Project 4

ME-328-102: ME Analysis

Abstract

Implementing the Liebmann Method to sole the temperature distribution of a heated plate.

Justin Dyer

University of South Alabama student, Author

Landon Freeman

University of South Alabama student, Program Discussion

Index

4-1 Problem Description:

A rectangular plate with boundary conditions is shown below in Figure 4.1. Assuming that the temperature distribution in the plate is in steady state, one can start with the following equation (Eqn 4.1).

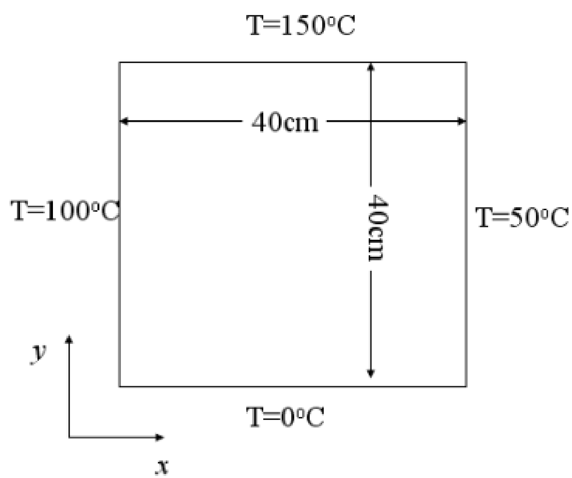


Figure 4.1 (Rectangular plate heated with boundary conditions.)

$$\frac{\delta T}{\delta t} = 0 \qquad \text{(Eqn 4.1)}$$

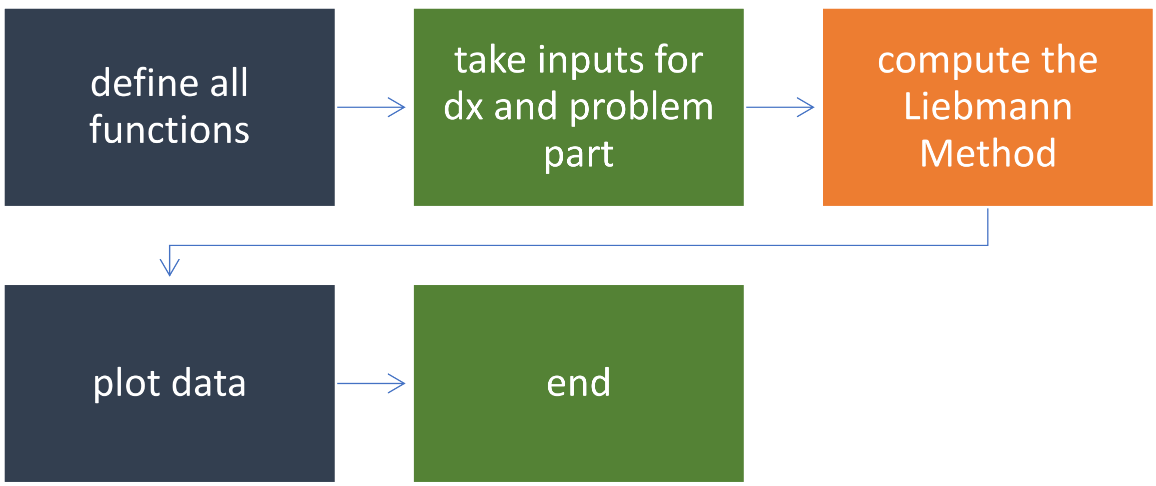
Given the heat conductivity, $k' = 0.49 \frac{cal}{s{-}cm{-}°C}$, and the governing equation to be the following (Eqn 4.2), one could write a Python program to implement the Liebmann Method to solve the temperature distribution of the plate.

$$\frac{\delta^2 T}{\delta x^2} + \frac{\delta^2 T}{\delta y^2} = 0 \qquad \text{(Eqn 4.2)}$$

4-2 Flowchart:

To begin this problem, we need to derive a flowchart to guide us while writing the program. We found that the following flowchart was best suited to guide us:

(Figure 4-1: Flowchart)



Using this flowchart, we have a layout of the steps needed to take to complete what is asked. First, input values for dx and define functions. Second, take the values and define all the variables. Third, plug the values into the functions to compute the Liebmann Method. Fourth, save the plots and show them. Finally, end the program.

Following this layout should guide us and let us successfully create the program needed.

