Clay Freeman

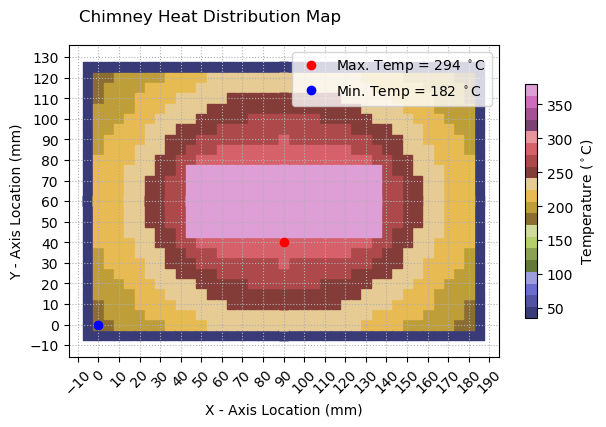
ME 399 : Heat and Mass Transfer

J. Mahoney - Spring 2018

Homework Assignment 7

Chimney, 5.1, 5.2, 5.3, 5.8

8 March 2018

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

import pprint

def chimney():

'''

The chimney cross section is given. The outside material is composed of a

fire brick with a thermal conductivity of 4.1 W/m-C. Determine the

temperature profile of the stack using the following inside and outside

convective conditions.

|--------- x\_outside --------|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

| | |

\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |

| | | | | |

y\_inside | | | | y\_outside

\_|\_\_\_\_ | |\_\_\_\_\_\_\_\_\_\_\_\_\_\_| | |

| | |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_|\_

|-- x\_inside --|

Inside temp. T\_i = 380 C

Inside conv. coeff. h\_i = 85 W/m^2-C

Outside temp. T\_o = 35 C

Outside conv. coeff. h\_o = 22.0 W/m^2-C

This program can take a variable node size, geometry, temperatures and material constants to

generate a heat plot of cross section for the chimney. To reduce computation time, the program

uses symmetry to break the plot into quarters and then rebuilds the full chimney size once the

mean error threshold has been met.

'''

node\_size = 5 # Square node side length (mm)

# Geometry

x\_outside = 180 # mm

x\_inside = 100 # mm

y\_outside = 120 # mm

y\_inside = 40 # mm

# Error estimation and iteration limits

c\_error = 100 # Current error percentage

er\_max = .5 # Maximum error tolerance

c\_iter = 0 # Instantiate iterations

iter\_max = 1000 # Maximum iterations

init\_temp = 0 # Initial temp to start the iterator

delta\_x = node\_size/1000 # Physical size of the node in meters

# Heat Transfer

in\_temp = 380 # Inside temp (C)

h\_in = 85 # Inside convection coefficient

out\_temp = 35 # Outside temp

h\_out = 22 # Outside convection coefficient

k\_brick = 4.1 # conduction coefficient for fire bricks

# Chop the diagram and generate points

cols\_x = int((x\_outside / (2 \* node\_size)) + 1)

rows\_y = int((y\_outside / (2 \* node\_size)) + 1)

# find the inner edges

ineg\_x = int((cols\_x - (x\_inside/(2\*node\_size)) - 1))

ineg\_y = int((rows\_y - (y\_inside/(2\*node\_size)) - 1))

# Generate an array of dictionaries to hold the values we need

node\_matrix = np.array([[{'temp':init\_temp, 'oldtemp' : 0, 'OB' : False} for i in range(cols\_x)] for j in range(rows\_y)])

# While we are under the iteration limit and outside the error threshold...

while c\_error > er\_max and c\_iter < iter\_max:

# cycle the iterator

c\_iter += 1

# Set the current error

tot\_er = 0

# instantiate min and max temp. assumes the object is being heated from the inside

max\_temp = out\_temp

min\_temp = in\_temp

# Break the node array into an indexed list

for y\_index, row in enumerate(node\_matrix):

# break the rows into an indexed list. Each instance of col is an instance of the dictionary

for x\_index, col in enumerate(row):

# Store the old value

col['oldtemp'] = col['temp']

# BEGIN CRAZY ZONE

# nested if statements suck because they are hard to understand

if x\_index == 0:

# left side

if y\_index == 0:

# Lower Left Corner

above = node\_matrix[y\_index + 1][x\_index]['temp']

right = node\_matrix[y\_index][x\_index + 1]['temp']

