Imperial College London

Heat flow coursework

Imperial College London

Department of Materials

MATE95006 Materials Engineering 2

Author:
Fengyi Li (CID: 01524021)

Date: October 1, 2020

Table 1: Variables used in the report all in SI units

Variables	Steel	Cordierite	Gas
Thermal contact conductance	2 (h_s)	2 (h_c)	N/A
Thermal conductivity	16.3 (k_s)	2.5 (k_c)	N/A
Radius	0.027 (r_1), 0.028 (r_2)	0.027	N/A
Length	0.1 (L_s)	0.01 (L_c)	N/A
Heat flux loss	0.475 ($Q_{s,trans}$)	0.280 ($Q_{c,trans}$)	N/A
Total heat flux	N/A	N/A	1391.97 (Q_{Tot})
Density	8030 (ρ_s)	2300 (ρ_c)	1.15 (ρ_a)
Heat capacity	450+0.28T (C_s)	900 (C_c)	N/A
Velocity	N/A	N/A	21.8 (v_g)
Initial temperature	298.15 (T_s)	298.15 (T_c)	537.15 (T_g)
Mass	0.135 (m_s)	0.015(m_c)	N/A
C	N/A	4×10^{27}	N/A
E	N/A	3×10^{4}	N/A

1 Tasks

1.1 Time to reach the equilibrium

According to the gas velocity and the length of the pipe, given incompressible and laminar flow, the total time for the gas to reach from the source to the other end is calculated in Eq.(1).

$$t_{\text{Tot}} = \frac{L}{v} = 4.122 \times 10^{-3}[s] \tag{1}$$

where L is the total length along the axis of the pipe, and v is the speed of the exhaust gas. According to the numerical analysis of the Fourier's second law of heat conduction, the system is divided into a huge amount of cubes for obtaining the temperature distribution based on the Eq.(2).

$$\begin{aligned} T(x,t,z,\Delta t+t) = \frac{\Delta\Delta t}{(\Delta h)^2}[&T(x+\Delta h,y,z,t)+T(x,y+\Delta h,z,t)+T(x,y,z+\Delta h,t)+T(x,y,z+\Delta h,t)\\ &+T(x-\Delta h,y,z,t)+T(x,y-\Delta h,z,t)+T(x,y,z-\Delta h,t)-6T(x,y,z,t)]\\ &+T(x,y,z,t) \end{aligned} \tag{2}$$

Due to the high symmetry of the system, the model will simulate the heat flow in a 2D grids for the pipe section. The terms related to the above and bottom ones in the Eq.(2), $T(x,y,z+\Delta h,t)$ and $T(x,y,z-\Delta h,t)$, will be replaced by $T(x,y,z,t)$ because the heat conducted radially will be isotropic. The boundary condition will be set as four extra sides of cubes surrounding the system. The code is in the Appendix.

The temperature change of the catalyst is also included in the Appendix, and the main function of it is named main_cata. It is credited to my colleague Wong han, and I changed a bit of it to work under my simulation environment.

