On Modelling the Two-Dimensional Heat Equation in Python With a Source and Various Boundary Conditions

Demetrios Lambrou

Vanier Cégep/College, Montréal, Quebec, H4L 3X9, Canada
2127464@edu.vaniercollege.qc.ca

Submitted: 2023-05-17

Abstract

The code for the two-dimensional heat equation for Dirichlet and Neumann boundary conditions was successfully written in Python. A room of 10 metres by 10 metres at 293 K with three walls at 280 K (Dirichlet condition),one window diffusing heat (Neumann), and a source of 500 W was programmed in Python using the scientific computation package Numpy. Using the finite difference method for numerically computing partial differential equations, the heat flow was simulated for 100 time steps. The output animation was done using the Matplotlib Python package and saved as a GIF.

1 Introduction

A differential equation is an equation relating an unknown function and one or more of its derivatives [10]. Ordinary differential equations (ODE) are the simplest types of differential equations since they only involve a function of one variable [10]. ODEs require initial conditions in order to be solved for specific functions. Initial conditions are the initial values that a function and its derivatives have at specific points in space. In general, an nth-order ordinary differential equation requires the function and its n-1 derivatives to be specified at some points in space to be solved for a specific original function [10]. While ordinary differential equations are good for modelling simple systems, like radioactive decay and compound interest [10], they are less suitable for more complex ones. This is where partial differential equations come into the mix.

A partial differential equation (PDE) is a differential equation that involves a function of multiple variable [10]. Partial differential equations have a plethora of applications in physics and engineering like the Schrödinger equation and the heat equation [5]. Much like an ordinary differential equation, to solve a partial differential equation for a specific function, certain conditions need to be imposed on the equation at hand. There are two such conditions: initial conditions and boundary conditions [9]. An initial condition indicates the physical state of a system at a particular time [9]. Any PDE that models a physical system is only valid on a specific domain. This is referred to as the boundary. Boundary conditions are imposed on the defined boundary of the PDE. There are three common boundary conditions: Dirichlet, Neumann and Robin. The Dirichlet condition is a boundary condition that imposes a function on the boundary. A Neumann condition is a boundary condition that specifies the value of the derivative of a function on the boundary. The Robin condition specifies the sum of the derivative of a function with a constant multiplied by a function. [9]

Laplacians are critical to the understanding of the heat equation. The Laplacian is the divergence of the gradient of a function [9]. In Cartesian space (i.e. x-y-z coordinates) , the gradient is the partial derivative of a function with respect to each of its components [1].

$$\nabla f = \frac{\partial f}{\partial x}\hat{\imath} + \frac{\partial f}{\partial y}\hat{\jmath} + \frac{\partial f}{\partial z}\hat{k} \tag{1}$$

The divergence is the sum of the partial derivatives with respect to each component of a function [1].

$$\nabla \cdot \mathbf{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z} \tag{2}$$

Therefore, the Laplacian is the sum of the second-order partial derivatives of a function with respect to each of its components [9].

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \tag{3}$$

The heat equation is a second-order partial differential equation that describes the diffusion of heat in a given area. The constant alpha is a combination of other constants. Specifically, it is the thermal conductivity of the material divided by the density and the specific heat capacity of the material. The thermal conductivity of an object is its ability to transfer heat. The specific heat capacity of an object is the amount of heat required on one unit of mass to raise the temperature by one unit. The density, of course, is the mass divided by the volume. Modelling the heat equation is important since it allows one to examine the flow of heat on various surfaces, objects and materials. [7].

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T \tag{4}$$

Fourrier's Law is another important concept to understand. It is commonly referred to as thermal conduction or heat flux. Fourrier's Law states that the heat flux q is proportional to the magnitude of the temperature gradient and opposite in sign. The proportionality constant k is the thermal conductivity of an object. The negative sign indicates the flow of heat from a higher temperature to a lower temperature. [7]

$$\vec{q} = -k \nabla T \tag{5}$$

This paper explores a simpler version of the heat equation. Specifically, it analyzes the two-dimensional case of the heat equation with Dirichlet and Neumann boundary conditions. In the case of the heat equation, the Dirichlet condition is the temperature at the boundary whereas the Neumann condition is the heat flux at the boundary [6]. In order to solve the two-dimensional heat equation with the prescribed boundary conditions, the finite difference approximation of the Laplacian in two-dimensions [6] was implemented in Python [8] using Numpy [3] and visualized using Matplotlib [4]. This paper will explain the process of programming the finite difference method for the heat equation, implementing boundary conditions, and visualizing the final results in Python.

