MT5846: Advanced Computational Techniques

Project Two

Name: Cameron Sander
Matriculation number: 210005973

Hand in Date: 1^{st} April

Abstract

This paper analyses different numerical methods used to solve the Laplace equation. The primary focus is investigating the following methods: Jacobi, Gauss-Seidel, Line Gauss-Seidel, Sucessive Over-Relaxation (SOR) and Line Successive Over-Relaxation (LSOR). The paper investigates how the relaxation parameter ω affects the rate of convergence for the SOR and LSOR methods, and compares how all the methods' rates of convergences compare to each other. In addition, the impact of altering the initial guess on the solution of two versions of the line Gauss-Seidel method is discussed. The paper finds that the LSOR method converges the fastest when using the numerically calculated optimal ω. However, the SOR method seemed the most appropriate in the context of the problem.

Contents

1 Introduction

In this project, we will be investigating the Laplace equation.

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

For this problem a rectangular domain will be used with a height $H = 2$m and a length of $L = 1$m. The boundary conditions given for this problem are that $T_1 = 100C$ at $y = 0$ and $T = 0$ on all other boundary points. This was used in conjunction with a grid in which $im = 50$ and $jm = 100$ as well as, $\beta = \frac{\Delta x}{\Delta y} = 1$. To solve this PDE numerous methods have been employed these have been outlined below. These all being valid for $1 < i < im, 1 < j < jm$ and $0 < \omega < 2$:

Jacobi Method:

$$u_{i,j}^{k+1} = \frac{1}{2(1+\beta^2)}[u_{i-1,j}^{k} + u_{i+1,j}^{k} + \beta^2(u_{i,j-1}^{k} + u_{j+1}^{k})] \tag{2}$$

Gauss-Seidel Method:

$$u_{i,j}^{k+1} = \frac{1}{2(1+\beta^2)}[u_{i-1,j}^{k+1} + u_{i+1,j}^{k} + \beta^2(u_{i,j-1}^{k+1} + u_{i,j+1}^{k})] \tag{3}$$

Line Gauss-Seidel Method:

$$u_{i-1,j}^{k+1} - 2(1+\beta^2)u_{i,j}^{k+1} + u_{i+1,j}^{k+1} = -\beta^2(u_{i,j-1}^{k+1} + u_{i,j+1}^{k}) \tag{4}$$

Successive Over-Relaxation (SOR):

$$u_{i,j}^{k+1} = (1-\omega)u_{i,j}^{k} + \frac{\omega}{2(1+\beta^2)}[u_{i-1,j}^{k+1} + u_{i+1,j}^{k} + \beta^2(u_{i,j-1}^{k+1} + u_{i,j+1}^{k})] \tag{5}$$

Line Successive Over-Relaxation Method (LSOR):

$$\omega u_{i-1,j}^{k+1} - 2(1+\beta^2)u_{i,j}^{k+1} + \omega u_{i+1,j}^{k+1} = -(1-\omega)[2(1+\beta^2)]u_{i,j}^{k} - \omega\beta^2(u_{i,j-1}^{k+1} + u_{i,j+1}^{k}) \tag{6}$$

Stopping Condition : Since the analytical solution of this problem is unknown in this case a stopping condition has to be employed to detect when the iterations of the methods has converged. For this investigation the stopping condition has been defined below:

$$\sum_{i=1,j=1}^{i=im-1,j=jm-1} (|T_{i,j}^{k+1} - T_{i,j}^{k}|) < \epsilon \tag{7}$$

where $\epsilon = (im-1)(jm-1) \times 1.3 \times 10^{-5}$

An additional stopping condition of a maximum number of iterations of 10,000 was also imposed to allow for if the condition above was not met.

2 Results

Before going through all of the solutions for the questions raised in the project outline, it is important to obtain a holistic view of what the result of using any of these methods achieved.

