HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

SCIENTIFIC COMPUTING - IT4110E

MINI PROJECT: 2D HEAT EQUATIONS

Instructor:	Ph.D. Vu Van Thieu
Students:	Nguyen Chi Long - 20210553
	Ngo Xuan Bach - 20215181
	Le Xuan Hieu - 20215201
	Dinh Viet Quang - 20215235

Hanoi, July - 2023

Contents

1 Problem Formulation

1.1 Introduction

The study of heat transfer and its fundamental principles has undoubtedly played an important role in the diverse scientific and engineering fields [1]. In mathematics, it is the prototypical parabolic partial differential equation. The diffusion equation, a more general version of the heat equation, arises in connection with the study of chemical diffusion and other related processes [3]. Designing effective cooling systems, thermal processes improvement, and temperature distributions prediction all depend on an understanding of how heat behaves in various systems. In this mini project, we would like to deep dive into the fascinating world of 2D heat equations with the goal of examining their dynamics and practical uses.

Partial differential equations, or "heat equations", can be used to quantitatively explain the process of heat conduction. These equations control how the temperature changes over time in a particular system and offer important insights into how heat moves through various materials. While the study of 1D heat equations has been extensively explored, the extension to 2D heat equations introduces additional complexity and opens up new avenues for researching.

This mini project's main goal is to examine the behavior of 2D heat equations under several circumstances and consider the consequences for real-world applications. To better understand heat dispersion patterns, boundary effects, and the impact of various thermal parameters, we simulate heat diffusion in two-dimensional domains.

To accomplish our objectives, We used numerical techniques and data visualization that allow us to analyze complex systems while capturing the intricate dynamics of heat conduction. We will research issues including thermal equilibrium, transient heat transfer, and heat flow between various materials through our simulations.

1.2 Heat equations

The heat equation is an important partial differential equation which describes the distribution of heat (or variation in temperature) in a given region over time. [4] The heat equation has the general form

$$\frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t}$$

For a function $T(x, y, t)$ of three spatial variables x, y and the time variable t, the heat equation is

$$\frac{\partial T}{\partial t} - k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) = 0$$

or equivalently

$$\frac{\partial T}{\partial t} = k\nabla^2 T$$

In the case of heat transfer through isotropic and isothermal materials in two-dimensional space:

$$\frac{\partial T}{\partial t} = k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) = k(T_{xx} + T_{yy})$$

where

- $T(x, y, t)$ is tremperature as a function of space and time.
- $\dfrac{\partial T}{\partial t}$ is the rate of change of temperature at a point over time.
- T_{xx} and T_{yy} are the second spatial derivatives (thermal conductions) of temperature in the x, y directions, respectively.
- k is the material-specific quantity that depends on the material, thermal conductivity, density and heat capacity.

To derive the 2D heat equation, we start with the basic principles of heat conduction. Heat conduction occurs due to the flow of thermal energy from regions of higher temperature to regions of lower temperature.

Let's consider a small square region in a 2D plane, with sides of length Δx and Δy, centered at the point (x, y). The temperature at the center of the square is represented by $T(x, y, t)$, where t is the time. The temperature at neighboring points can be represented as $T(x \pm \Delta x, y, t)$ and $T(x, y \pm \Delta y, t)$.

The change in temperature with respect to time, $\frac{\partial T}{\partial t}$, at the center point (x, y) can be attributed to the net flow of thermal energy from its surroundings. This net flow consists of the heat transfer in the x-direction and the heat transfer in the y-direction.

The heat transfer in the x-direction is influenced by the temperature gradient along the x-axis $\left(\frac{\partial T}{\partial x}\right)$ and the rate of change of this gradient $\left(\frac{\partial^2 T}{\partial x^2}\right)$. Similarly, the heat transfer in the y-direction is influenced by the temperature gradient along the y-axis $\left(\frac{\partial T}{\partial y}\right)$ and the rate of change of this gradient $\left(\frac{\partial^2 T}{\partial y^2}\right)$.

