Computational Physics 301: Exercise 2 - Partial Differential Equations

Caspar Fuller - 1310006 *6th March 2015*

A partial differential equation is a function of multiple independent variables and their partial derivatives. The theory of partial differential equations (PDE) is a fundamental area of mathematics leading to the formulation of many of the foundations of classical physics such as Maxwell's equations. Only limited specific PDEs have a complete analytical solution so approximation of solutions via numerical algorithms is required for a wider understanding of PDEs. This report aims to implement these numerical algorithms to solve various partial differential equations, specifically Laplace's equations and the diffusion equation. Each program was written in the C language and each numerical algorithm was analysed to find it's limits and understand errors that arise from implementation.

Problem 1: Solving Laplace's equation

Theory and Computation

The first task was to solve Laplace's equation, $\nabla^2 V = 0$, for a two dimensional system of charges where V is the potential. If well-defined boundary conditions are imposed on this second order differential equation the iterative technique of relaxation can be employed to approximate a solution. This is carried out on a grid of nodes with a spacing of h between nodes. The second derivative of Laplace's equation can therefore be approximated as equation (1) through Taylor expansion around an arbitrary point x_i. This is known as the finite difference approximation.

$$\frac{d^2V(x_i)}{dx^2} = \frac{V(x_{i-1}) + V(x_{i+1}) - 2V(x_i)}{h^2} \qquad (1)$$

This can be generalised through rearrangement and substitution to two dimensions to give a final approximation of Laplace's equation seen in equation (2). Essentially this equation states the new value of each node is the mean of its four nearest neighbours.

$$V(x_i, y_j) = \frac{1}{4}(V(x_{i-1}, y_j) + V(x_{i+1}, y_j) + V(x_i, y_{j-1}) + V(x_i, y_{j+1})) \qquad (2)$$

Equation (2) can be solved using various iterative numerical techniques, for this task the Jacobi method was utilised. This calculates the new nodes of each grid points then replaces the old solution with the new solution and iterates again until a convergence condition is met. This is known as a relaxation technique as an initial guess of the solution is made and the system is allowed to 'relax' towards the true solution.

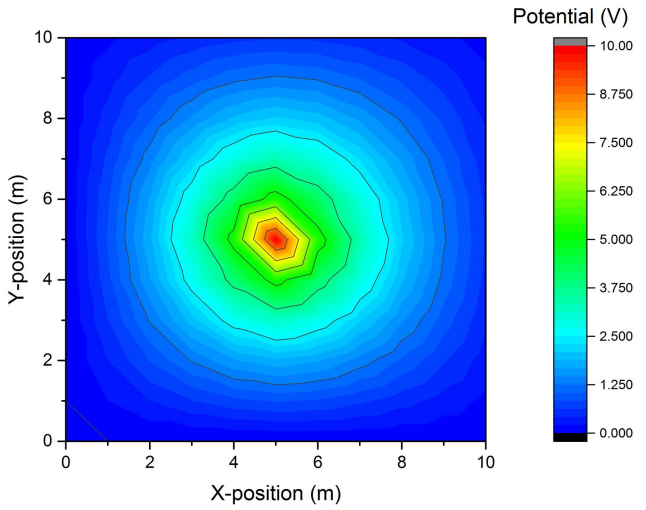

Figure 1: Plot of node values for a 10V point charge in a 10x10 grid with an initial guess of 0 for all other nodes and a convergence condition of 0.0000001. The boundaries were fixed at 0V

Computationally this was a straightforward procedure to set up. Care was taken to ensure the dynamic arrays initialised satisfied the required grid density, (the nodal spacing) h. Once the node boundaries are fixed and point charges placed, equation (2) is implemented iteratively until a convergence condition is met. This condition was that every node value is not changed by more than a specified percentage from the previous value. During initial testing this condition was set as a constant value such as a change of 0.001 however it was found if large values were inputted the system would never converge, meaning the percentage condition is necessary to handle inputs of any size. If point charges were placed it was essential they remained constant to ensure an accurate approximation. By initialising an integer array to store the position of the charge checks could be made to ensure the value of the charge is never altered. Efforts were made to make analysis of this method simple for a user so functions were put in to produce text files of certain tests on the code automatically without the need of extensive user operation.

