2D Heat conduction problem with irregular boundaries

A final course project by:

Farzad Azizi Zade

Department of Mechanical Engineering
Ferdowsi University of Mashhad
July 2020

Contents

List of figures

1 Project 1 "Algebraic non-uniform grid generation"

1.1 Definition

We are required to write a script to create a non-uniform algebraic grid for the geometry shown in Figure 1-1. Table 1 summarizes the inputs and outputs of the project.

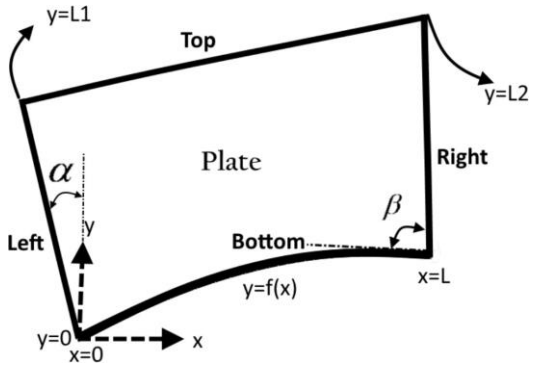

Fig. 1-1: Geometry

Table 1-1: inputs and outputs of Project 1

input/output	variable	definition
input	I max	# of nodes in x-direction
input	J max	# of nodes in y-direction
input	B	Beta Parameter
input	A	Alpha Parameter
input	L, L1, L2, Alpha, Beta	Geometrical Parameters
output	X, Y	X and Y of the generated grid

Modular programming allows us to debug our programs better. Therefore, a function called "Mesh Generator" is programmed to generate the desired mesh based on the maximum number of nodes in X and Y directions and other geometrical parameters. The output of this function is the X and Y coordinates of the generated grid. This part of the project creates a Tecplot

(.plt) compatible file containing the number of nodes in each direction and the coordinates of the respective nodes.

1.2 Results and Discussion

Figure 1-2 shows the algebraic generated non-uniform grid for I max = 100, J max = 100. Due to the noticeable geometrical gradients in all four corners, especially at (0,0), the mesh is more concentrated near the walls. This is illustrated for (0,0) corner in Figure 1-2.

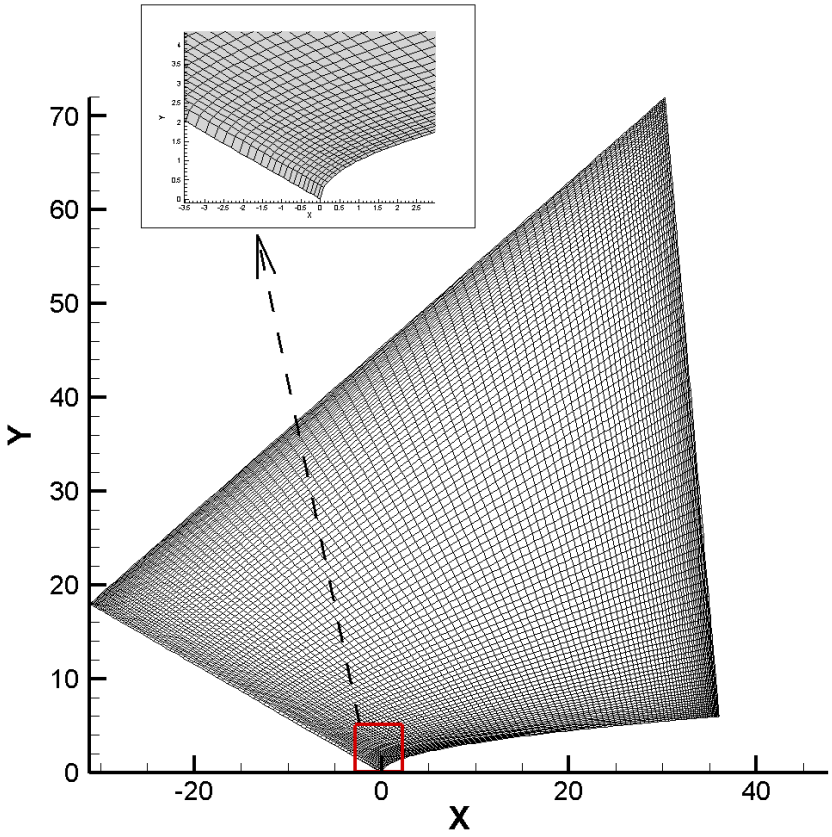

Fig. 1-2: Generated grid with Imax=Jmax=100

Figure 1-3 represents the generated Tecplot (.plt) compatible file.

```
VARIABLES="X","Y","T"
ZONE I= 100 J= 100
0 0 1300
-0.081497 0.047052 130
-0.16809 0.097045 1300
-0.26006 0.15014 1300
-0.35771 0.20652 1300
-0.46135 0.26636 1300
-0.5713 0.32984 1300
-0.68791 0.39716 1300
-0.8115 0.46852 1300
-0.94245 0.54412 1300
-1.0811 0.62418 1300
-1.2279 0.70891 1300
-1.3831 0.79853 1300
-1.5472 0.89327 1300
-1.7205 0.99336 1300
-1.9036 1.099 1300
-2.0966 1.2105 1300
-2.3001 1.328 1300
-2.5145 1.4518 1300
-2.7401 1.582 1300
-2.9774 1.719 1300
-3.2266 1.8629 1300
-3.4881 2.0139 1300
-3.7624 2.1722 1300
-4.0496 2.338 1300
-4.35 2.5115 1300
-4.664 2.6927 1300
-4.9916 2.8819 1300
-5.3331 3.0791 1300
-5.6886 3.2843 1300
-6.0581 3.4976 1300
-6.4416 3.7191 1300
-6.8391 3.9485 1300
-7.2504 4.186 1300
-7.6753 4.4313 1300
-8.1136 4.6844 1300
```

Fig. 1-3: Generated. plt file

2 Project 2 "FDM Approaches for 2D Conduction"

2.1 Definition

The second project aims to solve the Laplace equation (2D Conduction without any source terms), Eqn. 1, on the grid which was generated in Project 1.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \qquad (Eqn.\,1)$$

In order to solve Eqn. 1, one needs to discretize it using finite difference methods. It is important to note that the grid is non-uniform and we need to derive the FDE equations using Taylor series. We will first derive the required equations and then investigate the results of each one in other sections of this report.

$$f(x_{i+1}) = f(x_i) + \Delta x_{i+1} f'(x_i) + \frac{\Delta x_{i+1}^2}{2!} f''(x_i) + \frac{\Delta x_{i+1}^3}{3!} f'''(x_i) + \cdots$$

$$f(x_{i-1}) = f(x_i) - \Delta x_i f'(x_i) + \frac{\Delta x_i^2}{2!} f''(x_i) - \frac{\Delta x_i^3}{3!} f'''(x_i) + \cdots$$

Where $\Delta x_i = x_i - x_{i-1}$ and $\Delta x_{i+1} = x_{i+1} - x_i$.

We need to somehow remove $f'(x_i)$ from the RHS of the equations above. Multiply the first and the second equations by A *and* B, and then solving for A and B, one finds:

$$A_x = \frac{2}{\Delta x_{i+1}^2 + \Delta x_i^2 R_x} \; ; \; A_y = \frac{2}{\Delta y_{j+1}^2 + \Delta y_j^2 R_y}$$

$$B_x = \frac{2R_x}{\Delta x_{i+1}^2 + \Delta x_i^2 R_x} \; ; \; B_y = \frac{2R_y}{\Delta y_{j+1}^2 + \Delta y_j^2 R_y}$$

Finally:

$$T_{i,j} = \frac{1}{A_x + A_y + B_x + B_y}\left(A_x T_{i+1,j} + A_y T_{i,j+1} + B_x T_{i-1,j} + A_y T_{i,j-1}\right) \quad (Eqn.\,2)$$

This is the equation we will use for the mandatory parts of Project 2.

2.2 Mesh Study

Before investigating the other parts of the project, it is necessary to find a suitable and reliable grid size. In order to do that, different grid sizes from I max = J max = 25 to Imax = J max = 100 has been created and the temperature on X = L/2 line using each method has been computed. Figure 2-1 illustrates the mentioned mesh study for the PGS method. As it can be seen, the Imax = J max = 25 is showing promising results. However, after studying other methods, it is best to use I max = J max = 100 grid size.
Therefore, from now on, we are going to use 100*100 grid sizes.

