ME 357 Gas Turbine Design

Problem Set #3

Rui Xu (ruixu@stanford.edu)

1. Turbofan with a Combustion Model

(a) Complete the skeletal code

The MATLAB code of "combustor.m" and the driver code "Problem1Driver.m" is in Appendix 1 on Pages 8-11.

9

(b) Using the combustor model from Part a and plot figures

(i)

The plot is shown in Figure 1 as follows.

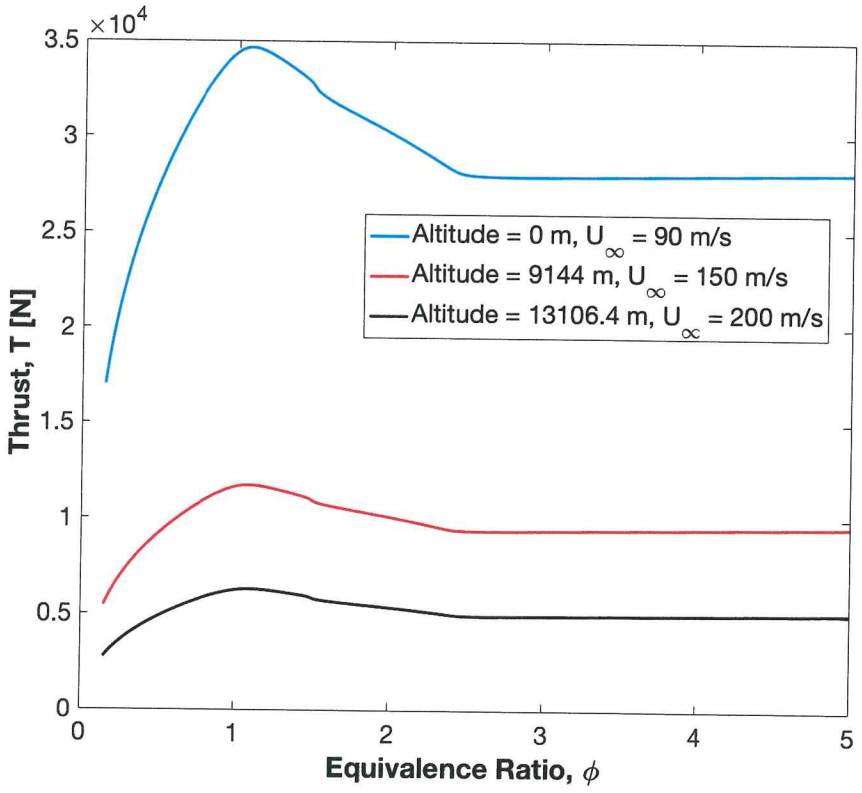

Figure 1

- At 0 ft altitude, the thrust is optimized at ϕ = 1.08;
- At 30000 ft altitude, the thrust is optimized at ϕ = 1.08;
- At 43000 ft altitude, the thrust is optimized at ϕ = 1.09.

The reason why the thrusts peak at those equivalence ratios is that the adiabatic flame temperature T_4 peaks at those equivalence ratios (1.03~1.10 equivalence ratio for typical hydrocarbon fuels). The peak of adiabatic flame temperature indicates the complete combustion.

(ii)

The plot is shown in Figure 2 as follows.

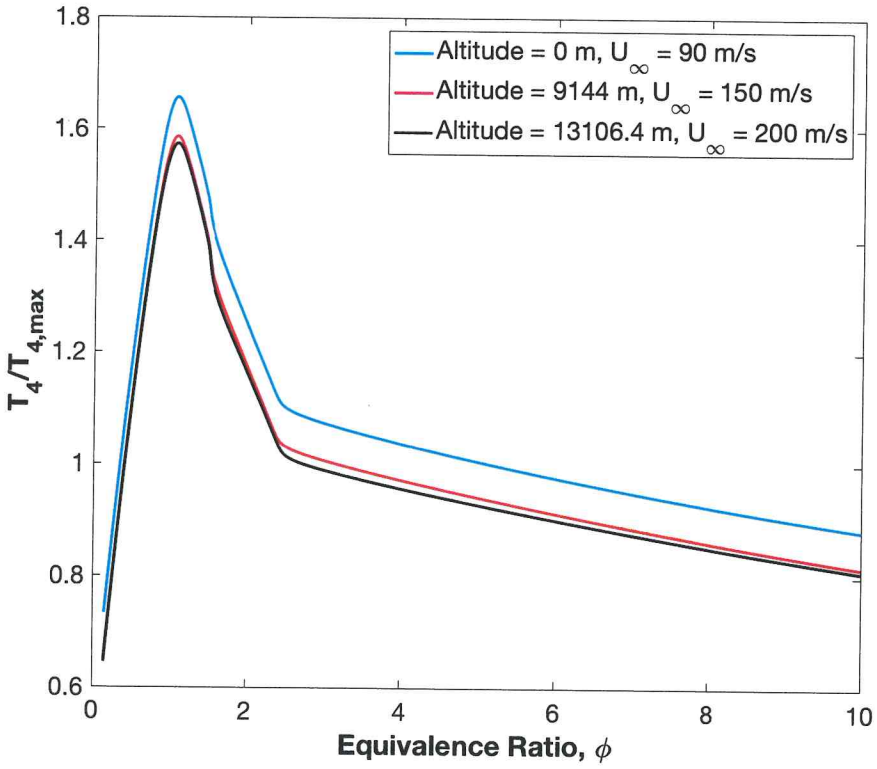

Figure 2.

- At 0 ft altitude, $T_4/T_{4,max}$ = 1 happens at ϕ = 0.35;
- At 30000 ft altitude, $T_4/T_{4,max}$ = 1 happens at ϕ = 0.41;

- At 43000 ft altitude, $T_4/T_{4,max} = 1$ happens at $\phi = 0.41$.

