Computational Physics 301 (2018-19)

Exercise 2: Partial Differential Equations

Oliver Sellers

University of Bristol

April 26, 2020

1 Introduction

This computational exercise focuses on solving partial differential equations through the finite difference method. This report documents solving the Laplace equation with the Jacobi and Gauss-Seidel relaxation methods; calculating the potential and electric field for a parallel plate capacitor; and calculating the temperature distribution along a rod thrust into a furnace.

2 Relaxation

Equation 2 from the problem sheet (with $\rho = 0$) was implemented in the code by defining a square grid with a set number of nodes. The Jacobi method could then be applied by creating a new array at the start of each iteration, then comparing it to the previous grid at the end of each iteration to find if the convergence criteria has been met. The convergence criteria throughout this exercise is chosen to be the absolute maximum amount any element can change between iterations. The Gauss-Seidel method was implemented in a similar way but by updating the array for the calculation of each node during an iteration using a checkerboard method. As the Gauss-Seidel is continuously updating, it is expected that it will converge in faster. To test this hypothesis, the program was run with no boundary conditions and a random initial guess for each node. This simulates an empty box and the result should be an array of values close to zero.

Figures 1 and 2, show the results of running simulations of an empty box with the Jacobi and Gauss-Seidel methods averaged over five runs per number of nodes per convergence criteria. In the top plot of figure 1, it can be seen that the error on the output array is not dependent on the total number of nodes. For increasing convergence criteria, the error is proportional.

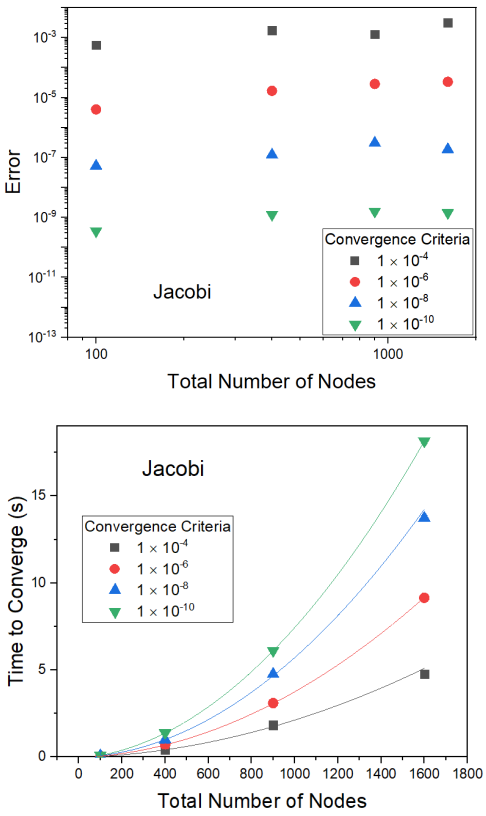

Figure 1: Top: a plot showing the absolute error when comparing to an array of zeros against the total number of nodes for various convergence criteria. Bottom: a plot showing the time taken to converge for a set number of nodes with varying convergence criteria. This data is fitted with a second order polynomial.

This is expected as the degree with which the solution has converged is proportional to the accuracy of the solution. The absolute error is not equal to the convergence criteria in any case but this is not unexpected as the absolute error on the final array and the convergence criteria are not measuring the same thing. The error bars on this data set are not visible. The bottom plot reveals a quadratic relationship between the to-

tal number of nodes and the time to converge for each convergence criteria as the error bars are not visible and the R^2 values are all close to unity showing a good fit. This quadratic relationship is expected as the number of iterations taken to achieve a convergence criteria 10^{-p} for a total number of nodes (N) is roughly $1/2pN$ [1]. Then, the time to perform one iteration is also proportional to N [2], so the total time is proportional to $1/2pN^2$.

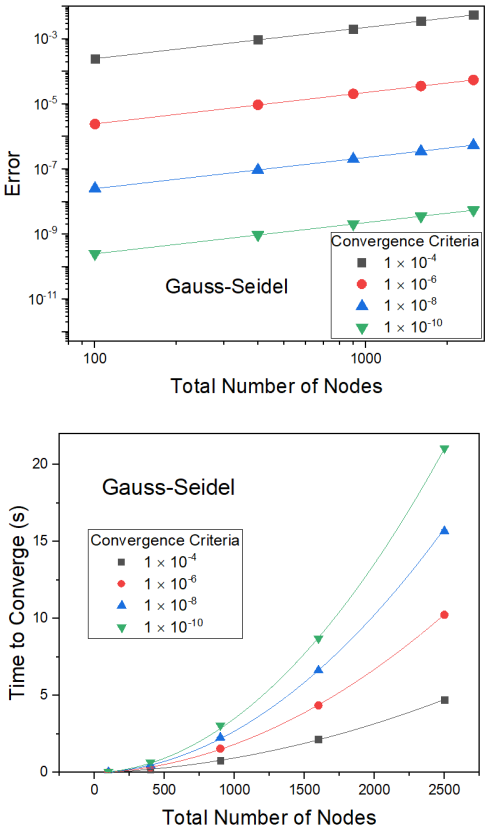

Figure 2: Top: a plot showing the absolute error when comparing to an array of zeros against the total number of nodes for various convergence criteria. This data has been given a linear fit. Bottom: a plot showing the time taken to converge for a set number of nodes with varying convergence criteria. This data is fitted with a second order polynomial.

