Theoretical Physics Group Project 2016

L. Gundelach, C. Osborne, F. O'Sullivan, L Renfrew, O. Windeman
University of Glasgow
Year 3 Theoretical Physics
Group 3

March 11, 2016

Abstract

This report describes the use of both analytical and numerical methods in the calculation of the electric field within and edge-coupled stripline, a high-speed data link commonly used in particle physics. To better understand the electric configuration within the stripline a C++ based software suite was developed to numerically solve Laplace's equation for the electric potential for an arbitrary electrode setup that is specified using an input image. From the electric potential the electric field can then easily be calculated. The final software package features an optional interactive graphical interface and four numeric algorithms to choose from: standard finite difference (Jacobi), Gauss-Seidel iteration, red-black ordering and matrix inversion. The accuracy of each method is tested against a simple problem of known analytic solution. The methods are then applied to the stripline problem. In the case of a high resolution image the red-black method is always the most efficient method. For lower resolution images the red-black and Jacobi methods are comparable. The software presented in this report is widely applicable to generic geometries specified by an input image and can be easily used from the command line or graphical interface.

1 Introduction

It is the purpose of this report to detail the process of solving Laplace's equation in order to obtain the form of the electric potential and the electric field for a given electrode setup. In many situations Laplace's equation cannot be solved analytically and numerical approaches are required. The aim of this project is to create a software package that can calculate the electric field within an edge coupled stripline and for general electrode setups using various numerical methods. The edge coupled stripline as illustrated in Figure 3 is a datalink commonly used in particle physics for high speed data transfer. The potential difference between the positive and negative electrodes that run though the wire is used to transmit data. The open sides to the left and right allow for multiple striplines to be connected in parallel. Gaining an understanding of the electric field which these electrodes, the two grounds and the grounded boundaries produce can help implement the edge coupled stripline in practical applications.

2 Laplace's Equation for Electric Potential

The aim of this project is to determine the potential and electric field for a given electrode setup. Only the special electrostatic case with no free charges present within the setup is considered. Gauss's Law (1) gives the divergence of an electric field in terms of the density of charges within

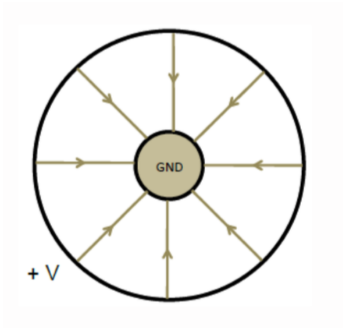


Figure 1: Concentric Cylinders

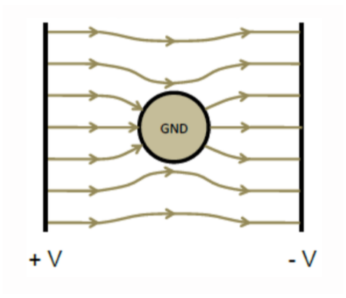


Figure 2: Ground Cylinder Between Plates

Images taken from the Project Description [1].

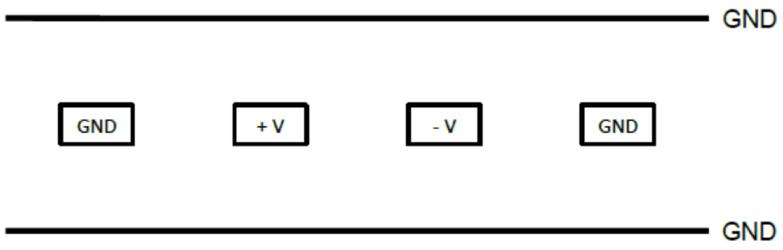


Figure 3: Edge Coupled Stripline

the system.

$$\vec{\nabla} \cdot \vec{E} = \frac{\rho}{\varepsilon_0} \tag{1}$$

In the special case where no charges are present, the divergence of the electric field is 0 and Gauss's law becomes:

$$\vec{\nabla} \cdot \vec{E} = 0 \tag{2}$$

The Maxwell-Faraday equation (3) states that the curl of an electric field is equal to the negative rate of change of the Magnetic Field with respect to time. In the electrostatic case there are no moving charges and hence no magnetic field is induced. The curl of the electric field is therefore zero for a steady, time-independent state:

$$\vec{\nabla} \times \vec{E} = -\frac{\partial^2 B}{\partial t^2} = 0 \tag{3}$$

Since the curl of the electric field is zero it can be expressed in terms of the gradient of a scalar potential function ϕ :

$$\vec{E} = -\vec{\nabla}\phi \tag{4}$$

This scalar potential is known as the electric potential or voltage. Taking the divergence of Gauss's law (2) and the definition of the electric potential (4) yields Laplace's equation for the electric potential:

$$\vec{\nabla}^2\phi = 0 \tag{5}$$

Solving this second order homogeneous differential equation to determine the electric potential is the core requirement of this project. From the electric potential the electric field is trivially calculated by taking the negative gradient of the potential.

