h
        nj = self.n_j
        pa = self.pa
        R = self.R

        # Fan nozzle - perfectly expanded assumption
        M19 = np.sqrt((1/(1-nj*(-(pa/self.P_02_5)**((yc-1)/yc)+1))-1)*2/(yc-1))
        T19 = self.T_02_5/(1 + (yc-1)/2 * M19**2)
        self.C19 = M19*np.sqrt(yc*R*T19)
        self.M19 = M19

        # Core nozzle - perfectly expanded assumption
        M9 = np.sqrt((1/(1-nj*(-(pa/self.P_05)**((yh-1)/yh) + 1))-1)*2/(yh-1))
        T9 = self.T_05/(1 + (yh-1)/2 * M9**2)
        self.C9 = M9*np.sqrt(yh*R*T9)
        self.M9 = M9

        return True

    def mdotCalc(self):
        # Calculates required mdot, mdot_h, mdot_c, mdot_g, mdot_f for given flight conditions
        B = self.BPR
        F = self.T_r
        V = self.M*np.sqrt(self.y_c*self.R*self.Ta)

        if self.usefuel == 1:
            self.mdot = F/(B/(B+1)*self.C19 + (1+self.f)/(B+1)*self.C9 - V)
        else:
            self.mdot = F/(B/(B+1)*self.C19 + (1)/(B+1)*self.C9 - V)
        self.mdot_h = self.mdot/(B+1)
        self.mdot_c = self.mdot*B/(B+1)
        self.mdot_f = self.mdot_h*self.f*3600
        self.mdot_g = self.mdot_h + self.mdot_f/3600

        return True

    def diaCalc(self):
        pa = self.pa
        Ta = self.Ta

        rho = pa/(self.R*Ta)
        V = self.M*np.sqrt(self.y_c*self.R*Ta)

        A = self.mdot/(V*rho)
        self.D = 2*np.sqrt(A/np.pi)
        return True

    def effCalc(self):
        # Calculate thermal, propulsive, and overall efficiencies
        V = self.M*np.sqrt(self.y_c*self.R*self.Ta)

        if self.usefuel == 1:
            self.n_p = self.T_r*V/(0.5*(self.mdot_g*self.C9**2+self.mdot_c*self.C19**2-self.mdot*V**2))
            self.n_e = 0.5*(self.mdot_g*self.C9**2+self.mdot_c*self.C19**2-self.mdot*V**2)/(self.mdot_f/3600*self.Q*1000)
        else:
            self.n_p = self.T_r*V/(0.5*(self.mdot_h*self.C9**2+self.mdot_c*self.C19**2-self.mdot*V**2))
            self.n_e = 0.5*(self.mdot_h*self.C9**2+self.mdot_c*self.C19**2-self.mdot*V**2)/(self.mdot_f/3600*self.Q*1000)

        self.C0 = self.M*((self.y_c*self.R*self.Ta)**0.5)

        self.n_o = self.n_e*self.n_p

        self.TSFC = V/(self.n_p*self.n_e*self.Q)*1000 # g/(kN*s)

        return True
```