# special calculation for this corner only

col['temp'] = (above + right + (2 \* h\_out \* delta\_x \* out\_temp / k\_brick)) / (2 \* ((h\_out \* delta\_x / k\_brick) + 1))

else:

# Left Column

if y\_index == rows\_y - 1:

# Top Outside Left Edge

above = node\_matrix[y\_index - 1][x\_index]['temp']

else:

# Outside Left Edge

above = node\_matrix[y\_index + 1][x\_index]['temp']

below = node\_matrix[y\_index - 1][x\_index]['temp']

right = node\_matrix[y\_index][x\_index + 1]['temp']

# similar to 3 other regions but this way seems to work better.

col['temp'] = (2 \* right + above + below + (2 \* h\_out \* delta\_x \* out\_temp / k\_brick)) / (2 \* ((h\_out \* delta\_x / k\_brick) + 2))

elif x\_index > 0 and y\_index == 0:

# Bottom Edge

if x\_index == cols\_x - 1:

# Outside Bottom Right Edge

right = node\_matrix[y\_index][x\_index - 1]['temp']

else:

# Outside Bottom Edge

right = node\_matrix[y\_index][x\_index + 1]['temp']

above = node\_matrix[y\_index + 1][x\_index]['temp']

left = node\_matrix[y\_index][x\_index - 1]['temp']

# see above column temp line

col['temp'] = (2 \* above + right + left + (2 \* h\_out \* delta\_x \* out\_temp / k\_brick)) / (2 \* ((h\_out \* delta\_x / k\_brick) + 2))

elif x\_index == ineg\_x and y\_index >= ineg\_y:

if y\_index == ineg\_y:

# Inside Corner

left = node\_matrix[y\_index][x\_index - 1]['temp']

right = node\_matrix[y\_index][x\_index + 1]['temp']

above = node\_matrix[y\_index + 1][x\_index]['temp']

below = node\_matrix[y\_index - 1][x\_index]['temp']

# this is a long equation

col['temp'] = (2 \* (left + below) + above + right + (2 \* h\_in \* delta\_x \* in\_temp / k\_brick)) / (2 \* (3 + (h\_in \* delta\_x / k\_brick)))

else:

# Inside Edge

if y\_index == rows\_y - 1:

# Top Inside Left Edge

above = node\_matrix[y\_index - 1][x\_index]['temp']

else:

# Inside Left Edge

above = node\_matrix[y\_index + 1][x\_index]['temp']

left = node\_matrix[y\_index][x\_index - 1]['temp']

below = node\_matrix[y\_index - 1][x\_index]['temp']

col['temp'] = (2 \* left + above + below + (2 \* h\_in \* delta\_x \* in\_temp / k\_brick)) / (2 \* ((h\_in \* delta\_x / k\_brick) + 2))

elif x\_index > ineg\_x and y\_index == ineg\_y:

# Bottom edge chimney side

if x\_index == cols\_x - 1:

# Inside Bottom Right Edge

right = node\_matrix[y\_index][x\_index - 1]['temp']

else:

# Inside Bottom Edge

right = node\_matrix[y\_index][x\_index + 1]['temp']

left = node\_matrix[y\_index][x\_index - 1]['temp']

below = node\_matrix[y\_index - 1][x\_index]['temp']

col['temp'] = (2 \* below + left + right + (2 \* h\_in \* delta\_x \* in\_temp / k\_brick)) / (2 \* ((h\_in \* delta\_x / k\_brick) + 2))

elif x\_index > ineg\_x and y\_index > ineg\_y:

# Out of bounds - these points are inside the chimney

col['temp'] = in\_temp

# do not count these points as part of the error total or min/max range.

col['OB'] = True

else:

# Filling pieces

if y\_index == rows\_y - 1:

# Inside Top Edge

above = node\_matrix[y\_index - 1][x\_index]['temp']

else:

# this is a core piece

above = node\_matrix[y\_index + 1][x\_index]['temp']

if x\_index == cols\_x - 1:

# Inside Right Edge

right = node\_matrix[y\_index][x\_index - 1]['temp']

else:

# Core piece

right = node\_matrix[y\_index][x\_index + 1]['temp']

left = node\_matrix[y\_index][x\_index - 1]['temp']

below = node\_matrix[y\_index - 1][x\_index]['temp']

col['temp'] = (above + below + left + right) / 4

# estimated error since last iteration

this\_error = abs((col['temp'] - col['oldtemp'])/col['temp'])\*100

if col['OB'] == False:

# if this point is within the geometry of the chimney then count the error

tot\_er += this\_error

# finding the minimum temp is easiest here because this dictionary is nested too far

if col['temp'] < min\_temp:

min\_temp = col['temp']

min\_index = [x\_index, y\_index]

if col['temp'] > max\_temp:

max\_temp = col['temp']

max\_index = [x\_index, y\_index]

# set the current error

c\_error = tot\_er

print('Leaving calculation loop with an estimated error of {:.2f} {} after {} iterations.'.format(c\_error, '%', c\_iter))

plot\_data = []

# This is a wonky way to do this but i wanted to put a border of outside air around the plot

for i in range(cols\_x + 1):

# the x-coordinates on the plot are generated from the node size

x\_coord = (i - 1) \* node\_size

for j in range(rows\_y + 1):

# store the y-coordinate from the node number and size

y\_coord = (j - 1) \* node\_size

# if the coordinate is on the outside, create a point that is equal to exterior air

if (i == 0) or (j == 0):

this\_temp = out\_temp

else:

# store the quarter cut matrix value in the temp variable for this run

this\_temp = node\_matrix[j-1][i-1]['temp']

# All of this nonsense is to avoid chopping off or duplicating the edge piece when

# putting the full size plot back together. Each section is fed in as a tuple so the values stay together.

# this ends up getting passed as (x, y, temp)

if (x\_coord == (cols\_x - 1) \* node\_size) and (y\_coord != (rows\_y - 1) \* node\_size):

plot\_data.append((x\_coord, y\_coord, this\_temp))

plot\_data.append((x\_coord, y\_outside-y\_coord, this\_temp))

if (y\_coord == (rows\_y - 1) \* node\_size) and (x\_coord != (cols\_x - 1) \* node\_size):

plot\_data.append((x\_coord, y\_coord, this\_temp))

plot\_data.append((x\_coord, y\_outside-y\_coord, this\_temp))

if (y\_coord == (rows\_y - 1) \* node\_size) and (x\_coord == (cols\_x - 1) \* node\_size):

plot\_data.append((x\_coord, y\_coord, this\_temp))

else:

plot\_data.append((x\_coord, y\_coord, this\_temp))

plot\_data.append((x\_outside - x\_coord, y\_outside - y\_coord, this\_temp))

plot\_data.append((x\_outside - x\_coord, y\_coord, this\_temp))

plot\_data.append((x\_coord, y\_outside - y\_coord, this\_temp))

# These list comprehensions store the data in vertical slices to be fed into the scatter plot function

x\_list = np.array([item[0] for item in plot\_data])

y\_list = np.array([item[1] for item in plot\_data])

z\_list = np.array([item[2] for item in plot\_data])

# Plot points and jack with settings until it looks good

plt.scatter(x\_list, y\_list, c=z\_list, cmap=cm.tab20b, marker='s', s=node\_size \* 10)

plt.plot(max\_index[0] \* node\_size, max\_index[1] \* node\_size, 'ro', label='Max. Temp = %.0f $^\circ$C' % max\_temp)

plt.plot(min\_index[0] \* node\_size, min\_index[1] \* node\_size, 'bo', label='Min. Temp = %.0f $^\circ$C' % min\_temp)

plt.legend()

# Add labels ant titles I guess

plt.title('Chimney Heat Distribution Map', y=1.05, x=0.33)

plt.xlabel('X - Axis Location (mm)')

plt.ylabel('Y - Axis Location (mm)')

plt.xticks(np.arange(-10, x\_outside + 20, 10),rotation='45')

plt.yticks(np.arange(-10, y\_outside + 20, 10))

# Dicking with settings until I get the plot area to show up the way I want to.

# I am not sure that all of these are even doing anything at this point but it

# prints a pretty graph so I am done messing with it.

plt.colorbar(shrink=0.75,label='Temperature ($^\circ$C)')

plt.grid(True, linestyle='dotted')

plt.axes().set\_aspect('equal','datalim')

# I think that this is the most important setting in deciding the plot area and shape of the

# axes. Sets padding for the plot. Pyplot is still easier to use than excel.

plt.subplots\_adjust(left=.15, right=0.99, bottom=.2, top=.85)

# SAVE IT AND YOU ARE DONE! YAY

plt.savefig('ME\_399\_chimney\_diagram.png', bbox\_inches='tight')

# By default the plot collapses after the show command, so this line must be the last one to execute

plt.show()

chimney()