Perfect and Inviscid Point Design Simulation for TBCC Analysis

Ryan Wijaya

Introduction and Objective

For supersonic and hypersonic vehicles to reach multitudes about the speed of sound, investment in the analysis of turbined-based combined cycles (TBCC) may be necessary. This type of analysis consists of a turbojet and/or ramjet analytical steps depending on the flight conditions, most notably the Mach number. This is due to the fact the tradeoff between turbojets and ramjets happens along a certain Mach range[2] and turbojets have a theoretical speed limit of approximately Mach 2. Besides speed, the engines are also limited to the turbine inlet material, fuel heating ratio, and other design criteria. Such factors are limitations to the maximum range of a TBCC. In this report, the methodologies for a Mach 4 turbojet and ramjet cycle analysis will be discussed and how they play along to compute the maximum theoretical range. The reader should also be noted the main biggest assumptions are the engines are assumed to be in perfect efficiency (not adiabatic losses) and inviscid (no drag forces). These assumptions are justified as to identify the absolute limitation on design and manufacturing refinement for a turbojet-ramjet system. The objective of this work is to assess the maximum possible range for a combined turbojet-ramjet engine in a perfect and inviscid environment given certain but justified design constants.

Before proceeding to the results, a discussion on the computing approaches is necessary. For analysis coupling between engine analysis, atmospheric data, and the kinematic trajectory, OpenMDAO[1] library is utilized. Although OpenMDAO is designed for multidisciplinary design optimization in mind, a point design simulation is only pursued in this report, but optimization may be explored by the author in the future. The Python script is organized into 3 main portions: engine analysis, 2D trajectory equations of motion, and Euler integration. The engine analysis is responsible for calculating turbojet and/or ramjet at various cruise conditions and includes calculations for an afterburner. The afterburner is modeled as a turbojet where its exit flow conditions serve as the ramjet (afterburner) inlet condition. The total thrust is the combined turbojet and ramjet thrust. The 2D trajectory equations of motion obtain the calculated thrust and current atmospheric conditions to calculate the next iterative change in horizontal and vertical range components via simple kinematic equations. Every script component interacts into a main function that time steps through the calculations using a simple 1st order Euler Integration.

The assumptions, such as design constants and propulsion equations, are influential to the analytical outcome. Because design aspects for a TBCC fall within certain values, some design variables have been chosen to stay constant as they are dependent on material and type of application. These designs are fuel heating ratio, maximum turbine inlet temperature, specific heat at constant pressure, takeoff angle of attack, pressure ratios across compress and burner, and the inlet diameter. The values are tabulated in Table 1.

Since the simulation assumes a perfect and inviscid environment, the equations from Hill and Peterson[5] for isentropic and adiabatic flows with not losses are the main analytical equations used in this analysis. For turbojet analysis, equations 5.35-5.45 are used in succession with each other since the conditions from one portion of the cycle is fed into the other equations to compute the next stage condition. Meanwhile for ramjet analysis, equations 5.22, 5.24, 5.26, 5.27, 5.32, 5.33, and 5.44 are used to compute the ramjet stage conditions and are also used to

supplement calculations for turbojets. Equation 5.24 is most notable since the computation of the exit flow velocity, necessary to compute thrust, could be easily computed from inlet velocity, atmospheric stagnation temperature and material temperature. To compute the inlet mass flow rate, and the efficiencies, equations listed in the appendix were utilized. Many could be derived from the content in Hill and Peterson, but will be restated for convenience.

Table 1: Constant Design Variables

Variable	Value	Unit
QR	42,800.0	kJ/kg
Tmax	2,500.0	K
Cp	1,005.0	J/Kg*K
Alpha	75.0	Deg.
Prc	30.0	~
Prb	12.0	~
Dinlet	1.0	m

The fuel heat ratio, QR, value is the average heat ratio for the chosen fuel in mind, Jet-A. The Cp is staying constant since calorically perfect gas is assumed. Since the environment is “perfect”, there will be not adiabatic or isentropic losses, so such efficiencies will be equal to 1.0. Because this is a pure engine analysis, considerations for lift and drag forces are removed. The main “lift” force is the vertical thrust component. Drag can be assumed negligible anyway since inviscid flow is assumed and the engine is assumed to be a “hollow” tube where the flow passes through.

To start the simulation, values for the starting altitude, mass, and takeoff Mach number were decided. The simulation assumed the engine took off at sea-level. The starting gross mass is the summation of the fuel mass and engine mass and one of the exit criteria in the script is if the total mass is equal to the engine mass, meaning all fuel is burned. The engine is expected to startup and launch at a typical takeoff Mach number of 0.2 for jet-powered aircraft.