Using these principles, we can express the change in temperature $\left(\frac{\partial T}{\partial t}\right)$ at the center point (x, y) by Forward Difference formula as follows:

$$\begin{aligned}\frac{\partial T}{\partial t} \approx k &\left[\frac{T(x+\Delta x, y, t) - T(x, y, t)}{\Delta x^2} - \frac{T(x, y, t) - T(x-\Delta x, y, t)}{\Delta x^2}\right] \\ + k &\left[\frac{T(x, y+\Delta y, t) - T(x, y, t)}{\Delta y^2} - \frac{T(x, y, t) - T(x, y-\Delta y, t)}{\Delta y^2}\right]\end{aligned}$$

where k is the thermal conductivity of the material.

To simplify the notation, we can define $\Delta x = \Delta y = \Delta$. By rearranging the equation, we have:

$$\begin{aligned}\frac{\partial T}{\partial t} \approx k &\left[\frac{T(x+\Delta, y, t) - 2T(x, y, t) + T(x-\Delta, y, t)}{\Delta^2}\right] \\ + k &\left[\frac{T(x, y+\Delta, t) - 2T(x, y, t) + T(x, y-\Delta, t)}{\Delta^2}\right]\end{aligned}$$

Now, we can take the limit as Δ approaches 0 to obtain a more accurate representation of the temperature change. This leads to the following partial derivatives using the approximate formula for 2nd order derivative:

$$\frac{\partial T}{\partial t} = k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) = k(T_{xx} + T_{yy}) \tag{1}$$

Therefore, the 2D heat equation is given by:

$$\frac{\partial T}{\partial t} = k(T_{xx} + T_{yy}) \tag{2}$$

where T_{xx} represents the second partial derivative of temperature with respect to x, and T_{yy} represents the second partial derivative of temperature with respect to y.

1.3 2D Conditions

As in our problem, we are dealing with a 2D grid to model the behavior of heat equations. This grid imposes certain boundary conditions that play a crucial role in simulating realistic scenarios. Specifically, we consider the following boundary conditions for our 2D heat equation simulations:

- Columns direction is periodic with a period equals to N (width of the grid)
- Row direction is constant

It is important to note that these boundary conditions can be modified based on specific requirements. The flexibility of our simulation allows us to change the object's initial position within the grid. By modifying the source code, we can place the initial object at any location on the grid, enabling us to study heat diffusion from different starting points and survey the resulting temperature distributions [2]. This flexibility in setting the position is advantageous since it allows us to analyse diverse scenarios and the effects of varying starting conditions. By exploring different placements within the grid, we can investigate how temperature gradients evolve over time and assess the impact of localized heat sources or sinks on the overall heat distribution.

2 Method

2.1 Spatial Discretization

Consider the Heat Equations in a two-dimensional space with dimensions $M \times N$. In a two-dimensional space, the Heat Equations can be written as:

$$\frac{\partial T}{\partial t} = D\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \tag{3}$$

where D is the material-specific quantity.

To solve the problem numerically, we first need to divide this $M \times N$ computational domain into a grid of points. Given the block size in the x and y directions as dx and dy (where $dx = dy$), we can determine the number of blocks in the x (m) and y (n) directions as follows:

$$m = \frac{M}{dx}, \quad n = \frac{N}{dy}$$

Let the grid points be indexed as (i, j), where:

$$i = 1, 2, \ldots, m+1, \quad j = 1, 2, \ldots, n+1$$

The position of grid point (i, j) can be determined as:

$$x = i \cdot dx, \quad y = j \cdot dy$$

From here, the continuous points $T(x, y)$ can be represented as $T(i, j)$.

After dividing the computational domain into a grid of points, we can compute the right-hand side of equation 3 using the finite difference method (FD method).