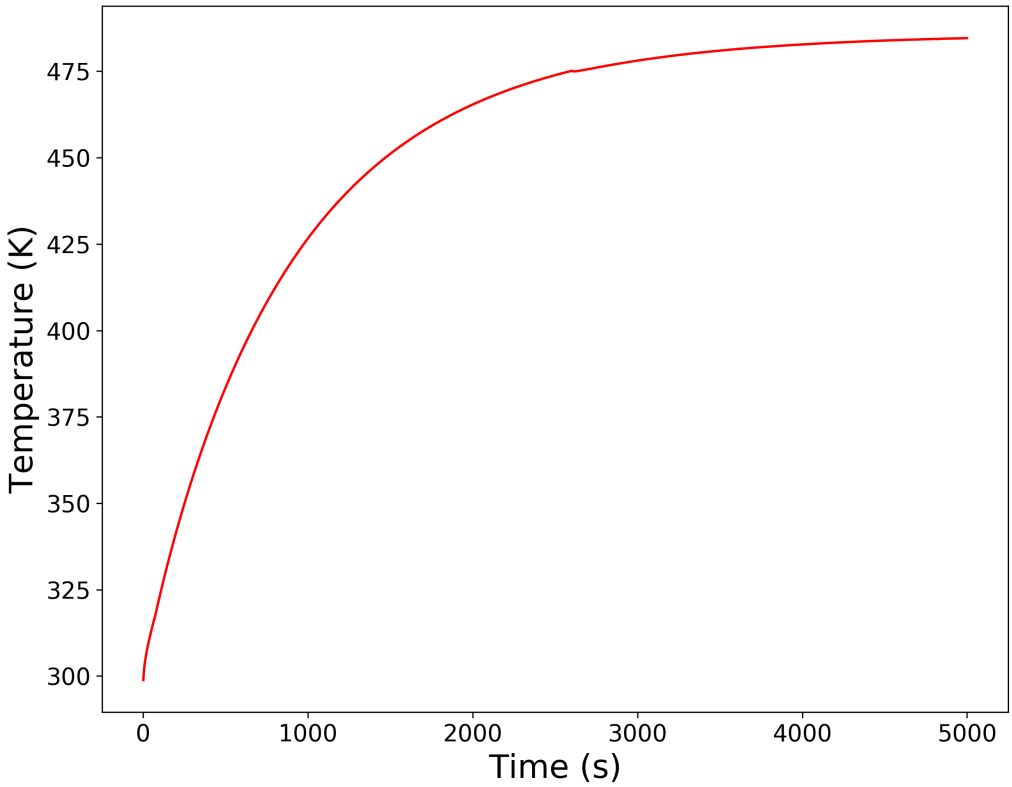

Figure 1: Temperature of the surface against the time by SolidWorks

This finite difference method takes lots of processing time for the computer to do when the Δh and Δt is infinitesimal small. With large values, the results will be high inaccurate.

Due to the low thickness of the wall, the value of dh can only be smaller than 0.001mm in order to obtain an enough amount of cubes along the pipe. However, according to the Eq.(1), dt should be changed with dh with a similar level at the same time to keep the accuracy of the results. A very small value of dt will cause large computational work for the computer, which leads to a long time running that my personal computer cannot afford even multiprocessing was implemented based on the number of cores in the laptop. Therefore, the SolidWorks simulation is used in the report.

1.2 Time profile of outermost surface

The Fig.1 shows the temperature change with the time of heating by SolidWorks simulation. The temperature is the maximum temperature of the pipe. The following Fig.2 gives the result of the temperature against the distance in SolidWorks. In the Fig.2, along the distance of the pipe, the temperature drops due to the advancing front face and the end of the pipe contacting the exterior atmosphere. When the pipe reaches the equilibrium, the temperature along the pipe is displayed in the Fig.2. This is achieved by using the 'cut plot' by drawing a line from one end of the pipe to the other one.

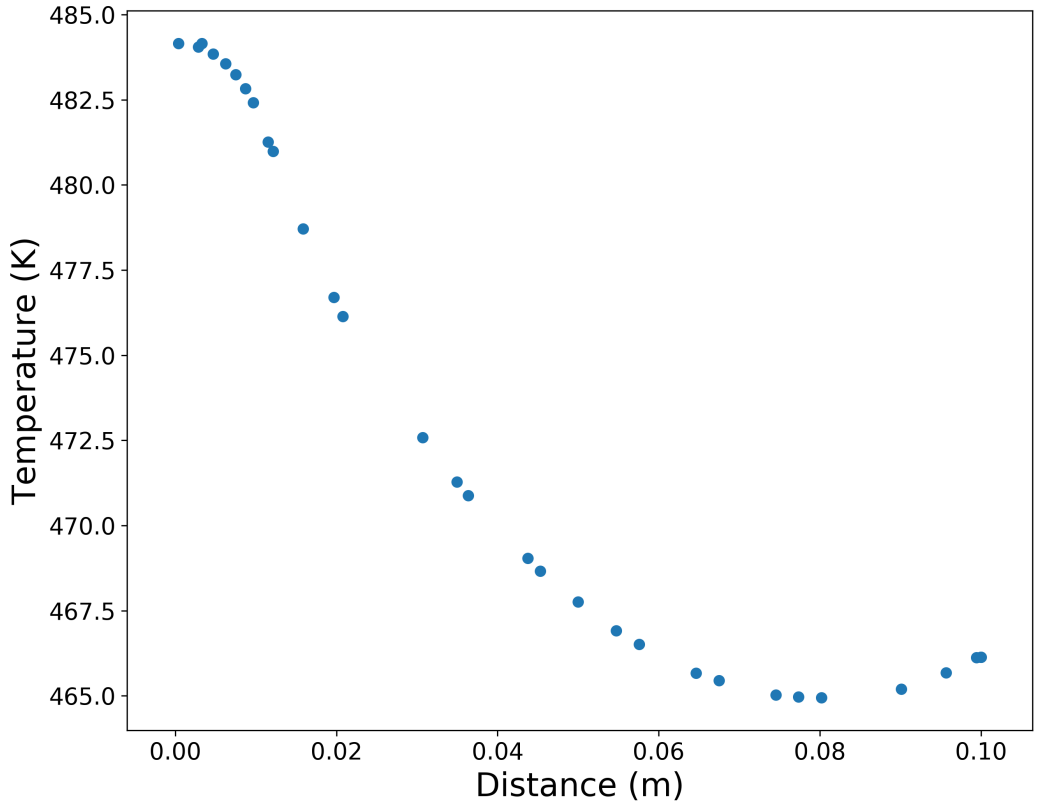

Figure 2: Temperature of the surface along the length of the pipe by SolidWorks

1.3 Time profile of catalyst

The Fig.3 shows the maximum temperature change with the time of heating by SolidWorks simulation. The following Fig.4 gives the result of the downstream temperature. In Fig.4, the temperature decreases when coming to the end of the catalyst from the beginning of the catalyst. The temperature drops in an irregular pattern due to the structure of the catalyst grids.