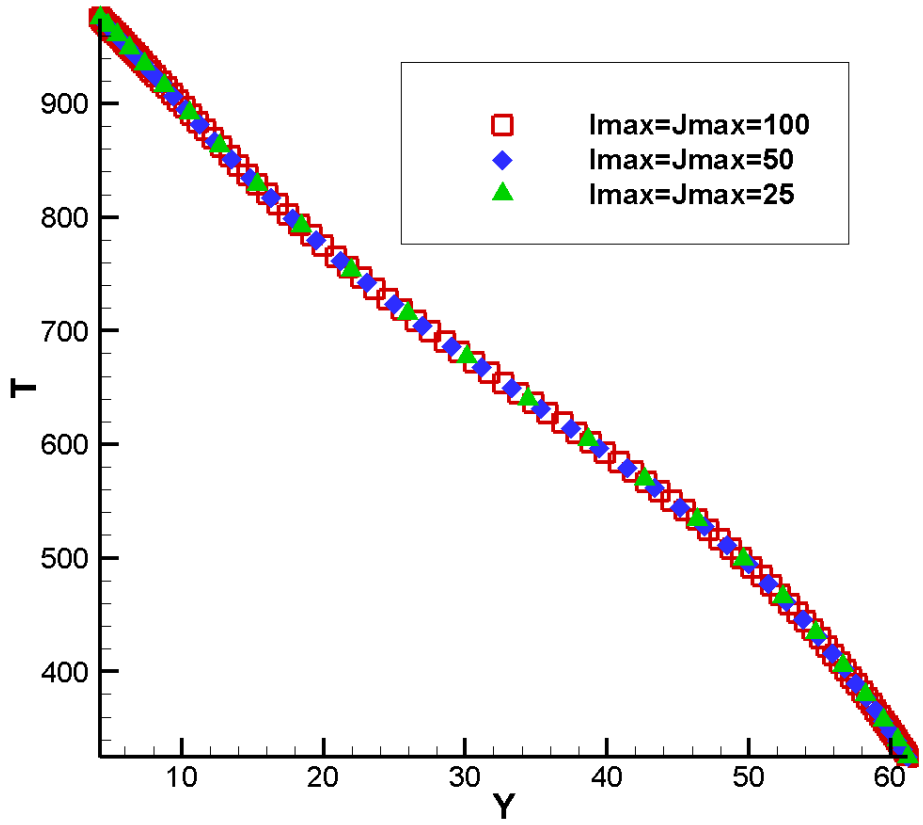

Fig. 2-1: Mesh study for the PGS method

2.3 Temperature contours using Jacobi and Point Gauss Seidel methods:

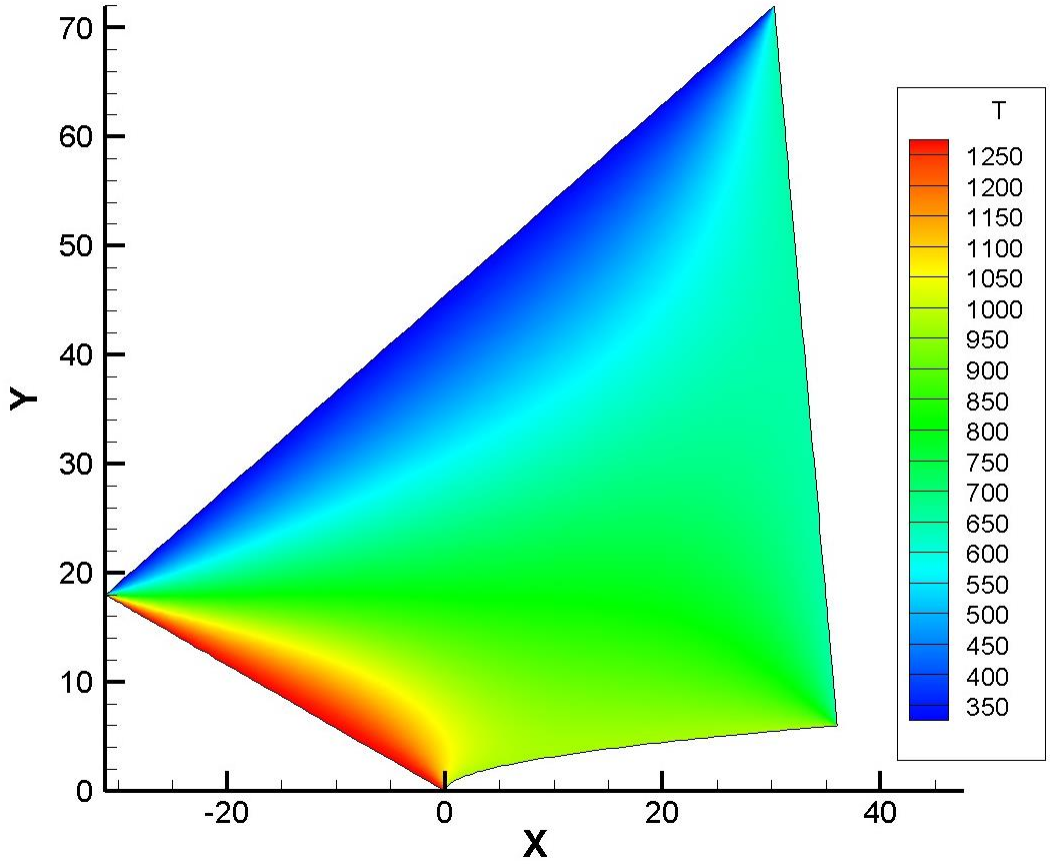

Fig. 2-2: Temperature contour using Jacobi method

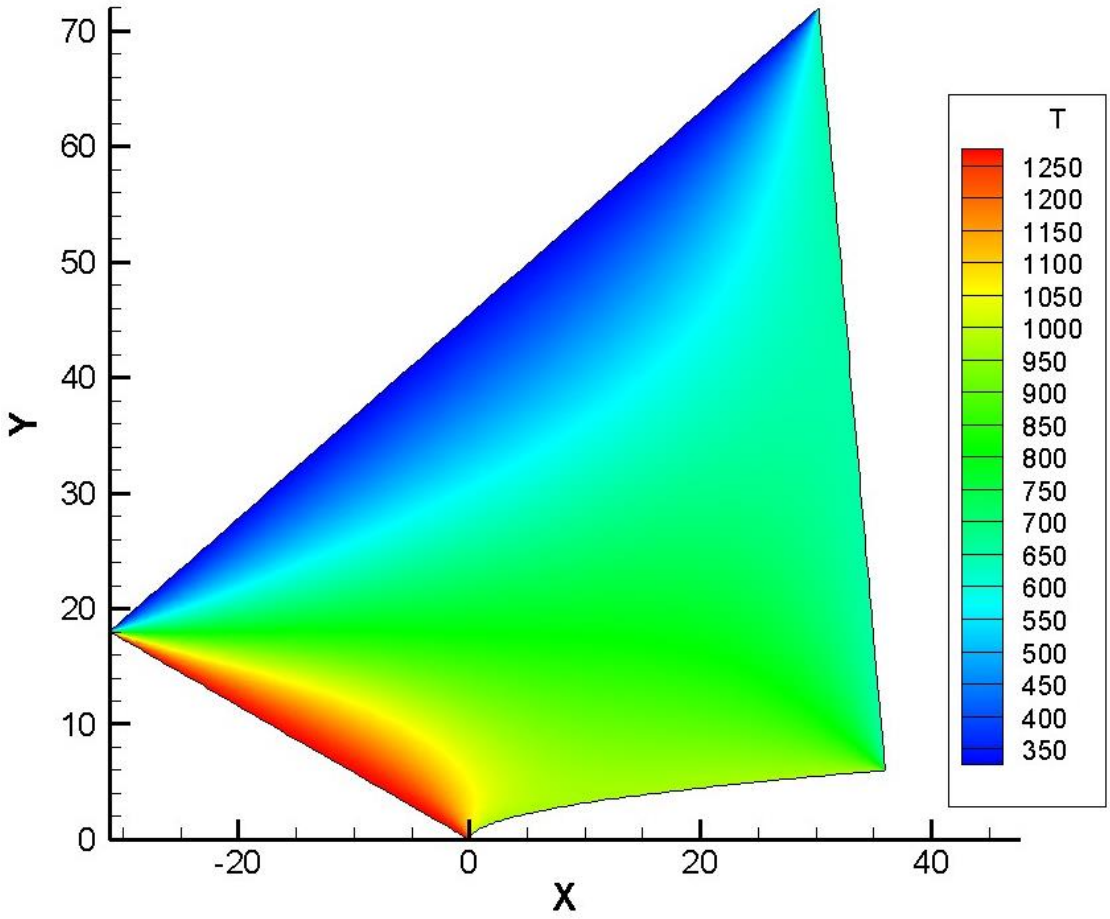

Fig. 2-3: Temperature contour using PGS method

2.4 Comparing PGS and Jacobi methods in regard of their speed and iterations needed to solve the problem:

Figure 1-6 shows how different grid sizes affect the iterations needed in each method to reach the desired same error. As the number of grid nodes increases, the Jacobi method requires more and more iterations in comparison to the Point Gauss Seidel method. For example, at I max = J max = 400, the Jacobi method requires 128750 iterations to reach 1E-04 error, whereas the PGS method requires 70350 iterations to reach the same error. In other words, the iterations needed to reach the same 1E-04 error at Imax = J max = 400, can decrease about 45% by using the PGS method instead of the Jacobi method. One can say that the speed of the PGS is about 45% better than the Jacobi method.

