DALHOUSIE
UNIVERSITY

MECHANICAL
ENGINEERING

Simulating Steady State Temperature Distribution of a Flat Plate Using the Finite Volume Method

Report submitted to:
Dr. Mohammed Saeedi

Report prepared by:
Peter Oldreive (B00894035)

Mechanical Engineering, Dalhousie University
Halifax, Nova Scotia, Canada, November 2025

TABLE OF CONTENTS

LIST OF FIGURES iii

LIST OF ABBREVIATIONS AND SYMBOLS USED iv

1 INTRODUCTION 1

2 THEORY 2

3 NUMERICAL MODEL AND METHOD 3

3.1 SIMULATION OVERVIEW AND BOUNDARY CONDITIONS 3

3.2 APPROXIMATING THE HEAT DIFFUSION EQUATION 4

3.3 DERIVING SIMULATION-SPECIFIC EQUATIONS 5

3.3.1 Interior Nodes 5

3.3.2 Southwest Corner 5

3.3.3 Southeast Corner 5

3.3.4 Northwest Corner 5

3.3.5 Northeast Corner 6

3.3.6 South Boundary, $x < 400$ mm (Neumann Condition) 6

3.3.7 South Boundary, $x \geq 400$ mm (Newton/Robin Boundary) 6

3.3.8 East Boundary 6

3.3.9 North Boundary 6

3.3.10 West Boundary 7

3.4 NUMERICAL MODEL PROGRAMMING 7

3.4.1 Iterative Solver 7

3.4.2 Overall Numerical Model 8

4 RESULTS and Discussion 10

4.1 MESH CONVERGENCE STUDY AND COMPUTE TIMES 10

4.2 COMPARISON OF SOR VALUES 15

4.3 STEADY STATE TEMPERATURE DISTRIBUTION 16

4.4 HEAT TRANSFER RATES AT BOUNDARIES 19

5 CONCLUSIONS 21

BIBLIOGRAPHY 22

6 APPENDIX A 23

6.1 MATLAB NUMERICAL MODEL 23

6.2 GAUSE-SEIDEL SOLVER 31

LIST OF FIGURES

Figure 3.1 Experimental Plate with Boundary Conditions........3
Figure 3.2 Example Discretized Grid with Δ = 80 mm........4
Figure 4.1 Discretized Grid with Δx =Δy = 10 mm........10
Figure 4.2 Discretized Grid with Δx =Δy = 5 mm........10
Figure 4.3 Discretized Grid with Δx =2Δy = 5 mm........11
Figure 4.4 Colourmap with Contour Lines for Δx =Δy = 10 mm........12
Figure 4.5 Colourmap with Contour Lines for Δx =Δy = 5 mm........12
Figure 4.6 Colourmap with Contour Lines for Δx =2Δy = 5 mm........13
Figure 4.7 Temperature Along a Diagonal Cutline from (0,0) to (*Nx*, *Ny*) for Δx =Δy = 10 mm........13
Figure 4.8 Temperature Along a Diagonal Cutline from (0,0) to (*Nx*, *Ny*) for Δx =Δy = 5 mm........14
Figure 4.9 Temperature Along a Diagonal Cutline from (0,0) to (*Nx*, *Ny*) for Δx =2Δy = 5 mm........14
Figure 4.10 Temperature Along Horizontal Cutlines at Various Heights WRT *x*-Coordinate........17
Figure 4.11 Temperature Along Vertical Cutlines at Various Positions WRT *y*-Coordinate........18
Figure 4.12 Heat Transfer Rate along Fluid Flow Boundaries WRT *x*-Coordinate........19

LIST OF TABLES

Table 4.2 Maximum and Minimum Cell Temperatures for Each Resolution........11
Table 4.1 Table of Omega Values and Required Iterations........15
Table 4.2 Table of Omega Values and Required Iterations Near Optimal Value........16
Table 4.4 Heat Transfer Rates at Fluid Flow Boundaries........20

LIST OF ABBREVIATIONS AND SYMBOLS USED

Dimensional Variables

k	Thermal conductivity (W/mK)
$\dot{q}$	Rate of energy generation (W/m^3)
q''	Heat transfer rate (W/m^2)
T	Temperature (°C or K)
h	Convection Coefficient (W/m^2)
$T_{i,j}$, T_P	Temperature at point with coordinate (i, j)
$T_{i+1,j}$, T_E	Temperature at one point east of point (i, j)
$T_{i-1,j}$, T_W	Temperature at one point west of point (i, j)
$T_{i,j+1}$, T_N	Temperature at one point north of point (i, j)
$T_{i,j-1}$, T_S	Temperature at one point south of point (i, j)

Greek Letters

ω	Relaxation Parameter

Subscripts

boundary	Property of the stated boundary condition
x	Property related to x-dimension
y	Property related to y-dimension
tot	Total quantity

Abbreviations

SOR	Successive Over Relaxation
WRT	With Respect To

Definition of non-dimensional variables

N	Number of discrete points

1 INTRODUCTION

Steady-state simulations are a key area of numerical modeling, offering a fast way to predict the behavior of fluid, thermal, and other systems once they have passed their transient response period. To better understand how complex simulation software performs two-dimensional steady state numerical modeling, the steady state temperature distribution in a flat plate subject to a series of boundary conditions was simulated from first principles. To perform this simulation, the governing equation was simplified to meet experimental conditions before being modified for application of the finite volume method. The divergence theorem was applied to the finite volume equation, with second order accurate Taylor Series expansions used to approximate the flux terms. This equation was discretized to a grid of finite volume cells, with care taken to derive ten different forms to appropriately represent each boundary condition. A MATLAB script was developed to compute the steady state temperature distribution of the plate for various cell sizes and generate a series of one and two-dimensional plots to analyze the temperature distribution within the plate. To solve the steady state temperature distribution within the plate, the Gause-Seidel method was utilized, with successive over relaxation SOR to speed up solution convergence. The solver algorithm was programmed for the situation from first principals, offering valuable insight into developing a numerical solver for use in rapid simulations. Different relaxation coefficients were compared, to understand how it effects convergence as well as iterate to a most effective value. Heat transfer rates were compared at fluid interfaces to verify that the plate was in fact simulated in steady state.

2 THEORY

The core problem to be solved in this simulation is determining the temperature field in a medium resulting from a series of boundary conditions. Temperature distribution within a medium is governed by the *heat diffusion equation,* shown in (1) for Cartesian coordinates [1].

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right)+\dot{q}=\rho c_p\frac{\partial T}{\partial t} \tag{1}$$

Equation 1 represents the *heat diffusion* equation in a general form, including terms for transient heat storage and energy generation. Since the specific case to be simulated does not generate heat and is to be calculated at steady state, the equation can be simplified to (2).

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right)=0 \tag{2}$$

Going further, the simulation is to be performed in a two-dimensional domain, and the thermal conductivity of the material remains constant; further simplification to (3) can be made.

$$k\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right)=0 \tag{3}$$

Finding a direct solution to the 2nd order, elliptic, partial differential, heat diffusion equation is not a trivial task. To rapidly simulate the steady-state temperature distribution in the plate, the solution will be approximated using a series of equations derived using the finite difference method, solved using the Gauss-Seidel method.

Thermal boundary conditions can take many forms. For this report, two simple boundary conditions will be considered. The Neumann boundary condition is applied to any boundary that is to be assumed perfectly insulated. From this perfect insulation assumption, the temperature gradient along the boundary is assumed not to change with position, as seen in (4). The second boundary condition modeled in this analysis is the Newton or Robin condition. When this condition is applied to a boundary experiencing fluid flow, the convective heat flux of the fluid flow on one side of the boundary is equal to the conductive heat flux to the other side of the boundary (5). Convective heat transfer between the boundary and fluid is governed by Newton's law of cooling (6), proportional to the convection coefficient, *h,* and the temperature gradient between the boundary and the fluid. Conductive head flux is related to the thermal conductivity of the material and the temperature gradient within it (7). These assumed behaviors for the Neumann and Newton/Robin boundary conditions allow the numerical model of the steady state heat condition within the plate to be simplified, while still providing a good result for the temperature distribution.

$$\frac{\partial T}{\partial x}=0, \qquad \frac{\partial T}{\partial y}=0 \tag{4}$$

$$q''_{cond}=q''_{conv} \tag{5}$$

$$q''_{conv}=h\left(T_{boundary}-T_{\infty}\right) \tag{6}$$

$$q''_{cond}=k\frac{dT}{dx}\ or\ \frac{dT}{dy} \tag{7}$$

3 NUMERICAL MODEL AND METHOD

3.1 SIMULATION OVERVIEW AND BOUNDARY CONDITIONS

The steel plate to be modeled is shown in Fig. 3.1, with the boundary conditions clearly indicated. The experimental plate is 800 mm wide and 400 mm tall and has a unique boundary condition along each edge. Insulation is applied to the westmost boundary (x = 0 mm); this is assumed to provide perfect thermal isolation, resulting in a Neumann boundary condition. The northmost boundary of the plate (y = 400 mm) is under hot gas flow with a convection coefficient and fluid temperature given, this boundary will be assumed to follow the Newton/Robin boundary condition. The east boundary (x = 800 mm) is also given as insulated, assumed to follow the Neumann condition. The bottom boundary shares two different conditions, with all cells west of x = 400 mm falling under the Neumann condition, and all cells east of x = 400 mm experiencing a cold gas flow following the Newton/Robin condition.

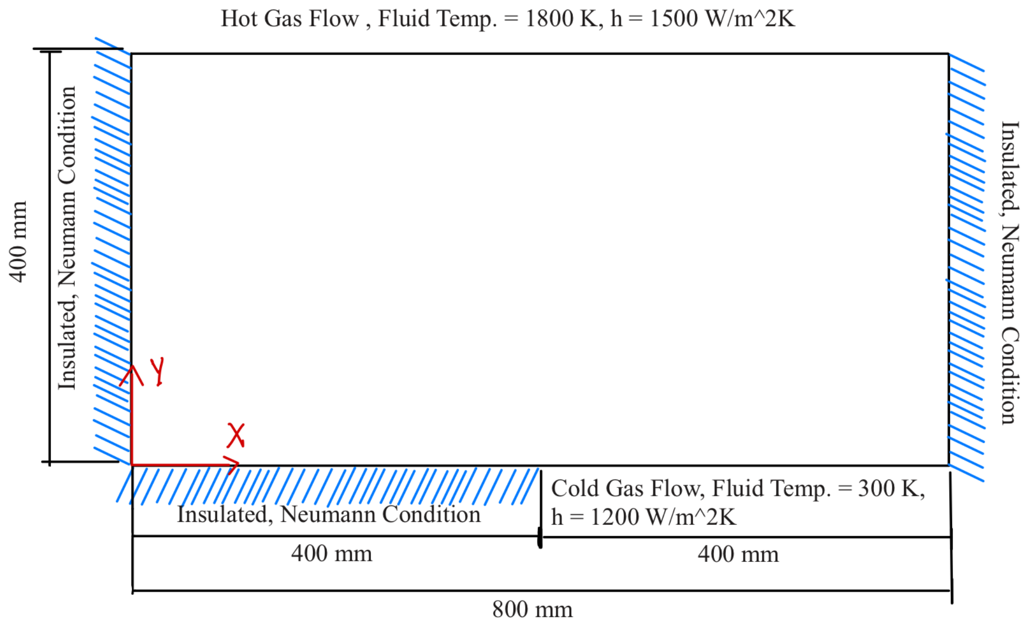
Hot Gas Flow , Fluid Temp. = 1800 K, h = 1500 W/m^2K
400 mm
Insulated, Neumann Condition
Y
X
Insulated, Neumann Condition
Insulated, Neumann Condition
Cold Gas Flow, Fluid Temp. = 300 K,
h = 1200 W/m^2K
400 mm
400 mm
800 mm

Figure 3.1 Experimental Plate with Boundary Conditions

The goal of the numerical simulations is to determine the steady-state temperature distribution within the plate using the simplified *heat diffusion equation* (3). To avoid having to solve the equation directly, a numerical model was developed for the problem using the finite volume method. To apply the finite volume method to the governing equation, it was first re-written to the form of (8) for further analysis. The volume integral was applied to the equation in this form (9), which then allowed the application of the divergence theorem (10). Simplifying the result of applying the divergence theorem, the governing equation for the heat diffusion equation can be written in the form show in (11).

$$k\left(\frac{\partial}{\partial x}\hat{\imath}+\frac{\partial}{\partial y}\hat{\jmath}\right)\cdot\left(\frac{\partial T}{\partial x}\hat{\imath}+\frac{\partial T}{\partial y}\hat{\jmath}\right) = 0 \tag{8}$$

$$\oiiint k\left(\frac{\partial}{\partial x}\hat{\imath}+\frac{\partial}{\partial y}\hat{\jmath}\right)\cdot\left(\frac{\partial T}{\partial x}\hat{\imath}+\frac{\partial T}{\partial y}\hat{\jmath}\right) dv = 0 \tag{9}$$

$$\oiint k\left(\frac{\partial}{\partial x}\hat{\imath}+\frac{\partial}{\partial y}\hat{\jmath}\right)\cdot\left(ds_x\hat{\imath}+ds_y\hat{\jmath}\right) = 0 \tag{10}$$

$$\left(\frac{\partial T}{\partial x}\right)_e (\Delta y) + \left(\frac{\partial T}{\partial y}\right)_n (\Delta x) + \left(\frac{\partial T}{\partial x}\right)_w (-\Delta y) + \left(\frac{\partial T}{\partial y}\right)_s (-\Delta x) = 0 \tag{11}$$

This derivation has reduced the second order heat diffusion equation to a first order form using the finite volume method. The differential terms for the east, north, west, and south represent the heat flux in each direction from a single point or cell. The combination of these heat flux terms in the net heat flux vector in Cartesian coordinates for this two-dimensional problem. While the equation is simplified in this state, further steps need to be taken as to not produce a direct solution to the first order differential terms in (11). To apply the finite volume method, the plate will be broken up into a discrete grid of N_x by N_y cells, that have the same finite volume. For this simulation, the difference between the points in the x and y direction will be variable meaning $\Delta x = \Delta y$ is not true for all cases. This will result in an equally spaced, discrete grid inside the plate boundary with $N_x * N_y = N_{tot}$ points to calculate the temperature at. A course (high Δ) grid is shown in Fig. 3.2 to visualize the finite cells that are generated from this technique, with the blue points signifying the center of each cell. The boundary cells are particularly important in this problem, with the edge of the cell on the boundary wall driving the problem as there is no heat generation within the plate.