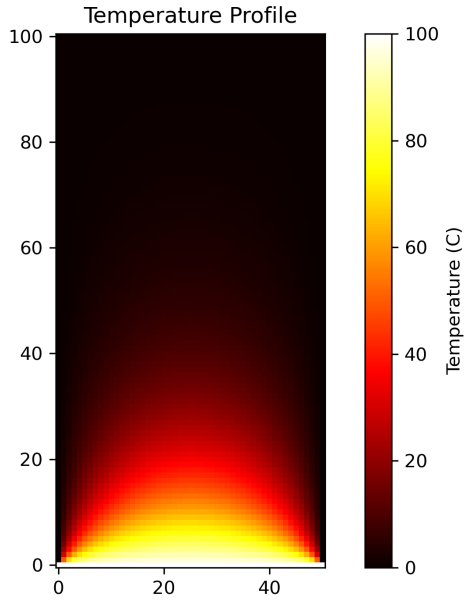

Figure 1: Temperature profile of converged methods

In light of this figure 1 shows the temperature profile of in this case what the Jacobi method has converged to. Visually, this is the same profile that all the other methods converged too, this was used as an important validation tool to ensure that the method were working as predicted and converging to a similar temperature profile. Intuitively, this distribution makes sense since the only source of heat lies on the $y = 0$ boundary. We notice here that a bell shape of temperature bands appear which is what we might intuitively understand the profile to appear as.

In addition to this, below an overlook of the number of iterations needed for each method to converge has been summarised. These have been calculated using the initial guess of $0C$ on all interior grid points. **Note:** The SOR/LSOR methods are using their analytical/numerically calculated optimal ω values respectively, which will be discussed subsequently in the remainder of this results section.

Jacobi	Gauss-Seidel	Line Gauss-Seidel	SOR	LSOR
5240	2878	1570	121	78

Table 1: Summary of number of iterations required for convergence for methods

2.1 Varying omega for SOR method

The values of omega was varied from $1 \leq \omega < 2$. Values of $0 < \omega < 1$ were not tested as this caused under-relaxation which would have increased the number of iterations. We can see from figure 2 that the number of iterations to reach convergence gradually decreases as we approach the optimal ω from the left and then steeply increases once passing this value. The analytically calculated optimal value agrees with our results from our numerical experimentation as it lies near where the fewest number of iterations is required for convergence.

