Numerical Solution of the 2 - Dimensional Heat Equation using Python: Implementation and Analysis

Dr. Vivek Parkash

Assistant Professor of Mathematics, Dyal Singh College, Karnal (Haryana), India
Email: *lethal007[at]hotmail. com*

Abstract: ***Python is widely acknowledged for its formidable ability in handling intricate mathematical challenges, including the simulation of the two - dimensional heat equation-a critical partial differential equation governing temperature distribution evolution in a specified region over varying time steps. The versatility of Python, coupled with its extensive library support, positions it as a premier choice for conducting numerical simulations and analyses. This encompasses solving the heat equation through diverse methods such as finite differences, finite elements, or spectral methods. This paper is focused on exploring the finite difference method as a tool to solve the two - dimensional heat equation. It investigates the temperature distribution within a predefined domain across a range of parameter configurations. The process involves discretizing the partial differential equation in both spatial dimensions (x, y) and progressing in time steps t using an iterative approach.***

Keywords: python, mathematics, finite difference, heat equation, temperature distribution

1. Introduction

The mathematics behind the two - dimensional heat equation involves differential calculus for expressing how temperature changes over time and space, and numerical methods for approximating solutions due to the complexity of analytical solutions. The two - dimensional heat equation is a partial differential equation that describes the distribution of heat (or temperature) in a two - dimensional domain over time. It is a fundamental equation in the field of heat transfer and has applications in various areas such as physics, engineering, and environmental science.

The heat equation in two dimensions has the form

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

This equation signifies how the temperature *u (x, y, t)* changes with respect to time *t* in a region of space having coordinates *(x, y)*, where *u (x, y, t)* represents the temperature at given point (*x*, *y*) and time *t*, α represents the thermal diffusivity, having positive value, $\frac{\partial u}{\partial t}$ is rate of change of temperature with time *t*, $\frac{\partial^2 u}{\partial x^2}$ *t* , is second order partial derivative of *u* with respect to , *x*, $\frac{\partial^2 u}{\partial y^2}$ is second order partial derivative of *u* with respect to *y*. As α is a constant therefore from equation (1), we can say that the rate of change of temperature at any point in the domain is proportional to the Laplacian of the temperature field *u (x, y, t)*, where the Laplacian $\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ denotes the variation of *u (x, y, t)* in space (spatial variation). We need boundary conditions to solve the heat equation. Boundary conditions specify the temperature on the boundary of the region under consideration, the initial value of temperature taken as zero. Python's flexibility and the availability of libraries like NumPy and Matplotlib enable researchers and engineers to effectively simulate and analyse complex physical phenomena such as heat diffusion and numerically solving partial differential equations like the two dimensional heat equation.

2. Methodology

Discretization is the process of representing continuous data or variables as discrete values or categories. We use the finite difference method to discretize this equation in both spatial dimensions and in time as below:

- **Spatial Discretization**:

We divide the spatial domain into a grid, having step sizes Δx and Δ*y*. Suppose $u_{i,j}^{n}$ represents the value of temperature at (*i*, *j*)th place of the grid and *n*th time step.

- **Spatial Discretization**: We use central difference for calculation of second order derivatives of *u (x, y, t)* as below:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u^n{}_{i+1,j} - 2u^n{}_{i,j} + u^n{}_{i-1,j}}{(\Delta x)^2}$$ and

$$\frac{\partial^2 u}{\partial y^2} \approx \frac{u^n{}_{i,j+1} - 2u^n{}_{i,j} + u^n{}_{i,j-1}}{(\Delta y)^2}$$

- **Temporal Discretization**: We use forward difference for calculating time derivative:

$$\frac{\partial u}{\partial t} \approx \frac{u^{n+1}{}_{i,j} - u^n{}_{i,j}}{\Delta t}$$

The finite difference method yields the values of temperature at spatial points given by

