Emission

A project with great Potential

November 23, 2018

Abstract

The Laplace equation is one of the corner stones of Classical Electrostatics. In this report we will examine different numerical methods which may be used to obtain numerical estimations to 2-dimensional problems where the geometry is too complex for analytical manipulation. Specifically, we employed iterative methods of Successive Over Relaxation (SOR) as well as direct methods which utilise the LU-decomposition algorithm. We then compared the two methods for problems of existing analytical solutions as well as those demanding numerical simulation, conducting both error and performance analysis. Parallel to this, a GUI was developed to encapsulate the project into a single executable, allowing for user defined geometry and charge generated from dynamic drawing.

1 Introduction

We initially solved two problems mathematically in order to have some analytical solutions which could be used to reconsile with the results that our numerical approximations might come up with. The two problems are illustrated and enumerated below:

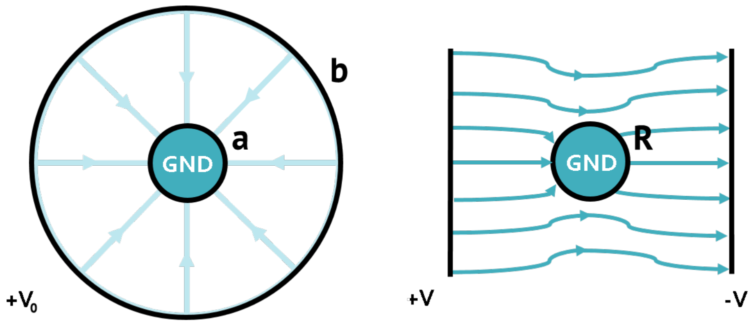


Figure 1: Problem 0 (radii a, b) and Problem 1 (radius R)

2 Analytical problems

2.1 Analytical Solution to problem 0

To solve problem 0 we need to begin from the expression of Laplace's equation in cylindrical coordinates, where as usual, r is the radial distance, ϕ is the angular coordinate and z the height:

$$\nabla^2 V = \frac{1}{r}\frac{\partial}{\partial r}\left(\frac{r\partial V}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 V}{\partial \phi^2} + \frac{\partial^2 V}{\partial z^2} = 0 \tag{1}$$

Firstly, we identified that there is circular geometry, so we could omit the angular coordinate, as well as disregard the height, thus reducing the problem to 1-dimensional geometry, described by the truncation of equation 1 seen underneath.

$$\nabla^2 V = \frac{1}{r}\frac{\partial}{\partial r}\left(\frac{r\partial V}{\partial r}\right) = 0 \tag{2}$$

Solving this equation for V is relatively easy, and integrating twice will suffice.

$$\int \frac{1}{r}\frac{\partial}{\partial r}(\frac{r\partial V}{\partial r})dr = \int 0dr \implies r\frac{\partial V}{\partial r} = constant = c \tag{3}$$

$$\implies \int \frac{\partial V}{\partial r}dr = \int \frac{c}{r}dr \implies V(r) = c\ln(r) + d \tag{4}$$

Finally, using our boundary conditions $V(a) = 0$ and $V(b) = V_0$ we obtained an equation governing the potential field:

$$\boxed{V(r) = \frac{V_0}{k}\ln\left(\frac{r}{a}\right) \text{ where k} = \ln\left(\frac{b}{a}\right)} \tag{5}$$

We then proceeded to solve problem 0, which posed a more interesting challenge. Due to this problem's geometry, only the height variable can be omitted and both the remaining variables in equation 1 needed to be considered as seen underneath.

$$\nabla^2 V = \frac{1}{r}\frac{\partial}{\partial r}\left(\frac{r\partial V}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 V}{\partial \phi^2} = 0 \tag{6}$$

This is particular equation has solutions of the form:

$$(r, \phi) = \sum_{n=1}^{\infty}(C_n r^n + \frac{D_n}{r^n})(A_n\cos(n\phi) + B_n\sin(n\phi)) \tag{7}$$

we obtained this by assuming separation of variables $V(r) = Q(\phi)R(r)$ such that

$$\nabla^2 V = \frac{1}{r}\frac{\partial}{\partial r}\left(\frac{rQ\partial R}{\partial r}\right) + \frac{1}{r^2}R\frac{\partial^2 Q}{\partial \phi^2} = 0 \tag{8}$$

$$\implies \frac{Q}{r}\left(\frac{\partial R}{\partial r} + r\frac{\partial^2 R}{\partial r^2}\right) + \frac{1}{r^2}R\frac{\partial^2 Q}{\partial \phi} = 0 \tag{9}$$

multiplying both sides by $\frac{r^2}{QR}$ and reordering yields

$$\frac{r^2}{R}\frac{\partial^2 R}{\partial r^2} + \frac{r}{R}\frac{\partial R}{\partial r} = -\frac{1}{Q}\frac{\partial^2 Q}{\partial \phi^2} = n^2 \tag{10}$$

for some constant n. The part with Q(ϕ) can be solved through considering the auxiliary equation which yields imaginary roots, thus obtaining solutions of the form:

$$Q(\phi) = (A_n\cos(n\phi) + B_n sin(n\phi)) \tag{11}$$

with the restriction that $n \in \mathbb{Z}$ because of the single-valuedeness of Q(ϕ), which means that $Q(\phi) = Q(\phi + 2\pi)$, corresponding to a full rotation around the conductor, leading to a behaviour of $cos(n\phi)) = cos(n\phi + 2n\pi)$ which in turn restricts n to be an intiger.

The part with $R(r)$ is called a radial equation where we notice that the order of differentiation associated with each term is equal to the power of each r term, implying that a solution of the form R(k) $= r^k$ would suffice, given the exception that $k \neq 0$. Attempting such an approach gives

$$r^2\frac{\partial^2 r^k}{\partial r^2} + r\frac{\partial r^k}{\partial r} = n^2 r^k \tag{12}$$

$$\implies r^2 k\frac{\partial^2 r^{k-1}}{\partial r^2} + rkr^{k-1} = n^2 r^k \tag{13}$$

$$\implies r^2 k(k-1)r^{k-2} + rkr^{k-1} = n^2 r^k \tag{14}$$

$$\implies k^2 - k + k = n^2 \tag{15}$$

$$\implies k = \pm n \tag{16}$$

This means that by using the Principle of Superposition we obtain solutions of the form

$$R(r) = C_n r^n + D_n r^{-n} \tag{17}$$

Combining the two forms found in equations 11 and 17 we deduce the general solution quoted initially in equation 7 which can be used to solve our geometry.

