Study of Point Successive Over-Relaxation and Alternating Direction Implicit method for solving Steady State Heat Diffusion in 2-D

Mid-Term Project

Course: MEEN 489/689 Special topic: Computational Fluid Dynamics

Prepared by,

Yatharth Kishor Vaishnani

327005677

Date: 11/07/2018

TABLE OF CONTENTS

1 Introduction

A major objective in a thermal conduction analysis is to determine the temperature distribution in a medium resulting from the conditions imposed on its boundaries, which represents how temperature varies with position in the medium. Once this distribution is known, the conduction heat flux at any point in the medium or on its surface may be computed from the Fourier's law.

The steady state heat diffusion in 2-D domain is the simplest example for the elliptical type of governing partial differential equation. For calculating the temperature distribution over the domain, analytical solution is available using the separation of variable method. The numerical schemes are also used for calculating the same. Here, the point Successive Over Relaxation method and Alternative Direction Implicit method are used to compute the given heat equation in 2-D domain. The table 1 shows the different cases considered for the study of these two computational methods using MATLAB.

The heat diffusion problem considered here is governed by the Laplace equation for temperature distribution (Eq. (1)) in a homogeneous constant property (thermal conductivity) solid in two dimension space. The plate is considered to have rectangular shape and assumed to have constant thermal conductivity with respect to space and temperature.

Table 1 Different cases considered for study of computational methods for steady state 2-D heat diffusion

Case No.	Description	Grid points (Nx,Ny)	Relaxation Factor (point SOR, ADI)
1	Temperature distribution	(40,30)	(1.8, 1.3)
2	Sensitivity analysis	Varying from (20,15) to (80,60)	(1.8, 1.3)
3	Spatial accuracy analysis	Varying from (20,15) to (80,60)	(1.8, 1.3)
4	Effect of relaxation factor	(40,30) and (80,60)	Varying from (1,1.99) to (1,1.33)
5	Time complexity	Varying from (40,30) to (80,60)	(1.8,1.3)

The study presented in this report is aimed to analyze the computational methods – point Successive Over Relaxation method and Alternating Direction Implicit method for solving the elliptical type of partial differential equation of steady state heat diffusion in 2-D domain without heat generation and with Diritchlet type of boundary conditions including investigation of the spatial accuracy, effect of relaxation factor and discussing the computational complexity of both algorithms.

2 Method of Solution

The steady state heat diffusion in 2-D domain can be explained by the Fig. (1). The boundaries of the domain are kept at T1, T2, T3 and T4 temperature as shown in the figure, where, T1 = 10 degree C, T2 = 0 degree C, T3 = 40 degree C and T4 = 0 degree C.

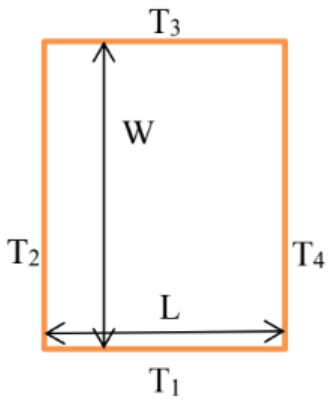

Figure 1 Schematic diagram of the computational domain

For the case of steady state heat diffusion in 2-D Cartesian system without any heat generation, the Laplace equation applies, assuming the constant thermal conductivity.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1}$$

Equation (1) is elliptical partial differential equation which has no real characteristic curves. A disturbance is propagated instantly in all directions within the region. The domain of solution for an elliptic partial differential equation is a closed region. On the closed boundary of this region, either the value of the dependent variable is normal gradient, or a linear combination of the two is prescribed. For developing a well posed problem providing the boundary conditions uniquely is required, which in the given problem statement are Diritchlet type of boundary conditions.

Table 2 Non-dimensional form of different quantities

Quantity	Non-dimensionalized form	Nomenclature
T	$T^* = \frac{T}{T_1}, T_{31} = \frac{T_3}{T_1}$	T= Temperature (°C) T^*= Non-dimensional temperature T_1= Temperature at bottom edge (= 40 °C) T_{31}= Ratio of Temperature at top to bottom edge (= 1/4)
x	$x^* = {}^{x}/_{L}$	x= Distance of the node from the left wall (m) x^*= Non-dimensional distance L= Size of the domain in x-direction (= 0.4 m)
y	$y^* = {}^{y}/_{W}$	y= Distance of the node from the bottom wall (m) y^*= Non-dimensional distance W= Size of the domain in y-direction (= 0.3 m)

The non-dimensional form of the Eq. (1) can be written by using the non-dimensional form of the quantities shown in table 2.

Using the non-dimensional terms, the Eq. (1) can be written by

$$\frac{\partial^2 T^*}{\partial {x^*}^2} + \alpha^2 \frac{\partial^2 T^*}{\partial {y^*}^2} = 0 \tag{2}$$

where, $\alpha = {}^{L}/_{W}$

For the non-dimensionalized form in the Eq. (2), the boundary conditions for the given problem statement can be written as shown in table 3.

Table 3 Boundary conditions in non-dimensional form

Boundary Conditions	$x^* = 0, T^* = 0 \quad ; \quad x^* = 1, T^* = 0$ $y^* = 0, T^* = 1 \quad ; \; y^* = 0, T^* = 1/4$

The solution to the Eq. (2) can be obtained by analytical and numerical method. The analytical solution of the Eq. (2) is calculated by the separation of variable method. With the available problem statement, subject to the imposed boundary conditions, the analytical solution to the Eq. (2) is given by the Eq. (2).

$$T(x, y) = T_A(x, y) + T_B(x, y) \tag{3}$$

where,

$$T_A(x, y) = 2T_1 \sum_{m=1}^{\infty} \frac{1 - cos\,(m\pi)}{m\pi} \frac{sinh\left(\frac{m\pi(W-y)}{L}\right)}{sinh\left(\frac{m\pi W}{L}\right)} sin\left(\frac{m\pi x}{L}\right) \tag{4}$$

and

$$T_B(x, y) = 2T_3 \sum_{m=1}^{\infty} \frac{1 - \cos\,(m\pi)}{m\pi} \frac{\sinh\left(\frac{m\pi y}{L}\right)}{\sinh\left(\frac{m\pi W}{L}\right)} sin\left(\frac{m\pi x}{L}\right) \tag{5}$$

In the non-dimensional terms, the analytical solution can written by

$$T^*(x^*, y^*) = T_A^*(x^*, y^*) + T_B^*(x^*, y^*) \tag{6}$$

$$T_A^*(x^*, y^*) = 2 \sum_{m=1}^{\infty} \frac{1 - \cos\,(m\pi)}{m\pi} \frac{\sinh\left(\frac{m\pi(1-y^*)}{\alpha}\right)}{\sinh\left(\frac{m\pi}{\alpha}\right)} sin(m\pi x^*) \tag{7}$$

$$T_B^*(x^*, y^*) = 2 * \frac{1}{4} \sum_{m=1}^{\infty} \frac{1 - \cos\,(m\pi)}{m\pi} \frac{\sinh\left(\frac{m\pi y^*}{\alpha}\right)}{\sinh\left(\frac{m\pi}{\alpha}\right)} sin(m\pi x^*) \tag{8}$$

For the numerical solution of the Eq. (1), of the various existing finite difference formulations, the so-called "five-point formula" is the most commonly used. In this representation of the partial difference equation, central differencing which is second-order accurate is utilized. Therefore, the Eq. (2) is approximated for the node (i,j) as

$$\frac{T^*_{i-1,j}-2T^*_{i,j}+T^*_{i+1,j}}{(\Delta x^*)^2} + \alpha^2 \frac{T^*_{i,j-1}-2T^*_{i,j}+T^*_{i,j+1}}{(\Delta y^*)^2} = 0 \quad (9)$$

The grid points are shown in the Fig. (2).

