

Space Shuttle Main Engine

November 4, 2022

Contents

1	Introduction	1
2	Oxygen	2
2.1	First Enthalpy Exchanger	2
2.2	Combustion Chamber	4
2.3	Nozzle Performance	9
A	Code listings	10
A.1	SSMEMain.py	10
A.2	exchanger.py	11
A.3	Iterater.py	18

1 Introduction

The Space Shuttle Main Engine (SSME) is a liquid-fueled rocket engine used to power the Space Shuttle Orbiter. This report will discuss the design and methodologies of the SSME code, as well as the results of the code. For this project, we were given the parameters below:

Combustion Chamber Data	
Fuel	Hydrogen
Oxidizer	O_2
Fuel/Oxidizer Ratio	0.166
Hydrogen Injection Temperature	850 K
Oxidizer Injection Temperature	530 K
Chamber Pressure	204 atm

R_t	5.15 inch
R_e	45.35 inch
θ_p	32 degree

2 Oxygen

2.1 First Enthalpy Exchanger

We begin with computing the equivalence ratio ϕ for the oxygen first enthalpy exchanger. The equivalence ratio is defined as the ratio of the mass of fuel to the mass of oxidizer. The mass of fuel is the mass of hydrogen, and the mass of oxidizer is the mass of oxygen. The mass of hydrogen is $\phi \frac{mol_{O_2}}{mol_{H_2}} = \Phi$.

From here we are to calculate the standard heat of formation by:

$$\Delta \hat{h}_{H_2} = \hat{h}_{H_2}(850) + \hat{R}T^2 \int_{pf}^{pi} \frac{1}{p} \frac{-\hat{b}_{H_2}p}{\hat{R}T^2} dp \quad (1)$$

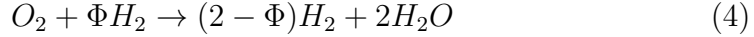
$$\Delta \hat{h}_{O_2} = \hat{h}_{O_2}(530) + \hat{R}T^2 \int_{pf}^{pi} \frac{1}{p} \frac{-\hat{b}_{O_2}p}{\hat{R}T^2} dp \quad (2)$$

Species	\hat{b}
H_2	0.0266
O_2	0.0318
H_2O	0.03049
N_2	0.0391
F_2	0.02896
HF	0.0739

Where \hat{b} is read from a table developed in `exchanger.py` file. $\Delta\hat{h}_i$ is calculated from both hydrogen and oxygen. Once we calculate $\Delta\hat{h}_i$ for each, we can calculate the enthalpy of the mixture.

$$\Delta\hat{h}_{mix} = \Phi\Delta\hat{h}_{H_2} + \Delta\hat{h}_{O_2} \quad (3)$$

We can then apply this to our chemical equation.



Note: notice the $(2 - \Phi)$ in the equation above. This is because we calculated the enthalpy of formation for hydrogen and not hydrogen gas H_2 .

From here we can sum the molar enthalpies of formation to get the first estimate of the first reaction enthalpy

$$\Delta\hat{h}_r = \Phi\Delta\hat{h}_{H_2} + 2(\Delta\hat{h}_{H_2O}) \quad (5)$$

From here we go into our Temperature finding Algorithm. We are given the following parameters:

$$r = \Phi - 1 \quad (6)$$

$$q = 1 \quad (7)$$

$$r_{percent} = \frac{r}{r + q} \quad (8)$$

$$q_{percent} = \frac{q}{r + q} \quad (9)$$

From here we use the values of (Eq. 6) - (Eq. 9) to calculate the temperature of the first enthalpy exchanger.

$$T_{H_2} = \text{h2_tableT}(\Delta\hat{h}_r) \quad (10)$$

$$T_{H_2O} = \text{ho2_tableT}(q_{percent}\Delta\hat{h}_r) \quad (11)$$

Once we have returned the values of Temperature, we then average the two values to get the final temperature of the first enthalpy exchanger.

$$T_C = \frac{T_{H_2} + T_{H_2O}}{2} \quad (12)$$

2.2 Combustion Chamber

For the Combustion Chamber, we have the two equations below:



The Gibbs Free Energy for each species is given by:

$$\hat{g}_{H_2O} = \text{h2o_table_g}(T_C) \quad (15)$$

$$\hat{g}_{H_2} = \text{h2_table_g}(T_C) \quad (16)$$

$$\hat{g}_{OH} = \text{oh_table_g}(T_C) \quad (17)$$

$$\hat{g}_H = \text{h_table_g}(T_C) \quad (18)$$

We can then calculate the Gibbs Free Energy of the reaction by:

$$\Delta\hat{g}_r = 2(\hat{g}_H - \hat{g}_{H_2}) \quad (19)$$

$$\Delta\hat{g}_r = \hat{g}_H - \hat{g}_{OH} + \hat{g}_{H_2O} \quad (20)$$

Now is the point where we involve the **Law of Mass Action**:

Species	$\nu'_{(i)}$	$\nu''_{(i)}$	$\nu''_{(i)} - \nu'_{(i)}$
H_2O	1	0	-1
OH	0	1	1
H	0	1	1

We an now use:

$$K_n = \prod_{i=1}^{N_s} X_{(n)}^{(\nu''_{(i)} - \nu'_{(i)})} = \left(\frac{p}{p_a}\right)^{-\sigma_v} K_p \quad (21)$$

Where;

$$\sigma_v = \sum_{i=1}^{N_s} (\nu''_{(i)} - \nu'_{(i)}) \quad (22)$$

$$K_p = e^{-\frac{\Delta G}{RT}} \quad (23)$$

$$\Delta G = \sum_{i=1}^{N_s} (\nu''_{(i)} \hat{g}_{(i)}) \hat{g}_{(i)} \quad (24)$$

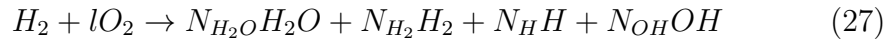
From here we can solve for the equilibrium constant K_n .

$$K_{H_2O} = \left(\frac{204atm}{1atm}\right)^{-\frac{\Delta G_{H_2O}}{RT_C}} \quad (25)$$

$$K_{H_2} = \left(\frac{204atm}{1atm}\right)^{-\frac{\Delta G_{H_2}}{RT_C}} \quad (26)$$

Now we move into the chemistry problem.

Atom Balance:



Recall, due to the fule being Hydrogen;

$$mass_{fuel} = 2.02 \quad (28)$$

$$mass_{Oxidizer} = \frac{mass_{fuel}}{\phi} \quad (29)$$

$$l = \frac{mass_{Oxidizer}}{2\hat{m}_{O_2}} \quad (30)$$

Where \hat{m}_{O_2} is the molar mass of Oxygen. And ϕ is the Fuel to Oxidizer ratio, and not the equivalence ratio. Revisiting the **Atom Balance** equation, we can now solve for the number of moles of each species.