Table 2: Starting Conditions

Variable	Value	Unit
alt	0.0	m
Gross Mass	3,600.0	kg
Fuel Mass	2,000.0	kg
Engine Mass	1,600.0	kg
Takeoff Mach	0.20	~

Results and Discussion

The flight path in Figure 1 shows 6 distinct phases in the TBCC flight path analyzed. From the given constant designs and starting conditions from Tables 1 and 2, the maximum range achieved is 10,222.95 km, but this can only be achieved with a glide descent after burnout. For powered flight, the range achieved is about 6,500 km. Another distinct characteristic of the graph is the nearly vertical ascent. Even though the angle of attack at take-off was 75 degrees, the engine traveled nearly vertical because of the larger vertical thrust component. Since a level

Mach 4 flight is of interest, the script assumes the engine has a way to control its direction and maintain level flight at a given altitude and flight speed. The descent is driven mainly by the final momentum and remaining weight of the engine, resulting in the inclined approached back towards sea-level. Table 3 shows the values from the 6 distinct phases.

Table 3: Flight Path Analytical Results

Step	Altitude, Range (km/km)	Fuel Left (kg)	Total Weight (kg)	TSFC (kg/s/kN)
Takeoff	0, 0	2000	3600	9.12e-05
TJ Subsonic	6, 40	1700	3300	5.12e-05
Afterburners	12, 66	1000	2600	5.08e-05
RJ Accel.	18,76	780	2580	4.78e-05
Mach 4 cruise	20, 100	540	2140	6.78e-05
Burnout	20, 6,500	0	1600	7.838e-05

As shown in Figure 2, the overall efficiency of the turbojet decreases as the efficiency for the ramjet increases around Mach 2. To take advantage of this, the turbojet and ramjet are both switched on to overcome the high Mach number barrier for turbojets to reach Mach 4 cruise. The reason why the efficiencies for the ramjet are abnormally low at the beginning and began to spike near Mach 4 is because the program is modeling the afterburner performance before switching to a pure ramjet. The afterburner is intaking the exhaust flow from the turbojets that already have been heated up, resulting in much lower efficiencies.

Shown in Figures 3 and 4, the thrust specific fuel consumption (TSFC) for turbojets and ramjets are initially large at lower Mach numbers in their operational regimes, but generally decrease with higher Mach, possibly due to the need of less fuel to reach high thrusts from given design variables.

Conclusion

The analysis is deemed to be successful and has given reasonable results per assumptions. The engine has proven to be capable of traveling about 10,223 km, a range typical of some airliners, in a manner of a few minutes. The range is influenced by the starting conditions and the assumed constant design variables. The assumption for a perfect, inviscid system computed an overly optimistic result to the problem due to the negligence of viscous forces and loses. Since the environment is assumed to be perfect and inviscid, the isentropic and adiabatic equations and assumptions from Hill and Peterson[5] have been used for the entirety of the analysis and the calculations can only be considered in such conditions. A future investigation would be to modify the script and approach to enable multidisciplinary optimization and analysis to account for other factors such as engine integration and sizing.