Now it's time to examine the problem's physics. It is safe to assume that for a radial distance r much larger than the radius of the conductor R then the Potential and Electric fields would behave as if unaffected by the presence of the conductor, i.e. the equation describing the Potential field at $r \ll R$ is $V(r,\phi) = -E_0 r\cos(\phi) + K$ where E_0 is how the strength of the Electric field relative to the distance traveled from some point and K is the value of the potential at that point. Therefore, by comparing with the form of equation 7 we can identify that for this to be matched, n should equal 1 and the $\sin(n\phi)$ term should be omitted. We can then denote the remaining multiplied constants as $C = C_1 A_1$ and $D = D_1 A_1$ for simplicity, arriving at

$$V(r,\phi) = Dr\cos(\phi) + \frac{C}{r}\cos(\phi) = -E_0 r\cos(\phi) + K \tag{18}$$

From the general solution and the behaviour of V at large radial distances respectively. It is then easy to see that $D = -E_0$. Finally we consider our boundary conditions at the surface of the conductor, r = R where we know that

$$V(R,\phi) = -E_0 R\cos(\phi)\frac{D}{R}\cos(\phi) = 0 \tag{19}$$

$$\implies D = E_0 R^2 \tag{20}$$

$$\implies \boxed{V(r,\phi) = -E_0\left(r - \frac{R^2}{r}\right)\cos(\phi)} \tag{21}$$

It's now easy to compute E from the definition of the cylindrical grad

$$\boldsymbol{E} = -\nabla V = -\frac{\partial V}{\partial r}\hat{r} - \frac{1}{r}\frac{\partial V}{\partial \phi}\hat{\phi} \tag{22}$$

$$\implies \boxed{\boldsymbol{E}(r,\phi) = E_0\left[\hat{r}\left(1 + \frac{R^2}{r^2}\right)\cos(\phi) - \hat{\phi}\left(1 - \frac{R^2}{r^2}\right)\sin(\phi)\right]} \tag{23}$$

Finally, to make a sanity check we can compute the charge density on the conductor's surface

$$\sigma = \epsilon_0 E(R,\phi) \implies \boxed{\sigma = 2\epsilon_0 E_0\cos(\phi)} \tag{24}$$

Giving us that the total charge density is zero as we would expect.

$$\sigma_{total} = 2\epsilon_0 E_0 \int_0^{2\pi} \cos(\phi)d\phi = 0 \tag{25}$$

As mentioned earlier these analytical solutions were obtained in order to provide values which the code we developed could be compared with. The main problem that interested us was the geometry of a multi-wire proportional chamber which can be seen underneath.

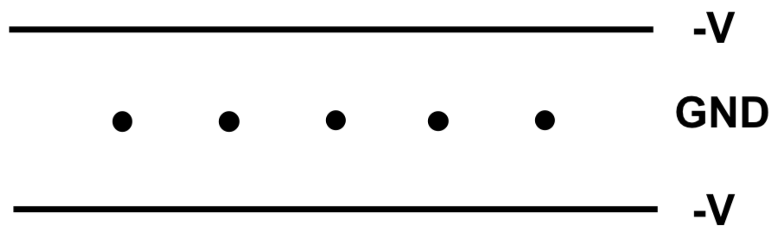


Figure 2: A cross section of the multi-wire proportional chamber (MWPC)

3 Numerical methods

The multi-wire proportional chamber was developed by George Charpak in 1968. As seen in the above image the MWPC consists of wires at ground potential, enclosed in a chamber at negative potential such that any negatively charged particles like electrons are drawn towards the wires due to the electric field. This effect is amplified because the chamber is filled with a gas that becomes ionised by incoming charged particles. The electric field then accelerates the ions and electrons which in turn create more ionisations which are drawn towards the array of wires, registered as a charge. This effect, known as Townsend avalanche is proportional to the ionisation effect of the incoming prticle which means that by combining information from the wires, the particle's trajectory may be determined. This method increased the rate of detection by a factor of 500 compared to the bubble chambers used earlier and resulted in George Charpak being awarded the Nobel Prize in Physics in 1992. Inventions like this further rectify our interest in modeling potential fields.

An analytical solution does not exist for such a geometry so we resorted to applying numerical techniques on the Laplace equation. This was done by considering a truncation of the Taylor expansion, for small distance h traveled in the x and y directions.

$$V(x \pm h, y) = V(x,y) \pm h\frac{\partial V(x,y)}{\partial x} + h^2\frac{\partial^2 V(x,y)}{\partial x^2} \pm h^3\frac{\partial^3 V(x,y)}{\partial x^3} + O(h^4) \tag{26}$$

$$V(x, y \pm h) = V(x,y) \pm h\frac{\partial V(x,y)}{\partial y} + h^2\frac{\partial^2 V(x,y)}{\partial y^2} \pm h^3\frac{\partial^3 V(x,y)}{\partial y^3} + O(h^4) \tag{27}$$

$$\implies V(x+h,y) + V(x-h,y) + V(x,y+h) + V(x,y-h) = 4V(x,y) + O(h^4) \tag{28}$$

since the terms of h and h^3 cancel out due to their signs and the terms of h^2 are equal to zero in the case of the Laplacian in free space. This is the quintessential equation to this project and it informs us that the value of the potential at some point in our 2-dimensional grid is approximately equal to the mean of the 4 surrounding points, hence its name 'the five point star (FPS)'.

At this instance the team split into 3 subgroups with the aim of creating a program with a GUI where the user could enter any desired geometry and charge, with the code approximating the potential field using the FPS. the first subgroup was tasked with developing an iterative solution to the FPS, the second a direct solution to the FPS and the third a working GUI which could encompass the all aspects of the code including the drawing and plotting phase. All of the inner workings will be elucidated further in this report. The first problem we faced after compartmentalising the work was how could the user set the charge of the drawn shapes and furthermore how could the code convey the presence of shapes and their charge to the numerical methods mentioned above. To that extend we came up with a simple and aesthetically appealing solution, where the charge was related to a colour gradient. The user first specifies the maximum voltage which he is interested in and that is then associated with a colour gradient, where positive charge relates to red and negative to green. Any undefined space remains white and ground charged shapes are black.

After the user has finished drawing the shapes and requests for a computation of the potential field we read the resulting image as a png and then pick out information pixel by pixel. Running the algorithm in reverse we can deduce the charge at each pixel from it's colour as well as if it is a defined shape of constant charge or empty space (i.e. if it is white) and store this information in a matrix. This means that for an image of NxN pixels the code is supplied with a 3-dimensional array of NxNx2 dimensions since every pixel corresponds to 2 values, the first one being the amount of charge, and the second one conveying whether there that pixel corresponds to a shape or not.

