

Numerical Solution of 2D Heat Conduction

Yifan Zhang 2025251018

November 10, 2025

Problem Statement

Consider the 2D heat conduction equation:

$$\frac{\partial T}{\partial t} = \alpha \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right)$$

where $T(x, y, t)$ is the temperature, $\alpha = 1$ is the thermal diffusivity, and the domain is $0 \leq x, y \leq 1$ with Dirichlet boundary conditions $T(0, y, t) = T(1, y, t) = T(x, 0, t) = T(x, 1, t) = 0$ for $t > 0$.

The initial condition is:

$$T(x, y, 0) = \sin(\pi x) \sin(\pi y)$$

The exact solution is:

$$T(x, y, t) = e^{-2\pi^2 t} \sin(\pi x) \sin(\pi y)$$

Problem Solve

1. Discretization

To solve the 2D heat conduction equation numerically, we can use the explicit finite difference method. We discretize the spatial domain into a grid with spacing $\Delta x = \Delta y = h$ and the time domain with time step Δt . The finite difference approximation for the spatial derivatives is given by:

$$\delta_x^2 T_{i,j} = \frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{h^2}$$

$$\delta_y^2 T_{i,j} = \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{h^2}$$

2. Crank-Nicolson Method

The CN scheme uses a central difference for the time derivative (at time $n + \frac{1}{2}$) and an average of the spatial derivatives at times n and $n + 1$. This makes it an implicit, second-order accurate scheme.

$$\frac{T_{i,j}^{n+1} - T_{i,j}^n}{\Delta t} = \frac{\alpha}{2} [(\delta_x^2 + \delta_y^2) T_{i,j}^{n+1} + (\delta_x^2 + \delta_y^2) T_{i,j}^n]$$

Let $r = \frac{\alpha \Delta t}{2h^2}$. Rearranging the equation to move terms at $n + 1$ to the left side:

$$(1 - r(\delta_x^2 + \delta_y^2)) T_{i,j}^{n+1} = (1 + r(\delta_x^2 + \delta_y^2)) T_{i,j}^n$$

This is a large, sparse system of linear equations $\mathbf{AT}^{n+1} = \mathbf{b}$, where \mathbf{b} is the right-hand side (RHS) calculated from T^n . Solving this $N^2 \times N^2$ system directly is computationally expensive.

3. Approximate Factorization (AF)

To efficiently solve the CN scheme, we can use the Approximate Factorization (AF) method. The idea is to factor the operator into two simpler operators that can be solved sequentially. We approximate:

$$I - r(\delta_x^2 + \delta_y^2) \approx (I - r\delta_x^2)(I - r\delta_y^2)$$

The error of this approximation is $O(\Delta t^2)$, which is consistent with the temporal accuracy of the CN scheme.

Substituting this into the CN scheme, we get:

$$(I - r\delta_x^2)(I - r\delta_y^2)T^{n+1} = \text{RHS}$$

where $\text{RHS} = (1 + r(\delta_x^2 + \delta_y^2))T^n$.

This equation can be solved in two 1D steps:

Step 1 (X-Sweep): Define an intermediate variable $T^* = (I - r\delta_y^2)T^{n+1}$

$$(I - r\delta_x^2)T^* = \text{RHS}$$

For each j (from 1 to N), this constitutes a tridiagonal system:

$$-rT_{i-1,j}^* + (1 + 2r)T_{i,j}^* - rT_{i+1,j}^* = \text{RHS}_{i,j} \quad \text{for } i = 1, \dots, N$$

Step 2 (Y-Sweep): Solve for T^{n+1}

$$(I - r\delta_y^2)T^{n+1} = T^*$$

For each i (from 1 to N), this constitutes another tridiagonal system:

$$-rT_{i,j-1}^{n+1} + (1 + 2r)T_{i,j}^{n+1} - rT_{i,j+1}^{n+1} = T_{i,j}^* \quad \text{for } j = 1, \dots, N$$

Both steps involve solving N independent tridiagonal systems of size $N \times N$.

