A Solution to Laplace Equation Using Finite Volume Method

Danial Rezaee[1]

Project information:
Course:
Computational Fluid Dynamics
Project No.:
No. 1
Professor:
Dr. A. Nejat[1]
Student No.:
810600196
Created:
March 2022

Abstract

In this project we solved Laplace Equation for a steady-state 2D heat transfer problem over a unit square domain. Although this problem has an analytical solution, we solved it using Finite Volume Method (FVM). This solution will give us a real understanding of the Finite Volume Method. also, we will see how to discretize the domain, writing the flux integral, comparing FVM and analytical solution, calculating the Errors, and finally discussing the results. A C++ program has been written to solve this equation using FVM.

1 Problem Description

The equation to be solve here is:

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

This equation is called "*Laplace Equation*", As you can see in Figure 1 the equation's Domain is a unit square with the following Boundary Conditions:

$$\begin{cases} @ & (0,y) \rightarrow T(0,y) = 0 \\ @ & (x,0) \rightarrow T(x,0) = 0 \\ @ & (1,y) \rightarrow T(1,y) = 0 \\ @ & (x,1) \rightarrow T(x,1) = \sin(\pi x) \end{cases} \qquad (2)$$

This equation with the mentioned Boundary Conditions (2) has an analytical solution as following:

$$T(x,y) = \frac{\sin(\pi x)\sinh(\pi y)}{\sinh(\pi)} \qquad (3)$$

We will use this analytical solution (3) to assess the accuracy of our FVM solution and norm of errors in the following sections.

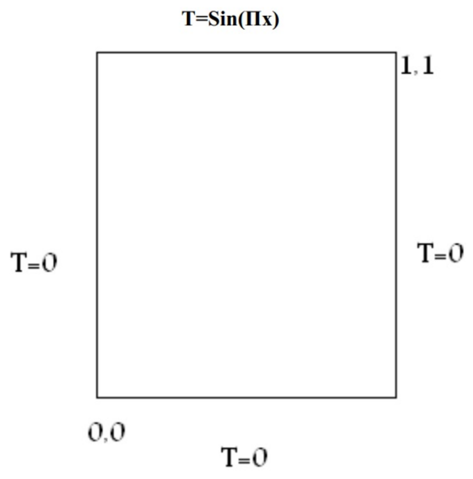


Figure 1: The Equation Domain

[1]School of Mechanical Engineering, College of Engineering, University of Tehran, Tehran, Iran

2 Solution Process

2.1 Converting Control Volume integral to Flux integral

We used Gauss' theorem to convert the Control Volume (CV) integral to Flux integral. this is a necessary step for the domain discretization, which we get to know it in the following sections. According to Gauss' theorem, if V is a subset of $\mathbb{R}^3$ which is a compact and has a piecewise smooth boundary S (Figure 2). If $\mathbf{F}$ is a continuously differentiable vector field defined on a neighborhood of V, then:

$$\iiint_V (\nabla \cdot \mathbf{F}) dV = \oiint_S (\mathbf{F} \cdot \mathbf{n}) dS \tag{4}$$

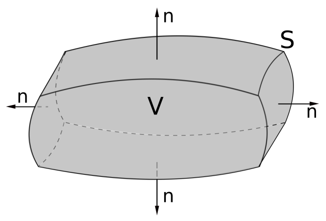


Figure 2: A region V bounded by the surface S with the surface normal $\mathbf{n}$.

This equation (4), apparently converts the integral over a volume to the flux integral on the surface enclosing it. This is the fundamental meaning of the FVM.

We now use this theorem to convert the integral of *Laplace* equation over a CV to a Flux integral over the surface enclosing it:

$$\iint_A \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) dA = \iint_A (0) dA \tag{5}$$

We can rewrite equation (5) as following:

$$\iint_A \nabla \cdot \left(\frac{\partial T}{\partial x} \hat{i} + \frac{\partial T}{\partial y} \hat{j} \right) dA = 0 \tag{6}$$

Regrading $\left(\frac{\partial T}{\partial x} \hat{i} + \frac{\partial T}{\partial y} \hat{j} \right) = \nabla T$ and comparing equation (6) with (4) we can write:

$$\oint_{CS} (\nabla T \cdot \hat{n}) dS = 0 \tag{7}$$

Where CS stands for Control Surface, in the next section we will use (7) to discretize the solution domain.

2.2 Domain Descretization

We discretize the domain and divide it into several squares, e.g., a 10 by 10 mesh, as shown in Figure 3. For simplicity we assumed equal mesh lengths in both directions ($\Delta x = \Delta y$).