In the top plot of figure 2, there is a strong linear trend between the total number of nodes and the absolute error on the final array as the error bars are not visible and the R^2 values are close to unity. This is equivalent to decreasing the node separation for the Laplace equation, leading to a denser grid. So grid density must be chosen carefully to ensure an accurate solution. Additionally, increasing the convergence criteria is proportional to the increase in absolute error, the same as the Jacobi. The bottom plot shows the expected quadratic relationship between the total number of nodes and the time to converge for the Gauss-Seidel due to the same argument as for the Jacobi method [1, 2]. The R^2 values are all between 0.999 and 1 for each data set and the error bars are not visible. It can be seen that the Gauss-Seidel roughly takes half the time to converge. This is explained as the iterations taken to achieve a convergence criteria is roughly $1/4pN$ for the Gauss-Seidel method [1]. Therefore, using the same idea as above, the total convergence time is half the Jacobi [2]. Naturally, this is not true for all cases.

The numerical error in solving the finite difference equation is defined by the convergence criteria. The plots show that decreasing the convergence criteria leads to a more accurate result but greater computational intensity. The finite difference method in itself is an approximation and this introduces a truncation error by moving from the integral operator to a finite difference operator. The truncation error tends to zero as the node separation tends to zero but very small separation can also lead to long computing times [3]. The truncation must be chosen to look at a scale where the interesting physics is. Additionally, the approximation assumes that the increments in both the x and y directions are equal. The accuracy of this method involves a trade-off between the computational time and the numerical and truncation errors. A balance must be found that gives sufficiently accurate solutions. Another limitation of iterative methods is can often not converge, the choice of initial guess has no influence on whether convergence will occur. In this case, the equations are simple so convergence can be achieved relatively easily [4].

The Gauss-Seidel method has been shown to be the faster way of solving the Laplace equation in 2-dimensions with the finite difference method. The absolute errors for both the Jacobi and the Gauss-Seidel are similar for the same initial conditions. Therefore, this method was chosen for use in calculating the field and potential of a parallel plate capacitor.

3 Parallel Plate Capacitor

Once the program had been written to solve the Laplace equation for an empty box, the extension to a parallel plate capacitor was relatively straightforward. Boundary conditions had to be implemented for the top and bottom plates to be $+1$ and -1, and the solver function had to be modified to reimpose the boundary conditions on each iteration. To find the electric field, the gradient was found of the converged array using the inbuilt numpy.gradient function.

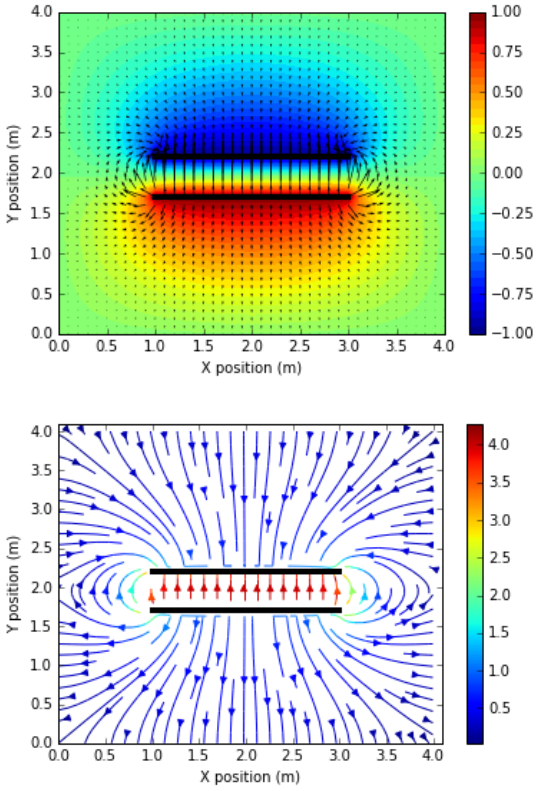

Figure 3: Plots showing the results of a simulation of a parallel plate capacitor with a plate length (a) of 2 m, and separation (d) of 0.5 m, and a convergence criteria of 1×10^{-6}. Top: a contour plot of the electric potential (V). Bottom: a stream plot of the gradient of the potential (V/m).

