MECH4450 Aerospace Propulsion

Major Assignment - Part 1

Muirhead, Alex
s4435062

Watt, Robert
s4431151

May 10, 2019

The following finite rate chemistry mechanisms provide an accurate model of ethene combustion within a scramjet engine.

$$C_2H_4 + 2\,O_2 \longrightarrow 2\,CO + 2\,H_2O \tag{1}$$

$$CO + \frac{1}{2}\,O_2 \longleftrightarrow CO_2 \tag{2}$$

Information regarding the compositions of each reaction are stored as arrays (see Listing. 1), with the form of rows representing compounds and columns representing reaction. This method of information storage relies on each species being represented once within a single reaction which, while sufficient for this problem, requires improvement. Species are listed in the order of appearance; `C2H4, O2, CO, H2O, CO2`.

```
46  ν = np.array([
47      [-1.,  0. ],
48      [-2., -0.5],
49      [ 2., -1. ],
50      [ 2.,  0. ],
51      [ 0.,  1. ]
52  ]).T
```

```
56  νExp = np.array([
57      [0.5 , 0. ],
58      [0.65, 0.5],
59      [2.  , 1. ],
60      [2.  , 0. ],
61      [0.  , 1. ]
62  ]).T
```

Listing 1: Storage of reaction information

Stoichiometric coefficients are stored in ν, while experimentally derived rate coefficients are stored in νExp. The signs of stoichiometric coefficients are used to determine whether the respective species is a reactant or product of the reaction formulation. These signs are pre-computed, and stored in masks.

```
65  maskF = np.zeros_like(ν, dtype=bool)
66  maskR = np.zeros_like(ν, dtype=bool)
67  maskF[ν < 0.] = True
68  maskR[ν > 0.] = True
```

Listing 2: Reactant (`maskF`) and Product (`maskR`) species masks

The formation rates in kmol/(m^3 s) for the individual species are given as:

$$\frac{d[C_2H_4]}{dt} = -k_{1,f}[C_2H_4]^{0.5}[O_2]^{0.65} \tag{3a}$$

$$\frac{d[O_2]}{dt} = -2k_{1,f}[C_2H_4]^{0.5}[O_2]^{0.65} - \frac{1}{2}k_{2,f}[CO][O_2]^{0.5} + \frac{1}{2}k_{2,r}[CO_2] \tag{3b}$$

$$\frac{d[CO]}{dt} = 2k_{1,f}[C_2H_4]^{0.5}[O_2]^{0.65} - k_{2,f}[CO][O_2]^{0.5} + k_{2,r}[CO_2] \tag{3c}$$

$$\frac{d[H_2O]}{dt} = 2k_{1,f}[C_2H_4]^{0.5}[O_2]^{0.65} \tag{3d}$$

$$\frac{d[CO_2]}{dt} = k_{2,f}[CO][O_2]^{0.5} - k_{2,r}[CO_2] \tag{3e}$$

where $k_{1,f}$ is the forward rate of reaction for Eq. 1, and $k_{2,f}$ and $k_{2,r}$ are the forward and backward reaction rates for Eq. 2 respectively. As the first reaction

is uni-directional, the reverse reaction rate $k_{1,r}$ is set to 0. Forward reaction rates are calculated from the Arrhenius equation, using experimental values for the pre-exponential constant A and activation energy E_a.

$$k_{i,f} = A_i \exp\left(-\frac{E_{a,i}}{R_u T}\right) \tag{4}$$

Values for these constants are given in SI units of kmol, m and s in Table. 1.