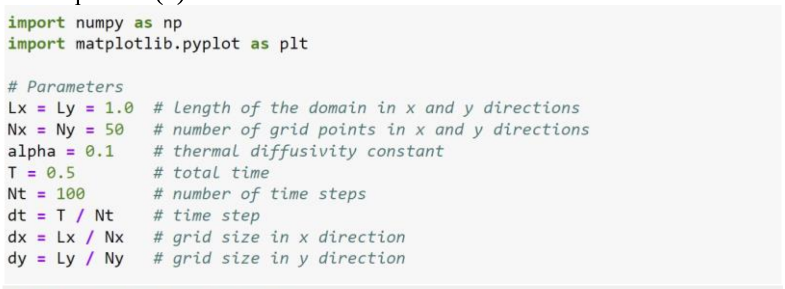
$$u^{n+1}_{i,j} = u^{n}_{i,j} + \frac{\alpha\Delta t}{(\Delta x)^2}\left(u^{n}_{i+1,j} - 2u^{n}_{i,j} + u^{n}_{i-1,j}\right) + \frac{\alpha\Delta t}{(\Delta y)^2}\left(u^{n}_{i,j+1} - 2u^{n}_{i,j} + u^{n}_{i,j-1}\right) \quad \text{....... (2)}$$

Where $u^{n}_{i,j}$ is the temperature at spatial grid point (*i, j*) and time step *n*, $u^{n+1}_{i,j}$ is the temperature at spatial grid point (*i, j*) and time step *n+ 1*.

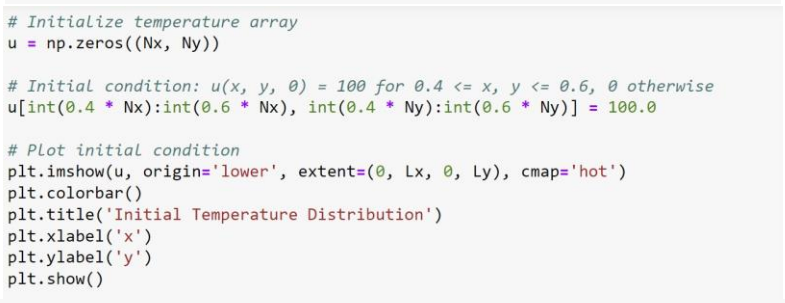
Steps involved in the Python script are:

- Initialization of temperature array: We initialize the temperature array *u (x, y, t)* after assigning the initial condition on it.
- We process the loop over required time steps and within each time step, the temperature is updated using the finite difference formula given in equation (2)
- It is assumed that the temperature remains fixed at the boundaries of our domain.
- The initial and final temperature distributions are then plotted using matplotlib **(Fig - 1 & Fig - 2)**

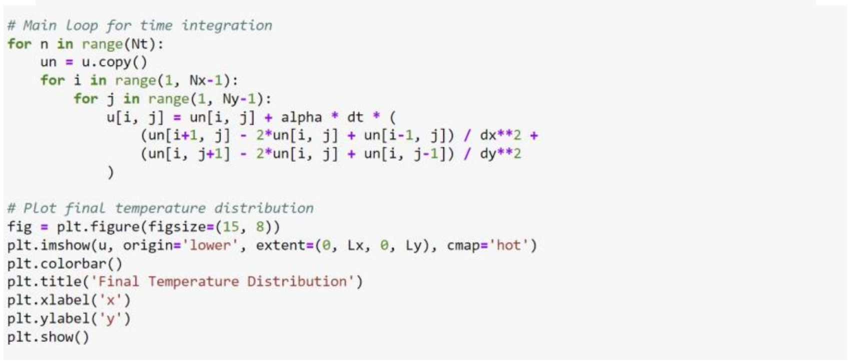
Given below is the Python script that illustrates the finite difference method to solve the two - dimensional heat equation:



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

# Parameters
Lx = Ly = 1.0   # length of the domain in x and y directions
Nx = Ny = 50    # number of grid points in x and y directions
alpha = 0.1     # thermal diffusivity constant
T = 0.5         # total time
Nt = 100        # number of time steps
dt = T / Nt     # time step
dx = Lx / Nx    # grid size in x direction
dy = Ly / Ny    # grid size in y direction
```



```
# Initialize temperature array
u = np.zeros((Nx, Ny))

# Initial condition: u(x, y, 0) = 100 for 0.4 <= x, y <= 0.6, 0 otherwise
u[int(0.4 * Nx):int(0.6 * Nx), int(0.4 * Ny):int(0.6 * Ny)] = 100.0