1.4 CO transmission

The Fig.5 shows the change of the CO molecules in the exhaust gas with the time. The blue horizontal line displays the gas only contains 0.01% CO molecules, and the red line represents the total number of gas molecules. The total molecules represent the gas in the system instead of being emitted to the atmosphere due to the very small time for the gas reaching the end of the pipe calculated in the Eq.(1). Therefore, the figure shows non-zero state during the whole process. The time required for reaching the purpose is about 1009s after the exhaust gas is emitted, given the C and E in the Table.1.

$$\frac{\partial^2}{\partial A \partial t} N_{CO} = -C \exp(-\frac{E}{RT}) \tag{3}$$

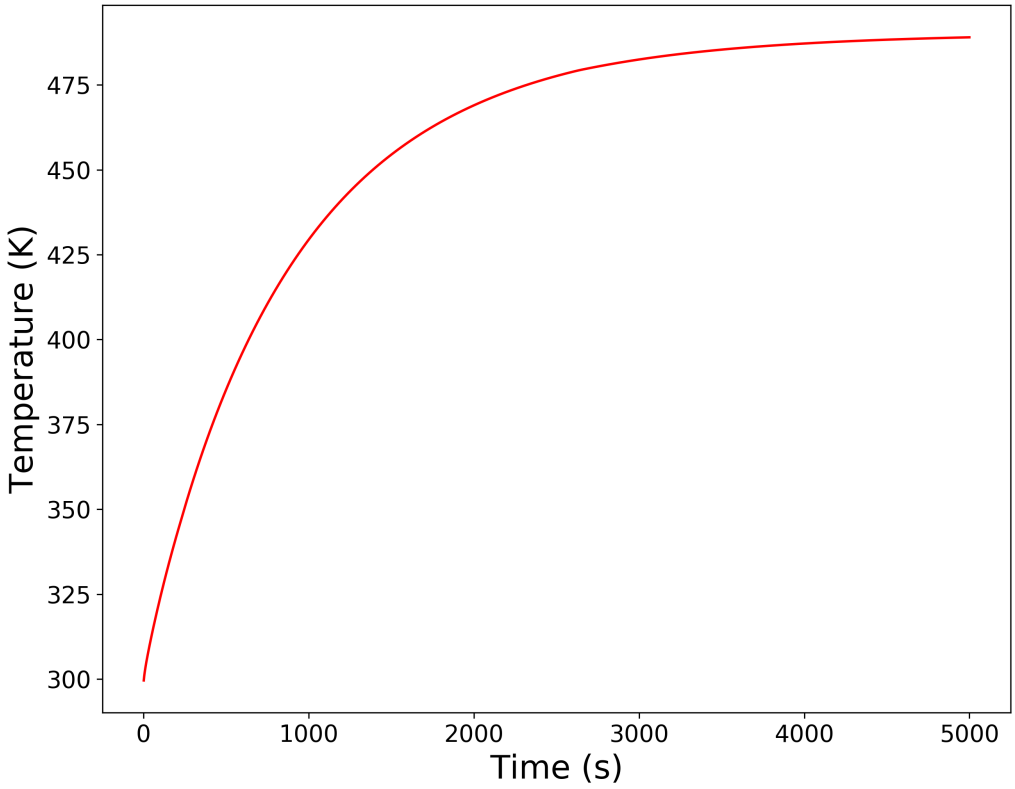

Figure 3: Temperature of the catalyst against the time by SolidWorks

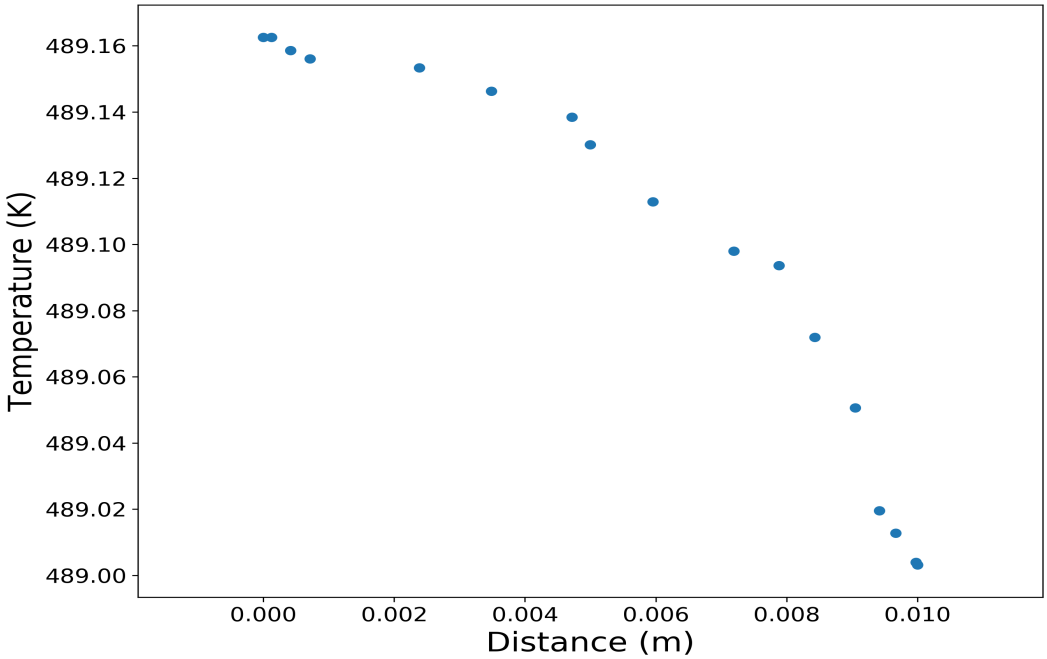

Figure 4: Temperature of the catalyst downstream against the distance by SolidWorks