2 Methods

2.1 Setup

Before any modelling can occur, the environment needs to be set up. Since the simulation is modeling a room filled with air, certain initial values need to be determined. These values are the density, the specific heat capacity and the thermal conductivity. Density is of course the mass per unit volume. The thermal conductivity of an object is its ability to transfer heat. The specific heat capacity of an object is the amount of heat required on one unit of mass to raise the temperature by one unit. The heat transfer coefficient is the ratio between [7] It is important to note that these values were provided courtesy of Prof. Nik Provatas and his graduate students. The exact values used are in Appendix A. Other values, like the room's temperature (293 K), the walls' temperature (280 K) and the outdoor temperature (200 K) are also initialized. The wattage of the source is set to 500 W and the source is set to have a length of 2 m. It is important to note that these values are "normalized" into two-dimensional space. This is done in the following way. Firstly, the length of the space heater is divided by the smallest increment, Δx. This new constant is called m.

Next, the wattage is divided by the square of m multiplied by the cube of Δx. This new number is the "normalized" source power.

The Λ constant requires a bit of algebra to explain. It is used exclusively for the window and is the smallest increment, delta x, multiplied by the heat transfer coefficient then divided by the thermal conductivity. It is derived from Fourier's Law as follows:

$$\begin{aligned} k\frac{\Delta T}{\Delta x} &= h(T_{wall} - T_{out}) \\ k\frac{T_n - T_{n-1}}{\Delta x} &= h(T_{wall} - T_{out}) \\ T_n - T_{n-1} &= \frac{h\Delta x}{k}(T_{wall} - T_{out}) \\ T_n - T_{n-1} &= \Lambda(T_{wall} - T_{out}) \\ T_n &= \frac{T_{n-1} + \Lambda T_{out}}{1 + \Lambda} \end{aligned} \tag{6}$$

This equation shows how the heat equation will be updated to account for the window.

The room is modeled using a two-dimensional Numpy array with each index in the array set to the value of the room temperature variable. [3] The three chosen temperatures to plot are stored using regular Python arrays. [8]

2.2 Finite-Difference Method & Boundary

The finite-difference method is a way of numerically approximating a differential equation. It can be used for ordinary of partial differential equations yet is primarily used for the latter. It can be expressed in the following way:

$$T_{i,j} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2} \tag{7}$$

Note that the equation above replaced the u from the textbook with T, and replaced the h in first denominators with the square of Δx and the h in the second denominator with the square of Δy. This form presented above is not the completed heat equation, however. While this may model the Laplacian present in the heat equation, the equation above still needs to be multiplied by constant α mentioned in the introduction and the small increment in time Δt. [6] It is important to note that Δt is subject to the following limit:

$$\Delta t = \frac{\Delta x^2}{2\alpha} \tag{8}$$

Any time increment which exceeds this value will cause the code to break. As such, the code selects a Δt which is equal to one half of the expression above. This formula was provided by the graduate students of Prof. Provatas.

The final part that needs to be included in the source. Adding the source into the equation is rather straightforward. The "normalized" power of the source is multiplied by α and Δt just as the finite-difference method. However, this value tends to be rather small and requires a ludicrously high wattage to see results. As such, the value is multiplied by a corrective term. This term is a two-dimensional Gaussian distribution with a standard deviation σ.

Lastly, to keep the three walls at a constant temperature, two for loops are added at the end of the finite-difference method. These loops set the indices of the wall back to their assigned temperature. These loops also apply equation 6 to the window to simulate the cold air blowing in.

2.3 Figures & Animation

The model was animated two different ways using Matplotlib. The first version of the animation uses the pyplot package of Matplotlib to draw the temperature array on a grid. The specific grid used is the pcolormesh grid with gouraud shading, jet colormap and a legend from 250 K to 320 K. Every 0.001 s and for 40 loops, the array being displayed is updated using the finite-difference method mentioned above. The plot is then displayed to the screen using the show fucntion of the pyplot package.