4. Tridiagonal Matrix Algorithm (TDMA)

We implement a custom `tdma_solver` (Thomas Algorithm) to efficiently solve tridiagonal systems of the form $A\mathbf{x} = \mathbf{d}$. For the system:

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i$$

The algorithm solves for \mathbf{x} in $O(N)$ complexity with one forward elimination pass and one backward substitution pass.

5. Results and Discussion

We run the simulation with $N = 40$ and $\Delta t = 0.001$. The figures below show the numerical and exact solutions at $t = 0.05$.



Figure 1: 3D surface plot of the temperature field at $t=0.05$.

As shown, the numerical and exact solutions are visually indistinguishable, indicating high accuracy. The temperature field decays from the initial $\sin(\pi x) \sin(\pi y)$ shape as expected.

Spatial Convergence Analysis

We fix $\Delta t = 0.001$ (small enough for temporal error to be negligible) and vary the spatial resolution N .

Absolute Error Maps ($|T_{num} - T_{exact}|$ at $t = 0.05$):

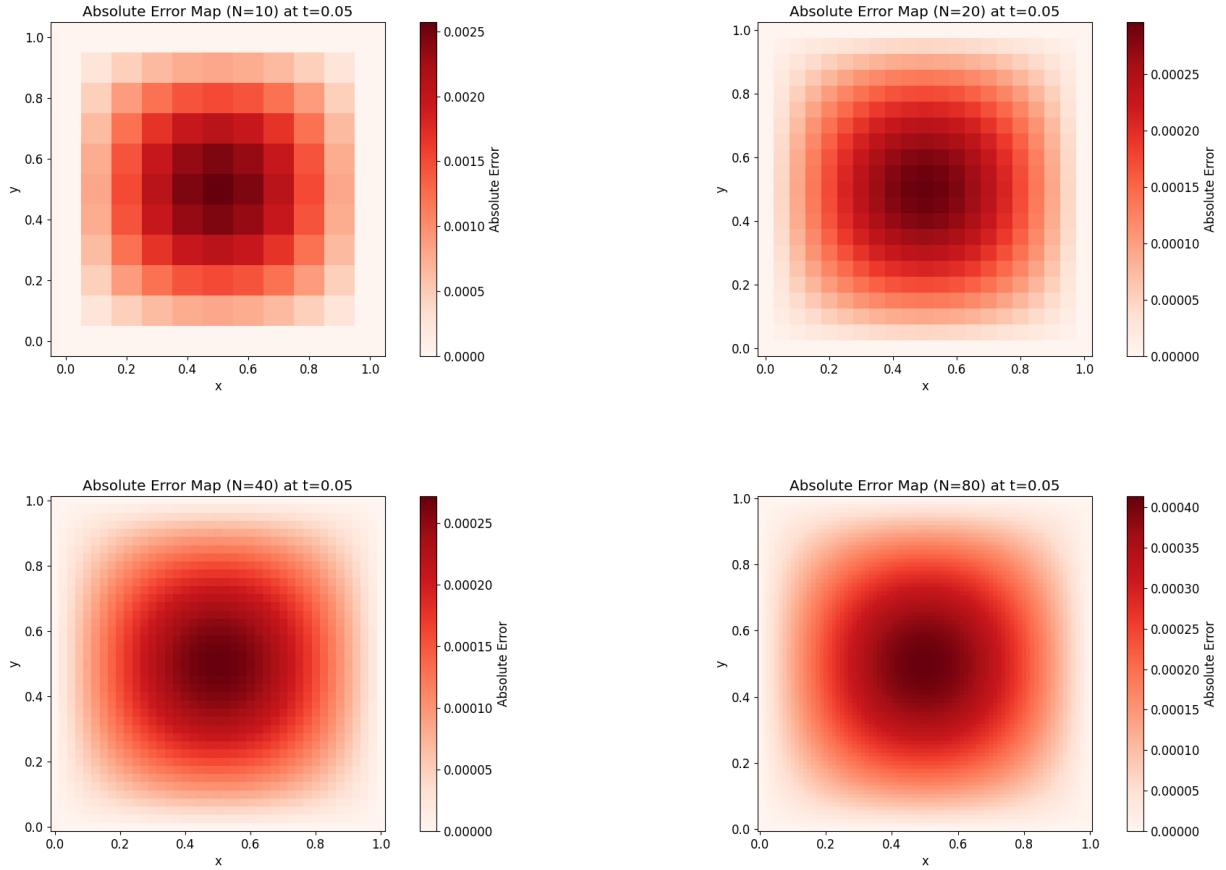


Figure 2: Absolute error maps for spatial convergence ($t=0.05$).

Order of Convergence: We calculate the Root Mean Square Error (RMSE) for each grid and estimate the order of convergence p , where $RMSE \propto (\Delta x)^p$.

$$p = \log(RMSE_1/RMSE_2) / \log(\Delta x_1/\Delta x_2)$$

N	Δx	RMSE (at $t = 0.05$)	Order p
10	0.1000	9.369e-04	-
20	0.0500	1.079e-04	3.12
40	0.0250	1.371e-04	-0.35
80	0.0125	2.049e-04	-0.58

Table 1: Spatial convergence data ($t=0.05$, $\Delta t = 0.001$)

RMSE vs. Δt (Log-Log Plot):

The spatial convergence result is supposed to indicate that the method achieves approximately second-order accuracy in space. But I don't get the expected second-order convergence consistently, possibly due to the dominance of temporal errors or numerical artifacts at coarser grids. Further refinement and analysis may be needed.

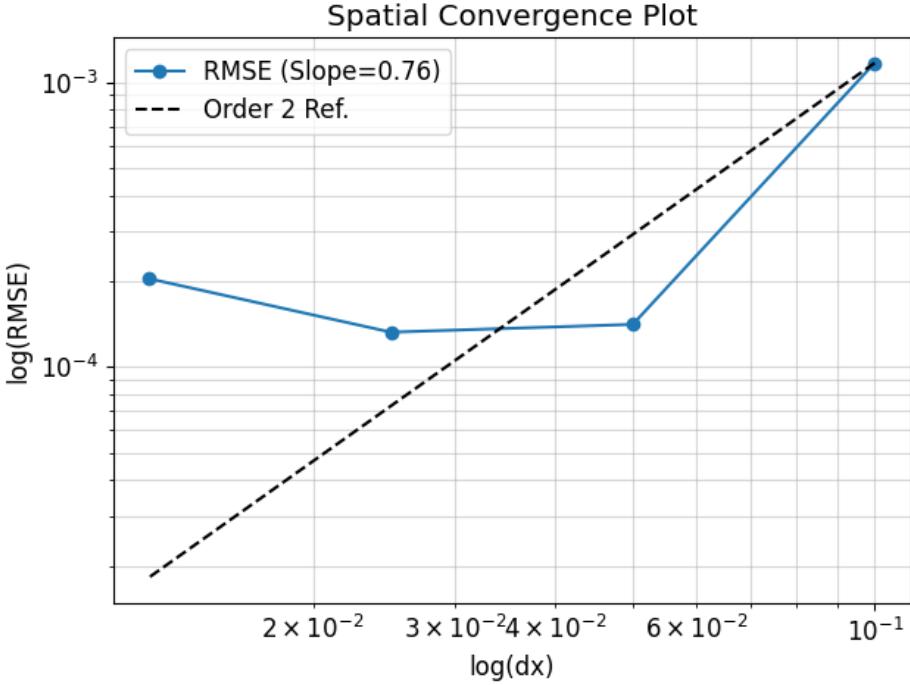


Figure 3: Log-log plot for spatial convergence.