Results and Discussion

The code was initially tested for a simple point charge model, where the boundary conditions on the grid were set at 0. Figure (1) shows a plot of the final node values once the convergence condition of 0.0000001

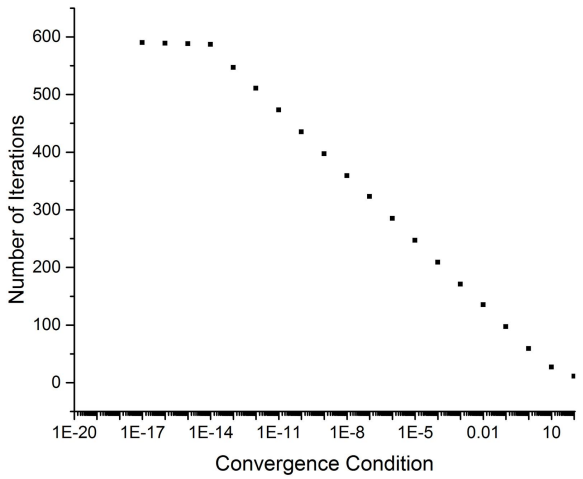

Figure 2: Plot of number of iterations needed for convergence for varying convergence conditions. This plot has a logarithmic x-axis scale. Setup was the same as figure (1).

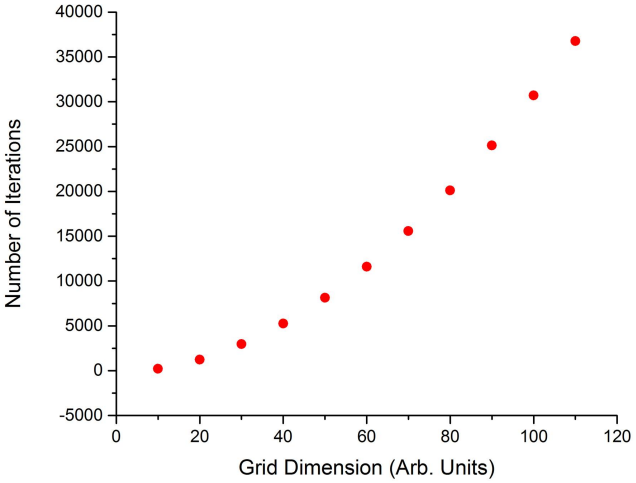

Figure 3: Plot of number of iterations needed for convergence for varying grid dimensions. Setup was the same as figure (1).
https://preview.overleaf.com/public/ysdcznsbbmvy/images/4dde1c

was met. This plot makes physical sense although circular contour lines would be expected which is not seen. This is simply an artifact of the grid dimension's being set at 10x10 leaving only 100 node values to be plotted. The code was then tested by varying different variables and investigating the codes sensitivity to this.

Sensitivity is found when investigating how the convergence condition affects the number of iterations required to reach convergence as shown in figure (2). Between convergence conditions of $10^{-14}\%$ and 10% the number of iterations scales negatively proportionally to the natural logarithm of the convergence condition. At smaller conditions than $10^{-14}\%$ the iteration number flattens out. This is not physically correct and is the result of the machines limit on precision. Double precision floating point numbers have a precision of 15 significant figures[1]. This limit causes the program to terminate as soon as the convergence is at or below this limit regardless of the set convergence condition.

Figure (3) shows the relationship between the number of iterations required for convergence as the grid size is changed. This has a parabolic relationship showing the iterations scale with N^2 where N is the grid dimension. This is expected as this program is for a 2 dimensional system. It would therefore by expected that for a 3 dimensional system this scaling is proportional to N^3. This highlights a big flaw in the Jacobi method as at higher dimensions the number of iterations increases very rapidly causing the convergence to become very slow and inefficient. Another technique would be required to resolve this issue as discussed below.

