Space Shuttle Main Engine

November 4, 2022

Contents

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 stndard heat of formation by:

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

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

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.

$$\Delta\hat{h}_{mix} = \Phi\Delta\hat{h}_{H_2} + \Delta\hat{h}_{O_2} \tag{3}$$

We can then apply this to our chemical equation.

$$O_2 + \Phi H_2 \rightarrow (2-\Phi)H_2 + 2H_2O \tag{4}$$

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

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

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

$$r = \Phi - 1 \tag{6}$$

$$q = 1 \tag{7}$$

$$r_{percent} = \frac{r}{r+q} \tag{8}$$

$$q_{percent} = \frac{q}{r+q} \tag{9}$$

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

$$T_{H_2} = \texttt{h2\_tableT}(\Delta\hat{h}_r) \tag{10}$$

$$T_{H_2O} = \texttt{ho2\_tableT}(q_{percent}\Delta\hat{h}_r) \tag{11}$$

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

$$T_C = \frac{T_{H_2} + T_{H_2O}}{2} \tag{12}$$

2.2 Combustion Chamber

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

$$H_2O \rightarrow OH + H \tag{13}$$

$$H_2 \rightarrow 2H \tag{14}$$

The Gibbs Free Energy for each species is given by:

$$\hat{g}_{H_2O} = \texttt{h2o\_table\_g}(T_C) \tag{15}$$

$$\hat{g}_{H_2} = \texttt{h2\_table\_g}(T_C) \tag{16}$$

$$\hat{g}_{OH} = \texttt{oh\_table\_g}(T_C) \tag{17}$$

$$\hat{g}_{H} = \texttt{h\_table\_g}(T_C) \tag{18}$$

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

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

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

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:

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

Where;

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

$$K_p = e^{-\frac{\Delta G}{\hat{R}T}} \tag{23}$$

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

From here we can solve for the equilibrium constant K_n.

$$K_{H_2O} = (\frac{204atm}{1atm})^{-\frac{\Delta G_{H_2O}}{\hat{R}T_C}} \tag{25}$$

$$K_{H_2} = (\frac{204atm}{1atm})^{-\frac{\Delta G_{H_2}}{\hat{R}T_C}} \tag{26}$$

Now we move into the chemistry problem.
Atom Balance:

$$H_2 + lO_2 \rightarrow N_{H_2O} H_2O + N_{H_2} H_2 + N_H H + N_{OH} OH \tag{27}$$

Recall, due to the fule being Hydrogen;

$$mass_{fuel} = 2.02 \tag{28}$$

$$mass_{Oxidizer} = \frac{mass_{fuel}}{\phi} \tag{29}$$

$$l = \frac{mass_{Oxidizer}}{2\hat{m}_{O_2}} \tag{30}$$

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 $H_2O \rightarrow OH + H$:

$$H_2 : 2 = 2N_{H_2O} + 2N_{H_2} + N_H + N_{OH} \tag{31}$$

$$O_2 : 2l = N_{H_2O} + N_{OH} \tag{32}$$

$$1 = N_{H_2O} + N_{H_2} + \frac{1}{2}N_H + \frac{1}{2}N_{OH} \tag{33}$$

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

$$1 = N_{H_2O} + N_{H_2} \tag{34}$$

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

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

$$0.760592 = N_{H_2O} \tag{36}$$

$$X = \frac{N_{Molecule}}{N_{Total}} \tag{37}$$

$$X_{H_2O} = \frac{N_{H_2O}}{N_{Total}} \tag{38}$$

$$X_H = \frac{N_H}{N_{Total}} \tag{39}$$

$$X_{OH} = \frac{N_{OH}}{N_{Total}} \tag{40}$$

$$X_{H_2O} = \frac{N_{H_2}}{N_{H_2O} + N_{H_2} + N_H + N_{OH}} \tag{41}$$

$$X_H = \frac{N_H}{N_{H_2O} + N_{H_2} + N_H + N_{OH}} \tag{42}$$

$$X_{OH} = \frac{N_{OH}}{N_{H_2O} + N_{H_2} + N_H + N_{OH}} \tag{43}$$

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

$$K_n = \frac{\frac{N_{OH}}{N_{H_2O}+N_{H_2}} \frac{N_H}{N_{H_2O}+N_{H_2}}}{\frac{N_{H_2O}}{N_{H_2O}+N_{H_2}}} \tag{44}$$