	A	E_a
Eq. 1	1.739×10^9	1.485×10^5
Eq. 2	6.324×10^7	5.021×10^4

Table 1: Arrhenius equation constants

The reverse reaction rates are calculated using the equilibrium concentration coefficient K_c. This is related to the Gibbs free energy through the equilibrium pressure coefficient K_p.

$$K_p = \exp\left(-\frac{\Delta G^\circ(T)}{R_u T}\right) = K_c \left(\frac{p}{p^\circ}\right)^{\sum \nu} \tag{5}$$

The total change in Gibbs free energy is calculated as in Eq. 6, noting that the *signed* stoichiometric coefficients are used.

$$\Delta G^\circ(T) = \sum_i \nu_i \bar{g}^\circ_{f,i}(T) \tag{6}$$

$$K_p = \exp\left(-\frac{\sum_i \nu_i \bar{g}^\circ_{f,i}}{R_u T}\right) \tag{7}$$

This allows for a reduction in computation complexity, by calculating $K_{p,i}$ values for each individual species only once. Then the equilibrium concentration coefficient K_c can be calculated as

$$K_c = \exp\left(-\frac{\sum_i \nu_i \bar{g}^\circ_{f,i}}{R_u T}\right)\left(\frac{p^\circ}{p}\right)^{\sum_i \nu_i} = \prod_i \left[\exp\left(\ln\, K_{p,i}\right)\frac{p^\circ}{p}\right]^{\nu_i} \tag{8}$$

$$2.303 \log K_{p,i} = \ln\, K_{p,i} = -\frac{\bar{g}^\circ_{f,i}}{R_u T} \tag{9}$$

Values of $\log K_{p,i}$ are tabulated from The National Institute of Standards and Technology (1998), and interpolated for the current temperature. Storing the natural log of the value assists in quadratic interpolation, which is considered faster than attempting to use a spline interpolation method over the $k_{p,i}$ values themselves. This is implemented in a vectorised form shown in Listing 3, to improve efficiency.

```
84  def Kc(T, p):
85      Kf_i    = np.array([pow(10, Kf(T)) for Kf in logKfuncs]) * (pRef/p)
86      forward = pow(Kf_i, maskF*νExp)
87      reverse = pow(Kf_i, maskR*νExp)
88      return np.prod(reverse, axis=1) / np.prod(forward, axis=1)
```

Listing 3: Calculation of equilibrium concentration coefficients

Note that the code uses the *unsigned* experimental coefficients, such as to explicitly match the ratio of rate coefficients. However, the definiton from Gibbs free energy is based off stoichiometric coefficients. As these two sets of values are equal for the case of the second reaction, this is no concern for the specific case. If $\nu \neq \nu\text{Exp}$ for a reaction requiring calculation of reverse reaction rates, it is predicted that using the νExp values will provide the correct response.

The reverse reaction rate can then be calculated from the expression

$$K_c = \frac{k_f}{k_r} \quad \longrightarrow \quad k_r = \frac{k_f}{K_c} \tag{10}$$

where K_c is guaranteed to be a non-zero positive number. The rates of each species formation can then be expressed more generally as

$$\frac{\mathrm{d}[X_i]}{\mathrm{d}t} = k_f \prod_{i \in \text{reac}} \nu_i [X_i]^{\nu\text{Exp}_i} - k_r \prod_{i \in \text{prod}} \nu_i [X_i]^{\nu\text{Exp}_i} \tag{11}$$

The volumetric heat released per second from this reaction can be expressed as Eq. 12, with values of $\Delta h^\circ_{f,i}(T)$ tabulated from The National Institute of Standards and Technology (1998).

$$\dot{Q}''' = \sum_i \Delta h^\circ_{f,i}(T) \frac{\mathrm{d}[X_i]}{\mathrm{d}t} \tag{12}$$

Using the `maskF` and `maskR` arrays, both these calculations are implemented in a vectorised form as

```
115     ...
116     forward = kf * np.prod(pow(χ, maskF*νExp), axis=1)
117     reverse = kr * np.prod(pow(χ, maskR*νExp), axis=1)
118     χGrad   = ν.T @ forward - ν.T @ reverse
119     ...
120     hGrad = -sum([dχ_i*h_i(T) for dχ_i, h_i in zip(χGrad, deltaHfuncs)])
```

Listing 4: Calculation of species rates of formation

where the array of concentrations is represented by χ. The full code is presented in the Appendix.