The Gauss-Seidel method is an alternative choice, here instead of the node values being copied and swapped at the end of each iteration they are continually replaced as the iterations progress. This has the advantage of memory efficient as the Jacobi requires 2 arrays to store the old and new values whereas Gauss-Seidel only functions with one[2]. The Gauss-Seidel method also is found to converge at a quicker pace. The rate of convergence of the Jacobi method is shown in equation (3) where r is the number of iterations needed to reduce the overall error by a factor of 10^{-p}, N is the grid dimension and ρ_s is the spectral radius, the modulus of the largest eigenvalue of the iteration matrix. This is proof of the scaling of the number of iterations with N^2 as stated earlier. The Gauss-Seidel method has a spectral radius equal to the square of the spectral radius for the Jacobi method giving a factor of two improvement on the number of iterations needed to reduce the error by the same amount.

$$r \approx \frac{pln10}{(-ln\rho_s)} \simeq \frac{1}{2}pJ^2 \tag{3}$$

Even more efficient algorithms exist and to improve this program to handle larger dimensions one of the following algorithms would be implemented. One such algorithm is successive overrelaxation (SOR), here instead of iterating via equation (2) each node is updated using equation (4) where σ is an overcorrec-

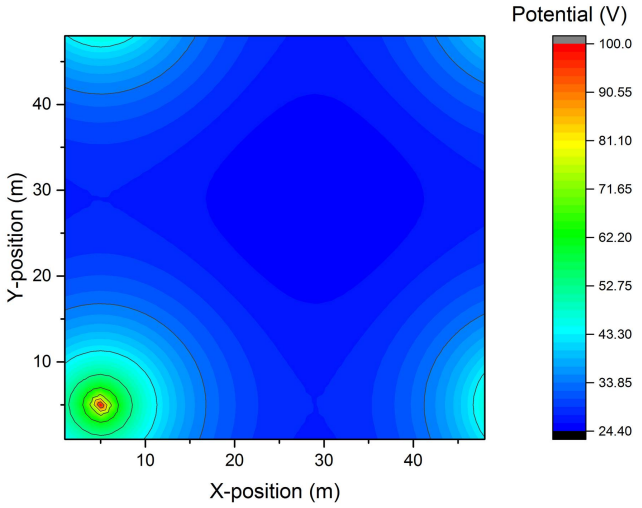

Figure 4: Plot of node values for a 100V point charge in a 50x50 grid with an initial guess of 0 for all other nodes and a convergence condition of 0.1. Born-von Karman boundary conditions were added to model a torus shape.

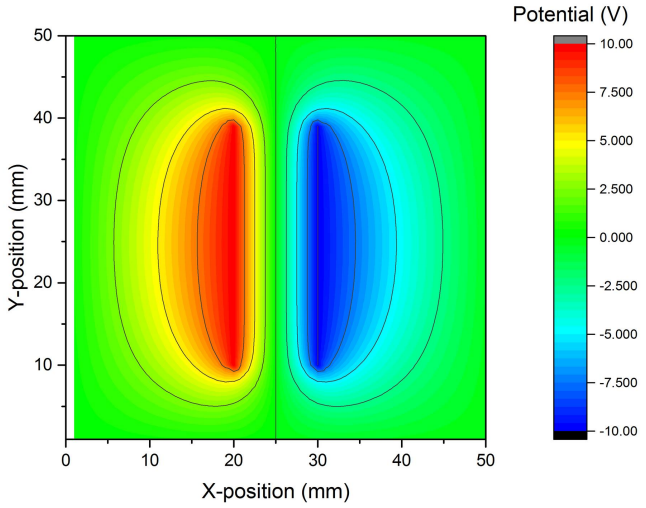

Figure 5: Plot of the potential around a parallel plate capacitor. The plates have potentials 10V and -10V and have a 10mm separation. Convergence condition was set at 0.0000001.

tion parameter.

$$V'(x_i, y_j) = (1-\sigma)V(x_i, y_j) + \sigma(\frac{1}{4}(V(x_{i-1}, y_j) + V(x_{i+1}, y_j) + V(x_i, y_{j-1}) + V(x_i, y_{j+1})) \quad (4)$$

SOR converges far quicker than both the Jacobi and the Gauss-Seidel method so is a good choice for larger systems. The number of iterations,r, needed to reduce the error by 10^{-p} is given by equation (5) where N is grid dimension again. This scaling of order N is far more efficient than the previous scaling of order N^2. Its biggest failing however is over the choice of σ as the optimal value can only rarely be found analytically and will often vary as the iterations progress. If σ is not set correctly the advantages over the other methods disappear as convergence time increases again.

$$r \simeq \frac{1}{3}pN \quad (5)$$

Finally the program was extended to model a torus instead of a flat 2-dimensional grid. This was implemented through the use of Born-von Karman periodic boundary conditions so the boundaries effectively 'ran onto' the opposite side. Figure (4) shows the results of this when a charge is placed in the corner of the grid and the results look physically accurate. This extension could be applied to topology and help understand solutions of PDE's on deformed surfaces[3].