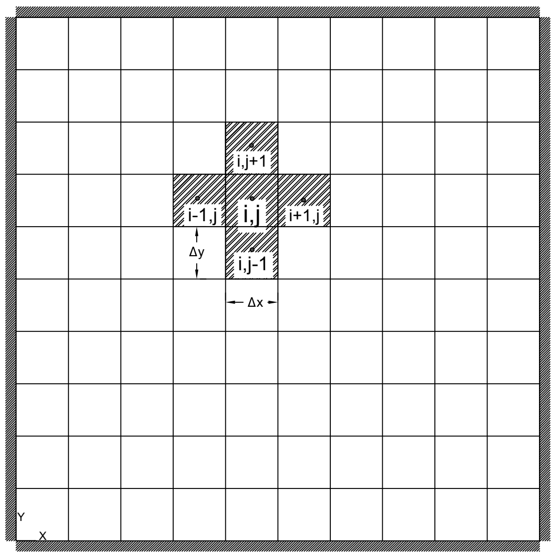


Figure 3: A 10 by 10 square mesh

We always discretize the equations using center points of CVs or CSs, if ϕ is a scalar function over $\mathbb{D} \subset \mathbb{R}^n$, then we can say:

$$\phi_c \equiv \bar{\phi}_{\mathbb{D}} + O(\Delta^k) \tag{8}$$

Where ϕ_c is the value of ϕ at the center of $\mathbb{D}$, $\overline{\phi}_{\mathbb{D}}$ is the average of ϕ over $\mathbb{D}$, k is order of solution, and Δ is the mesh size.

Consider (i, j) cell in Figure 3, this cell in shown more detailed in Figure 4 we write the equation (7) for this cell:

$$\oint_{CS} (\nabla T \cdot \hat{n}) dS = (\nabla T \cdot \hat{n})|_{(i-\frac{1}{2},j)} \Delta y + (\nabla T \cdot \hat{n})|_{(i+\frac{1}{2},j)} \Delta y + (\nabla T \cdot \hat{n})|_{(i,j+\frac{1}{2})} \Delta x + (\nabla T \cdot \hat{n})|_{(i,j-\frac{1}{2})} \Delta x = 0 \tag{9}$$

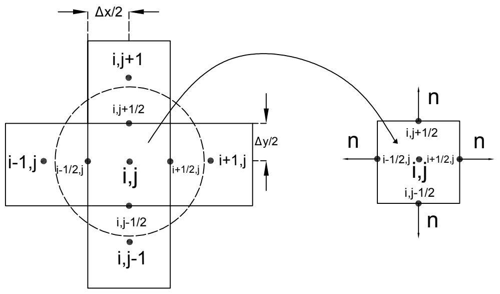


Figure 4: Cell (i, j)

As you see in Figure 4 the unit vector on each face of CS is either $\pm\hat{i}$ or $\pm\hat{j}$, rewriting equation (9):

$$\oint_{CS} (\nabla T \cdot \hat{n}) dS = (\nabla T \cdot (-\hat{i}))|_{(i-\frac{1}{2},j)} \Delta y + (\nabla T \cdot \hat{i})|_{(i+\frac{1}{2},j)} \Delta y + (\nabla T \cdot \hat{j})|_{(i,j+\frac{1}{2})} \Delta x + (\nabla T \cdot (-\hat{j}))|_{(i,j-\frac{1}{2})} \Delta x = 0 \tag{10}$$

Regarding $\left(\frac{\partial T}{\partial x}\hat{i} + \frac{\partial T}{\partial y}\hat{j}\right) = \nabla T$, the equation (10) becomes:

$$\oint_{CS} (\nabla T \cdot \hat{n}) dS = -\frac{\partial T}{\partial x}\bigg|_{(i-\frac{1}{2},j)} \Delta y + \frac{\partial T}{\partial x}\bigg|_{(i+\frac{1}{2},j)} \Delta y + \frac{\partial T}{\partial y}\bigg|_{(i,j+\frac{1}{2})} \Delta x - \frac{\partial T}{\partial y}\bigg|_{(i,j-\frac{1}{2})} \Delta x = 0 \tag{11}$$

So, we need to evaluate $\frac{\partial T}{\partial x}$ and $\frac{\partial T}{\partial y}$ on the mentioned points. We can use second-order approximation like following:

$$\begin{cases} \frac{\partial T}{\partial x}\Big|_{(i+\frac{1}{2},j)} &= \frac{T_{i+1,j}-T_{i,j}}{\Delta x} + O(\Delta x^2) \\ \frac{\partial T}{\partial x}\Big|_{(i-\frac{1}{2},j)} &= \frac{T_{i,j}-T_{i-1,j}}{\Delta x} + O(\Delta x^2) \\ \frac{\partial T}{\partial y}\Big|_{(i,j-\frac{1}{2})} &= \frac{T_{i,j}-T_{i,j-1}}{\Delta y} + O(\Delta y^2) \\ \frac{\partial T}{\partial y}\Big|_{(i,j+\frac{1}{2})} &= \frac{T_{i,j+1}-T_{i,j}}{\Delta y} + O(\Delta y^2) \end{cases} \tag{12}$$

Substituting equation (12) into equation (11) we will have:

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2} = 0 \tag{13}$$

This equation is valid everywhere but the boundary cells. For the boundary cells $i\pm1$ or $j\pm1$ might not be existing. So, for these cells we have to rewrite equation (13), for the fixed temperature B.C. we can use *Ghost Cells*, like Figure 5 which are imaginary cells beyond the domain of solution.