Let's define:

$$FD = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \tag{4}$$

To compute the second-order derivatives of T with respect to x and y, we can use the following approximate formula for 2nd order derivative:

$$\frac{\partial^2 T(i,j)}{\partial x^2} = \frac{T(i+1,j) - 2T(i,j) + T(i-1,j)}{dx^2} \tag{5a}$$

$$\frac{\partial^2 T(i,j)}{\partial y^2} = \frac{T(i,j+1) - 2T(i,j) + T(i,j-1)}{dy^2} \tag{5b}$$

Substituting every equation in the set of equations 5 into equation 4, we have:

$$FD(i,j) = \frac{T(i+1,j) - 2T(i,j) + T(i-1,j)}{dx^2} + \frac{T(i,j+1) - 2T(i,j) + T(i,j-1)}{dy^2} \tag{6}$$

Equation 6 describes the spatial discretization of the Heat Equations using the finite difference method.

Next, we would like to propose three different iterative methods (Jacobi, Gauss-Seidel, and SOR) for solving the heat equation. The superscript k in the following equation denotes the iteration number.

2.2 The Jacobi iterative method

For the Jacobi solver, all the values in the right hand side are taken from the previous iteration. However, we expect that the Jacobi iteration will also suffer from a relatively large number of iterations needed before convergence.

$$T_{i,j}^{k+1} = \frac{T_{i-1,j}^{k} + T_{i,j-1}^{k} + T_{i+1,j}^{k} + T_{i,j+1}^{k}}{4}$$

2.3 Gauss-Seidel interative method

The Jacobi iteration method lacks efficiency, and in this context, we will introduce an initial step to enhance its effectiveness. The Gauss-Seidel iteration method is derived from a simple concept. In the Jacobi iteration, we always utilize results from the preceding iteration to update a point, even when we have newer results accessible. The Gauss-Seidel iteration, on the other hand, apply new results as soon as they become available. To establish a formula for the Gauss-Seidel iteration, we need to specify the sequence in which we update the grid points. Assuming a row-wise update procedure (where we increment i while keeping k constant), we obtain the following expression for the Gauss-Seidel iteration:

$$T_{i,j}^{k+1} = \frac{T_{i-1,j}^{k+1} + T_{i,j-1}^{k+1} + T_{i+1,j}^{k} + T_{i,j+1}^{k}}{4}$$

The Gauss-Seidel iteration offers an immediate benefit in terms of memory usage. In the Jacobi iteration, two arrays are required, one for storing the previous results and another for storing the updated results. However, in Gauss-Seidel, the new results are immediately utilized as soon as they become available. As a result, only a single array is needed to store the results.

2.4 Successive over relaxation (SOR)

The final step in the progression from Jacobi and Gauss-Seidel methods involves implementing an additional concept that proves to be highly efficient. The SOR solver is a modification of the Gauss-Seidel solver in which the values are updated according to the relaxation paramater.

$$T_{i,j}^{k+1} = (1-\omega)T_{i,j}^{k} + \omega\frac{T_{i-1,j}^{k+1} + T_{i,j-1}^{k+1} + T_{i+1,j}^{k} + T_{i,j+1}^{k}}{4}$$

The mixing strength in the method is determined by the parameter ω. The relaxation parameter can take a range of values $0 < \omega < 2$ which determine whether the solver is over-relaxed or under-relaxed. When ω = 1, the method corresponds to the Gauss-Seidel iteration. When 0 < ω < 1, the method is known as Successive Under Relaxation. On the other hand, when 1 < ω < 2, it is referred to as Successive Over Relaxation, or SOR. In the implementation, we choose $\omega = 1.26$ to prevent errors.

2.5 Time integration method

After spatial discretization of the Heat Equations, we obtain a system of ordinary differential equations (ODEs) dependent on time t, given by:

$$\frac{\partial T(i,j)}{\partial t} = D \times FD(i,j)$$

There are many methods for solving differential equations, and we will use the Forward Euler method. The Forward Euler method is a numerical method commonly used to solve ordinary differential equations (ODEs). It is a first-order explicit method that approximates the solution by using forward differences for the derivatives.