For $H_2O \rightarrow OH + H$:

$$H_2 : 2 = 2N_{H_2O} + 2N_{H_2} + N_H + N_{OH} \quad (31)$$

$$O_2 : 2l = N_{H_2O} + N_{OH} \quad (32)$$

$$1 = N_{H_2O} + N_{H_2} + \frac{1}{2}N_H + \frac{1}{2}N_{OH} \quad (33)$$

Where $\frac{1}{2}N_H + \frac{1}{2}N_{OH} \rightarrow 0$. Resulting in:

$$1 = N_{H_2O} + N_{H_2} \quad (34)$$

$$2(0.38050899) = N_{H_2O} + N_{OH} \quad (35)$$

Where $N_{OH} \rightarrow 0$. Resulting in:

$$0.760592 = N_{H_2O} \quad (36)$$

$$X = \frac{N_{Molecule}}{N_{Total}} \quad (37)$$

$$X_{H_2O} = \frac{N_{H_2O}}{N_{Total}} \quad (38)$$

$$X_H = \frac{N_H}{N_{Total}} \quad (39)$$

$$X_{OH} = \frac{N_{OH}}{N_{Total}} \quad (40)$$

$$X_{H_2O} = \frac{N_{H_2O}}{N_{H_2O} + N_{H_2} + N_H + N_{OH}} \quad (41)$$

$$X_H = \frac{N_H}{N_{H_2O} + N_{H_2} + N_H + N_{OH}} \quad (42)$$

$$X_{OH} = \frac{N_{OH}}{N_{H_2O} + N_{H_2} + N_H + N_{OH}} \quad (43)$$

Recall, $N_{OH} \rightarrow 0$ and $N_H \rightarrow 0$. Resulting in:

$$K_{H_2O} = \frac{\frac{N_{OH}}{N_{H_2O}+N_{H_2}} \frac{N_H}{N_{H_2O}+N_{H_2}}}{\frac{N_{H_2O}}{N_{H_2O}+N_{H_2}}} \quad (44)$$

Resulting in,

$$K_{H_2O} = \frac{N_H N_{OH}}{(N_H + N_{H_2O}) N_{H_2O}} \quad (45)$$

Similarly we follow the same steps for the $H_2 \rightarrow H + H$ reaction.

$$X_{H_2} = \frac{N_{H_2}}{Total} \quad (46)$$

$$X_H = \frac{N_H}{Total} \quad (47)$$

$$X_{H_2} = \frac{N_{H_2}}{N_{H_2} + N_{H_2O} + N_H + N_{OH}} \quad (48)$$

$$X_H = \frac{N_H}{N_{H_2} + N_{H_2O} + N_H + N_{OH}} \quad (49)$$

$$K_{H_2} = \frac{\left(\frac{N_H}{N_{H_2O}+N_{H_2}}\right)^2}{\frac{N_{H_2}}{N_{H_2O}+N_{H_2}}} \quad (50)$$

$$K_{H_2} = \frac{N_H^2}{(N_{H_2O} + N_{H_2}) N_{H_2}} \quad (51)$$

As a result we have 4 coupled equations:

$$1 = N_{H_2O} + N_{H_2} \quad (52)$$

$$0.760592 = N_{H_2O} \quad (53)$$

$$K_{H_2O} = \frac{N_H N_{OH}}{(N_H + N_{H_2O}) N_{H_2O}} \quad (54)$$

$$K_{H_2} = \frac{N_H^2}{(N_{H_2O} + N_{H_2}) N_{H_2}} \quad (55)$$

Where K_{H_2O} and K_{H_2} are calculated values.
Solving these equations we get:

$$N_{H_2O} = 0.760592 \quad (56)$$

$$N_{H_2} = 0.239408 \quad (57)$$

$$N_H = 0.05333 \quad (58)$$

$$N_{OH} = 0.101958 \quad (59)$$

Species	$N_{(i)}$	$X_{(i)}$	$m_{(i)}$	$Y_{(i)}$	$\hat{c}_{p(i)}$	$c_{p(i)}$	$Y_{(i)}c_p$
H_2O	N_{H_2O}	$X_{H_2O} = N_{H_2O}$	$\hat{m}N_{H_2O}$	$\frac{\hat{m}}{m_{(i)}}$	$\hat{c}_{p(H_2O)}(T_C)$	$\frac{\hat{c}_p}{\hat{m}}$	$Y_{H_2O}c_p$
H_2	N_{H_2}	$X_{H_2} = N_{H_2}$	$\hat{m}N_{H_2}$	$\frac{\hat{m}}{m_{(i)}}$	$\hat{c}_{p(H_2)}(T_C)$	$\frac{\hat{c}_p}{\hat{m}}$	$Y_{H_2}c_p$
H	N_H	$X_H = N_H$	$\hat{m}N_H$	$\frac{\hat{m}}{m_{(i)}}$	$\hat{c}_{p(H)}(T_C)$	$\frac{\hat{c}_p}{\hat{m}}$	Y_Hc_p
OH	N_{OH}	$X_{OH} = N_{OH}$	$\hat{m}N_{OH}$	$\frac{\hat{m}}{m_{(i)}}$	$\hat{c}_{p(OH)}(T_C)$	$\frac{\hat{c}_p}{\hat{m}}$	$Y_{OH}c_p$

Now we can calculate c_p , we sum the $Y_{(i)}c_p$ values for each species.

$$c_p = \sum_i Y_{(i)}c_p \quad (60)$$

$$c_p = Y_{H_2O}c_p + Y_{H_2}c_p + Y_Hc_p + Y_{OH}c_p \quad (61)$$

We perform now calculate the gas constant R .

$$R = \frac{\hat{R}}{m_{Total}} \quad (62)$$

$$c_v = c_p - R \quad (63)$$

$$\gamma = \frac{c_p}{c_p - \frac{\hat{R}}{\sum_i X_{(i)}m_{(i)}}} = \frac{c_p}{c_p - \frac{\hat{R}}{m_{Total}}} = \frac{c_p}{c_p - R} = \frac{c_p}{c_v} \quad (64)$$

2.3 Nozzle Performance

We begin by defining the Area Ratio:

$$A_r = \frac{\pi R_e^2}{\pi R_t^2} \quad (65)$$

Allowing for us to solve for the Mach number:

$$A_r = \left(\frac{\gamma + 1}{2} \right)^{-\frac{\gamma+1}{2(\gamma-1)}} \frac{1}{M_e} \left[1 + \frac{\gamma - 1}{2} M_e^2 \right]^{\frac{\gamma+1}{2(\gamma-1)}} \quad (66)$$

We can solve for Mach number M_e by making use of the `scipy.optimize.fsolve` function, which returns the roots of a non-linear function. After solving for M_e we can calculate the exit velocity v_e , exit Pressure p_e and exit Temperature T_e .