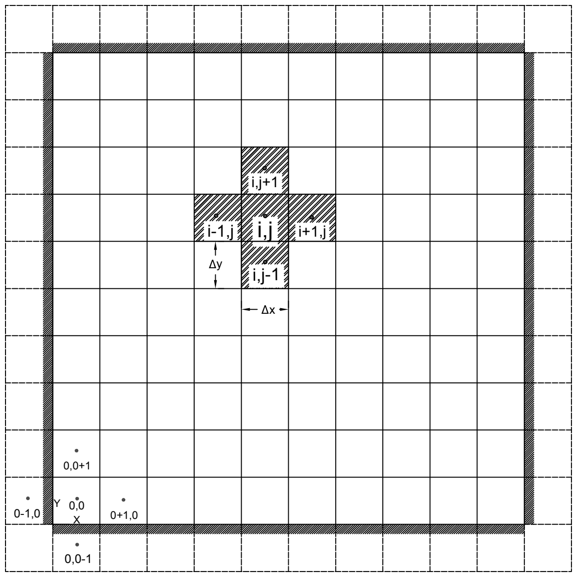


Figure 5: Ghost Cells

For example, for cell $(0,0)$ [1] in the Figure 5 we can write $T_{0-1,0}$ and $T_{0,0-1}$ (which are outside of the domain) as:

$$\frac{T_{0-1,0}+T_{0,0}}{2} = T_{W_1}$$
$$\frac{T_{0+1,0}+T_{0,0}}{2} = T_{W_2}$$

Where T_{W_1} and T_{W_2} are the fixed temperatures of left and bottom boundary lines (boundary conditions are known) So, we have:

$$\begin{cases} T_{0-1,0} &= 2T_{W_1} - T_{0,0} \\ T_{0,0-1} &= 2T_{W_2} - T_{0,0} \end{cases} \tag{14}$$

Substituting equation (14) into (13) we will find an equation for $T_{0,0}$ we have to do similar operations on other boundary cells. At the end we have an equation for each cell, in other words we have a system of linear equations. An augmented matrix will be resulted as following:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} T_{0,0} \\ \vdots \\ T_{n-1,n-1} \end{bmatrix} = \begin{bmatrix} b_{0,0} \\ \vdots \\ b_{n-1,n-1} \end{bmatrix} \tag{15}$$

No need to say that most of the elements of matrix (15) are zeros, in the following sections we will discuss how to solve equation (15).

2.3 Solving the system of Linear Equations

There are different methods to solve equation (15), from these methods, 4 of them including Gaussian Elimination Method, Jacobi Method, Gauss-Sidel Method and, Successive Overrelaxation (SOR) have been discussed and compared in this project.

2.3.1 Gaussian Elimination

In this method an augmented matrix is created as following:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_{0,0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_{n-1,n-1} \end{array}\right] \tag{16}$$

Then the matrix (16) is transformed into a upper triangular matrix using elementary row operations:

$$\left[\begin{array}{cccc|c} a'_{11} & a'_{12} & \dots & a'_{1n} & b'_{0,0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & a'_{nn} & b'_{n-1,n-1} \end{array}\right] \tag{17}$$

The system of equations can be solved now using backward substitution.

2.3.2 Jacobi

In this method an initial value is allocated to all the cells (e.g. 0 for all $T_{i,j}$) and, every (i,j) cell equation is solved for $T_{i,j}$. This operation will be continued for many iterations each iteration using the values of $T_{i,j}$ from previous iteration. For example if $T_{i,j}$ is a non-boundary cell we can write:

$$T_{i,j}^{(k+1)} = \frac{\Delta y^2}{2(\Delta x^2+\Delta y^2)}(T_{i+1,j}^{(k)} + T_{i-1,j}^{(k)}) + \frac{\Delta x^2}{2(\Delta x^2+\Delta y^2)}(T_{i,j+1}^{(k)} + T_{i,j-1}^{(k)}) \tag{18}$$

Where $k+1$ is current iteration and k is the previous iteration.

[1] We used $(0,0)$ instead of common $(1,1)$ for the first cell because most of the programming languages (in case of this project C++) use $(0,\dots,0)$ to address the first element of an array.

2.3.3 Gauss-Sidel

This method is like *Jacobi*, but instead of using previous iteration values k, we use current iteration values $k+1$ if they exist. For example if we are solving the domain column by column, the previous column and values are already updated, So if $T_{i,j}$ is a non-boundary cell we can rewrite equation (18):

$$T_{i,j}^{(k+1)} = \frac{\Delta y^2}{2(\Delta x^2+\Delta y^2)}(T_{i+1,j}^{(k)} + T_{i-1,j}^{(k+1)}) + \frac{\Delta x^2}{2(\Delta x^2+\Delta y^2)}(T_{i,j+1}^{(k)} + T_{i,j-1}^{(k-1)}) \quad (19)$$

2.3.4 SOR

This method is like *Gauss-Sidel* but using an Overrelaxation Factor (ω) to make the convergence faster:

$$T_{i,j}^{(k+1)'} = T_{i,j}^{(k)'} + \omega(T_{i,j}^{(k+1)} - T_{i,j}^{(k)'}) \quad (20)$$

Where:

$T_{i,j}^{(k+1)}$	:	Most recent value of $T_{i,j}$
$T_{i,j}^{(k)'}$	:	Value from previous iteration if the overrelaxation was applied

2.4 Accuracy Assessment

2.4.1 Local Error Norms

We can define Error for each cell (in case of 2 dimensional domain) as:

$$\overline{E}_{i,j} = \frac{1}{A_{i,j}} \iint_{CV_{i,j}} E(x,y)dA$$
$$\overline{E}_{i,j} = \frac{1}{A_{i,j}} \iint_{CV_{i,j}} T_{i,j} - T(x,y)dA$$
$$\overline{E}_{i,j} = \frac{1}{A_{i,j}} \iint_{CV_{i,j}} T_{i,j}dA - \frac{1}{A_{i,j}} \iint_{CV_{i,j}} T(x,y)dA \quad (21)$$