With an stoichiometric air-fuel ration, the molar fractions for C_2H_4 and O_2 can be calculated using Eq. 13. With an initial pressure and temperature of $p_0 = 70\,\text{kPa}$ and $T_0 = 1400\,\text{K}$, these equate to

$$
\begin{aligned}
n_{\text{init}} &= n_F + n_{O_2} + n_{N_2} = 1 + 3 \times (1 + 3.76) \\
[C_2H_4] &= \frac{n_F}{n_T} \frac{p_0}{R_u T_0} = 6.5445 \times 10^{-5}\,\text{kmol/m}^3 \\
[O_2] &= \frac{n_{O_2}}{n_T} \frac{p_0}{R_u T_0} = 1.9633 \times 10^{-4}\,\text{kmol/m}^3
\end{aligned}
\tag{13}
$$

Starting with these concentrations, and a reference enthalpy of $0\,\text{kJ/(kg K)}$, the rates of species formation and heat release were integrated over 0.1 ms using the Python `scipy.intergrate.solve_ivp` integrator with the `LSODA` method. The resulting graphs of concentration, rate of heat release, and net heat released are given in Fig. 1, 2 and 3 respectively.

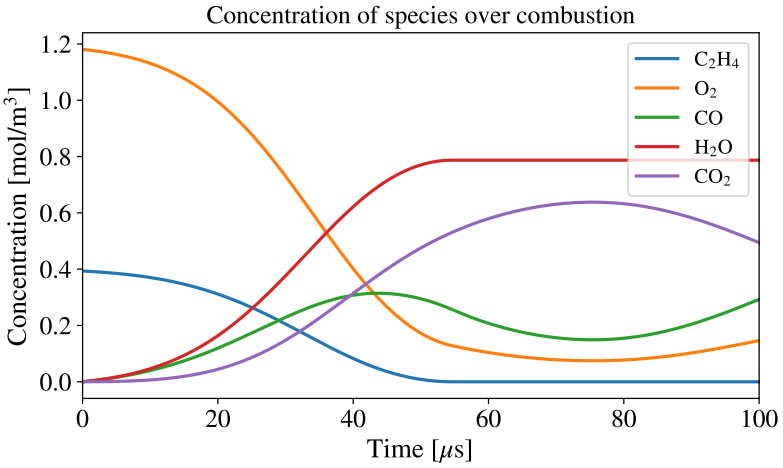

Figure 1: Concentration of species over combustion

It should be noted that as the solution progressed, it was necessary to clip values that tended below zero due to machine precision, to prevent complex valued non-physical solutions from being created. This leads to a discontinuity in the reaction rate when C_2H_4 is fully combusted, which is not noticeable in the species concentration solution, however can be seen in the rate of heat release (Fig. 2).