4 Introduction to the Direct Method

The direct method is possibly the most logically straightforward way to solve for the values of the mesh points. It involves expressing the relationships of the various grid points with each other through a system of simultaneous equations, and then solving this system to obtain a vector containing the solutions to the mesh points. Thus, in contrast to the iterative methods, results are obtained at their final accuracy rather than being successively improved over several cycles.

In our implementation we consider as system of the form:

$$\begin{aligned}
-4V_1 + V_2 + 0V_3 + 0V_4 + V_5 + \cdots + 0V_{16} &= 0 \\
V_1 - 4V_2 + V_3 + 0V_4 + 0V_5 + V_6 + \cdots + 0V_{16} &= 0 \\
\vdots \\
0V_1 + \cdots + V_{12} + 0V_{13} + 0V_{14} + V_{15} - 4V_{16} &= 0
\end{aligned}$$

Which can be expressed by the matrix equation:

$$\mathbf{A}\boldsymbol{v} = \boldsymbol{b} \tag{29}$$

Where $\mathbf{A}$ is a matrix holding the coefficients of the system, $\boldsymbol{v}$ holds the voltage solutions, and $\boldsymbol{b}$ represents the right hand values of the system. We could now apply a variety of generic matrix solving techniques to find $\boldsymbol{v}$ but these methods quickly become impossible to practically implement in larger problems for reasons discussed in the following sections.

5 Implementation of the Direct Method

The first thing to notice is that if we have a $N \times N$ grid then the coefficient matrix has N^4 elements. This quartic growth means that the storage space required quickly overtakes any realistic memory capacity. Furthermore, even if one could procure such an expansive memory device, we would still leave much to be desired in terms of efficiency. To understand why this is, consider the structure of the matrix $\mathbf{A}$. We can notice that on any given line there are at most five non zero elements: the point of interest, and the four points around it. The rest of the elements on that row are initialised to zero. The net effect of this is that the matrix $\mathbf{A}$ is *extremely* sparse, especially for large N. In fact the sparseness of the matrix is the principle factor on which optimisations are made [1]. In the next section we discuss methods of storing large sparse matrices and the challenges they present.

5.1 Storage Systems for Sparse Matrices

The essential idea behind sparse matrix compression is to minimise the required storage space by only storing non-zero elements and there respective positions such that all the information of the original matrix is still preserved. Although there several compression schemes available [2] we utilised coordinate storage (COO) and compressed column storage (CCS).

The COO storage format is possibly the simplest to understand and implement. It essentially involves storing three columns or arrays of data: one that tracks the row numbers of the non-zero elements, another that tracks the column numbers, and finally one that tracks the values themselves. While this method offers a fairly good compression rate for large N and is easy to implement, it does suffer from a few drawbacks. Firstly, it does not optimise on the fact that several non-zero elements share the same column (or row) number; secondly, and more significantly, if we wanted to access the non-zero element v with row number i and column number j we would have to scan through all the elements preceding it.

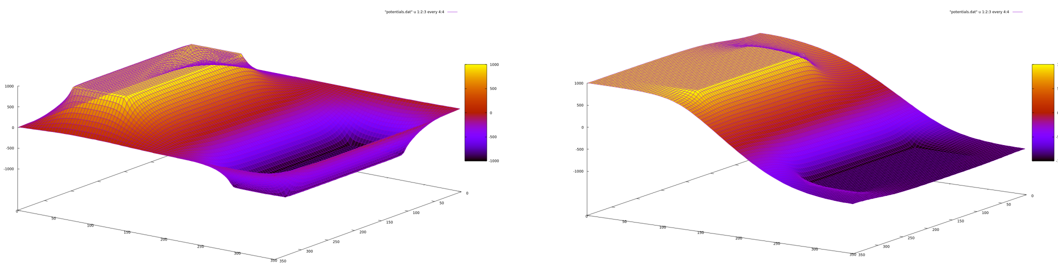
To overcome these shortfalls we may use the CCS format. As done previously we will maintain three data sets, one of which will hold the non-zero values. The second data set will hold the corresponding row values of those elements. Finally, the third set holds holds *column offsets* which tell us where in the value set the next column begins. This not only reduces the space required but also allows us to *jump* to a requested value by referring to the column offsets rather than scanning through every element [2].

5.2 Formulating the Linear System

The system of equations as described thus far holds no information with regard to any specific geometry and therefore solves no particular problem. Thus, an essential step is finding a mechanism where by the system can be tweaked to reflect a given problem.

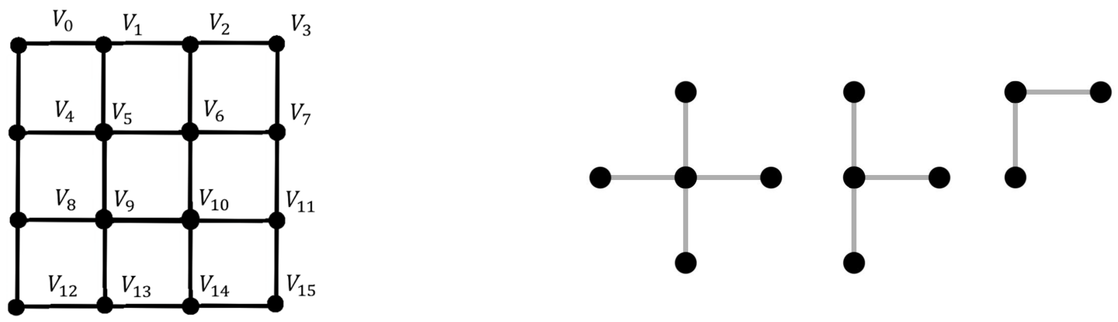
To see how this was achieved, consider the 4×4 grid shown in Fig. 4a with a voltage of 25V at v_6. How can this information be represented in the system? The solution employed was to zero out the row corresponding to V_6 (row 7 if we index from 0) and write a 1 into its diagonal. We then write 25 in the corresponding element of the $\boldsymbol{b}$ vector. The reason this works can be seen by multiplying out the left hand side of (29). Notice that all the zero elements in row 7 will ensure almost all the variables disappear, leaving only the 1 in the diagonal to be multiplied to yield the equation $v_6 = 25$ which is precisely the restriction we wished to specify.

To deal with the boundaries we use a method we shall call *adaptive stencilling*[1]. The idea is to adapt the number of *legs* of our stencil as we crawl the mesh depending on where we are; changing from a three point stencil to a four point one when we are at the corners or the sides respectively. Without this modification, every time we wish to calculate a point on the side of the grid (e.g. v_{11} in Fig. 4a) we end up dividing the by 4 even though we only have 3 neighbours. Thus it is as if we have assumed there to be a fourth point that is always 0. The net result of this is the boundaries are fixed at nearly zero which, needless to say, is in general incorrect.