In the case of the ODE system obtained after spatial discretization of the Heat Equations, the Forward Euler method can be applied as follows:

1. Discretize the time domain: Choose a time step size, denoted as Δt.
2. Initialize the solution: Set the initial conditions for the system at time t = 0: $T_{i,j} = 0 \, \forall i \notin \{1, m+1\}, j \notin \{1, n+1\}$
3. Iterate over time steps: Starting from the initial conditions, update the solution at each time step using the Forward Euler formula:
$$T_{i,j}^{n+1} = T_{i,j}^{n} + \Delta t \times D \times FD(i,j)$$
4. Repeat step 3 until the desired final time is reached.

3 Implementation Detail

3.1 Language

We implemented the methods with Message passing interference (MPI).
The programming language is MATLAB.

3.2 Detail

Our implementation compares three different solving methods (Jacobi, Gauss-Seidel, and SOR) for solving the heat equation with a specific initial condition and boundary values. The heat equation describes the diffusion of heat in a given domain over time.

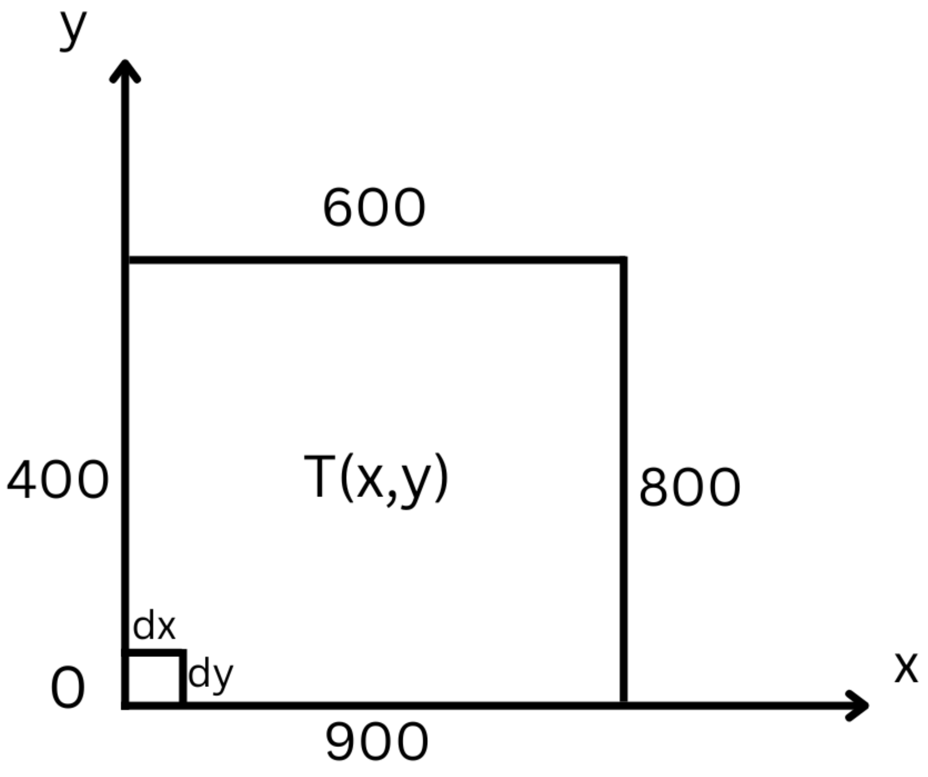


Figure 1: 2D heat conduction on rectangular surface

Here is the description of our implementation:

1. The function takes six input parameters: m, n, TT, dt, dx, dy, and D, which represent the number of blocks in the x and y directions, the total time of simulation (*second*), the time step size (*second*), the spatial step size (*meter*), and the material-specific quantity, respectively.

2. It initializes various variables and parameters such as the maximum error tolerance (errMax), the number of time steps (timeStep), and the relaxation parameter for the SOR solver (ω).