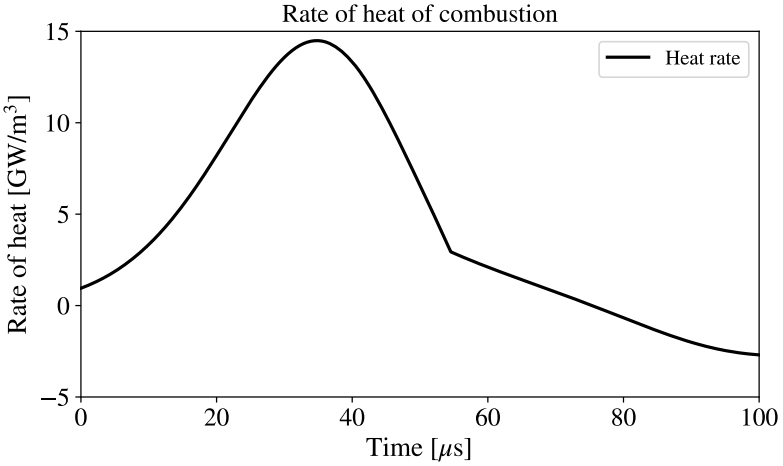

Figure 2: Rate of heat of combustion

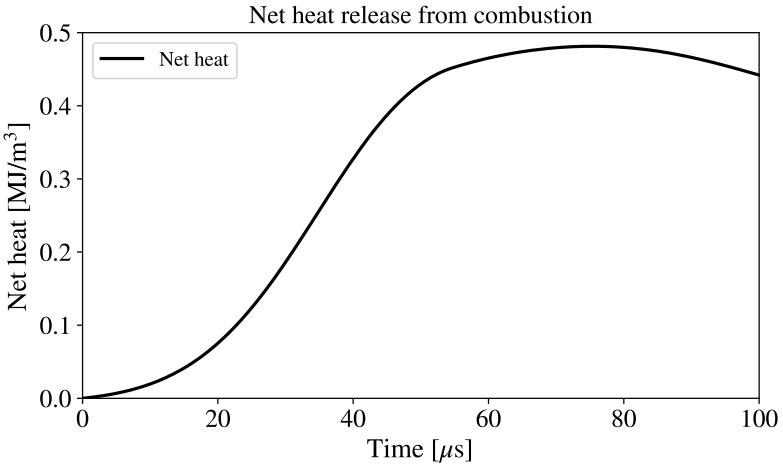

Figure 3: Net heat of combustion

References

The National Institute of Standards and Technology (1998). *NIST-JANAF Thermochemical Tables*. URL: https://janaf.nist.gov/.