(a) (b)

Figure 3: Results with and without adaptive stenciling respectively. Notice how the boundaries are no longer fixed at zero.



(a) (b)

Figure 4: (a)An example of a 4×4 grid. (b) Some incarnations of the stencil at various points on the grid

Now that the matrix can be stored and tailored to our problems we can explore how the system is solved.

5.3 Solving the Linear System

To explain how SLU works we will break up the stages of the simplest algorithm it employs to solve a linear equation $AX = B$. The stage 1 concentrates on memory and the stage 2 on speed.

[1]this is a fairly simple modification of the five point star, but at the time of writing we failed to find reference or utilisation of it in the scientific literature

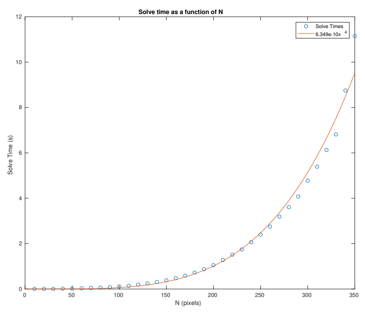


Figure 5: Variation of the solve time (in seconds) to N. The data fits quite well to a N^4 curve which is to be expected as this is the rate at which the matrix grows.

As mentioned earlier the matrix A is a very large, unsymmetric and sparse matrix so we store it in CCS form. This however would pose a problem since computing simple LU decomposition would result in dense L and U matrices, instantly defeating the purpose of storing A in CCS format. This is treated by SLU in part 1 of the following algorithm called 'Simple Driver Algorithm' seen underneath.

Compute $P_rAP_c = LU$ where P_r is a permutation matrix which reorders the rows and P_c reorders the columns. L is a unit lower triangular matrix and U an upper triangular one. P_c is chosen to order the columns of L and U to increase their sparsity. This is done by making use of another data structure called supernodes. Supernodes group together columns with the same non-zero structure thus conserving space. It makes sense then that to maximise the effectiveness of supernodes we permute the matrix to bring together columns with the same non-zero structures. This is can be done without destroying important information by a postordering on A's column elimination tree [7] which is computed in time almost directly proportional to the number of non-zero entries [8], giving us P_c. SLU can then perform dynamic pivoting and row-interchanging to simultaneously find the LU factorisation of P_c, A and P_r.

$AX = B$ can be solved via $X = A^{-1}B = (P_r^{-1}LUP_c^{-1})^{-1}B = P_c(U^{-1}(L^{-1}(P_r(B))))$ corresponding to a permutation of the rows of B, followed by solving a number of triangular systems equal to the entries in B by substitution. Similarly, followed by solving the same number of triangular systems, to obtain an answer for X without explicitely computing the inverse of A.

6 Performance of the Direct Method

We evaluated various metrics to gauge the performance of the algorithm. First we measured execution time with image size (see Fig. 5). The code was tested with the same image at various sizes and the results were plotted. Although exact solve times vary depending on the specific geometry, the general trend should be the same.

Error was analysed by comparing the analytical results of Problem 0 and Problem 1. For each pixel an absolute difference of the values was calculated and we were able to use this to calculate the relative error of the numerical solution.

7 Iterative Methods

7.1 Theory

In this section, the Iterative Relaxation Method will be discussed, analysing its three different approaches: The Jacobi's Method, The Gauss-Seidel Method and the Successive Overrelaxation Method (SOR).

The iterative methods rely on the fact that the Laplace Equation could be rewritten as a diffusive equation: [1]

$$\frac{\partial u}{\partial t} = \nabla^2 u \tag{30}$$

Here u defines Potential, and the equation solved by this program is set in 2D by assuming symmetry in the z direction. It is important to note here that the Laplace equation is time independent by design, so the t in equation (1) can be seen as a mathematical trick denoting the evolution of the equation rather than time. It is also evident from the equation that as the differential $\frac{\partial u}{\partial t}$ goes to 0, the solution for the Laplace equation is reached. Now, using FTCS (Forward Time Centred Space) differencing, the equation becomes [1]:

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{\Delta t}{\Delta^2}(u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n} - 4u_{i,j}^{n}) \tag{31}$$

Where Δ^2 is the grid spacing in the x or y axis. This is the Jacobi equation, with the subscript n denoting the simulated time evolution of the distribution u, while the terms i and j represent the real evolution in space along the two axes. This nomenclature describes how the iterative methods discussed in this report use the current point $u_{i,j}^{n}$ and its neighbouring points to affect the next point in the evolution $u_{i,j}^{n+1}$. Moreover, using Von Neumann stability analysis shows that the equation is stable if $\frac{\Delta t}{\Delta^2} < \frac{1}{4}$ when dealing with two dimensions in space [3] [1].

We can make a slight improvement to the Jacobi method by introducing the Gauss-Seidel method which makes use of updated values of $u_{i,j}^{n}$ as soon as they become available:

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{\Delta t}{\Delta^2}(u_{i+1,j}^{n} + u_{i-1,j}^{n+1} + u_{i,j+1}^{n} + u_{i,j-1}^{n+1} - 4u_{i,j}^{n}) \tag{32}$$

Finally, we introduce the Successive Overrelaxation Method (SOR):

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{\Delta t}{\Delta^2}\omega(u_{i+1,j}^{n} + u_{i-1,j}^{n+1} + u_{i,j+1}^{n} + u_{i,j-1}^{n+1} - 4u_{i,j}^{n}) \tag{33}$$

The SOR method (33) introduces a relaxation parameter ω which makes an over correction to the value of $u_{i,j}^{n}$, de facto anticipating future corrections of the value. This over correction produce a considerable increase in the speed of the method. While the SOR method is convergent for $0 < \omega < 2$, we found that the only way to speed up the process is through overrelaxation, which occurs for: $1 < \omega < 2$. The optimal value of the overrelaxation parameter omega for a square grid is thus given by the expression:[4] [3]

$$\omega \approx \frac{2}{1 + sin(\frac{\pi}{N})} \tag{34}$$

Where N is the size of the square grid in one dimension

7.2 Boundary Condition

One of the most complex problems encountered during this project has been dealing with the grid's boundaries. The problem arise from the fact that all the methods discussed previously need four points in order to be successful, something which does not happen at the boundaries as only three (and two at the corners) points are available. Through literature readings we decided that the best method to be used was the *Von Neumann Boundary Condition* [5].

The idea at the core of its method is to assume that close to the boundaries the slopes between those points can be approximated to be equal.