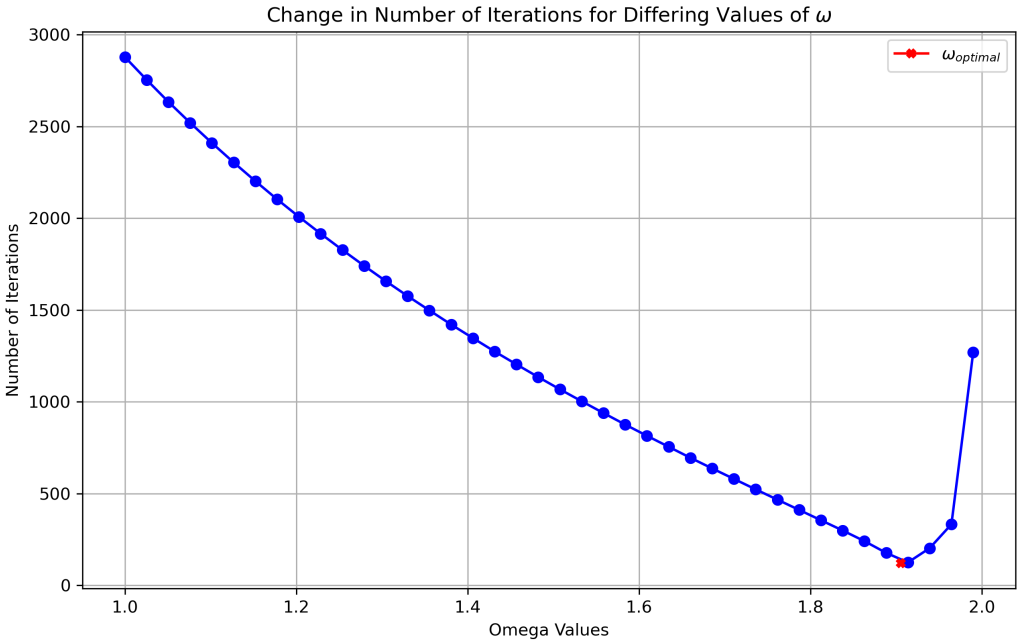

Figure 2: varying ω for SOR Method

In particular, using the grid spacing above we deduce the numerical approximation for the optimal ω for the SOR method to be $\omega = 1.914$ (3.d.p) at which point the number of iterations is **125**. Using the analytical value of $\omega = 1.905$ (3.d.p) we reach convergence in **121** iterations. This small deviation is due to the grid sizing selected to calculate the numerical values of ω. This error could have been reduced by increasing the number of grid points on the given domain but this would have increased computation cost.

Note: In collaboration with another member of the class it was found that two techniques could be used to reduce the time taken for cells to run. Firstly, for methods that required the solution of linear systems. Sparse matrix decomposition was used for preconditioning and sparse matrices solvers were employed as the tri-diagonal systems created allowed the computations to be cut significantly. Also a package *numba* was used, this is a python compiler that is designed to optimize and speed up code execution, it is effective at speeding up the *for* loops used in the code.

2.2 Varying omega for LSOR method

The figure 3 outlines how the number of iterations varies as ω is changed. Again similar to the SOR method values above the optimum caused a significant increase in number of iterations and were excluded from the figure. If the value of ω was raised too far above the optimal value, the errors grew so large that an overflow error occurred which prevented the methods from running. Which is why the grid size has been limited. Similarly the number of iterations for below 1 were excluded too, since the optimum value of ω is known to not lie in that region.

To have a better approximation for the optimum ω value, a second simulation was done with a much smaller range of ω values to try and determine a better approximation for the optimal value. This can be seen in figure 4. The value obtained with this experimentation for the optimal value is $\omega_{optimal} = 1.3068$

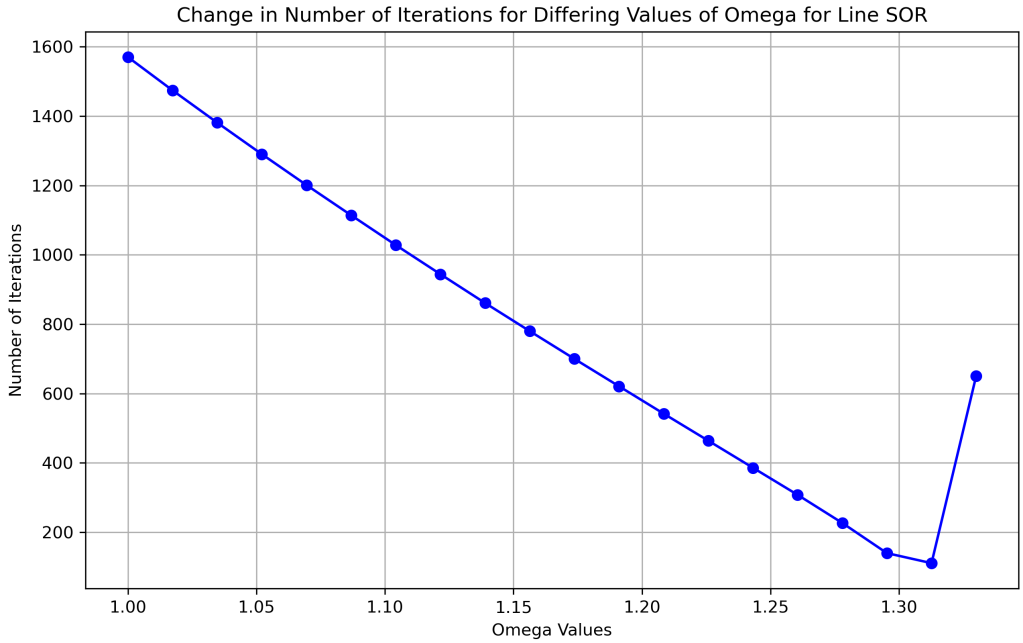

Figure 3: varying ω for LSOR Method

(4.d.p). Which reached convergence in **78** iterations. In this case I don't believe that this technique was useful since the computation time required to find the numerical optimal value of ω far exceeded the time required of calculating any of the other methods, hence defeating the reason this numerical optimum was being calculated for (to speed up computation).