$$T_e = T_c \left[1 + \frac{\gamma - 1}{2} M_e^2 \right]^{-1} \quad (67)$$

$$p_e = p_c \left[1 + \frac{\gamma - 1}{2} M_e^2 \right]^{-\frac{\gamma}{\gamma-1}} \quad (68)$$

$$v_e = M_e \sqrt{\gamma R T_e} \quad (69)$$

After solving for (Eq. 69), (Eq. 68) and (Eq. 67) we can calculate the mass flow rate \dot{m} .

$$\dot{m} = \frac{A_t p_c \sqrt{\gamma}}{\sqrt{R T_c}} \left[\frac{2}{\gamma + 1} \right]^{\frac{\gamma+1}{2(\gamma-1)}} \quad (70)$$

Then once you have the mass flow rate you can calculate the thrust \mathcal{T} , specific impulse I_{sp} and the thrust coefficient C_T .

$$\mathcal{T} = \dot{m} v_e + p_e A_e - p_c A_t \quad (71)$$

$$I_{sp} = \frac{\mathcal{T}}{\dot{m} g_0} \quad (72)$$

$$C_T = \frac{\mathcal{T}}{p_c A_t} \quad (73)$$

A Code listings

A.1 SSMEMain.py

```
1 import pandas as pd
2 from exchanger import *
3
4
5 if __name__ == '__main__':
6
7     p_starting = 204 #starting pressure in atm
8     p_starting = p_starting * 101325 #convert to pascals
9     FuelOxidizer_ratio = 0.166
10    mol_O2 = 31.998 # g/mol #single molecule mass
11    mol_H2 = 2.016 # g/mol #single molecule mass
12    phi = (FuelOxidizer_ratio* mol_O2 / mol_H2)/2
13    Hydr_inject_temp = 850 #K
14    Ox_inject_temp = 530 #K
15    ThroatRadius = 5.15 #inches
16    ExitRadius = 45.35 #inches
17    FEE_T, FEE_del_h_total, del_h2_hat, b_hat_table = FEE(
18    Hydr_inject_temp, p_starting, Ox_inject_temp, phi)
19    print('phi = ', phi)
20
21    print('Temperature of the first enthalpy exchanger = ',
22    FEE_T, 'K')
23
24    print('Total enthalpy change in the first enthalpy
25    exchanger = ', FEE_del_h_total, 'J')
26
27
28    gamma, R, cp_total, cv, LawOfMassAction_df = OCC(FEE_T,
29    p_starting, FuelOxidizer_ratio)
30    print('gamma = ', gamma)
31
32
33    m_dotOx, v_eOx, p_eOx, ThrustOx, c_TOx, I_spOx, T_e_Ox =
34    OxNozzlePerf(ThroatRadius, ExitRadius, gamma, FEE_T, R,
35    p_starting)
36
37    print('Ox mass flow rate = ', m_dotOx, 'kg/s')
38    print('Ox exit velocity = ', v_eOx, 'm/s')
39    print('Ox exit pressure = ', p_eOx, 'Pa')
40    print('Ox thrust = ', ThrustOx, 'N')
41    print('Ox thrust coefficient = ', c_TOx)
42    print('Ox specific impulse = ', I_spOx, 's')
43    print('Ox exit temperature = ', T_e_Ox, 'K')
```

```

36
37
38     print('Flourine Problem')
39     b_hat_table, FEE_T_flourine = FEE_F(phi, p_starting,
40     del_h2_hat, b_hat_table, Ox_inject_temp)
41
42     print('Temperature of the first enthalpy exchanger (
43     Flourine) = ', FEE_T_flourine, 'K')
44
45     FCC(FEE_T_flourine)

```

A.2 exchanger.py

```

1 from Iterater import *
2 import pandas as pd
3 import numpy as np
4 from sympy import Eq, solve, var
5
6
7 def FEE(Hydr_inject_temp, p_starting, Ox_inject_temp, phi):
8     '''
9     This is a function that returns the calculations of the
10    first enthalpy exchanger
11    :param Hydr_inject_temp: temperature of hydrogen
12    injection in K
13    :param p_starting: starting pressure in Pa
14    :param Ox_inject_temp: temperature of oxygen injection in
15    K
16    :param phi: equivalence ratio
17    :return T the temperature of the first enthalpy exchanger
18    :return del_h_total the total enthalpy change in the
19    first enthalpy exchanger
20    :return b_hat_table the table of b_hat values
21    :return del_h2_hat the enthalpy change of H2
22    '''
23
24    h_f_h2o = abs(h2o_table_f(298)) #J/mol
25    h_f_h = h_table_f(298)/1000
26    h_oh_f = oh_table_f(298)/1000 # KJ/mol
27
28    b_hat_table = pd.DataFrame(columns=['substance', 'b_hat'
29    ])
30    b_hat_table = b_hat_table.append({'substance': 'H2', '
31    b_hat': 0.0266}, ignore_index=True)

```

```

27     b_hat_table = b_hat_table.append({'substance': 'H2O', '
b_hat': 0.03049}, ignore_index=True)
28     b_hat_table = b_hat_table.append({'substance': 'O2', '
b_hat': 0.0318}, ignore_index=True)
29     b_hat_table = b_hat_table.append({'substance': 'N2', '
b_hat': 0.0391}, ignore_index=True)
30
31     b_hat_h2 = b_hat_table['b_hat'][0]
32     b_hat_o2 = b_hat_table['b_hat'][2]
33
34     del_h2_hat = h2_tableH(Hydr_inject_temp) + b_hat_h2*(
p_starting - 101325)/1000
35     del_o2_hat = o2_tableH(Ox_inject_temp) + b_hat_o2*(
p_starting - 101325)/1000
36     mols_h2 = phi*2
37     mols_o2 = 1
38
39     del_h_total = del_h2_hat*mols_h2 + del_o2_hat*mols_o2
40     # At this point the chemical equation is O2 + mols*H2 ->
(phi-2)*H2 +2H2O
41     del_h_total = del_h_total + 2*h_f_h2o
42
43     # weighting function
44     r = 0.17947269916082
45     q = (1- r)/2
46
47     T_h2o = h2o_tableT(q*del_h_total)
48     T_h2 = h2_tableT(r*del_h_total)
49
50     T = (T_h2o + T_h2)/2
51
52     return T, del_h_total, del_h2_hat, b_hat_table
53
54 def OCC(FEE_T, p_starting, FO_ratio):
55     '''
56     This is a function that returns the calculations of the
oxygen combustion chamber
57     :param FEE_T: temperature of the first enthalpy exchanger
in K
58     :param p_starting: starting pressure in Pa
59     :param FO_ratio: fuel to oxidizer ratio
60     :return gamma: the ratio of specific heats
61     :return R: the gas constant
62     :return cp_total: the total specific heat capacity
63     :return cv : the specific heat capacity at constant