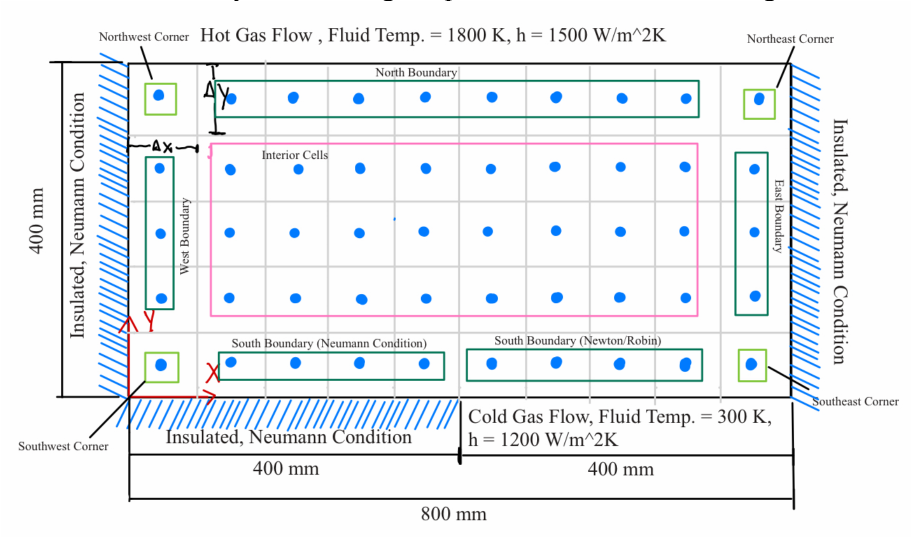
Northwest Corner
Hot Gas Flow , Fluid Temp. = 1800 K, h = 1500 W/m^2K
Northeast Corner
North Boundary
Δy
Δx
Interior Cells
400 mm
Insulated, Neumann Condition
West Boundary
East Boundary
Insulated, Neumann Condition
Y
X
South Boundary (Neumann Condition)
South Boundary (Newton/Robin)
Southeast Corner
Southwest Corner
Insulated, Neumann Condition
Cold Gas Flow, Fluid Temp. = 300 K, h = 1200 W/m^2K
400 mm
400 mm
800 mm

Figure 3.2 Example Discretized Grid with Δ = 80 mm

3.2 APPROXIMATING THE HEAT DIFFUSION EQUATION

Since the *heat diffusion equation for* the finite volume method will not be solved for this simulation, the heat flux terms of each cell will be approximated using a second order accurate Taylor series expansion of each term. In efforts of time saving in this report, the derivation of these flux terms using Taylor Series expansions will not be shown. The simplified flux terms are shown in (12) to (15), with T_P representing the temperature of the analyzed cell. The simplification of these flux terms allows the calculation of the temperature distribution in the plate using the finite difference method, without having to solve the ODE directly. Using the simplified flux terms, specific equations can be derived for each cell type shown in Fig. 3.2 based on the given boundary conditions.

$$\left(\frac{\partial T}{\partial x}\right)_e = \frac{T_{i+1,j} - T_P}{\Delta x} + \sigma(\Delta x)^2 \tag{12}$$

$$\left(\frac{\partial T}{\partial y}\right)_n = \frac{T_{i,j+1} - T_P}{\Delta y} + \sigma(\Delta y)^2 \tag{13}$$

$$\left(\frac{\partial T}{\partial x}\right)_w = \frac{T_P - T_{i-1,j}}{\Delta x} + \sigma(\Delta x)^2 \tag{14}$$

$$\left(\frac{\partial T}{\partial y}\right)_s = \frac{T_P - T_{i,j-1}}{\Delta y} + \sigma(\Delta y)^2 \tag{12}$$

3.3 Deriving Simulation-Specific Equations

Each group of cells boxed in Fig. 3.2 is subject to a unique boundary, necessitating a situation-specific equation to be derived for the case. The boundary conditions of the simulation need to be worked into the simulation for each case.

3.3.1 Interior Nodes

The interior nodes are not subject to any boundary conditions, meaning the equation for calculating the temperature of these cells will be (11), with each flux term replaced by the approximation. Re-arranging this equation yields (16), for the interior nodes.

$$2\left(\frac{\Delta x}{\Delta y}+\frac{\Delta y}{\Delta x}\right)T_p - \left(\frac{\Delta y}{\Delta x}\right)T_E - \left(\frac{\Delta x}{\Delta y}\right)T_N - \left(\frac{\Delta y}{\Delta x}\right)T_W - \left(\frac{\Delta x}{\Delta y}\right)T_S = 0 \tag{16}$$

3.3.2 Southwest Corner

For the southwest corner, the cell to the west (*i*-1, *j*) is along the western Neumann Boundary. Similarly, the southern cell (*i, j*-1) is along the southern Neumann Boundary. Since the temperature gradient along the Neumann Boundary is zero, the temperature of these cells can be approximated as $T_{i,j}$. This assumption was validated using a fictitious node derivation but is not included in this report for space savings. The cells to the north and east of this corner are within the domain; therefore, they will be taken as the point itself. Modifying (11) for these conditions yields (17) for the southwest corner temperature.

$$\left(\frac{\Delta x}{\Delta y}+\frac{\Delta y}{\Delta x}\right)T_p - \left(\frac{\Delta y}{\Delta x}\right)T_E - \left(\frac{\Delta x}{\Delta y}\right)T_N = 0 \tag{17}$$

3.3.3 Southeast Corner

For the southeast corner, the point to the east (*i*+1, *j*) is along the eastern Neumann boundary, and the south point (*i, j*-1) is subject to cold gas flow. The northern and western points are within the domain; therefore, they are taken as the point itself. The simplification of the heat flux equality for the cold gas flow by combining (6) ad (7) will not be shown in this report for space savings. The fictitious cell method was used to solve this equality, with the coefficients shown separate from the equation. Modifying (11) for these conditions yields (18) for the southeast corner temperature.

$$\left(\left(2+\frac{a_{NFS}}{a_{PFS}}\right)\frac{\Delta x}{\Delta y}+\frac{\Delta y}{\Delta x}\right)T_p - \left(\frac{\Delta y}{\Delta x}\right)T_W - \left(\frac{\Delta x}{\Delta y}\right)T_N = \left(\frac{\Delta x}{\Delta y}\right)\left(\frac{b_{FS}}{a_{PFS}}\right) \tag{18}$$

Where: $a_{PFS} = 1 + \frac{\Delta y h_s}{2k}$, $a_{NFS} = \frac{\Delta y h_s}{2k} - 1$, $b_{FS} = \frac{h_s \Delta y}{k} T_{\infty S}$

3.3.4 Northwest Corner

For the northwest corner, the cell to the west (*i*-1, *j*) is along the western Neumann Boundary, and the north cell (*i, j*+1) is subject to the hot gas flow condition. The southern and eastern points are within the domain; therefore, they are taken as the point itself. Modifying (11) for these conditions yields (14) for the northwest corner temperature.

$$\left(\left(2+\frac{a_{SFN}}{a_{PFN}}\right)\frac{\Delta x}{\Delta y}+\frac{\Delta y}{\Delta x}\right)T_p - \left(\frac{\Delta y}{\Delta x}\right)T_E - \left(\frac{\Delta x}{\Delta y}\right)T_S = \left(\frac{\Delta x}{\Delta y}\right)\left(\frac{b_{FN}}{a_{PFN}}\right) \tag{19}$$

Where: $a_{PFN} = 1 + \frac{\Delta y h_N}{2k}$, $a_{SFN} = \frac{\Delta y h_N}{2k} - 1$, $b_{FN} = \frac{h_N \Delta y}{k} T_{\infty N}$

3.3.5 Northeast Corner

For the northeast corner, the cell to the east (*i*+1, *j*) is along the eastern Neumann Boundary, and the north cell (*i, j*+1) is subject to hot gas flow. The southern and western points are within the domain; therefore, they are taken as the point itself. Modifying (11) for these conditions yields (20) for the northwest corner temperature.

$$\left(\left(2+\frac{a_{SFN}}{a_{PFN}}\right)\frac{\Delta x}{\Delta y}+\frac{\Delta y}{\Delta x}\right)T_p-\left(\frac{\Delta y}{\Delta x}\right)T_W-\left(\frac{\Delta x}{\Delta y}\right)T_S=\left(\frac{\Delta x}{\Delta y}\right)\left(\frac{b_{FN}}{a_{PFN}}\right) \tag{20}$$

Where: $a_{PFN}=1+\frac{\Delta y h_N}{2k}$, $a_{SFN}=\frac{\Delta y h_N}{2k}-1$, $b_{FN}=\frac{h_N \Delta y}{k}T_{\infty N}$

3.3.6 South Boundary, *x* < 400 mm (Neumann Condition)

For the portion of the south boundary that falls under the Neumann Condition, the south cell (*i, j*-1) will match the temperature of the cell (*i, j*). No other cells, north, east, or west are subject to a boundary condition, meaning the points will be unaltered. Modifying (11) for these conditions yields (21) for the south boundary temperature for *x* < 400 mm.

$$\left(\frac{\Delta x}{\Delta y}+2\frac{\Delta y}{\Delta x}\right)T_p-\left(\frac{\Delta y}{\Delta x}\right)T_E-\left(\frac{\Delta x}{\Delta y}\right)T_N-\left(\frac{\Delta y}{\Delta x}\right)T_W=0 \tag{21}$$

3.3.7 South Boundary, *x* ≥ 400 mm (Newton/Robin Boundary)

For the portion of the south boundary that falls under the Newton/Robin boundary, the south cell (*i, j*-1) will be subject to cold gas flow. No other cells, north, east, or west are subject to a boundary condition, meaning the points will be unaltered. Modifying (11) for these conditions yields (22) for the south boundary temperature for *x* ≥ 400 mm.

$$\left(\left(2+\frac{a_{NFS}}{a_{PFS}}\right)\frac{\Delta x}{\Delta y}+2\frac{\Delta y}{\Delta x}\right)T_p-\left(\frac{\Delta y}{\Delta x}\right)T_E-\left(\frac{\Delta x}{\Delta y}\right)T_N-\left(\frac{\Delta y}{\Delta x}\right)T_W=\left(\frac{\Delta x}{\Delta y}\right)\left(\frac{b_{FS}}{a_{PFS}}\right) \tag{22}$$

Where: $a_{PFS}=1+\frac{\Delta y h_s}{2k}$, $a_{NFS}=\frac{\Delta y h_s}{2k}-1$, $b_{FS}=\frac{h_s \Delta y}{k}T_{\infty s}$

3.3.8 East Boundary

For the east boundary, the east cell (*i*+1, *j*) will is subject to the east Neumann boundary. No other cells, north, south, or west are subject to a boundary condition, meaning the points will be unaltered. Modifying (11) for these conditions yields (23) for the east boundary temperature.

$$\left(2\frac{\Delta x}{\Delta y}+\frac{\Delta y}{\Delta x}\right)T_p-\left(\frac{\Delta x}{\Delta y}\right)T_N-\left(\frac{\Delta y}{\Delta x}\right)T_W-\left(\frac{\Delta x}{\Delta y}\right)T_S=0 \tag{23}$$