3 Analytical Solutions For Two Situations

In order to test the validity of any numerical method utilised to solve Laplace's equation against an analytic solution two given situations were considered and the form of potential in each case was obtained by solving or at least attempting to solve the equation analytically. The first situation is that of concentric cylinders shown in Figure 1. Solving for this setup relies on circular symmetry. By expressing the Laplace equation in polar coordinates it is evident that the potential only depends on the distance from the central ground cylinder and not on the radial direction. Hence the boundary conditions on the potential of $0V$ at the inner (R_i) and $+V$ at the outer radius (R_o) may be applied to the general solution to the polar form of the Laplace equation in terms of only the radius r (see Appendix A).

A conformal mapping approach is also considered to verify this result. By expressing the concentric cylinders on a complex plane $z = x + iy$ and mapping them from the z plane into the complex $w = u + iv$ plane via the transformation $w = \ln z$ the circles in z become straight lines in w. This simplifies the application of boundary conditions in w and the solution to the Laplace equation in w is mapped back to z via the inverse of the initial transformation.

Both methods agree:

$$\phi(r) = \frac{V}{\ln\left|\frac{R_o}{R_i}\right|} \ln\left|\frac{r}{R_i}\right| \tag{6}$$

where R$_i$ and R$_o$ are the radii of the inner and outer cylinder respectively and r is the distance from the origin, in this case the centre of the concentric cylinders.

The second case shown in Figure 2 of a grounded cylinder confined by two infinite parallel plates is more complicated in that the potential features an angular dependence. In fact, the fundamental topological setup of this system makes it impossible to define an origin from which the potential can be continuously defined for all points. The central ground essentially cuts a hole of $0V$ into the physical setup creating discontinuities. To arrive at an approximate solution it is postulated that the potential far from the central ground is determined only by the parallel plates and the central ground becomes negligible. The outer boundary condition becomes the parallel plate solution at a radius equivalent to the distance from the centre to one of the outer plates. In polar form the potential along this bounding circle is then given by

$$\phi(R_2, \theta) = -V \cos\theta. \tag{7}$$

Applying this boundary condition as well as the condition of $0V$ along the inner radius R_1 to the general solution of the Laplace equation in polar form results in a potential given by equation 8. The region outside the circle of radius R_2 is described only by the trivial parallel plates situation given in equation 9 where x is the horizontal distance from the centre.

$$\phi(r, \theta) = (r - R_1) - \frac{V}{R_2 - R_1} \cos(\theta) \text{ for } r < R_2 \tag{8}$$

$$\phi(x) = -\frac{2V(x + R_2)}{L} \text{ for } r > R_2 \tag{9}$$

This solution is only an approximation and its usefulness in comparison with numerical solutions to the potential is limited. For inner grounds with small radii qualitative analysis of the plotted electric fields indicates that the approximation made is accurate up to the condition of $5R_1 : R_2$.

4 The Numerical Methods

This section outlines the ideas behind the various numerical methods considered in the project to solve Laplace's equation. Analysis of each method is deferred until the discussion of the results.

4.1 The Standard Finite Difference (Jacobi) Method

The first of the numerical methods implemented is the standard finite difference (Jacobi) method. It is required that Laplace's equation (5) be solved for a specified grid that represents a discretised version of the space in question. First Laplace's equation is written in terms of a cartesian coordinate system:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0 \qquad (10)$$

The above second derivatives in x and y are written as central difference expressions:

$$\frac{\partial^2 \phi}{\partial x^2} = \frac{\phi_{x+1,y} - 2\phi_{x,y} + \phi_{x-1,y}}{h^2} \qquad (11) \qquad\qquad \frac{\partial^2 \phi}{\partial y^2} = \frac{\phi_{x,y+1} - 2\phi_{x,y} + \phi_{x,y-1}}{h^2} \qquad (12)$$

n.b the size of the cells with which space is discretized is $h = \Delta x = \Delta y$ since the grid is symmetric.

By summing these difference expressions and setting the result equal to zero an expression for the potential (13) can be found for a point in terms of its adjacent points. This is known as the five point stencil and a schematic diagram of this can be seen in Figure 4 where an iteration is being calculated on the central green point and depends on the blue points:

$$\phi_{(x,y)} = \frac{1}{4}(\phi_{x+1,y} + \phi_{x-1,y} + \phi_{x,y+1} + \phi_{x,y-1}).$$

(13)

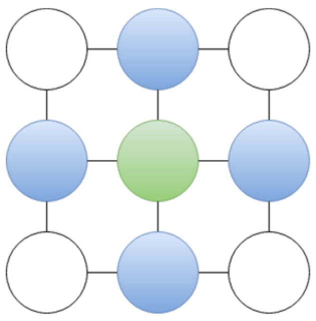


Figure 4: The Five Point Stencil

The basic finite difference method is implemented by looping over all points that are not fixed at the edge of the grid. Such points are assumed constant. A new potential grid is created representing an improved numerical approximation of the solution by applying (13) at each point.