Resulting in,

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

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

$$X_{H_2} = \frac{N_{H_2}}{Total} \tag{46}$$

$$X_H = \frac{N_H}{Total} \tag{47}$$

$$X_{H_2} = \frac{N_{H_2}}{N_{H_2} + N_{H_2O} + N_H + N_{OH}} \tag{48}$$

$$X_H = \frac{N_H}{N_{H_2} + N_{H_2O} + N_H + N_{OH}} \tag{49}$$

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

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

As a result we have 4 coupled equations:

$$1 = N_{H_2O} + N_{H_2} \tag{52}$$

$$0.760592 = N_{H_2O} \tag{53}$$

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

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

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

$$N_{H_2O} = 0.760592 \tag{56}$$

$$N_{H_2} = 0.239408 \tag{57}$$

$$N_H = 0.05333 \tag{58}$$

$$N_{OH} = 0.101958 \tag{59}$$

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.

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

$$c_p = Y_{H_2O}c_p + Y_{H_2}c_p + Y_Hc_p + Y_{OH}c_p \tag{61}$$

We perform now calulate the gas constant R.

$$R = \frac{\hat{R}}{m_{Total}} \tag{62}$$

$$c_v = c_p - R \tag{63}$$

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

2.3 Nozzle Performance

We begin by defining the Area Ratio:

$$A_r = \frac{\pi R_e^2}{\pi R_t^2} \tag{65}$$

Allowing for us to solve for the Mach number:

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

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.

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

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

$$v_e = M_e\sqrt{\gamma R T_e} \tag{69}$$

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

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

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.