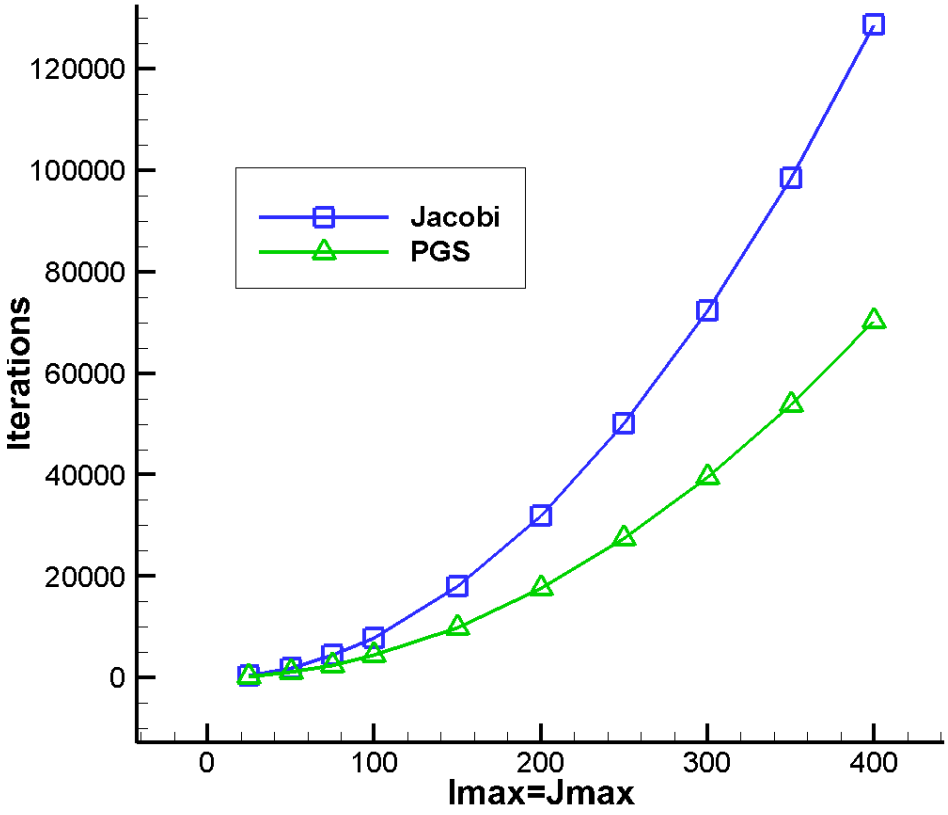

Fig. 2-4: Comparing the iterations needed for reaching the desired errors in different grid sizes

2.5 The effects of initial conditions on temperature distribution and the number of iterations to reach the desired error:

Changing the initial conditions will highly impact the convergence speed in finite difference and other numerical methods. 5 different initial conditions of T Down, T Up, T Left, T Right, and T Mean have been investigated. T Mean is the average of four boundaries. Figures 2-5 and 2- 6 illustrate how changing the initial conditions affect the iterations needed to reach the convergence criteria. In both of the PGS and Jacobi methods, using T Mean as the initial temperature shows the most promising results.

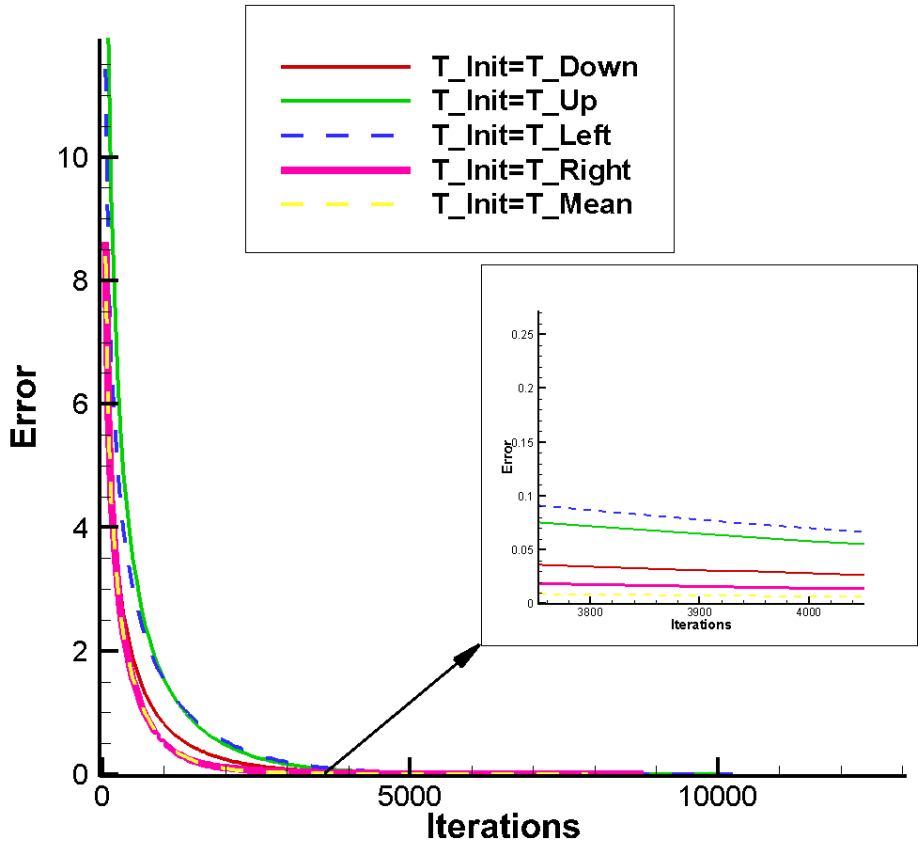

Fig. 2-5: Effects of different initial conditions of the Jacobian method on the error and iterations

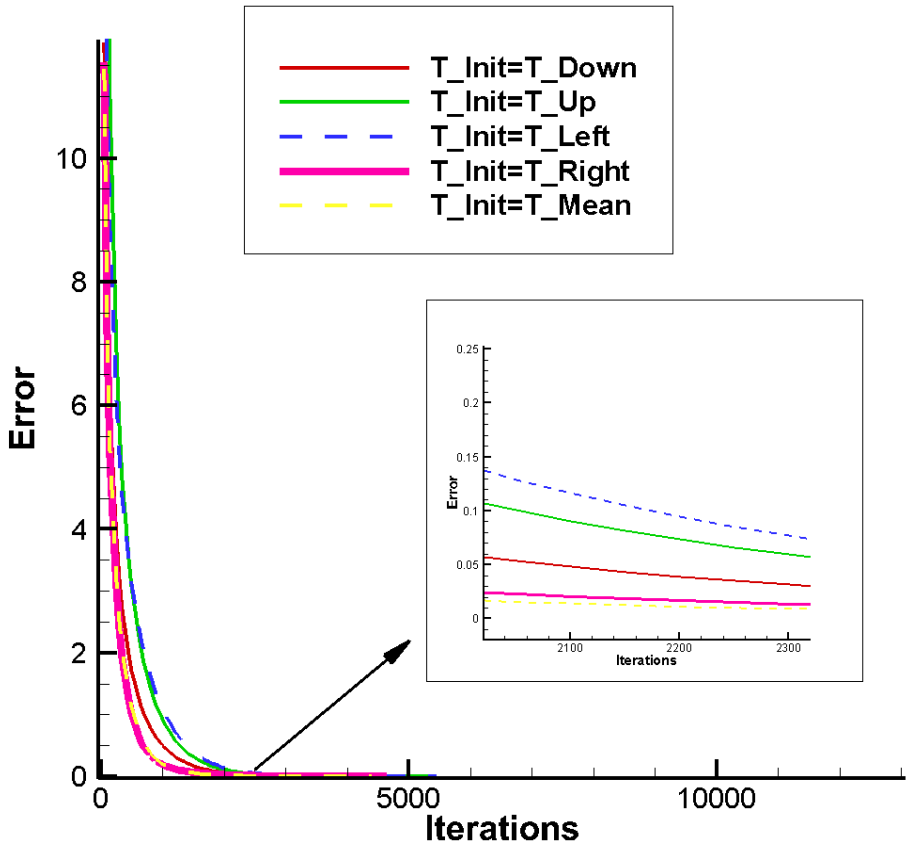

Fig. 2-6: Effects of different initial conditions of the PGS method on the error and iterations

As can be seen in figures 2-5 and 2-6, only two of the conditions, T Mean and T Left need to be examined and see if they affect the final contours. Figures 2-7, 2-8, 2-9, and 2-10 show the final temperature contours of the PGS and Jacobi methods using two different initial conditions. It is obvious that the final temperature does not change noticeably, mainly because we have set the convergence criteria very low (10E-04), and the results will be the same.

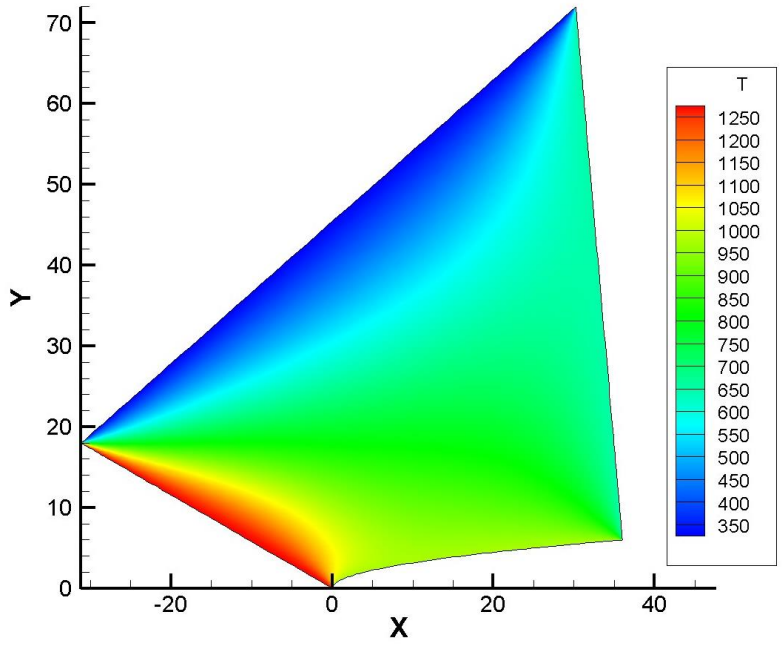

Fig. 2-7: Final temperature contour using the Jacobi method with T Mean as the initial condition