If, for example, the left hand side boundary needs to be calculated, then it is safe to say that

$$\frac{u_{i,1}^{n} - u_{i,-1}^{n}}{2} = \frac{u_{i,3}^{n} - u_{i,1}^{n}}{2} = f'(u)$$

Where $u_{i,-1}^{n}$ is a *ghost* point outside the grid.

Now, since $u_{i,-1}^{n} = u_{i,1}^{n} - 2f'(u)$, the point $u_{i,0}^{n}$ can be calculated as:

$$u_{i,0}^{n} = \frac{1}{4}(2u_{i,1}^{n} - 2f'(u) + u_{i+1,0}^{n} + u_{i-1,0}^{n}) \tag{35}$$

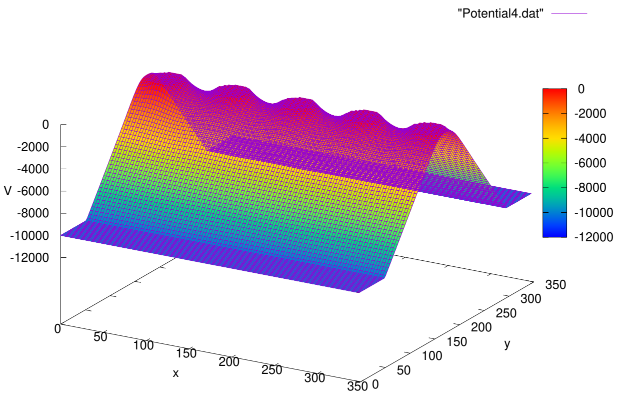
Equation 35 is used to calculate the points at the boundaries with the exception of the corner points.

$$u_{0,0}^{n} = \frac{1}{2}(u_{1,0}^{n} + u_{0,1}^{n}) \tag{36}$$

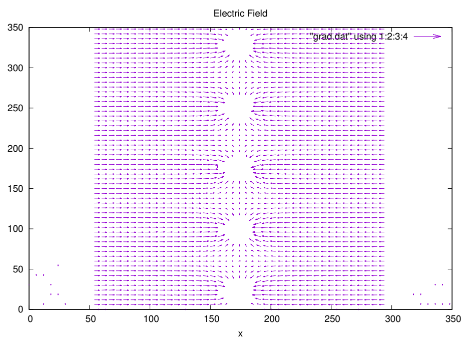
In order to get those point, for example for the upper left corner, we take an avrage of its two immediate neighbours, as described by equation 36.

7.3 Results

This section aims to discuss and compare the differences between the Jacobi, the Gauss-Seidel and the SOR method. To do this, we consider a 350x350 PNG file depicting the layout of (ODY FIGURE) as outputted by the GUI. The boundary conditions of this problem are thus set to two parallel infinite plates set to a voltage of -1000, enclosing a strip of centred grounded wires. Updating the rest of the points iteratively using the three aforementioned methods provided the following results.



(a)



(b)

Figure 6: The final version of the GUI, color highlighted to aid with explanation.

Method	CPU time (s)	Iterations	error tolerance
$Jacobi's$	54.8108	5915	1×10^{-2}
$Gauss - Seidel$	40.7208	4279	1×10^{-2}
SOR	5.79559	461	1×10^{-2}

Table 1: Iterative Methods

Table 1 shows how the choice of the method affects the time taken by the program to produce solution as shown in figure 12. We note here that what is considered to be 'acceptable solution' is quantified by the user defined error tolerance, which triggers a stopping mechanism that will be discussed later in the report.Table 1 shows how the choice of the method affects the time taken by the program to produce an acceptable solution as shown in figure 12. It is clear from table 1 that the SOR method is indeed the fastest method among the ones discussed in section 7.1. Moreover, we can see that the relaxation parameter allows us to get to the same solution produced by the other methods with a considerably less number of iterations, thus requiring considerably less computation time.

In section 7.1 it has been discussed how the SOR method depends on the overrelaxation factor ω. If equation 34 is applied to the 350x350 grid mentioned before, we get the following result for the optimal factor: $\omega \approx 1.98221$. A simple plot of computational time versus applied ω shown in 7 confirms that the optimal value of ω indeed lies in the immediate neighbourhood of 1.98221.

Figure 7 confirms that the optimal value of ω lies in the immediate neighbourhood of 1.98221, and, as a consequence, so is equation 34.

Having chosen the optimal combination of iterative method and laxation parameter, we now how another dependable method for simulating electrostatic problems. The following section will further illustrate the optimisations and limitations that have been done to achieve the best possible accuracy and computational time, as well as providing a comparison between the direct and iterative methods as a whole.

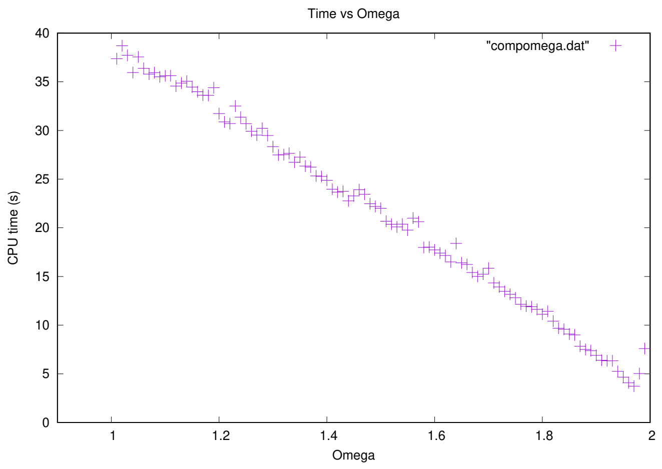


Figure 7: Omega agains Time

8 Optimisations, Errors and Comparisons

Having discussed the inner workings of the iterative method, we can now consider the optimization and error analysis that the code underwent. In particular, we tackle boundary inaccuracies with different combinations of different approaches, as well as a stopping mechanism which gives the GUI user control of the iterative method. Lastly an error analysis is presented by comparing Direct and Iterative schemes with themselves and the analytical results.