2.2.1 Stochastic Gradient Decent:

As an extension, a new method was investigated called Stochastic Gradient Decent which was employed to attempt to find the optimal ω. This method attempted to minimize the number of iterations the LSOR method required by iteratively moving towards the optimal value using a numerically calculated gradient. The stochastic nature was required since the function as seen in 4 had multiple local minima, so the method attempted to avoid getting stuck in these false global minima by adding a random component to how the values were adjusted. Using this method a result of $\omega_{SGD} = 1.302$ (4.dp) was obtained, which required **94** iterations. Although this was not as accurate as the method described above it was still an interesting aside, to show that there are multiple ways one could approach the problem to find the optimal value.

2.3 New Initial Guess

In this section the initial guess for the interior points was changed to $T = 50C$ for the $k = 0$ iteration. The Line Gauss-Seidel was then used to solve the PDE, using lines of both constant i/j, to compare how this impacted the results. When lines of constant j was used the number of iterations was **1881** and when constant i was used it was **1873**.

The reason for this minor difference is because the gradient in the temperature profile was greater along vertical lines than horizontal lines when using the updated initial guess. The iterative method

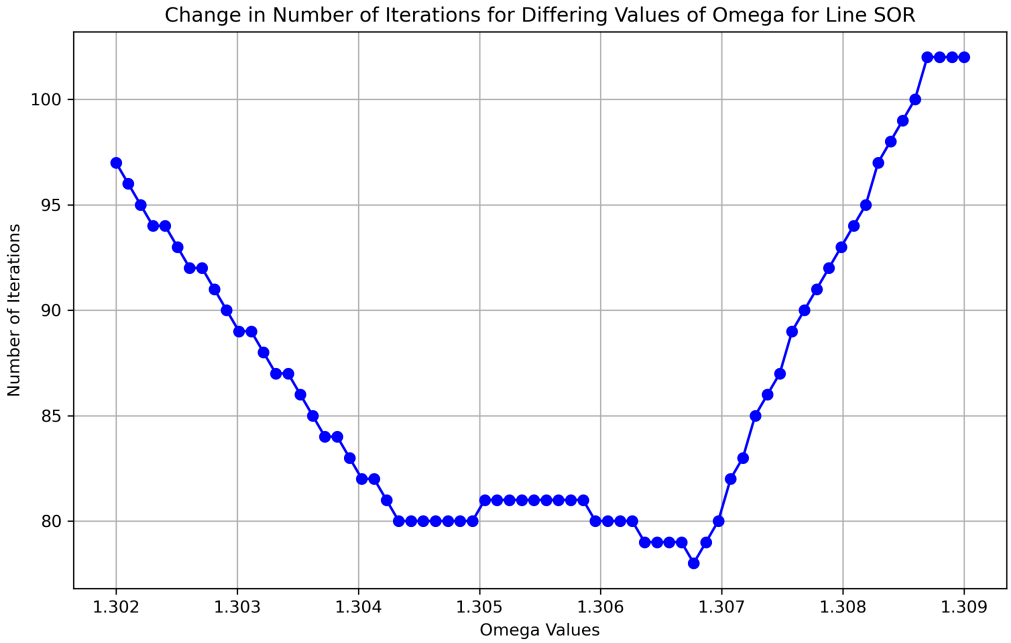

Figure 4: varying ω for LSOR Method

converges faster when the lines it is iterating over have a steeper gradient, hence why the number of iterations required for the line Gauss-Seidel for lines of constant i (vertical lines) converged in a fewer number of iterations than for constant j (horizontal lines).

We compare this to when we used our original guess of $0C$ on all interior points in which case the method required **1570** iterations for line gauss-seidel with constant j and **1592** for constant i. We can see that both methods with the updated guess required more iterations compared to the original setup, this is due to the fact that the updated initial guess lies further from the true equilibrium point and hence requires more iterations to reach the stopping condition.