3.3.9 North Boundary

For the north boundary, the north cell (*i*, *j*+1) is subject to hot gas flow. No other cells, south, east, or west are subject to a boundary condition, meaning the points will be unaltered. Modifying (11) for these conditions yields (24) for the north boundary temperature.

$$\left(\left(2+\frac{a_{SFN}}{a_{PFN}}\right)\frac{\Delta x}{\Delta y}+2\frac{\Delta y}{\Delta x}\right)T_p-\left(\frac{\Delta y}{\Delta x}\right)T_E-\left(\frac{\Delta y}{\Delta x}\right)T_W-\left(\frac{\Delta x}{\Delta y}\right)T_S=\left(\frac{\Delta x}{\Delta y}\right)\left(\frac{b_{FN}}{a_{PFN}}\right) \tag{24}$$

Where: $a_{PFN}=1+\frac{\Delta y h_N}{2k}$, $a_{SFN}=\frac{\Delta y h_N}{2k}-1$, $b_{FN}=\frac{h_N \Delta y}{k}T_{\infty N}$

3.3.10 West Boundary

For the west boundary, the west cell (i - 1, j) is subject to the Neumann Condition, meaning the west cell will take the temperature of the point itself, $T_{i,\,j}$. No other cells, south, north, or east, are subject to a boundary condition, meaning the points will be unaltered. Modifying (11) for these conditions yields (25) for the west boundary temperature.

$$\left(2\frac{\Delta x}{\Delta y}+\frac{\Delta y}{\Delta x}\right)T_p-\left(\frac{\Delta y}{\Delta x}\right)T_E-\left(\frac{\Delta x}{\Delta y}\right)T_N-\left(\frac{\Delta x}{\Delta y}\right)T_S=0 \tag{25}$$

3.4 NUMERICAL MODEL PROGRAMMING

The numerical model was programmed in MATLAB to calculate the temperature at each cell defined in the N_xxN_y grid defined for applying the finite volume method. The model was set up to be modular, allowing for changes to be made to the plate dimensions, boundary conditions, and cell size without making significant modifications to the program. The MATLAB script takes the series of boundary specific equations derived in Section 3.3, and builds a matrix of coefficients, A, and a vector of knowns, B. These build a system of linear equations with N_{TOT} equations and N_{TOT} unknowns that must be solved to calculate the vector of temperatures, T, for each cell in the grid. The system that must be solved is in the form shown below.

$$[A][T]=[B]$$

3.4.1 Iterative Solver

Matrix A is of dimension N_{tot} by N_{tot}, to have a row and column for every point. The coefficient of the temperature at each point is found along the diagonal of the matrix. Each row represents the equation defining each point, with coefficients placed in the column number that corresponds to the appropriate north, south, east, and west points. Vector B represents a collection of known values for the boundary conditions, particularly the hot and cold gas flow boundaries. The vector of unknowns, T, represents the unknown temperature at each point to be solved for, hence why the points are found along the main diagonal. With the equations in this form, an iterative solver algorithm was developed using the Gause-Seidel method incorporating Successive Over Relaxation (SOR). This technique represents a good balance of compute efficiency, and ease of programming the algorithm [2]. The Gause-Seidel method is like the Jacobi iterative method, with the primary difference being that the iterative solving of the equations uses the most recent value for T_i rather than the one from the previous iteration. The element wise equation for the Gause-Seidel algorithm is shown in (26), this equation calculates a single unknown for a single iteration [2]. This equation is applied to each unknown for each iteration until convergence is reached, with the convergence threshold defined by the user of the algorithm. In (26) T_i represents the unknown calculated, T_j represents all other unknowns in the solution vector, b_i is the known value for point i, a_{ii} is the coefficient of T_i, and a_j represents all other coefficients. The designation (k+1) represents the new iteration and (k) the previous iteration of (26). To use the Gause-Seidel method to solve the system of equations representing the flat plate, an initial temperature guess must be made to start the first iteration.

$$T_i^{k+1}=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_jT_j^{k+1}-\sum_{j=i+1}^{N_{TOT}}a_jT_j^{k}\right) \tag{26}$$

While the Gause-Seidel method is attractive to programmers due to its algorithmic simplicity, the compute times can become excessive if a large system is to be solved using a very small convergence tolerance [2]. To help reduce the compute time inherent with this algorithm, implementing successive over relaxation (SOR) can help to significantly reduce the total number of iterations required to teach convergence [2]. The use of the SOR parameter, ω, to adjust each iteration of the Gause-Seidel algorithm is shown in (27).

$$T_i^{k+1}=\frac{\omega}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_jT_j^{k+1}-\sum_{j=i+1}^{N_{TOT}}a_jT_j^{k}\right)+(1-\omega)T_i^k \tag{27}$$

After the value of temperature has been computed for every equation in each iteration, a convergence check is performed, this check determines if another iteration is to be performed or if convergence has been reached. The convergence check used in this script is quite simple, with the difference between every temperature in the $(k+1)$ iteration and the (k) iteration taken. If the absolute difference between any one of these temperatures is greater than or equal the convergence tolerance, another iteration is performed and checked again. This simple calculation is shown in (28) below.

$$|T_i^{k+1} - T_i^k| \leq tolarance \tag{28}$$

To facilitate greater program modularity, the Gause-Seidel iterative solver was implemented in a separate MATLAB script than the numerical model for this problem. The main numerical model calls this script after the matrix of coefficients and vector of knows has been defined. The script containing the Gause-Seidel algorithm is modular, having the option to read the inputs from a .csv file or directly from the workspace, this gives the option to run the solver independently of the main numerical model. The complete algorithm for the solver is described below. The MATLAB script is shown in Appendix A.

- Declare convergence parameters
  - Convergence Tolerance
  - SOR (ω)
  - Maximum allowable iterations
  - Initial temperature guess
- Take inputs from workspace or .csv files
- Check that the coefficient matrix and vector of knowns are compatible
- Initialize vector of guessed values as the current iteration
- Run for loop for iter = 1: maximum allowable iterations
  - Update previous iteration solution with current solution
  - For loop for i = 1:*Ny*
    - For loop for j = *1:Nx*
    - Use if statements to pull coefficients of the north, east, south, and west temperatures directly rather than looping through each column for every row
    - Apply coefficients to element-wise equation (26)
    - Update solution using SOR method (27)
    - Add 1 to the current index
  - Check for solution convergence
    - If convergence reached, break loop and print number of iterations taken
- Write output vector to .csv file (optional)
- End

3.4.2 Overall Numerical Model

With the solver delegated to its own script, the primary MATLAB script must be responsible for:

- Developing the system of equations to be solved
- Call the solver script
- Post process solution vector
- Generate descriptive plots of the result

To complete these tasks, the numerical model uses the following algorithm, with the entire program shown in Appendix A:

- Declare variables for the x and y dimensions of the plate
- Declare the transition point of the south boundary
- Declare parameters for the Newton/Robin boundary conditions
- Declare thermal conductivity
- Ask for user input for the grid spacing
  - Ensure grid spacing is valid

- Define N_x, N_y, and N_{tot} based on the dimensions of the plate and the defined grid spacing
- Print Δx, Δy, N_x, N_y, and N_{tot} to the command window to confirm to the user the simulation parameters they defined
- Define matrices and vectors of zeros for solving the simultaneous system:
    - N_{tot} by N_{tot} matrix of coefficients (A)
    - N_{tot} by 1 vector of unknowns (T)
    - N_{tot} by 1 vector of knowns (B)
- Use a for loop of N_{tot} iterations to fill matrix A with coefficients and vector B with knowns
    - Within the loop, use a series of conditionals to apply equations (16) through (25) to the correct node and fill the matrix and vector accordingly
- Call Gause-Seidel solver script

- Take in results of solver as vector T
- Process T into an N_y by N_x matrix for 2D plotting
- Find heat transfer rates at boundaries
- Process T into several vectors for 1D plots
- Generate 1D and 2D plots

4 RESULTS AND DISCUSSION

4.1 MESH CONVERGENCE STUDY AND COMPUTE TIMES

The MATLAB numerical model was run at a series of cell sized to determine the effect of reducing cell size on the output temperature field and compute time. Three spacings were chosen with, $\Delta x = \Delta y = 10$ mm, $\Delta x = \Delta y = 5$ mm, and $\Delta x = 2\Delta y = 5$ mm. Running the simulations for any cell volume smaller than the last case resulted in memory limitations within MATLAB. The numerical model was computed for each mesh spacing, with two-dimensional colour maps plotted as well as the temperature distribution across the diagonal cutline of the plate from (0,0) to (800, 400). Before generating these plots, the points in the mesh grid were plotted to visualize each finite cell in the grid. Fig. 4.1 shows the grid for $\Delta x = \Delta y = 10$ mm, Fig. 4.2 shows the grid for, $\Delta x = \Delta y = 5$ mm, and Fig. 4.3 shows $\Delta x = 2\Delta y = 5$ mm.