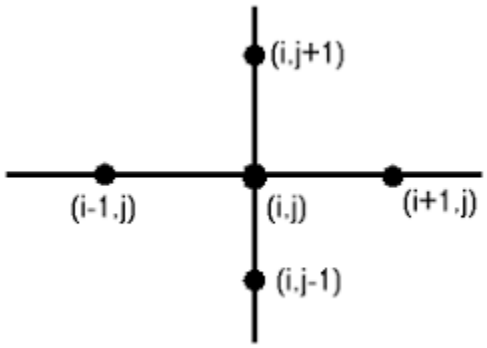

Figure 2 Stencil diagram for 5-point method as 2nd order central difference scheme

The order of accuracy of the 5-point central difference formula is $(\Delta x^{*2}, \Delta y^{*2})$. For higher accuracy, formulation with 9-points can be considered which is fourth order accurate in the space. For the simplicity and understanding the computational algorithms, the study is represented considering the 5-point formulation.

Further simplifying the Eq. (9),

$$T^*_{i-1,j} - 2T^*_{i,j} + T^*_{i+1,j} + \beta^2\left(T^*_{i,j-1} - 2T^*_{i,j} + T^*_{i,j+1}\right) = 0 \quad (10)$$

where, $\beta = {L\Delta x^*}/{W\Delta y^*} = \alpha * \left({\Delta x^*}/{\Delta y^*}\right)$

$$T^*_{i-1,j} + T^*_{i+1,j} + \beta^2\left(T^*_{i,j-1} + T^*_{i,j+1}\right) - 2(1+\beta^2)T^*_{i,j} = 0 \quad (11)$$

For solving the Eq. (11), there are two types of methods available: direct methods and indirect methods. The direct methods include the Gaussian elimination method, which is computationally costly as order of (n^3). The indirect methods are iterative methods, which solve the Eq. (11) by computing solution based on the previously computed solution and repeating the process until a specified convergence is obtained. There are various iterative methods, out of which Point Successive Over-Relaxation method and Alternating Direction Implicit method are discussed here in terms of spatial accuracy, relaxation factors and computational complexity of each algorithm.

2.1 Point Successive Over-Relaxation (PSOR) Method

PSOR method is an improved Gauss-Seidel iterative method, in which the convergence rate is accelerated by extrapolating the solution for the next iteration in the direction of change of the variable. The step by step calculation of the Eq. (11) using PSOR method is explained as follows:

Step-1: Consider the Point Gauss-Seidel iterative method, the solution for the variable $T^*_{i,j}$ as shown in the Eq. (12)

$$\overline{T_{i,j}^*}^{k+1} = \frac{1}{2(1+\beta^2)}\left[{T_{i-1,j}^*}^{k+1} + {T_{i+1,j}^*}^{k} + \beta^2\left({T_{i,j-1}^*}^{k+1} + {T_{i,j+1}^*}^{k}\right)\right] \quad (12)$$

Step-2: Consider the extrapolation of the solution in the $k+1$ iteration as shown in the Eq. (13)

$${T_{i,j}^*}^{k+1} = {T_{i,j}^*}^{k} + \omega\left(\overline{T_{i,j}^*}^{k+1} - {T_{i,j}^*}^{k}\right) \quad (13)$$

where, ω is the relaxation factor. For the solution to converge, it is necessary that $0 < \omega < 2$. If $0 < \omega < 1$, then it is called under-relaxation and for $1 < \omega < 2$, it is called over-relaxation method. It can be seen from the Eq. (13) that the solution for the next iteration ${T_{i,j}^*}^{k+1}$ is extrapolated in the direction of vector $\left(\overline{T_{i,j}^*}^{k+1} - {T_{i,j}^*}^{k}\right)$ by the value ω. And for $\omega = 1$, the Eq. (13) is same as the solution by Point Gauss-Seidel method (Eq. (12)). Simplifying the Eq. (13),

$${T_{i,j}^*}^{k+1} = (1-\omega){T_{i,j}^*}^{k} + \omega\overline{T_{i,j}^*}^{k+1} \quad (14)$$

Step-3: From the Eq. (12) and (14), the value of the non-dimensional temperature for the next iteration can be solved by,

$${T_{i,j}^*}^{k+1} = (1-\omega){T_{i,j}^*}^{k} + \frac{1}{2(1+\beta^2)}\left[{T_{i-1,j}^*}^{k+1} + {T_{i+1,j}^*}^{k} + \beta^2\left({T_{i,j-1}^*}^{k+1} + {T_{i,j+1}^*}^{k}\right)\right] \quad (15)$$

The solution using the PSOR method converges faster than Point Gauss-Seidel method, but the rate of acceleration depends on the value of the relaxation factor. There is no general guideline available for calculating the optimum value of relaxation factor. We can obtain the optimum value by trial and error method. The dependency of the number of iterations for the solution to converge on the relaxation factor is further discussed in detail in the section 3.4.

2.2 Alternating Direction Implicit (ADI) Method

ADI method is two stage method in which the discretized equation is to be solved over x-direction in first step and then in y-direction in the second step. With calculating the discretized equation, the relaxation factor is also considered to add acceleration to convergence.

Step-1: (x-sweep)

For the iteration over the x-direction, the Eq. (10) can be written by

$${T_{i-1,j}^*}^{k+1/2} - 2{T_{i,j}^*}^{k+1/2} + {T_{i+1,j}^*}^{k+1/2} + \beta^2\left({T_{i,j-1}^*}^{k+1/2} - 2{T_{i,j}^*}^{k+1/2} + {T_{i,j+1}^*}^{k}\right) = 0 \quad (16)$$

The relaxation factor can be added to the system by

$${T_{i,j}^*}^{k+1/2} = (1-\omega){T_{i,j}^*}^{k} + \omega\overline{T_{i,j}^*}^{k+1/2} \quad (17)$$

From the Eq. (16) and (17), simplified equation for x-sweep can be written as

$$\omega{T_{i-1,j}^*}^{k+1/2} - 2(1+\beta^2){T_{i,j}^*}^{k+1/2} + \omega{T_{i+1,j}^*}^{k+1/2} = -2(1+\beta^2)(1-\omega){T_{i,j}^*}^{k} - \omega\beta^2\left({T_{i,j-1}^*}^{k+1/2} + {T_{i,j+1}^*}^{k}\right) \quad (18)$$

Equation (18) can be written in terms of matrix as

$$\begin{bmatrix} -2(1+\beta^2) & \omega & 0 & \cdots & 0 \\ \omega & -2(1+\beta^2) & \omega & 0 & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \omega \\ 0 & \cdots & 0 & \omega & -2(1+\beta^2) \end{bmatrix} \begin{bmatrix} T_{2,j}^{*\ k+1/2} \\ \vdots \\ T_{i,j}^{*k+1/2} \\ \vdots \\ T_{Nx-1,j}^{*k+1/2} \end{bmatrix} = \begin{bmatrix} -2(1+\beta^2)(1-\omega)T_{2,j}^{k} - \omega\beta^2(T_{2,j-1}^{k+1/2} + T_{2,j+1}^{k}) - \omega T_{1,j}^{k+1/2} \\ \vdots \\ -2(1+\beta^2)(1-\omega)T_{i,j}^{k} - \omega\beta^2(T_{i,j-1}^{k+1/2} + T_{i,j+1}^{k}) \\ \vdots \\ -2(1+\beta^2)(1-\omega)T_{Nx-1,j}^{k} - \omega\beta^2(T_{Nx-1,j-1}^{k+1/2} + T_{Nx-1,j+1}^{k}) - \omega T_{Nx,j}^{k+1/2} \end{bmatrix} \quad (19)$$

The matrix in Eq. (19) is a tri-diagonal matrix, which can be solved by the Thomas algorithm mentioned in the appendix 5.6.