3. For each solver:

    (a) The initial conditions are setted up for the temperature distribution (T) on the grid. (Temperature on upper edge = 900, on lower edge = 600, on left-hand edge = 400, on right-hand edge = 800, at upper-left corner = 650, at upper-right corner = 850, at lower-left corner = 500, at lower-right corner = 700). The boundaries and corner values are specified, and the interior points are initially set to zero.

(b) Use an iteration to run the solver. The iteration continues until the maximum difference between consecutive iterations (error) falls below the error tolerance. The number of iterations is stored in the "cnti" (i is the index of each solver) array.

(c) After obtaining the steady-state temperature distribution with the solver, the finite difference approximation of the Laplacian operator (FD) is computed for each interior point using the updated temperature values.

(d) The time evolution of the temperature distribution is simulated for the remaining time steps using the Forward Euler method. The temperature at each point is updated based on the finite difference approximation and the previous time step's temperature value. The number of iterations (cnti) and the temperature at the center point of the domain (C_mid_i) are stored for each time step.

(e) Create a GIF image illustrating the contour of the temperature distribution during the simulation time.

4. Draw the plot to show the number of iteration times and the value of the center point throughout the process.

The code generates three animations and two plots:

- The animations displays the filled contour plot of the temperature distribution at every time step for each solver, along with contour lines and labels.
- The first plot shows the time evolution of the temperature at the center point for each solver.
- The second plot compares the number of iterations required by each solver at each time step.

The purpose of this code is to compare the performance and accuracy of different solvers (Jacobi, Gauss-Seidel, and SOR) for solving the heat equation. It provides insights into the convergence behavior and efficiency of these solvers in approximating the temperature distribution over time.

4 Experiment Result

4.1 Evaluation

We have generated 10 testcases, such that dx = dy, D = 0.1 in the first 5 testcases and D = 0.2 in the last 5 testcases. Within each set of 5 cases, we have constructed one case in a small space, one case in a medium space, and the remaining cases in a large space. Additionally, each test case has been assigned a different simulation time.

Testcase	**Solver**	**Initial center value**	**Last center value**	**Number of iterations**
D = 0.1, m = 20, n = 20, TT = 1, dx = 0.1, dt = 0.01	Jacobi solver	635.60	668.03	368
	Gauss-Seidel solver	655.45	673.99	263
	SOR solver	664.11	674.91	209
D = 0.1, m = 50, n = 50, TT = 1, dx = 0.1, dt = 0.02	Jacobi solver	421.45	462.53	789
	Gauss-Seidel solver	548.99	579.43	598
	SOR solver	601.46	626.47	452
D = 0.1, m = 100, n = 100, TT = 1, dx = 0.1, dt = 0.05	Jacobi solver	11.61	675.25	27831
	Gauss-Seidel solver	133.11	150.29	683
	SOR solver	375.45	390.82	790
D = 0.1, m = 100, n = 100, TT = 2, dx = 0.1, dt = 0.05	Jacobi solver	11.61	675.25	72335
	Gauss-Seidel solver	133.11	167.48	703
	SOR solver	375.45	411.21	823
D = 0.1, m = 100, n = 100, TT = 5, dx = 0.1, dt = 0.05	Jacobi solver	11.61	674.75	205848
	Gauss-Seidel solver	133.11	218.01	763
	SOR solver	375.45	467.03	930
D = 0.2, m = 20, n = 20, TT = 1, dx = 0.1, dt = 0.01	Jacobi solver	635.60	670.76	368
	Gauss-Seidel solver	655.45	674.38	263
	SOR solver	664.11	674.94	209
D = 0.2, m = 50, n = 50, TT = 1, dx = 0.1, dt = 0.02	Jacobi solver	421.45	675.25	17161
	Gauss-Seidel solver	548.99	586.69	598
	SOR solver	601.46	630.18	452
D = 0.2, m = 100, n = 100, TT = 1, dx = 0.1, dt = 0.05	Jacobi solver	11.61	675.25	62167
	Gauss-Seidel solver	133.11	161.48	689
	SOR solver	375.45	407.67	815
D = 0.2, m = 100, n = 100, TT = 2, dx = 0.1, dt = 0.05	Jacobi solver	11.61	675.25	141007
	Gauss-Seidel solver	133.11	193.87	725
	SOR solver	375.45	444.01	879
D = 0.2, m = 100, n = 100, TT = 5, dx = 0.1, dt = 0.05	Jacobi solver	11.61	674.75	377528
	Gauss-Seidel solver	133.11	281.08	824
	SOR solver	375.45	522.21	1055