Since the $T_{i,j}$ has only one value all over the CV (except for faces), it can be got out of the integral. The second term $\frac{1}{A_{i,j}} \iint_{CV_{i,j}} T(x,y)dA$ is the definition of Average Temperature, So, equation (21) becomes:

$$\overline{E}_{i,j} = T_{i,j} - \overline{T}_{CV_{i,j}} \quad (22)$$

Recalling equation (8) we can write (22) as:

$$T_{i,j} \equiv \overline{T}_{CV_{i,j}} + O(\Delta^k) \quad (23)$$

Where Δ is Mesh Length and k is the order of solution.

using equation (23) the Error (22) becomes:

$$\overline{E}_{i,j} \equiv O(\Delta^k) \quad (24)$$

Equation (24) implies an important issue. If $M1$ and $M2$ are meshes with mesh length Δ and $\frac{\Delta}{2}$ we can say:

$$\frac{\|E|_{M_1}\|}{\|E|_{M_2}\|} \equiv \frac{O(\Delta^k)}{O((\frac{\Delta}{2})^k)} = 2^k \quad (25)$$

2.4.2 Global Error Norms

Local Error Norms do not show the overall error of the domain, also, we need a criteria to know when to stop the iterations or when our overall mesh size is good enough. So, we define Global Error Norms as follows:

$$L_p = \left(\frac{\Sigma_{j=0}^{n-1}\Sigma_{i=0}^{n-1} A_{i,j} \left|\overline{E_{i,j}^p}\right|}{\Sigma_{j=0}^{n-1}\Sigma_{i=0}^{n-1} A_{i,j}} \right)^{\frac{1}{p}} \quad (26)$$

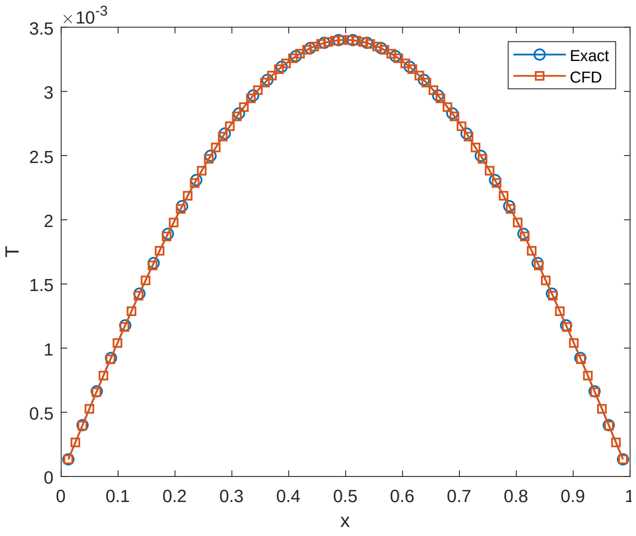
From these Global norms we are mostly interested in L_1, L_2, and, $L_\infty = max\{\left|\overline{E_{i,j}}\right|\}$

3 Results and Discussions

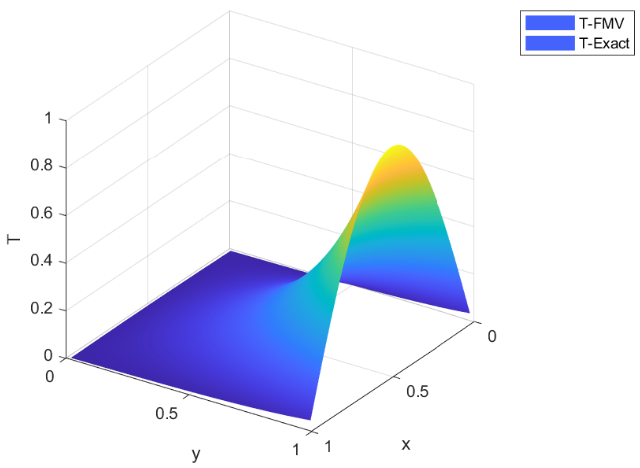
3.1 Verification

3.1.1 FVM vs. Exact Solution Plot

In Figure 6 the FVM solution is plotted against Exact solution for 1600 CVs. As you see in Figure 6a the curves are laid on each other perfectly.



(a) FVM vs. Exact solution curve



(b) FVM vs. Exact solution surface

Figure 6: FVM vs. Exact solution

3.1.2 Second Order Solution

Recalling Equation (25), the L_2 or L_1 plot against mesh length must be a line with the slope equals the order of solution. So, the line slope must be around 2, as you see in Figure 7.

