1 PSet-5.py

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

fileName = "Problem-Set-5"

num_time_steps = 100
delta = 0.01            # 1 cm

# Grid points:
height = 30
width = 30

# 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

# Create array and initialize to T-initial
data = np.zeros((width, height)) + T_initial

```

```python
# Set the right boundary to T_right
for j in range(height):
    data[(width - 1), j] = T_right

history = [data.copy()]

error_flag = True
error_limit = 1e-4
while error_flag:
    large_error_term_found = False

    data_old = data.copy()

    # Internal Nodes
    for m in range(1, width - 1):
        for n in range(1, height - 1):
            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]

    # Convective Boundary Nodes (Left)
    for n in range(1, height - 1):
        m = 0
        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]

    # Insulated Boundary Nodes (Top)
    for m in range(1, width - 1):
        n = height - 1
        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]

    # Exterior Corner with Convection Boundary
    m = 0
    n = height - 1
```

```
    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]


    # Check if reached steady state
    if not large_error_term_found:
        error_term = abs(data[m, n] - data_old[m, n]) / data_old[m, n]
        if (error_term <= error_limit):
            error_flag = False
        else:
            error_flag = True
            large_error_term_found = True

    history.append(data.copy())

#print(len(history))

# Print the data in the console (readable format)
#print(np.rot90(data))

figNum = 1
plt.figure(figNum)
x = np.linspace(0, 1, height)
y = history[num_time_steps][0, :]
plt.plot(x, y)
plt.xlabel("Position Along Left Convective Boundary (Normalized)")
plt.ylabel("Temperature (\N{DEGREE SIGN}C)")
plt.suptitle("Temperature Along the Left Convective Boundary")
plt.title("Bottom to Top; 100 Time Steps")
plt.xlim(0, 1)
plt.savefig(fileName + "/images/" + fileName + "-Figure-" + str(figNum) + ".png")
plt.show()

figNum = 2
plt.figure(figNum)
```

```python
x = np.linspace(0, 1, width-1)
y = history[num_time_steps][0:(width-1), height - 1]
plt.plot(x, y)
plt.xlabel("Position Along Insulated Surface Boundary (Normalized)")
plt.ylabel("Temperature (\N{DEGREE SIGN}C)")
plt.suptitle("Temperature Along the Insulated Surface Boundary")
plt.title("Left to Right; 100 Time Steps")
plt.xlim(0, 1)
plt.savefig(fileName + "/images/" + fileName + "-Figure-" + str(figNum) + ".png")
plt.show()

figNum = 3
plt.figure(figNum)
history_length = len(history)
y = []
for state in history:
    y.append(state[0, 15])
x = dt * np.linspace(0, (len(history) - 1), len(history))
plt.plot(x, y)
plt.xlabel("Time (s)")
plt.ylabel("Temperature (\N{DEGREE SIGN}C)")
plt.suptitle("Temperature At Specific Point Until Steady State Reached")
plt.title("Halfway Up Convective Boundary")
plt.xlim(0, max(x))
plt.savefig(fileName + "/images/" + fileName + "-Figure-" + str(figNum) + ".png")
plt.show()



figNum = 4
plt.figure(figNum)
plt.axes().set_aspect('equal')
plt.style.use('classic')
data_graphable = np.flipud(np.rot90(data))
heatmap = plt.pcolor(data_graphable)
```

```
plt.text(0.5, -0.02, "T = " + str(T_initial) + "\N{DEGREE SIGN}C",
         horizontalalignment='center',
         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-" + str(figNum) + ".png")
plt.show()
```