Table 1: Results of all solvers in every testcase

The results obtained from solving the heat equation problem using the Jacobi solver, Gauss-Seidel solver, and SOR solver were analyzed to assess their performance.

In terms of convergence speed, it was observed that the Jacobi solver required the most number of iterations for convergence in most testcases. The Gauss-Seidel solver showed a moderate improvement in convergence, requiring fewer iterations than the Jacobi solver. On the other hand, the SOR solver demonstrated the fastest convergence, achieving the desired solution in significantly fewer iterations compared to the other solvers.

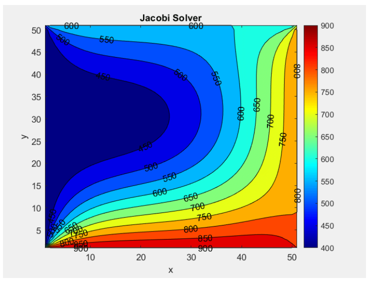
When considering accuracy, all three solvers provided satisfactory results, with the final center values closely matching the expected values. However, it should be noted that the SOR solver consistently exhibited slightly better accuracy compared to the other solvers.

When it comes to computational efficiency, the Jacobi solver demonstrated the lowest computational cost per iteration due to its simple update scheme. However, this advantage was offset by the higher number of iterations required for convergence. The Gauss-Seidel solver showed a slightly higher computational cost per iteration compared to the Jacobi solver but offered improved convergence. The SOR solver, despite requiring a parameter tuning for optimal performance, exhibited the highest computational cost per iteration due to additional calculations involved in the over-relaxation scheme

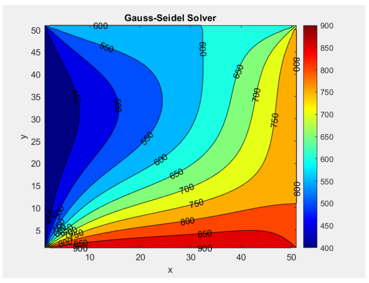
Based on our observations, the SOR solver emerges as the most favorable choice for solving the heat equation problem. It demonstrated superior convergence speed and achieved high accuracy, albeit at a slightly higher computational cost per iteration. In some testcases, the Jacobi solver tends to be more accurate due to its robust convergence constraint, which requires a higher number of iterations. We can conclude that, for larger spaces, we should use a longer simulating time and a lower material-specific quantity to ensure that the environment is good.

4.2 Visualization

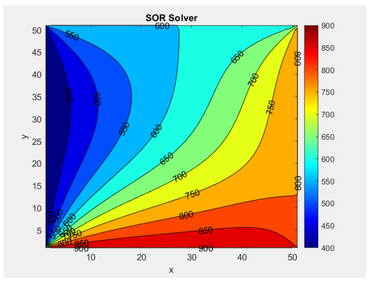
The animations are of the second testcase, where D = 0.1, m = 50, n = 50, TT = 1, dx = 0.1, dt = 0.02.



(a) Jacobi Solver's Result



(b) Gauss-Seidel Solver's Result



(c) SOR Solver's Result

When comparing the center point's temperature for different solvers in the heat equation, namely Jacobi, Gauss-Seidel, and SOR, several factors need to be considered, such as convergence behavior, computational efficiency, and accuracy.