2.4 Rate of Convergence

Note: In the calculations for this graph, the numerical/analytical optimal ω have been used for the SOR/LSOR methods.

In figure 5 we interpret the gradient of the lines for each method as their rate of convergence. This is because the gradient represents how the sum of residuals decreases, showing how the results of each array become closer to their previous iteration, which in turn shows a convergence to the equilibrium position of the system. Additionally a horizontal dashed line has been added to the graph to show the tolerance required for the methods to be considered converged.

We observe that both the Line SOR and SOR method appear to have a steep but constant rate of convergence over the course of their iterations. This is in contrast to the other methods, which all at first appear to have a steep gradient, showing a rapid convergence to the equilibrium position but as the methods Jacobi, Point Gauss-Seidel and Line Gauss-Seidel residuals fall below e^4 the methods convergence rates diverge significantly, this is showing how as the methods get closer to this equilibrium

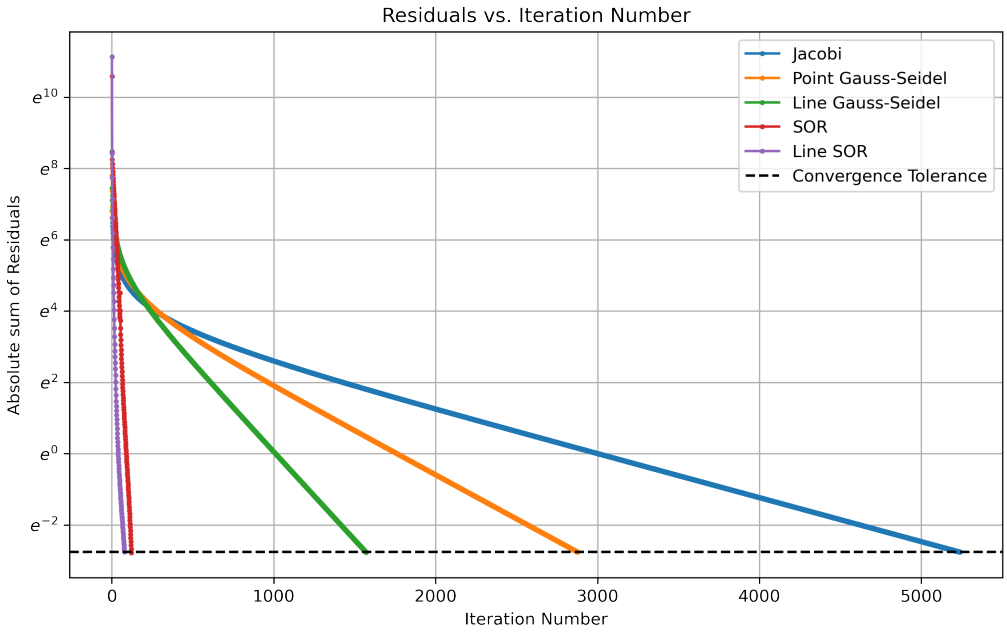

Figure 5: Rate of Convergence for different methods

position, how each of the methods have a different rate at which their successive iterations converge to each other, and thus the equilibrium position. We observe that the Jacobi method converges the slowest (least steep gradient), hence requiring the most iterations. The Point Gauss-Seidel Method converges slightly faster and hence requires fewer iterations, and finally the Line-Gauss converges even faster hence requiring even fewer iterations.

2.4.1 Instability Investigation

It was also investigated what happened to the errors when the stopping condition was made more strict, it was found that the Jacobi, Point Gauss-Seidel and Line Gauss-Seidel, all continued to converge at a similar rate and no unexpected behaviour was observed. However for both the SOR and LSOR methods instability was discovered at different points as seen in figure 6. The instability for LSOR occurred at iterations which are multiple of the number of points in the y-direction (101 points), this is caused since error accumulates due to the nature of the method relying on previous lines to calculate the new iterations, resulting in a jump in error when this accumulated error is at its maximum, this error is then corrected by the method in the next step, hence the error reduces again after this jump. The Point SOR method also shows instability at iterations 51,201. This is suspected to be from the same root cause but creates less of an impact since the error built up is only across the points rather than the lines.

