1 PSet-6.py

```
import numpy as np
import matplotlib.pyplot as plt

fileName = "Problem-Set-6"

num_time_steps = 100
delta = 0.01            # 1 cm

# Grid points:
height = 200
width = 500

diameter = 50           # Diameter Cylinder

side_length = 0.30      # meters

# Constants
rho = 3000              # kg/m^3
c = 840                 # J/(kg*C)
h = 28                  # W/(m^2*C)  Convective Heat Transfer Coefficient
k = 5.2                 # W/(m*C)    Thermal Conductivity
alpha = k / (rho * c)   # m^2/s      Thermal Diffusivity

dt_1 = rho * c * (delta * delta) / (2 * h * delta + 4 * k)  # Characteristic time (
    convective boundary)
dt_2 = (delta  * delta) / (4 * alpha)                       # Characteristic time (
    internal grid)
dt = min(dt_1, dt_2)
Fo = alpha * dt / (delta * delta)                           # Fourier Number
Bi = h * delta / k                                          # Biot Number
T_initial = 10
T_right = 38
T_inf = 0
```

```
32
33 # Create array and initialize to T-initial
34 data = np.zeros((width, height)) + T_initial
35
36 # Set the right boundary to T_right
37 for j in range(height):
38     data[(width - 1), j] = T_right
39
40 history = [data.copy()]
41
42 error_flag = True
43 error_limit = 1e-4
44 while error_flag:
45     large_error_term_found = False
46
47     data_old = data.copy()
48
49     # Internal Nodes
50     for m in range(1, width - 1):
51         for n in range(1, height - 1):
52             data[m, n] = alpha * dt / (delta * delta) * (data_old[m + 1, n] + data_old[m
    - 1, n] + data_old[m, n + 1] + data_old[m, n - 1]) + (1 - 4 * alpha * dt / (delta *
   delta)) * data_old[m, n]
53
54     # Convective Boundary Nodes (Left)
55     for n in range(1, height - 1):
56         m = 0
57         data[m, n] = Fo * (2 * Bi * (T_inf - data_old[m, n]) + 2 * data_old[m + 1, n] +
   data_old[m, n + 1] + data_old[m, n - 1] - 4 * data_old[m, n]) + data_old[m, n]
58
59     # Insulated Boundary Nodes (Top)
60     for m in range(1, width - 1):
61         n = height - 1
62         data[m, n] = Fo * (2 * data_old[m, n - 1] + data_old[m - 1, n] + data_old[m + 1,
    n]) + (1 - 4 * Fo) * data_old[m, n]
```

```
63
64      # Exterior Corner with Convection Boundary
65      m = 0
66      n = height - 1
67      data[m, n] = 2 * Fo * (data_old[m + 1, n] + data_old[m, n - 1] - 2 * data_old[m, n]
        + 2 * Bi * (T_inf - data_old[m, n])) + data_old[m, n]
68
69
70      # Check if reached steady state
71      if not large_error_term_found:
72          error_term = abs(data[m, n] - data_old[m, n]) / data_old[m, n]
73          if (error_term <= error_limit):
74              error_flag = False
75          else:
76              error_flag = True
77              large_error_term_found = True
78
79      history.append(data.copy())
80
81  #print(len(history))
82
83  # Print the data in the console (readable format)
84  #print(np.rot90(data))
85
86
87  figNum = 1
88  plt.figure(figNum)
89  plt.axes().set_aspect('equal')
90  plt.style.use('classic')
91  data_graphable = np.flipud(np.rot90(data))
92  heatmap = plt.pcolor(data_graphable)
93
94  plt.text(0.5, -0.02, "T = " + str(T_initial) + "\N{DEGREE SIGN}C",
95           horizontalalignment='center',
96           verticalalignment='top',
```

```
          rotation=0,
          clip_on=False,
          transform=plt.gca().transAxes)
plt.text(0, 0.5, "Convective Boundary",
          horizontalalignment='right',
          verticalalignment='center',
          rotation=90,
          clip_on=False,
          transform=plt.gca().transAxes)
plt.text(0.5, 1, "Insulated Surface",
          horizontalalignment='center',
          verticalalignment='bottom',
          rotation=0,
          clip_on=False,
          transform=plt.gca().transAxes)
plt.text(1, 0.5, "T = " + str(T_right) + "\N{DEGREE SIGN}C",
          horizontalalignment='left',
          verticalalignment='center',
          rotation=270,
          clip_on=False,
          transform=plt.gca().transAxes)

plt.axis("off")

plt.xlim(0, width)
plt.ylim(0, height)

cbar = plt.colorbar(heatmap)
cbar.set_label("Temperature (\N{DEGREE SIGN}C)")
plt.clim(np.amin(data), np.amax(data))

plt.savefig(fileName + "/images/" + fileName + "-Figure.png")
plt.show()
```