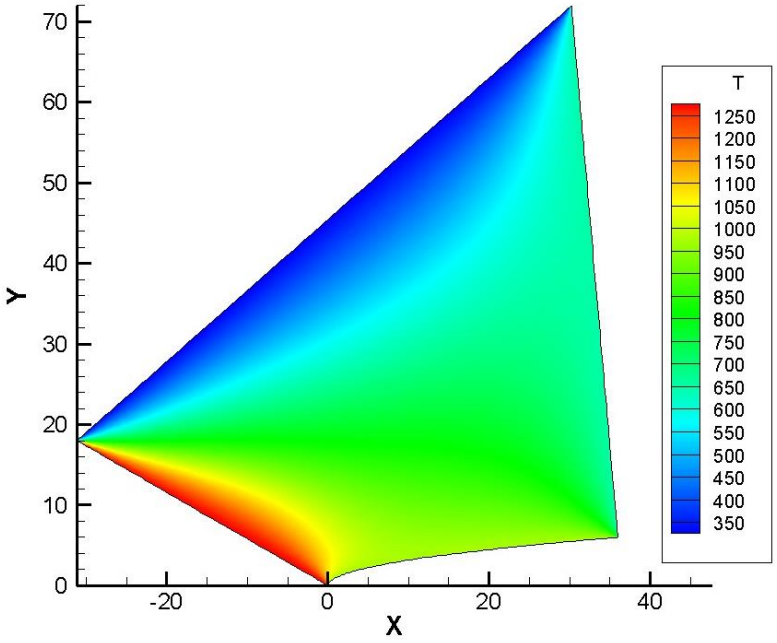

Fig. 2-8: Final temperature contour using the PGS method with T Mean as the initial condition

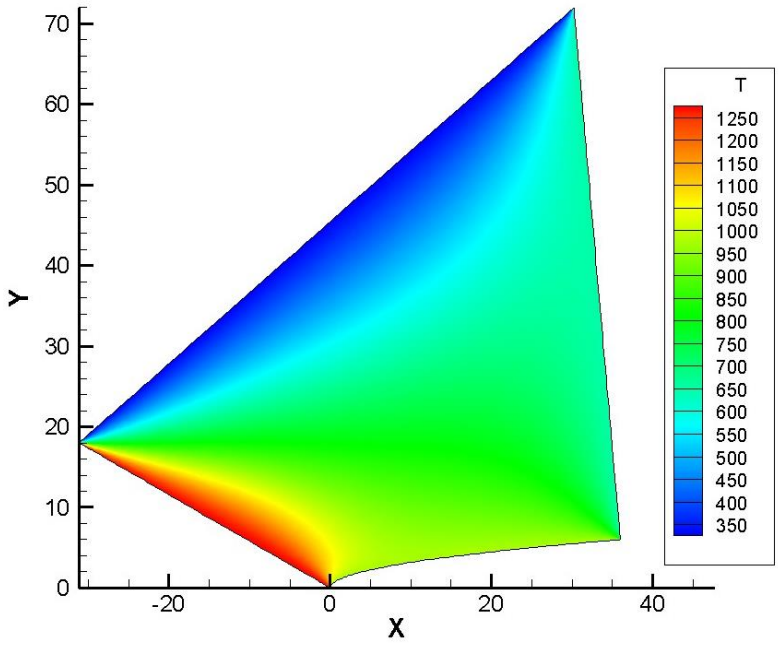

Fig. 2-9: Final temperature contour using the Jacobi method with T Left as the initial condition

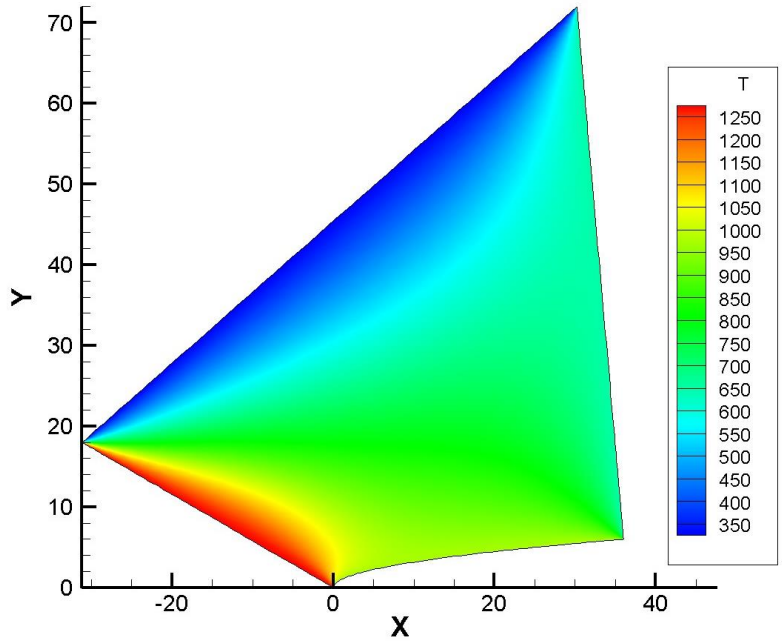

Fig. 2-10: Final temperature contour using the PGS method with T Left as the initial condition

Figure 2-11 shows how many iterations are needed to reach the convergence criteria for both the PGS and Jacobi methods with different initial conditions. It is obvious that the PGS methods requires about half of that of the Jacobi method.

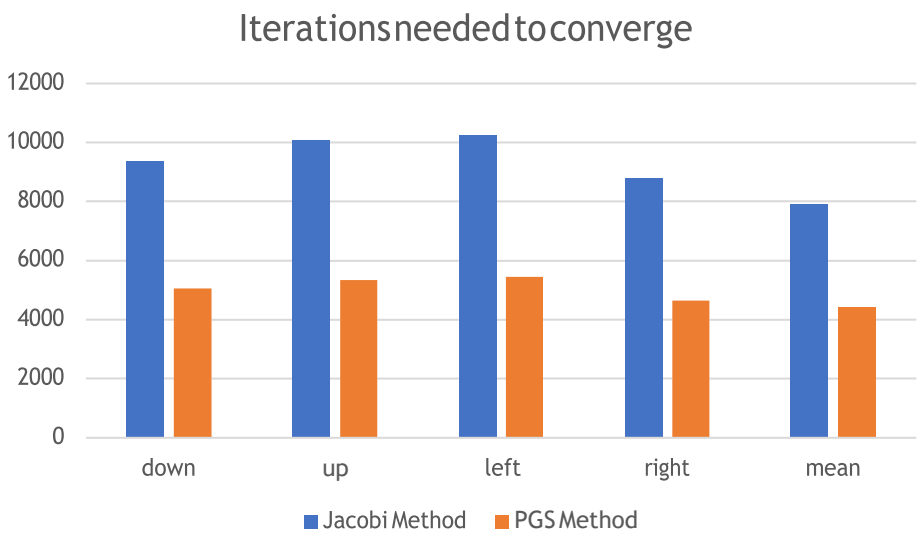

Fig. 2-11: Iterations needed to converge using different initial conditions

2.6 Comparing PGS, Jacobi, and ANSYS Fluent methods on y = L2/2 line

After finding the corresponding nodes (using two conditions -ifs-), the XY Plotter functions writes the coordinates and the temperature of the nodes lying nearest to the y=L2/2 line. It can be seen that both of the PGS and Jacobi methods' solutions are very close, and they both differ from the ANSYS Fluent solution. More will be discussed in section 3.3.

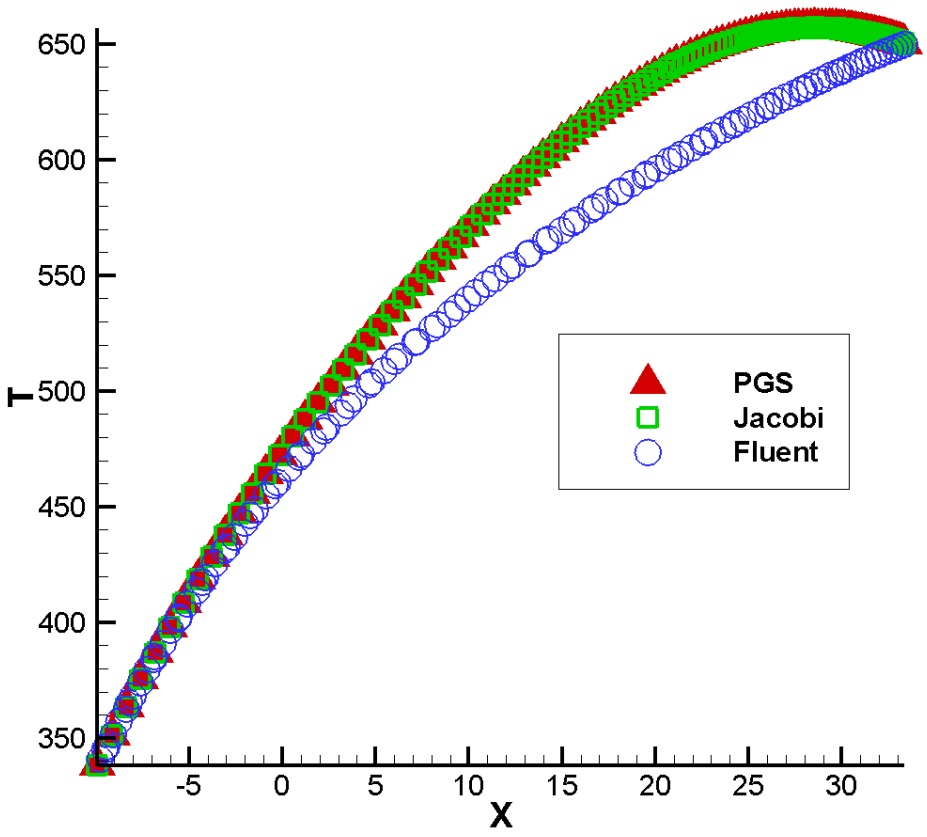

Fig. 2-12: Comparing the PGS, Jacobi, and Ansys Fluent solutions on Y=L2/2

2.7 Comparing PGS, Jacobi, and ANSYS Fluent methods on x = L/2 line

After finding the corresponding nodes (using two conditions -ifs-), the XY Plotter functions writes the coordinates and the temperature of the nodes lying nearest to the x=L/2 line. It can be seen that both of the PGS and Jacobi methods' solutions are very close, and they both differ from the ANSYS Fluent solution. More will be discussed in section 3.3