Figures

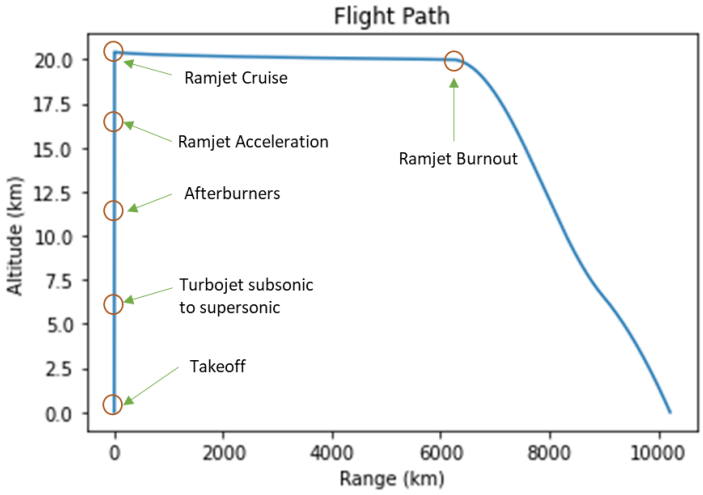

Figure 1. TBCC Flight Path with Distinct Analysis Points

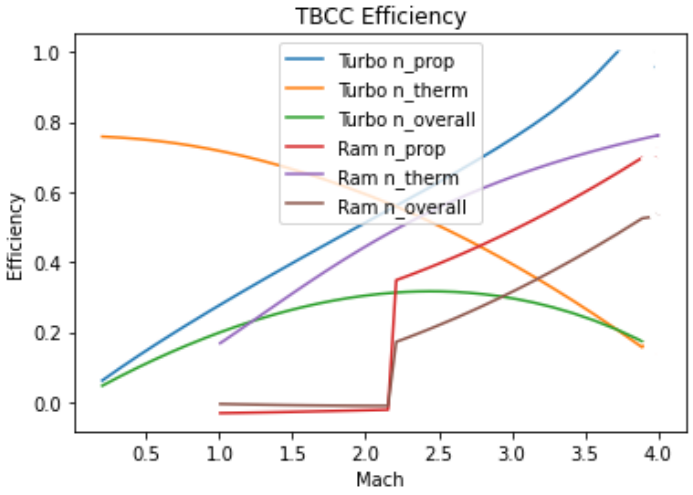

Figure 2. Thermodynamic, Propulsive, and Overall Efficiencies for Turbojet and Ramjet

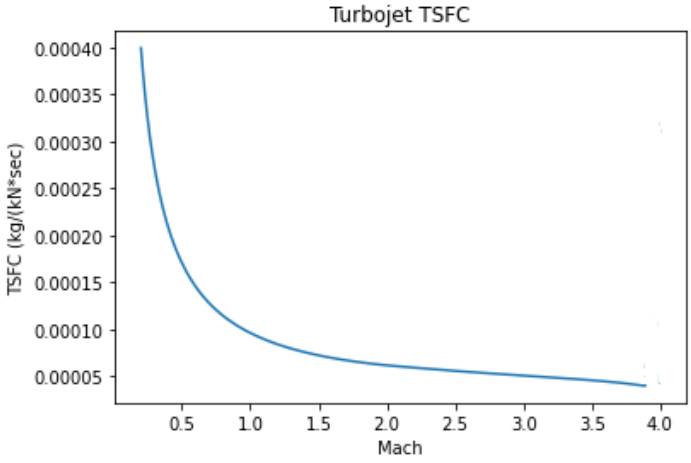

Figure 3. TSFC for a Turbojet

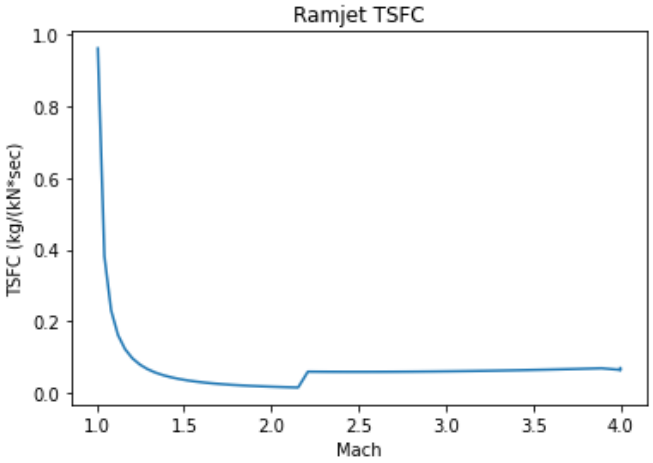

Figure 4. TSFC for a Ramjet

References

1. Gray, Hwang, Martins, Moore, Naylor. “OpenMDAO: an open-source framework for multidisciplinary design, analysis, and optimization”.

2. Wuters. “Turbojet-Ramjet Propulsion System for All-body Hypersonic Aircraft”

3. Clough. “Modeling and Optimization of Turbin-Based Combined Cycle Engine Performance”

4. Hill, and Peterson. “Mechanics and Thermodynamics of Propulsion, 2nd Edition”.

Appendix

1. $n_{prop} = (gamma - 1) * M^2 * \frac{\frac{Ve}{V_1} - 1}{\frac{T_4}{T_1} - \frac{T_3}{T_1}}$
2. $n_{overall} = n_{prop} n_{therm}$
3. $mdot = v1 * pi * (\frac{di}{2})^2$

Where,

Gamma: Specific heat ratio M: Freestream Mach number Ve: Exit Flow Velocity Mdot: mass flow rate V1: inlet flow velocity T1: Inlet temperature	n_overall: overall efficiency n_therm: thermodynamic efficiency n_prop: propulsive efficiency di: inlet diameter T4: Max turbine inlet temperature T3: Combustor exhaust temperature

Running the code:

The script consists of 4 main python files:

1. Main.py: The main iterative loop for analysis
2. Turbojet.py: Functions to calculate an ideal turbojet
3. Ramjet.py: Functions to calculate an ideal ramjet
4. Atmospherec.py: Functions to obtain atmospheric conditions

The code is written in Python 3.5.7 and uses the OpenMDAO version 3.16.0, numpy version, and written in the Spyder editor version. The best way to install the libraries necessary is through the use of pip, and instructions for those will be given below, along with for the libraries:

Pip: https://pip.pypa.io/en/stable/installation/

Spyder: https://docs.spyder-ide.org/current/installation.html

Numpy: 'pip install numpy'

OpenMDAO: 'pip install OpenMDAO[all]'

The use of the Conda environment is also preferred by the other, though the python code should work without it, but its use is for good practice:

https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html

-* coding: utf-8 -*-

"""

Created on Tue Apr 12 21:46:34 2022

@author: ryant

"""

import math
import Ramjet
import Turbojet
from atmosphere import atmos

import numpy as np
import openmdao.api as om

class JetAnalysisComp(om.ExplicitComponent):
    def initialize(self):
        self.options.declare('num_nodes', types=int)

    def setup(self):
        # inlet conditions
        self.add_input(name='ua', val=0.01, units='m/s', desc='Inlet Velocity')
        self.add_input(name='a', val=340.0, units='m/s', desc='Atmospheric speed of sound')
        self.add_input(name='pa', val=101300, units='N/m**2', desc='Atmospheric Pressure')
        self.add_input(name='Ta', val=288.5, units='K', desc='Atmospheric Temperature')
        self.add_input(name='rho', val=1.225, units='kg/m**3', desc='Atmospheric Density')

        # parameters of interest
        self.add_input(name='Prc', val=5.0, desc='Compressor Pressure Ratio')
        self.add_input(name='Prb', val=1.0, desc='Burner Pressure Ratio')
        self.add_input(name='di', val=0.1, units='m', desc='Inlet Diameter')
        self.add_input(name='Fuel', val=0.0, units='kg', desc='Amount of Fuel')