Step-2: (y-sweep)

Now, for iterations in the y-direction, the Eq. (10) can be written by

$$T_{i,j-1}^{*\ k+1} - 2T_{i,j}^{*\ k+1} + T_{i,j+1}^{*\ k+1} + \beta^2\left(T_{i-1,j}^{*\ k+1} - 2T_{i,j}^{*\ k+1} + T_{i+1,j}^{*\ k+1/2}\right) = 0 \quad (20)$$

The relaxation factor can be added to the system by

$$T_{i,j}^{*\ k+1} = (1-\omega)T_{i,j}^{*\ k+1/2} + \omega\overline{T_{i,j}^{*}}^{k+1} \quad (21)$$

From the Eq. (20) and (21), simplified equation for x-sweep can be written as

$$\omega\beta^2 T_{i,j-1}^{*\ k+1} - 2(1+\beta^2)T_{i,j}^{*\ k+1} + \omega\beta^2 T_{i,j+1}^{*\ k+1} = -2(1+\beta^2)(1-\omega)T_{i,j}^{*\ k+1/2} - \omega\left(T_{i-1,j}^{*\ k+1} + T_{i+1,j}^{*\ k+1/2}\right) \quad (22)$$

Equation (22) can be written in terms of matrix as

$$\begin{bmatrix} -2(1+\beta^2) & \omega\beta^2 & 0 & \cdots & 0 \\ \omega\beta^2 & -2(1+\beta^2) & \omega\beta^2 & 0 & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \omega\beta^2 \\ 0 & \cdots & 0 & \omega\beta^2 & -2(1+\beta^2) \end{bmatrix} \begin{bmatrix} T_{i,2}^{*\,k+1} \\ \vdots \\ T_{i,j}^{*k+1} \\ \vdots \\ T_{i,Ny-1}^{*k+1} \end{bmatrix} = \begin{bmatrix} -2(1+\beta^2)(1-\omega)T_{i,2}^{k+1/2} - \omega(T_{i-1,2}^{k+1} + T_{i+1,2}^{k+1/2}) - \omega\beta^2 T_{i,1}^{k+1} \\ \vdots \\ -2(1+\beta^2)(1-\omega)T_{i,j}^{k+1/2} - \omega(T_{i-1,j}^{k+1} + T_{i+1,j}^{k+1/2}) \\ \vdots \\ -2(1+\beta^2)(1-\omega)T_{i,Ny-1}^{k+1/2} - \omega(T_{i-1,Ny-1}^{k+1} + T_{i+1,Ny-1}^{k+1/2}) - \omega\beta^2 T_{i,Ny}^{k+1} \end{bmatrix} \quad (23)$$

Again, the matrix in Eq. (23) is a tri-diagonal matrix, which can be solved by the Thomas algorithm mentioned in the appendix 5.6.

In both the numerical methods, the convergence criteria to be considered is

$$\frac{\sum_{j=2}^{Ny-1}\sum_{i=2}^{Nx-1} ABS\left(T_{i,j}^{*\,k+1} - T_{i,j}^{*\,k}\right)}{\sum_{j=2}^{Ny-1}\sum_{i=2}^{Nx-1} ABS\left(T_{i,j}^{*\,k}\right)} < \varepsilon \quad (24)$$

where ε is a small number, e.g., 10^{-5}.

3 Discussion of Results

3.1 Temperature Distribution

As shown in the Eq. (15), the point SOR method is implemented to numerically calculate the temperature distribution in the given 2-D domain for steady state condition. The MATLAB code for the algorithm is presented in the Appendix 5.4. The temperature distribution is shown in the Fig. (3) with 40 nodes in x-direction and 30 nodes in y-direction. In all the numerical as well as analytical solutions discussed in this Section 3, are considered to have nodes in each direction in proportion to the physical length in respective directions. For example, the contours presented in the Section 3.1, are generated with 100*L nodes in x-direction and 100*W in y-direction.

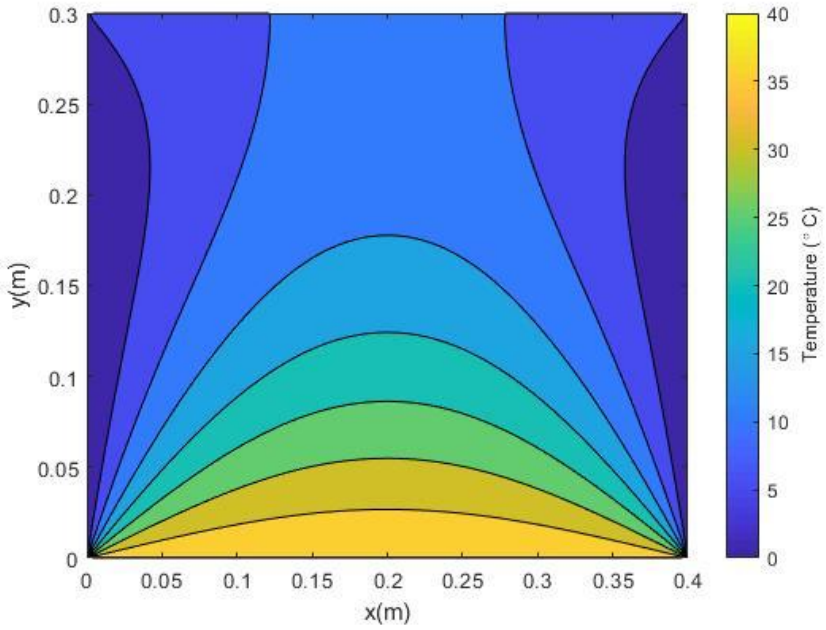

Figure 3 Solution by Point SOR method

In the same way, implementing the ADI method for the given steady state heat diffusion in 2-D domain, the temperature distribution is shown in the Fig. (4). The ADI method includes solving 2 tri-diagonal system in each iterations using the Thomas algorithm. The MATLAB code for ADI method as well as for the Thomas algorithm for tri-diagonal systems is shown in Appendix 5.5 and 5.6 respectively.

The analytical solution discussed in the Eq. (3), (4) and (5) is shown in the Fig. (5) using the same node configuration as point SOR and ADI method. The MATLAB code for the analytical solution is presented in the Appendix 5.2.