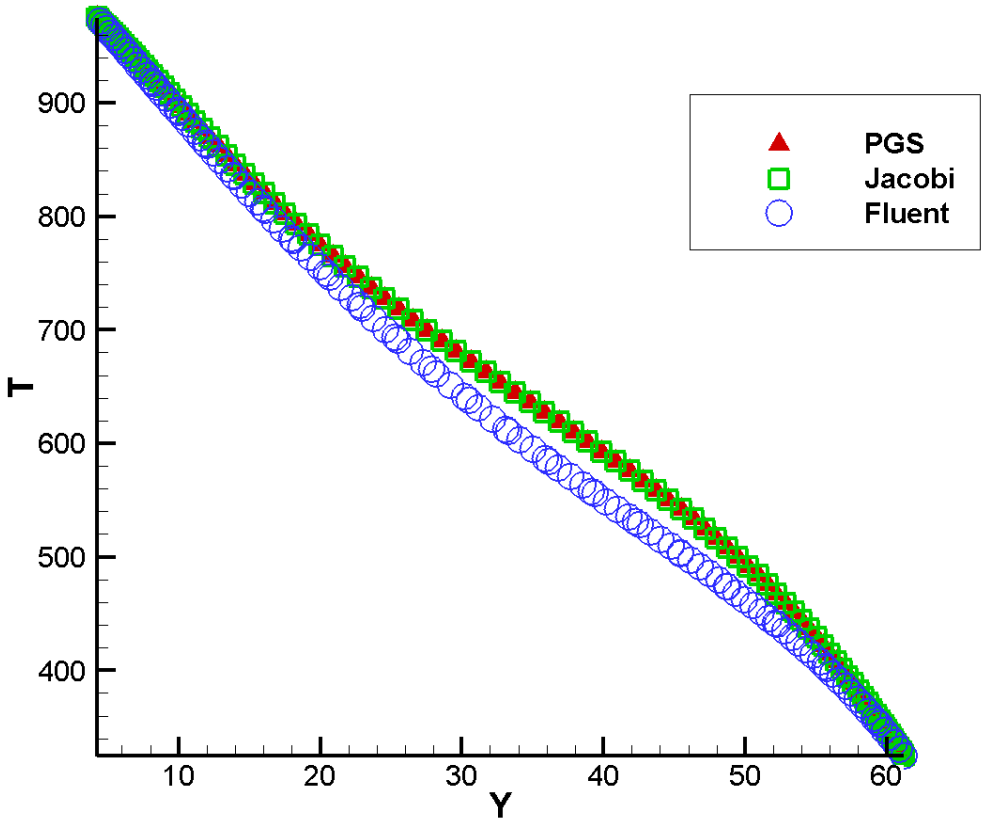

Fig. 2-13: Comparing the PGS, Jacobi, and Ansys Fluent solutions on X=L/2

3 Project 2 Bonus Sections

3.1 Contours of the temperature using LGS, PSOR, and LSOR and comparing the iterations needed using all methods

Figures 3-1, 3-2, and 3-3 show the final temperature contours using LGS, PSOR, and LSOR methods. It is apparent that they do not differ noticeably.

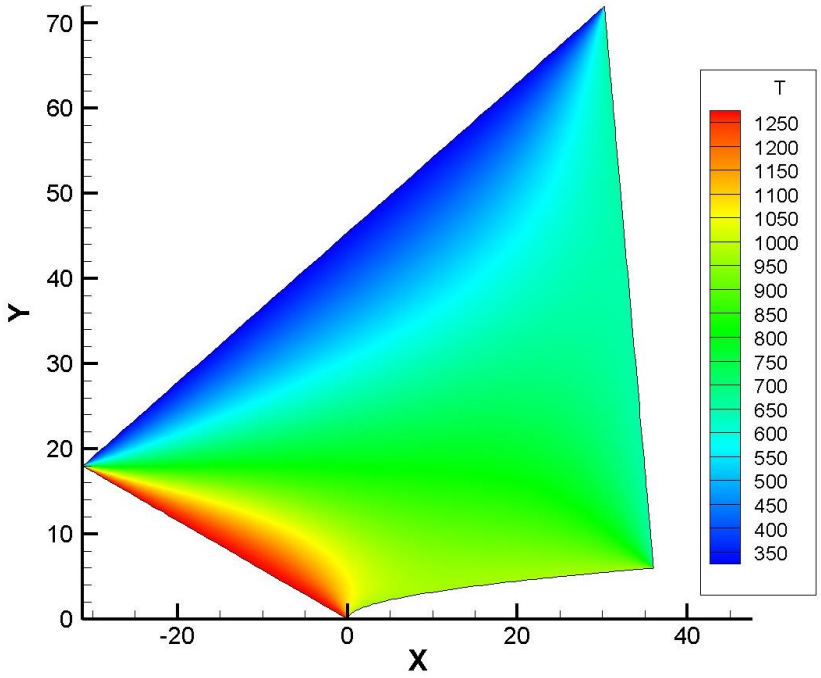

Fig. 3-1: Final temperature contour using the LGS method

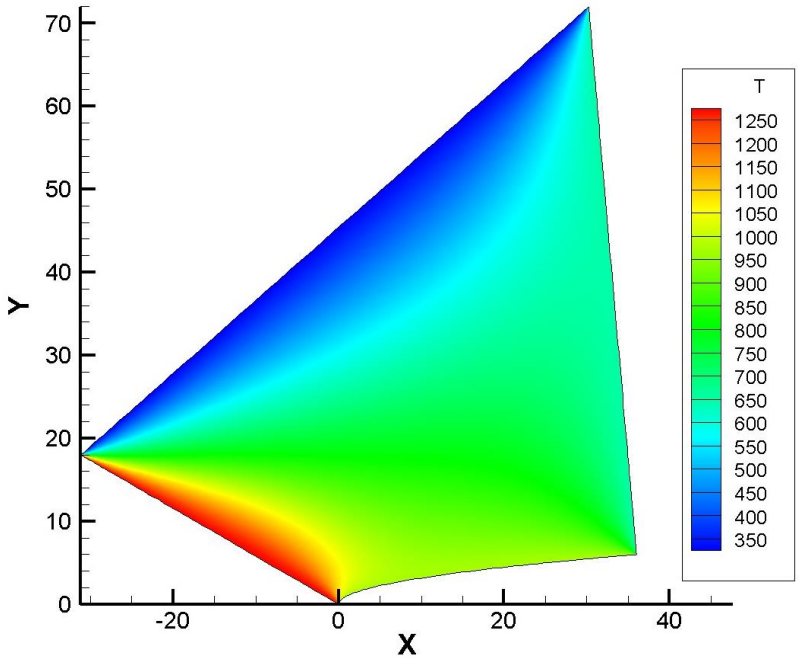

Fig. 3-2: Final temperature contour using the PSOR method

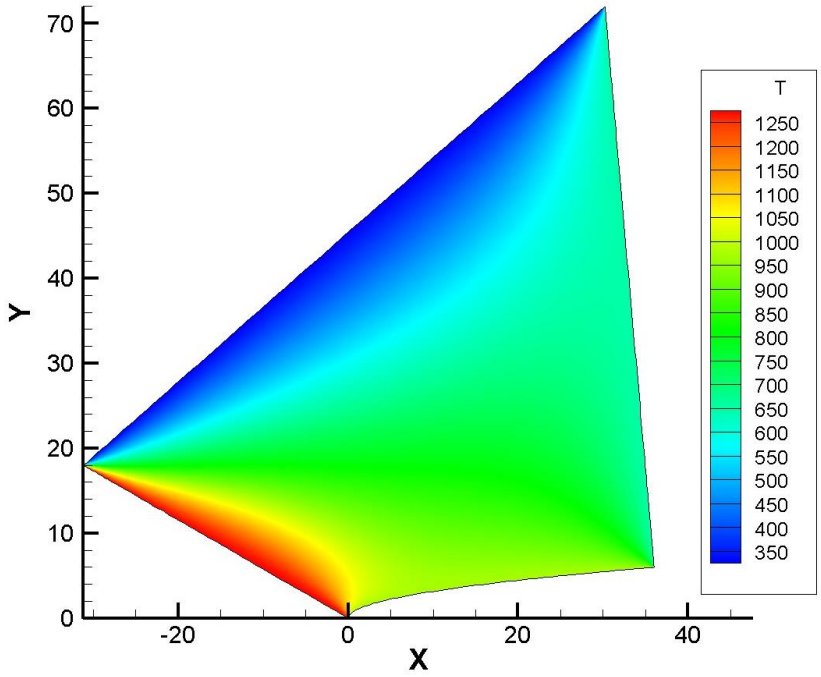

Fig. 3-3: Final temperature contour using the LSOR method

Table 3-1 shows how many iterations are needed to reach the 1E-04 error using 100*100 non-uniform grid.

Table 3-1: # of iterations needed to converge using different methods

Method	**# of Iterations**
Jacobi	7900
PGS	4400
LGS	3150
PSOR	4800 (w=0.95), 4400 (w=1)
LSOR	3550 (w=0.95), 3150 (w=1)

It is evident that the Jacobi method is the slowest one. PGS and LGS are faster, respectively. PSOR and LSOR have the same speed as the PGS and LGS when w=1 and less speed with w<1. More will be discussed in section 3.2.