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

$$I_{sp} = \frac{\mathcal{T}}{\dot{m}g_0} \tag{72}$$

$$C_T = \frac{\mathcal{T}}{p_c A_t} \tag{73}$$

A Code listings

A.1 SSMEMain.py

```
import pandas as pd
from exchanger import *


if __name__ == '__main__':

    p_starting = 204 #starting pressure in atm
    p_starting = p_starting * 101325 #convert to pascals
    FuelOxidizer_ratio = 0.166
    mol_O2 = 31.998 # g/mol #single molecule mass
    mol_H2 = 2.016 # g/mol #single molecule mass
    phi = (FuelOxidizer_ratio* mol_O2 / mol_H2)/2
    Hydr_inject_temp = 850 #K
    Ox_inject_temp = 530   #K
    ThroatRadius = 5.15 #inches
    ExitRadius = 45.35 #inches
    FEE_T, FEE_del_h_total, del_h2_hat, b_hat_table = FEE(
    Hydr_inject_temp, p_starting, Ox_inject_temp, phi)
    print('phi = ', phi)

    print('Temperature of the first enthalpy exchanger = ',
    FEE_T, 'K')
    print('Total enthalpy change in the first enthalpy
    exchanger = ', FEE_del_h_total, 'J')


    gamma, R, cp_total, cv, LawOfMassAction_df = OCC(FEE_T,
    p_starting, FuelOxidizer_ratio)
    print('gamma = ', gamma)

    m_dotOx, v_eOx, p_eOx, ThrustOx, c_TOx, I_spOx, T_e_Ox =
    OxNozzlePerf(ThroatRadius, ExitRadius, gamma, FEE_T, R,
    p_starting)

    print('Ox mass flow rate = ', m_dotOx, 'kg/s')
    print('Ox exit velocity = ', v_eOx, 'm/s')
    print('Ox exit pressure = ', p_eOx, 'Pa')
    print('Ox thrust = ', ThrustOx, 'N')
    print('Ox thrust coefficient = ', c_TOx)
    print('Ox specific impulse = ', I_spOx, 's')
    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,
        del_h2_hat, b_hat_table, Ox_inject_temp)
40
41      print('Temperature of the first enthalpy exchanger (
        Flourine) = ', FEE_T_flourine, 'K')
42
43      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
       first enthalpy exchanger
10     :param Hydr_inject_temp: temperature of hydrogen
       injection in K
11     :param p_starting: starting pressure in Pa
12     :param Ox_inject_temp: temperature of oxygen injection in
       K
13     :param phi: equivalence ratio
14     :return T the temperature of the first enthalpy exchanger
15     :return del_h_total the total enthalpy change in the
       first enthalpy exchanger
16     :return b_hat_table the table of b_hat values
17     :return del_h2_hat the enthalpy change of H2
18     '''
19
20
21     h_f_h2o = abs(h2o_table_f(298)) #J/mol
22     h_f_h = h_table_f(298)/1000
23     h_oh_f = oh_table_f(298)/1000  # KJ/mol
24
25     b_hat_table = pd.DataFrame(columns=['substance', 'b_hat'
       ])
26     b_hat_table = b_hat_table.append({'substance': 'H2', '
       b_hat': 0.0266}, ignore_index=True)
```

```
    b_hat_table = b_hat_table.append({'substance': 'H2O', 'b_hat': 0.03049}, ignore_index=True)
    b_hat_table = b_hat_table.append({'substance': 'O2', 'b_hat': 0.0318}, ignore_index=True)
    b_hat_table = b_hat_table.append({'substance': 'N2', 'b_hat': 0.0391}, ignore_index=True)

    b_hat_h2 = b_hat_table['b_hat'][0]
    b_hat_o2 = b_hat_table['b_hat'][2]

    del_h2_hat = h2_tableH(Hydr_inject_temp) + b_hat_h2*(p_starting - 101325)/1000
    del_o2_hat = o2_tableH(Ox_inject_temp) + b_hat_o2*(p_starting - 101325)/1000
    mols_h2 = phi*2
    mols_o2 = 1

    del_h_total = del_h2_hat*mols_h2 + del_o2_hat*mols_o2
    # At this point the chemical equation is O2 + mols*H2 -> (phi-2)*H2 +2H2O
    del_h_total = del_h_total + 2*h_f_h2o

    # weighting function
    r = 0.17947269916082
    q = (1- r)/2

    T_h2o = h2o_tableT(q*del_h_total)
    T_h2 = h2_tableT(r*del_h_total)

    T = (T_h2o + T_h2)/2

    return T, del_h_total, del_h2_hat, b_hat_table

def OCC(FEE_T, p_starting, FO_ratio):
    '''
    This is a function that returns the calculations of the oxygen combustion chamber
    :param FEE_T: temperature of the first enthalpy exchanger in K
    :param p_starting: starting pressure in Pa
    :param FO_ratio: fuel to oxidizer ratio
    :return gamma: the ratio of specific heats
    :return R: the gas constant
    :return cp_total: the total specific heat capacity
    :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/FO_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*
    N_oh)/((N_h2+N_h2o)*N_h2o), k_h2o),
121                  Eq((N_h**2)/((N_h2+N_h2o)*N_h2), k_h2)], [
    N_h, N_oh, N_h2, N_h2o])
122     # split sol into two separate lists
123     sol1 = sol[0]
124     sol2 = sol[1]
125
126     # if all the solutions are real and positive then assign
    mols_sol to the solution
127     if all(i > 0 for i in sol1) and all(i > 0 for i in sol2):
128         mols_sol = sol1
129     else:
130         mols_sol = sol2
131
132     # print('mols_sol = ', mols_sol)
133     N_h = mols_sol[0]
134     N_oh = mols_sol[1]
135     N_h2 = mols_sol[2]
136     N_h2o = mols_sol[3]
137     N_total = N_h + N_oh + N_h2 + N_h2o
138
139     molar_mass_h2o = 18.01528
140     molar_mass_h = 1.00794
141     molar_mass_oh = 17.00734
142     molar_mass_h2 = 2.01588
143     molar_mass_total = molar_mass_h2o + molar_mass_h +
    molar_mass_oh + molar_mass_h2
144
```