        # constant relative to problem

```
        self.add_input(name='QR', val=42800000.0, units='J/kg', desc='Fuel Heating Ratio')
        self.add_input(name='Tmax', val=2500.0, units='K', desc='Maximum Temperature Limit')
        self.add_input(name='cp', val=1005.0, units='J/(kg*K)', desc='Air Specific Heat at Constant Pressure')

        # output
        self.add_output(name='TSFC', val=0.0, units='kg/(N*s)', desc='Thrust Specific Fuel Consumption')
        self.add_output(name='ue', val=0.0, units='m/s', desc='Outlet velocity')
        self.add_output(name='Thrust', val=0.0, units='N', desc='Thrust')
        self.add_output(name='Fuel_Cons', val=0.0, units='kg', desc='Amount of Fuel Consumed at Instant')
        self.add_output(name='Mach', val=0.0, desc='Inlet/Flight Mach Number')

    def compute(self, inputs, outputs):
        ua = inputs['ua']
        a = inputs['a']
        Pa = inputs['pa']
        Ta = inputs['Ta']
        rho = inputs['rho']
        Prc = inputs['Prc']
        QR = inputs['QR']
        To4 = inputs['Tmax']
        cp = inputs['cp']
        Prb = inputs['Prb']
        di = inputs['di']
        Fuel = inputs['Fuel']

        mach = ua/a

        tsfc = 0.0
```

```
    thrust = 0.0
    ue = 0.0
    f= 0.0
    madot = 0.0

    if mach < 1.0:
        tj_madot = Turbojet.mass_flow_rate(ua, di, rho)
        print(f"inlet mass inflow: {tj_madot} kg/s")
        To2 = Turbojet.Compr_Inlet_Stag_Temp(mach, Ta)
        To2_Ta = To2/Ta
        Po2 = Turbojet.Compr_Inlet_Stag_Pres(To2_Ta, Pa)
        To3 = Turbojet.Comp_Outlet_Stag_Temp(Prc, To2)
        To4_To3 = To4/To3
        tf_f = Turbojet.fuel_air_ratio(To4_To3, QR, cp, To3)
        To5 = Turbojet.Turb_Outlet_Stag_Temp(To3, To2, To4)
        To5_To4 = To5/To4
        Po3 = Turbojet.Comp_Outlet_Stag_Pres(Prc, Po2)
        Po4 = Turbojet.Turb_Inlet_Pres(Prb, Po3)
        Po5 = Turbojet.Turb_Outlet_Stag_Pres(To5_To4, Po4)
        Pa_Po5 = Pa/Po5
        tj_ue = Turbojet.Exhaust_Velocity(To5, Pa_Po5)
        tj_Thrust = Turbojet.Thrust(tj_madot, tf_f, tj_ue, ua)
        tj_tsfc = Turbojet.TSFC(tf_f, tj_Thrust)

#
================================================================================
========
#           print("Turbojet")
#           print(f"ua: {ua}")
#           print(f"Toa: {Ta}  Poa: {Pa}")
```

```
#           print(f"To2: {To2}  Po2: {Po2}")
#           print(f"To2/Ta: {To2_Ta}")
#           print(f"To3: {To3}  Po3: {Po3}")
#           print(f"To4: {To4}  Po4: {Po4}")
#           print(f"To4/To3: {To4_To3}")
#           print(f"To5: {To5}  Po5: {Po5}")
#           print(f"To5/To4: {To5_To4}")
#           print(f"tsfc {tj_tsfc}")
#
================================================================================


            f = tf_f
            tsfc = tj_tsfc
            thrust = tj_Thrust
            ue = tj_ue
            madot = tj_madot

            # assume turbojets work best in the subsonic regime until sonic
            # use turbojet and ramjet to dash towards high mach until
            # though its expected the mach number to switch from tj to rj fully is ~2.2
        elif mach >= 1.0 and mach < 2.2:
            tj_madot = Turbojet.mass_flow_rate(ua, di, rho)
            print(f"inlet mass inflow: {tj_madot} kg/s")
            To2 = Turbojet.Compr_Inlet_Stag_Temp(mach, Ta)
            To2_Ta = To2/Ta
            Po2 = Turbojet.Compr_Inlet_Stag_Pres(To2_Ta, Pa)
            To3 = Turbojet.Comp_Outlet_Stag_Temp(Prc, To2)
            To4_To3 = To4/To3
            tf_f = Turbojet.fuel_air_ratio(To4_To3, QR, cp, To3)
```

```
To5 = Turbojet.Turb_Outlet_Stag_Temp(To3, To2, To4)
To5_To4 = To5/To4
Po3 = Turbojet.Comp_Outlet_Stag_Pres(Prc, Po2)
Po4 = Turbojet.Turb_Inlet_Pres(Prb, Po3)
Po5 = Turbojet.Turb_Outlet_Stag_Pres(To5_To4, Po4)
Pa_Po5 = Pa/Po5
tj_ue = Turbojet.Exhaust_Velocity(To5, Pa_Po5)
tj_Thrust = Turbojet.Thrust(tj_madot, tf_f, tj_ue, ua)
tj_tsfc = Turbojet.TSFC(tf_f, tj_Thrust)
# calculation for pure ramjet so the code knows when to only use
# ramjet or use tj+rj
f = tf_f
tsfc = tj_tsfc
thrust = tj_Thrust
ue = tj_ue
madot = tj_madot
# if turbojet with afterburner (ramjet), where the exhaust gas from
# the turbojet stage is reheated
# compare with propulsive efficiency instead

To4_Toa = To4/To5
Toa = To5
rj_f = Ramjet.Fuel_Air_Ratio(To4_Toa, QR, cp, Toa)
T_madot = Ramjet.Thrust_madot_Ratio(mach, Toa, To4_Toa, rj_f)
rj_tsfc = Ramjet.TSFC(rj_f, T_madot)
rj_ue = Ramjet.Exhaust_Velocity(To4, Toa, ua)
rj_thrust = Ramjet.Thrust(madot, rj_f, rj_ue, ua)
rj_madot = (1/T_madot) * rj_thrust

print(f"T_madot {T_madot}")
```

```
        print(f"rj f {rj_f}")
        print(f"rj tsfc {rj_tsfc}")

        tsfc = tsfc + rj_tsfc
        thrust = thrust + rj_thrust
        ue = ue + rj_ue
        f = f + rj_f
        madot = madot + rj_madot
    elif mach >= 2.2:
        To4_Toa = To4/Ta
        Toa = Ta
        rj_f = Ramjet.Fuel_Air_Ratio(To4_Toa, QR, cp, Toa)
        T_madot = Ramjet.Thrust_madot_Ratio(mach, Toa, To4_Toa, rj_f)
        rj_tsfc = Ramjet.TSFC(rj_f, T_madot)
        rj_ue = Ramjet.Exhaust_Velocity(To4, Toa, ua)
        rj_madot = Turbojet.mass_flow_rate(ua, di, rho)
        rj_thrust = Ramjet.Thrust(rj_madot, rj_f, rj_ue, ua)
        madot = rj_madot
        print(f"inlet mass inflow: {madot} kg/s")
        f = rj_f
        tsfc = rj_tsfc
        thrust = rj_thrust
        ue = rj_ue

    fuel_consumption = (f * madot)
    if Fuel <= 0.0:
        thrust = 0.0
        fuel_consumption = 0.0
```

```
        print(f"Mach: {mach}, \t Thrust: {thrust/1000} kN, \t ,TSFC: {tsfc}, \t ,ua: {ua} m/s \t ue: {ue} m/s \t ,f: {f}")


        outputs['TSFC'] = tsfc
        outputs['ue'] = ue
        outputs['Thrust']= thrust
        outputs['Fuel_Cons'] = fuel_consumption