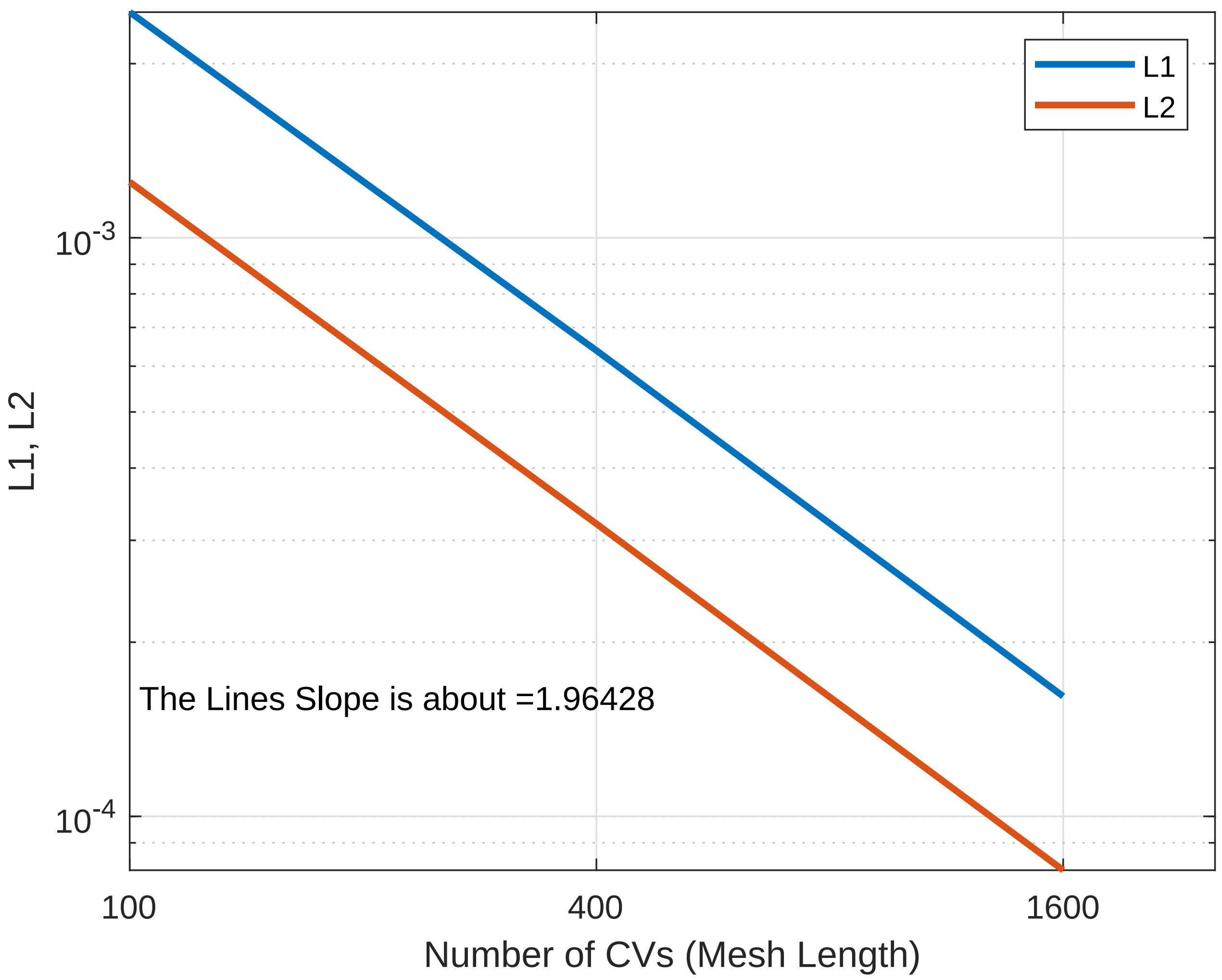


Figure 7: Verification of second order solution

3.2 How Fast Different Methods Converge

3.2.1 Overall Number of Iterations

The iterations are terminated when L_2 does not reduce anymore, but it still may reduce by a very small amount. For this project we terminated the iterations when $\left|L_2^{k+1} - L_2^k\right| \leq 1e - 10$.

As you see in Figure 8 *SOR* is significantly faster than other methods. As we expected the *Jacobi* method is the slowest one.

As the number of CVs increases the convergence happens at larger iterations and the L_2 decreases, this means larger number of CVs leads to more precise results, but requires more resources and time.

For *SOR* method we see by increasing of *Overrelaxation Factor* (ω) the convergence happens in lower number of iterations.