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

```
    This is a function that returns the calculations of the fuel enthalpy exchanger - flourine
    :param phi the equivalence ratio
    :param p_starting combustion pressure in Pa
    :param del_h2_hat heat of formation of H2 from FEE - Oxygen
    :param b_hat_table Van der Waals table of constants
    :param Ox_inject_temp: Oxidizer injection temperature in K
    :return T the temperature of the FEE in K
    :return b_hat_table the Van der Waals table of constants - edited
    '''

    b_hat_table.append({'substance': 'F2', 'b_hat': 0.02896}, ignore_index = True)
    b_hat_table.append({'substance': 'HF', 'b_hat': 0.0739}, ignore_index = True)

    b_hat_f2 = b_hat_table['b_hat'][3]
    F2_inject_temp = Ox_inject_temp

    del_f2_hat = f2_tableH(F2_inject_temp) + b_hat_f2*(p_starting - 101325)/1000
    del_hf_hat = abs(hf_tableH(298))
    del_h_total = phi*del_h2_hat + del_f2_hat
    del_h_total = del_h_total + del_hf_hat


    r = phi - 1
    q = 1

    r_perc = r/(r+q)
    q_perc = 1-r_perc

    T_h2 = h2_tableT(del_h_total)
    T_hf = hf_tableT(q_perc*del_h_total)

    T = (T_hf + T_h2) / 2


    return b_hat_table, T

def OxNozzlePerf(ThroatRadius, ExitRadius, gamma, FEE_T, R, p_starting):
```

```
    '''
    This is a function that returns the performance of the oxidizer nozzle
    :param ThroatRadius:
    :param ExitRadius:
    :param gamma:
    :param FEE_T:
    :return:
    '''

    Area_ratio = (ExitRadius/ThroatRadius)**2
    # change np.float64 values to a float
    gamma = float(gamma)
    FEE_T = float(FEE_T)
    R = float(R)
    R = R*1000 # convert to J/kg-K

    M_e = MachSolve(gamma, Area_ratio, 5.0)
    print('M_e = ', M_e)
    T_e = FEE_T*((1+((gamma-1)/2)*M_e**2)**(-1))
    v_e = M_e * np.sqrt(gamma * R * T_e)
    p_e = p_starting * (1 + ((gamma - 1) / 2) * M_e ** 2) ** (-gamma / (gamma - 1))
    ExitRadius = ExitRadius/39.37 # convert to meters
    ThroatRadius = ThroatRadius/39.37 # convert to


    m_dot = (p_starting * np.pi * (ThroatRadius)**2) / (np.sqrt(FEE_T)) * \
        np.sqrt(gamma/R)*((2/(gamma+1))**((gamma+1)/(2*(gamma-1))))

    Thurst = m_dot * v_e + p_e * np.pi * ExitRadius**2 - p_starting * np.pi * ThroatRadius**2
    c_T = Thurst / (p_starting * np.pi * ThroatRadius**2)
    I_sp = Thurst/ (m_dot * 9.81)

    return m_dot, v_e, p_e, Thurst, c_T, I_sp, T_e

def FCC(FEE_F):
    '''
    This is a function that returns the performance of the flourine combustion chamber
    :return:
    '''
```

```
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):
```

```
    '''
    :param Temp: temperature in K
    :return h_hat: enthalpy in J/mol also known as heat of formation
    '''
    h2_table = pd.read_csv('./tables/Hydrogen_H2_table.csv')

    idx = (h2_table['T']-Temp).abs().idxmin()
    T_closest = h2_table['T'][idx]
    h_hat_close = h2_table['H*-H*_298'][idx]
    if Temp > T_closest:
        T_upper = h2_table['T'][idx+1]
        h_hat_upper = h2_table['H*-H*_298'][idx+1]
        h_hat = h_hat_close + (h_hat_upper-h_hat_close)/(T_upper-T_closest)*(Temp-T_closest)
    else:
        T_lower = h2_table['T'][idx-1]
        h_hat_lower = h2_table['H*-H*_298'][idx-1]
        h_hat = h_hat_close + (h_hat_lower-h_hat_close)/(T_lower-T_closest)*(Temp-T_closest)
    h_hat = h_hat*4184

    return h_hat

def f2_tableH(Temp):
    '''
    :param Temp: temperature in K
    :return h_hat: enthalpy in J/mol also known as heat of formation
    '''
    f2_table = pd.read_csv('./tables/Fluorine_F2_table.csv')

    idx = (f2_table['T']-Temp).abs().idxmin()
    T_closest = f2_table['T'][idx]
    h_hat_close = f2_table['H*-H*_298'][idx]
    if Temp > T_closest:
        T_upper = f2_table['T'][idx+1]
        h_hat_upper = f2_table['H*-H*_298'][idx+1]
        h_hat = h_hat_close + (h_hat_upper-h_hat_close)/(T_upper-T_closest)*(Temp-T_closest)
    else:
        T_lower = f2_table['T'][idx-1]
        h_hat_lower = f2_table['H*-H*_298'][idx-1]
        h_hat = h_hat_close + (h_hat_lower-h_hat_close)/(T_lower-T_closest)*(Temp-T_closest)
```