        outputs['Mach'] = mach



class TrajectoryEOM2D(om.ExplicitComponent):
    """
    Computes the position and velocity equations of motion using a 2D flight path.
    """
    def initialize(self):
        self.options.declare('num_nodes', types=int)


    def setup(self):
        self.add_input(name='m', val=1.0, units='kg',
                       desc='total Mass')
        self.add_input(name='v', val=0.01, units='m/s',
                       desc='Velocity magnitude')
        self.add_input(name='T', val=0.0, units='N',
                       desc='thrust')
        self.add_input(name='alpha', val=0.0, units='rad',
                       desc='angle of attack')
        self.add_input(name='dt', val=0.2, units='s',
                       desc='Time Increments')


        self.add_output(name='v_dot', val=0.0, units='m/s**2',
```

```
                desc='rate of change of velocity magnitude')
        self.add_output(name='hchng_dot', val=0.0, units='m/s',
                desc='rate of change of vertical velocity')
        self.add_output(name='rchng_dot', val=0.0, units='m/s',
                desc='rate of change of horizontal velocity')

    def setup_partials(self):
        self.declare_partials(['hchng_dot', 'rchng_dot'], ['T', 'm','alpha'])

    def compute(self, inputs, outputs):
        g =  9.80665 # m/s^2
        m = inputs['m']
        dt = inputs['dt']
        T = inputs['T']
        alpha = inputs['alpha']

        h_2dot = ((T * np.sin(alpha)) / m) - g
        r_2dot = (T * np.cos(alpha)) / m
        hchng_dot = h_2dot * dt
        rchng_dot = r_2dot * dt
        outputs['v_dot'] = math.pow(math.pow(h_2dot,2) + math.pow(r_2dot,2), 0.5)
        outputs['hchng_dot'] = hchng_dot
        outputs['rchng_dot'] = rchng_dot


class TbccODE(om.Group):

    def setup(self):
        self.add_subsystem(name='jetanal',
                           subsys=JetAnalysisComp(),
```

```
                promotes=['*'])

        self.add_subsystem(name='eom',
                subsys=TrajectoryEOM2D(),
                promotes=['*'])

        self.connect('Thrust', 'T')



def eval_range(design_init, prob, complex_step=False):
    h = 0.0
    r = 0.0
    hdot = 0.0
    rdot = 0.0
    dt = 0.1
    t = 0.0
    [_,_,_,_,a] = atmos(h)
    Mach = 0.2 # typical takeoff Mach
    ua = a * Mach
    emass = 1600  # empty engine mass (tj and ramjet)
    fuel_mass = 2000.0 # fuel mass at takeoff
    in_mission = True # only false when it crashes from lack of fuel after takeoff during glide descent


    if complex_step:
        prob.set_complex_step_mode(True)


    while in_mission:
        print("====================================")
        print(f"Time {t} sec.     Mass {fuel_mass+emass} kg")
```

```
[T, P, rho, _, a] = atmos(h)

# Set values
prob.set_val('ua', ua, units='m/s')
prob.set_val('v', ua, units='m/s')
prob.set_val('a', a, units='m/s')
prob.set_val('pa', P, units='N/m**2')
prob.set_val('Ta', T, units='K')
prob.set_val('rho', rho, units='kg/m**3')
prob.set_val('m', fuel_mass + emass, units='kg')
prob.set_val('Fuel', fuel_mass, units='kg')

prob.set_val(name='Prc', val=design_init)
prob.set_val(name='Prb', val=12.0)
prob.set_val(name='di', val=1.0, units='m')

Mach = ua/a

# Run the model
prob.run_model()

# Extract rates
v_dot = prob.get_val('v_dot')
hchng_dot = prob.get_val('hchng_dot')
rchng_dot = prob.get_val('rchng_dot')
f_cons = prob.get_val('Fuel_Cons')
Mach = prob.get_val('Mach')

h_last = h
r_last = r
```

```
    # Euler Integration
    fuel_mass = fuel_mass - (dt * f_cons)
    ua = ua + (dt * v_dot)
    hdot = hdot + hchng_dot
    rdot = rdot + rchng_dot
    h = h + (dt * hdot)
    r = r + (dt * rdot)
    t += dt

    if Mach >= 4.0:
        ua = 4.0*a

    if h >= 20000:
        prob.set_val('alpha', 1.0*(math.pi/180.0), units='rad')
        h = 20000

    print(f"Height: {h/1000.0} km\t Range: {r/1000.0} km \t fuel consumed: {f_cons*dt} kg")
    print("====================================")
    if h <= -0.01:
        in_mission = False

# Linear interpolation between last two points to get the landing point accurate.
r_final = r_last + (r - r_last) * h_last / (h_last - h)

if complex_step:
    prob.set_complex_step_mode(False)

return -r_final
```

```
def gradient_range(init, prob):
    """
    Uses complex step to compute gradient of range wrt initial angle.

    Parameters
    ----------
    init : float
        Initial conditions
    prob : <Problem>
        OpenMDAO problem that contains the equations of motion.

    Returns
    -------
    float
        Derivative of range wrt pressure ratio
    """
    step = 0.1
    dr_dgam = eval_range(init + step, prob, complex_step=False)
    return dr_dgam.imag / step


if __name__ == "__main__":
    prob = om.Problem(model=TbccODE())
    prob.setup(force_alloc_complex=False)

    ###### Constants ######
    prob.set_val('QR', 42800.0, units='kJ/kg')
    prob.set_val('Tmax', 2500.0, units='K')
    prob.set_val('cp', 1005.0, units='J/(kg*K)')
```

```
    prob.set_val('alpha', 75.0*(math.pi/180), units='rad')



    #Prc = np.linspace(4, 40, 20)
    #Prb = np.linspace(4, 40, 20)
    #Diam_inlet = np.linspace(0.05, 0.5, 20) # meters
    ####################
    to_optimize = False

    if to_optimize:
        result = minimize(eval_range, 30.0,
                          method='SLSQP',
                          jac=gradient_range,
                          args=(prob))
        print(result['x'])
    else:
        #result = eval_range(45.0, prob)
        result = eval_range(13.0, prob)

########### Atmosphere .py ################
# -*- coding: utf-8 -*-
"""
Modified on Wed Apr 13 02:10:15 2022
By Ryan Wijaya
for use in metric
"""

import math

def atmos(h):
```