Note: The stopping condition could not have been made more strict than shown here otherwise the methods would not have converged, this is likely due to floating point errors having an impact at this small scale.

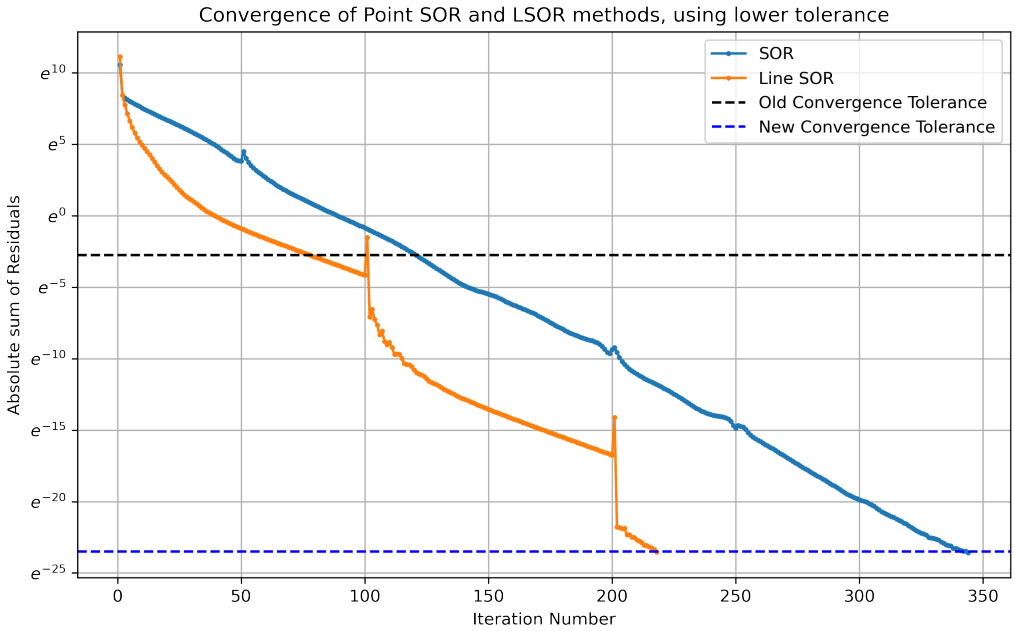

Figure 6: Rate of Convergence for different methods

3 Discussion and Conclusion

From the analysis above we can see how the iterative methods employed affect how the numerical solution converges to the solution of this PDE. Each method has different benefits and drawbacks as we will discuss further here; The Jacobi method, although having the slowest rate of convergence did have the simplest implementation. The point Gauss-Seidel method had a faster rate of covergence but required two loops to iterate over each point in the system for each iteration which increased computational costs. The Line Gauss-Seidel again converged at a faster rate but required a system of equations to be solved for each horizontal/vertical line of points being calculated and hence was again more computationally expensive than the previous methods. The SOR method was significantly faster at converging when using the calculated optimum value of ω, however again required two for loops to iterate over each point in the array, for each iteration. The final method investigated was the LSOR method, which converged at the fastest rate when using the numerically calculated optimal ω value. Again, this required a tri-diagonal system of equations to be solved for each line of the array for each time step, increasing computational cost.

It is important to note that to find this optimal ω parameter for the LSOR method required costly numerical experimentation. Whereas, for the SOR method the optimal ω value could have been calculated directly using the analytical formulae. So although the SOR method, took a few more iterations to converge, it did not require the additional numerical experimentation to find the optimal value for the parameter.

Hence in this context the SOR method seems the most appropriate since, although it required more iterations than the LSOR method. One was able to calculate this optimal ω, unlike the LSOR method. Meaning that overall the time required to compute the method was less than for the LSOR method.