# Plot initial condition
plt.imshow(u, origin='lower', extent=(0, Lx, 0, Ly), cmap='hot')
plt.colorbar()
plt.title('Initial Temperature Distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



```
# Main loop for time integration
for n in range(Nt):
    un = u.copy()
    for i in range(1, Nx-1):
        for j in range(1, Ny-1):
            u[i, j] = un[i, j] + alpha * dt * (
                (un[i+1, j] - 2*un[i, j] + un[i-1, j]) / dx**2 +
                (un[i, j+1] - 2*un[i, j] + un[i, j-1]) / dy**2
            )

# Plot final temperature distribution
fig = plt.figure(figsize=(15, 8))
plt.imshow(u, origin='lower', extent=(0, Lx, 0, Ly), cmap='hot')
plt.colorbar()
plt.title('Final Temperature Distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

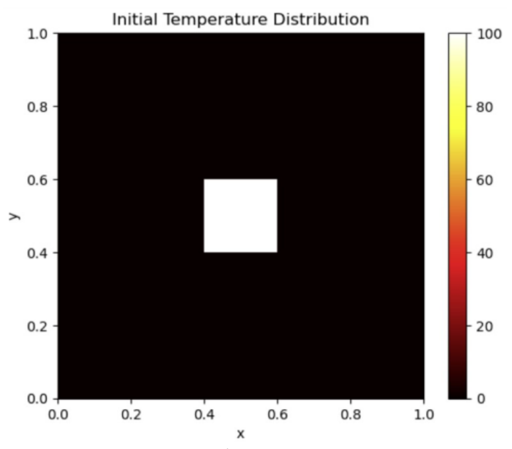


Figure 1

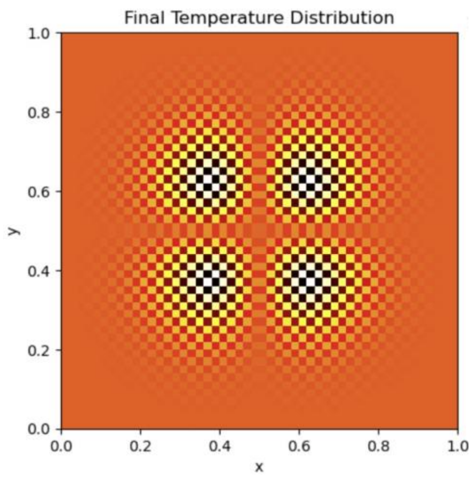


Figure 2

On changing the length of the domains to 5.0 from 1.0, we notice the temperature distribution as below (**Fig - 3),**

```
# Parameters
Lx = Ly = 5.0  # length of the domain in x and y directions
Nx = Ny = 50   # number of grid points in x and y directions
alpha = 0.1    # thermal diffusivity constant
T = 0.5        # total time
Nt = 100       # number of time steps
dt = T / Nt    # time step
dx = Lx / Nx   # grid size in x direction
dy = Ly / Ny   # grid size in y direction
```

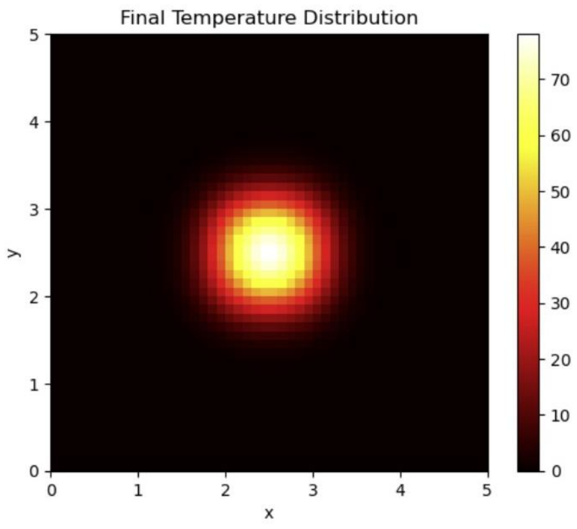


Figure 3

Alteration of thermal diffusivity and number of grid points from 50 to 100 yields (**Fig - 4**)