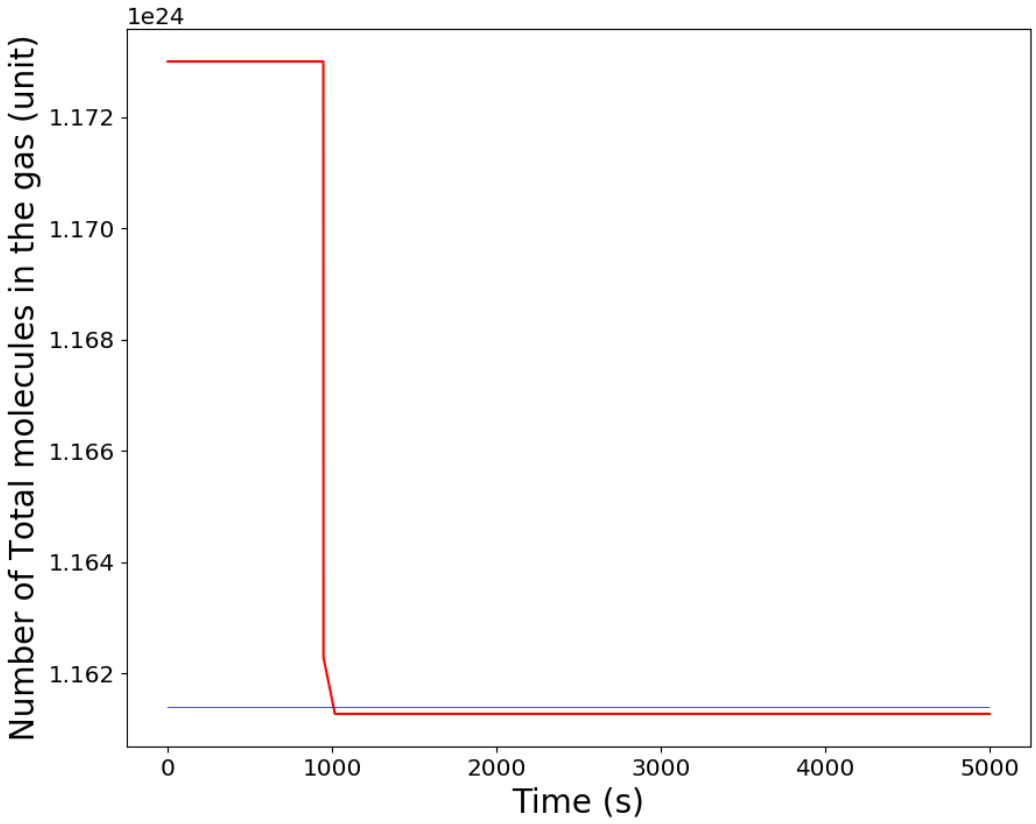

Figure 5: Change of the number of molecules in gas against time with the catalyst activated at 423.15K

2 Questions

2.1

If the exhaust gas is incompressible, the density of it is pressure-dependent instead of a constant through the whole simulation. The compressibility can be calculated in the Eq.(4).

$$\beta = -\frac{1}{V}\frac{\partial V}{\partial P} \tag{4}$$

Where V is the volume and P is the pressure. For an incompressible fluid, the compressibility is zero. In the established flow area, the incompressible fluid has an unchanged velocity profile along the channel due to the constant pressure gradient along the channel. However, for the compressible fluid, the pressure drop from inlet to outlet causes the expansion of volume of fluid, which leads to an increase of the fluid in low pressure area to keep the constant flow rate at inlet and outlet (1).