The second animation is not displayed to the screen but instead saves a GIF file in the same folder as the code. This is done using the animation package of Matplotlib. Extracting each frame of the GIF results in Figure 1.

The final element being plotted is a graph of the temperature of three different points on the grid. Three empty Python arrays are initialized at the beginning of the program. Every loop of the main code block, each array gets the new temperature value of the specified points appended to the array. Once the main code block is completed, the array are henceforth filled with the temperature values of three points. These arrays are plotted versus the number of time the simulation ran and return Figure 2. [4]

3 Results

3.1 Heat Animation

The following images are the first couple frames of the heat equation animation. They were extracted from the GIF that was created by the program. We can see how the source slowly starts to heat up the room

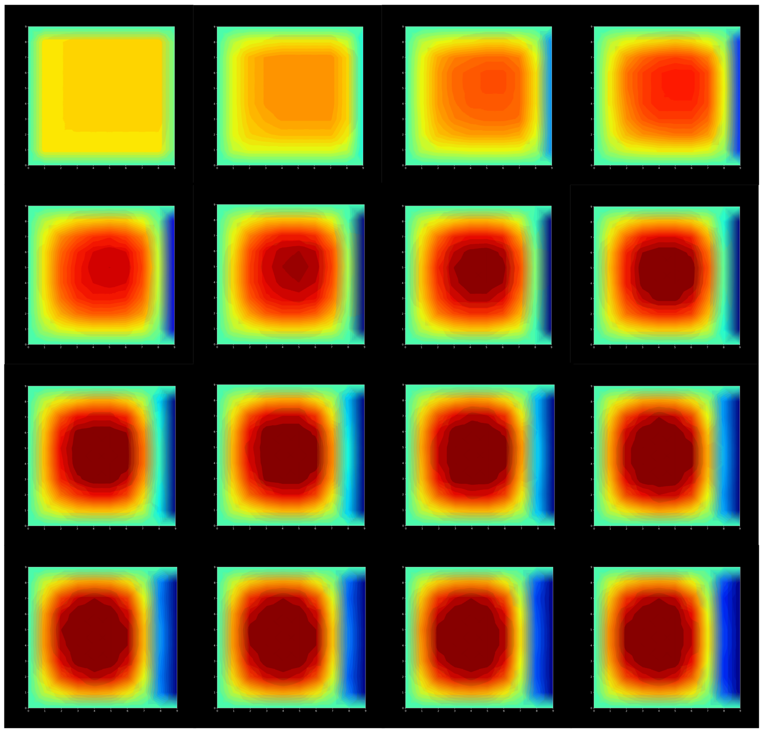

Figure 1: First 16 Frames of the Animation

as the window lets in the cool air. Obviously, the redder the colour, the hotter the area, and the bluer the colour, the cooler the area. Eventually, an equilibrium is reached a no more heating or cooling occurs.

3.2 Equilibrium Temperature

The following graph is a plot of the temperature of three different points vs the number of times the model has been run. We can see how different points in the room reach the equilibrium temperature after a different