```

```

volume
64     :return LawOfMassAction_df: the dataframe of the law of
mass action
65     '''
66
67     # The chemical equation is H2O -> H + OH
68     molfrac_h2o_react = 1
69     molfrac_h_react = 0
70     molfrac_oh_react = 0
71     molfrac_h2o_prod = 0
72     molfrac_h_prod = 1
73     molfrac_oh_prod = 1
74
75     # The chemical equation is H2 -> 2H
76     molfrac_h2_react = 1
77     molfrac_2h_react = 0
78     molfrac_2h_prod = 2
79     molfrac_h2_prod = 0
80
81     molfrac_h2o_total = molfrac_h2o_prod - molfrac_h2o_react
82     molfrac_h_total = molfrac_h_prod - molfrac_h_react
83     molfrac_oh_total = molfrac_oh_prod - molfrac_oh_react
84     sigma_nu = molfrac_h2o_total + molfrac_h_total +
molfrac_oh_total
85     # print('sigma_nu = ', sigma_nu)
86
87     molfrac_h2_total = molfrac_h2_prod - molfrac_h2_react
88     molfrac_2h_total = molfrac_2h_prod - molfrac_2h_react
89     sigma_nu = molfrac_h2_total + molfrac_2h_total
90     # print('sigma_nu = ', sigma_nu)
91
92     # Gibbs free energy of formation
93     g_f_h2o = h2o_table_g(FEE_T)
94     g_f_h = h_table_g(FEE_T)
95     g_f_oh = oh_table_g(FEE_T)
96     g_f_h2 = h2_table_g(FEE_T)
97
98     # Gibbs free energy of reaction
99     g_r_h2 = 2*(g_f_h - g_f_h2)
100    g_r_h2o = g_f_h + g_f_h2o - g_f_oh
101
102    k_h2 = ((p_starting/101325)**-1)*np.exp(-g_r_h2/(8.314*
FEE_T))
103    k_h2o = ((p_starting/101325)**-1)*np.exp(-g_r_h2o/(8.314*
FEE_T))

```

```

104
105
106 # k_h2o = 0.007149
107 # k_h2 = 0.011874839
108
109 mass_fuel = 2.02 #H2
110 mass_oxider = mass_fuel/F0_ratio
111 l = mass_oxider/ (2*15.99)
112
113 N_h = var('N_h')
114 N_oh = var('N_oh')
115 N_h2 = var('N_h2')
116 N_h2o = var('N_h2o')
117
118
119 # solve the system of equations - Law of mass action
120 sol = solve([Eq(N_h2o, 2*l), Eq(N_h2o+N_h2, 1), Eq((N_h*
121 N_oh)/((N_h2+N_h2o)*N_h2o), k_h2o),
122 Eq((N_h**2)/((N_h2+N_h2o)*N_h2), k_h2)], [
123 N_h, N_oh, N_h2, N_h2o])
124 # split sol into two separate lists
125 sol1 = sol[0]
126 sol2 = sol[1]
127
128 # if all the solutions are real and positive then assign
129 mols_sol to the solution
130 if all(i > 0 for i in sol1) and all(i > 0 for i in sol2):
131     mols_sol = sol1
132 else:
133     mols_sol = sol2
134
135 # print('mols_sol = ', mols_sol)
136 N_h = mols_sol[0]
137 N_oh = mols_sol[1]
138 N_h2 = mols_sol[2]
139 N_h2o = mols_sol[3]
140 N_total = N_h + N_oh + N_h2 + N_h2o
141
142 molar_mass_h2o = 18.01528
143 molar_mass_h = 1.00794
144 molar_mass_oh = 17.00734
145 molar_mass_h2 = 2.01588
146 molar_mass_total = molar_mass_h2o + molar_mass_h +
147 molar_mass_oh + molar_mass_h2

```

```

145     LawOfMassAction_df = pd.DataFrame({'substance': ['N_H2O',
146     'N_H2', 'N_H', 'N_OH', 'Total'],
147     'mols': [N_h2o, N_h2,
148     N_h, N_oh, N_total]})
149     LawOfMassAction_df['X_(i)'] = LawOfMassAction_df['mols']
150     LawOfMassAction_df['m_hat_(i)'] = pd.Series([
151     molar_mass_h2o, molar_mass_h2, molar_mass_h, molar_mass_oh
152     , molar_mass_total])\
153     * LawOfMassAction_df['
154     X_(i)']
155
156     # sum m_hat_(i)[i=0 to i=3]
157     m_hat_total = sum(LawOfMassAction_df['m_hat_(i)'])-
158     LawOfMassAction_df['m_hat_(i)'][4]
159     LawOfMassAction_df['m_hat_(i)'][4] = m_hat_total
160     LawOfMassAction_df['Y_(i)'] = LawOfMassAction_df['m_hat_(
161     i)'] / m_hat_total
162
163     h2_cp = h2_table_cp(FEE_T)
164     h2o_cp = h2o_table_cp(FEE_T)
165     h_cp = h_table_cp(FEE_T)
166     oh_cp = oh_table_cp(FEE_T)
167
168     cp_hats = pd.Series([h2o_cp, h2_cp, h_cp, oh_cp])
169     cps = cp_hats / pd.Series([molar_mass_h2o, molar_mass_h2,
170     molar_mass_h, molar_mass_oh])
171     cp_df = pd.DataFrame({'cp_hat': cp_hats, 'cp': cps})
172     cp_df['Y_(i)*cp_(i)'] = cp_df['cp'] * LawOfMassAction_df[
173     'Y_(i)'][0:4]
174
175     cp_total = sum(cp_df['Y_(i)*cp_(i)'])
176
177     LawOfMassAction_df = pd.concat([LawOfMassAction_df, cp_df
178     ], axis=1)
179
180     R = 8.314/m_hat_total
181     cv = cp_total - R
182
183     gamma = cp_total/cv
184
185     return gamma, R, cp_total, cv, LawOfMassAction_df
186
187 def FEE_F(phi, p_starting, del_h2_hat, b_hat_table,
188     Ox_inject_temp):
189     '''