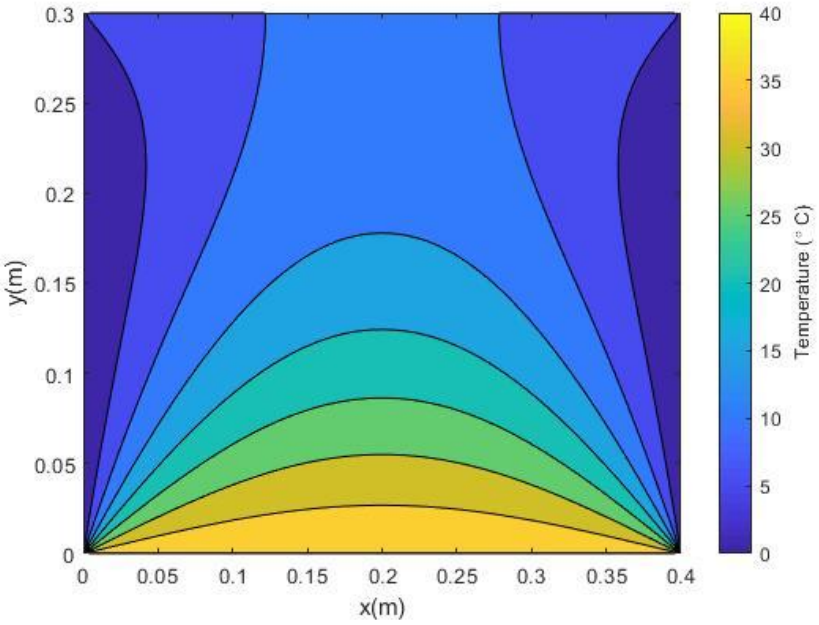

Figure 4 Solution by ADI method

The function written in MATLAB for both ADI and point SOR method are independent of the value of the temperature provided as the boundary conditions. But the analytical method is specifically for the given temperature condition – symmetric in x-direction. Although, the temperature values can be changed for T_1 and T_3 for analytical solutions and it will give the correct results. In the case of point SOR and ADI method, it is also necessary to provide Diritchlet type of boundary conditions for the code to give correct results.

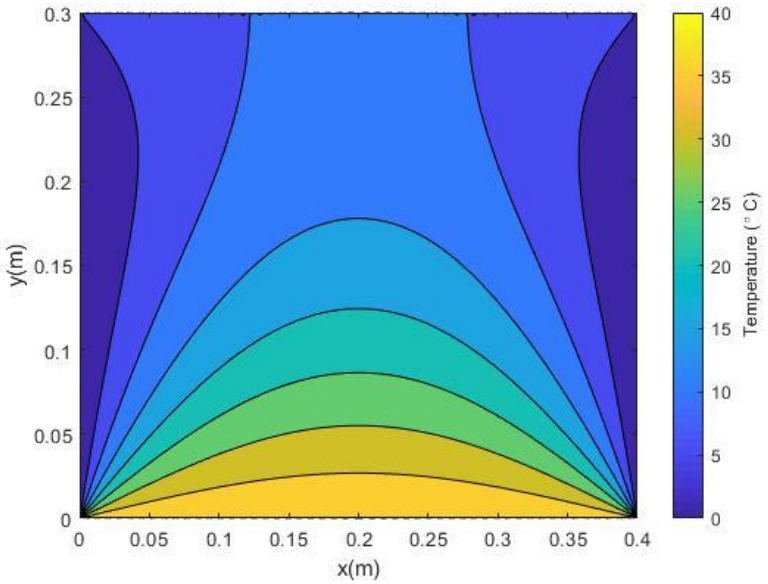

Figure 5 Analytical solution

3.2 Sensitivity Analysis

Before further discussing over the accuracy of the methods, a grid sensitivity analysis is presented in the Fig. (6) and (7) for point SOR and ADI method respectively. The temperature values on the symmetry axis (x = 0.2 m line) is shown in the Fig. (6) and (7). Here, r in the legend is the number being multiplied with the length in the respective direction. For example, for r = 150, total nodes in x-direction, Nx = r*L = 150*0.4 = 60 and total nodes in y-direction, Ny = r*W = 150*0.3 = 45.

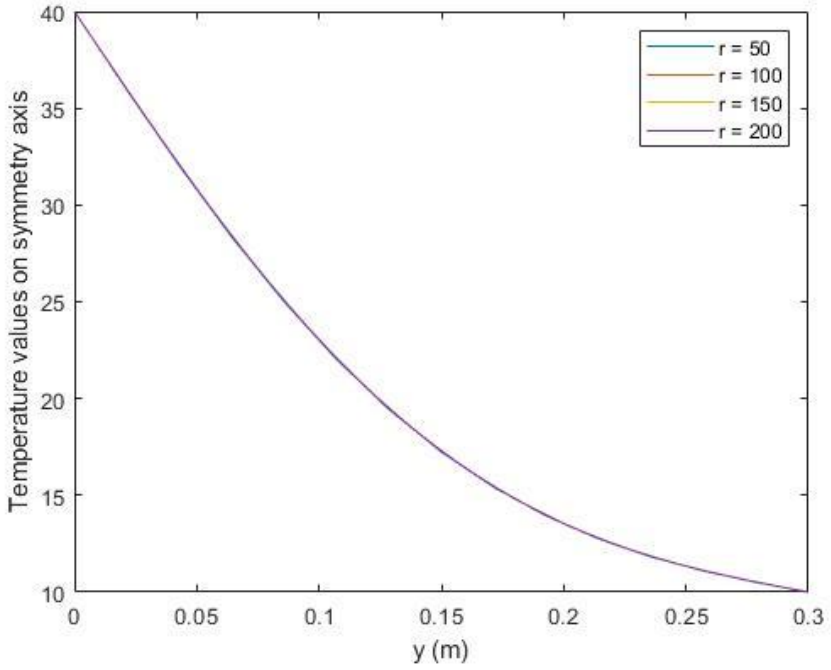

Figure 3 Grid sensitivity analysis in Point SOR method

The plot in both the figures overlap over a single curve and shows that the difference between the calculated temperature values is almost zero. From this conclusion, for the results discussed in further study (Section 3.3 to 3.5), the grid size is considered within this range only.

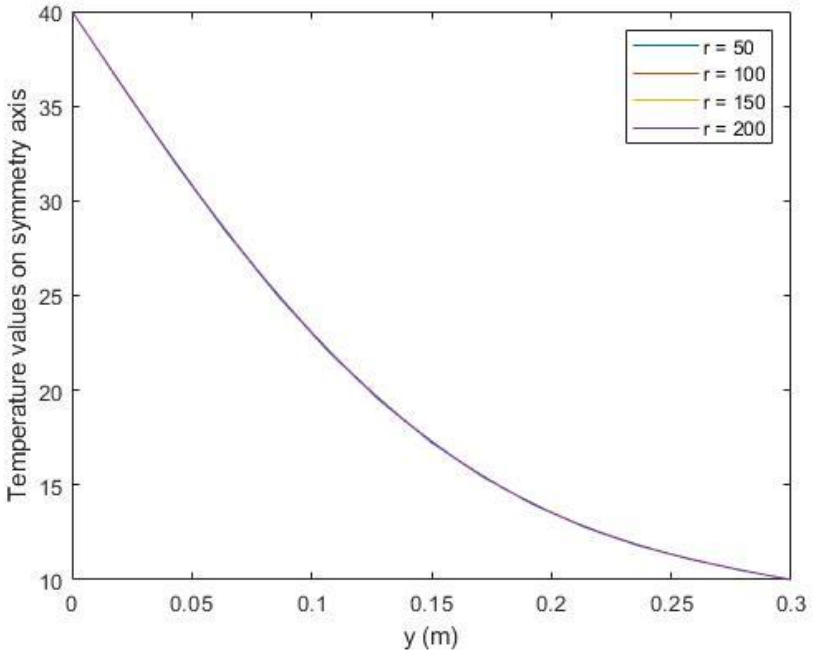

Figure 4 Grid sensitivity analysis in ADI method

3.3 Accuracy Analysis

The error in the space is calculated by

$$error = \frac{1}{Ny*Nx}\sqrt{\sum_{j=2}^{Ny-1}\sum_{i=2}^{Nx-1}\left(T_{i,j}^{computed} - T_{i,j}^{analytical}\right)^2} \quad (25)$$

The error calculated by the Eq. (25) in point SOR and ADI method are presented in the Fig. (8) and (9) respectively. The graphs in these figures are in log-log scale, which provides easy calculation of the spatial accuracy in terms of order of the variable. Figure (8) represents the spatial error in the x-direction and figure (9) represents spatial errors in the y-direction. Here the number of nodes considered in each case is within range of r = 50 to 200. Thus, the ratio of the nodes in x-direction and y-direction is kept same with increasing or decreasing the total nodes. The error in each direction is decreasing with increase in the grid size. Both the methods – point SOR and ADI – are having same value of error in both the directions. For higher grid size (r = 200), the point SOR method becomes less accurate compared to ADI method, but still the order of error is in power of -8.5.