Based on the thrust results in Figure 1, the thrusts generated under those equivalence ratios are not optimal.

2. Cruise Conditions for the HondaJet

(a)

To calculate the available thrust:

$$T_{avail} = \dot{m}_{AC}[(1+f)U_e + \beta U_{1e} - (1+\beta)U_0]$$

and to match the required thrust:

$$T_{req} = \frac{1}{2}C_{D,0}\rho_0 {U_0}^2 + \frac{2W^2}{\rho_0 {U_0}^2 S\pi eAR}$$

from the given parameters in HW1 Problem 2(b), $\dot{m}_{AC} = 8kg/s, \beta = 2.9, \pi_F = 2, \pi_C = 24$ can be unchanged. However, the turbine inlet temperature T_{04} will change since we are using the realistic combustor model.

(b)

The code (mFuelDot.m) are written in MATLAB in Appendix 2 on Pages 13-14. The key idea is to solve the $\dot{m}_F$ (or $f = \frac{\dot{m}_F}{\dot{m}_{AC}}$) iteratively, by solving the equation: $g(f) = g(\dot{m}_F) = T_{avail} - T_{req} = 0$. Note that T_{req} is a function of W, and W is also a function of $\dot{m}_F$. So finally the solution of both $\dot{m}_F$ and W needs to be solved numerically, through combining the work in (c) and (d).

(c)

The weight:

$$W = W_{takeoff} - \dot{m}_F g t$$

If we take the derivative of the above equation, we get the ODE:

$$\frac{dW}{dt} = -\dot{m}_F g$$

$\dot{m}_F$ depends on $T_{req}, \beta, \rho_0, T_0, U_0$, altitude h, and some other parameters.

(d) Solve the ODE for aircraft weight as a function of time

The aircraft weight (kN) as a function of time (hour) is shown below in Figure 3. For the range, the aircraft weight (kN) as a function of range (km) is shown in Figure 4. From the MATLAB result, the range is about 6420 km, assuming all fuel is consumed.

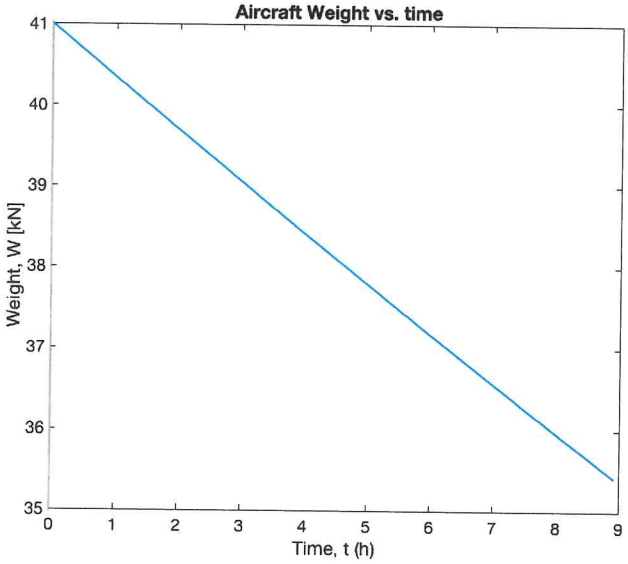

Figure 3.

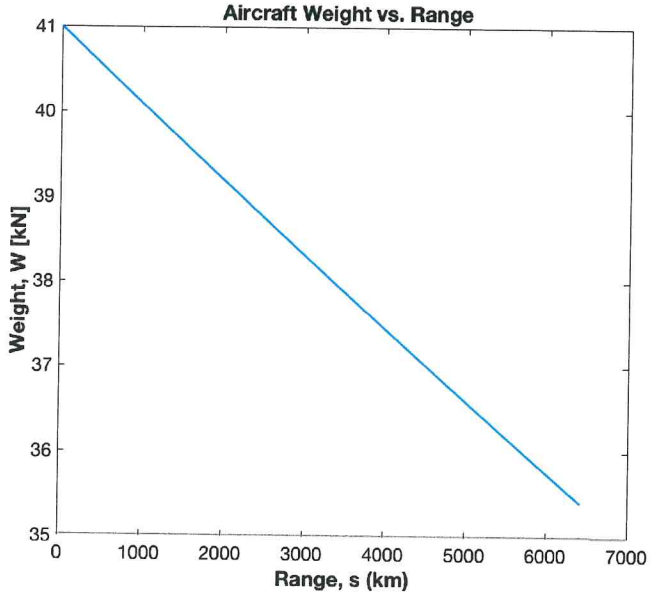

Figure 4.

(e) The range found using both the ODE and Breguet range equation

The comparison of results from solving the ODE and Breguet Range Equation is shown below in Figure 5. The range calculated by Breguet equation is about 2120 km, which is quite different from the result calculated through numerical solution of the ODE.

The reason is that, the Breguet range equation made several assumptions that should not be applicable:

- The thrust specific fuel consumption (TSFC) is a constant: $TSFC = \frac{\dot{m}_F}{T}$, but in reality it is not a constant;
- The thrust specific fuel consumption (TSFC) is derived through an empirical formula (Eqn 2.59 in course reader);
- The ratio: C_D/C_L is a constant. In reality it is not.