```

```

179     This is a function that returns the calculations of the
fuel enthalpy exchanger - flourine
180     :param phi the equivalence ratio
181     :param p_starting combustion pressure in Pa
182     :param del_h2_hat heat of formation of H2 from FEE -
Oxygen
183     :param b_hat_table Van der Waals table of constants
184     :param Ox_inject_temp: Oxidizer injection temperature in
K
185     :return T the temperature of the FEE in K
186     :return b_hat_table the Van der Waals table of constants
- edited
187     '''
188
189     b_hat_table.append({'substance': 'F2', 'b_hat': 0.02896},
ignore_index = True)
190     b_hat_table.append({'substance': 'HF', 'b_hat': 0.0739},
ignore_index = True)
191
192     b_hat_f2 = b_hat_table['b_hat'][3]
193     F2_inject_temp = Ox_inject_temp
194
195     del_f2_hat = f2_tableH(F2_inject_temp) + b_hat_f2*(
p_starting - 101325)/1000
196     del_hf_hat = abs(hf_tableH(298))
197     del_h_total = phi*del_h2_hat + del_f2_hat
198     del_h_total = del_h_total + del_hf_hat
199
200
201     r = phi - 1
202     q = 1
203
204     r_perc = r/(r+q)
205     q_perc = 1-r_perc
206
207     T_h2 = h2_tableT(del_h_total)
208     T_hf = hf_tableT(q_perc*del_h_total)
209
210     T = (T_hf + T_h2) / 2
211
212
213     return b_hat_table, T
214
215 def OxNozzlePerf(ThroatRadius, ExitRadius, gamma, FEE_T, R,
p_starting):

```



```

216     '''
217     This is a function that returns the performance of the
oxidizer nozzle
218     :param ThroatRadius:
219     :param ExitRadius:
220     :param gamma:
221     :param FEE_T:
222     :return:
223     '''
224
225     Area_ratio = (ExitRadius/ThroatRadius)**2
226     # change np.float64 values to a float
227     gamma = float(gamma)
228     FEE_T = float(FEE_T)
229     R = float(R)
230     R = R*1000 # convert to J/kg-K
231
232     M_e = MachSolve(gamma, Area_ratio, 5.0)
233     print('M_e = ', M_e)
234     T_e = FEE_T*((1+((gamma-1)/2)*M_e**2)**(-1))
235     v_e = M_e * np.sqrt(gamma * R * T_e)
236     p_e = p_starting * (1 + ((gamma - 1) / 2) * M_e ** 2) **
(-gamma / (gamma - 1))
237     ExitRadius = ExitRadius/39.37 # convert to meters
238     ThroatRadius = ThroatRadius/39.37 # convert to
239
240
241     m_dot = (p_starting * np.pi * (ThroatRadius)**2) / (np.
sqrt(FEE_T)) * \
242         np.sqrt(gamma/R)*((2/(gamma+1))**((gamma+1)/(2*(gamma
-1))))
243
244     Thrust = m_dot * v_e + p_e * np.pi * ExitRadius**2 -
p_starting * np.pi * ThroatRadius**2
245     c_T = Thrust / (p_starting * np.pi * ThroatRadius**2)
246     I_sp = Thrust/ (m_dot * 9.81)
247
248     return m_dot, v_e, p_e, Thrust, c_T, I_sp, T_e
249
250 def FCC(FEE_F):
251     '''
252     This is a function that returns the performance of the
flourine combustion chamber
253     :return:
254     '''

```

```

255
256
257     # The chemical equation is H2 +F2-> 2HF
258     molfrac_h2_react = 1
259     molfrac_f2_react = 1
260     molfrac_hf_react = 0
261     molfrac_h2_prod = 0
262     molfrac_f2_prod = 0
263     molfrac_hf_prod = 2
264
265
266     molfrac_h2_total = molfrac_h2_react - molfrac_h2_prod
267     molfrac_f2_total = molfrac_f2_react - molfrac_f2_prod
268     molfrac_hf_total = molfrac_hf_react - molfrac_hf_prod
269     sigma_nu = molfrac_hf_total + molfrac_h2_total +
molfrac_f2_total
270     # print('sigma_nu = ', sigma_nu)
271
272     # Gibbs free energy of formation of H2, F2, and HF
273     del_h_h2 = h2_tableH(FEE_F)
274     del_h_f2 = f2_tableH(FEE_F)
275     del_h_hf = hf_tableH(FEE_F)
276
277     s_h2 = h2_tableS(FEE_F)
278     s_f2 = f2_tableS(FEE_F)
279     s_hf = hf_tableS(FEE_F)
280
281     # Gibbs
282     del_g_h2 = del_h_h2 - FEE_F*s_h2
283     del_g_f2 = del_h_f2 - FEE_F*s_f2
284     del_g_hf = del_h_hf - FEE_F*s_hf
285
286     del_g_total = -del_g_f2 - del_g_h2 + del_g_hf
287     print('del_g_total = ', del_g_total)

```

A.3 Iterater.py

```

1 import numpy as np
2 import pandas as pd
3 from sympy import Eq, var, solve, diff
4 from scipy.optimize import fsolve
5 import matplotlib.pyplot as plt
6
7
8 def h2_tableH(Temp):

```

```

9      '''
10     :param Temp: temperature in K
11     :return h_hat: enthalpy in J/mol also known as heat of
formation
12     '''
13     h2_table = pd.read_csv('./tables/Hydrogen_H2_table.csv')
14
15     idx = (h2_table['T']-Temp).abs().idxmin()
16     T_cloest = h2_table['T'][idx]
17     h_hat_close = h2_table['H*-H*_298'][idx]
18     if Temp > T_cloest:
19         T_upper = h2_table['T'][idx+1]
20         h_hat_upper = h2_table['H*-H*_298'][idx+1]
21         h_hat = h_hat_close + (h_hat_upper-h_hat_close)/((
T_upper-T_cloest)*(Temp-T_cloest)
22     else:
23         T_lower = h2_table['T'][idx-1]
24         h_hat_lower = h2_table['H*-H*_298'][idx-1]
25         h_hat = h_hat_close + (h_hat_lower-h_hat_close)/((
T_lower-T_cloest)*(Temp-T_cloest)
26     h_hat = h_hat*4184
27
28     return h_hat
29
30 def f2_tableH(Temp):
31     '''
32     :param Temp: temperature in K
33     :return h_hat: enthalpy in J/mol also known as heat of
formation
34     '''
35     f2_table = pd.read_csv('./tables/Fluorine_F2_table.csv')
36
37     idx = (f2_table['T']-Temp).abs().idxmin()
38     T_cloest = f2_table['T'][idx]
39     h_hat_close = f2_table['H*-H*_298'][idx]
40     if Temp > T_cloest:
41         T_upper = f2_table['T'][idx+1]
42         h_hat_upper = f2_table['H*-H*_298'][idx+1]
43         h_hat = h_hat_close + (h_hat_upper-h_hat_close)/((
T_upper-T_cloest)*(Temp-T_cloest)
44     else:
45         T_lower = f2_table['T'][idx-1]
46         h_hat_lower = f2_table['H*-H*_298'][idx-1]
47         h_hat = h_hat_close + (h_hat_lower-h_hat_close)/((
T_lower-T_cloest)*(Temp-T_cloest)