If the grid size is further increased, both methods shows increment in spatial errors. This is due to the accuracy of the MATLAB to save the correct value and that is why the analysis is kept within the limit of order of -8.5 for spatial errors.

Also, for the study of spatial errors, the value of relaxation factor is kept near to the optimum value, which is discussed in the Section 3.4. In particular, for the ADI method, $\omega = 1.3$ and for point SOR method, $\omega = 1.8$ is chosen to keep the iterations minimum and obtain the results under optimum effect of the acceleration of relaxation factor.

From the graphs, the slope of the line can be calculated – which shows the order of accuracy in the space for the computational method. For each method, the order of accuracy is same in both direction. The slope for point SOR method is 1.8448 and for ADI method, 1.9358. From the theory, we can show from the Von-Neumann method that both SOR and ADI method are second order accurate in space for elliptical type of equation.

- Order of accuracy of the point SOR method: 1.8448
- Order of accuracy of the ADI method: 1.9358

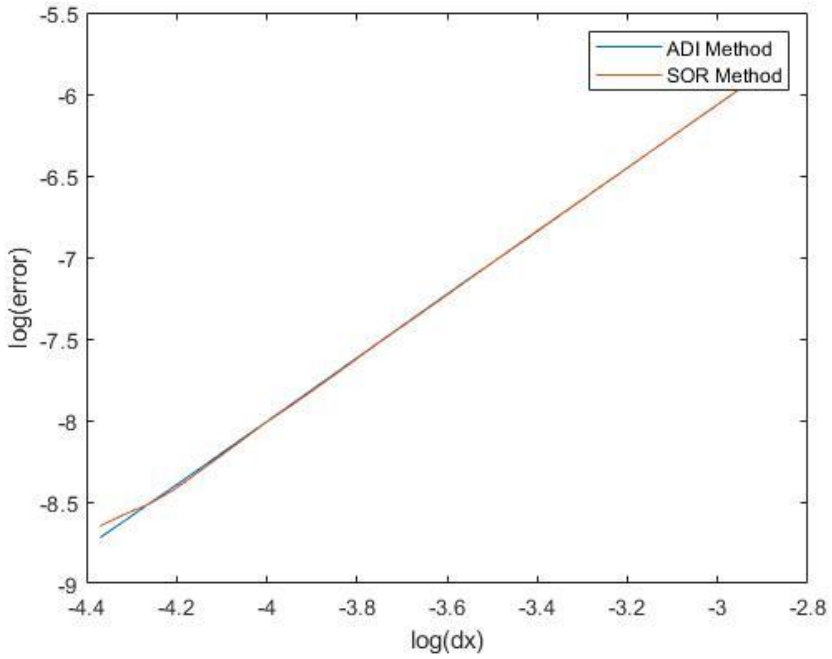

Figure 5 Spatial error for both Point SOR method and ADI method in x-direction

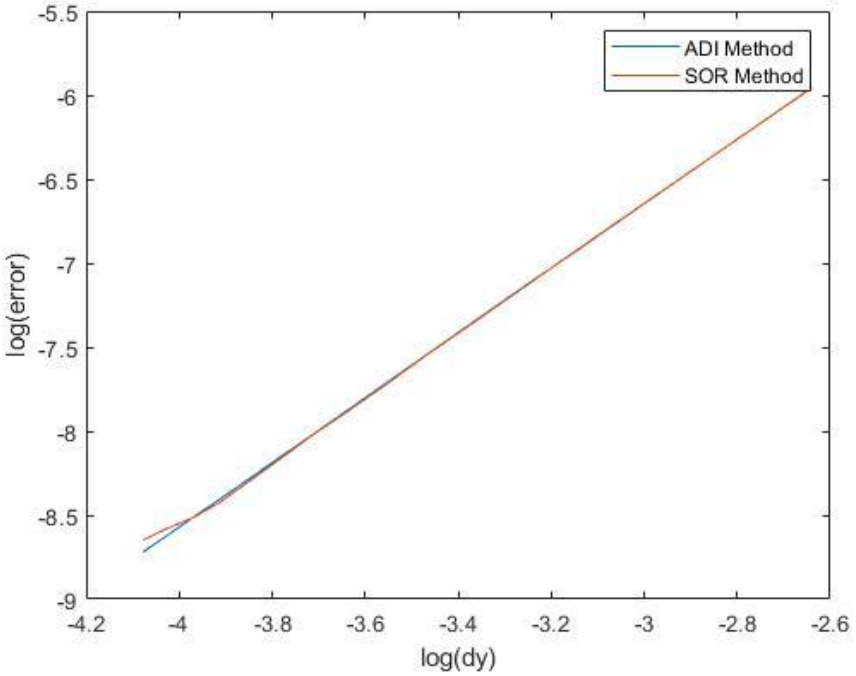

Figure 6 Spatial error in both Point SOR and ADI method in y-direction

3.4 Effect of Relaxation Factor

The relaxation factor in both the methods, is playing a vital role in accelerating the solutions with less number of iterations. The effect of relaxation factor in terms of the number of iterations to reach convergence criteria is shown in the Fig. (10) and Fig. (11) for point SOR and ADI method respectively.

The relaxation factor can be varied from 1 to 2. There is no analogy available to define the optimum relaxation factor (ω_{opt}) for computational methods. As shown in the Fig. (10), for the point SOR method, ω_{opt} is obtained at 1.83 for r = 100 and at 1.91 for r = 200. From the Eq. (14), we can say that for $\omega = 1$, the method becomes the same without any effect of relaxation. In point SOR method, for r = 100, the difference between the total iterations with $\omega = \omega_{opt}$, and $\omega = 1$ is 9.85 times and for r = 200 it is 17 times. In other words, the algorithm becomes 9.85 and 17 times faster than normal condition with usage of ω_{opt} in the case of r = 100 and r = 200 respectively.

In the same way, the effect of relaxation factor is shown in the Fig. (11) is for ADI method. In ADI method, the limiting value for ω is 1.33. Beyond that value, the algorithm provides fluctuations in the results which are not correct. For ADI method, ω_{opt} is obtained at 1.31 for r = 100 and at 1.32 for r = 200. Compared to the number of iterations in normal algorithm, ADI method becomes 9 and 17.6 times faster with $\omega = \omega_{opt}$.