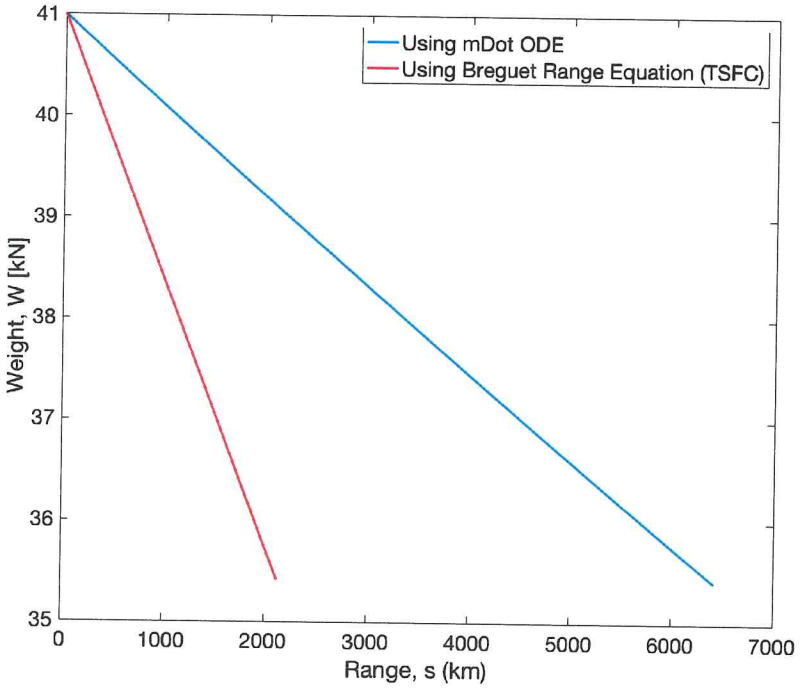

Figure 5.

The MATLAB driver code for Problem 2 is in Appendix 2.1 on Pages 11-13.

3. Nitric Oxide Formation

(a) Standard Combustor

The ratio of $\dot{m}_{NO}/\dot{m}_{NO,max}$ as a function of equivalence ratio is shown below in Figure 6. When the equivalence ratio is higher than 1.5, the mass production of NO meets the standard. At ϕ = 1.5, which is a fuel rich condition, CO and soot will be much produced, and the system will leave over some unburned hydrocarbon fuels (UHC).

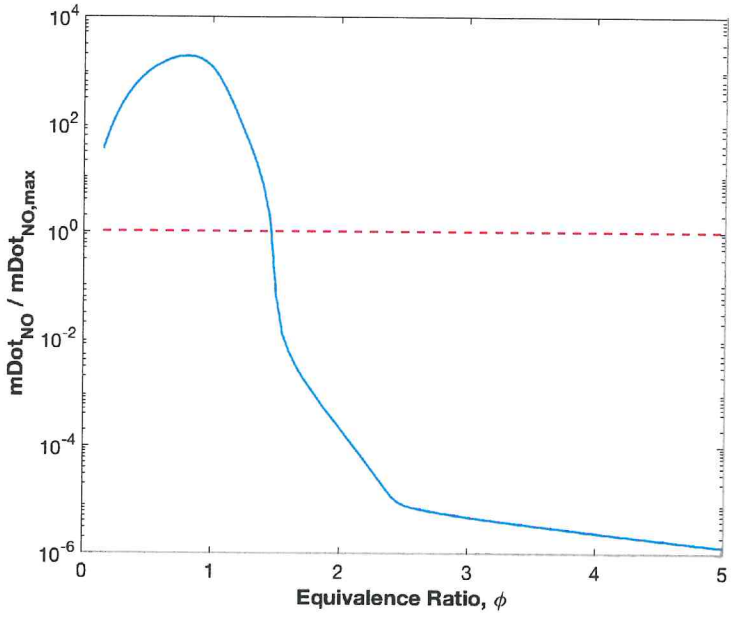

Figure 6.

(b) RQL Combustor

(i)

Plug in numbers:

$$\frac{d\dot{m}_{NO}}{dt} = k\dot{m}_{N_2}\dot{m}_O$$

$$= \frac{2.66}{kg} \times \frac{0.403kg}{s} \times \frac{3.15\times10^{-7}kg}{s}$$

$$= 3.377\times10^{-7}kg/s^2$$

So:

$$\dot{m}_{NO} = 3.377\times10^{-7}t$$

To produce maximum NO mass production rate:

$$\dot{m}_{NO,max} = 3.377\times10^{-7} t_{max}$$
$$= 3.377\times10^{-7} \frac{L_{dil}}{U_4}$$

Then,

$$L_{dil} = \frac{\dot{m}_{NO,max} \cdot U_4}{3.377\times10^{-7}} = \frac{0.04\times \frac{10^{-3}kg}{s} \times \frac{11.8m}{s}}{3.377\times \frac{10^{-7}kg}{s^2}}$$

$$= 1398\, m$$

The maximum dilution length L_{dil} is very large. In reality, the length would be in the order of O(10) cm. Therefore, it is feasible to dilute the burned gas before significant NO is produced.

(ii)

The plot is shown below in Figure 7. From the results, we can find that when $\beta_C = 1$, the combustor needs to start rich burn at a richer equivalence ratio than the standard combustor, in order to meet the maximum NO production standard. The equivalence ratio value is about ϕ = 2.95.

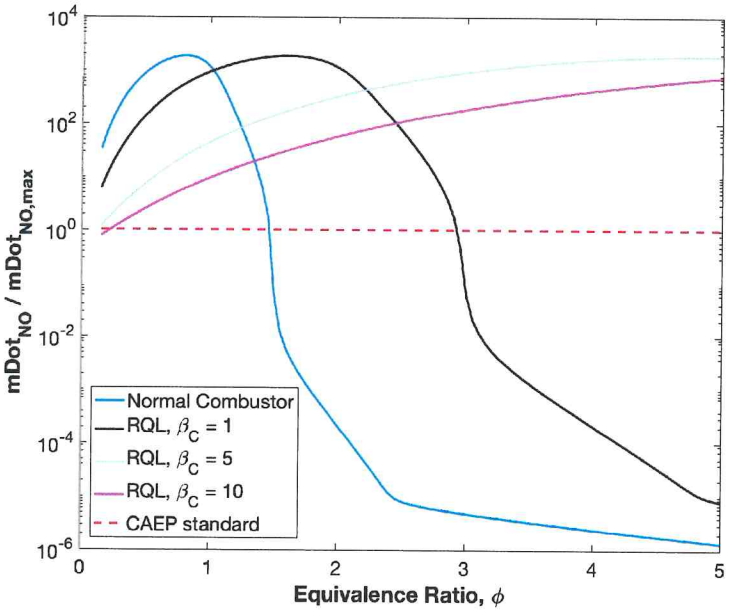

Figure 7.