2.2

If the exterior is insulator, there will be no heat flux released to the atmosphere through the pipe wall. The Eq.(2) and (3) should be modified to approximate this situation by removing

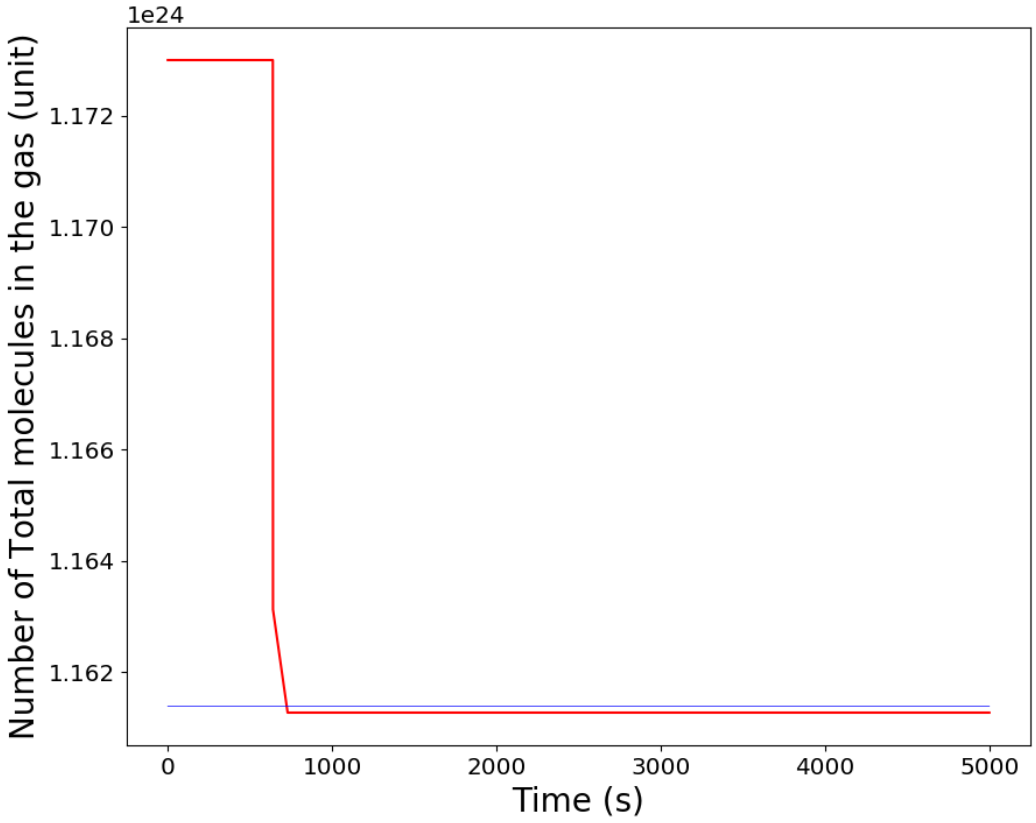

Figure 6: Change of number of molecules in gas against time with the catalyst activated at 398.15K

the terms related to the heat flux to the atmosphere. In SolidWorks, the simulation can be done by adding an idea wall feature to the exterior surface of the pipe to make it adiabatic. In Python3, there are four extra sides in the grid matrix to act as the boundary conditions, and we can remove the exterior boundary condition for the pipe contacting the atmosphere to achieve the adiabatic state.

2.3

2.3.1

When the operation temperature decreases to 398.15K, the time required to reach 0.01% CO molecules will be diminished. This can be deduced from the Eq.(3).

2.3.2

The Fig.6 shows the number of CO molecules left in the system, and the time reaching 0.01% CO molecules is 724s, given the same value of C in the Table.1 and E equal to 2.85×10^4.

2.4

If the gas is not incompressible, the speed of the gas will increase when it reaches the end of the pipe based on the Eq.(4), which cannot be considered as the constant anymore. This

should be modified by a function to find the speed change along the pipe to work for finding the right value of the time to convert CO molecules. Also, the temperature distribution of the system should be changed due to the uneven distribution of the speed of the gas. The time step could be adjusted to keep a constant advancing distance for the gas during the simulation. In the SolidWorks, the exhaust gas should be changed from the incompressible into compressible (1).