```

```

48     h_hat = h_hat*4184
49
50     return h_hat
51
52 def o2_tableH(Temp):
53     '''
54     :param Temp: temperature in K
55     :return h_hat: enthalpy in J/mol also known as heat of
56     formation
57     '''
58     o2_table = pd.read_csv('./tables/o2_table.csv')
59
60     idx = (o2_table['T']-Temp).abs().idxmin()
61     T_cloest = o2_table['T'][idx]
62     h_hat_close = o2_table['H*-H*_298'][idx]
63     if Temp > T_cloest:
64         T_upper = o2_table['T'][idx+1]
65         h_hat_upper = o2_table['H*-H*_298'][idx+1]
66         h_hat = h_hat_close + (h_hat_upper-h_hat_close)/(
67         T_upper-T_cloest)*(Temp-T_cloest)
68     else:
69         T_lower = o2_table['T'][idx-1]
70         h_hat_lower = o2_table['H*-H*_298'][idx-1]
71         h_hat = h_hat_close + (h_hat_lower-h_hat_close)/(
72         T_lower-T_cloest)*(Temp-T_cloest)
73     h_hat = h_hat
74
75     return h_hat
76
77 def hf_tableH(Temp):
78     '''
79     :param Temp: temperature in K
80     :return h_hat: enthalpy in J/mol also known as heat of
81     formation
82     '''
83     hf_table = pd.read_csv('./tables/
84     HydrogenFluoride_HF_table.csv')
85
86     idx = (hf_table['T']-Temp).abs().idxmin()
87     T_cloest = hf_table['T'][idx]
88     h_hat_close = hf_table['h_hat'][idx]
89     if Temp > T_cloest:
90         T_upper = hf_table['T'][idx+1]
91         h_hat_upper = hf_table['h_hat'][idx+1]
92         h_hat = h_hat_close + (h_hat_upper-h_hat_close)/(

```

```

T_upper-T_cloest)*(Temp-T_cloest)
88     else:
89         T_lower = hf_table['T'][idx-1]
90         h_hat_lower = hf_table['h_hat'][idx-1]
91         h_hat = h_hat_close + (h_hat_lower-h_hat_close)/((
T_lower-T_cloest)*(Temp-T_cloest)
92         h_hat = h_hat*4184
93
94     return h_hat
95
96 def h2o_table_f(Temp):
97     '''
98     :param Temp: temperature in K
99     :return h_hat: enthalpy in J/mol also known as heat of
formation
100     '''
101     h2o_table = pd.read_csv('./tables/H2O_table.csv')
102
103     idx = (h2o_table['T']-Temp).abs().idxmin()
104     T_cloest = h2o_table['T'][idx]
105     h_hat_close = h2o_table['h_hat'][idx]
106     if Temp > T_cloest:
107         T_upper = h2o_table['T'][idx+1]
108         h_hat_upper = h2o_table['h_hat'][idx+1]
109         h_hat = h_hat_close + (h_hat_upper-h_hat_close)/((
T_upper-T_cloest)*(Temp-T_cloest)
110     else:
111         T_lower = h2o_table['T'][idx-1]
112         h_hat_lower = h2o_table['h_hat'][idx-1]
113         h_hat = h_hat_close + (h_hat_lower-h_hat_close)/((
T_lower-T_cloest)*(Temp-T_cloest)
114
115     return h_hat
116
117 def oh_table_f(Temp):
118     '''
119     :param Temp: temperature in K
120     :return h_hat: enthalpy in J/mol also known as heat of
formation
121     '''
122     oh_table = pd.read_csv('./tables/OH_table.csv')
123
124     idx = (oh_table['T']-Temp).abs().idxmin()
125     T_cloest = oh_table['T'][idx]
126     h_hat_close = oh_table['h_hat'][idx]

```

```

127     if Temp > T_cloest:
128         T_upper = oh_table['T'][idx+1]
129         h_hat_upper = oh_table['h_hat'][idx+1]
130         h_hat = h_hat_close + (h_hat_upper-h_hat_close)/((
T_upper-T_cloest)*(Temp-T_cloest)
131     else:
132         T_lower = oh_table['T'][idx-1]
133         h_hat_lower = oh_table['h_hat'][idx-1]
134         h_hat = h_hat_close + (h_hat_lower-h_hat_close)/((
T_lower-T_cloest)*(Temp-T_cloest)
135
136     return h_hat
137
138 def h_table_f(Temp):
139     '''
140     :param Temp: temperature in K
141     :return h_hat: enthalpy in J/mol also known as heat of
formation
142     '''
143     h_table = pd.read_csv('./tables/H_table.csv')
144
145     idx = (h_table['T']-Temp).abs().idxmin()
146     T_cloest = h_table['T'][idx]
147     h_hat_close = h_table['h_hat'][idx]
148     if Temp > T_cloest:
149         T_upper = h_table['T'][idx+1]
150         h_hat_upper = h_table['h_hat'][idx+1]
151         h_hat = h_hat_close + (h_hat_upper-h_hat_close)/((
T_upper-T_cloest)*(Temp-T_cloest)
152     else:
153         T_lower = h_table['T'][idx-1]
154         h_hat_lower = h_table['h_hat'][idx-1]
155         h_hat = h_hat_close + (h_hat_lower-h_hat_close)/((
T_lower-T_cloest)*(Temp-T_cloest)
156
157     return h_hat
158
159 def h2o_tableT(h_hat):
160     '''
161     :param h_hat: enthalpy in J/mol also known as heat of
formation
162     :return Temp: temperature in K
163     '''
164     h2o_table = pd.read_csv('./tables/H2O_table.csv')
165

```

```

166     idx = (h2o_table['h_f']-h_hat).abs().idxmin()
167     h_hat_cloest = h2o_table['h_f'][idx]
168     T_close = h2o_table['T'][idx]
169     if h_hat > h_hat_cloest:
170         h_hat_upper = h2o_table['h_f'][idx+1]
171         T_upper = h2o_table['T'][idx+1]
172         T = T_close + (T_upper-T_close)/(h_hat_upper -
h_hat_cloest)*(h_hat-h_hat_cloest)
173     else:
174         h_hat_lower = h2o_table['h_f'][idx-1]
175         T_lower = h2o_table['T'][idx-1]
176         T = T_close + (T_lower-T_close)/(h_hat_lower -
h_hat_cloest)*(h_hat-h_hat_cloest)
177
178     return T
179
180 def h2_tableT(h_hat):
181     '''
182     :param h_hat: enthalpy in J/mol also known as heat of
formation
183     :return Temp: temperature in K
184     '''
185     h2_table = pd.read_csv('./tables/Hydrogen_H2_table.csv')
186     h_hat = h_hat/4184
187     idx = (h2_table['H*-H*_298']-h_hat).abs().idxmin()
188     h_hat_cloest = h2_table['H*-H*_298'][idx]
189     T_close = h2_table['T'][idx]
190     if h_hat > h_hat_cloest:
191         h_hat_upper = h2_table['H*-H*_298'][idx+1]
192         T_upper = h2_table['T'][idx+1]
193         T = T_close + (T_upper-T_close)/(h_hat_upper -
h_hat_cloest)*(h_hat-h_hat_cloest)
194     else:
195         h_hat_lower = h2_table['H*-H*_298'][idx-1]
196         T_lower = h2_table['T'][idx-1]
197         T = T_close + (T_lower-T_close)/(h_hat_lower -
h_hat_cloest)*(h_hat-h_hat_cloest)
198
199     return T
200
201 def hf_tableT(h_hat):
202     '''
203     :param h_hat: enthalpy in J/mol also known as heat of
formation
204     :return Temp: temperature in K