Problem 2: Parallel Plate Capacitor

Theory and Computation

The program written in problem 1 was extended to model the potential and electric field within and around a parallel plate capacitor. The capacitor was assumed to be in a vacuum and 1-dimensional so Laplace's equation could again be solved for the approximation. There was little extension needed computationally. The initial conditions were set up to model parallel plates at differing constant potentials. Numerical analysis was used to find the electric potential, $E = -\nabla V$, in the system using the 5-point stencil method[4].

Results and Discussion

The program was tested with a centered capacitor of 30mm length with a potential difference of 20V put across them at a separation of 10mm. Plots of the potential and electric field surrounding the capacitor were made, shown in figure (5) and figure (6) respectively. These figures correspond to the prediction of what the field should look like. A relatively smooth uniform electric field is seen between the plates as is expected. The fringe fields on the edges of the capacitor do however influence the potential and hence electric field along the length of the plates shown by the curved contour lines seen. Theoretically this effect should be negated as the configuration approaches the 'infinite' plate solution where the potential and electric fields can be described by equation (6). This can be investigated by increasing the value of $\frac{a}{d}$ where a

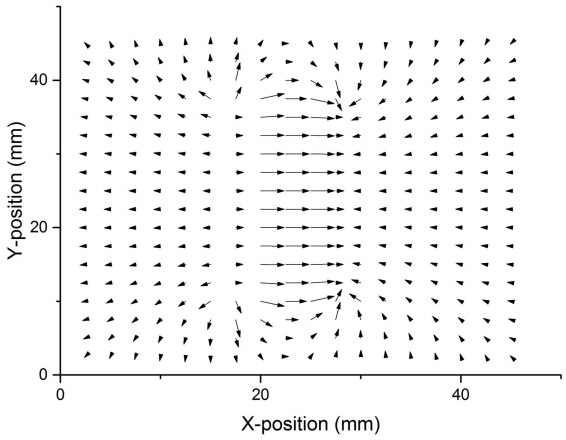

Figure 6: Vector plot of the electric field around a parallel plate capacitor.The plates have potentials 10V and -10V and have a 10mm separation. Convergence condition was set at 0.0000001.

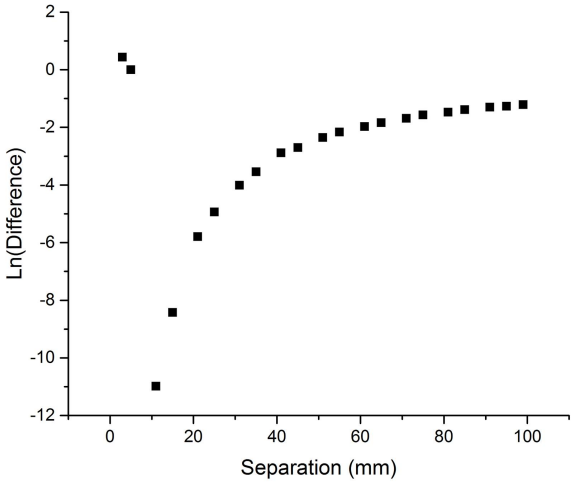

Figure 7: Plot of how the difference between the electric field value at a central node between the capacitor and the electric field calculated from $E = \frac{V}{d}$ varies with capacitor separation. The X-axis scales with the logarithm of the separation distance. The potential difference across plates was 60V and the length of the capacitor was 100mm.

is the capacitor plate length and d is the separation of the plates and recording if the model approaches the value given by equation (6).

$$E = \frac{V}{d} \tag{6}$$

Figure (7) shows the plot of varying separation distances and the corresponding difference between analytical and simulated results for a capacitor of length 100mm. As expected as the value of $\frac{a}{d}$ increases the model approaches the correct infinite plane solution. At very close separation the machine precision caused the program to malfunction leaving the two anomalies seen.