However, when the reroute ratio increases to 5 or 10, the system can never meet the NO production standard. This is because in the lean burn process, the more the reroute air provided into the mixture, the more the unburned fuel would react with oxygen and increase the adiabatic flame temperature. The higher the adiabatic flame temperature, the more the NO production.

Therefore, in RQL combustor, we need to smartly find an optimal value of reroute ratio, β_C. The MATLAB code of the RQL combustor can be found in Appendix 3 on Page 16.

(c) Empirical Correlation

The plot of $\dot{m}_{NO}/\dot{m}_{NO,max}$ vs. equivalence ratio is shown below in Figure 8. As we can see, the result using the correlation (green curve in the figure) shows that our Honda Jet cannot meet the CAEP emission standards, as the ratio lies above 1 for all the equivalence ratios we've tested.

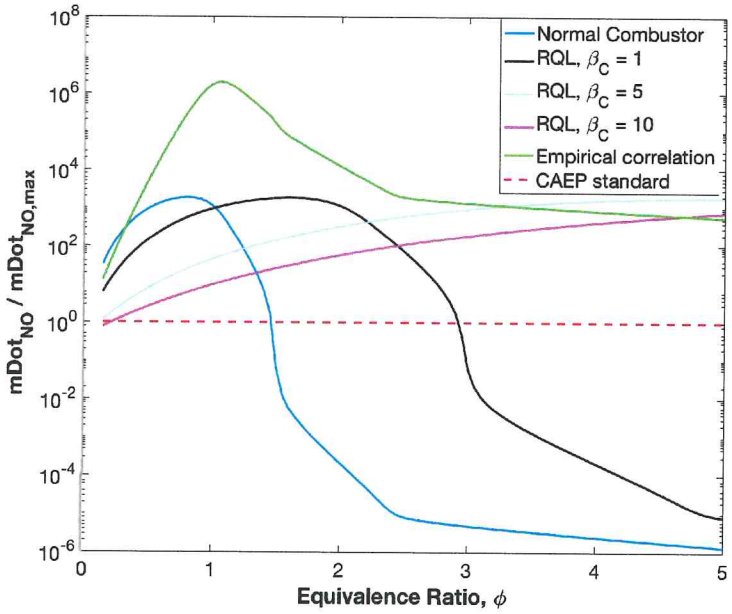

Figure 8.

The reasons why the correlation gives an discrepancies with our combustor model are the following:

- The correlation is derived for lean combustion condition, not under all equivalence ratios;

- The factor in front of T04 should not be a constant (0.008) as the condition in each state changes;
- The power of pressure p4 should not be a constant (0.5). Although this power value indicates a somewhat global reaction order, in reality, this order shouldn't be a constant.

The MATLAB driver code for Problem 3 is in Appendix 3.1 on Pages 14-15.

Appendix 1. MATLAB code for Problem 1.

A1.1. Driver code.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  ME357 Spring 2017
%  Problem Set #3
%  1. Turbofan with a Combustion Model
%  (b) Driver code
%
%  Rui Xu (ruixu@stanford.edu)
%  Uses as the driver code for Problem 1.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
clc

%define a structure containing the HondaJet's parameters
HondaJet.airframe.S = 17.3;                     %[m^2] , reference area
HondaJet.airframe.b = 12.12;                    %[m] , wing span
HondaJet.airframe.W = 41000;                    %[N] , weight (which is max takeoff weight)
HondaJet.airframe.CD0 = 0.01;                   %[] , zero-lift drag coefficient (or parasite drag
coeff.)
HondaJet.airframe.e = 0.8;                      %[] , Oswald Efficiency
HondaJet.engine.overallCompressionRatio = 24;   %[]
HondaJet.engine.turbineInletTemperature=1600;   %[K]
HondaJet.engine.mDotSeaLevel = 8.0;             %[kg/s]
HondaJet.engine.nEngines = 2;                   %[]
HondaJet.engine.bypassRatio = 2.9;              %[]
HondaJet.engine.fanCompressionRatio = 2.0;      %[]
HondaJet.engine.efficiencies.diffuser = 0.95;   %[]
HondaJet.engine.efficiencies.fan = 0.92;        %[]
HondaJet.engine.efficiencies.compressor = 0.87; %[]
HondaJet.engine.efficiencies.combustor = 1.00;  %[]
HondaJet.engine.efficiencies.turbine = 0.91;    %[]
HondaJet.engine.efficiencies.coreNozzle = 0.98; %[]
HondaJet.engine.efficiencies.fanNozzle = 0.98;  %[]


%define a structure containing the fluid

fluid.gas = importPhase('ndodecane_mech.cti');
fluid.fuel.name = 'nc12h26';
fluid.fuel.nC = 12;
fluid.fuel.nH = 26;
fluid.gamma = 1.4;
fluid.R = 287;

%Input a vector of altitudes, speeds, and equivalence ratios
altitudes = [0, 9144, 13106.4];    %[m]
UInf = [90, 150, 200];             %[m/s]
phi = 0.15:0.05:10.0;

%Create the plots for Part b
nAltitudes = length(altitudes);
nPhi = length(phi);
Ts = zeros(nPhi,nAltitudes);
```