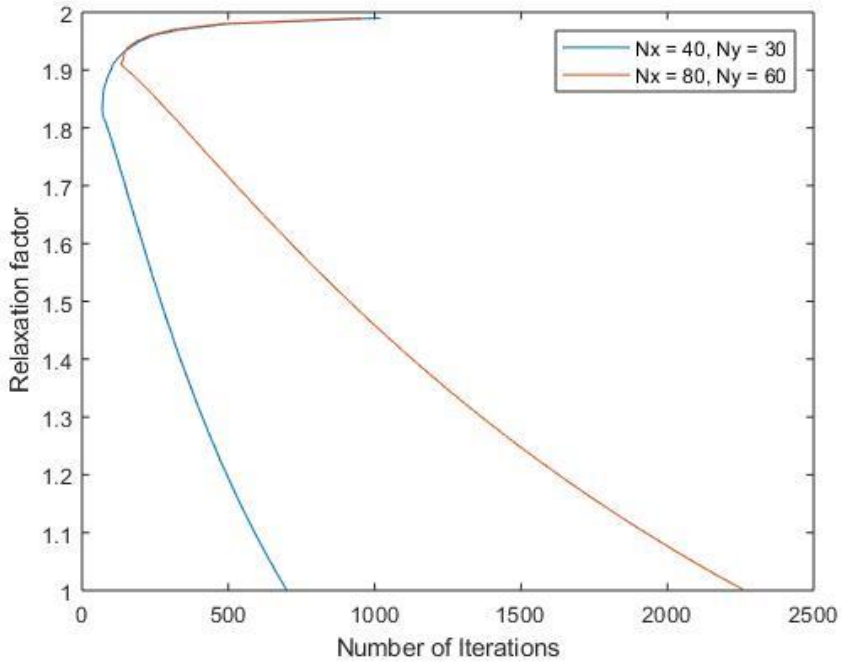

Figure 7 Effect of relaxation factor in Point SOR method

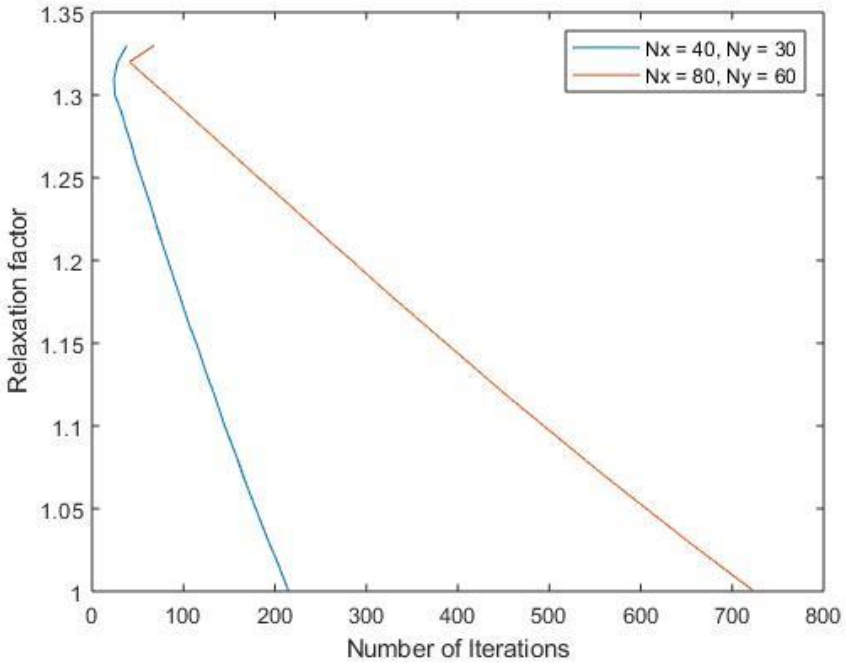

Figure 8 Effect of relaxation factor in ADI method

3.5 Computational Cost and Accuracy

The complexity of the algorithm can be explained in terms of computational time and space which are required for the algorithm to compute the given problem.

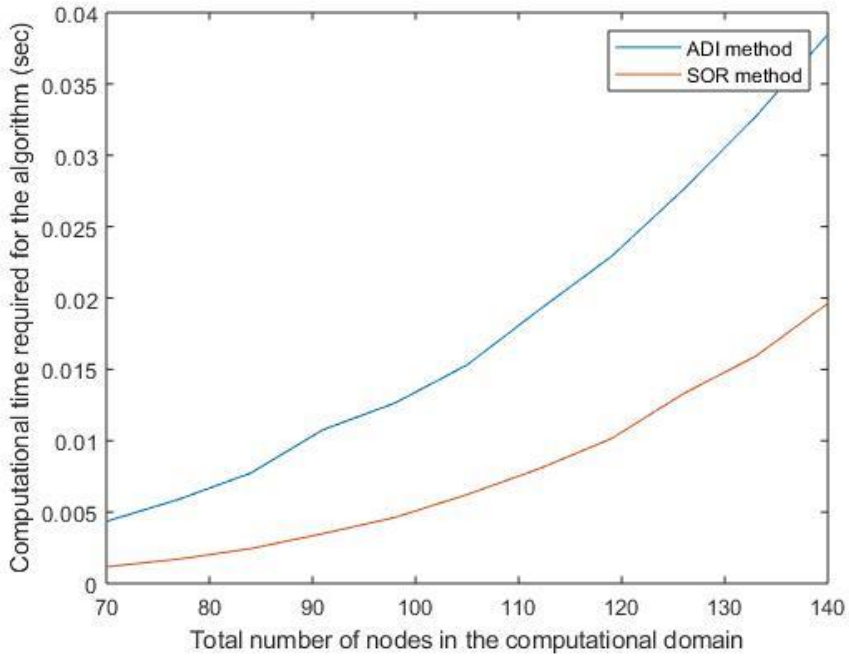

Figure 9 Time complexity comparison in Point SOR method and ADI method by MATLAB function 'timeit'

The computational time is calculated for both the methods by an in-built function in MATLAB called "timeit", which gives the time required for the given function to operate. The values obtained from the function is presented in the Fig. (12) for different total number of nodes in the domain. From the Fig. (12), we can say that the ADI method is actually requiring more computational time than point SOR method. This is because the ADI method is solving the tri-diagonal system twice in one iteration. The ADI method is requiring about 3.5 times more time compared to point SOR method when total nodes are equal to 70. With increase in total number of nodes, the ratio decreases and becomes about 2.

Here, the same criteria is chosen for the value of relaxation factor as used previously for the spatial accuracy analysis. In particular, for the ADI method, $\omega = 1.3$ and for point SOR method, $\omega = 1.8$ is chosen to keep the iterations minimum and obtain the results under optimum effect of the acceleration of relaxation factor.

In terms of number of iterations, we can say that ADI method is requiring less iterations to reach to convergence criteria, but the computational time required to reach to convergence is higher when compared to the point SOR method.

For the space complexity also, the ADI method is costlier than the point SOR method. In point SOR method, only 2 matrices are required to be saved as variable apart from the other common variables. Whereas in ADI method, the algorithm has to save 2 matrices for the temperature variable and other 8 arrays apart

from the common variables to solve the tri-diagonal system of matrix twice in each iteration. This is why the ADI method is costlier than point SOR method in terms of space complexity.

For the accuracy of the computational results, both the methods are providing same amount of error in space – as concluded from the Section 3.4. And describing cost for the computational methods, in terms of time and space complexity of the algorithm, it can be stated that ADI method is costlier than the point SOR method even for the same requirement of accuracy.