9 The Boundary Problem

As previously explained, the finite difference schemes obtain their values by considering the neighboring points, with the boundary condition points remaining fixed. However, this also fixes all the edges to 0 which yields unphysical solutions of 'bounded free space' with often sharp slopes forced to zero as shown in Figure 8

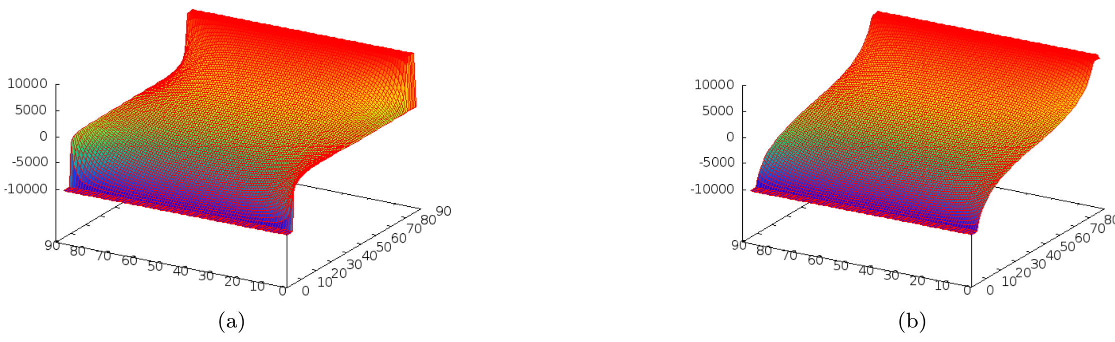
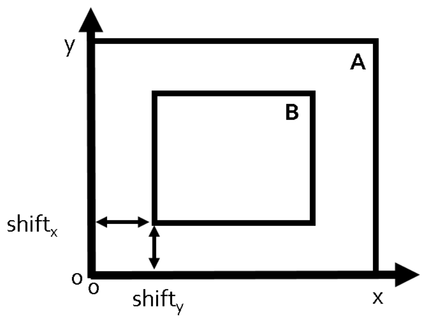


Figure 8: Demonstrating the problem of bounded boundaries and a solution using Concentric Canvases

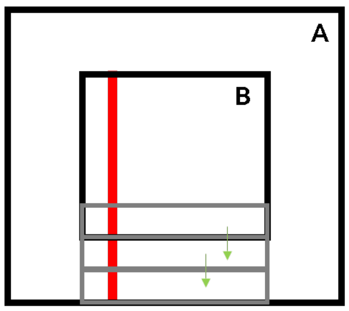
There are two ways to address this issue as we check for how the work individually and in tandem. Namely we can apply the previously discussed von Neumann boundary conditions, and the method of Concentric canvases. The latter method in particular was improvised, and will thus have it's workings elaborated further below.

9.1 Concentric Canvases

In order to simulate a natural boundary behavior for any geometry we introduce a bigger canvas A, concentric with our area of interest canvas B, as shown in Figure 9. By applying the boundary conditions on the bigger canvas we minimize the error effect on the simulated area with the assumption that potentials will tend towards zero at large distances. This is an attempt to simulate free space, based on the assumption that potentials will tend towards zero at large distances.



(a)



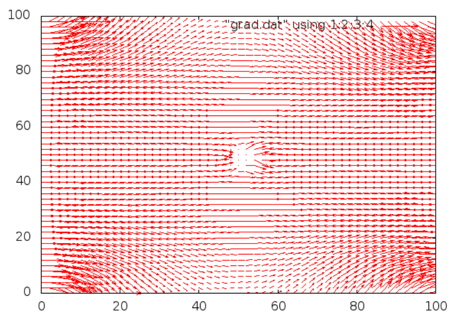
(b)

Figure 9: Concentric Canvases A and B with B the area of interest(left), Infinite Mode methodology; duplicating boundary of Canvas B to the edge of canvas A (right)

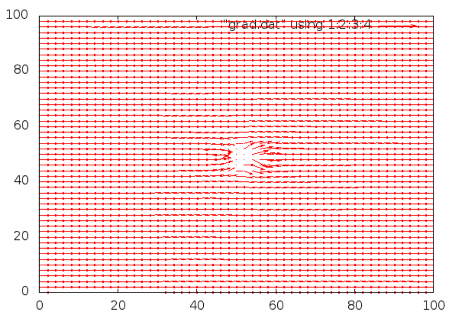
9.1.1 Simulating Infinite Plates

Although this approach successfully simulates finite geometries, fails to simulate scenarios of infinite plates since plates that span the small canvas B, are effectively finite in respect to canvas A. However, here we can take advantage of the bigger canvas A.

This is possible to do that by extending the plate from canvas B to the edges of canvas A. This is done by simply duplicating the boarders of canvas B to the edges of canvas A as shown in Figure 9 but still plotting potentials within canvas B. The result is to eliminate gradient components out or towards the canvas, parallel to the two plates as show in Figure 10



(a)

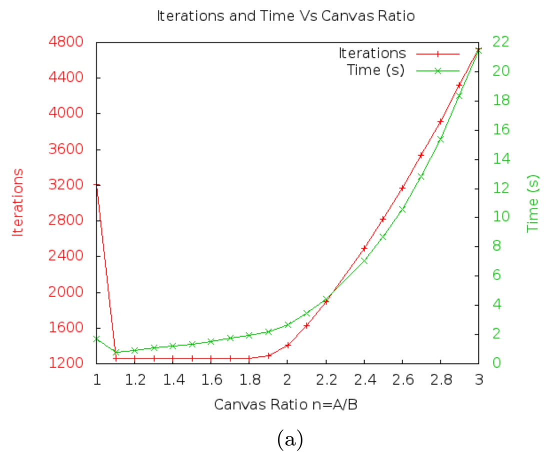


(b)

Figure 10: Plotting the vector field **E** without and with Infinite Mode.

9.1.2 Optimum Canvas Size

However, we should carefully choosing a canvas size since it heavily affects computational speed, requiring more iterations to solve. Below we can see computational time to be proportional to N^2 for the canvas size we are solving. By considering again Problem 1 and it's analytical answer, the relative error is plotted against different canvas ratios. Complimentary to that, Figure 1.4 shows both iterations and computational time to reach a certain relative error against different canvas ratios. As expected a bigger canvas ratio is computationally heavier, but more accurate. Any canvas ratio though between 1.2 to 1.5 it provides increasing accuracy with a relatively small time costs. We finally found the preferred size is a ratio to be n=1.2, followed by further analysis on the combination of this approach and boundary conditions.



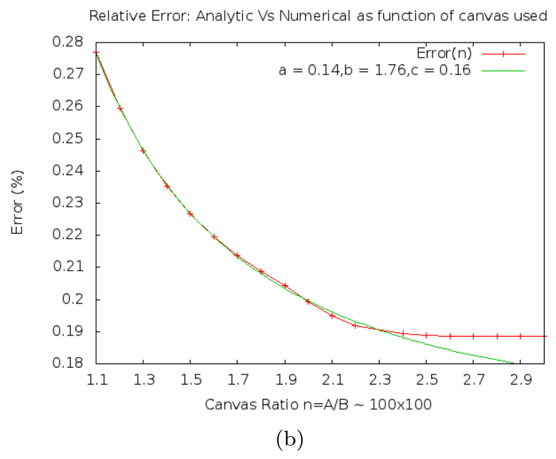


Figure 11: Plotting Iterations and Computational Time against different canvas sizes as well as the relative error compared to the analytical results for Problem 1.