```
T4 = zeros(nPhi,nAltitudes);
for iAltitude = 1:nAltitudes
    [Ts(:,iAltitude), T4(:,iAltitude)] =realTurbofanThrust(phi,...
        UInf(iAltitude),altitudes(iAltitude),HondaJet.engine,fluid);
end
T4_ratio = T4 / HondaJet.engine.turbineInletTemperature;

figure(1)
hold on
plot(phi,Ts(:,1),'b','LineWidth',2.0);
plot(phi,Ts(:,2),'r','LineWidth',2.0);
plot(phi,Ts(:,3),'k','LineWidth',2.0);
l = legend('Altitude = 0 m, U_\infty = 90 m/s',...
    'Altitude = 9144 m, U_\infty = 150 m/s',...
    'Altitude = 13106.4 m, U_\infty = 200 m/s');
xlabel('Equivalence Ratio, \phi','FontSize',22,'FontWeight','bold');
ylabel('Thrust, T [N]','FontSize',22,'FontWeight','bold');
scale = axis;
axis([0 5 scale(3) scale(4)]);
axesh = findobj('Type', 'axes');
set(axesh, 'Box','on');
set(l,'FontSize',20);
set(gca,'FontSize',20);

figure(2)
hold on
plot(phi,T4_ratio(:,1),'b','LineWidth',2.0);
plot(phi,T4_ratio(:,2),'r','LineWidth',2.0);
plot(phi,T4_ratio(:,3),'k','LineWidth',2.0);
xlabel('Equivalence Ratio, \phi','FontSize',22,'FontWeight','bold');
ylabel('T_4/T_{4,max}','FontSize',22,'FontWeight','bold');
l = legend('Altitude = 0 m, U_\infty = 90 m/s',...
    'Altitude = 9144 m, U_\infty = 150 m/s',...
    'Altitude = 13106.4 m, U_\infty = 200 m/s');
axesh = findobj('Type', 'axes');
set(axesh, 'Box','on');
set(l,'FontSize',20);
set(gca,'FontSize',20);
```

A1.2. Combustor.m code.

```
function [T4,mDot4] = combustor(phi, UInf, altitude, engine, fluid)
    % function: combustor
    % ------------------------------------------------------------------------
    % This function returns the turbine inlet temperature and mass
    % fractions.
    %   Inputs:
    %       phi = equivalence ratio
    %       UInf = free stream velocity [m/s]
    %       engine = structure containing the engine parameters
    %       fluid = structure containing the fluid properties including the
    %           cantera solution object
    %   Outputs:
    %       T4 = The turbine inlet temperature [K]
    %       mDot4 = composition of the turbine inlet gas (in mass flow) [kg/s]

    [~, ~, ~, rhoSL] = atmosisa(0);
    [~, ~, ~, rho] = atmosisa(altitude);
    mDotAir = engine.mDotSeaLevel * rho / rhoSL; % The mass flow rate at certain altitude.

    setCombustorInletConditions(phi, UInf, altitude, engine, fluid);
    f = fuelAirRatio(fluid);
    mDotFuel = mDotAir * f;
    mDot4 = mDotFuel + mDotAir;
    equilibrate(fluid.gas,'HP');    % Simulate combustion using HP equilibrium
    T4 = temperature(fluid.gas);
end
function setCombustorInletConditions(phi, UInf, altitude, engine, fluid)
    % function: setCombustorInletConditions
    % ------------------------------------------------------------------------
    % This function sets the fluid.gas solution object to the combustor
    % inlet conditions after the fuel is mixed with the oxidizer
    %   Inputs:
    %       phi = equivalence ratio
    %       UInf = free stream velocity [m/s]
```

```
%       engine = structure containing the engine parameters
%       fluid = structure containing the fluid properties including the
%           cantera solution object
%   Outputs:
%       fluid.gas object is altered to inlet state

% Freestream conditions
[T0 , ~ , p0 , ~] = atmosisa(altitude);
gamma = fluid.gamma;
R = fluid.R;
U0 = UInf;

% The efficiencies
n.d=engine.efficiencies.diffuser;
n.f=engine.efficiencies.fan;
n.c=engine.efficiencies.compressor;
n.b=engine.efficiencies.combustor;
n.t=engine.efficiencies.turbine;
n.n=engine.efficiencies.coreNozzle;
n.n1=engine.efficiencies.fanNozzle;

% Compressor ratios
pi.f = engine.fanCompressionRatio;
pi.c = engine.overallCompressionRatio./pi.f;

% The reference conditions
M0 = U0./sqrt(gamma.*R.*T0);
tau.r = 1+(gamma-1)./2.*M0.^2;
pi.r = tau.r.^(gamma/(gamma-1.0));

% Diffuser pressure ratio
tau.d = 1.0; %by assumption
pi.d = (1.0+n.d.*(tau.r-1.0)).^(gamma./(gamma-1.0))./pi.r;

% Compute the fan and compressor temperature ratois
tau.c = 1.0+1.0./n.c.*(pi.c.^((gamma-1.0)./gamma)-1.0);
tau.f = 1.0+1.0./n.f.*(pi.f.^((gamma-1.0)./gamma)-1.0);

p3 = p0 * pi.r * pi.d * pi.f * pi.c;
T3 = T0 * tau.r * tau.d * tau.f * tau.c;

%set the inlet gas
nSp = nSpecies(fluid.gas);
X3 = zeros(nSp,1);
iFuel = speciesIndex(fluid.gas,fluid.fuel.name);
X3(iFuel)=phi;
iO2 = speciesIndex(fluid.gas,'o2');
iN2 = speciesIndex(fluid.gas,'n2');
X3(iO2)= fluid.fuel.nC+fluid.fuel.nH/4.0;
X3(iN2)=3.76*X3(iO2);
set(fluid.gas,'T',T3,'P',p3,'X',X3)
end