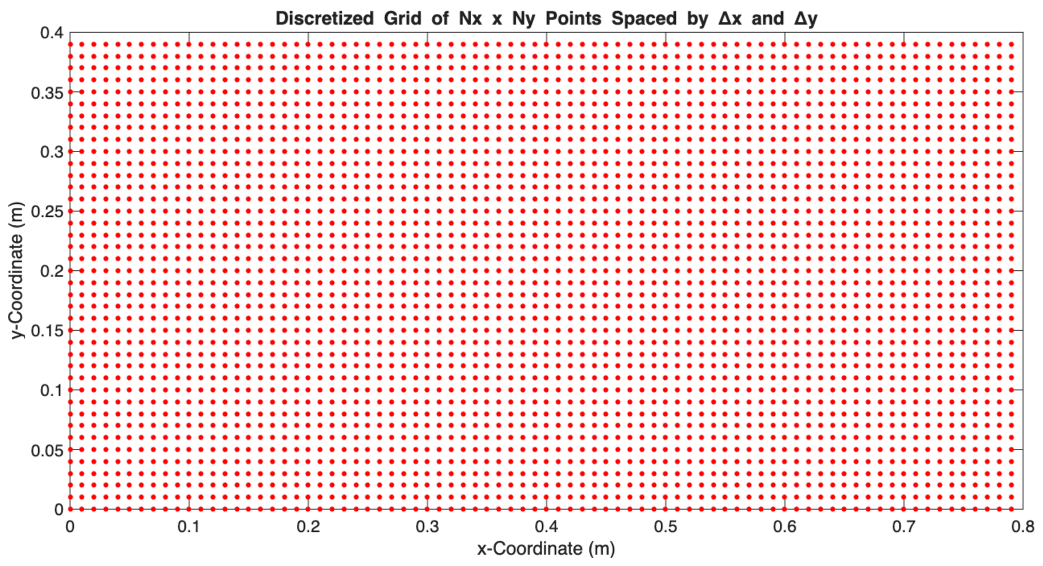

Figure 4.1 Discretized Grid with Δx =Δy = 10 mm

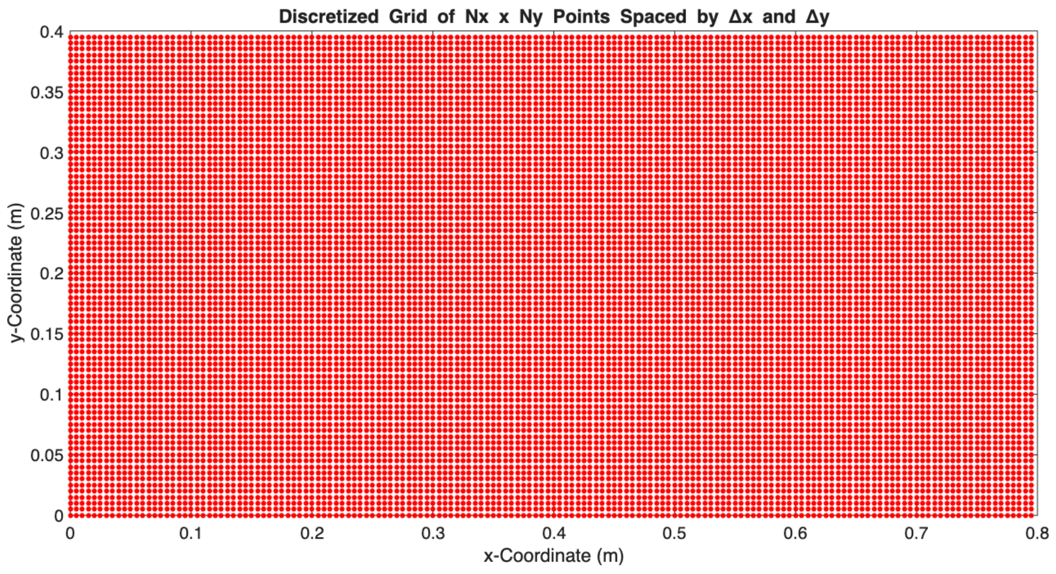

Figure 4.2 Discretized Grid with Δx =Δy = 5 mm

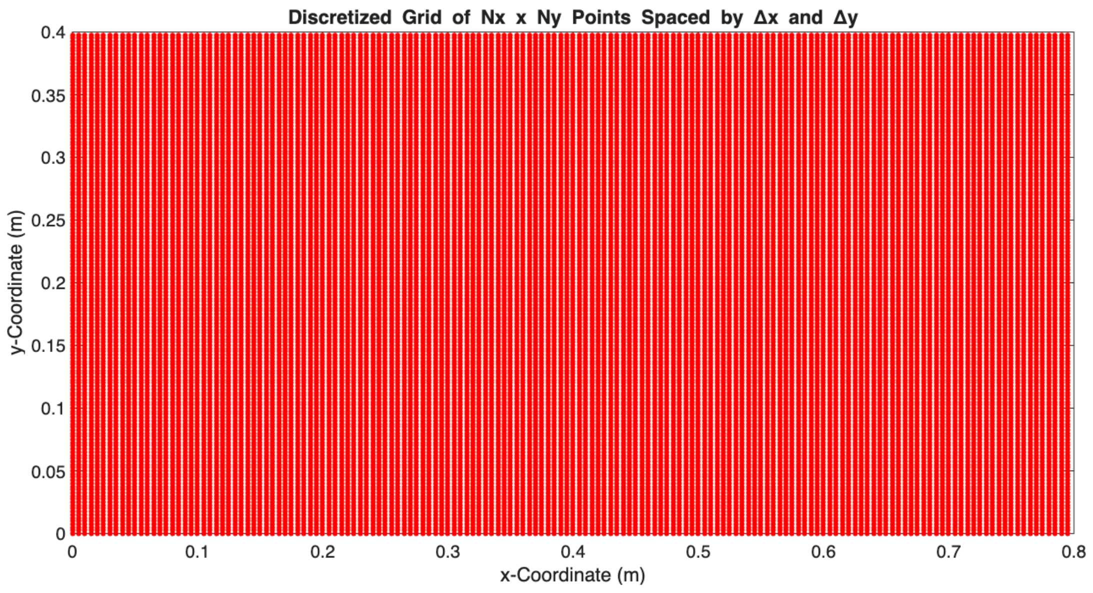

Figure 4.3 Discretized Grid with Δx =2Δy = 5 mm

As seen in Fig. 4.1-4.3, the number of points becomes exponentially higher with each decrease in cell size. For the $\Delta x = \Delta y = 10$ mm case, $N_{tot} = 3200$, the $\Delta x = \Delta y = 5$mm case, $N_{tot} = 12800$, and the $\Delta x = 2\Delta y = 5$ mm case $N_{tot} = 25600$. Since the number of points increases exponentially with the shrinking of the mesh size, going finer and finer will require exponentially more simulation time, meaning a balance needs to be made between simulation time and accuracy. For this simulation, running on a MacBook Pro M5, with 16 gb of ram, the $\Delta x = \Delta y = 10$ mm case required 0.07 s of compute time (279 iterations), the $\Delta x = \Delta y = 5$ mm case took 0.24 s (930 iterations), and the $\Delta x = 2\Delta y = 5$ mm case took 0.71 s to compute (2147 iterations). These simulations were run with a ω value of 1.95, and a solution tolerance of 1×10^{-3} K. The resolution of the simulation greatly impacts the compute time, with each change in the grid size requiring exponentially longer to compute than the previous size. This is in part due to the larger number of equations that need to be solved for each iteration as well as the increase in the number of iterations to reach solution convergence. In engineering practice, a crucial part of building a simulation model used in the design of a certain product is determining what is most important to simulate and balancing the resolution of the simulation with what is required for the project and the budget allocated to running these simulations.

To compare the response to each mesh size, the colour map and temperature along a diagonal cutline will be observed for all three cases. The colour maps for each mesh size are shown in Fig. 4.4-4.5, and the diagonal cutlines in Fig. 4.6-4.8. Comparing these plots will show how changing grid parameters affects the experimental results. To compare the effects of resolution on the maximum and minimum temperature in the plate, these temperatures are shown in Table 4.1 with the appropriate cell sizing.

Table 4.1 Maximum and Minimum Cell Temperatures for Each Resolution

Grid Resolution	Minimum Temperature (K)	Maximum Temperature (K)
$\Delta x = \Delta y = 10$ mm	525.4	1759.8
$\Delta x = \Delta y = 5$ mm	518.1	1755.4
$\Delta x = 2\Delta y = 5$ mm	514.1	1760.2

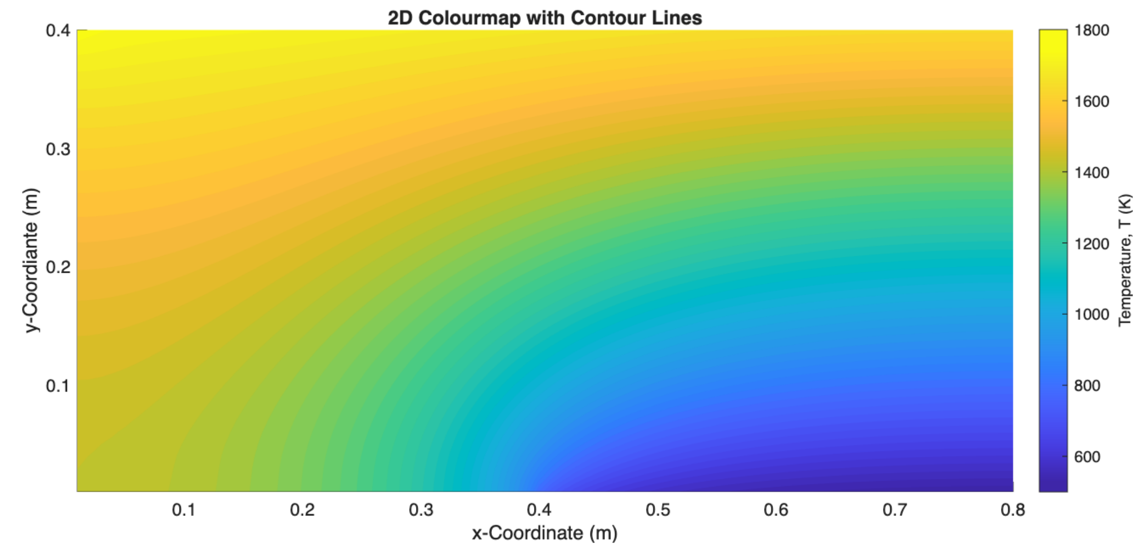

Figure 4.4 Colourmap with Contour Lines for Δx =Δy = 10 mm

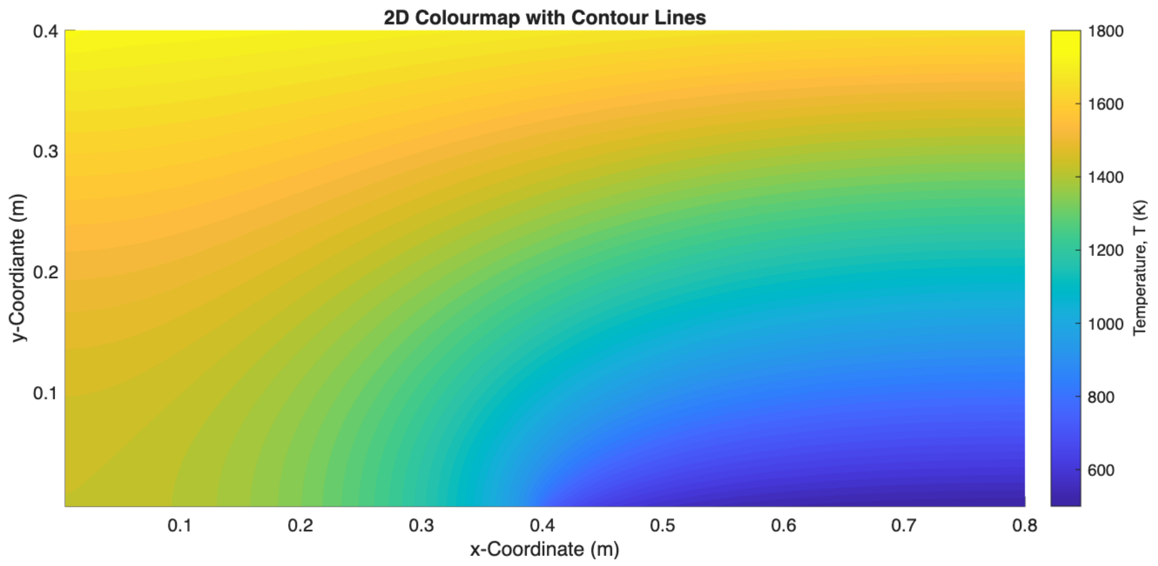

Figure 4.5 Colourmap with Contour Lines for Δx =Δy = 5 mm

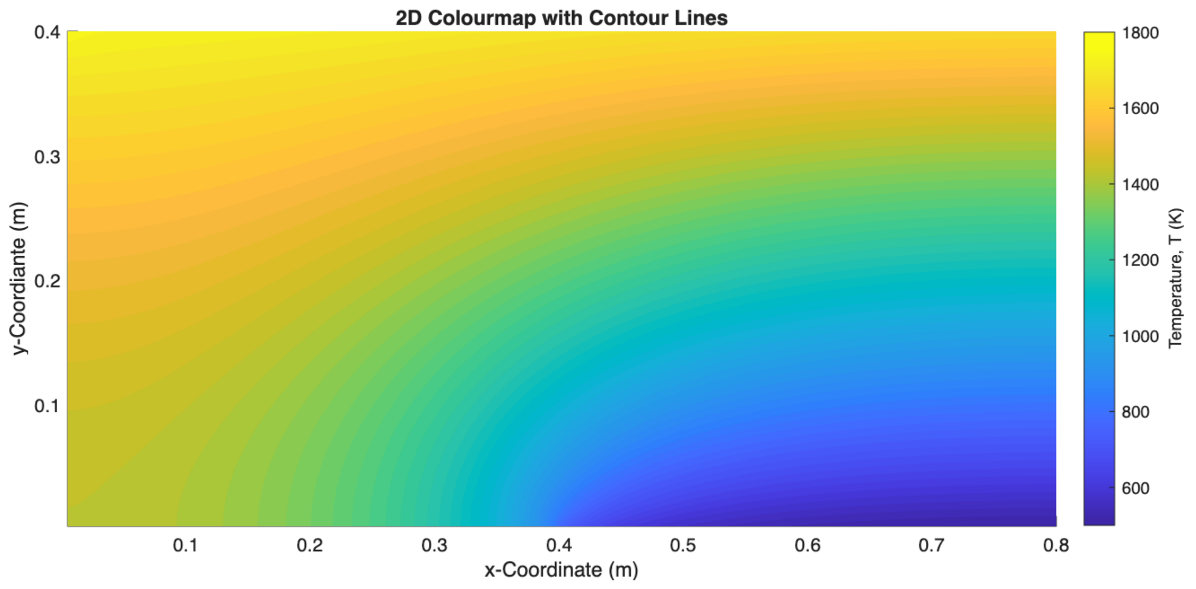

Figure 4.6 Colourmap with Contour Lines for Δx =2Δy = 5 mm

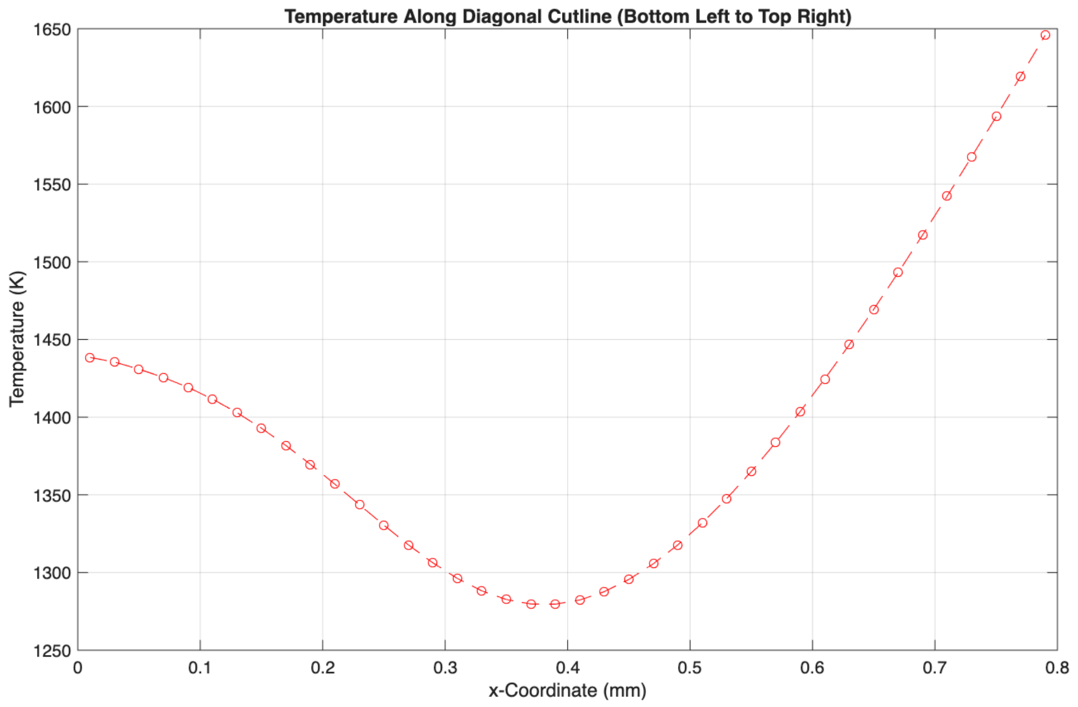

Figure 4.7 Temperature Along a Diagonal Cutline from (0,0) to (*Nx*, *Ny*) for Δx =Δy = 10 mm