If the pressure difference between two ends cannot be neglected, the force due to pressure differences between two ends of cylinder should be considered in the Eq.(5).

$$F = (P_a - P_b)\pi r^2 \tag{5}$$

where P_a and P_b are pressures at two ends of the pipe and r is the inner radius. This force will contribute to the velocity in the channel with established laminar flow in the form of the Eq.(6).

$$v = \frac{1}{4\mu}\frac{P_a - P_b}{s}(r_0^2 - r^2) \tag{6}$$

where s is the length of the pipe (1).

If the exhaust gas is not laminar flow, the velocity of the turbulent channel flow should be considered by using the Eq.(7), power law (1).

$$v = v_{max}(1 - \frac{r}{r_0})^{1/n} \tag{7}$$

where v is the velocity of the fluid at r and v_{max} is the maximum velocity of the fluid. The exponent value varies with Reynolds number.

If the thermal conductance between the steel and the cordierite is finite, there will be a temperature drop at the contacting surface of the steel and the cordierite. This can be changed in Python to add extra functions. In SolidWorks, the contacting surface property should be changed (1).

If the air outside is not constant, in Python, the extra columns representing the atmosphere temperature should be updated instead of remaining as constant 298.15K.

References

[1] R. Qin, ”MSE206 Process Principles”, Department of Materials, Imperial college London, England. pages 6, 8

Appendix

```
import numpy as np
import matplotlib.pyplot as plt
import multiprocessing as mp
import matplotlib.pylab as plb
import gc

# Global parameters
v = 21.8
L = 0.1
t_Tot = L / v
T_g = 573.15
k_s = 16.3
rho_s = 8030
reco_step = 1
thickness = 0.001
L = 0.1
T_a = 298.15

paras = {
    'figure.figsize': (10, 8),
    'axes.labelsize': 'x-large',
    'axes.titlesize': 'x-large',
    'xtick.labelsize': 'x-large',
    'ytick.labelsize': 'x-large',
}
plb.rcParams.update(paras)


def main():
    pool = mp.Pool(mp.cpu_count())
    time, temp = pool.apply(body)
    pool.close()

    dt = (np.roll(time, -1, axis=0) - time)[: -1]
    Tbar = ((np.roll(temp, -1, axis=0) + temp) / 2)[: -1]
    Ti, = np.where(Tbar >= T_eff)

    N_target = N * (1 - f)
    N_transmitted = 0
    N_left = []

    for i in Ti:
        N_transmitted = C * ma.exp(-E / (R*i)) * A * dt[i]
        N_left.append(N - N_transmitted)

        if N_transmitted >= N_target:
            print('Time required is: {:.2f}'.format(dt[i]))

    N_left = np.concatenate((np.full((1, Ti[0]+1), N), N_transmitted), axis
    =0)
    fig = plt.figure(dpi=300)
    ax = fig.add_subplots(111)
    ax.plot(time, N_left, 'r-', linewidth=1.2)
    ax.set_xlabel('Number of unprocessed CO molecules (unit)')
    ax.set_ylabel('Temperature of the catalyst (K)')
    ax.ticklabel_format(axis='x', style='sci', scilimits=(0,0))
    ax.xaxis.get_offset_text().set_fontsize(15)
```

```
    ax.xaxis.major.formatter._useMathText = True
    fig.savefig('N_T_plot.png')
    plt.close(fig)
    gc.collect()

    fig = plt.figure(dpi=300)
    ax = fig.add_subplot(111)
    ax.plot(time, temp,'r-')
    fig.savefig('image.png')
    plt.close(fig)
    gc.collect()

    return


def run(grid, T_g, k_s, rho_s, T_Tot, dh, dt):
    for x in np.arange(1, grid.shape[0]-1):
            for y in np.arange(1, grid.shape[1]-1):
                if grid[x, y] == T_g:
                    pass
                else:
                    grid_last = T_Tot[-1]
                    c_s = 450 + 0.28 * (grid[x, y]-273.15)
                    alpha = k_s / (rho_s*c_s)
                    neighbours = getNeighbours((x, y))
                    coeff = alpha / (dh**2)
                    sum_T = 0
                    for pair in neighbours:
                        sum_T += grid_last[pair[0], pair[1]]

                    grid[x, y] = dt * (coeff * (sum_T - 6*grid_last[x, y])) +
 grid_last[x, y]

    return


def getNeighbours(point):
    neighours = np.array(
        [(point[0], point[1]+1),
         (point[0], point[1]-1),
         (point[0]+1, point[1]),
         (point[0]-1, point[1]),
         (point[0], point[1]),  # Cell above
         (point[0], point[1])]  # Cell below
    )

    return neighours


def PipeGrid(dh):
    grid = np.full((int(L/dh)+2, int(thickness/dh)+2), T_a)
    grid[0, :], grid[-1, :], grid[:, -1] = T_a, T_a, T_a
    grid[:, 0] = T_g

    return grid


def body():
    dh = 0.0001
    dt = 0.00001
```

```
    time = 100
    t_finish = int(time/dt)
    grid = PipeGrid(dh)
    X = int(grid.shape[0]/2)
    Y = grid.shape[1] - 2
    T_Tot = []

    file = open('stats.csv', 'w')
    file.write('Time (s), Temperature (K)\n')

    T_Tot.append(grid)
    file.write('{0}, {1:.2f}\n'.format(0, grid[X, Y]))
    for t in np.arange(1, t_finish):
        print("----{}----".format(t*dt))
        z = v * dt * t

        if z < L:
            heated = int(z/dh)
            grid[:heated, 1] = T_g

            if t % reco_step == 0:
                T_Tot.append(grid)
                file.write('{0:.2f}, {1:.2f}\n'.format(t*dt, grid[X, Y]))
        else:
            grid[:, 1] = T_g
            break

        run(grid, T_g, k_s, rho_s, T_Tot, dh, dt)

    t_start = t

    for t in np.arange(t_start, t_finish):
        print("----{}----".format(t*dt))
        if t % reco_step == 0:
            T_Tot.append(grid)
            file.write('{0:.2f}, {1:.2f}\n'.format(t*dt, grid[X, Y]))

        run(grid, T_g, k_s, rho_s, T_Tot, dh, dt)

    t_list = np.arange(len(T_Tot)) * dt
    T_plot = []
    for T in T_Tot:
        T_plot.append(T[X, Y])

    file.close()

    return t_list, T_plot

################### FOR CATALYST ########################
gridRad = 0.027
OD = 0.028
ID = 0.027
dh = 0.0001
catGap = 0.004
catWid = 0.001
catL = 0.01
T0 = 298.15
L = 0.1