3.2 Effect of ω on iterations needed using the PSOR and LSOR methods

Figures 3-4 and 3-5 point out how changing the "w" parameter in LSOR and PSOR methods affect the number of iterations needed to converge. It is important to note that the range of possible "w" found for LSOR and PSOR methods are [0.05,1.0213], and [0.05,1.95]. I am not sure why LSOR has such a small range of possible "w" s.

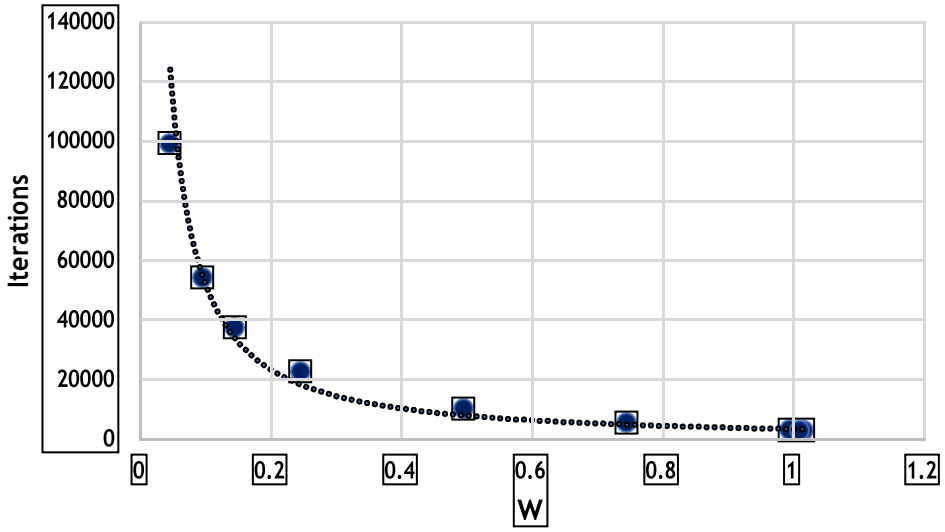

Fig. 3-4: Effect of "w" on iterations needed using LSOR method

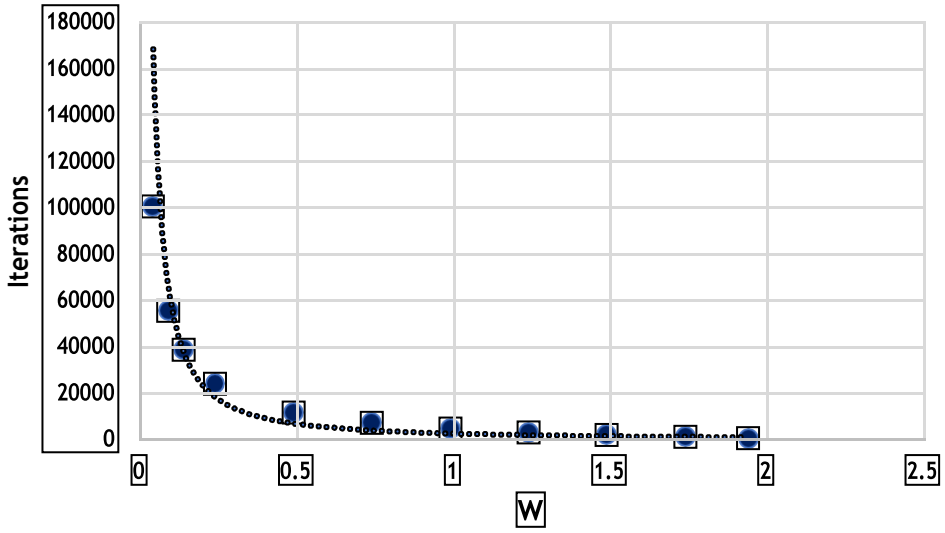

Fig. 3-5: Effect of "w" on iterations needed using PSOR method

It is evident that increasing w dramatically decreases the iterations needed to reach the convergence. On the other hand, decreasing w increases the computational costs.

3.3 Comparing ANSYS Fluent and our FDM solutions

As it was discussed in sections 2.6 and 2.7, the FDM methods' solutions are very identical. However, the ANSYS Fluent results differ from them noticeably. It only takes 7-10 iterations for the ANSYS Fluent to solve the problem, whereas our code takes (at least) 3500 iterations to solve it. The deviation of our results from the ANSYS Fluent, is mainly because of the different methods used, and our simple discretization methods. Figures 3 and 3- illustrate the difference between our code and ANSYS Fluent.

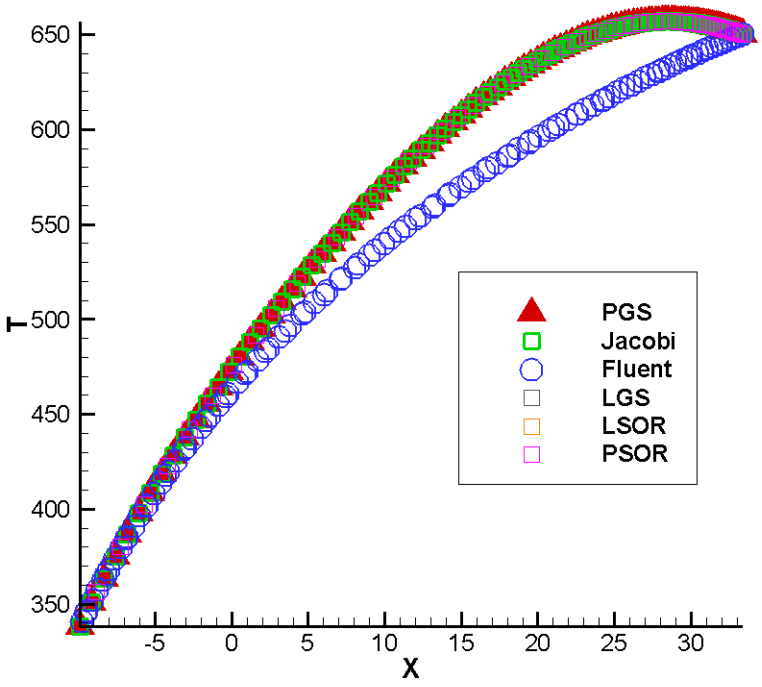

Fig. 3-6: Comparing the PGS, Jacobi, and Ansys Fluent solutions on Y=L2/2

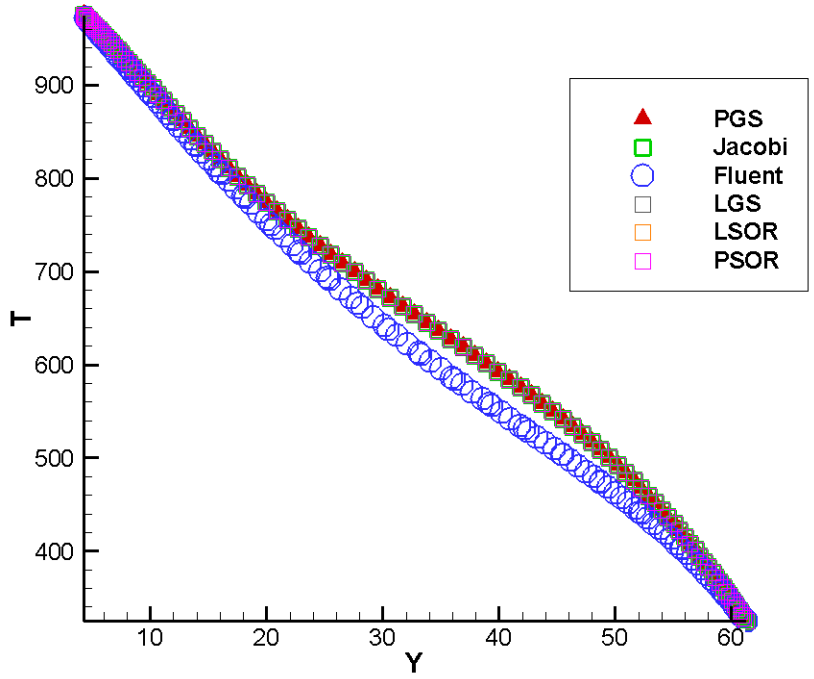

Fig. 3-7: Comparing the PGS, Jacobi, and Ansys Fluent solutions on X=L/2

3.4 insulating the left and right boundaries

This can be done by solving the FDEs on the boundaries, too. However, we should use second-order central difference method to substitute the temperatures outside the boundaries. Figures 3-8, and 3-9 show the contours using our code and ANSYS Fluent.