```

```

205     '''
206     f2_table = pd.read_csv('./tables/
HydrogenFluoride_HF_table.csv')
207     h_hat = h_hat/4184
208     idx = (f2_table['H*-H*_298']-h_hat).abs().idxmin()
209     h_hat_closest = f2_table['H*-H*_298'][idx]
210     T_close = f2_table['T'][idx]
211     if h_hat > h_hat_closest:
212         h_hat_upper = f2_table['H*-H*_298'][idx+1]
213         T_upper = f2_table['T'][idx+1]
214         T = T_close + (T_upper-T_close)/(h_hat_upper-
h_hat_closest)*(h_hat-h_hat_closest)
215     else:
216         h_hat_lower = f2_table['H*-H*_298'][idx-1]
217         T_lower = f2_table['T'][idx-1]
218         T = T_close + (T_lower-T_close)/(h_hat_lower-
h_hat_closest)*(h_hat-h_hat_closest)
219
220     return T
221
222
223 def oh_table_g(Temp):
224     '''
225     :param Temp: temperature in K
226     :return g_hat: Gibbs free energy in J/mol
227     '''
228     oh_table = pd.read_csv('./tables/OH_table.csv')
229
230     idx = (oh_table['T']-Temp).abs().idxmin()
231     T_closest = oh_table['T'][idx]
232     g_hat_close = oh_table['g_hat'][idx]
233     if Temp > T_closest:
234         T_upper = oh_table['T'][idx+1]
235         g_hat_upper = oh_table['g_hat'][idx+1]
236         g_hat = g_hat_close + (g_hat_upper-g_hat_close)/(
T_upper-T_closest)*(Temp-T_closest)
237     else:
238         T_lower = oh_table['T'][idx-1]
239         g_hat_lower = oh_table['g_hat'][idx-1]
240         g_hat = g_hat_close + (g_hat_lower-g_hat_close)/(
T_lower-T_closest)*(Temp-T_closest)
241
242     return g_hat
243
244 def h_table_g(Temp):

```



```

245     '''
246     :param Temp: temperature in K
247     :return g_hat: Gibbs free energy in J/mol
248     '''
249     h_table = pd.read_csv('./tables/H_table.csv')
250
251     idx = (h_table['T']-Temp).abs().idxmin()
252     T_cloest = h_table['T'][idx]
253     g_hat_close = h_table['g_hat'][idx]
254     if Temp > T_cloest:
255         T_upper = h_table['T'][idx+1]
256         g_hat_upper = h_table['g_hat'][idx+1]
257         g_hat = g_hat_close + (g_hat_upper-g_hat_close)/(
258 T_upper-T_cloest)*(Temp-T_cloest)
259     else:
260         T_lower = h_table['T'][idx-1]
261         g_hat_lower = h_table['g_hat'][idx-1]
262         g_hat = g_hat_close + (g_hat_lower-g_hat_close)/(
263 T_lower-T_cloest)*(Temp-T_cloest)
264
265     return g_hat
266
267 def h2o_table_g(Temp):
268     '''
269     :param Temp: temperature in K
270     :return g_hat: Gibbs free energy in J/mol
271     '''
272     h2o_table = pd.read_csv('./tables/H2O_table.csv')
273
274     idx = (h2o_table['T']-Temp).abs().idxmin()
275     T_cloest = h2o_table['T'][idx]
276     g_hat_close = h2o_table['g_hat'][idx]
277     if Temp > T_cloest:
278         T_upper = h2o_table['T'][idx+1]
279         g_hat_upper = h2o_table['g_hat'][idx+1]
280         g_hat = g_hat_close + (g_hat_upper-g_hat_close)/(
281 T_upper-T_cloest)*(Temp-T_cloest)
282     else:
283         T_lower = h2o_table['T'][idx-1]
284         g_hat_lower = h2o_table['g_hat'][idx-1]
285         g_hat = g_hat_close + (g_hat_lower-g_hat_close)/(
286 T_lower-T_cloest)*(Temp-T_cloest)
287
288     return g_hat

```

```

286 def h2_table_g(Temp):
287     '''
288     :param Temp: temperature in K
289     :return g_hat: Gibbs free energy in J/mol
290     '''
291     h2_table = pd.read_csv('./tables/Hydrogen_H2_table.csv')
292
293     idx = (h2_table['T']-Temp).abs().idxmin()
294     T_cloest = h2_table['T'][idx]
295     g_hat_close = h2_table['g_hat'][idx]
296     if Temp > T_cloest:
297         T_upper = h2_table['T'][idx+1]
298         g_hat_upper = h2_table['g_hat'][idx+1]
299         g_hat = g_hat_close + (g_hat_upper-g_hat_close)/((
300 T_upper-T_cloest)*(Temp-T_cloest)
301     else:
302         T_lower = h2_table['T'][idx-1]
303         g_hat_lower = h2_table['g_hat'][idx-1]
304         g_hat = g_hat_close + (g_hat_lower-g_hat_close)/((
305 T_lower-T_cloest)*(Temp-T_cloest)
306
307     return g_hat
308
309 def h2o_table_cp(Temp):
310     '''
311     :param Temp: temperature in K
312     :return cp_hat: heat capacity in J/mol-K
313     '''
314     h2o_table = pd.read_csv('./tables/H2O_table.csv')
315
316     idx = (h2o_table['T']-Temp).abs().idxmin()
317     T_cloest = h2o_table['T'][idx]
318     cp_hat_close = h2o_table['cp_hat'][idx]
319     if Temp > T_cloest:
320         T_upper = h2o_table['T'][idx+1]
321         cp_hat_upper = h2o_table['cp_hat'][idx+1]
322         cp_hat = cp_hat_close + (cp_hat_upper-cp_hat_close)/((
323 T_upper-T_cloest)*(Temp-T_cloest)
324     else:
325         T_lower = h2o_table['T'][idx-1]
326         cp_hat_lower = h2o_table['cp_hat'][idx-1]
327         cp_hat = cp_hat_close + (cp_hat_lower-cp_hat_close)/((
328 T_lower-T_cloest)*(Temp-T_cloest)
329
330     return cp_hat