4 Summary and Conclusion

The computational methods – point SOR and ADI are used for the analysis of steady state heat diffusion in 2-D domain. Rigorous analysis is done for the spatial accuracy, effect of relaxation factor on accelerated convergence, and computational complexity of the respective algorithm. We can conclude the following points from the study:

- Both the point SOR and ADI methods provide accurate temperature distribution compared to the analytical solution. The sensitivity analysis concludes that the grid size considered in the study is not showing any sensitivity towards the temperature distribution. However, the accuracy obtained depends on the grid size of the computational domain.
- The spatial accuracy of both the methods is order of 2, as per the theoretical analysis by Von-Neumann method. In the computational calculations, the point SOR method is projecting spatial accuracy of the order of 1.8448 and in the same way ADI method is found spatially accurate of order of 1.9358, which are near to the theoretical values for the given heat diffusion equation.
- Here, both the computational method uses the acceleration in convergence using the relaxation factor. The variance in the value of relaxation factor affects the total number of iterations required for reaching the convergence criteria. By trial and error, the optimum value of the relaxation factor is obtained for both methods. It can be seen that the relaxation factor accelerates the convergence about 9 times faster for the coarse grid size, whereas for the fine grid size, the ratio of iterations without and with the relaxation factor reaches to 17. The relaxation factor is more useful with the fine grid sizes.
- From the computational complexity analysis of the point SOR and ADI methods, it can be concluded that the ADI method is quite costly in both aspects – computational time as well as the space (memory required for using the algorithm) compared to the point SOR method for the given heat diffusion problem in 2-D domain. Both the methods predicts the temperature distribution with exact same spatial accuracy for the given grid size, but compared to point SOR method, ADI method is solving 2 tri-diagonal systems in each iteration which makes it computationally costlier.

5 Appendices

5.1 Analytical Solution

```
clear
L = 0.4;
W = 0.3;
alpha = L/W;
r = 200;
Nx = r*L;
x = linspace(0,1,Nx);
Ny = r*W;
y = linspace(0,1,Ny);
T1 = 40;
T2 = 0;
T3 = 10;
T4 = 0;
T31 = T3/T1;

T = analyticf(alpha,Nx,Ny,T31);
contourf(x*L,y*W,T1*T)
xlabel('x(m)')
ylabel('y(m)')
c = colorbar;
c.Label.String = 'Temperature (\circ C)';
```

5.2 Analytical Function

```
function [T,n] = analyticf(alpha,Nx,Ny,T31)

T = zeros(Ny,Nx);

for i = 1:Nx
    x = (i-1)/(Nx-1);
    for j = 1:Ny
        y = (j-1)/(Ny-1);
        TA = 0;
        TB = 0;
        for m = 1:300
            A = ((1-cos(m*pi))*(sin(m*pi*x))*(sinh(m*pi*(1-
y)/alpha))/((m*pi)*(sinh(m*pi/alpha))));
            B = ((1-
cos(m*pi))*(sin(m*pi*x))*(sinh(m*pi*(y)/alpha))/((m*pi)*(sinh(m*pi/alpha))));
            TA = TA + A;
            TB = TB + B;
        end
        T(j,i) = 2*TA + 2*(T31)*TB;
    end
end
end
```

5.3 SOR/ADI Solution

```
clear
L = 0.4;
W = 0.3;
alpha = L/W;
r = 200;
Nx = r*L;
x = linspace(0,1,Nx);
Ny = r*W;
y = linspace(0,1,Ny);
T1 = 40;
T2 = 0;
T3 = 10;
T4 = 0;
T11 = T1/T1;
T21 = T2/T1;
T31 = T3/T1;
T41 = T4/T1;
w = 1.8;

[T,n] = SORf(alpha,Nx,Ny,T11,T21,T31,T41,w);
%[T,n] = ADIf(alpha,Nx,Ny,T11,T21,T31,T41,w);
contourf(x*L,y*W,T*T1)
xlabel('x(m)')
ylabel('y(m)')
c = colorbar;
c.Label.String = 'Temperature (\circ C)';
disp(n)
```

5.4 SOR Function

```
function [T,n] = SORf(alpha,Nx,Ny,T11,T21,T31,T41,w)

Ta = zeros(Nx,Ny);
e = 1;
dx = 1/(Nx-1);
dy = 1/(Ny-1);
b = alpha*dx/dy;

%assigning boundary values
Ta(:,1) = T11;
Ta(:,Ny) = T31;

Ta(1,:) = T21;
Ta(Nx,:) = T41;

Tb = Ta;

%SOR iterations
n = 0;
while e > (10^-5)
    Ta = Tb;
    for i = 2:Nx-1
        for j = 2:Ny-1
```

```
            Tb(i,j) = ((Tb(i+1,j)) + (Tb(i-1,j)) + (b^2)*(Tb(i,j+1)) +
(b^2)*(Tb(i,j-1)))/(2*(1+(b^2)));
            Tb(i,j) = ((1-w)*(Ta(i,j)) + w*(Tb(i,j)));
        end
    end
    e1 = abs(Tb - Ta);
    e2 = sum(e1,1);
    e2 = sum(e2,2);
    e3 = sum(Ta,1);
    e3 = sum(e3,2);
    e = e2/e3;
    n = n + 1;
end
T = Ta';
end
```

5.5 ADI Function

```
function [T,n] = ADIf(alpha,Nx,Ny,T11,T21,T31,T41,w)

Ta = zeros(Nx,Ny);
e = 1;
dx = 1/(Nx-1);
dy = 1/(Ny-1);
b = alpha*dx/dy;

%assigning boundary values
Ta(:,1) = T11;
Ta(:,Ny) = T31;

Ta(1,:) = T21;
Ta(Nx,:) = T41;

%ADI iterations
n = 0;
while e > 10^-5
    Tb = Ta;

    %x-sweep
    for j = 2:(Ny-1)

        %Matrix formation
        for i = 2:(Nx)
            px(1,i) = w;
            qx(1,i) = -2*(1+(b^2));
            rx(1,i) = w;
            sx(1,i) = (-2*(1+(b^2))*(1-w)*Ta(i,j)) - (w*(b^2)*(Ta(i,j+1) +
Ta(i,j-1)));
        end

        sx(1,2) = sx(1,2) - (w)*Ta(1,j);
        sx(1,Nx-1) = sx(1,Nx-1) - (w)*Ta(Nx,j);

        Ta(:,j) = TRI(2,Nx-1,px,qx,rx,sx);
```

```
        %Re-assigning boundary values
        Ta(:,1) = T11;
        Ta(:,Ny) = T31;

        Ta(1,:) = T21;
        Ta(Nx,:) = T41;
    end


    %y-sweep
    for i = 2:(Nx-1)

        %Matrix formation
        for j = 2:(Ny)
            py(1,j) = w*(b^2);
            qy(1,j) = -2*(1+(b^2));
            ry(1,j) = w*(b^2);
            sy(1,j) = (-2*(1+(b^2))*(1-w)*Ta(i,j)) - (w*(Ta(i+1,j) + Ta(i-
1,j)));
        end

        sy(1,2) = sy(1,2) - (w)*(b^2)*Ta(i,1);
        sy(1,Ny-1) = sy(1,Ny-1) - (w)*(b^2)*Ta(i,Ny);

        Ta(i,:) = TRI(2,Ny-1,py,qy,ry,sy);

        %Re-assigning boundary values
        Ta(:,1) = T11;
        Ta(:,Ny) = T31;

        Ta(1,:) = T21;
        Ta(Nx,:) = T41;
    end

    e1 = abs(Ta - Tb);
    e2 = sum(e1,1);
    e2 = sum(e2,2);
    e3 = sum(Tb,1);
    e3 = sum(e3,2);
    e = e2/e3;
    n = n + 1;
end
T = Ta';
end
```