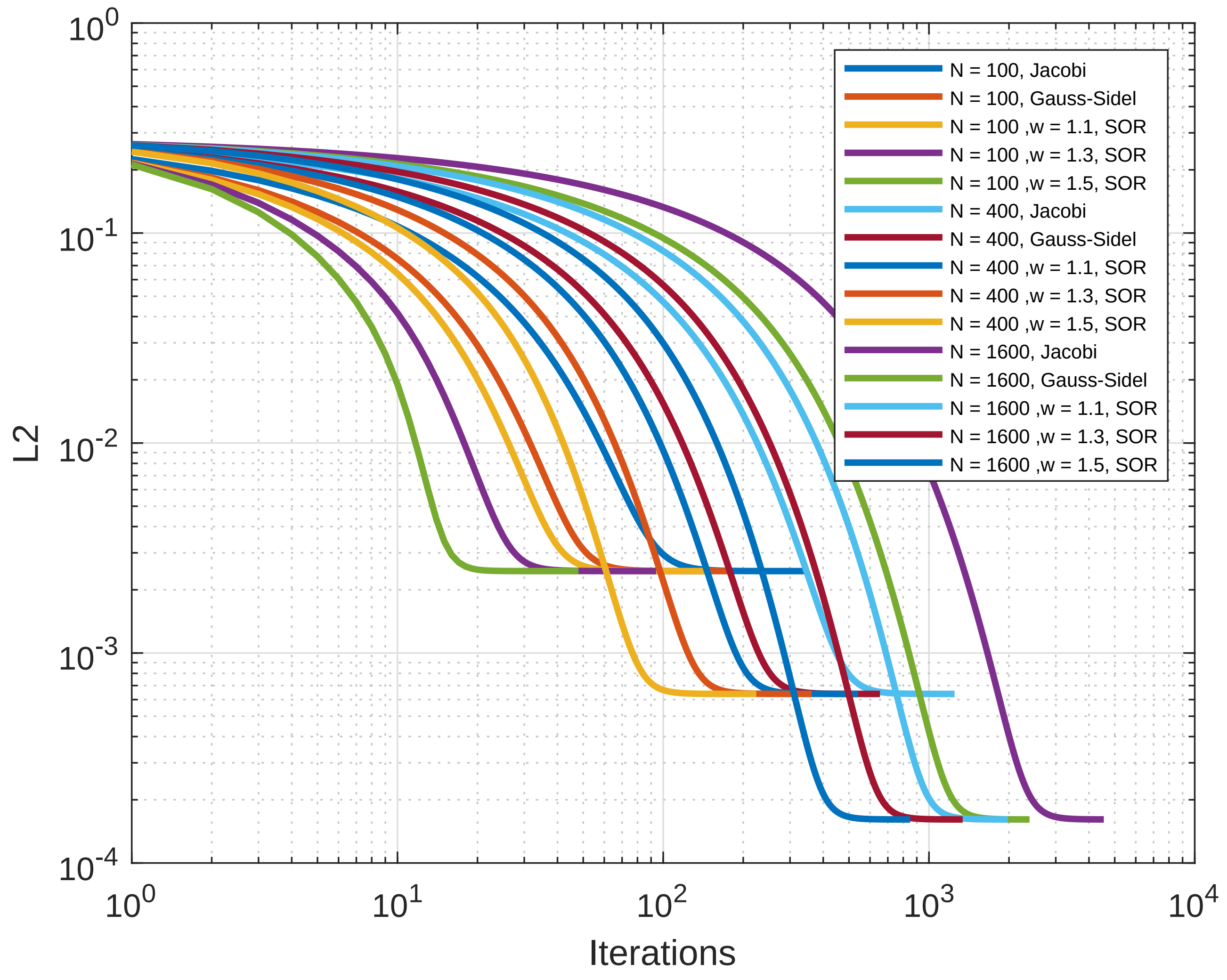


Figure 8: Iterations for each method L_2 convergence

3.2.2 SOR Overrelaxation Factor

In the Figure 9, you see how the *Overrelaxation Factor* (ω) affects the convergence in *SOR* method. As the ω increases the convergence happens faster.