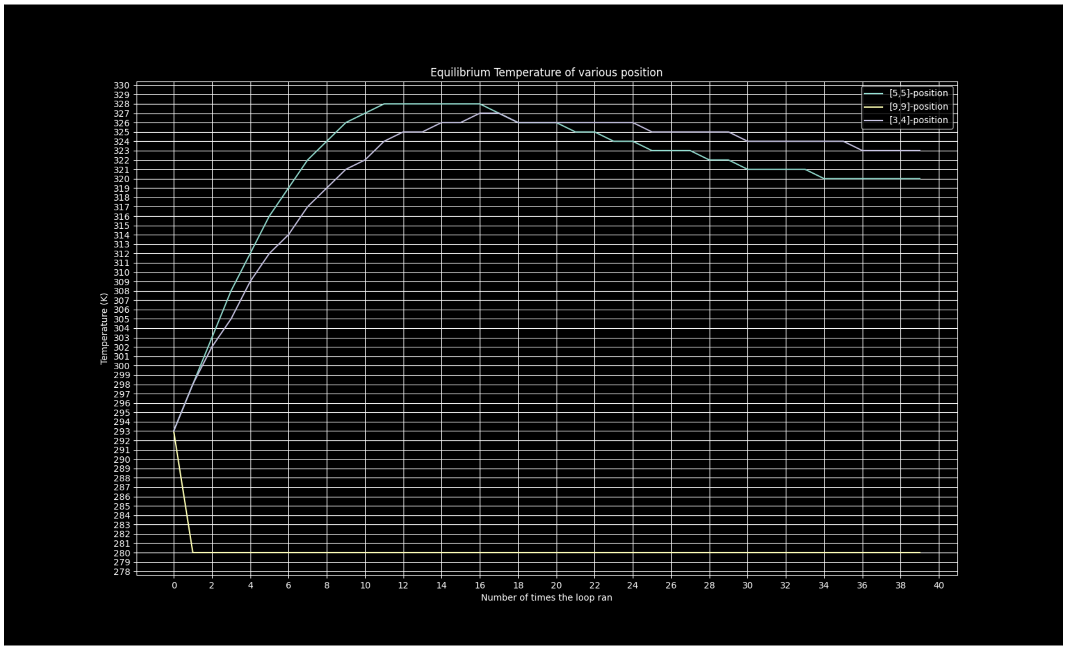

Figure 2: Temperature of the [5,5], [9,9] and [3,4] positions

number of loops. Those closer to the heat source or the windoe reach it quicker compared to those farther away.

4 Conclusion

In summary, this model that run a simulation of the 2D heat equation rather quickly. The specific case under study was a 10 m by 10 m room at 293 K. Three out the four walls of the room were set at a constant temperature of 280 K and the final wall mas made into a window allowing 200 K air to blow into the room. In effect, the three walls are acting as insulators. The values for the specific heat capacity, the thermal conductivity and the heat transfer coefficient were set to those of air. Finally, a 500 W 2 m by 2m heat source was placed in the center of the room. In order to use the values of the heat source, they needed to be "normalized" to two dimensions. Once the new "normalized" wattage and length were computed, they were fed into the model which employed the finite difference. In the end, an animation was produced and a graph of three points reaching their respective equilibrium temperature was presented.

There are some limitations with model, however. Chief among them is the fact that the model is simulating a three-dimensional room in two dimensions. While the two-dimensional case was chosen due to its simplicity, it obviously fails to take into account the flow of heat in the third dimension. The obvious solution to this is to model the three-dimensional case of the heat equation, but there is a minor issue. Visualizing the flow of heat in a 3D room is rather difficult to see compared to a 2D room. This issue can be fixed by showing H (height of the room) different 2D slices of room to show how the heat is flowing in the third dimension. This limitations do not mean the model is useless. This model can be appropriated to visualize the flow of heat on a thin sheet since, for all intents and purposes, a thin sheet is two-dimensional. The material's specific parameters, like specific heat capacity and thermal conductivity, can be provided and the finite-difference method can still be used. All in all, despite its simplicity, the model is still relatively advanced.

The explicit finite-difference method implemented in this paper can be improved by using the Crank-Nicolson method, which takes half of the sum in finite-difference method. The Crank-Nicolson would be a very ideal algorithm for the heat equation since it is commonly used for diffusion equations, of which the heat equation is one. [2]

5 Acknowledgments

I would like to thank Prof. Nik Provatas of the Physics Department of McGill University and his graduate students for their help throughout the project.

References

[1] Jeffrey R. Chasnov. *Vector Calculus for Engineers.* Jeffrey R. Chasnov, 2022.

[2] J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society*, 43(1):50–67, 1947. `doi:10.1017/S0305004100023197`.

[3] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. `doi:10.1038/s41586-020-2649-2`.

[4] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90, 2007.

[5] Victor Ivrii. *Partial Differential Equations.* University of Toronto, 2021.

[6] Volker John. *Numerical Methods for Partial Differential Equations.* Weierstrass Institute, 2013.

[7] John H. Lienhard IV and John H. Lienhard V. *A Heat Transfer Textbook.* Phlogiston Press, 2019.

[8] Python Core Team. *Python: A dynamic, open source programming language.* Python Software Foundation, 2019. URL: `https://www.python.org/`.

[9] Walter A. Strauss. *Partial Differential Equations: An Introduction.* Wiley, 2009.