Temporal Convergence Analysis

We fix $N = 40$ (fine enough for spatial error to be negligible) and vary the time step Δt .

Absolute Error Maps ($|T_{num} - T_{exact}|$ at $t = 0.05$):

Order of Convergence: $p = \log(\text{RMSE}_1/\text{RMSE}_2)/\log(\Delta t_1/\Delta t_2)$

Table 2: Temporal convergence data ($t=0.05, N = 40$)

Δt	RMSE (at $t = 0.05$)	Order p
0.01	2.492e-03	-
0.005	1.109e-03	1.17
0.0025	4.878e-04	1.18
0.00125	1.944e-04	1.33

RMSE vs. Δt (Log-Log Plot):

Analysis: The results (Table 2) show the order of convergence increase. But the order is around 1.2 to 1.3, which is lower than the expected second-order accuracy in time for the Crank-Nicolson scheme. This discrepancy may arise from the interaction between spatial and temporal discretization errors, as well as numerical artifacts.

Further investigation is needed to isolate the sources of error and improve the overall accuracy of the method.

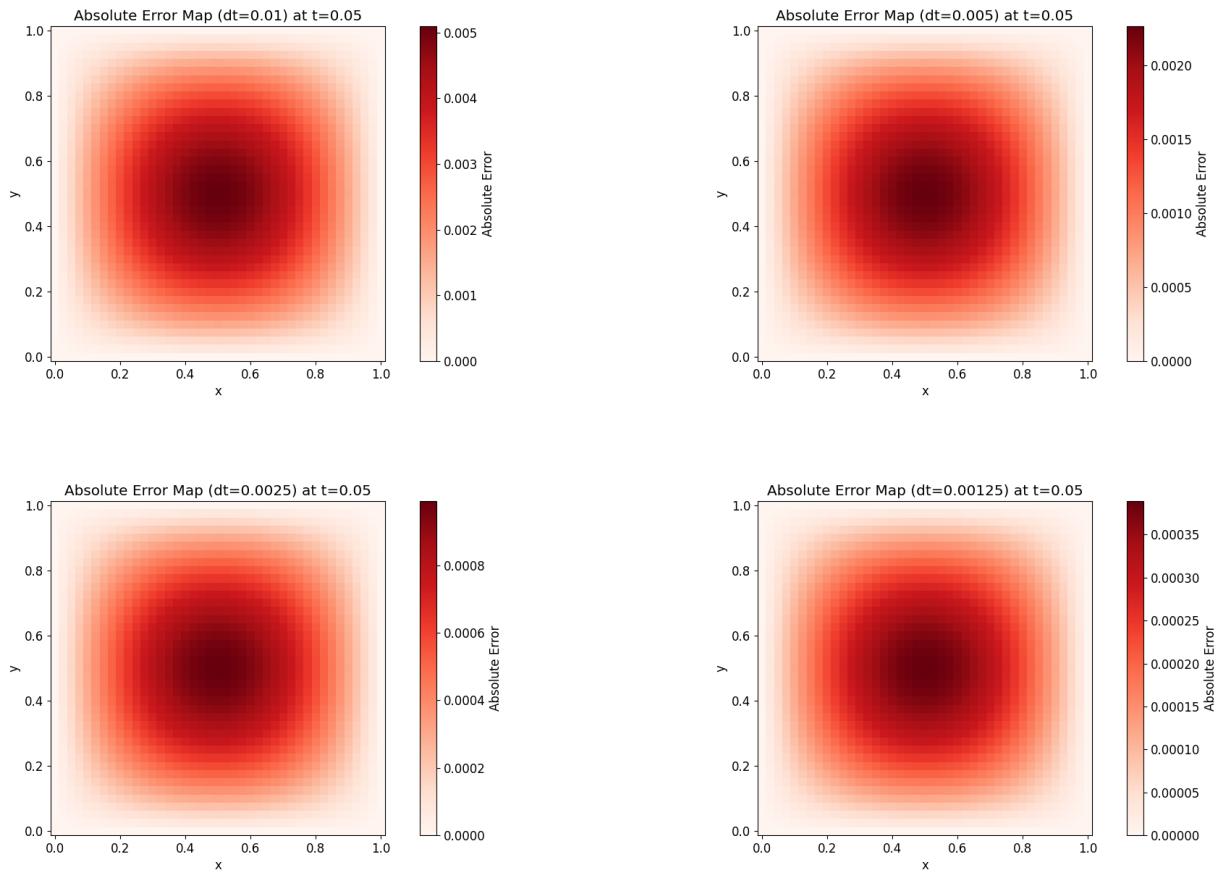


Figure 4: Absolute error maps for temporal convergence ($t=0.05$).