Problem 3: Diffusion

Theory and Computation

The final task was to solve the diffusion equation, equation (7), in 1 dimension to simulate an iron poker being placed in a furnace, where ϕ is temperature and t is time. α is thermal diffusivity, a constant derived from the materials thermodynamic properties and the materials density.

$$\alpha \nabla^2 \phi = \frac{\partial \phi}{\partial t} \tag{7}$$

Once again finite differencing techniques can be used to approximate a solution with the node values propagating as time progresses. The standard finite difference approximation for the second differential results in a form known as the forward time method however this is unstable for the initial conditions of the task[5]. Instead the backward-time form is used shown in equation (8) where $\phi'(x_i)$ is the updated temperature at x_i at the next time step, h is grid spacing as before, α is the thermal diffusivity and ΔT is the time step.

$$\frac{\phi'(x_i) - \phi(x_i)}{\Delta t} = \frac{\alpha}{h^2}[\phi'(x_{i-1}) + \phi'(x_{i+1}) - 2\phi'(x_i)] \tag{8}$$

To solve the set of equations equation (8) generates it is appropriate to write it in matrix-vector form and exploit the properties of the this form to provide a quick and efficient solution. This is shown in equation (9), where $\mathbf{A}$ is a matrix of constants dependent on the system boundary conditions.

$$\mathbf{A} \begin{pmatrix} \vdots \\ \phi'(x_{i-1}) \\ \phi'(x_i) \\ \phi'(x_{i+1}) \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ \\ \phi(x_i) \\ \\ \vdots \end{pmatrix} \tag{9}$$

Initially models would be made of the iron poker placed with one end in the furnace at 1273.15K with the rest of the poker initially at 293.15K. Heat losses were ignored along the length of the poker and the evolution of the temperature distribution along the poker was investigated. Ignoring heat loss results in a constant heat distribution at an arbitrary cross section of the poker making it valid to model this task in 1 dimension. Rearranging equation (8) to group like terms allows for clear formulation of the tridiagonal matrix, $\mathbf{A}$. Equation (10) shows this matrix for the first task, however due to the Dirichlet boundary conditions, the end node must remain at 1273.15K there

is a need to change certain matrix elements. The element b_1 is set as $b_1 = 1 + \frac{\alpha\Delta t}{h^2}$ to account for the fact the node furthest from the furnace only has one adjacent node. The elements a_N and b_N are set to $a_N = 0$ and $b_N = 1$. This is due to the corresponding node representing where the poker is in the furnace and is therefore held at constant temperature.

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & a_{N-1} & b_{N-1} & c_{N-1} \\ 0 & 0 & 0 & 0 & a_N & b_N \end{pmatrix} \tag{10}$$

Where:

$$_i = -\frac{\alpha\Delta t}{h^2}, b_i = 1 + 2\frac{\alpha\Delta t}{h^2}, c_i = -\frac{\alpha\Delta t}{h^2}, i = 2, 3, ..., N-1 \tag{11}$$

The program was extended to model the system when the end furthest from furnace was held in a block of ice. This new constant boundary condition requires the matrix **A** to be slightly reconfigured. Here b_1 is set to1 and c_1 is set to 0 to hold the temperature of the poker at the ice end at a constant temperature of 273.15K.

Equation (9) can be solved using the LU Decomposition functions from the GNU Scientific Library (GSL). The matrix of constants, **A**, was first decomposed then equation (9) was solved giving updated temperatures, $\phi'(x_i)$, at each node. By overwriting the old temperatures with the updated ones it was possible to iterate this process over a time step to investigate the change in temperature distribution. This occurred until a specified convergence condition was met as before.