[10] William F. Trench. *Elementary Differential Equations.* Trinity University, 2013.

A Appendix: Heat Equation Code

The following is the code for the model of the heat equation. Commenting or un-commenting the parts mentionned in the code will change the results displayed. Currently, the animation is using a shader to make the final product look more visually. appealing. Removing the ”shading = ’gouraud’” from the regular plot and the animation will allow for a clearer picture of the heat flow.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
import time
start_time = time.time()

plt.style.use('dark_background')

#size of room
room_x = 10 # (m)
room_y = 10 # (m)

#Constants
sigma = 0.1 #standard deviation of gaussian distribution
rho = 1.225 #denisty of air (kg/m^3)
cp = 1000 #specific heat capacity of air (J/kg*K)
k = 10**(-2) #Thermal conductivity of air (W/m*K)
h = 1 #heat transfer coefficient (W/m^2*K)
delta_x = 0.001 #small change in x (m)
delta_y = 0.001 #small chage in y (m)
lamda = delta_x*h/k
T_out = 200 #outside temp (K)
w_h = 500 #power of space heater (W)
L = 2 #dimension of heater (m)
m = L/delta_x
total_time = 40 #repetitions of simulation
source_power = w_h/(m**2 * delta_x**3)
source_x = room_x/2 #x-coordinate of source
source_y = room_y/2 #y-coordinate of source
b_temp_1 = 280 #temp of boundary 1 (K)
b_temp_2 = 280 #temp of boundary 2 (K)
b_temp_3 = 280 #temp of boundary 3 (K)
room_temp = 293 #temp of room (K)
delta_t = 1/2 * (delta_x*delta_x)/(2*k/(rho*cp)) #time interval (s)

#Temperature values
T = np.full((room_x,room_y), room_temp)


fig, ax = plt.subplots(figsize=(room_x,room_y))

#Uncomment for equilibrium temp at various points
# t1 = []
# t2 = []
# t3 = []
# ts = np.empty(100, total_time)

#Uncomment for animation
# ims = []
```

```
for t in range(total_time):
    # t1.append(T[int(room_x/2),int(room_y/2)])
    # t2.append(T[int(9),int(9)])
    # t3.append(T[int(3),int(4)])
    for  j in range(1,room_y-1):
        for i in range(1,room_x-1):
            #Equation
            Eq_1 = (T[i-1,j]-2*T[i,j]+T[i+1,j])/(delta_x*delta_x)
            Eq_2 = (T[i,j-1]-2*T[i,j]+T[i,j+1])/(delta_y*delta_y)
            T[i,j] = T[i,j] + delta_t*(k/(rho*cp))*(Eq_1+Eq_2) +
                delta_t*source_power*(np.exp(-(delta_x*delta_x*(i-source_x)**2)/sigma -
                (delta_y*delta_y*(j-source_y)**2)/sigma))/(rho*cp)

    for i in range(0,room_x):
        T[i, room_y-1] = (T[i, room_y-1] + lamda*T_out)/(1+lamda)
        T[i, 0] = b_temp_1

    for j in range(0,room_y):
        T[room_x-1, j] = b_temp_2
        T[0, j] = b_temp_3


    plt.clf()
    plt.pcolormesh(T, cmap=plt.cm.jet, vmin=250, vmax=320, shading='gouraud')
    plt.colorbar()
    plt.pause(0.001)

    #Uncomment for animation
    # im =  ax.pcolormesh(T, cmap=plt.cm.jet, vmin=250, vmax=320, shading='gouraud')
    # ims.append([im])

#Uncomment for animation
# ani = animation.ArtistAnimation(fig, ims)
# ani.save('out.gif', writer='imagemagick')
# plt.close()
# ani

#Uncomment for equilibrium temp at various points
# plt.title("Equilibrium Temperature of various position")
# plt.xlabel("Number of times the loop ran")
# plt.ylabel("Temperature (K)")
# plt.xticks(range(0, 100, 2))
# plt.yticks(range(200, 400, 1))
# plt.grid()
# plt.plot(t1)
# plt.plot(t2)
# plt.plot(t3)
# plt.legend(["[5,5]-position","[9,9]-position","[3,4]-position"])

#Uncomment for default plot
print(time.time()-start_time)
plt.show()
```