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

```
    T_upper-T_cloest)*(Temp-T_cloest)
    else:
        T_lower = hf_table['T'][idx-1]
        h_hat_lower = hf_table['h_hat'][idx-1]
        h_hat = h_hat_close + (h_hat_lower-h_hat_close)/(
    T_lower-T_cloest)*(Temp-T_cloest)
    h_hat = h_hat*4184

    return h_hat

def h2o_table_f(Temp):
    '''
    :param Temp: temperature in K
    :return h_hat: enthalpy in J/mol also known as heat of
    formation
    '''
    h2o_table = pd.read_csv('./tables/H2O_table.csv')

    idx = (h2o_table['T']-Temp).abs().idxmin()
    T_cloest = h2o_table['T'][idx]
    h_hat_close = h2o_table['h_hat'][idx]
    if Temp > T_cloest:
        T_upper = h2o_table['T'][idx+1]
        h_hat_upper = h2o_table['h_hat'][idx+1]
        h_hat = h_hat_close + (h_hat_upper-h_hat_close)/(
    T_upper-T_cloest)*(Temp-T_cloest)
    else:
        T_lower = h2o_table['T'][idx-1]
        h_hat_lower = h2o_table['h_hat'][idx-1]
        h_hat = h_hat_close + (h_hat_lower-h_hat_close)/(
    T_lower-T_cloest)*(Temp-T_cloest)

    return h_hat

def oh_table_f(Temp):
    '''
    :param Temp: temperature in K
    :return h_hat: enthalpy in J/mol also known as heat of
    formation
    '''
    oh_table = pd.read_csv('./tables/OH_table.csv')

    idx = (oh_table['T']-Temp).abs().idxmin()
    T_cloest = oh_table['T'][idx]
    h_hat_close = oh_table['h_hat'][idx]
```

```
    if Temp > T_cloest:
        T_upper = oh_table['T'][idx+1]
        h_hat_upper = oh_table['h_hat'][idx+1]
        h_hat = h_hat_close + (h_hat_upper-h_hat_close)/(T_upper-T_cloest)*(Temp-T_cloest)
    else:
        T_lower = oh_table['T'][idx-1]
        h_hat_lower = oh_table['h_hat'][idx-1]
        h_hat = h_hat_close + (h_hat_lower-h_hat_close)/(T_lower-T_cloest)*(Temp-T_cloest)

    return h_hat

def h_table_f(Temp):
    '''
    :param Temp: temperature in K
    :return h_hat: enthalpy in J/mol also known as heat of formation
    '''
    h_table = pd.read_csv('./tables/H_table.csv')

    idx = (h_table['T']-Temp).abs().idxmin()
    T_cloest = h_table['T'][idx]
    h_hat_close = h_table['h_hat'][idx]
    if Temp > T_cloest:
        T_upper = h_table['T'][idx+1]
        h_hat_upper = h_table['h_hat'][idx+1]
        h_hat = h_hat_close + (h_hat_upper-h_hat_close)/(T_upper-T_cloest)*(Temp-T_cloest)
    else:
        T_lower = h_table['T'][idx-1]
        h_hat_lower = h_table['h_hat'][idx-1]
        h_hat = h_hat_close + (h_hat_lower-h_hat_close)/(T_lower-T_cloest)*(Temp-T_cloest)

    return h_hat

def h2o_tableT(h_hat):
    '''
    :param h_hat: enthalpy in J/mol also known as heat of formation
    :return Temp: temperature in K
    '''
    h2o_table = pd.read_csv('./tables/H2O_table.csv')