```
# Parameters
Lx = Ly = 5.0  # length of the domain in x and y directions
Nx = Ny = 100   # number of grid points in x and y directions
alpha = 0.6    # thermal diffusivity constant
T = 0.5        # total time
Nt = 100       # number of time steps
dt = T / Nt    # time step
dx = Lx / Nx   # grid size in x direction
dy = Ly / Ny   # grid size in y direction
```

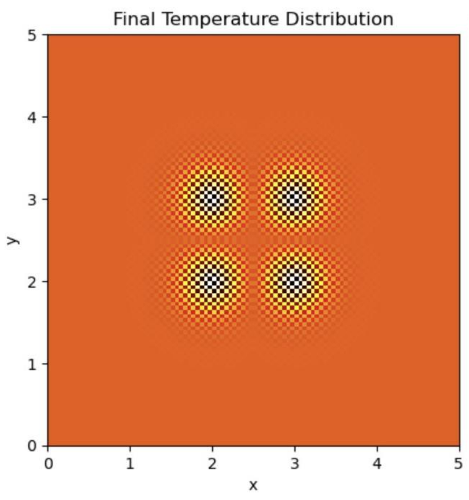
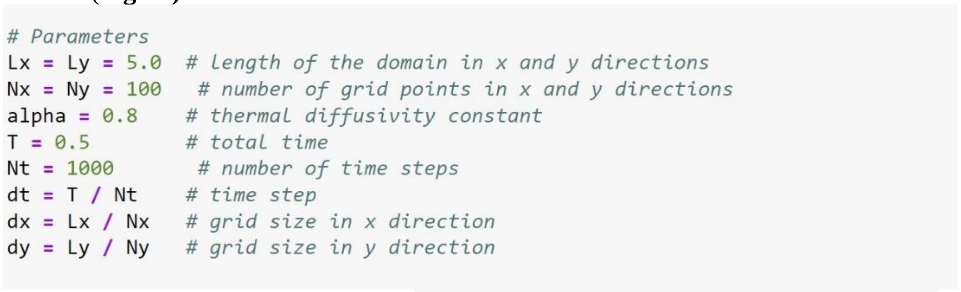


Figure 4

On increasing thermal diffusivity and the number of time steps to 1000, we notice the following temperature spread in the domain (**Fig - 5)**



```
# Parameters
Lx = Ly = 5.0  # length of the domain in x and y directions
Nx = Ny = 100   # number of grid points in x and y directions
alpha = 0.8    # thermal diffusivity constant
T = 0.5        # total time
Nt = 1000       # number of time steps
dt = T / Nt    # time step
dx = Lx / Nx   # grid size in x direction
dy = Ly / Ny   # grid size in y direction
```

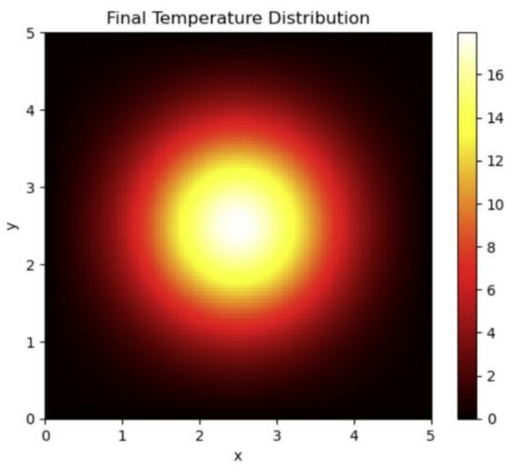
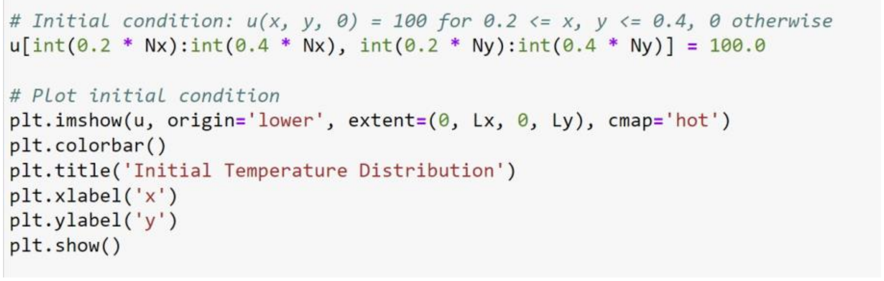


Figure 5

On varying the range along x - direction, the temperature spread location also displaces accordingly (**Fig - 6 & Fig - 7)**