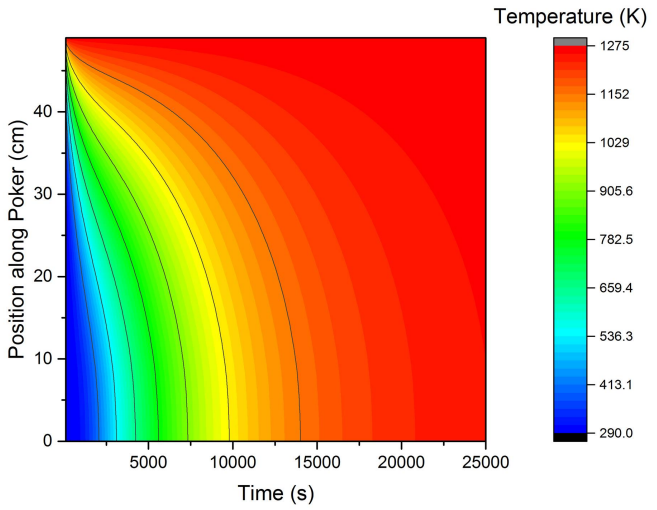

Figure 8: Plot showing the time evolution of the temperature distribution along a poker of length 0.5m with one end in a furnace of temperature 1273.15K. Modeled as 50 nodes with a convergence condition of 0.00001 and a time step of 100s. This figure has been cut short to emphasize temperature evolution. Convergence condition was met after 44600 seconds.

Results and Discussion

Figure (8) shows the heat distribution along the poker as time evolved for the no ice case with a time step of 100 seconds. This plot was found to be an accurate representation of the task, with the expected rate of heat flow seen. The plot shown does not show the equilibrium position of the program as the rate of heat flow is proportional to the temperature gradient. As the poker heats the temperature gradient decreases causing a slower rate of heat flow. This equilibrium occurred at 44600 seconds for a convergence condition of 0.00001.

The case when the other poker end was situated in ice is represented in figure (9). Here the convergence condition of 0.00001 was met after 13200 seconds. Once again we see the proportional relationship between rate of heat flow and temperature gradient causing dramatic temperature changes initially that slow down as the poker heats. A clear equilibrium position is seen in which a constant temperature gradient is found. This causes a linear relationship for the temperature distribution along the poker. The assumption of ignoring heat loss along the length of the poker was then investigated. Obviously this assumption is not true as a red hot poker is clear evidence of radiative heat loss so attempts were made to include a rough approximation to include this heat loss in the results. The poker was modeled as a 50cm cylinder with a radius of 1cm however a constant heat distribution within the poker was still assumed to allow for a 1-dimensional grid to be used for modeling.

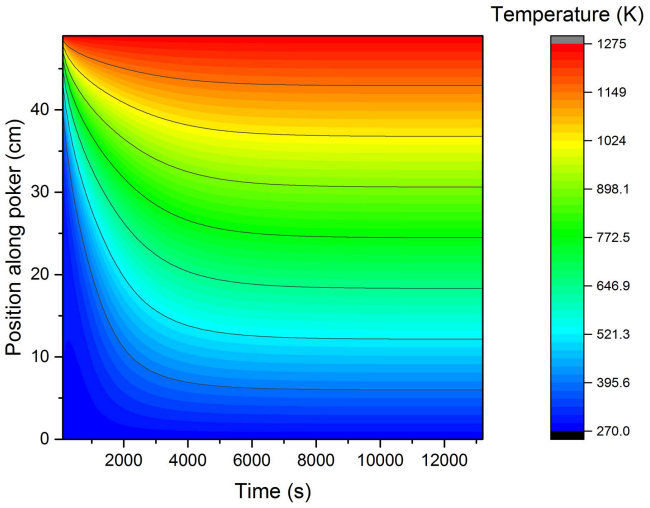

Figure 9: Plot showing the time evolution of the temperature distribution along a poker of length 0.5m with one end in a furnace of temperature 1273.15K and the other in ice (temperature is 273.15K). Modeled as 50 nodes with a convergence condition of 0.00001 and a time step of 100s. Convergence condition was met after 13200 seconds.