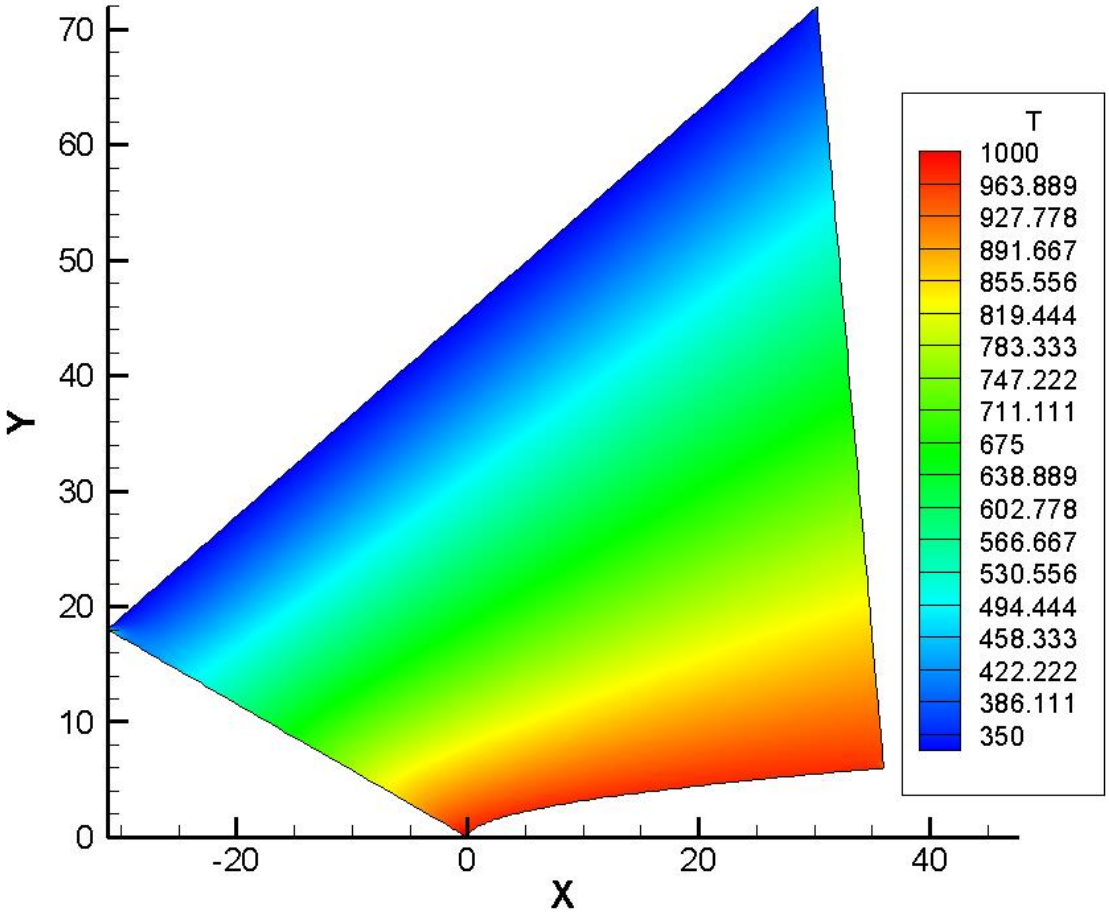

Fig. 3-8: Temperature contour using PGS

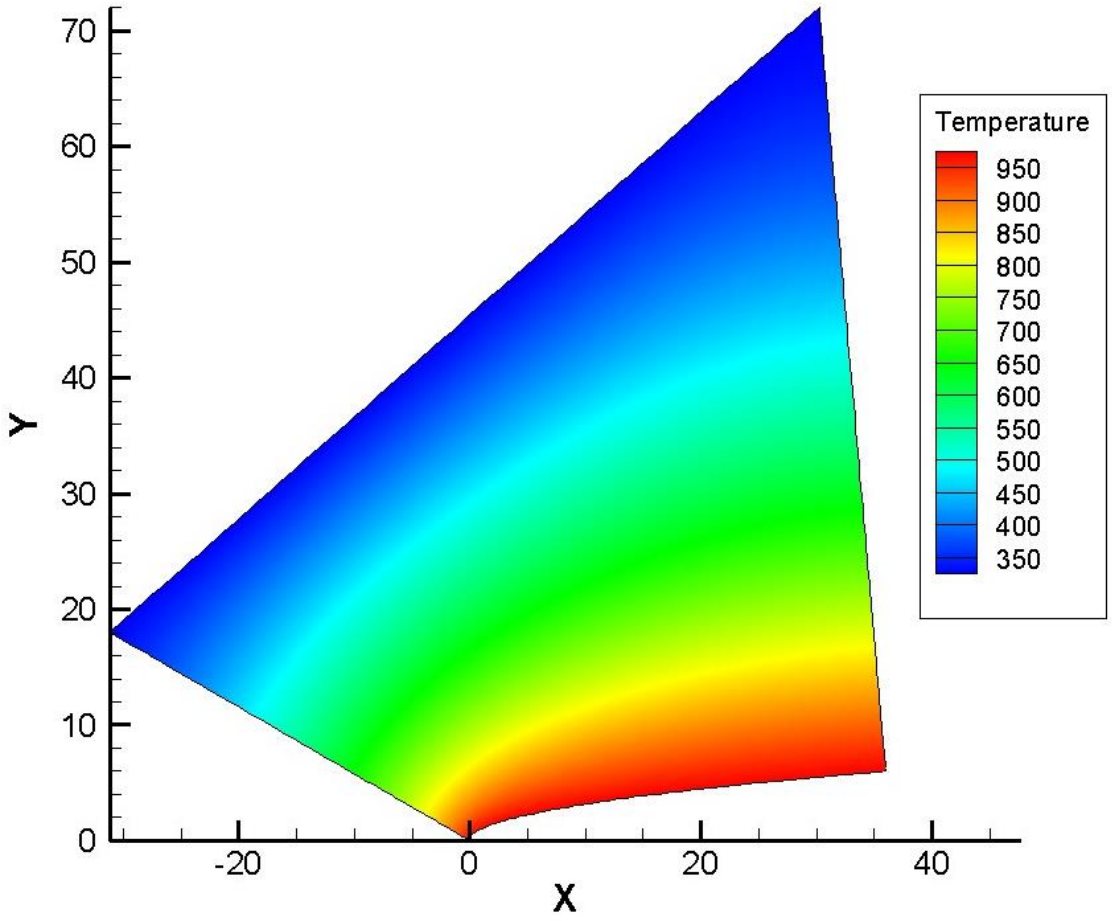

Fig. 3-9: Temperature contour using ANSYS Fluent

3.5 Insulating the upper and lower boundaries

This can be done by solving the FDEs on the boundaries, too. However, we should use second-order central difference method to substitute the temperatures outside the boundaries. Figures 3-10, and 3-11 show the contours using our code and ANSYS Fluent.