Figure 3 shows plots for a parallel plate capacitor with finite extent. The top plot shows the potential and direction in and around the plates with a contour plot and a quiver plot superimposed. This shows the potential around the source ($+1V$) and the sink ($-1V$) is evenly spread and there is uniform flow from the source to the sink. The bottom plot shows the gradient of the electric potential (electric field magnitude) in and around the capacitor with a stream plot. It can be seen the field between the plates is close to 4, which is the infinite plate solution (V/d) for this capacitor but this is not completely uniform. The field outside is close to zero at points but becoming non-zero at points near the ends of the capacitor. The field can be seen to tend to zero with distance from the capacitor, as is expected.

Figure 6, in the appendix, shows the results of a simulation of a parallel plate capacitor with a much larger ratio of capacitor plate length (a) to separation (d). The plate separation in this case was 0.2 m, the smallest possible for an node spacing of 0.1 m. The top plot shows that the field between the capacitor plates is much more uniform and roughly equal to 10, the infinite plate solution for this capacitor. The field outside the plates appears closer to zero than the previous simulation. It also tends towards zero with distance from the capacitor. This is supported by the bottom plot showing a stream plot of the gradient of the potential field. The arrows within the plates can no longer be seen. The arrows outside show the field is more uniform than previously, apart from at the end of the capacitor. Thus, the capacitor has approached the infinite plate solution for increased a/d. If the ends were to be removed completely, the field would be constant outside the capacitor.

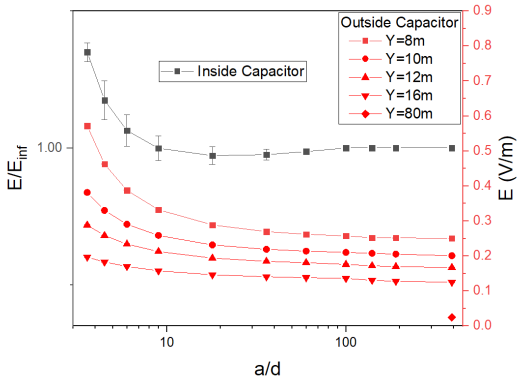

Figure 4: A plot showing the average field as the ratio a/d becomes large. The grey squares represent the field inside the capacitor compared to the infinite plate solution. The red symbols represent the field outside of the capacitor. The error bars are the variation in the output arrays.

To further explore the approach to the infinite solution, the capacitor from figure 6 was used as a baseline while the length of the plates and separation were varied. The X size was such that there was 1 m of free space either side of the plates. It can be seen in figure 4 that the field (E) inside the capacitor quickly approaches the infinite plate solution (E_{inf}). The error bars are larger for smaller a/d, showing greater variation in the output array. It can be seen that the field outside the capacitor decreases for increasing a/d but this is dependent on the Y size of the input array. As Y increases, the field outside the capacitor approaches zero. Only one model was run for an 80×80 as it was computationally intensive. The plot shows that this value is closest to zero which can be explained by the result being averaged over values further from the capacitor plates. These values will have a field closer to zero. Therefore, if a square grid is input with increasing size, it is anticipated that the infinite solution will be further approached.

4 Diffusion

In the final part of this exercise, the finite difference method was applied to heat diffusion using equation 3 from the problem sheet in matrix form. To implement in the code, the coefficients matrix was initialised and the inverse was computed before the solver was called. By computing the inverse before the solver function, the optimised numpy.dot matrix multiplication method can be used. To confirm the solver was working, a function was built to test the solver with heat sinks at 0°C at each end of the poker so the temperature along the rod should go to zero within a certain acceptance range of the convergence criteria.

