Final_Project\Adobe_Hut.py

```
# By: Daniel Shklyarman
#     100851439

# A simplified model of heat transfer
# in an adobe hut

# The adobe hut is made out of brick and has no windows
# It is assumed to be small enough that it always receives
# an equal amount of sunlight/heat on all sides so that there
# is no angular dependence

# Imports
import numpy as np
import math
import matplotlib.pyplot as plt
import time

# Physical Constants

# | Name     | Description                                       | Value   | Units         |
# |---------|--------------------------------------------------|---------|---------------|
# | k_wall   | thermal conductivity of brick                     | 0.75    | W m^-1 K^-1   |
# | k_air    | thermal conductivity of air                       | 4e-2    | W m^-1 K^-1   |
# | rho_wall| density of the wall/brick                         | 1.8e+3  | kg m^-3       |
# | rho_air  | density of the air                                | 1.2     | kg m^-3       |
# | cp_wall  | heat capacity of the wall/brick                   | 840     | J kg^-1 K^-1  |
# | cp_air   | heat capacity of the air                          | 700     | J kg^-1 K^-1  |
# | R_b      | radius of the hut (from center to outside         | 3.0     | m             |
# | R_a      | radius of the hut (from center to internal wall)  | 2.7     | m             |
# | h        | heat transfer coefficient of brick                | 2.0     | W  m^-2  K^-1 |
# | tau      | time scale (12 hours)                             | 4.32e+4 | s             |
# |---------|--------------------------------------------------|---------|---------------|

# number of 'lines' through the hut
n = 50

# Parameters

# P1 = [k/(rho*cp)]_wall * [tau/(R_b)^2]
P1 = 2.4e-3
# P2 = [k/(rho*cp)]_air * [tau/(R_b)^2]
P2 = 0.23
# P3 = R_a / R_b
P3 = 0.9
# P4 = k_air / k_wall
P4 = 5.3e-2
# P5 = h * R_b / k_wall
P5 = 8

# Width of a 'line'
# r0 = 1
# r0 - (N-1)*delta_r = r_(N-1) = 0
h = 1/(n-1)
```

```

# External Temperature as a function of time
def u_ext(t):
    return max(0, np.sin(np.pi * t))

# Array of 'lines'
# the phi at Phi[0] is the external edge
# progressive phi's go closer to the center
Phi = np.zeros(n)

# Set temperatures at time = 0
T_init = 0
for i in range(len(Phi)):
    Phi[i] = T_init

# Number of measurement taken per tau
# aka dt
resolution = 1000

# Number of time steps
it_number = 10000

# Array to store temperature values over time
Phi_history = np.zeros([n,it_number-1])

# Finite Difference estimate of the derivative wrt time
def dphi_dt(temp_vec, n, i, parameter):
    temp_i_minus_1 = temp_vec[0]
    temp_i = temp_vec[1]
    temp_i_plus_1 = temp_vec[2]

    curr_r = ((n-1-i)*h)**2

    dphi_dt = parameter * (
                      ((n-1)**2)*(temp_i_minus_1 - 2*temp_i + temp_i_plus_1) +
                      ((n-1)/curr_r)*(temp_i_plus_1 - temp_i_minus_1)
                      )

    return dphi_dt

# Runge-Kutta method, only returns the increment not
# the full new estimate, hence the '+=' in the loop
def Runge_Kutta(temp_vec, dphi_dt, h, i, n, parameter):
    k1 = dphi_dt(temp_vec, n, i, parameter)

    k2_temp_vec = temp_vec + [temp*(h*k1)/2 for temp in temp_vec]
    k2 = dphi_dt(k2_temp_vec, n, i, parameter)

    k3_temp_vec = temp_vec + [temp*(h*k2)/2 for temp in temp_vec]
    k3 = dphi_dt(k3_temp_vec, n, i, parameter)

    k4_temp_vec = temp_vec + [temp*(h*k3) for temp in temp_vec]
    k4 = dphi_dt(k4_temp_vec, n, i, parameter)

    return (h/6) * (k1 + 2*k2 + 2*k3 + k4)

```

```python
# Live Plotting stuff
# x = np.linspace(0,50,50)
# y = Phi

# plt.ion()

# fig,ax = plt.subplots(figsize=(10,5))
# line, = ax.plot(x,y)

# plt.title("Heat in an Adobe Hut")

# plt.xlabel("Lines from the outside in")
# plt.ylabel("Temperature")

# plt.ylim(-1, 2)

# The actual fancy stuff
for t in range(1, it_number):
    for i in range(len(Phi)):
        # BC3
        if i == 0:
            Phi[i] = u_ext(t/resolution) + (u_ext(t/resolution) - Phi[1])/(2*P5*h)
            Phi_history[i,t-1] = Phi[i]

        # Heat equation in wall
        elif i < n*P3:
            temp_vec = [Phi[i-1],Phi[i],Phi[i+1]]
            Phi[i] += Runge_Kutta(temp_vec, dphi_dt , h, i, n, P1)
            Phi_history[i,t-1] = Phi[i]

        # BC1/2
        elif i == (n - n*P3):
            Phi[i] = (P4*Phi[i+1] - Phi[i-1]) / (P4-1)
            Phi_history[i,t-1] = Phi[i]

        # Heat equation in the air
        elif i > n-n*P3 and i < n-1:
            temp_vec = [Phi[i-1],Phi[i],Phi[i+1]]
            Phi[i] += Runge_Kutta(temp_vec, dphi_dt , h, i, n, P2)
            Phi_history[i,t-1] = Phi[i]

        # BC4
        else:
            Phi[i] = Phi[i-1]
            Phi_history[i,t-1] = Phi[i]

    if t < 100:
        print(Phi_history[:,t-1][:5])

    # Also Live Plotting Stuff
    # new_y = Phi

    # line.set_xdata(x)
    # line.set_ydata(new_y)

    # fig.canvas.draw()
```

```python

    # fig.canvas.flush_events()

    # time.sleep(0.01)


# Ask user for what plot they want
plot = input('What kind of plot would you like? \n * 2              = 2d plot \n * 3 = 3d plot \n * [other input] = no plot \n')

# Create 2D plot
if plot == '2':
    # Ask user if they want the legend since the legend is very crowded due to havin n lines
    legend = input("Legend or no legend? (Y/N , other input assumes N)")

    # Create Plot
    for i in range(0, n):
        plt.plot(np.linspace(0,it_number, it_number-1), Phi_history[i], label=f'Phi{i}')

    if legend == 'Y':
        plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.15), ncol=10, fancybox=True, shadow=True)

    # Make the plot a bit nicer to look at/read
    plt.xticks(np.arange(0,it_number, step=resolution), np.arange(0, it_number/resolution, step=1))
    plt.xlabel('Time (1 = 12 hours)')
    plt.ylabel('Nondimensionalized Temperature (T-T_0)/(delta_T)')

    plt.show()

# Create 3D plot
elif plot == '3':
    # Create Plot
    ax = plt.figure().add_subplot(projection='3d')
    for i in range(0, n):
        ax.plot(np.linspace(0,it_number, it_number-1), Phi_history[i], zs=i, label=f'Phi{i}')

    # Make the plot a but nicer to look at/read
    ax.set_xticks(np.arange(0,it_number, step=resolution), np.arange(0, it_number/resolution, step=1))
    ax.set_xlabel('Time (1 = 12 hours)')
    ax.set_ylabel('(T-T_0)/(delta_T))')
    ax.set_zlabel('Shell number (0 is outermost shell)')

    plt.show()

else:
    print("Have a nice day.")
```