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……………………………………………………………..……pg2

4.2 Flowchart………………………………………………………………………………..….pg3

4.3 Python Source Codes………………………………………………………...……pg4-9

4.4 Results and Discussion……………………………………………………..….pg10-12

4.5 Conclusions…………………………………………………………………….………..pg13

# 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).

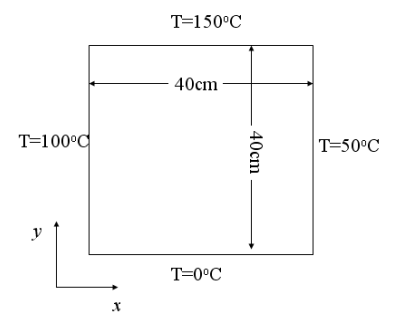


Figure 4.1 (Rectangular plate heated with boundary conditions.)

(Eqn 4.1)

Given the heat conductivity, , 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.

(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'

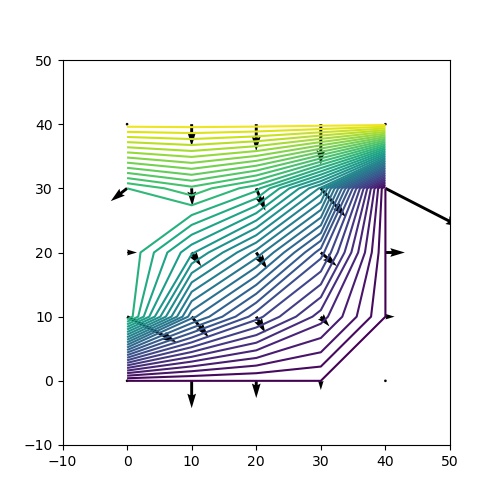
plt.savefig(plotFileName, format = 'jpg')

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:

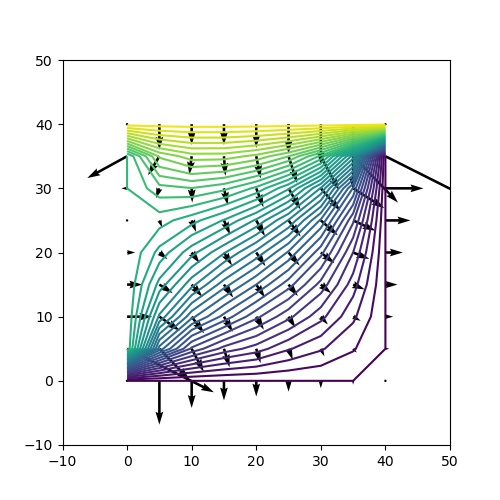
Part A

Dx = 10cm



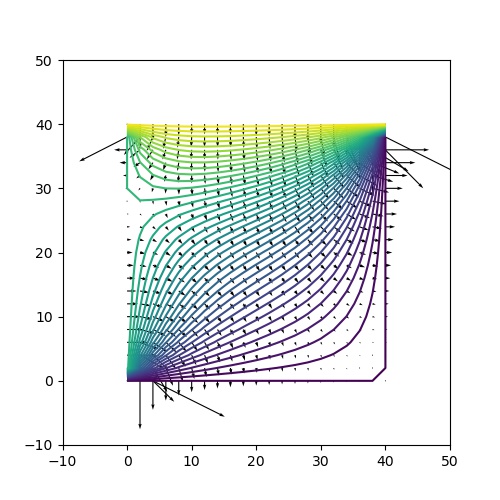
Part A

Dx = 5cm



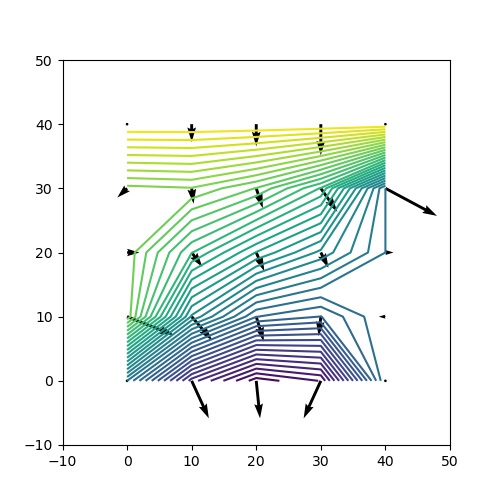
Part A

Dx = 2cm



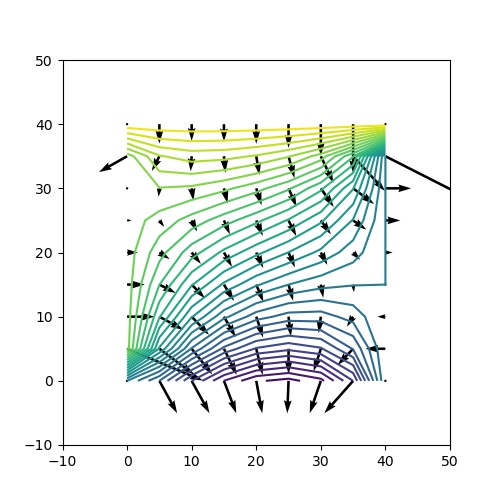
Part B

Dx = 10cm



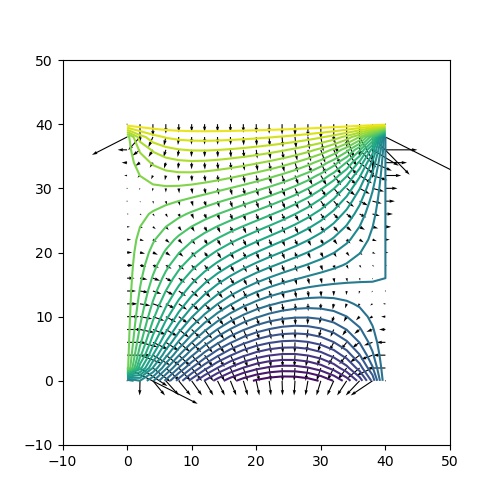
Part B

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.