5.6 Tri-diagonal matrix solver

```
%Tri-diagonal matrix solver
function [s] = TRI(i1,i2,p,q,r,s)

    %fwd substitution
    for i = (i1+1):i2
```

```
        q(1,i) = q(1,i) - r(1,i-1)*(p(1,i)/q(1,i-1));
        s(1,i) = s(1,i) - s(1,i-1)*(p(1,i)/q(1,i-1));
    end

    %bwd substitution
    s(1,i2) = s(1,i2)/q(1,i2);
    for i = (i2-1):-1:i1
        s(1,i) = (s(1,i) - (r(1,i)*s(1,i+1)))/(q(1,i));
    end
end
```

5.7 Grid sensitivity analysis

```
clear
L = 0.4;
W = 0.3;

T1 = 40;
T2 = 0;
T3 = 10;
T4 = 0;

i = 0;
Tas = zeros(4,200);
for r = 50:50:200
    Nx = r*L;
    x = linspace(0,L,Nx);
    Ny = r*W;
    y = linspace(0,W,Ny);

    i = i + 1;
    w = 1.3;
    [Ta,~] = ADIf(L,W,Nx,Ny,T1,T2,T3,T4,w);
    Tas(i,1:Ny) = (Ta(1:Ny,(Nx/2))+Ta(1:Ny,((Nx/2)+1)))/2;
    figure(1)
    plot(y,Tas(i,1:Ny))
    xlabel('y (m)')
    ylabel('Temperature values on symmetry axis')
    hold on
end

legend({'r = 50','r = 100','r = 150','r = 200',},'Location','northeast')
hold off

i = 0;
Tss = zeros(4,200);
for r = 50:50:200
    Nx = r*L;
    x = linspace(0,L,Nx);
    Ny = r*W;
    y = linspace(0,W,Ny);

    i = i + 1;
    w = 1.8;
    [Ts,~] = SORf(L,W,Nx,Ny,T1,T2,T3,T4,w);
```

```
    Tn = analyticf(L,W,Nx,Ny,T1,T2,T3,T4);
    Tss(i,1:Ny) = (Ts(1:Ny,(Nx/2))+Ts(1:Ny,((Nx/2)+1)))/2;
    figure(2)
    plot(y,Tss(i,1:Ny))
    xlabel('y (m)')
    ylabel('Temperature values on symmetry axis')
    hold on
end

legend({'r = 50','r = 100','r = 150','r = 200',},'Location','northeast')
hold off
```

5.8 Spatial error analysis

```
clear
L = 0.4;
W = 0.3;

T1 = 40;
T2 = 0;
T3 = 10;
T4 = 0;

i = 0;
for r = 50:10:200
    Nx = r*L;
    x = linspace(0,L,Nx);
    Ny = r*W;
    y = linspace(0,W,Ny);

    i = i + 1;
    w = 1.3;
    [Ta,~] = ADIf(L,W,Nx,Ny,T1,T2,T3,T4,w);
    w = 1.9;
    [Ts,~] = SORf(L,W,Nx,Ny,T1,T2,T3,T4,w);
    Tn = analyticf(L,W,Nx,Ny,T1,T2,T3,T4);

    %Error in ADI Solution
    e1 = 0;
    for j2 = 2:(Nx-1)
        for j1 = 2:(Ny-1)
            e1(j1,j2) = (Ta(j1,j2) - Tn(j1,j2))^2;
        end
    end
    e1 = sum(e1,1);
    e1 = sum(e1,2);
    error1(1,i) = log(sqrt(e1)/(Nx*Ny));

    %Error in SOR Solution
    e2 = 0;
    for j2 = 2:(Nx-1)
        for j1 = 2:(Ny-1)
            e2(j1,j2) = (Ts(j1,j2) - Tn(j1,j2))^2;
        end
    end
```

```
    e2 = sum(e2,1);
    e2 = sum(e2,2);
    error2(1,i) = log(sqrt(e2)/(Nx*Ny));

    dx(1,i) = 1/(Nx-1);
    dx(1,i) = log(dx(1,i));
    dy(1,i) = 1/(Ny-1);
    dy(1,i) = log(dy(1,i));

end

figure('Name','X-direction')
plot(dx,error1)
hold on
plot(dx,error2)
xlabel('log(dx)')
ylabel('log(error)')
legend({'ADI Method','SOR Method'},'Location','northeast')
hold off

slopeadi = diff(error1)./diff(dx);
slopeadi = sum(slopeadi,2)/(i-1);
disp(slopeadi)

slopesor = diff(error2)./diff(dx);
slopesor = sum(slopesor,2)/(i-1);
disp(slopesor)


figure('Name','Y-direction')
plot(dy,error1)
hold on
plot(dy,error2)
xlabel('log(dy)')
ylabel('log(error)')
legend({'ADI Method','SOR Method'},'Location','northeast')
hold off

slopeadi = diff(error1)./diff(dx);
slopeadi = sum(slopeadi,2)/(i-1);
disp(slopeadi)

slopesor = diff(error2)./diff(dx);
slopesor = sum(slopesor,2)/(i-1);
disp(slopesor)
```

5.9 Effect of relaxation factor

```
clear
L = 0.4;
W = 0.3;

T1 = 40;
T2 = 0;
```

```
T3 = 10;
T4 = 0;

for r = 100:100:200
    Nx = r*L;
    x = linspace(0,L,Nx);
    Ny = r*W;
    y = linspace(0,W,Ny);
%{
    i = 0;
    for w = 1.0:0.01:1.33
        i = i + 1;
        rela(1,i) = w;
        [~,na(1,i)] = ADIf(L,W,Nx,Ny,T1,T2,T3,T4,w);
    end
    plot(na,rela)
    hold on

    xlabel('Number of Iterations')
    ylabel('Relaxation factor')
%}
    j = 0;
    for w = 1.0:0.01:1.99
        j = j + 1;
        rels(1,j) = w;
        [~,ns(1,j)] = SORf(L,W,Nx,Ny,T1,T2,T3,T4,w);
    end
    plot(ns,rels)
    hold on

    xlabel('Number of Iterations')
    ylabel('Relaxation factor')
%}
end
hold off
legend('Nx = 40, Ny = 30','Nx = 80, Ny = 60')
```

5.10 Time complexity analysis

```
clear
L = 0.4;
W = 0.3;

T1 = 40;
T2 = 0;
T3 = 10;
T4 = 0;
i = 0;
for r = 100:10:200
    i = i + 1;
    Nx = r*L;
    x = linspace(0,L,Nx);
    Ny = r*W;
    y = linspace(0,W,Ny);
    N = Nx + Ny;
    n(1,i) = N;
```

```
    w = 1.3;
    [Ta,na] = ADIf(L,W,Nx,Ny,T1,T2,T3,T4,w);
    f = @() ADIf(L,W,Nx,Ny,T1,T2,T3,T4,w);
    ta(1,i) = timeit(f);

    w = 1.8;
    [Ts,ns] = SORf(L,W,Nx,Ny,T1,T2,T3,T4,w);
    g = @() SORf(L,W,Nx,Ny,T1,T2,T3,T4,w);
    ts(1,i) = timeit(g);

end

plot(n,ta,n,ts)
legend('ADI method','SOR method')
xlabel('Total number of nodes in the computational domain')
ylabel('Computational time required for the algorithm (sec)')
```