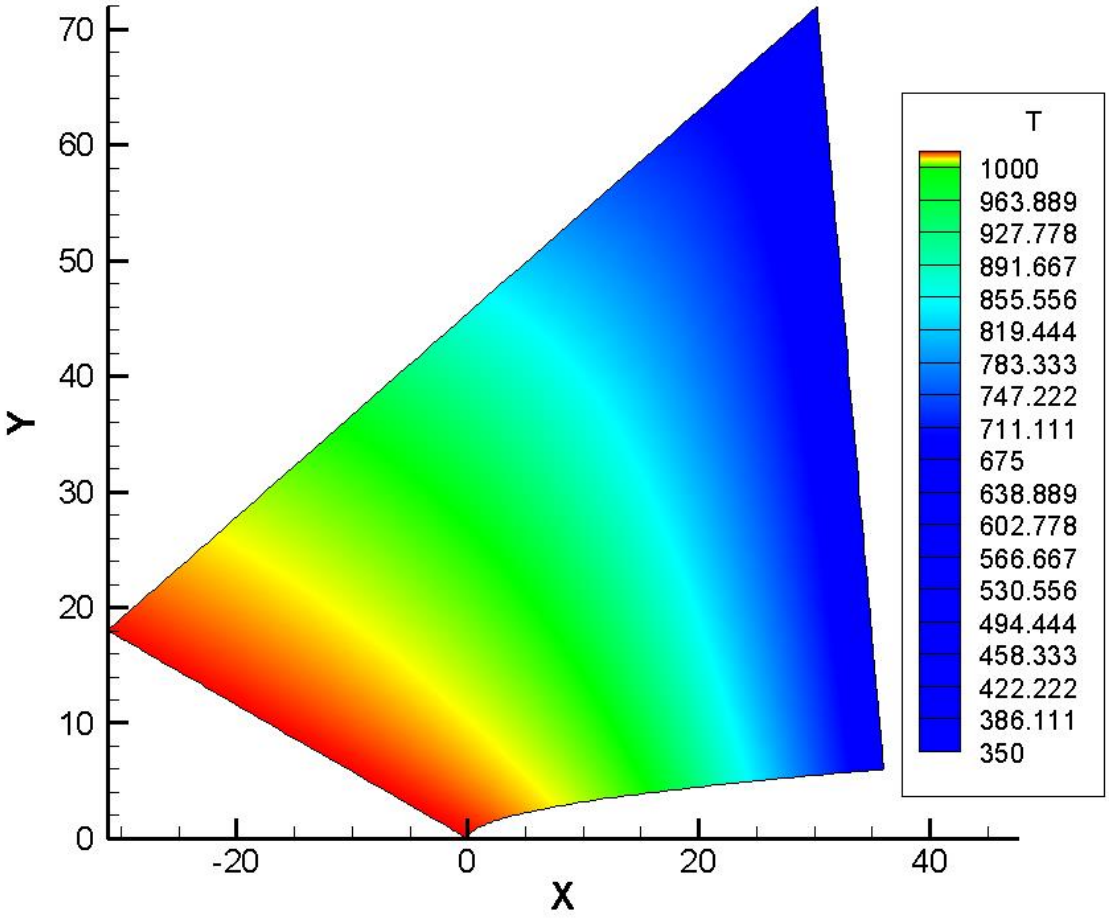

Fig. 3-10: Temperature contour using PGS

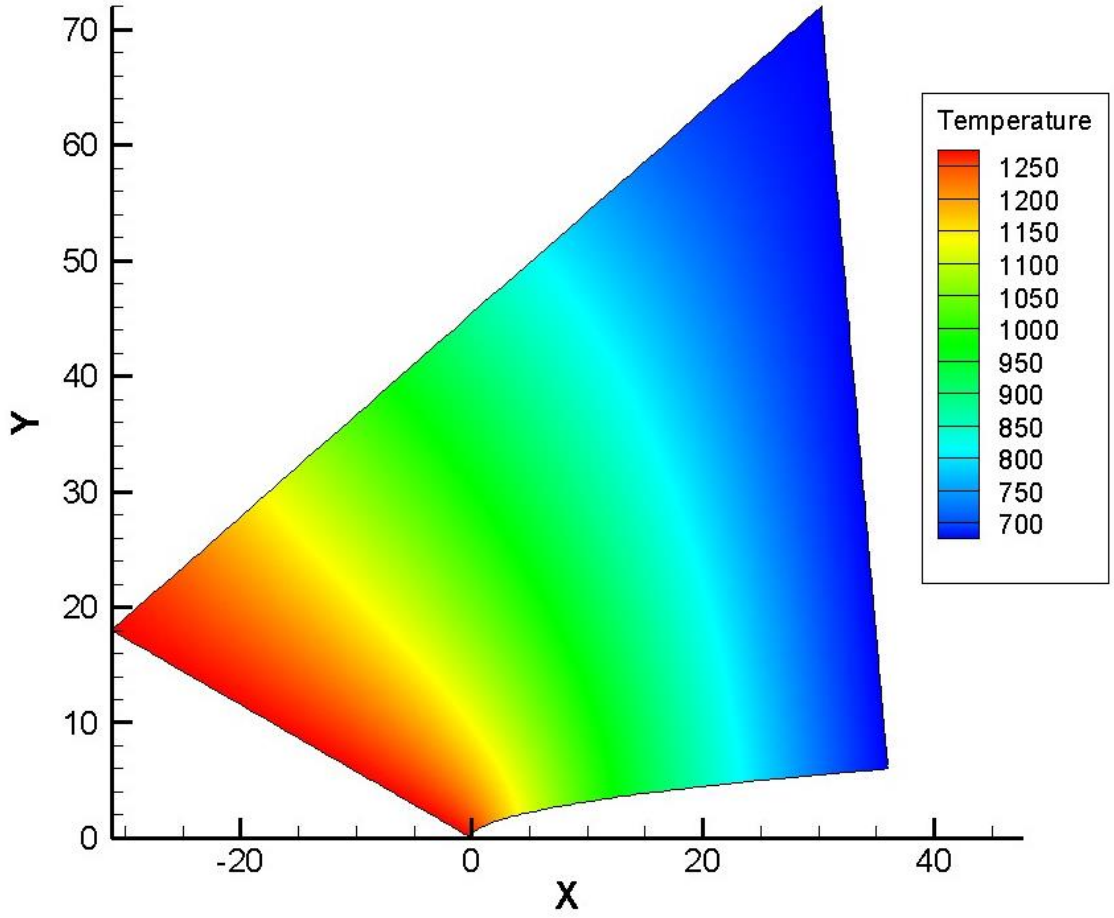

Fig. 3-11: Temperature contour using ANSYS Fluent

3.1 With the iso-thermal square

After finding the position of the square inside our grid, we will set its temperature to the left boundary temperature and solve other nodes. Figures 3-12 and 3-13 illustrate ours and ANSYS Fluent's solutions.

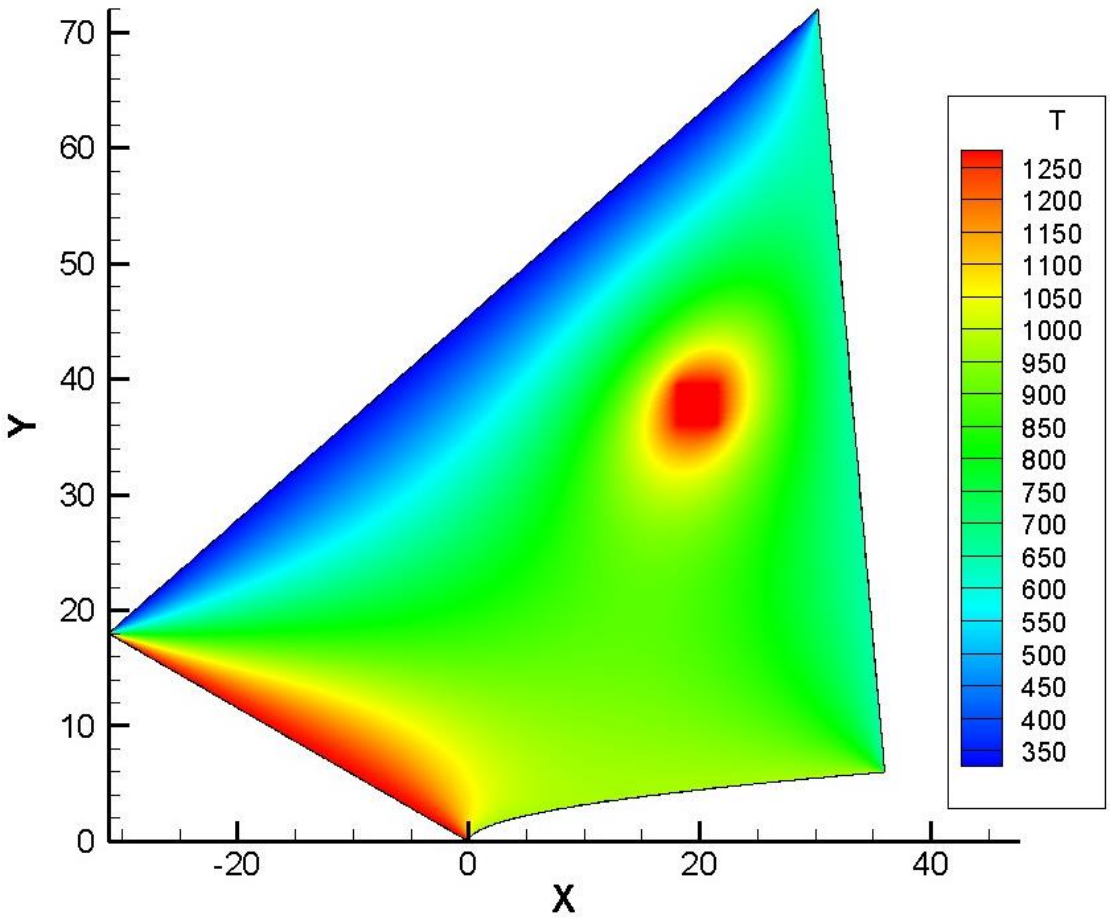

Fig. 3-12: Temperature contour using PGS

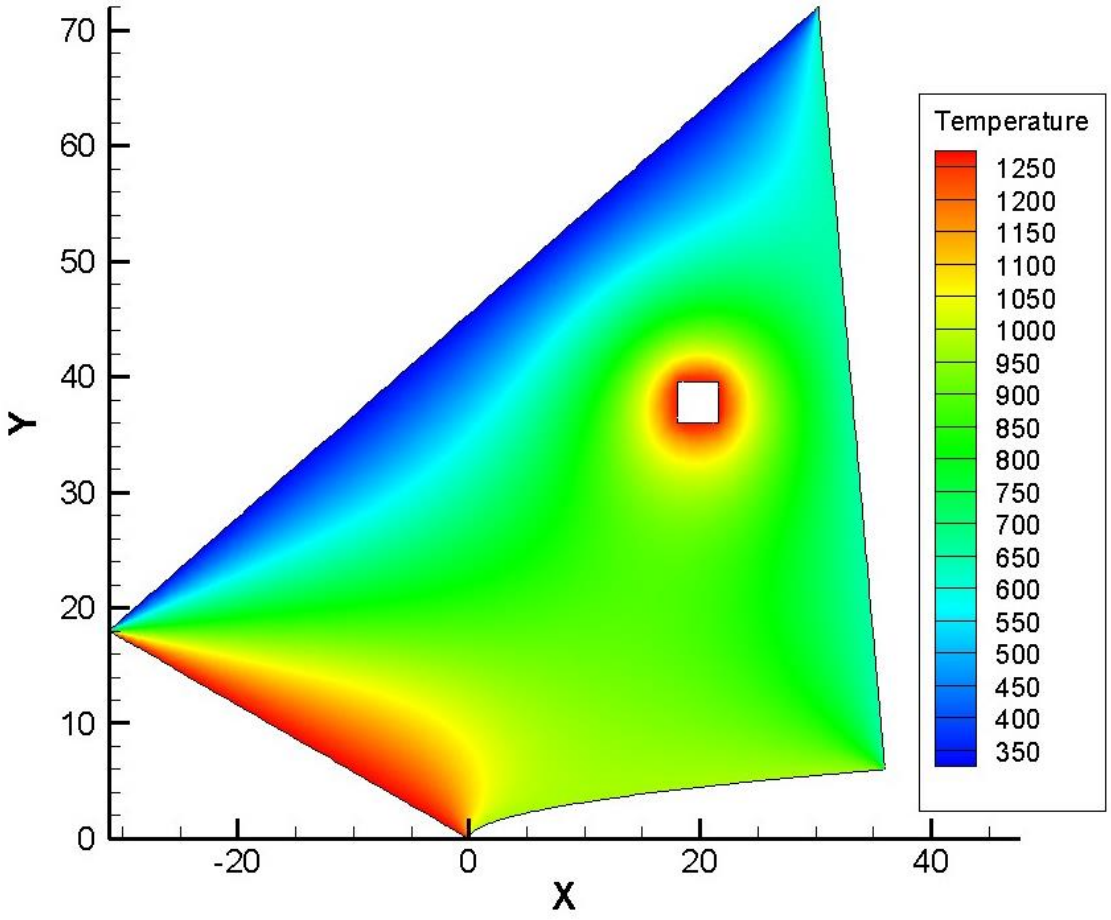

Fig. 3-13: Temperature contour using ANSYS Fluent