```

```
    idx = (h2o_table['h_f']-h_hat).abs().idxmin()
    h_hat_cloest = h2o_table['h_f'][idx]
    T_close = h2o_table['T'][idx]
    if h_hat > h_hat_cloest:
        h_hat_upper = h2o_table['h_f'][idx+1]
        T_upper = h2o_table['T'][idx+1]
        T = T_close + (T_upper-T_close)/(h_hat_upper-h_hat_cloest)*(h_hat-h_hat_cloest)
    else:
        h_hat_lower = h2o_table['h_f'][idx-1]
        T_lower = h2o_table['T'][idx-1]
        T = T_close + (T_lower-T_close)/(h_hat_lower-h_hat_cloest)*(h_hat-h_hat_cloest)

    return T

def h2_tableT(h_hat):
    '''
    :param h_hat: enthalpy in J/mol also known as heat of formation
    :return Temp: temperature in K
    '''
    h2_table = pd.read_csv('./tables/Hydrogen_H2_table.csv')
    h_hat = h_hat/4184
    idx = (h2_table['H*-H*_298']-h_hat).abs().idxmin()
    h_hat_cloest = h2_table['H*-H*_298'][idx]
    T_close = h2_table['T'][idx]
    if h_hat > h_hat_cloest:
        h_hat_upper = h2_table['H*-H*_298'][idx+1]
        T_upper = h2_table['T'][idx+1]
        T = T_close + (T_upper-T_close)/(h_hat_upper-h_hat_cloest)*(h_hat-h_hat_cloest)
    else:
        h_hat_lower = h2_table['H*-H*_298'][idx-1]
        T_lower = h2_table['T'][idx-1]
        T = T_close + (T_lower-T_close)/(h_hat_lower-h_hat_cloest)*(h_hat-h_hat_cloest)

    return T

def hf_tableT(h_hat):
    '''
    :param h_hat: enthalpy in J/mol also known as heat of formation
    :return Temp: temperature in K
```

```
    '''
    f2_table = pd.read_csv('./tables/HydrogenFluoride_HF_table.csv')
    h_hat = h_hat/4184
    idx = (f2_table['H*-H*_298']-h_hat).abs().idxmin()
    h_hat_closest = f2_table['H*-H*_298'][idx]
    T_close = f2_table['T'][idx]
    if h_hat > h_hat_closest:
        h_hat_upper = f2_table['H*-H*_298'][idx+1]
        T_upper = f2_table['T'][idx+1]
        T = T_close + (T_upper-T_close)/(h_hat_upper-h_hat_closest)*(h_hat-h_hat_closest)
    else:
        h_hat_lower = f2_table['H*-H*_298'][idx-1]
        T_lower = f2_table['T'][idx-1]
        T = T_close + (T_lower-T_close)/(h_hat_lower-h_hat_closest)*(h_hat-h_hat_closest)

    return T


def oh_table_g(Temp):
    '''
    :param Temp: temperature in K
    :return g_hat: Gibbs free energy in J/mol
    '''
    oh_table = pd.read_csv('./tables/OH_table.csv')

    idx = (oh_table['T']-Temp).abs().idxmin()
    T_closest = oh_table['T'][idx]
    g_hat_close = oh_table['g_hat'][idx]
    if Temp > T_closest:
        T_upper = oh_table['T'][idx+1]
        g_hat_upper = oh_table['g_hat'][idx+1]
        g_hat = g_hat_close + (g_hat_upper-g_hat_close)/(T_upper-T_closest)*(Temp-T_closest)
    else:
        T_lower = oh_table['T'][idx-1]
        g_hat_lower = oh_table['g_hat'][idx-1]
        g_hat = g_hat_close + (g_hat_lower-g_hat_close)/(T_lower-T_closest)*(Temp-T_closest)

    return g_hat

def h_table_g(Temp):
```

```
    '''
    :param Temp: temperature in K
    :return g_hat: Gibbs free energy in J/mol
    '''
    h_table = pd.read_csv('./tables/H_table.csv')

    idx = (h_table['T']-Temp).abs().idxmin()
    T_closest = h_table['T'][idx]
    g_hat_close = h_table['g_hat'][idx]
    if Temp > T_closest:
        T_upper = h_table['T'][idx+1]
        g_hat_upper = h_table['g_hat'][idx+1]
        g_hat = g_hat_close + (g_hat_upper-g_hat_close)/(T_upper-T_closest)*(Temp-T_closest)
    else:
        T_lower = h_table['T'][idx-1]
        g_hat_lower = h_table['g_hat'][idx-1]
        g_hat = g_hat_close + (g_hat_lower-g_hat_close)/(T_lower-T_closest)*(Temp-T_closest)

    return g_hat

def h2o_table_g(Temp):
    '''
    :param Temp: temperature in K
    :return g_hat: Gibbs free energy in J/mol
    '''
    h2o_table = pd.read_csv('./tables/H2O_table.csv')

    idx = (h2o_table['T']-Temp).abs().idxmin()
    T_closest = h2o_table['T'][idx]
    g_hat_close = h2o_table['g_hat'][idx]
    if Temp > T_closest:
        T_upper = h2o_table['T'][idx+1]
        g_hat_upper = h2o_table['g_hat'][idx+1]
        g_hat = g_hat_close + (g_hat_upper-g_hat_close)/(T_upper-T_closest)*(Temp-T_closest)
    else:
        T_lower = h2o_table['T'][idx-1]
        g_hat_lower = h2o_table['g_hat'][idx-1]
        g_hat = g_hat_close + (g_hat_lower-g_hat_close)/(T_lower-T_closest)*(Temp-T_closest)

    return g_hat