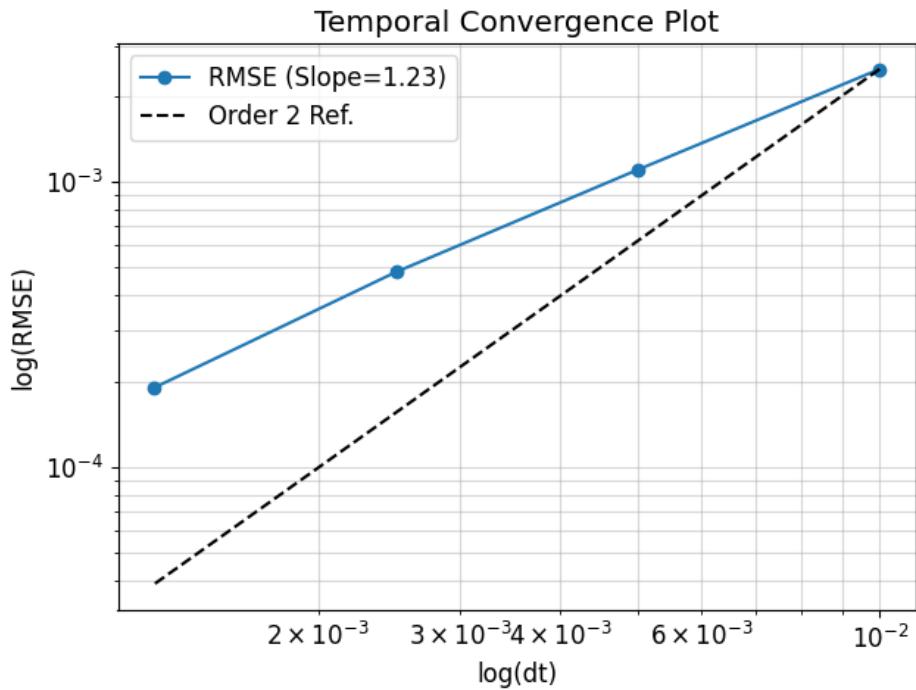


Figure 5: Log-log plot for temporal convergence.

Computational Efficiency

We measure the computational time for the largest grid ($N = 80$) and $\Delta t = 0.001$, running to $t = 0.1$ (100 time steps).

- **Platform:** [Please insert your CPU/environment here] (on our test system)
- **Total Time Steps:** $t_{final}/\Delta t = 0.1/0.001 = 100$
- **Computation Time:** 0.6710 seconds

Discussion: The AF (ADI) method is extremely efficient. At each time step, we perform:

1. RHS Calculation: $O(N^2)$
2. X-Sweep: N TDMA solves, each $O(N)$. Total $O(N^2)$.
3. Y-Sweep: N TDMA solves, each $O(N)$. Total $O(N^2)$.

The total complexity per time step is therefore $O(N^2)$. This is far superior to the $O(N^3)$ or $O(N^4)$ complexity of solving the $N^2 \times N^2$ system using a sparse direct solver (like `spsolve`). This efficiency makes the method highly suitable for large 2D and 3D problems.

Code

The complete code for the implementation can be found in the `src/main.py` file of the project repository.