"""
1976 US Standard Atmosphere
http://www.atmosculator.com/The#20Standard#20Atmosphere.html?
David Pate and Michael Patterson

Returns
    A list with the following:

Repurposed from AE 4802: Configuration Aerodynamics and Flight Performance
by Brian German

Parameters
----
h : float
    altitude in feet.

Returns
----
temp: float
    temperature in K
pres: float
    pressure in Pa
rho: float
    density in kg/m^3
mu: float
    viscosity in kg/(m-s)
a: float
    sound speed m/s assuming R=1716 ft-lb/(slugs-°R) and gamma = 1.4
"""

```
R = 1716;
gamma = 1.4;
T_SL = 518.69;      #°R
T0 = 491.6;         #°R
p_SL = 2116.2;      #lb/ft^2
rho_SL = .0023769;  #slug/ft^3
mu0 = 3.58394051e-7;    #slug/(ft-s)
# 3.7373e-7;    #slug/(ft-s)
S = 199;            #Sutherland's Constant °R
theta = 0
delta = 0
sigma = 0

h = h * 3.281

if h <= 36809:
    theta = 1 - h / 145442;
    delta = math.pow(1 - h / 145442, 5.255876);
    sigma = math.pow(1 - h / 145442, 4.255876);



# Isothermal
elif h > 36089 and h <= 65617:
    theta = 0.751865;
    delta = 0.223361 * math.exp(-(h-36089)/20806);
    sigma = 0.297076 * math.exp(-(h-36089)/20806);



# (Inversion)
elif h > 65617 and h <= 104987:
```

```
    theta = 0.682457 + h / 945374;
    delta = math.pow(0.988626 + h / 652600, -34.16320);
    sigma = math.pow(0.978261 + h / 659515, -35.16320);


# (Inversion)
elif h > 104987 and h <= 154199:
    theta = 0.482561 + h / 337634;
    delta = math.pow(0.898309 + h / 181373, -12.20114);
    sigma = math.pow(0.857003 + h / 190115, -13.20114);


# (Isothermal)
elif h > 154199 and h <= 167323:
    theta = 0.939268;
    delta = 0.00109456 * math.exp(-(h-154199)/25992);
    sigma = 0.00116533 * math.exp(-(h-154199)/25992);


elif h > 167323 and h <= 232940:
    theta = 1.434843 - h / 337634;
    delta = math.pow(0.838263 - h / 577922, 12.20114);
    sigma = math.pow(0.798990 - h / 606330, 11.20114);


elif h > 232940 and h <= 278386:
    theta = 1.237723 - h / 472687;
    delta = math.pow(0.917131 - h / 637919, 17.08160);
    sigma = math.pow(0.900194 - h / 649922, 16.08160);
```

```
    temp = theta * T_SL;
    pres = delta * p_SL;
    rho = sigma * rho_SL;
    mu = mu0 * math.pow(temp/T0,3/2) * (T0 + S) / (temp + S);
    a = math.pow(gamma * R * temp,0.5);

    # convert to metric
    temp = temp * 5/9
    pres = (pres / 20.88543)*1000
    rho = rho * 14.594/(0.3048*0.3048*0.3048)
    mu = mu * 14.594 * 3.281
    a = a / 3.281

    return [temp, pres, rho, mu, a]
```