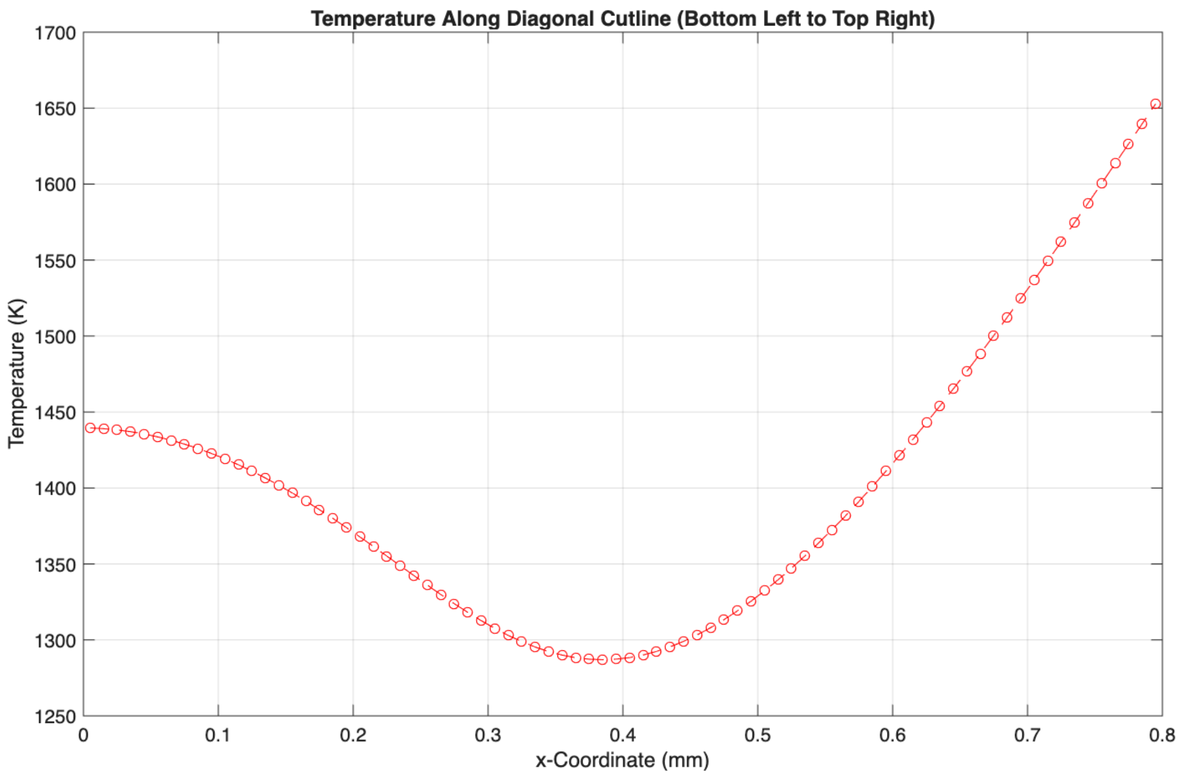

Figure 4.8 Temperature Along a Diagonal Cutline from (0,0) to (*Nx*, *Ny*) for Δx =Δy = 5 mm

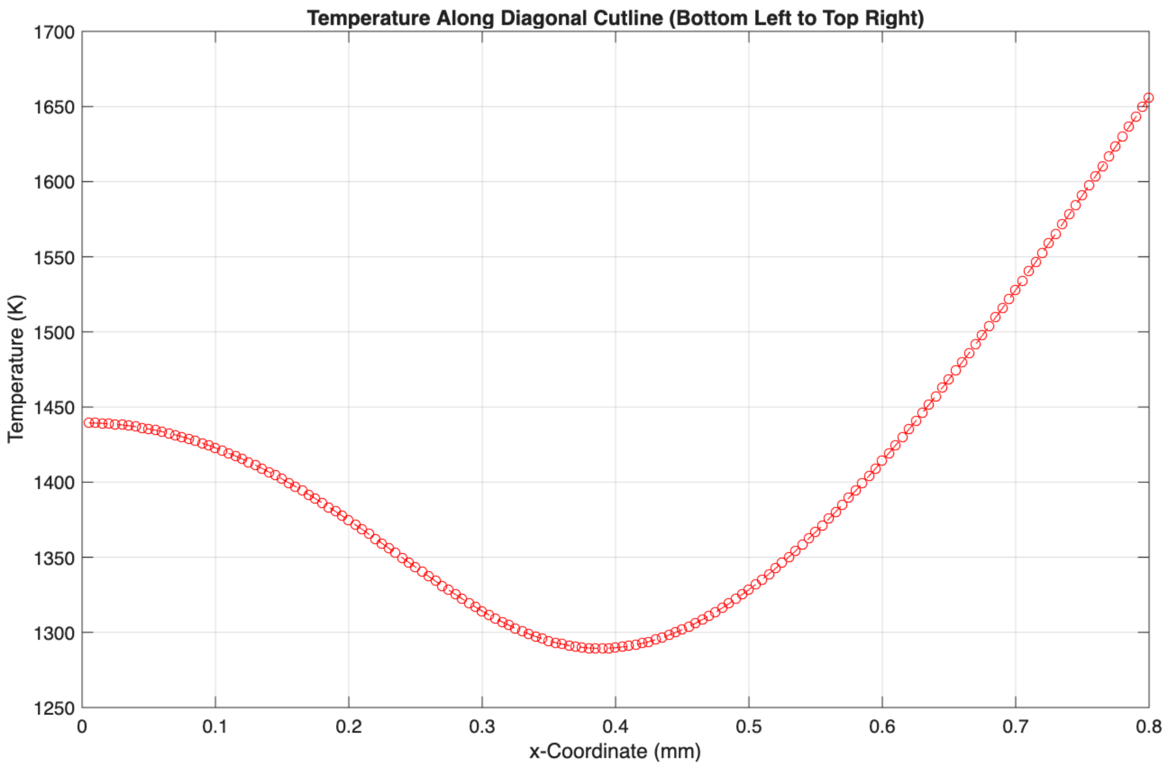

Figure 4.9 Temperature Along a Diagonal Cutline from (0,0) to (*Nx*, *Ny*) for Δx =2Δy = 5 mm

Comparing the three contour maps, the coarsest mesh of $\Delta x = \Delta y = 10$ mm is quite coarse, with the grid not appearing nearly as continuous as the two finer meshes. While the discrete cells in the colourmap are not obvious, the outline of each cell is still visible on close inspection, especially when compared to the two finer resolutions. The $\Delta x = \Delta y = 5$ mm case is much finer in resolution, with 4x the cells as the 10 mm case, while this did increase the simulation time substantially, the quality improvement in the colourmap is noticeable. The grid is much finer, making the cells nearly indistinguishable from one another on the plot, offering smooth contour lines from the hot to the cold side of the plate. The difference between the $\Delta x = \Delta y = 5$ mm plot, and the $\Delta x = 2\Delta y = 5$ mm plot is much less noticeable, with the drastically higher simulation time severely limiting the effectiveness of this resolution. If the highest resolution were to be applied to an iterative design process, the additional time required to perform the high-resolution simulation would not be worth the benefit of a slightly finer temperature gradient, if the problem required a longer simulation time than this simple 2D plate.

Looking at the diagonal cutline temperature plots, the coarseness of the $\Delta x = \Delta y = 10$ mm case is shown dramatically. The step size along the diagonal for this case is quite large, leading to a coarse plot that makes it difficult to observe a clear trend. The plot for the 5 mm case is much smoother, with the behavior of the plot converging to the upper wall temp becoming clearer. This becomes even more clear with the $\Delta x = 2\Delta y = 5$ mm case. The smoother and easier-to-interpret plots of the smaller grid sizes are indicative of simulation resolution being important for interpreting results. For the remainder of this report, the smallest grid size of $\Delta x = 2\Delta y = 5$ mm, since the compute time was insignificant in this case.

4.2 Comparison of SOR Values

To compare the effectiveness of the SOR coefficient in reducing the compute time, the $\Delta x = \Delta y = 5$ mm case was computed for a range of ω values from 0.05 to 2.0, with a step of 0.05. For the other two convergence parameters, the maximum iterations were set at 20 000, and the convergence tolerance at 0.001. Initial temperatures were guessed at 800 K, a temperature seen often in the colourmap, allowing the algorithm to get closer in the first few iterations. The maximum iterations were set at 20 000, as anything above this number becomes excessive. The convergence tolerance was set at 0.001 K, which represents a percent error on the lowest temperature of $1\text{x}10^{-4}$ %, which is indicative of a very precise solution. Setting the convergence tolerance this low will exaggerate the effects of the changing omega value in the number of iterations required to reach solution convergence. While an exact value for omega can be calculated, finding the exact solution requires a complex eigenvalue analysis of the coefficient matrix outside the scope of the report. Since the solution vector can be computed quite rapidly using a standard PC, the optimal value for the relaxation parameter can be quicky estimated through trial and error through the method described above. By iterating through the set of ω values described above, the results for total iterations require for convergence are summarized in Table 4.2.

Table 4.2 Table of Omega Values and Required Iterations

ω Value	Iterations to Reach Convergence	ω Value	Iterations to Reach Convergence	ω Value	Iterations to Reach Convergence	ω Value	Iterations to Reach Convergence
0.05	20000	0.55	20000	1.05	20000	1.55	8118
0.1	20000	0.6	20000	1.1	19256	1.6	7154
0.15	20000	0.65	20000	1.15	17712	1.65	6223
0.2	20000	0.7	20000	1.2	16266	1.7	5321
0.25	20000	0.75	20000	1.25	14908	1.75	4443
0.3	20000	0.8	20000	1.3	13626	1.8	3582
0.35	20000	0.85	20000	1.35	12414	1.85	2731
0.4	20000	0.9	20000	1.4	11263	1.9	1874
0.45	20000	0.95	20000	1.45	10167	1.95	967
0.5	20000	1.0	20000	1.5	9121	2.0	20000

The solution was reached in the least number of iterations for a relaxation parameter of 1.95, with 967 iterations required to reach convergence. Using this value of 1.95 nearly halved the number of iterations from the next closest number for ω = 1.90 at 1874 iterations. For low relaxation parameters, the solution did not begin converging within the 20 000-iteration limit until the value reached 1.1, with 19 256 iterations required with this ω. It is an interesting note that while the relaxation parameter of 1.95 provided an optimal solution time, when an additional test was performed at ω = 2.0, the solution did not converge. Going further, relaxation parameters were tested in steps of 0.005 from 1.950 to 1.995, to get the optimal value within 5 thousandths. Results from this trial are shown in Table 4.3.

Table 4.3 Table of Omega Values and Required Iterations Near Optimal Value

ω Value	Iterations to Reach Convergence	ω Value	Iterations to Reach Convergence
1.950	**967**	**1.975**	**540**
1.955	**865**	**1.980**	**651**
1.960	**758**	**1.985**	**819**
1.965	**637**	**1.990**	**1273**
1.970	**444**	**1.995**	**2343**

From Table 4.3, the optimal relaxation parameter was found to be 1.970, with iteration counts increasing above and below this value. Using ω = 1.970, the iteration count was reduced to 444, significantly reducing compute time. For the algorithm used in this numerical model, simulation time is directly proportional to the number of iterations required as each iteration requires a fixed compute time. Adjusting the relaxation parameter from 1.95 to 1.97 would result in a 54% decrease in compute time. Finding the optimal relaxation parameter for a given situation the Gause-Seidel method with SOR is applied to can have a significant impact in reducing the compute time required to converge on a solution.