```

```
176 def RasterQuad(r): # sub-function of DiscretePipe and DiscreteCat. Rasterizes
     a 2D quadrant of radius r pixels
177     inner = []
178     outer = []
179     x, y = r, 0     # midpoint circle algorithm
180     while x >= y:
181         while x <= np.sqrt(r**2 - y**2) + 0.5:
182             inner.append((x, y))
183             outer.append((x+1, y))
184             y += 1
185         x -= 1
186     outer.append((x+1, y))
187     inner = inner + [(y,x) for (x,y) in inner]
188     outer = outer + [(y,x) for (x,y) in outer]
189     return inner, outer # inner has the given radius. outer is the shell with
     radius r+1
190 
191 
192 def DiscreteCat(OD, ID, L, dh, catGap, catWid, catL, T0): # quarters and
    discretises the catalyst
193     catGrid = np.zeros((gridRad+2, gridRad+2))
194     inner, _ = RasterQuad(int(ID/dh/2-1))
195 
196     for point in inner:
197         catGrid[:point[1]+1, point[0]] = 1
198 
199     for i in range(catGrid.shape[0]):
200         if 0 <= (i*dh + catGap/2) % (catGap+catWid) < catGap:
201             for j in range(catGrid.shape[1]):
202                 if 0 <= (j*dh + catGap/2) % (catGap+catWid) < catGap:
203                     catGrid[i,j] = 0
204 
205     catNodes2D = np.argwhere(catGrid == 1)
206     catNodes3D = np.vstack((np.hstack((catNodes2D, np.full((len(catNodes2D)
    ,1), z))) for z in range(int(catL/dh))))
207     catNodes3D[:,2] += int((L-catL)/dh)
208     catNodesT = dict(zip([tuple(row) for row in catNodes3D.tolist()], [T0 for
     i in range(len(catNodes3D))]))
209 
210     return catNodesT # a dictionary with spatial coordinates as keys and
    temperatures as values
211 
212 
213 def getNeighbours_3D(point):
214     neighbours = np.array([
215         (point[0], point[1]+1, point[2]),
216         (point[0], point[1]-1, point[2]),
217         (point[0]+1, point[1], point[2]),
218         (point[0]-1, point[1], point[2]),
219         (point[0], point[1], point[2]+1),  # Cell above
220         (point[0], point[1], point[2]-1)  # Cell below
221     ])
222 
223     return neighbours
224 
225 
226 def run_cata(grid, T_g, k_s, rho_s, T_Tot, dh, dt):
227 
228     for x in np.arange(1, int(ID/dh)):
229             for y in np.arange(1, int(catL/dh)):
```

```
                for z in np.arange(1, int(ID/dh)):
                    if grid[(x,y,z)] == T_g:
                        pass
                    else:
                        grid_last = T_Tot[-1]
                        c_s = 450 + 0.28 * (grid[(x,y,z)]-273.15)
                        alpha = k_s / (rho_s*c_s)
                        neighbours = getNeighbours_3D((x, y, z))
                        coeff = alpha / (dh**2)
                        sum_T = 0
                        for pair in neighbours:
                            sum_T += grid_last[pair[0], pair[1]]

                        grid[(x, y, z)] = dt * (coeff * (sum_T - 6*grid_last
    [(x, y, z)])) + grid_last[(x, y, z)]

    return grid


def main_cata():
    grid = DiscreteCat(OD, ID, L, dh, catGap, catWid, catL, T0)
    grid_T = []
    grid_T.append(grid)
    for t in time:
        grid = run_cata(grid, T_g, k_s, rho_s, T_Tot, dh, dt)
        grid_T.append(grid)

    return grid_T

######################### CO transfer ###################################
import matplotlib.pyplot as plt
import matplotlib.pylab as plb
import pandas as pd
import scipy.constants as constant
import pandas as pd
import numpy as np
import gc
import math as ma


# Global variables
C = 4 * 10**27
E = 2.85 * 10**4
R = constant.R
T_eff = 273.15 + 125
N = (1.173 * 10**24)
N_CO = N * 0.01
T_a = 298.15
f = 0.01 * 10**-2
A = 0.01351

# Globle setting for the figure drawing
params = {'legend.fontsize': 'x-large',
          'figure.figsize': (10, 8),
          'axes.labelsize': 20,
          'axes.titlesize': 20,
          'xtick.labelsize':'x-large',
          'ytick.labelsize':'x-large'}
plb.rcParams.update(params)