############## Turbojet.py ####################

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 14 01:31:20 2022

@author: ryant
"""
import math

def Thrust(madot, f, ue, u):
    a = (1+f)*ue
    return madot * (a - u)

# returns To2
```

```
def Compr_Inlet_Stag_Temp(mach, Ta):
    a = 1 + (0.5*(1.4-1)*mach*mach)
    return Ta * a

# returns Po2
def Compr_Inlet_Stag_Pres(To2_Ta, Pa):
    nd = 1.0
    y = 1.4
    a = nd * (To2_Ta - 1)
    b = math.pow(1 + a, y / (y - 1))
    return Pa * b

# returns Po3
def Comp_Outlet_Stag_Pres(Prc, Po2):
    return Prc * Po2

# returns To3
def Comp_Outlet_Stag_Temp(Prc, To2):
    nc = 1.0
    y = 1.4
    a = math.pow(Prc, (y - 1) / y)
    b = (1 / nc) * (a - 1)
    return To2 * (1 + b)

def fuel_air_ratio(To4_To3, QR, cp, To3):
    a = To4_To3 - 1
    b = (QR / (cp * To3)) - To4_To3
    return a / b

def Turb_Inlet_Pres(Prb, Po3):
```

```
    """
    Parameters
    ----------
    Prb : float
        Burner Pressure Ratio (Po4/Po3).
    Po3 : float
        Stagnation Pressure Outside Burner.

    Returns
    -------
    Po4: float
        Stagnation Pressure After Burner

    """
    Po4 = Po3 * Prb
    return Po4

def Turb_Outlet_Stag_Temp(To3, To2, To4):
    """
    Parameters
    ----------
    To3 : float
        Compressor outlet stagnation temperature (K).
    To2 : float
        Compressor inlet stagnation temperature (K).
    To4 : float
        Maximum turbine material temperature (K).

    Returns
    -------
```

```
    To5 : float
        Turbine outlet temperature (K).

    """

    To5 = To4 - To3 + To2
    return To5

def Turb_Outlet_Stag_Pres(To5_To4, Po4):
    nt = 1.0
    y = 1.4
    a = y / (y - 1)
    b = 1 - ((1.0/nt)*(1 - To5_To4))

    print(b)
    if b < 0.0:
        b = 0.0
    Po5 = Po4*math.pow(b, a)
    return Po5

def Exhaust_Velocity(To5, Pa_Po5):
    y = 1.4
    nn = 1.0
    R = 287
    a = y / (y - 1)
    b = 1 - math.pow(Pa_Po5, (y - 1) / y)
    if b < 0.0:
        b = 0.0
    ue = math.pow(2 * nn * a * R * To5 * b, 0.5)
    return ue
```

```
def TSFC(f, T):
    return f/T

def mass_flow_rate(u, diam, rho):
    area = math.pi*math.pow(diam*0.5,2)
    return area * u * rho

def therm_eff():
    None

def prop_eff(mach, v6_v1, T4_T1, T3_T1):
    y = 1.4
    a = (y - 1)*math.pow(mach, 2)
    b = v6_v1 - 1
    c = (T4_T1) - (T3_T1)
    return a * (b / c)
########### Ramjet.py ######################
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 14 01:20:31 2022

Functions for an Ideal Ramjet

@author: ryant
"""
import math

def Thrust(madot, f, ue, u):
    return madot*((1+f)*ue - u)
```

```
def Isentropic_Total_Pressure(mach, pres):
    a = 1.4 / (1.4 - 1)
    b = (1.4 - 1) * 0.5
    c = 1 + (b*mach*mach)
    return pres * math.pow(c, a)

def Exhaust_Velocity(To4, Toa, ua):
    return ua * math.pow(To4/Toa,0.5)

def Fuel_Air_Ratio(To4_Toa, QR, cp, Toa):
    a = To4_Toa - 1
    b = (QR/(cp*Toa)) - To4_Toa
    return a/b

def Thrust_madot_Ratio(mach, Toa, To4_Toa, f):
    a = mach * math.pow(1.4*287*Toa, 0.5)
    b = (1+f)*math.pow(To4_Toa, 0.5)
    c = 1 + ((1.4 - 1) * 0.5*mach*mach)
    d = (b * math.pow(c, -0.5)) - 1
    return a * d

def TSFC(f, T_madot):
    return f/T_madot

def mfdot(TSFC, Thrust):
    return TSFC*Thrust

def mass_flow_rate(u, diam, rho):
    area = math.pi*math.pow(diam*0.5,2)
```

```
    return area * u * rho

def therm_eff(prc):
    return 1 - (1/prc)

def prop_eff():
    return
```