4.3 Steady State Temperature Distribution

Since the plate is subject to convective heat transfer along the north boundary with a fluid temperature of 1800 K, and at the south boundary with a fluid temperature of 300 K it is expected that the maximum temperature within the plate be slightly less than 1800 K as the cooler boundary drives the maximum temperature down. On the southern side of the plate, the minimum temperature is expected to be above 300 K, with a higher temperature difference than what was seen at the north boundary between the plate and fluid. The reason for this larger temperature difference is the necessity for the total heat flux at each boundary to be equal for maintaining the plate in steady state. This hypothesis is confirmed with the maximum observed temperature being 1761 K, at cell (1, 160) corresponding to the northwest corner. The placement of this temperature maximum makes sense as it is the cell that is farthest from the cold gas flow and adjacent to the hot gas flow and insulated boundary. The lowest temperature calculated in the plate was 514 K, at point (160, 1) representing the southeast corner. Having the hottest and coolest temperatures at opposite corners of the plate makes sense for this case as these points represent the farthest distance possible from the opposite boundary. Additionally, the extreme points being near the insulated surfaces is sensible as the influence of other cells is minimized. As predicted, the difference between the maximum temperature and the hot gas flow is less than the difference between the minimum temperature and the cold gas flow to maintain the equality of heat flux in the plate.

Looking at the colour map of Fig. 4.6, the temperature distribution within the plate can be clearly visualized. The plot is contoured in steps of 20 K from 500 K to 1800 K, making the temperature zones within the plate clearly visible. The contours show the heat "flowing" from the northwestern side of the plate to the southeastern side of the plate. Looking left to right starting at the western boundary, the temperature is highest for the northernmost point, with the overall gradient from north to south being ~200 K. The gradient increases north to south as the point of interest moves further east due to the cold gas flow being approached. The lower temperatures emanate from the southern cold gas flow boundary, with contour lines getting wider as the northern hot gas flow on the longer boundary is approached. The temperature contours is much smaller on the eastern side of the plate compared to the western, this can be attributed to the closer proximity of the hot and cold boundaries on this side of the plate. While the 2D contour plot is good at getting the general picture of the temperature distribution within the plate, cutlines were taken along the horizontal and vertical axis to get a more granular look at the temperature distribution via 1D plots. Cutlines were taken parallel to the *x*-axis at *y* = 0, 100, 200, 300, and 400mm, plotted in Fig. 4. 10 For the *y*-axis, cutlines were taken at *x* = 0, 200, 400, 600, and 800 mm, plotted in Fig. 4.11.

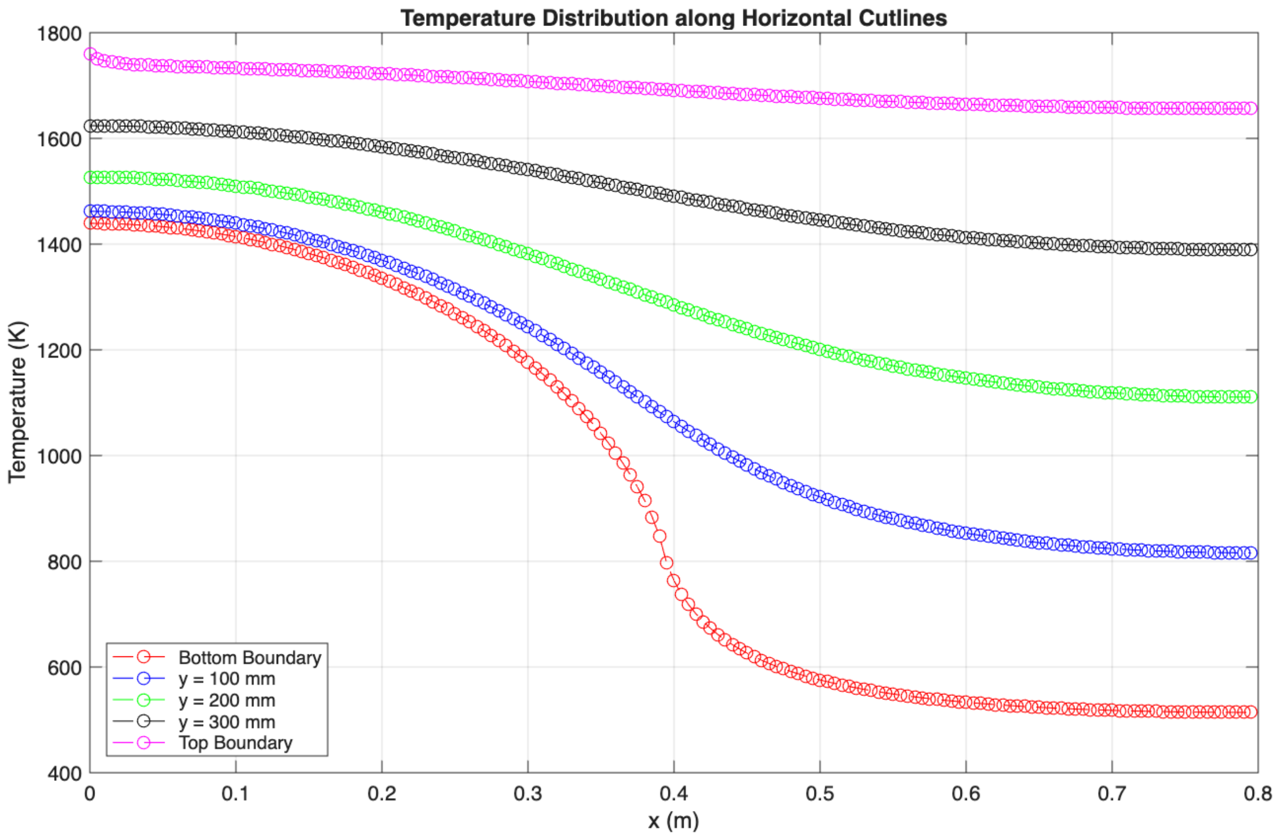

Figure 4.10 Temperature Along Horizontal Cutlines at Various Heights WRT *x*-Coordinate

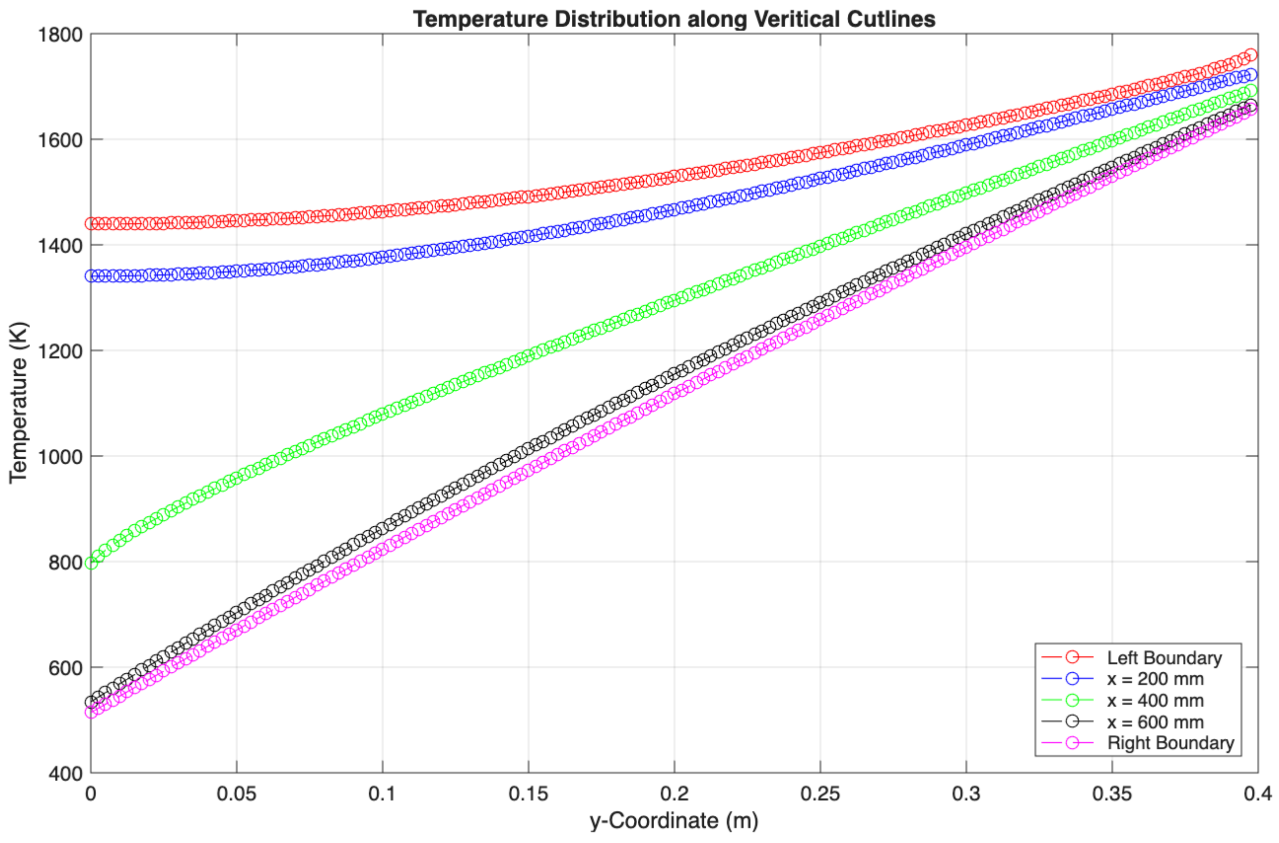

Figure 4.11 Temperature Along Vertical Cutlines at Various Positions WRT *y*-Coordinate

The horizontal cutlines give an interesting look into the behavior of the steady-state temperature in the plate from west to east. The lowest temperature of any point is observed in this plot at x = 797.5 mm (at middle of cell) for the bottom boundary cutline at 514 K. The maximum temperature is also seen on this plot, with the temperature on the top boundary at x = 2.5 mm (at middle of cell) for the top boundary line plot at 1755 K. The temperature gradient from east to west for the northern cutlines on the plate is much lower than the temperature gradient found on the southernmost cutlines. This trend follows the behavior expected from the contour plot, where the greatest number of colour contours were observed near the cold gas flow boundary. The most interesting characteristic of the cutlines, is the behavior before and after the southern boundary change at x = 200 mm. Before the transition, the temperature at each cutline decreases slowly from west to east, with the rate of change in temperature increasing around the transition point between the Neumann and Newton/Robin boundary before beginning to level off again farther west. This effect is most pronounced for the cutlines along the southern boundary and at y = 100 mm. It is worth noting that all these cutlines follow the expected behavior based on the colourmap, with the northern cuts having a consistently higher temperature than their southern neighbors. Additionally, the cutlines are grouped close together in temperatures to the west, but diverge as they move east, a behavior also observed on the colourmap.

The vertical cutlines also provide significant insight into the temperature distribution from south to north at various horizontal points in the plate. These plots show the trend of decreasing temperature from west to east within the plate, as the western boundary cutline retains the highest temperature values, and the eastern cutline the lowest temperature values. Additionally, the south to north trend observed on the colourmap with these cutline plots. The cutlines have the largest temperature spread at the southernmost point, grouping closer together as the northernmost point is reached. Cutlines east of the transition point at *x* = 400 mm show less of a temperature delta from north to south than cutlines to the west of the transition point. This is expected behavior, as the insulated condition in the southwest section of the plate reduces heat transfer in this zone, allowing the norther hot gas flow to dominate the situation.

Looking at the diagonal temperature distribution from the southwest to the northeast corner of the plate, plotted in Fig. 4.8 gives an interesting look into the nature of this temperature distribution. East of the southern boundary transition, the temperature does not change drastically along the diagonal. A valley in temperature is noted near the transition point before the shooting up as the *y*-coordinate of the diagonal cutline increases. The behavior of this diagonal cutline shows how rapidly the temperature changes in the region between the hot and cold gas flow regimes in both the *x* and *y* directions.

4.4 Heat Transfer Rates at Boundaries

Since the plate is simulated in 2-Dimensions with no given thickness, the heat transfer rate at each fluid boundary cannot be calculated as a thermal power. Instead, the heat transfer rate at each boundary will be calculated in terms of unit thickness, which could later be used to find the actual thermal power for a given thickness plate under these boundary conditions. The absolute heat transfer per unit thickness along the north and south boundary is shown in Fig. 4.12. In reality, heat transfer along the north boundary is considered positive as heat flows into the plate, and heat transfer along the south boundary is considered negative as heat leaves the plate. A summary of the thermal powers is shown in Table 4.4. It is expected that the sum of the heat transfer at the north and south boundary be zero as the plate is at steady state, meaning there is no energy being stored or released from the cells within the plate.