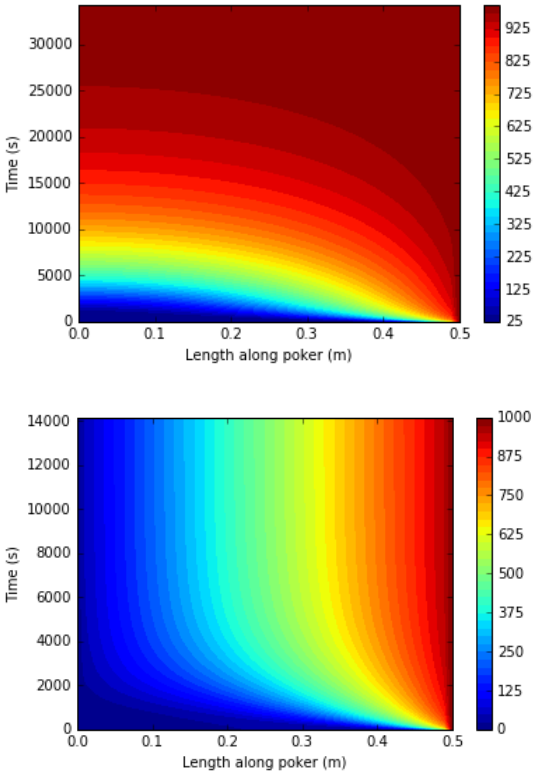

Figure 5: Plots showing the temperature distribution (°C) along the poker as a function of time. A time step of 10 s was used with a node separation of 1 cm. Top: no heat loss from the poker with a convergence criteria of 1×10^{-2}. Bottom: far end of the poker in an ice bath with a convergence criteria of 1×10^{-3}.

Figure 5 shows the results of running the heat diffusion solver for the two scenarios presented in the problem sheet. The top plot shows the heat distribution for no heat loss converges on at 1000 °C at around 30,000 s. This is what was expected as there is no heat loss so the total temperature distribution will eventually converge at the heat source temperature. The bottom plot shows the ice condition has converged on a distribution from 0 to 1000 °C. This is expected as well as the heat source and sink will come into equilibrium. The second scenario reaches equilibrium in less than half the time to the first scenario. This can be explained as the average amount each node has to change from the initial conditions in the second scenario is considerably less. The errors introduced through the modelling method are the same as discussed in section 2 as the formulation of the finite difference equations are very similar for the Laplace and heat diffusion equations. The truncation error results from the node separation, and the numerical error from the convergence criteria. The time step chooses the time period that is to be looked at. These had to be chosen to observe the required physics, in this case the rod coming into equilibrium.

Finite difference and other numerical methods for solving partial differential equations are of great importance in science. They allow accurate models to be made for all types of physical situations, some even where the governing equations cannot be solved analytically. Models for electric fields and heat diffusion considered here allow designing systems with much greater accuracy than could be done by hand. The models developed here can also be improved in the future: alternative numerical methods could be considered to deal with more complex meshing scenarios, such as the finite volume or element method. Furthermore, alternative relaxation methods could be implement; the Gauss-Seidel successive over-relaxation method leads to faster convergence, allowing larger arrays to be input [3].

References

[1] W. H. Press, *Numerical Recipes in C*, 2nd Edition, Cambridge University Press, Cambridge (1992).

[2] C. H. Rycroft, *Iterative Methods for Linear Systems*, School of Engineering and Applied Sciences, Harvard University, Cambridge, (2007).

[3] D. M. Causon, C. G. Mingham, *Introductory Finite Difference Methods for PDEs*, Ventus Publishing ApS, (2010).

[4] J. Kiusalaas, *Numerical Methods in Engineering with Python*, 1st Edition, Cambridge University Press, New York (2005).

Appendices

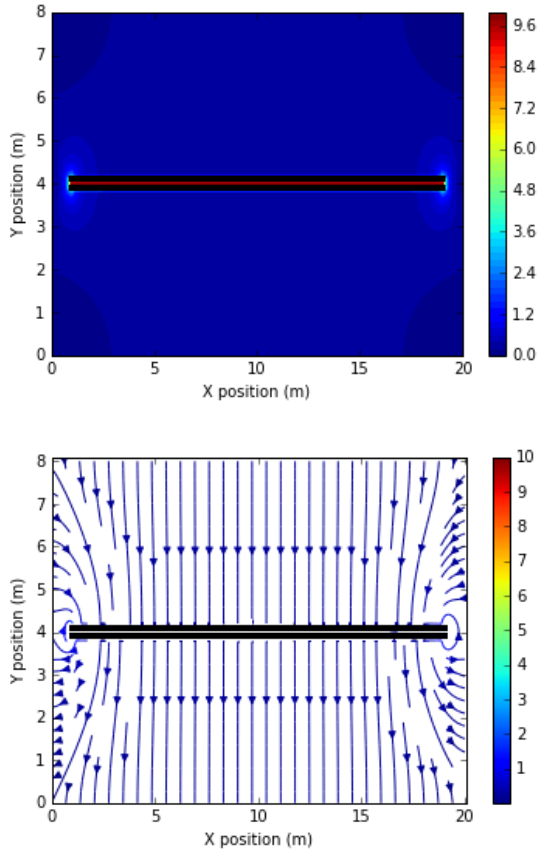

Figure 6: Plots showing the results of a simulation of a parallel plate capacitor with a plate length (a) of 18 m, and separation (d) of 0.2 m, and a convergence criteria of 1×10^{-6}. Top: a contour plot of the electric field (V/m). Bottom: a stream plot of the gradient of the potential (V/m).