function f = fuelAirRatio(fluid)
    % function: fuelAirRatio
    % ------------------------------------------------------------------------
    % Determines the fuel-air ratio of the unburned gas.
    %   Inputs:
    %       fluid = structure containing the fluid properties including the
    %           cantera solution object
    %   Outputs:
    %       f = fuel-air ratio
    Y = massFractions(fluid.gas);
    iFuel = speciesIndex(fluid.gas,fluid.fuel.name);
    iO2 = speciesIndex(fluid.gas,'o2');
    iN2 = speciesIndex(fluid.gas,'n2');
    f = Y(iFuel)/(Y(iO2)+Y(iN2));
end
```

Appendix 2. MATLAB code for Problem 2.

A2.1. Driver Code.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  ME357 Spring 2017
```

```
%  Problem Set #3
%  2. Criuse Conditions for the HondaJet
%  (d)(e) Driver code
%
%  Rui Xu (ruixu@stanford.edu)
%  Uses as the driver code for Problem 2.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
clc

%define a structure containing the HondaJet's parameters
HondaJet.airframe.S = 17.3;                          %[m^2] , reference area
HondaJet.airframe.b = 12.12;                         %[m] , wing span
HondaJet.airframe.W = 41000;                         %[N] , weight (which is max takeoff weight)
HondaJet.airframe.CD0 = 0.01;                        %[] , zero-lift drag coefficient (or parasite drag
coeff.)
HondaJet.airframe.e = 0.8;                           %[] , Oswald Efficiency
HondaJet.engine.overallCompressionRatio = 24;        %[]
HondaJet.engine.turbineInletTemperature=1600;        %[K]
HondaJet.engine.mDotSeaLevel = 8.0;                  %[kg/s]
HondaJet.engine.nEngines = 2;                        %[]
HondaJet.engine.bypassRatio = 2.9;                   %[]
HondaJet.engine.fanCompressionRatio = 2.0;           %[]
HondaJet.engine.efficiencies.diffuser = 0.95;        %[]
HondaJet.engine.efficiencies.fan = 0.92;             %[]
HondaJet.engine.efficiencies.compressor = 0.87;      %[]
HondaJet.engine.efficiencies.combustor = 1.00;       %[]
HondaJet.engine.efficiencies.turbine = 0.91;         %[]
HondaJet.engine.efficiencies.coreNozzle = 0.98;      %[]
HondaJet.engine.efficiencies.fanNozzle = 0.98;       %[]

%define a structure containing the fluid
fluid.gas = importPhase('ndodecane_mech.cti');
fluid.fuel.name = 'nc12h26';
fluid.fuel.nC = 12;
fluid.fuel.nH = 26;
fluid.gamma = 1.4;
fluid.R = 287;

% Input the altitude and cruise speed
altitude = 13106.4;
UInf = 200;

%% (d) Calculate the range by solving the ODE
airframeWBackup = HondaJet.airframe.W;
g = 9.8;
dt = 100;    % numerical time step (s)
i = 1;
WFuel = 5600;
W(i) = HondaJet.airframe.W;
t(i) = 0;
s(i) = 0;

while (true)
   HondaJet.airframe.W = W(i);
   mDotFuel(i) = mFuelDot(altitude,UInf,HondaJet,fluid);
   dW = -mDotFuel(i) * HondaJet.engine.nEngines * g * dt;
   WFuel = WFuel + dW;
   if (WFuel <= 0)
       break;
   end
   W(i+1) = W(i) + dW;
   t(i+1) = t(i) + dt;
   s(i+1) = s(i) + UInf * dt;
   i = i + 1;
end

figure(1)
plot(t/3600,W/1000,'b','LineWidth',2.0);
xlabel('Time, t (h)','FontSize',22,'FontWeight','bold');
ylabel('Weight, W [kN]','FontSize',22,'FontWeight','bold');
title('Aircraft Weight vs. Time','FontSize',22,'FontWeight','bold');
axesh = findobj('Type', 'axes');
set(axesh, 'Box','on');
set(gca,'FontSize',20);
```

```
figure(2)
plot(s/1000,W/1000,'b','LineWidth',2.0);
xlabel('Range, s (km)','FontSize',22,'FontWeight','bold');
ylabel('Weight, W [kN]','FontSize',22,'FontWeight','bold');
title('Aircraft Weight vs. Range','FontSize',22,'FontWeight','bold');
axesh = findobj('Type', 'axes');
set(axesh, 'Box','on');
set(gca,'FontSize',20);

%% (e) Calculate the range using the Breguet range equation
i = 1;
WFuel = 5600;
W_TSFC(i) = airframeWBackup;
[T0, a0, ~, ~] = atmosisa(altitude);
Tref = 288;
M0 = UInf / a0;
TSFC = (0.4 + 0.45 * M0) * sqrt(T0 / Tref);  % Empirical formula of TSFC
TSFC = TSFC * 0.453592 / 4.44822 / 3600;     % Keep unit consistent
T_Req = ThrustRequired(UInf, altitude, HondaJet.airframe);
t_TSFC(i) = 0;
s_TSFC(i) = 0;
while (true)
   HondaJet.airframe.W = W_TSFC(i);
   dW_TSFC = -TSFC * T_Req * HondaJet.engine.nEngines * g * dt;
   WFuel = WFuel + dW_TSFC;
   if (WFuel <= 0)
       break;
   end
   W_TSFC(i+1) = W_TSFC(i) + dW_TSFC;
   t_TSFC(i+1) = t_TSFC(i) + dt;
   s_TSFC(i+1) = s_TSFC(i) + UInf * dt;
   i = i + 1;
end

figure(3)
hold on
plot(s/1000,W/1000,'b','LineWidth',2.0);
plot(s_TSFC/1000,W_TSFC/1000,'r','LineWidth',2.0);
xlabel('Range, s (km)','FontSize',22,'FontWeight','bold');
ylabel('Weight, W [kN]','FontSize',22,'FontWeight','bold');
l = legend('Using mDot ODE','Using Breguet Range Equation (TSFC)');
scale = axis;
axesh = findobj('Type', 'axes');
set(axesh, 'Box','on');
set(l,'FontSize',20);
set(gca,'FontSize',20);
```