4-3 Python Source Codes:

For this project, we used the following Python source codes:

Functions:

```
##PROJ4 FUNCTIONS

import math
import numpy as np


def LiebmannM(T_old, relax_factor, i,j):
   T_new_i_j = (T_old[i+1][j] + T_old[i-1][j] + T_old[i][j+1] + T_old[i][j-1])/4.0
   T_new_i_j = relax_factor*T_new_i_j + (1-relax_factor)*T_old[i][j]
   if T_new_i_j != 0:
      es_i_j = abs((T_new_i_j-T_old[i][j])/T_new_i_j)
   elif T_old[i][j] !=0:
      es_i_j = abs((T_new_i_j-T_old[i][j])/T_old[i][j])
   else:
      es_i_j = 1.0;

   return T_new_i_j,es_i_j

def heatFlux(T, kp, dx, dy):
   m = len(T)
   m=m-1
   n = len(T[0])
   n=n-1

   qx = np.zeros((m+1,n+1), dtype = float)
   qy = np.zeros((m+1,n+1), dtype = float)

   for i in range(1, m, 1):
      for j in range(1, n, 1):
         qx[i][j] = -kp*( T[i+1][j] - T[i-1][j] )/(2*dx)
```

```
        qy[i][j] = -kp*( T[i][j+1] - T[i][j-1] )/(2*dy)

    i = 0
    for j in range(1, n, 1):
        T_0_j = 4*T[i,j]-(T[i+1][j]+T[i][j+1]+T[i][j-1]);
        qx[i][j] = -kp*(T[i+1][j]-T_0_j)/(2*dx);
        qy[i][j] = -kp*(T[i][j+1]-T[i][j-1])/(2*dy);

    i = m;
    for j in range(1, n, 1):
        T_mplus2_j = 4*T[i][j]-(T[i-1][j]+T[i][j+1]+T[i][j-1]);
        qx[i][j] = -kp*(T_mplus2_j-T[i-1][j])/(2*dx);
        qy[i][j] = -kp*(T[i][j+1]-T[i][j-1])/(2*dy);

    j = 0
    for i in range(1, m, 1):
        T_i_0 = 4*T[i][j]-(T[i+1][j]+T[i-1][j]+T[i][j+1]);
        qx[i][j] = -kp*(T[i+1][j]-T[i-1][j])/(2*dx);
        qy[i][j] = -kp*(T[i][j+1]-T_i_0)/(2*dy);

    j = n
    for i in range(1, m, 1):
        T_i_nplus2 = 4*T[i][j]-(T[i+1][j]+T[i-1][j]+T[i][j-1]);
        qx[i][j] = -kp*(T[i+1][j]-T[i-1][j])/(2*dx);
        qy[i][j] = -kp*(T_i_nplus2-T[i][j-1])/(2*dy);

    return qx,qy

def y_exact_solution(t):
    t2=0.0866025*t
    y_exact_solution = math.exp(-
0.05*t)*(0.2*math.cos(t2)+0.11547*math.sin(t2));
    return y_exact_solution

def derivs(y,z,t,m,c,k):
```

```
    dy_over_dt = z
    dz_over_dt = -c/m*z-k/m*y
    return dy_over_dt, dz_over_dt

def explicit_euler(y,z,t,h,m,c,k):
    dy_over_dt,dz_over_dt = derivs(y,z,t,h,m,c,k)
    y_iplus1 = y+dy_over_dt*h
    z_iplus1 = z+dz_over_dt*h
    return y_iplus1, z_iplus1

def midpoint(y,z,t,h,m,c,k):
    dy_over_dt,dz_over_dt = derivs(y,z,t,m,c,k)
    y_mid=y+dy_over_dt*0.5*h
    z_mid=z+dz_over_dt*0.5*h
    dy_over_dt,dz_over_dt = derivs(y_mid,z_mid,t,m,c,k)
    y_iplus1=y+dy_over_dt*h
    z_iplus1=z+dz_over_dt*h
    return y_iplus1, z_iplus1

def RK4(y,z,t,h,m,c,k):
    y_iplus1 = 0
    z_iplus1 = 0
    dy_over_dt=0
    dz_over_dt=0

    dy_over_dt, dz_over_dt = derivs(y,z,t,h,m,c,k)
    k1y=dy_over_dt
    k1z=dz_over_dt

    dy_over_dt, dz_over_dt = derivs(y+k1y*h/2, z+k1z*h/2, t+h/2, m,c,k)
    k2y=dy_over_dt
    k2z=dz_over_dt

    dy_over_dt, dz_over_dt = derivs(y+k2y*h/2, z+k2z*h/2, t+h/2, m,c,k)
    k3y=dy_over_dt
```

```
    k3z=dz_over_dt

    dy_over_dt, dz_over_dt = derivs(y+k3y*h, z+k3z*h, t+h, m,c,k)
    k4y=dy_over_dt
    k4z=dz_over_dt

    y_iplus1 = y+(k1y+2*k2y+2*k3y+k4y)*h/6;
    z_iplus1 = z+(k1z+2*k2z+2*k3z+k4z)*h/6;

    return y_iplus1, z_iplus1
```