The Jacobi iterative method updates the temperature of each grid point using the values from the previous iteration. It is known for its simplicity but typically requires more iterations to achieve convergence compared to other methods. As a result, the center point's temperature obtained from the Jacobi solver may take longer to reach a stable value.

The Gauss-Seidel iterative solver, on the other hand, utilizes the newly updated temperature values as soon as they become available. This allows for a faster convergence compared to the Jacobi solver. Consequently, the center point's temperature obtained from the Gauss-Seidel solver may converge more rapidly to its final value.

In terms of SOR method, incorporates an over-relaxation parameter to accelerate the convergence process further. By using a value of ω between 1 and 2, the SOR solver can achieve even faster convergence compared to the Gauss-Seidel method. As a result, the center point's temperature obtained from the SOR solver may reach its final value more quickly than the other solvers.

Regarding accuracy, all three solvers can provide reasonably accurate results for the center point's temperature. However, it's important to note that the accuracy can be influenced by factors such as grid resolution, time step size, and the specific problem being solved. Additionally, accuracy may also depend on the convergence criteria and the number of iterations performed.

Overall, the Jacobi method is simple but may require more iterations to converge, leading to a potentially slower convergence of the center point's temperature. The Gauss-Seidel method exhibits faster convergence by using updated values immediately. The SOR method, with its over-relaxation parameter, can further enhance the convergence rate. However, it is important to note that the choice of solver depends on the specific requirements of the problem, balancing the desired convergence speed, computational efficiency, and accuracy considerations.