Appendix

```python
import math

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy import integrate
from scipy import interpolate

Ru   = 8.314  # kJ/kmol.K
pRef = 100    # kPa

plt.style.use("PaperDoubleFig.mplstyle")


def lin(lower, upper, deltaX):
    deltaY = upper - lower
    grad = deltaY / deltaX

    def inner(x):
        return lower + grad*x

    return inner


def vectorInterface(lengths):
    L = [0, *np.cumsum(lengths)]

    def wrapper(func):
        def inner(t, args):
            splitArgs = [args[l:r] for l, r in zip(L[:-1], L[1:])]
            output = func(t, *splitArgs)
            return np.hstack([*output])
        return inner
    return wrapper


#  (row) species 0 :: C2H4
#                1 ::   O2
#                2 ::   CO
#                3 ::  H2O
#                4 ::  CO2

# (col) reaction 0 :: C2H4 + 2 O2 --> 2 CO + 2 H2O
#                1 :: CO + 1/2 O2 <-> CO2

# Stoichiometric coefficients
ν = np.array([
    [-1.,  0. ],
    [-2., -0.5],
    [ 2., -1. ],
    [ 2.,  0. ],
    [ 0.,  1. ]
]).T

# Experimental partial powers
νExp = np.array([
    [0.5 , 0. ],
    [0.65, 0.5],
    [2.  , 1. ],
```

```
    [2.  , 0. ],
    [0.  , 1. ]
]).T

# Forward and reverse masks
maskF = np.zeros_like(ν, dtype=bool)
maskR = np.zeros_like(ν, dtype=bool)
maskF[ν < 0.] = True
maskR[ν > 0.] = True

chemData = []
for species in ("C2H4", "O2", "CO", "H2O", "CO2"):
    data = pd.read_csv(f"chemData/{species}.txt", sep="\t", skiprows=1)
    chemData.append(data[1:])  # Skip T=0K

logKfuncs, deltaHfuncs = [], []
for data in chemData:
    T      = data["T(K)"].values.astype(float)
    logKf  = data["log Kf"].values.astype(float)
    deltaH = data["delta-f H"].values.astype(float) * 1e+03  # kJ/mol->kJ/kmol
    logKfuncs.append(interpolate.interp1d(T, logKf, kind="quadratic"))
    deltaHfuncs.append(interpolate.interp1d(T, deltaH, kind="quadratic"))


def Kc(T, p):
    """Kc = Kp * pow(pRef/Ru*T, νExp+...)"""
    # NOTE: Account for partial pressures
    Kf_i    = np.array([pow(10, Kf(T)) for Kf in logKfuncs]) * (pRef/(Ru*T))
    forward = pow(Kf_i, maskF*νExp)
    reverse = pow(Kf_i, maskR*νExp)
    return np.prod(reverse, axis=1) / np.prod(forward, axis=1)


def arrhenius(T):
    return np.array([
        1.739e+09 * math.exp(-1.485e+05 / (Ru*T)),
        6.324e+07 * math.exp(-5.021e+04 / (Ru*T))
    ])


ΔT   = 0.1e-03
temp = lin(1400, 2800, ΔT)  # K
pres = lin(70, 140, ΔT)     # kPa


@vectorInterface((5, 1))
def gradient(t, χ, h):
    limit = (χ < 0)
    χ[limit] = 0

    # Would normally calculate T from h = \int cp(T) dT
    T , p = temp(t), pres(t)
    kf    = arrhenius(T)
    kr    = kf / Kc(T, p)
    kr[0] = 0  # One way reaction

    forward = kf * np.prod(pow(χ, maskF*νExp), axis=1)
    reverse = kr * np.prod(pow(χ, maskR*νExp), axis=1)
    χGrad   = ν.T @ forward - ν.T @ reverse
    χGrad[(χGrad < 0)*limit] = 0
```

```
120
121     hGrad = -sum([dχ_i*h_i(T) for dχ_i, h_i in zip(χGrad, deltaHfuncs)])
122
123     return χGrad, hGrad
124
125
126 n = 1 + 3*(1 + 3.76)
127 χ0  = np.array(
128     [1/n, 3/n, 0.0, 0.0, 0.0]
129 ) * 70 / (Ru * 1400)
130 sol = integrate.solve_ivp(
131     gradient, (0, ΔT), np.append(χ0, 0.),
132     method="LSODA", events=None,
133     atol=1e-10, rtol=1e-10
134 )
135
136 t, y = sol.t, sol.y
137 print(f"The heat released is {y[-1][-1]*1e-03:.3f} MJ/m^3")
138 print(np.array([1/n, 3/n, 0.0, 0.0, 0.0]))
139
140 fig, ax = plt.subplots()
141 formula = ("C$_2$H$_4$", "O$_2$", "CO", "H$_2$O", "CO$_2$")
142 [ax.plot(t*1e+06, y[i]*1e+03, label=formula[i]) for i in range(5)]
143 ax.legend()
144 ax.set_xlim([0, 100])
145 plt.xlabel(r"Time [$\mu$s]")
146 plt.ylabel("Concentration [mol/m$^3$]")
147 plt.title("Concentration of species over combustion")
148 plt.savefig("../images/concentration.pdf")
149
150 fig, ax = plt.subplots()
151 ax.plot(sol.t*1e+06, sol.y[-1]*1e-03, "k-", label="Net heat")
152 ax.legend()
153 ax.set_xlim([0, 100])
154 ax.set_ylim([0, 0.5])
155 plt.xlabel(r"Time [$\mu$s]")
156 plt.ylabel("Net heat [MJ/m$^3$]")
157 plt.title("Net heat release from combustion")
158 plt.savefig("../images/netHeat.pdf")
159
160 fig, ax = plt.subplots()
161 ax.plot(
162     sol.t*1e+06,
163     np.gradient(sol.y[-1], sol.t)*1e-06,
164     "k-", label="Heat rate"
165 )
166 ax.legend()
167 ax.set_xlim([0, 100])
168 ax.set_ylim([-5, 15])
169 plt.xlabel(r"Time [$\mu$s]")
170 plt.ylabel("Rate of heat [GW/m$^3$]")
171 plt.title("Rate of heat of combustion")
172 plt.savefig("../images/heatRate.pdf")
173 plt.show()
```