```

```
def h2_table_g(Temp):
    '''
    :param Temp: temperature in K
    :return g_hat: Gibbs free energy in J/mol
    '''
    h2_table = pd.read_csv('./tables/Hydrogen_H2_table.csv')

    idx = (h2_table['T']-Temp).abs().idxmin()
    T_closest = h2_table['T'][idx]
    g_hat_close = h2_table['g_hat'][idx]
    if Temp > T_closest:
        T_upper = h2_table['T'][idx+1]
        g_hat_upper = h2_table['g_hat'][idx+1]
        g_hat = g_hat_close + (g_hat_upper-g_hat_close)/(T_upper-T_closest)*(Temp-T_closest)
    else:
        T_lower = h2_table['T'][idx-1]
        g_hat_lower = h2_table['g_hat'][idx-1]
        g_hat = g_hat_close + (g_hat_lower-g_hat_close)/(T_lower-T_closest)*(Temp-T_closest)

    return g_hat

def h2o_table_cp(Temp):
    '''
    :param Temp: temperature in K
    :return cp_hat: heat capacity in J/mol-K
    '''
    h2o_table = pd.read_csv('./tables/H2O_table.csv')

    idx = (h2o_table['T']-Temp).abs().idxmin()
    T_closest = h2o_table['T'][idx]
    cp_hat_close = h2o_table['cp_hat'][idx]
    if Temp > T_closest:
        T_upper = h2o_table['T'][idx+1]
        cp_hat_upper = h2o_table['cp_hat'][idx+1]
        cp_hat = cp_hat_close + (cp_hat_upper-cp_hat_close)/(T_upper-T_closest)*(Temp-T_closest)
    else:
        T_lower = h2o_table['T'][idx-1]
        cp_hat_lower = h2o_table['cp_hat'][idx-1]
        cp_hat = cp_hat_close + (cp_hat_lower-cp_hat_close)/(T_lower-T_closest)*(Temp-T_closest)

    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)/(
    T_upper-T_cloest)*(Temp-T_cloest)
342     else:
343         T_lower = h2_table['T'][idx-1]
344         cp_hat_lower = h2_table['cp_hat'][idx-1]
345         cp_hat = cp_hat_close + (cp_hat_lower-cp_hat_close)/(
    T_lower-T_cloest)*(Temp-T_cloest)
346
347     return cp_hat
348
349 def oh_table_cp(Temp):
350     '''
351     :param Temp: temperature in K
352     :return cp_hat: heat capacity in J/mol-K
353     '''
354     oh_table = pd.read_csv('./tables/OH_table.csv')
355
356     idx = (oh_table['T']-Temp).abs().idxmin()
357     T_cloest = oh_table['T'][idx]
358     cp_hat_close = oh_table['cp_hat'][idx]
359     if Temp > T_cloest:
360         T_upper = oh_table['T'][idx+1]
361         cp_hat_upper = oh_table['cp_hat'][idx+1]
362         cp_hat = cp_hat_close + (cp_hat_upper-cp_hat_close)/(
    T_upper-T_cloest)*(Temp-T_cloest)
363     else:
364         T_lower = oh_table['T'][idx-1]
365         cp_hat_lower = oh_table['cp_hat'][idx-1]
366         cp_hat = cp_hat_close + (cp_hat_lower-cp_hat_close)/(
    T_lower-T_cloest)*(Temp-T_cloest)
367
```

```
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_closest = h_table['T'][idx]