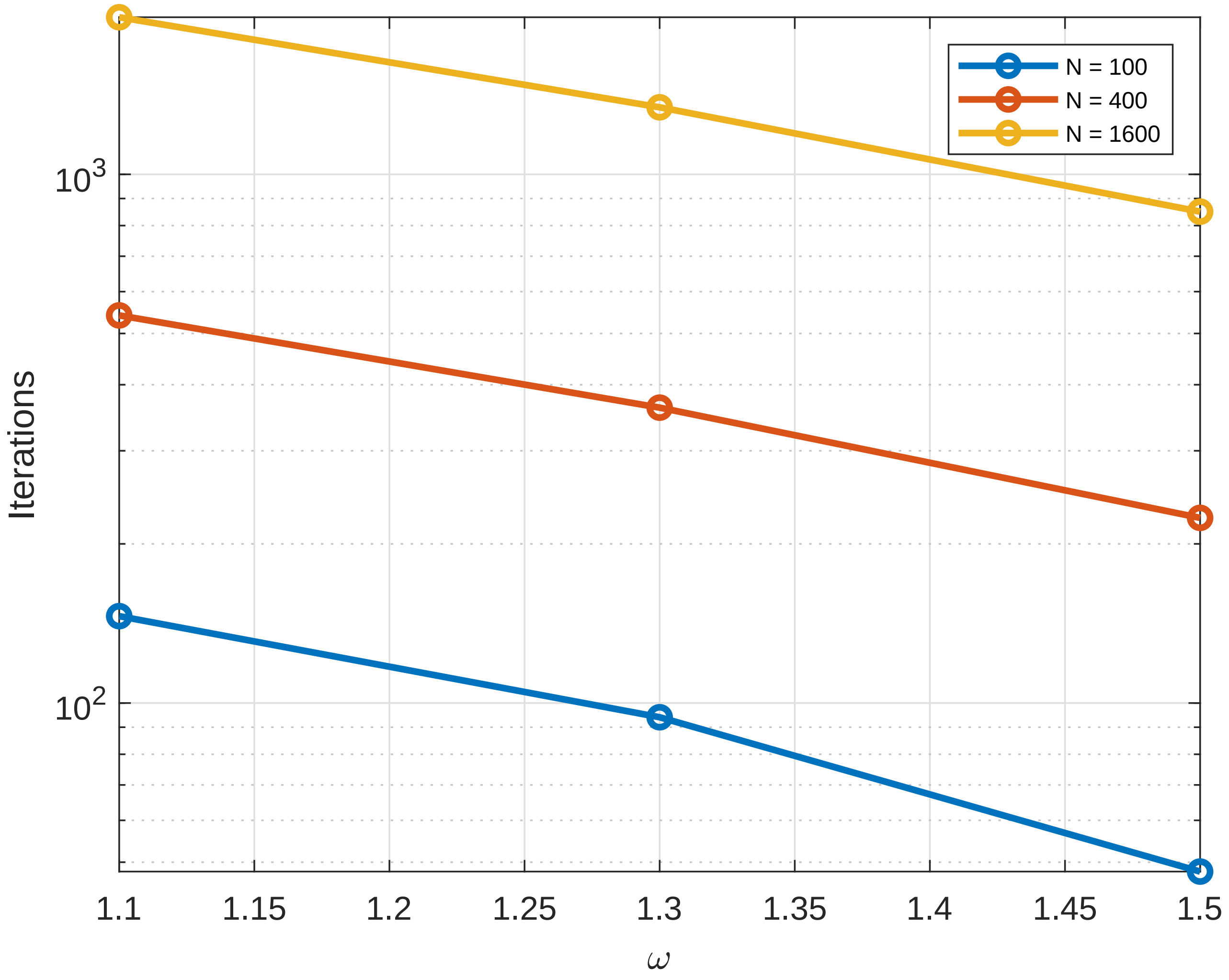


Figure 9: *Overrelaxation Factor* impact on the *SOR* method convergence

3.2.3 CPU Time

As you see in Figure 10 the CPU time[1] of *Gaussian Elimination Method* is significant for large number of CVs. This is due to the very big matrix this method has to deal with.