Solution:

```
##PROJ4 SOLUTION
import math
import numpy as np
import matplotlib.pyplot as plt

import project_4_functions as f
print('Project 4. Liebmann (Gauss-Siedel) Method for solving PDE.\n')

dx=float(input('input dx : '))
dy=dx
flag_Part_B = input('is this for Part B question (Y/N)? :')
es= 0.01
max_iter=1000
relax_factor= 1.5

kp=0.49
L=40
W=40

qy_buttom = -5.0
```

```
m=int(math.floor(L/dx))
n=int(math.floor(W/dy))


T= np.zeros((m+1, n+1), dtype = float)
ea = np.zeros((m+1, n+1), dtype = float)
e = 1.0
count = 0

for nj  in range(0, n+1, 1):
   T[0][nj] = 100.0;

if flag_Part_B != 'Y' or flag_Part_B != 'y':
   for mi in range(0, m+1,1):
      T[mi][0]= 0.0

for mi in range(0, m+1, 1):
   T[mi][n] = 150.0;

while (e > es and count < max_iter):
   e=0

   for i in range(1, m, 1):
      if flag_Part_B =='Y' or flag_Part_B == 'y':
         for mi in range(1, m, 1):
            T[mi][0]= 0.25*(T[mi+1][0]+T[mi-1][0]+2*T[mi][0+1]-2*dy*(-
qy_buttom/kp))

      for j in range(1, n, 1):
         [T_new_i_j,es_i_j] = f.LiebmannM(T, relax_factor, i, j);
         T[i][j]=T_new_i_j;
         ea[i][j]=es_i_j;
         e=e+es_i_j;

   count = count + 1;
```

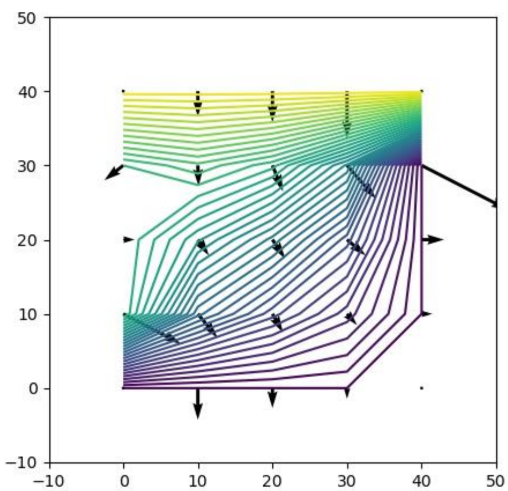
```
    iteration = count
    e = e/((m)*(n))
    print('e = ', e)

x = np.linspace(0, L, m+1)
y = np.linspace(0, W, n+1)
fig1 = plt.figure(1)
fig1.set_figheight(5.0)
fig1.set_figwidth(5.0)
plt.xlim(-10, L+10)
plt.ylim(-10, W+10)
[X, Y] = np.meshgrid(x,y)
plt.contour(X.transpose(), Y.transpose(), T, 40)
[qx,qy] = f.heatFlux(T, kp, dx, dy);
plt.quiver(X.transpose(), Y.transpose(),qx,qy)

fig1.show()
plotFileName = 'contours_and_flux_plot' + '.jpg'
plt.savefig(plotFileName, format = 'jpg')
```

4-4 Results and Discussion

The program outputs a plot for each value of dx. For our project, we ran the code three times for each equatio. The following 6 plots were received:

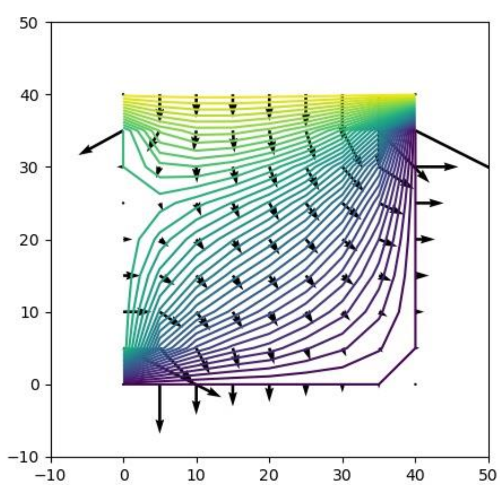
Part A

Dx = 10cm



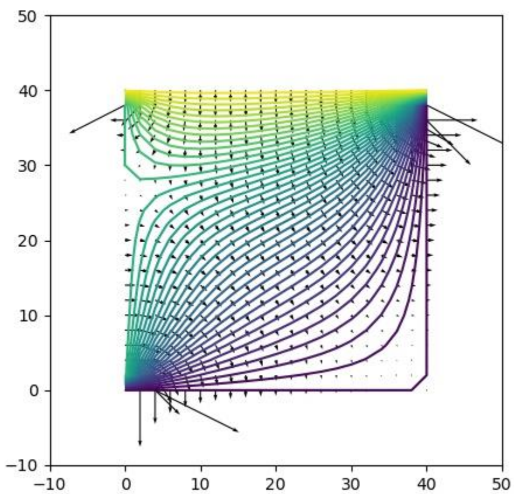
Part A

Dx = 5cm



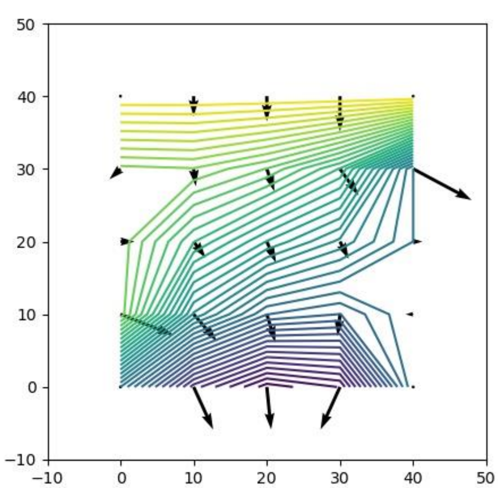
Part A

Dx = 2cm



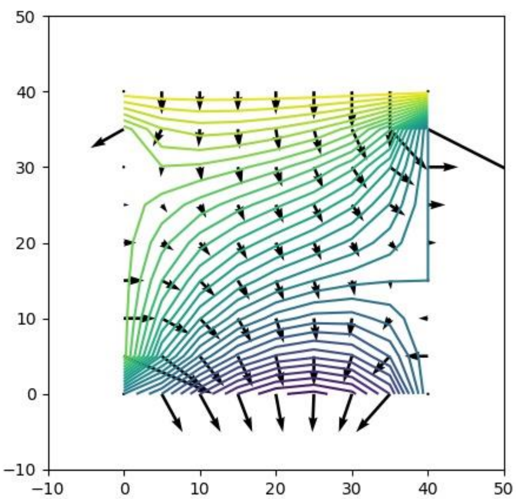
Part B

Dx = 10cm



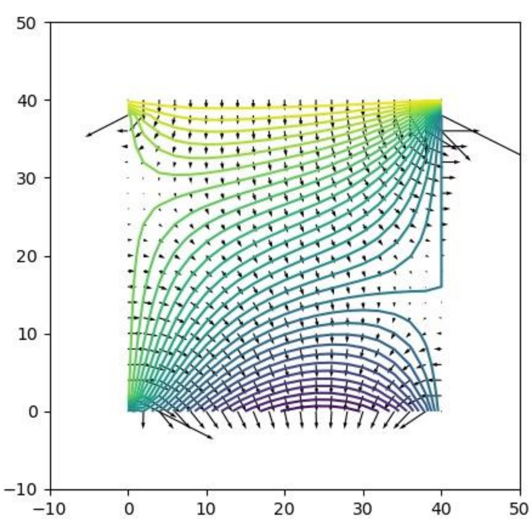
Part B

Dx = 5cm



Part B

Dx = 2cm



4-5 Conclusions:

We have learned to use Liebmann Method in previous classes. For each equation given, with our knowledge of Python, we wrote a code plot the temperature and heat flux distribution of the plate.

Our Python program successfully took the values inputted, and plotted the temperature and heat flux distribution of the plate. To improve this project, we could have more data input points and more accurate inputs; however, to our knowledge, the outputs we received are exact. We also could decrease the time step.