379     cp_hat_close = h_table['cp_hat'][idx]
380     if Temp > T_closest:
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)/(
    T_upper-T_closest)*(Temp-T_closest)
384     else:
385         T_lower = h_table['T'][idx-1]
386         cp_hat_lower = h_table['cp_hat'][idx-1]
387         cp_hat = cp_hat_close + (cp_hat_lower-cp_hat_close)/(
    T_lower-T_closest)*(Temp-T_closest)
388
389     return cp_hat
390
391 def MachSolve(gamma, r_A, guess):
392     '''
393     Solve for Mach number given gamma, r_A, and guess
394     :param gamma: the gamma of the gas
395     :param r_A: the ratio of the areas
396     :param guess: the initial guess of the Mach number
397     :return: Mach number
398     '''
399
400     # M = var('M')
401     front_cont = ((gamma+1)/2)**(-(gamma+1)/(2*(gamma-1)))
402     print('front_cont = ', front_cont)
403     sec_cont = (gamma-1)/2
404     print('sec_cont = ', sec_cont)
405     func = lambda M: (((front_cont)*((1+sec_cont*M**2)**((
    gamma+1)/(2*(gamma-1)))))/M) - r_A
406     sol = fsolve(func, guess)
407
408     return sol
409
```

```
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)/(
    T_upper-T_cloest)*(Temp-T_cloest)
424     else:
425         T_lower = h2_table['T'][idx-1]
426         S_hat_lower = h2_table['S'][idx-1]
427         S_hat = S_hat_close + (S_hat_lower-S_hat_close)/(
    T_lower-T_cloest)*(Temp-T_cloest)
428 
429     return S_hat
430 
431 def hf_tableS(Temp):
432     '''
433     :param Temp: temperature in K
434     :return S_hat: entropy in J/mol-K
435     '''
436     hf_table = pd.read_csv('./tables/
    HydrogenFluoride_HF_table.csv')
437 
438     idx = (hf_table['T']-Temp).abs().idxmin()
439     T_cloest = hf_table['T'][idx]
440     S_hat_close = hf_table['S'][idx]
441     if Temp > T_cloest:
442         T_upper = hf_table['T'][idx+1]
443         S_hat_upper = hf_table['S'][idx+1]
444         S_hat = S_hat_close + (S_hat_upper-S_hat_close)/(
    T_upper-T_cloest)*(Temp-T_cloest)
445     else:
446         T_lower = hf_table['T'][idx-1]
447         S_hat_lower = hf_table['S'][idx-1]
448         S_hat = S_hat_close + (S_hat_lower-S_hat_close)/(
    T_lower-T_cloest)*(Temp-T_cloest)
449 
```

```
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)/(
    T_upper-T_closest)*(Temp-T_closest)
466     else:
467         T_lower = f2_table['T'][idx-1]
468         S_hat_lower = f2_table['S'][idx-1]
469         S_hat = S_hat_close + (S_hat_lower-S_hat_close)/(
    T_lower-T_closest)*(Temp-T_closest)
470 
471     return S_hat
472 # Me = MachSolve(1.237115, 77.5426, 5)
473 # print('Me = ', Me)
```