```

```
289
290 def main():
291
292     data = pd.read_csv('data3.csv')
293     time = data.iloc[:, 0].to_numpy()
294     temp = data.iloc[:, 1].to_numpy()
295
296     dt = (np.roll(time, -1, axis=0) - time)[: -1]
297     dt = np.insert(dt, 0, time[1]-time[0])
298     Tbar = ((np.roll(temp, -1, axis=0) + temp) / 2)[: -1]
299     Tbar = np.insert(Tbar, 0, T_a)
300     Ti, = np.where(Tbar >= T_eff)
301
302     N_target = N_CO - N * f
303     N_transmitted = 0
304     N_left = []
305     N_t = N - N_target
306
307     for i in np.arange(Ti[0]):
308         N_left.append(N)
309
310
311     for i in Ti:
312         N_transmitted = C * ma.exp(-E / (R*Tbar[i])) * A * dt[i]
313         N_l = N - N_transmitted
314         N_ll = N_CO - N_transmitted
315
316         if N_ll < 0:
317             N_left.append(N*0.99)
318         else:
319             N_left.append(N_l)
320
321         if N_l <= N_t and i <1200:
322             print("Time: {},  {}".format(i, N_l))
323
324
325     fig = plt.figure()
326     ax = fig.add_subplot(111)
327     ax.plot(time, N_left, 'r-')
328     ax.plot(time, np.full((len(time), 1), N - N_target), 'b-', linewidth=0.5)
329     ax.set_xlabel('Time (s)')
330     ax.set_ylabel('Number of Total molecules in the gas (unit)')
331     fig.savefig('image.png')
```

Listing 1: Python codes for the heat flow simulation

2.5 SolidWorks

After drawing the model as the task sheet instructs with a lid to seal the pipe, in the flow simulation, several initial and boundary conditions are set to successfully obtain the final results.

2.5.1 Initial condition

1. External simulation environment
2. Solid conduction and time dependent
3. Define own materials (the steel and cordierite)

4. Perfectly smooth wall
5. Atmospheric temperature
6. Set the computation control variable as the physical time (*i.e.* 5000s)

2.6 Subdomain flow

The subddomain flow should be set by using the user-defined exhaust gases with right speed and direction based on the coordinate provided by the environment.

2.7 Solid Materials

Choose the pipe as the steel defined and the cordierite as the user-defined material.

2.7.1 Boundary condition

1. Ideal wall of the lateral and the outer face of the lid
2. Real wall of all the faces contacting with the fluid (atmosphere and the exhaust gas)
3. Initial temperature of the system for each wall
4. Set the inner face of the lid as the INLET VOLUME FLOW as the entrance where the flow comes into the system

2.8 Results

After the simulation finishes, in the results tag, several plots can be draw based on the data. Use the CUT PLOT to obtain the temperature change along the specified direction. The data could be also exported by the EXPORT.