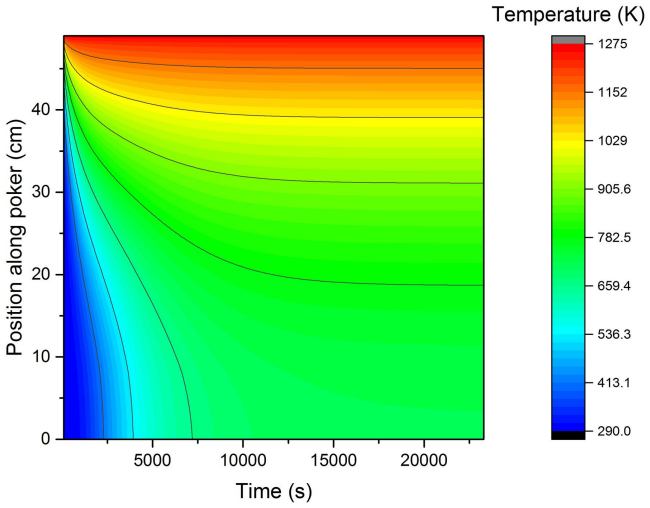

Figure 10: Plot showing the time evolution of the temperature distribution along a poker of length 0.5m with one end in a furnace of temperature 1273.15K. Modeled as 50 nodes with a convergence condition of 0.00001 and a time step of 100s. Rudimentary adjustments have been made to approximate heat loss along the poker.

The temperature difference, ΔT, at each node sector due to heat loss along the rod is shown in equation (12) where, α is the heat transfer coefficient approximated at 80[6], A is the heat transfer surface area which is the surface area of the cylinder between two nodes, T_{poker} is the temperature of the poker at each node calulated as before, T_{air} was set at a constant 293.15K, ϵ is the emissivity of the poker taken as 0.5 for a wrought iron unpolished poker[7], σ is the Stefan-Boltzmann constant and finally c_p is the specific heat capacity of iron. This first term in the numerator represents the rate of heat transfer due to convection as the heat is transferred from a solid to a fluid. The second term is the rate of transfer of radiative energy between two objects, a derivative of the Stefan-Boltzmann equation. The overall equation is found from the definition of specific heat capacity $C = \frac{Q}{\Delta T}$ multiplied by the time step as the rate of transfer has units of joules per second. equation (12) is applied to the vector elements of the updated temperatures every iteration. Although this is a crude approximation figure (9) shows encouraging results.

$$\Delta T = dt\frac{\alpha A(T_{poker} - T_{air}) + \epsilon\sigma A(T_{poker}^4 - T_{air}^4)}{c_p} \tag{12}$$

The temperature distribution initially progress in a similar fashion as previously found however when the entire poker begins to heat a different temperature distribution is found to evolve. The nodes further from the furnace are limited as to what temperature they can reach. An extension to this model would be to configure the program to solve for the 3-dimensional diffusion equation so the heat loss along the poker could be accurately quantified.

A further improvement can be made by replacing the backward-time form of finite differencing with the Crank-Nicolson method. This combines the forward and backward time methods to give a resultant scheme that has the unconditional stability of the backward method aswell as the second-order accuracy of the forward method in both time and space[8]. This second-order accuracy is desired as it requires fewer grid points, thus less computational time to obtain a solution within a given error bound.

[1] Harrington, R. F. "Introduction to Electromagnetic Engineering" *Dover Publications* 2003.

[2] Press, William H. "Numerical recipes 3rd edition: The art of scientific computing." *Cambridge university press* 2007.

[3] Buchstaber, Victor M., and Taras E. Panov. "Torus actions and their applications in topology and combinatorics." American Mathematical Soc. No.**24**. 2002.

[4] Axelsson, O., and V. Eijkhout. "Analysis of a recursive 5-point/9-point factorization method." Preconditioned Conjugate Gradient Methods. *Springer Berlin Heidelberg* 1990.

[5] Hughes, T. J. R., and W. K. Liu. "Implicit-explicit finite elements in transient analysis: stability theory." Journal of applied Mechanics **45.2** 1978.

[6] Whitelaw, Jim H. "Convective Heat Transfer" *Thermopedia - http://www.thermopedia.com/content/660/* Date Accessed: 06/03/2016.

[7] Haynes, William M., ed. "CRC handbook of chemistry and physics." *CRC press* 2014.

[8] Crank, John, and Phyllis Nicolson. "A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type." Mathematical Proceedings of the Cambridge Philosophical Society. Vol. **43** *Cambridge University Press* 1947.