[1]Intel® Core$^{\text{TM}}$ i7-4790K Processor

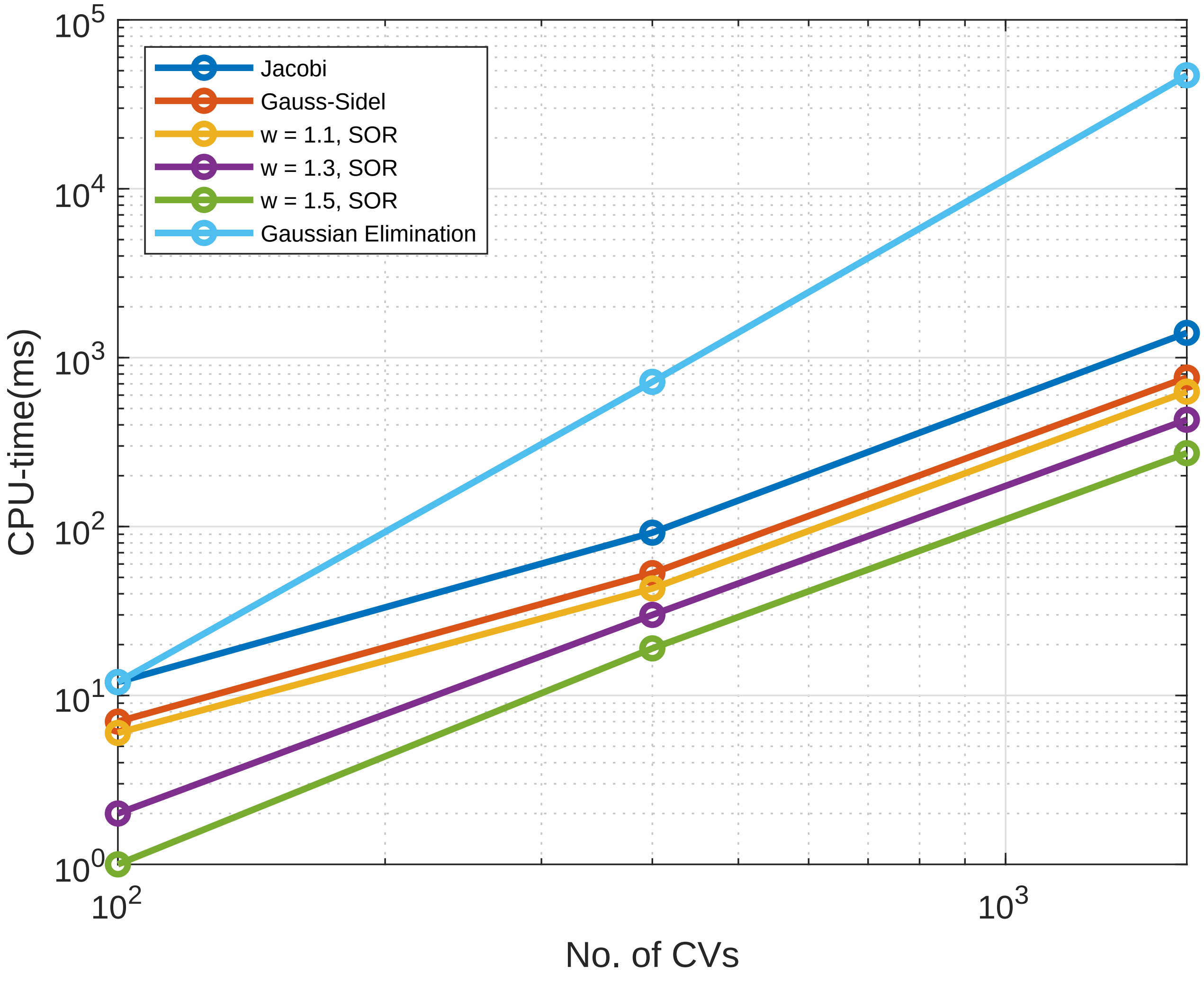


Figure 10: CPU Time of each method convergence

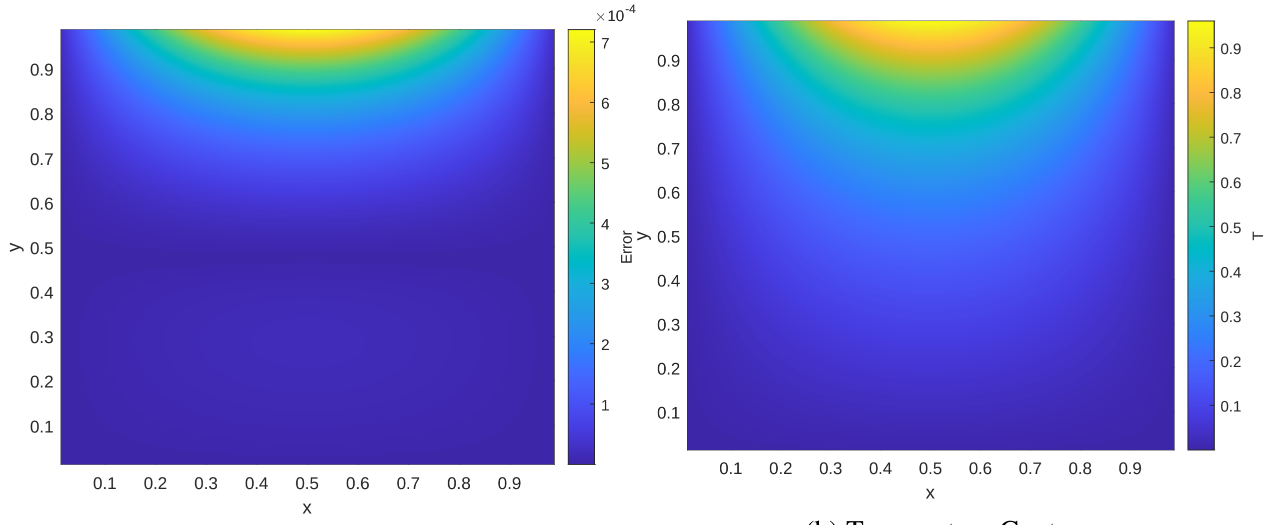
3.2.4 Overall Results

Method	Number of CVs	Overrelaxation Factor	Iterations	L_1	L_2	L_∞	CPU Time (ms)
Gauss-Sidel	100	-	178	0.00124883	0.0024553	0.00909018	7
SOR	100	1.1	146	0.00124883	0.0024553	0.00909018	6
SOR	100	1.3	94	0.00124883	0.0024553	0.00909018	2
SOR	100	1.5	48	0.00124883	0.0024553	0.00909018	1
Jacobi	400	-	1248	0.00032035	0.000638517	0.0026776	92
Gauss-Sidel	400	-	654	0.000320349	0.000638513	0.00267759	53
SOR	400	1.1	541	0.000320349	0.000638512	0.00267759	43
SOR	400	1.3	362	0.000320349	0.000638511	0.00267759	30
SOR	400	1.5	224	0.000320349	0.00063851	0.00267759	19
Jacobi	1600	-	4549	8.07031e-05	0.000161248	0.00072047	1402
Gauss-Sidel	1600	-	2390	8.0698e-05	0.000161232	0.000720465	762
SOR	1600	1.1	1982	8.06973e-05	0.000161229	0.000720464	629
SOR	1600	1.3	1339	8.06961e-05	0.000161224	0.000720462	429
SOR	1600	1.5	850	8.06952e-05	0.000161221	0.000720461	272
Gauss Elimination	100	-	-	0.00124883	0.0024553	0.00909018	12
Gauss Elimination	400	-	-	0.00032035	0.000638509	0.00267759	719
Gauss Elimination	1600	-	-	8.0695e-05	0.000161216	0.00072046	47003

Table 1: Overall Results

3.3 Error and Temperature Contours

In the Figure 11 the Error and Temperature contours for 1600 CVs have been depicted. As one expects in the regions where there are higher temperature gradients, the error increases.



(a) Error Contour

(b) Temperature Contour

Figure 11: Error and Temperature Contours

4 Conclusion

In this project we used a second order approximation and compared the results from four method of solutions. *SOR* is by far the fastest method. Further examinations can be done by increasing or decreasing the *Overrelaxation Factor*, or increasing the order of solution and examination of the L_2 convergence behavior.

Bibliography

[1] A. Nejat and C. Ollivier-Gooch, “A high-order accurate unstructured finite volume newtonkrylov algorithm for inviscid compressible flows,” *Journal of Computational Physics*, vol. 227, pp. 2582–2609, 2 2008.

[2] D. Anderson, J. C. Tannehill, and R. H. Pletcher, *Computational Fluid Mechanics and Heat Transfer*. CRC Press, 4 2016.