10 Combined Results

It is our interest to examine the combination of the two methods considering Problem 1, by comparing our results with the analytical ones for problem 1. As shown in the table below, both methods, Concentric Canvases and von Neumann Boundary conditions improve the accuracy of the scheme. However it is clear that the latter drastically do so with significant improvement. Specifically the best combination was using a canvas of ratio 1.2 with von Neumann Boundary conditions. Although there is a minuscule difference by using the 'Infinite Mode', keeping it is motivated when the iterative method is compared with the direct method which will be discussed later.

Boundary Cond.	Canvas Ratio (n)	Inf.Mode	Error(%)
off	1	off	26.75
on	1	off	0.81
off	1.2	off	15.19
off	1.2	on	13.44
on	1.2	on	0.85
on	1.2	off	0.74

11 Stopping Mechanism

As it comes to the optimization of our program, It is important to have some control over the iterative method and what any particular user considers to eb acceptable accuracy. Hence, we introduce a stopping mechanism as well as a maximum allowed iterations to prevent unreasonable long run times.

Since SOR method is a linear convergent method it can be shown that it satisfies the inequality

$$\frac{\|u^{n+1} - u^{n)}\|}{\|u^{n} - u^{n-1}} \approx r, \quad r < 1 \tag{37}$$

where 'n' is the 'n^{th}' iteration and 'r' the constant the ratio presented converges to.[6] By assuming the forward in time solution, 'u^{n+1}' to be an approximation to the true solution of the Laplace equation the relative error can be calculated.

$$\frac{r}{1-r}\frac{\|u^{n+1} - u^{n}\|}{\|u^{n+1}\|} = \epsilon_{tol} \tag{38}$$

where 'ϵ_{tol}' is the desired tolerance. 'r' was calculated to be 0.996.

12 Error

12.1 Comparison of Methods: Direct Vs Iterative

By now, Emission has developed two separate methods for solving the Nobel Prise winning problem. It is then in our interest to compare the the performance of the two methods with regards to the Nobel Price Problem. Since the iterative method by definition converges to the direct method, the later is used as the reference for calculating the relative error.

Boundary Cond.	Canvas Ratio (n)	Infinite Mode	Error(%)	Iteration	Time
on	1.2	on	0.718	468	6.69
on	1.2	off	0.518	458	6.76
on	1	-	0.075	460	4.48

It turns out that completely omitting the Concentric Canvas method yields the greatest agreement between the two methods (With a difference in order of magnitude by 1). This is expected since they are calculated in exactly the same manner as the direct method does not involve a larger canvas. That is, there are only relatively small errors mainly in near the grounded circles themselves which are expected to decrease if the method is allowed to iterate more. Furthermore, one can argue that even with a concentric canvas, the results turn out to be in greater disagreement with the infinite mode activated rather without it. However, as shown in Figure **??**, on the first graph starting from left, the error is much less at the boundaries. That is the method is forces the boundaries to converge at a straight line faster.

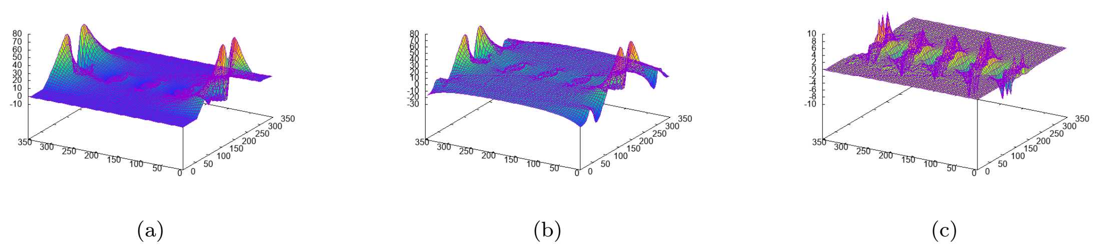


Figure 12: The final version of the GUI, color highlighted to aid with explanation.

Lastly, choosing the iterative method with combined Concentric Canvases approach and von Neumann Boundary conditions as the most general version we investigate the execution times of both methods on tackling problems 0, 1. The results are presented in the following tables.

Method	**Execution Time** (s)	**CPU Memory**
Iterative (SOR)	11.55911	-
Direct	8.6835	705.83

Table 2: Problem 0

Method	**Execution Time (s)**	**CPU Memory**
Iterative (SOR)	20.3519	-
Direct	15.1301	1059.96

Table 3: Problem 1

In conclusion, both methods are developed and optimized to a great flexibility, able to solve any given, finite or infinite plate geometries. However, the iterative method is slightly more time inefficient, but allows more control over the user through the stopping mechanism, which can be toggled to low tolerance allowing us to obtain quick estimations or general trends.

The Graphical User Interface (GUI)

The Emission project had always been intended to run using a GUI front end, allowing the user to have full control over the geometry and charge of the input potential. The following section of the report will document the development of such an interface, discussing the choice of development platform as well as the functionality of the final product.

Functionality

At it's core, the main purpose of a GUI is to provide an intuitive environment in which information necessary for solving a given problem can be extracted from the user. Given the fact that the problem this software package aims to solve is quite particular, it was important to lay out explicitly what information was needed from the user, as well as the methods used to communicate it.

As seen in the previous sections, both the direct and the iterative methods work by first reconstructing an image of arbitrary geometry, in which shapes and their color provide boundary conditions for solving the potential field. Hence, the main purpose of the GUI had always been to provide the user with an image generator in which one can dynamically draw basic shapes of charge, specified by colors in a user defined spectrum. This setup allows for notable optimization by having the GUI output images at computationally manageable sizes, tailored to include only separate shades of green and red for different charges (this heavily simplifies the translational algorithm). All this while not depriving the user of the freedom to create any desired 2D boundary conditions.

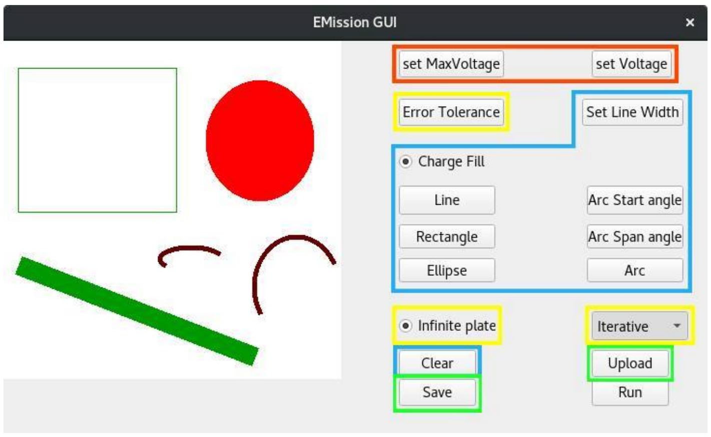
Apart from a dynamic canvas, it was essential that we have a GUI that is fully integrated into the project framework. What this means is that the GUI can run alongside the two methods in a single executable, with the ability to store and pass variables of any type, which can then be used as input arguments to functions solving the problem.