```
# Initial condition: u(x, y, 0) = 100 for 0.2 <= x, y <= 0.4, 0 otherwise
u[int(0.2 * Nx):int(0.4 * Nx), int(0.2 * Ny):int(0.4 * Ny)] = 100.0

# Plot initial condition
plt.imshow(u, origin='lower', extent=(0, Lx, 0, Ly), cmap='hot')
plt.colorbar()
plt.title('Initial Temperature Distribution')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

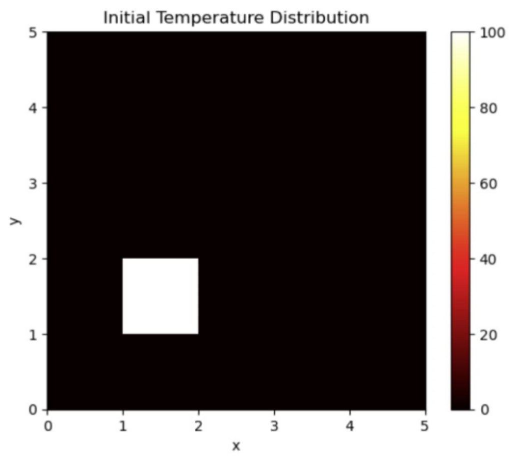


Figure 6

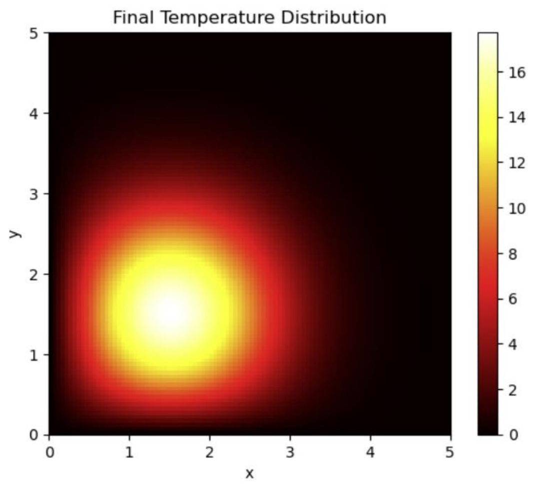


Figure 7

3. Conclusion

In this study, we have explored the numerical solution of the two - dimensional heat equation using Python, employing the finite difference method. The heat equation, a fundamental partial differential equation governing temperature distribution over a specified region, was discretized in both spatial dimensions and advanced in time steps to simulate the evolution of temperature. Through this paper, the effective application of Python script in studying the two - dimensional heat equation using the finite difference method is depicted. Our study intensely delineates the evolution of temperature distributions across a defined domain, highlighting the complex effects of thermal diffusivity on heat propagation. These findings enhance our understanding of heat transfer phenomena and their practical implications across diverse disciplines. This study underscores Python's capability as a robust tool for investigating complex physical processes, providing valuable insights into heat diffusion dynamics that contribute to advancements in both theoretical knowledge and practical applications.

References

[1] Wahida Zaman Loksar and Rama Sarkar, “A Numerical Solution of Heat Equation for Several Thermal Diffusitivity using Finite Difference Scheme with Stability Conditions”, Journal of Applied Mathematics and Physics, Volume 10, 2022, pp.449 - 465.

[2] https: //medium. com/[at]matiasortizdiez/beginners - introduction - to - natural - simulation - in - python - i - solving - the - heat - equation - bf0ae5d4c37f

[3] https: //levelup. gitconnected. com/solving - 2d - heat - equation - numerically - using - python - 3334004aa01a

[4] Xiao - Jun and Feng Gao, “A New Technology for Solving Diffusion and Heat Equation”, Thermal Science, Volume 21, January 2016, pp.01 - 09.

[5] G D Smith, “Numerical Solution of Partial Differential Equations: Finite Difference Method” Oxford 1992.

[6] Gerald W. Recktenwald, “Finite Difference Approximations to the Heat Equation”, Mechanical Engineering, Volume 10 (1), January 2004, pp 01 - 14