A2.2. The "mFuelDot.m" Code for Problem 2(b).

```
function mDotFuel = mFuelDot(altitude,UInf,jet,fluid)
    % function: mDotFuel
    % ------------------------------------------------------------------------
    % This function returns the fuel mass flow rate needed to produce
    % the required thrust.
    %   Inputs:
    %       altitude = the altitude. [m].
    %       UInf = free stream velocity [m/s]
    %       jet = structure of the HondaJet
    %       fluid = structure containing the fluid properties including the
    %           cantera solution object
    %   Outputs:
    %       mDotFuel = the fuel mass flow rate [kg/s]

    % Use bisection method to find the equivalence ratio
    phi_max = 1;
    phi_min = 0.01;
    tol = 1e-5;
    fun = 1e4;
    T_Req = ThrustRequired(UInf, altitude, jet.airframe);
    while (abs(fun) > tol)
        phi = 0.5 * (phi_max + phi_min);
        [T_avail, ~] = realTurbofanThrust(phi, UInf, altitude, jet.engine, fluid);
        fun = T_avail - T_Req;
        if (fun > 0)
            phi_max = phi;
```

```
        else
            phi_min = phi;
        end
    end

    % Calculate mDotFuel from equivalence ratio solution
    [~ , ~ , ~ , rho0] = atmosisa(altitude);
    [~ , ~ , ~ , rhoSeaLevel] = atmosisa(0.0);
    mDotAir = jet.engine.mDotSeaLevel*rho0/rhoSeaLevel;
    MW_fuel = 12.0107 * 12 + 1.00794 * 26;
    MW_o2 = 15.9994 * 2;
    MW_N2 = 14.0067 * 2;
    m_fuel = phi * MW_fuel;
    m_air = (12 + 26 / 4) * (MW_o2 + 79 / 21 * MW_N2);
    f = m_fuel / m_air;
    mDotFuel = mDotAir * f;
end
```

Appendix 3. MATLAB code for Problem 3.

A3.1. Driver code.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  ME357 Spring 2017
%  Problem Set #3
%  3. Nitric Oxide Formation
%  Driver code
%
%  Rui Xu (ruixu@stanford.edu)
%  Uses as the driver code for Problem 1.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
clc

%define a structure containing the HondaJet's parameters
HondaJet.airframe.S = 17.3;                     %[m^2] , reference area
HondaJet.airframe.b = 12.12;                    %[m] , wing span
HondaJet.airframe.W = 41000;                    %[N] , weight (which is max takeoff weight)
HondaJet.airframe.CD0 = 0.01;                   %[] , zero-lift drag coefficient (or parasite drag
coeff.)
HondaJet.airframe.e = 0.8;                      %[] , Oswald Efficiency
HondaJet.engine.overallCompressionRatio = 24;   %[]
HondaJet.engine.turbineInletTemperature=1600;   %[K]
HondaJet.engine.mDotSeaLevel = 8.0;             %[kg/s]
HondaJet.engine.nEngines = 2;                   %[]
HondaJet.engine.bypassRatio = 2.9;              %[]
HondaJet.engine.fanCompressionRatio = 2.0;      %[]
HondaJet.engine.efficiencies.diffuser = 0.95;   %[]
HondaJet.engine.efficiencies.fan = 0.92;        %[]
HondaJet.engine.efficiencies.compressor = 0.87; %[]
HondaJet.engine.efficiencies.combustor = 1.00;  %[]
HondaJet.engine.efficiencies.turbine = 0.91;    %[]
HondaJet.engine.efficiencies.coreNozzle = 0.98; %[]
HondaJet.engine.efficiencies.fanNozzle = 0.98;  %[]

%define a structure containing the fluid
fluid.gas = importPhase('ndodecane_mech.cti');
fluid.fuel.name = 'nc12h26';
fluid.fuel.nC = 12;
fluid.fuel.nH = 26;
fluid.gamma = 1.4;
fluid.R = 287;

% Input the altitude and cruise speed
altitude = 0;
UInf = 90;

mDot_NO_max = 0.04e-3;
phi = (0.15:0.05:5)';
nPhi = length(phi);
mDot_NO = zeros(nPhi,5);
iNO = speciesIndex(fluid.gas,'no');
```

```
%% (a) Standard Combustor
for i = 1:1:nPhi
    [~, mDot4] = combustor(phi(i), UInf, altitude, HondaJet.engine, fluid);
    Y = massFractions(fluid.gas);
    mDot_NO(i,1) = Y(iNO) * mDot4;
end

figure (1)
semilogy(phi,mDot_NO(:,1)/mDot_NO_max,'b','LineWidth',2.0);
hold on
semilogy(phi,ones(1,nPhi),'r--','LineWidth',2.0);
xlabel('Equivalence Ratio, \phi','FontSize',22,'FontWeight','bold');
ylabel('mDot_{NO} / mDot_{NO,max}','FontSize',22,'FontWeight','bold');
axesh = findobj('Type', 'axes');
set(axesh, 'Box','on');
set(gca,'FontSize',20);

%% (b) RQL Combustor
beta_C = [1 5 10];
nBetaC = length(beta_C);
for j = 1:1:nBetaC
    HondaJet.engine.betaC = beta_C(j);
    for i = 1:1:nPhi
        [~,mDot4] = RQLCombustor(phi(i),UInf, altitude, HondaJet.engine, fluid);
        Y = massFractions(fluid.gas);
        mDot_NO(i, j + 1) = Y(iNO) * mDot4;
    end
end

figure (2)
semilogy(phi,mDot_NO(:,1)/mDot_NO_max,'b','LineWidth',2.0);
hold on
semilogy(phi,mDot_NO(:,2)/mDot_NO_max,'k','LineWidth',2.0);
semilogy(phi,mDot_NO(:,3)/mDot_NO_max,'c','LineWidth',2.0);
semilogy(phi,mDot_NO(:,4)/mDot_NO_max,'m','LineWidth',2.0);
semilogy(phi,ones(1,nPhi),'r--','LineWidth',2.0);
l = legend('Normal Combustor','RQL, \beta_C = 1',...
    'RQL, \beta_C = 5','RQL, \beta_C = 10','CAEP standard');
xlabel('Equivalence Ratio, \phi','FontSize',22,'FontWeight','bold');
ylabel('mDot_{NO} / mDot_{NO,max}','FontSize',22,'FontWeight','bold');
axesh = findobj('Type', 'axes');
set(axesh, 'Box','on');
set(gca,'FontSize',20);
set(l,'FontSize',20);