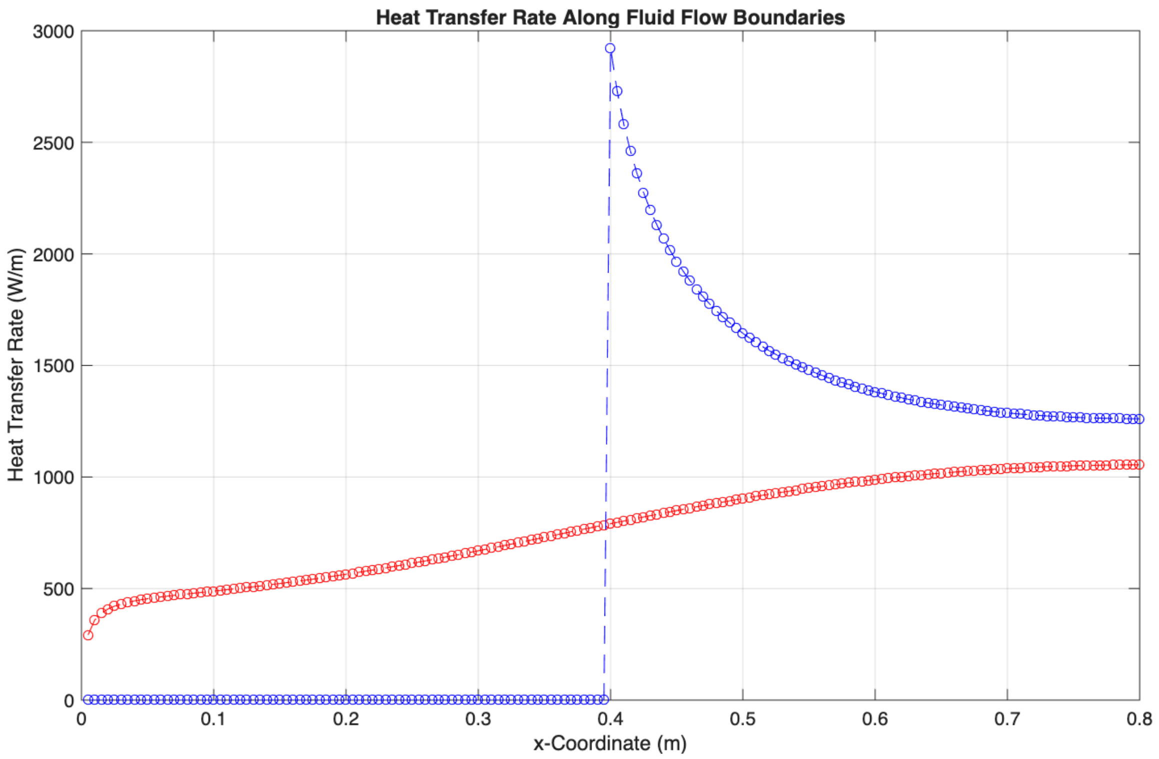

Figure 4.12 Heat Transfer Rate along Fluid Flow Boundaries WRT *x*-Coordinate

Table 4.4 Heat Transfer Rates at Fluid Flow Boundaries

Boundary	Heat Transfer Rate (kW/m)
North	**123.7**
South	**-124.9**
Difference	**-1.2**

From Fig. 4.12, the heat transfer rate of the south boundary is higher than that of the north boundary, this is the expected behavior as the sum of the rates along the boundary represents the total heat transfer rate. The total heat transfer on the northern boundary was 123.7 kW/m, with a total of -124.9 kW/m exciting the plate at the bottom boundary. This represents a total difference of -1.2 kW/m, or with the percent difference between the heat transfer magnitudes being 0.97%. While the difference of 1.2 kW/m seems substantial, indicating that the plate may not be at steady state, the percent difference is still below 1%. Since the percent difference is so low, the simulation can be considered to accurately record the heat transfer rates at steady state in and out of the plate. This result is an indicator that simulation may not be perfectly accurate at simulating real world phenomena, however, it can still yield results close enough to make engineering decisions. Using a finer grid size or a smaller convergence tolerance may help in getting the simulation results to more closely match steady state theory.

5 CONCLUSIONS

The steady-state temperature distribution for a two-dimensional flat plate was simulated using a MATLAB script programmed using the finite volume method. To apply the finite volume method to this situation, the general form of the *heat diffusion equation* was simplified for steady state condition in a domain with uniform thermal conductivity. This simplified equation was discretized using divergence theorem, with the heat flux terms approximated using a second order accurate Taylor Series expansion. Using the general form of the equation, ten situation-specific equations were derived to deal with the specific boundary conditions of the problem and solve for interior node temperatures.

A MATLAB script was developed to build a system of equations based on the cell size specified by the user. The system of equations was solved using the Gause-Seidel method, with an algorithm programed for the specific situation. This solver algorithm incorporated successive over relaxation (SOR) to reduce the number of iterations required to converge on a solution. The effects of cell size on compute times and the number of iterations required to converge on a solution were explored, to understand the balance needed between simulation accuracy and resource budget. With this study, the effects of changing the relaxation parameter were studied, with an optimal omega value of 1.97 found to produce convergence in the fewest iterations. A series of one and two-dimensional plots were generated to understand the temperature distribution in the plate at steady state and the specific effects of each boundary condition. Results were interpreted using a grid size of $\Delta x = 2\Delta y = 5$ mm, due to the improved quality of the finer grid size found during the mesh size study. Heat transfer rates at the boundary conditions were calculated to verify that the plate was in steady state. It was found that the plate received 123.7 kW/m of heat from the hot gas flow and lost 124.9 kW from the cold gas flow. This represented a percent difference of 0.97%, validating the steady state assumption.

Overall, the steady state temperature distribution within the plate followed the expected behavior, with temperatures peaking near the western side of the hot gas flow boundary and reaching the lowest value at the western side of the cold gas flow. The heat flow was visible in the plate, with a clear temperature distribution seen between the hot and cold boundaries. Predictions of the thermal behavior of the plate were assured through analyzing a series of one-dimensional plots along cutlines. This numerical simulation offered valuable insight into using the finite volume method for two-dimensional computational simulations, showing the importance of building an efficient, scalable algorithm for generating and computing a system of equations for rapid simulation. The Gause-Seidel algorithm offered a quick and efficient solver, that was simple to implement within a MATLAB script.

BIBLIOGRAPHY

[1] L. Bergman and A. D. Lavine, Fundamentals of Heat and Mass Transfer, 8th ed. Hoboken, NJ: John Wiley and Sons Inc, 2017.

[2] G. H. Golub and C. F. Van Loan, Matrix Computations, 3rd ed. Baltimore, MD: The John's Hopkins University Press, 1996

