**ENGN2020 – HOMEWORK8**

### Problem 1

### Part (a)

The initial condition is that at z = 0, where is exactly the outlet of , C = 0. That`s .

Let:

, then

The original equation can be written as:

Let:

, then ,

The equation above can be written as:

Where:

The initial condition becomes:

### Part (b)

The equation can be written as:

Let:

, then ,

The equation can be written as:

Let:

, then ,

The equation can be written as:

Where:

The initial condition becomes:

### Problem 2

Where:

The function can be rewritten as:

In python, the function can be defined as:

import math

from scipy.integrate import odeint

'''

\* @name: f\_fit

\* @description: the function to be fitted by the experimental points

\* @param t: time, in form of array

\* @param cp: heat capacity

\* @param h: surface heat-transfer coefficient

\* @return: the corresponding temperature of given t

'''

def f\_fit(t,cp,h):

#copy the input parameter of cp and h

alpha = cp

beta = h

#the pre-defined parameter

m = 1

a = 200

b = 0.1

A = 0.02

T\_inf = 25

#define the function to be solved by numerical method in lambda format

f = lambda y,t: -beta\*A\*(y-T\_inf)/m/alpha + a \* (1+ math.sin(b\*t))/m/alpha

#set the initial guess by T0

y0 = 25

#solve for temperature of given time

results = odeint(f, y0, t)

#resize the temperature as 1\*141

results.resize((141))

return results

The experiment data can be loaded by following code:

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import curve\_fit

#load the experimental data

data = np.load('thermal-block.npz')

#get time

t = data['times']

#get temperature

realTemperature = data['temperatures']

#resize the temperature as the same size with time

realTemperature.resize((141))

Use scipy.optimize.curve\_fit to fit for the experimental results to calculate cp and h. After calculation,

#curve fit to calculate cp and h

popt, pcov = curve\_fit(f\_fit, t, realTemperature)

#calculate the temperature based on fitting result

fitTemperature = f\_fit(t,popt[0], popt[1])

fig, axs = plt.subplots(1, 1)

axs.plot(t,fitTemperature,'r')

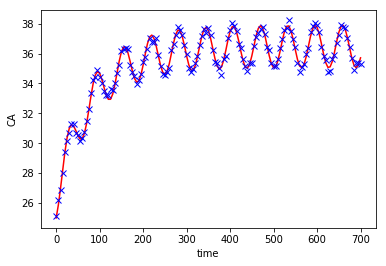
axs.plot(t,realTemperature,'bx')

axs.set\_xlabel('time')

axs.set\_ylabel('CA')

plt.show()

The plot containing the best-fit model solution and the experimental points is shown in Fig.1.



**Fig 1.** The plot containing the best-fit model solution and the experimental points

### Problem 3

### Part (a)

According to the problem:

At x = 0:

At x = L: **③**

Let , ,

Then, equation 3 becomes:

Let , ,

Then, equation 3 becomes:

Then equation 2 becomes:

Let *,* then equation 2 becomes:

From above and , then:

Then equation 1 becomes:

Then:

Let , then:

In conclusion:

Where:

### Part (b)

**Fig 2.** The finite element model

To discretizing these equations:

The first boundary condition becomes:

Therefore:

The second boundary condition becomes:

Therefore:

The governing function becomes:

The code for class Derivatives is shown as below:

class Derivatives:

'''

\* @name: \_\_init\_\_

\* @description: the constructor of class

\* @param n: parameter of number of elements

\* @param eta: parameter of eta

'''

def \_\_init\_\_(self,n,eta):

self.n = n

self.step = 1/n

self.eta = eta

'''

\* @name: \_\_call\_\_

\* @description: the call function

\* @param y: current parameter for 'theta'

\* @param t: input value of 'tao'

\* @return: dy, the value of dydt

'''

def \_\_call\_\_(self, y, t):

dy = np.zeros((self.n))

#the boundary condition at z=0

dy[0] = (2\*y[1]-2\*y[0]-2\*self.step\*﻿self.eta\*y[0])/self.step/self.step

#the governing function

for i in range(1,self.n-1):

dy[i] = (y[i+1]- 2\*y[i]+y[i-1])/self.step/self.step

#the boundary condition at z=n-1

dy[self.n-1] = (2\*y[self.n-2]-2\*y[self.n-1])/self.step/self.step

#return dy

return dy

### Part (c)I

In order to show the effect of , the plots of versus t at different positions, namely z = 0, z = 0.5, z = 1 are shown in Fig.3.



（a） = 0.1 (b) = 0

(c） = 1 (d) = 10

**Fig 3.** plots of versus t at different positions, namely z = 0, z = 0.5, z = 1

From Fig.3, it can be found that, when = 0, the left end is also perfectly insulated, there will be no heat exchange between block and environment, the block keeps its original temperature.

When 0, as increases, the system takes less time to reach the stable state. So has effect on the “speed” of transferring heat.

To solve this problem numerically, the following code was used. In this piece of code, ﻿scipy.integrate.odeint is used.

'''

\* @name: solve

\* @description: the function to get solve the finite element problem

\* @param n: the number of finite elements in the block

\* @param eta: parameter of transferring heat

'''

def solve(n,eta):

#define a object based on input n and eta

dydt = Derivatives(n, eta)

#get the initial guess of y0

y0 = np.ones((n))

#get t

t = np.linspace(0,10,1000)

#solve for y

y = odeint(dydt, y0, t)

#show y versus t at different positions, z = 0, z=0.5, z=1

fig, axs = plt.subplots(3, 1)

axs[0].plot(t,y[:,0],'r')

axs[0].set\_xlabel('time')

axs[0].set\_ylabel('theta at z = 0')

axs[1].plot(t,y[:,n//2],'g')

axs[1].set\_xlabel('time')

axs[1].set\_ylabel('theta at z = 0.5')

axs[2].plot(t,y[:,n-1],'b')

axs[2].set\_xlabel('time')

axs[2].set\_ylabel('theta at z = 1')

plt.show()