%% (c) Empirical Correlation
for i = 1:1:nPhi
    [T4,mDot4] = combustor(phi(i), UInf, altitude, HondaJet.engine, fluid);
    p4 = pressure(fluid.gas);
    MW4 = meanMolecularWeight(fluid.gas);
    MW_NO = 14.0067 + 15.9994;
    X_NO = 3.32e-12 * exp(0.008 * T4) * p4^0.5;
    mDot_NO(i,5) = mDot4 * X_NO * MW_NO / MW4;
end

figure (3)
semilogy(phi,mDot_NO(:,1)/mDot_NO_max,'b','LineWidth',2.0);
hold on
semilogy(phi,mDot_NO(:,2)/mDot_NO_max,'k','LineWidth',2.0);
semilogy(phi,mDot_NO(:,3)/mDot_NO_max,'c','LineWidth',2.0);
semilogy(phi,mDot_NO(:,4)/mDot_NO_max,'m','LineWidth',2.0);
semilogy(phi,mDot_NO(:,5)/mDot_NO_max,'g','LineWidth',2.0);
semilogy(phi,ones(1,nPhi),'r--','LineWidth',2.0);
l = legend('Normal Combustor','RQL, \beta_C = 1',...
    'RQL, \beta_C = 5','RQL, \beta_C = 10','Empirical correlation',...
    'CAEP standard');
xlabel('Equivalence Ratio, \phi','FontSize',22,'FontWeight','bold');
ylabel('mDot_{NO} / mDot_{NO,max}','FontSize',22,'FontWeight','bold');
axesh = findobj('Type', 'axes');
set(axesh, 'Box','on');
set(gca,'FontSize',20);
set(l,'FontSize',20);
```

A3.2. The "RQLCombustor.m" code.

```
function [T4,mDot4] = RQLCombustor(phi,UInf, altitude, engine, fluid)
    % function: RQLcombustor
    % ------------------------------------------------------------------------
    % This function returns the output mass flow vector as a function of
    % the rerouting ratio.
    %   Inputs:
    %       phi = the overall equivalence ratio for the combustor
    %       UInf = free stream velocity [m/s]
    %       engine = structure containing the engine parameters
    %       fluid = structure containing the fluid properties including the
    %           cantera solution object
    %   Outputs:
    %       mDot4 = composition of the turbine inlet gas (in mass flow) [kg/s]
    %
    % Prepare and pre-calculate some parameters
    beta_C = engine.betaC;
    [~, ~, ~, rhoSL] = atmosisa(0);
    [~, ~, ~, rho] = atmosisa(altitude);
    mDotAir = engine.mDotSeaLevel * rho / rhoSL; % The mass flow rate at certain altitude.
    mDotAir2 = mDotAir * beta_C / (1 + beta_C);
    mDotAir1 = mDotAir * 1 / (1 + beta_C);

    % 1. Rich Burn
    % Get rerouted air properties at State 3.
    setCombustorInletConditions(0, UInf, altitude, engine, fluid);
    MW3_a2 = meanMolecularWeight(fluid.gas);
    h3_a2 = enthalpy_mass(fluid.gas);
    H3_a2 = h3_a2 * mDotAir2;

    % Get the core mixture properties at State 3.
    setCombustorInletConditions(phi, UInf, altitude, engine, fluid);
    % Calculate the total mass flow rate and then eqilibrate the mixture
    f = fuelAirRatio(fluid);
    mDotFuel = mDotAir1 * f;
    mDot4 = mDotFuel + mDotAir1 + mDotAir2;
    equilibrate(fluid.gas,'HP');

    % Get properties after rich combustion
    T4_rich = temperature(fluid.gas);
    p4_rich = pressure(fluid.gas);
    h4_rich = enthalpy_mass(fluid.gas);
    H4_rich = h4_rich * (mDotFuel + mDotAir1);
    X4_rich = moleFractions(fluid.gas);

    % 2. Quenching and Dilution
    iNO = speciesIndex(fluid.gas,'no');
    iO2 = speciesIndex(fluid.gas,'o2');
    iN2 = speciesIndex(fluid.gas,'n2');

    X4_rich(iNO) = 0;               % Set NO mole fraction to zero
    X4_rich_sum = sum(X4_rich);     % and then renormalize
    X4_rich = X4_rich / X4_rich_sum;
    set(fluid.gas,'P',p4_rich,'T',T4_rich,'X',X4_rich); % Re-set the mixture
    MW4_rich = meanMolecularWeight(fluid.gas);

    % Get the mole values for rerouted air
    X_a2 = MW4_rich * 1 / MW3_a2 * mDotAir2 / (mDotFuel + mDotAir1);

    % Add the rerouted air moles into the burned mixture
    X4_rich(iO2) = X4_rich(iO2) + X_a2 * 0.21;
    X4_rich(iN2) = X4_rich(iN2) + X_a2 * 0.79;
    X4_sum = sum(X4_rich);

    % Set the diluted gas mixture
    X4_dil = X4_rich;
    X4_dil = X4_dil / X4_sum;
    H4_dil = H3_a2 + H4_rich;
    h4_dil = H4_dil / mDot4;
    p4_dil = p4_rich;
    set(fluid.gas,'P',p4_dil,'H',h4_dil,'X',X4_dil);

    % 3. Lean Burn
    equilibrate(fluid.gas,'HP');
    T4 = temperature(fluid.gas);
end
```