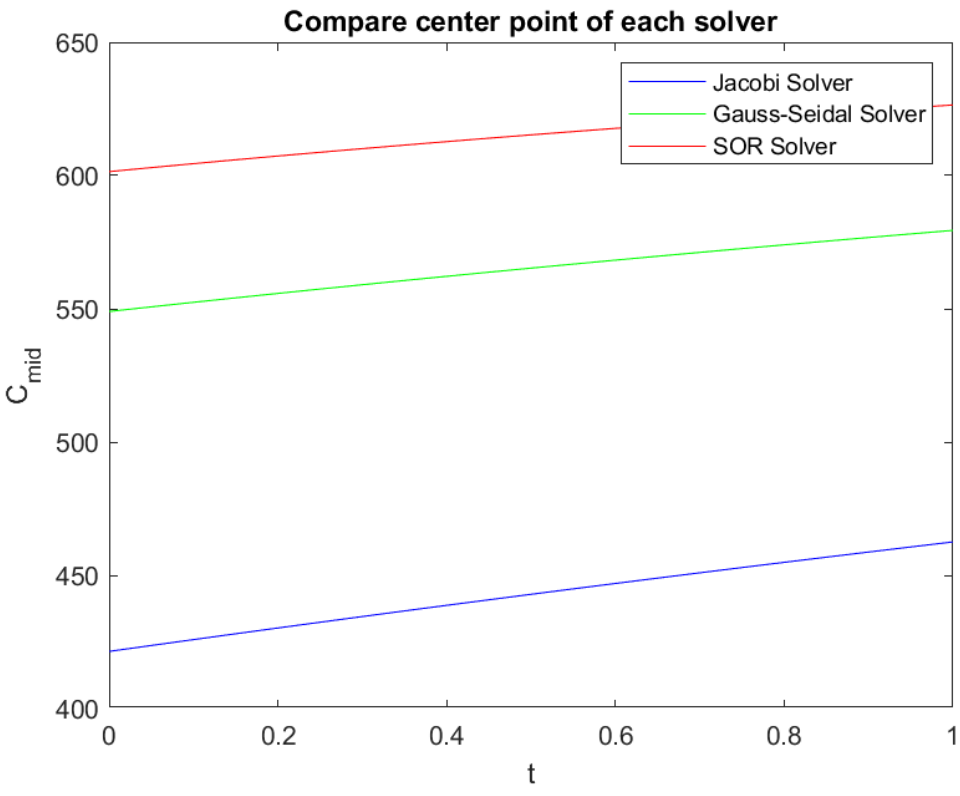


Figure 3: Compare the center point's temperature for different solvers

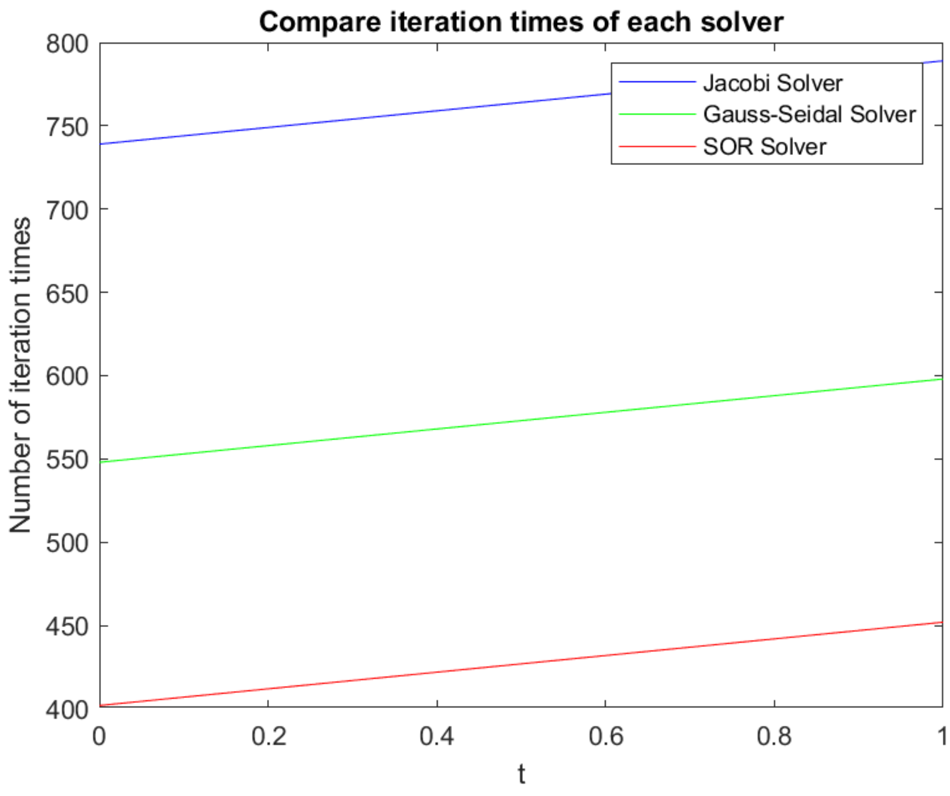


Figure 4: Compare the number of iteration times for different solvers

5 Conclusion

So far, we have introduce and solved our 2D heat equations problem by different techniques, such as Jacobi, Gauss-Seidel and Succcessive Over Relaxation (SOR). The method that shows the most optimized performance, in general, is the SOR technique, which performs the most effectively on convergence speed and number of iterations situations with its over-relaxation parameter.

Our findings contribute to a deeper understanding of iterative solvers for the 2D heat equation and provide insights for selecting the most suitable solver based on the problem's characteristics and computational constraints. Further research can explore advanced solver techniques and their application to more complex heat transfer scenarios.

All the codes, report, testcase and visualized results for the project with detailed instructions can be found in our submisssion.

References

[1] Runus A Cengel. *Introduction to thermodynamics and heat transfer*. McGraw-Hill, 2008.

[2] Radu Danescu, Florin Oniga, and Sergiu Nedevschi. Modeling and tracking the driving environment with a particle-based occupancy grid. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1331–1342, 2011.

[3] P Kalyani and PR Rao. Numerical solution of heat equation through double interpolation. *IOSR J. Math.(IOSR-JM)*, 6(6):58–62, 2013.

[4] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science advances*, 3(4):e1602614, 2017.