```

```

327
328 def h2_table_cp(Temp):
329     '''
330     :param Temp: temperature in K
331     :return cp_hat: heat capacity in J/mol-K
332     '''
333     h2_table = pd.read_csv('./tables/Hydrogen_H2_table.csv')
334
335     idx = (h2_table['T']-Temp).abs().idxmin()
336     T_cloest = h2_table['T'][idx]
337     cp_hat_close = h2_table['cp_hat'][idx]
338     if Temp > T_cloest:
339         T_upper = h2_table['T'][idx+1]
340         cp_hat_upper = h2_table['cp_hat'][idx+1]
341         cp_hat = cp_hat_close + (cp_hat_upper-cp_hat_close)/((
342         T_upper-T_cloest)*(Temp-T_cloest)
343     else:
344         T_lower = h2_table['T'][idx-1]
345         cp_hat_lower = h2_table['cp_hat'][idx-1]
346         cp_hat = cp_hat_close + (cp_hat_lower-cp_hat_close)/((
347         T_lower-T_cloest)*(Temp-T_cloest)
348
349     return cp_hat
350
351 def oh_table_cp(Temp):
352     '''
353     :param Temp: temperature in K
354     :return cp_hat: heat capacity in J/mol-K
355     '''
356     oh_table = pd.read_csv('./tables/OH_table.csv')
357
358     idx = (oh_table['T']-Temp).abs().idxmin()
359     T_cloest = oh_table['T'][idx]
360     cp_hat_close = oh_table['cp_hat'][idx]
361     if Temp > T_cloest:
362         T_upper = oh_table['T'][idx+1]
363         cp_hat_upper = oh_table['cp_hat'][idx+1]
364         cp_hat = cp_hat_close + (cp_hat_upper-cp_hat_close)/((
365         T_upper-T_cloest)*(Temp-T_cloest)
366     else:
367         T_lower = oh_table['T'][idx-1]
368         cp_hat_lower = oh_table['cp_hat'][idx-1]
369         cp_hat = cp_hat_close + (cp_hat_lower-cp_hat_close)/((
370         T_lower-T_cloest)*(Temp-T_cloest)

```

```

368     return cp_hat
369
370 def h_table_cp(Temp):
371     '''
372     :param Temp: temperature in K
373     :return cp_hat: heat capacity in J/mol-K
374     '''
375     h_table = pd.read_csv('./tables/H_table.csv')
376
377     idx = (h_table['T']-Temp).abs().idxmin()
378     T_cloest = h_table['T'][idx]
379     cp_hat_close = h_table['cp_hat'][idx]
380     if Temp > T_cloest:
381         T_upper = h_table['T'][idx+1]
382         cp_hat_upper = h_table['cp_hat'][idx+1]
383         cp_hat = cp_hat_close + (cp_hat_upper-cp_hat_close)/((
384         T_upper-T_cloest)*(Temp-T_cloest)
385     else:
386         T_lower = h_table['T'][idx-1]
387         cp_hat_lower = h_table['cp_hat'][idx-1]
388         cp_hat = cp_hat_close + (cp_hat_lower-cp_hat_close)/((
389         T_lower-T_cloest)*(Temp-T_cloest)
390
391     return cp_hat
392
393 def MachSolve(gamma, r_A, guess):
394     '''
395     Solve for Mach number given gamma, r_A, and guess
396     :param gamma: the gamma of the gas
397     :param r_A: the ratio of the areas
398     :param guess: the initial guess of the Mach number
399     :return: Mach number
400     '''
401     # M = var('M')
402     front_cont = ((gamma+1)/2)**(-(gamma+1)/(2*(gamma-1)))
403     print('front_cont = ', front_cont)
404     sec_cont = (gamma-1)/2
405     print('sec_cont = ', sec_cont)
406     func = lambda M: (((front_cont)*((1+sec_cont*M**2)**((
407     gamma+1)/(2*(gamma-1)))))/M) - r_A
408     sol = fsolve(func, guess)
409
410     return sol

```

```

410 def h2_tableS(Temp):
411     '''
412     :param Temp: temperature in K
413     :return S_hat: entropy in J/mol-K
414     '''
415     h2_table = pd.read_csv('./tables/Hydrogen_H2_table.csv')
416
417     idx = (h2_table['T']-Temp).abs().idxmin()
418     T_cloest = h2_table['T'][idx]
419     S_hat_close = h2_table['S'][idx]
420     if Temp > T_cloest:
421         T_upper = h2_table['T'][idx+1]
422         S_hat_upper = h2_table['S'][idx+1]
423         S_hat = S_hat_close + (S_hat_upper-S_hat_close)/((
424         T_upper-T_cloest)*(Temp-T_cloest)
425     else:
426         T_lower = h2_table['T'][idx-1]
427         S_hat_lower = h2_table['S'][idx-1]
428         S_hat = S_hat_close + (S_hat_lower-S_hat_close)/((
429         T_lower-T_cloest)*(Temp-T_cloest)
430
431     return S_hat
432
433 def hf_tableS(Temp):
434     '''
435     :param Temp: temperature in K
436     :return S_hat: entropy in J/mol-K
437     '''
438     hf_table = pd.read_csv('./tables/
439     HydrogenFluoride_HF_table.csv')
440
441     idx = (hf_table['T']-Temp).abs().idxmin()
442     T_cloest = hf_table['T'][idx]
443     S_hat_close = hf_table['S'][idx]
444     if Temp > T_cloest:
445         T_upper = hf_table['T'][idx+1]
446         S_hat_upper = hf_table['S'][idx+1]
447         S_hat = S_hat_close + (S_hat_upper-S_hat_close)/((
448         T_upper-T_cloest)*(Temp-T_cloest)
449     else:
450         T_lower = hf_table['T'][idx-1]
451         S_hat_lower = hf_table['S'][idx-1]
452         S_hat = S_hat_close + (S_hat_lower-S_hat_close)/((
453         T_lower-T_cloest)*(Temp-T_cloest)

```

```

450     return S_hat
451
452 def f2_tableS(Temp):
453     '''
454     :param Temp: temperature in K
455     :return S_hat: entropy in J/mol-K
456     '''
457     f2_table = pd.read_csv('./tables/Fluorine_F2_table.csv')
458
459     idx = (f2_table['T']-Temp).abs().idxmin()
460     T_closest = f2_table['T'][idx]
461     S_hat_close = f2_table['S'][idx]
462     if Temp > T_closest:
463         T_upper = f2_table['T'][idx+1]
464         S_hat_upper = f2_table['S'][idx+1]
465         S_hat = S_hat_close + (S_hat_upper-S_hat_close)/((
466         T_upper-T_closest)*(Temp-T_closest)
467     else:
468         T_lower = f2_table['T'][idx-1]
469         S_hat_lower = f2_table['S'][idx-1]
470         S_hat = S_hat_close + (S_hat_lower-S_hat_close)/((
471         T_lower-T_closest)*(Temp-T_closest)
472
473     return S_hat
474
475 # Me = MachSolve(1.237115, 77.5426, 5)
476 # print('Me = ', Me)

```