6 APPENDIX A

6.1 MATLAB Numerical Model

This script represents the overall script, that calls for the solver script shown in Section 6.2 below.

```
clear all; close all; clc;
%%
% Peter Oldreive
% B00894035
% MECH 4865 - CFD
% Project 2
% Created: Nov 24, 2025
% Updated: Nov 28, 2025
%
% This program calculates the steady state temperature field of a 2D plane
% subject to known boundary conditions. This is to statify the requirements
% of MECH 4865 Project 2. The program does not use a direct solver to
% calcualte the temperature distribution, the temperatures are calculated
% using the Gauss_Seidel method by referencing an external script.

%% Define Plate dimensions
xdim = 0.800; % x-dimension of plate (m)
ydim = 0.400; % y-deimension of plate (m)

%% Define Boundary Conditions

% Dirichlet Boundaries
T_top = 0; % Top Dirichlet condition (K)
T_bottom = 0; % Bottom Dirichlet condition (K)
T_right = 0; % Right Dirichlet condition (K)
T_left = 0;  % Left Dirichlet condtition (K)

% Neumann conditions
% West Wall
% South wall from 0 <= x <= 400 mm
% East Wall
bc_trasnition = 400; % Transition point of bottom boundary condtion from
                     % Neumann condtion to Newton/Robin

% Newton/Robin Boundaries
h_south = 1200; % South boundary convection coefficient (W/m^2)
T_inf_south = 300; % South boundary fluid temperature (K)

h_north = 1500; % North boundary convection coefficent (W/m^2)
T_inf_north = 1800; % North boundary fluid temperature (K)

% Thermal coductivity of the plate
k = 80; % Thermal conductivity of the plate (W/mK)

%% Define Finite Volume Cell Size
while(1)
    deltax = input("Cell x-dimension (mm): ")/1000; % Cell x-dimension (m)
    deltay = input("Cell y-dimension (mm): ")/1000; % Cell y-dimension (m)

    % Check for valid grid spacing
    if(mod(xdim, deltax) == 0 && mod(ydim, deltay) == 0)
        % Ensure the dimensions are divisable by the given cell dimensions
        break; % Exit the loop if valid grid spacing is provided
    else
        disp('Invalid grid spacing');
    end
end
% Define number of cells in the grid
Nx = xdim/deltax; % Number of cells in x-dimension
Ny = ydim/deltay; % Number of cells in y-dimension

Ntot = Nx*Ny; % Total Number of cells

% Display Grid information to user
fprintf(['For deltax = ', num2str(deltax), ' m and deltay = ',
num2str(deltay), ...
    ' m\nNx = ', num2str(Nx), ' and Ny = ', num2str(Ny), '\nNtot = ',
num2str(Ntot), '\n'])
tic; % Mark Start time

%% Define matrix and vectors

A = zeros(Ntot, Ntot); % Matrix of coefficents
B = zeros(Ntot, 1); % Vector of knowns

%% Fill matrix of coefficents and vector of knowns
yx = deltay/deltax; % Shorthand coefficient for use later
xy = deltax/deltay; % Shorthand coefficient for use later
% Define coefficents from ficticious node analysis

% North boundary
apfn = 1 + (deltay*h_north)/(2*k); % coefficent from equation derivation
asfn = -1 + (deltay*h_north)/(2*k); % coefficent from equation derivation
bpfn = (h_north*deltay*T_inf_north)/k; % coefficent from equation derivation

% South boundary
apfs = 1 + (deltay*h_south)/(2*k); % coefficent from equation derivation
anfs = - 1 + (deltay*h_south)/(2*k); % coefficent from equation derivation
bpfs = (h_south*deltay*T_inf_south)/k; % coefficent from equation derivation

for i = 1:Ntot % For all points
j = fix((i-1)/Nx) + 1; % Calculate the row number
  % Corner points
    if(i == 1) % Bottom left corner
       A(i, i) = (xy + yx); % Cell
       A(i, i+1) = -yx; % East Cell
       A(i, i+Nx) = -xy; % North cell
    elseif(i == Nx) % Bottom right corner
       A(i, i) = ((2+anfs/apfs)*xy + yx); % Cell
       A(i, i-1) = -yx; % West Cell
       A(i, i+Nx) = -xy; % North Cell
       B(i) = xy*bpfs/apfs; % Known
    elseif(i == Ntot - Nx + 1) % Top Left
        A(i, i) = ((2+asfn/apfs)*xy + yx); % Cell
        A(i, i+1) = -yx; % East Cell
        A(i, i-Nx) = -xy; % South Cell
        B(i) = xy*bpfn/apfn; % Known
    elseif(i == Ntot) % Top right
        A(i, i) = ((2+asfn/apfn)*xy + yx); % Point
        A(i, i-1) = -yx; % West point
        A(i, i-Nx) = -xy; % South Point
        B(i) = xy*bpfn/apfn; % Known
    % Boundary Nodes
    elseif(j == 1 && i*deltax < xdim/2 && i > 1) % Left Lower boundary
        A(i, i) = (2*yx+xy); % Point
        A(i, i+1) = -yx; % East point
        A(i, i-1) = -yx; % West point
        A(i, i+Nx) = -xy; % North point
    elseif(j == 1 && i*deltax >= xdim/2 && i < Nx) %Right lower boundary
        % Right lower boundary contains node at transition point
        %from Neumann condition to Newton/Robin Condition.
        A(i, i) = ((2+anfs/apfs)*xy + 2*yx); % Point
        A(i, i+1) = -yx; % East point
        A(i, i-1) = -yx; % West point
        A(i, i+Nx) = -xy; % North point
        B(i) = xy*bpfs/apfs; % Known
    elseif(j > 1 && mod(i, Nx) == 1 && j<Ny) % Left boundary
        A(i, i) = (2*xy + yx); % Point
        A(i, i+1) = -yx; % East point
        A(i, i+Nx) = -xy; % North point
        A(i, i-Nx) = -xy; % South point
    elseif(j > 1 && mod(i, Nx) == 0 && j < Ny) % Right boundary
        A(i, i) = (2*xy + yx); % Point
        A(i, i-1) = -yx; % West point
        A(i, i+Nx) = -xy; % North point
        A(i, i-Nx) = -xy; % South point
    elseif(j == Ny && i > Nx*(j-1) + 1 && i < Nx*j) % Top
        A(i, i) = ((2+asfn/apfn)*xy + 2*yx); % Point
        A(i, i-1) = -yx; % West point
        A(i, i+1) = -yx; % East point
        A(i, i-Nx) = -xy; % South point
        B(i) = xy*bpfn/apfn; % Known
    else % Interior nodes
        A(i, i) = 2*(xy+yx); % Point
        A(i, i-1) = -yx; % West point
        A(i, i+1) = -yx; % East point
        A(i, i+Nx) = -xy; % North point
        A(i, i-Nx) = -xy; % South point
    end
end

%% Call Solver Script

% writematrix(A, "tools/coeff_matrix.csv"); % Write Matrix of Coefficients
(A)
%                                           % to a .csv file
% writematrix(B, 'tools/knowns.csv'); % Write vector of knowns (B) to a .csv
file

% Run solver script saved one directory below within the tools folder.
% changing directory may be required to run script on a different PC
run('tools/gauss_seidel.m');

% Read in solution vector gererated by gauss seidel mechanism
% T = readmatrix('tools/solution_vector.csv');
% Take T from workspace
T = xj;

%Record compute time
endtime = toc;
disp(['Compute time:' num2str(round(endtime, 2)) 's']) % Print compute time

%% Post Process

% Turn temperature vector into a NyxNx matrix for plotting

T_plot = zeros(Ny, Nx); % Define matrix
index = 1; % Index prepresents point number
for y = 1:Ny % For all rows
    for x = 1:Nx % For all columns
     T_plot(y,x) = T(index);
     index = index + 1;
    end
end

% Write matrix of temperature field for later comparison
writematrix(T_plot, 'Temperature_Field.csv');

% Generate vector of discretized x-values
x_values = zeros(Nx,1);
for x = 1:Nx
    x_values(x) = (x-1) * deltax;
end

% Generate vector of discretized y-values
y_values = zeros(Ny,1);
for y = 1:Ny
    y_values(y) = (y-1) * deltay;
end

% Make a grid of coordinates
[X, Y] = meshgrid(x_values, y_values);


% Generate a vector of bottom boundary, and 100 mm steps to top
T_bottom_values = T_plot(1, :); % Make botttom boundary a vector
T_100mm_above_bottom = T_plot(fix(0.100/deltay), :); % Closest point to
                                                   % 50mm above bottom
                                                   % boundary
T_200mm_above_bottom = T_plot(fix(0.200/deltay), :); % Closest point to
                                                   % 100 mm above bottom
                                                   % boundary
T_300mm_above_bottom = T_plot(fix(0.300/deltay), :); % Closest point to
                                                   % 150mm above bottom
                                                   % boundary
```

```
T_top_values = T_plot(Ny, :); % Top boundary

% Generate cutlines paralell to the y-axis in 100 mm increments
T_left_values = T_plot(:,1); % 1st column is left boundary
T_x_200 = T_plot(:, fix(0.200/deltax)); % Temperatures at 100 mm right of the
                                         % left boundary
T_x_400 = T_plot(:, fix(0.400/deltax)); % temperatures at 200 mm right of the
                                         % left boundary
T_x_600 = T_plot(:, fix(0.600/deltax)); % Temperatures at 300 mm right of the
                                         % left boundary
T_right_values = T_plot(:, Nx); % Right boundary temperatues

% Find diagonal vector
if Nx == Ny
    T_diag = zeros(Nx, 1);
    for i = 1:Nx
      T_diag(i) = T_plot(i, i);
    end
else
    rowcount = 1; % Start in 1st row
    T_diag = zeros(Nx/2, 1); % Define diagonal vector
    for columncount = 2:Nx % count up to Nx
        if (mod(columncount, 2) == 0 && rowcount < Ny+2) % count up rows
until max
            T_diag(rowcount, 1) = T_plot(rowcount, columncount); % index
diagonals out of matrix
            rowcount = rowcount + 1; % Count up row
        end
    end
end

% Find heat transfer rate at north boundary
Tfn = zeros(Nx, 1); % Define ficticious north temperature
Tnw = zeros(Nx, 1); % Define north wall temperature
qn = zeros(Nx, 1); % Define heat transfer per unit thickenss
% For all x
for i = 1:Nx
    Tfn(i) = (-asfn*T_plot(Ny, i) + bpfn)/apfn;
    Tnw(i) = (T_plot(Ny, i) + Tfn(i))/2;
    qn(i) = - h_north*deltax*(Tnw(i) - T_inf_north);
end

% Find heat transfer rate at south boundary
Tfs = zeros(Nx, 1); % Define ficticious south temperature
Tsw = zeros(Nx, 1); % Define south wall temperature
qs = zeros(Nx, 1); % Define heat transfer per unit thickenss
% For all x
for i = 1:Nx
    if i < 0.4/deltax
        % No HT through insulation
    else
        Tfs(i) = (bpfs - anfs*T_plot(1, i))/apfs;
        Tsw(i) = (T_plot(1, i) + Tfs(i))/2;
        qs(i) = - h_south*deltax*(Tsw(i) - T_inf_south);
    end
```

```
end

% Find total q
QN = sum(qn, 'all'); % North Wall heat transfer per unit thickness (W/m)
QS = sum(qs, 'all'); % South Wall heat transfer per unit thickenss (W/m)

%% Plotting

% Show discretized 2D grid of Nx x Ny points
figure;
plot(X, Y, 'o', 'LineWidth', 2, 'color', 'r', 'MarkerSize',1)
set(gca,'YDir','normal'); % make y increase upward visually
axis equal tight;
xlim([0 xdim])
ylim([0 ydim])
title('Discretized Grid of Nx x Ny Points Spaced by \Deltax and \Deltay');
xlabel('x-Coordinate (m)');
ylabel('y-Coordinate (m)');

% 2D Colourmap
figure('Name', '2D Colour Map','NumberTitle','off');
hold on
contourf(T_plot,500:20:1800,'LineColor', "none"); % Add Contour Lines every
10 K
set(gca,'YDir','normal');
axis tight;
pbaspect([2 1 1])
a=colorbar;
a.Label.String = 'Temperature, T (K)';
a.Limits = [500 1800];
title('2D Colourmap with Contour Lines');
xlabel('x-Coordinate (m)');
ylabel('y-Coordiante (m)');
xticks(0:0.10/deltax:Nx)
xticklabels(0:0.10:xdim)
yticks(0:0.1/deltay:Ny)
yticklabels(0:0.10:ydim)


% Temperature along horizontal cutlines
figure();
plot(x_values, T_bottom_values, 'o', 'MarkerSize', 6, 'color', 'r', ...
    'LineStyle','--' );
hold on;
plot(x_values, T_100mm_above_bottom, 'o', 'MarkerSize', 6, 'color', 'b', ...
    'LineStyle','--' );
hold on;
plot(x_values, T_200mm_above_bottom, 'o', 'MarkerSize', 6, 'color', 'g', ...
    'LineStyle','--' );
hold on;
plot(x_values, T_300mm_above_bottom, 'o', 'MarkerSize', 6, 'color', 'k', ...
    'LineStyle','--' );
hold on;
plot(x_values, T_top_values, 'o', 'MarkerSize', 6, 'color', 'm', ...
    'LineStyle','--');
```

```
hold on;
xlabel('x (m)');
ylabel('Temperature (K)');
title('Temperature Distribution along Horizontal Cutlines');
legend('Bottom Boundary', 'y = 100 mm', 'y = 200 mm', 'y = 300 mm', ...
    'Top Boundary', 'Location','southwest');
grid on;

% Temperature cutlines paralell to y-axis
figure();
plot( y_values, T_left_values, 'o', 'MarkerSize', 6, 'color', 'r', ...
    'LineStyle','--');
hold on;
plot( y_values, T_x_200, 'o', 'MarkerSize', 6, 'color', 'b', ...
    'LineStyle','--');
hold on;
plot(y_values, T_x_400, 'o', 'MarkerSize', 6, 'color', 'g', ...
    'LineStyle','--');
hold on;
plot(y_values, T_x_600, 'o', 'MarkerSize', 6, 'color', 'k', ...
    'LineStyle', '--');
hold on;
plot(y_values , T_right_values, 'o', 'MarkerSize', 6, 'color', 'm', ...
    'LineStyle', '--');
hold on;
xlabel('y-Coordinate (m)');
ylabel('Temperature (K)');
title('Temperature Distribution along Veritical Cutlines');
legend('Left Boundary', 'x = 200 mm', 'x = 400 mm', 'x = 600 mm', ...
    'Right Boundary', 'Location','southeast');
grid on;

% Diagonal Temperature
figure();
if(length(T_diag) == Nx)
    plot(deltax:deltax:xdim, T_diag, 'o', 'MarkerSize', 5, 'color', 'r', ...
    'LineStyle','--');
else
plot(deltax:2*deltax:xdim, T_diag, 'o', 'MarkerSize', 5, 'color', 'r', ...
    'LineStyle','--');
end
hold on;
ylabel('Temperature (K)');
xlabel('x-Coordinate (mm)');
title('Temperature Along Diagonal Cutline (Bottom Left to Top Right)');
grid on;


% Heat Transfer Plot
figure();
plot(deltax:deltax:xdim, qn, 'o', 'MarkerSize', 5, 'color', 'r', ...
    'LineStyle','--');
hold on;
plot((deltax):deltax:xdim, abs(qs), 'o', 'MarkerSize', 5, 'color', 'b', ...
    'LineStyle','--');
```

```
ylabel('Heat Transfer Rate (W/m)');
xlabel('x-Coordinate (m)');
title('Heat Transfer Rate Along Fluid Flow Boundaries');
grid on;

%% Display Key Values
disp(['Maximum T = ' num2str(max(T)) 'K'])
disp(['Minimum T = ' num2str(min(T)) 'K'])
disp(['North Wall Heat Transfer = ' num2str(round(QN/1000, 1)) ' kW/m'])
disp(['South Wall Heat Transfer = ' num2str(round(QS/1000, 1)) ' kW/m'])
```

6.2 Gause-Seidel Solver

```
% This script implements the Gauss-Seidel method for approximating
% the solution to a linear system of equations.
%
% This script can be implemented into other scripts for solving systems
% using the Gauss-Seidel method
%
% Author: Peter Oldreive
% Created: 2025-11-24
% Updated: 2025-11-24
%% Convergance specific parameters
omega          = 1.97;     % relaxation factor (ω = 1 → pure Gauss–Seidel)
tol            = 0.001;    % convergence tolerance
maxIterations = 20000; % Maximum number of iterations before forcing
                       % convergance of the algorithm
guess = 800; % First guess of solution vector values

%% Import .csv Files from External Script

% % Import matrx of coefficents
% A = readmatrix('coeff_matrix.csv');
% % Import vector of knowns
% B = readmatrix('knowns.csv');
%% Take Matricies Directly from workspace
coeffMatrix = A;
knownVector = B;
%% Checks
% Check that vector has the same mumber of elements as rows in the
% coefficent matrix. If these are not compatible a warning will be given
if (length(coeffMatrix(:,1)) ~= length(knownVector)) | ...
    (length(coeffMatrix(1,:)) ~= length(knownVector))
    fprintf('Matrix and Vector not Compatble \n')
    % Write a 2x1 matrix of zeros to provide some output
    writematrix(zeros(2,1), 'solution_vector.csv');
    return; % Pause running script
end

%% Initialize first Guess of Solution Vector
% Initialize the guess for temperature values (assume 300 K)
coeffDim = size(coeffMatrix); % Save dimensions of coefficent matrix
% Initialize solution vector with 300 K
xj = guess * ones(coeffDim(1),1);

%% Run iteritve Gauss-Seidel Method
for iter = 1:maxIterations
    xj_1 = xj;
    for i = 1:Ny
        index = (i-1)*Nx + 1;
        for j = 1:Nx
            term_E = 0;
            if(j < Nx)
                term_E = xj_1(index+1)*A(index, index + 1);
            end
            term_W = 0;
            if(j > 1)
```

```
                term_W = xj(index-1)*A(index, index - 1);
            end
            term_N = 0;
            if(i < Ny)
                term_N = xj_1(index+Nx)*A(index, index + Nx);
            end
            term_S = 0;
            if(i > 1)
                term_S = xj(index-Nx)*A(index, index - Nx);
            end

            xj(index) = (B(index) - term_N - term_S - term_E -
term_W)/A(index,index);
            xj(index)= (omega)*xj(index) + (1-omega)*xj_1(index);
            index = index + 1;
        end
    end
         % Check for convergance
        if max(abs(xj - xj_1)) < tol
            % If the maximum absulte difference between all new and old
            % vaules is less than the tolarance, break out of the loop.
            %fprintf('Converged after %d iterations.\n', iter);
            disp(['Converged in: ' num2str(iter) ' Iterations'])
            break; % Exit the loop if converged
        end
end

% %% Output results
% % Save solution vector to a .csv file
% writematrix(xj, 'solution_vector.csv');

return;
```