Developer Software

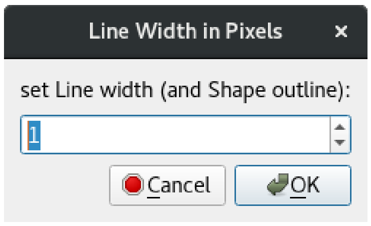
Having decided on the functionality of the GUI, the next step for us was to chose a developer platform which would allow us to create it within the given time frame. In the end, the GUI developing software chosen was Qt since it is geared towards C++ development which was our primary coding language. Moreover, Qt has it's own project builder with a static compiler, allowing us to distribute the final product of Emission as a small Linux application. All this, coupled with comprehensive documentation for beginners made Qt a very good choice and we could thus get started on building the GUI.

Implementation

Building a GUI through Qt is inherently done through object orientated programing, as the entire framework is based on a hierarchy of classes and objects. Thus, we needed to define our own class which would act as a blueprint of all the things that make up what the user interacts with in the application window, which in turn would be the object. This blueprint would lay out all the data that the application needs to extract and store from the user, the visual layout, and the functionality. Moreover, we define our class to be a parent, inheriting other predefined Qt classes to widget objects such as buttons and pixmaps as it's children. The final outcome is demonstrated by the figures below.



(a)



(b)

Figure 13: The final version of the GUI, color highlighted to aid with explanation.

The first feature to point out is the voltage feature (red) which is integral in the translational algorithm. The user is given two input buttons which lead to a menu similar to the one projected in picture (b); there, the user sets an integer maximum voltage x, which allows us to assign a voltage to all the possible shades of red from $[0, x]$ as positive charges, and all possible shades of green from $[-x, 0]$ as negative charges. The user can then set a voltage from the second menu and draw with it's respective color, while the maximum voltage is stored and passed onto the solving algorithms.

The maximum voltage is then the first value stored by the program, and is of type integer. The next few variables that the program is storing are highlighted in yellow, and consist of a double value for the error tolerance at which the stopping mechanism of the iterative method activates; a boolean value which is toggled by a radio button, telling the program if the drawn geometry extends to infinity or not; and lastly a boolean drop down menu allowing the user to choose between the iterative or direct method. Furthermore, green highlight points out functionality allowing the user to save their drawing into a gallery directory, or upload a picture of their liking by typing it's file path. The file path of the picture chosen to be solved is saved as a string. Finally, the run button passes all this information as input to the solving functions which run GNU Plotter internally.

The remaining functionality is highlighted in blue, and is demonstrated in the canvas of figure (a). First, we set the line width dialogue box to 1 pixel as shown in figure (b), and draw a rectangle with no fill at a negative voltage. Then, setting the voltage to be positive, we use the ellipse handle to draw a sphere with charge fill on. After this, the line width is set to 20 pixels, and voltage is changed back to negative as we draw a line which appears to be a diagonal rectangle. Finally, setting a relatively low positive voltage, we draw two arcs by first inputting their initial orientation with 'start angle' and then the angle which they will span using 'span angle'. The drawing is always done dynamically by selecting a shape and then clicking on the canvas, dragging to set the desired size, and then releasing. The clear function has also been included in case of an error.

The GUI described here is called as a function in the main program of the project, which allows for a segregation of Qt based programing from the rest of the project; the segregation thus only ever needs to be resolved through dependencies in the main header file. This type of compartmentalization is made possible thanks to the C feature of external programming which allows one script to access and overwrite variables of a different script, assuming that the two scripts are linked by the compiler. Hence, user keyboard and mouse interactions with the GUI can make real changes to input variables of the two solution methods, and Emission runs as a single compact unit.

Conclusion

The Emission Project has been developed with the intent to solve and simulate electrostatic problems of any geometry that can be described by the Laplace Equation. To do this we have implemented two separate approaches of numerical simulation in the form of direct and iterative methods. On comparing the two methods, advantages and disadvantages became clear. The iterative methods were found to be slower for a given relative error, perhaps partly due to the fact that the routines written were almost completely written from scratch while the direct method used heavily optimised libraries that linked to specialised Basic Linear Algebra System (BLAS) libraries. We implemented both Neumann and Dirichlet boundary conditions and achieved relative error percentages on the order of 1%. Furthermore the methods agreed with a high degree of accuracy with each other.

In addition to this, we have ensured that the entire Emission project is presented as an application which can be run through a single executable, and has an integrated GUI allowing the user to freely create his or upload his own specific electrostatic geometries. Despite our progress, there are still notable areas of improvement for the code performance, including second order Von Neumann boundary conditions for the iterative methods, as well as simulation using multi gird methods which could drastically increase code performance. Nevertheless, we feel that Emission provides the user with a comprehensive and intuitive solver for general electrostatic problems. The Emission project has been developed with the intent to solve and simulate electrostatic problems of any geometry. To do this

References

References

[1] *Numerical Recipes: The Art of Scientific Computing*
W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery
September 2007
ISBN: 9780521880688

[2] *101 Ways to Store a Sparse Matrix*
Max Grossman
`https://medium.com/@jmaxg3/101-ways-to-store-a-sparse -matrix-c7f2bf15a229`
Accessed: 12th March 2018

[3] *Fundamentals of matrix computations, second edition*
David S. Watkins
2002

[4] *The optimal relaxation parameter for the SOR method applied to the Poisson equation in any space dimensions*
Shiming Yang, Matthias K. Gobbert
2008

[5] `https://www.12000.org/my_courses/UC_davis/fall_2010/math_228a/HWs/HW3/Neumman_BC/Neumman_BC.htm`

[6] `http://ta.twi.tudelft.nl/users/vuik/burgers/lin_notes.pdf`

[7] *R. Gilbert and E. Ng, Predicting structure in nonsymmetric sparse matrix factorizations, in Graph Theory and Sparse Matrix Computation*
A. George, J. R. Gilbert, and J. W. H. Liu, eds.,
Springer–Verlag, New York, Berlin, 1993.

[8] *The role of elimination trees in sparse factorization* J. W. H. Liu