Sometimes it is useful to use periodic boundary conditions to describe a problem, for example in the case where the geometry simulated is part of a repeating motif. In this case the outer boundaries of the image can be left unfixed and the solver will "roll-over" the edges', e.g. if a point on the right-hand edge of an image is being calculated there is no point further to the right, so the solver wraps around and uses the left-most point on the line. This allows the image to form part of a repeating pattern where the specified geometry is the "base" of the pattern as is the case in the stripline problem.

4.2 Improving The Finite Difference Method: Gauss-Seidel Iteration

The Gauss-Seidel Method of iteration is very similar to the standard Jacobi method, (13), except that it does not defer use of updated values until the next iteration but instead uses them in the current iteration. While sweeping across the grid in the usual way (from left to right one row after the other) it is true that for any point that the potentials for two of the four adjacent points, namely $\phi_{x-1,y}$ and $\phi_{x,y-1}$, have already been updated in the current iteration. Using values from the current iteration as opposed to the last iteration as in the standard Jacobi method will in theory halve the storage requirements of the algorithm. This is because half of the stored values are being overwritten every iteration.

4.3 Improving The Finite Difference Method: Red-Black Ordering

The grid is considered to consist of red and black points alternating like black and red squares on a checkerboard. The grid is covered in two passes, first red points and then black points. If a point of one colour is at the centre of a five point stencil then the surrounding four point utilised to update potential are of the opposite colour. From this idea it is clear that no red points are required to calculate the update of any red point. The same is of course true for black points and as such the values for all points of one colour can be computed in parallel.

This method should take less time than the Jacobi method because when the red points are calculated the black ones need not be updated and may instead calculate the next step. This means that each time the algorithm switches from red to black one iteration is skipped, in theory allowing for this method to converge in half the runtime of that of the Jacobi method.

4.4 Matrix Inversion Method

The method of matrix inversion also uses the five point stencil model. It stores the information specified by the image as a system of linear equations:

$$Ax = b \tag{14}$$

A is a coefficient matrix, b is a vector containing all known initial values specified by the input image and x is a vector containing the solution of the potential at all points which can be obtained by inverting A:

$$x = A^{-1}b \tag{15}$$

The method moves through the grid point by point and orders the information following three conditions:

1. known potential at current point (Dirichlet)
2. known differential with respect to time at a current point (Neumann)
3. neither of the above

In this project only electrostatic problems are considered and as such there are no Neumann conditions. A Dirichlet condition at a point results in a 1 being placed in the corresponding position in matrix A. The known voltage is placed into the corresponding position in b. If potential is not known at a point then the five point stencil is used: a -4 is placed at the current position in A and a 1 is placed at each of the four adjacent positions. A 0 is placed at the corresponding point in the b vector. Once all points are assigned the coefficient matrix A is inverted using *Eigen* [5] and then multiplied with the known vector b to calculate all potentials in x at once. Only square matrices are invertible and as a result the method can only solve for pixel perfect square images.

5 Implementing the Methods

It is at this point in the report that we introduce the approach chosen to designing software that is able to implement the above methods in order to solve the problems at hand. The program, henceforth "Gridle", needs to be able to solve a limited number of operations. These can be resumed as:

1. Solving the potential in a charge-free region of space using various numerical solvers.
2. Providing the analytic solution (or approximation), to the first two problems presented.
3. Comparing various solutions.

Only a limited number of "recipes" combining these "ingredients" are of use, so these are individually implemented as a simple finite state machine. Each variant of this finite state machine is a pipeline that performs all of the useful work. An example pipeline is shown in Figure 5.

Command line parameters are parsed with the help of the TCLAP library [2]. This technique is based on the idea of pushing data through a pipeline of functions, each performing a new operation on the previous results. If at any stage a non-recoverable error is raised, the pipeline is aborted and the program is exited with error messages. Recoverable errors are non-fatal, and are signaled in the code by the use of Option types, types that may or may not contain a value, since some functions may not always return a value.

This Option type is roughly equivalent to the Maybe monad in Haskell [3]. As each of the solvers can be used within each pipeline they become part of the state variables of each pipeline and present a similar interface. This is due to the reasoning that since each solver takes in the same set of data and should output a similar solution then it should be callable in a similar way –they should be approximately black boxes as far as the user is concerned. It is important to note, however, that some of the solvers have additional limitations that the user must be aware of, for example, the matrix inversion solver can only solve square matrices and cannot handle periodic boundary conditions.

To allow the user to compare methods against each other the program emits timing information about each of the major function it runs (millisecond accuracy). This can be combined with the error information to aid in analysing methods.

5.1 Images as Input

Gridle was designed to be easy to configure, using plain text files in the JSON (JavaScript Object Notation) format for the most part. This is not, however, an easy way to specify the geometry of the problem to be evaluated, so for this images in the PNG format were chosen. These can easily be designed in any image editor and there is a mapping from the colours used to the voltages they represent in the JSON configuration file. Colours can also represent certain other boundary conditions such as a linear interpolation between two specified fixed voltages. These images are loaded using Sean T. Barrett's `stb_image` library [4].

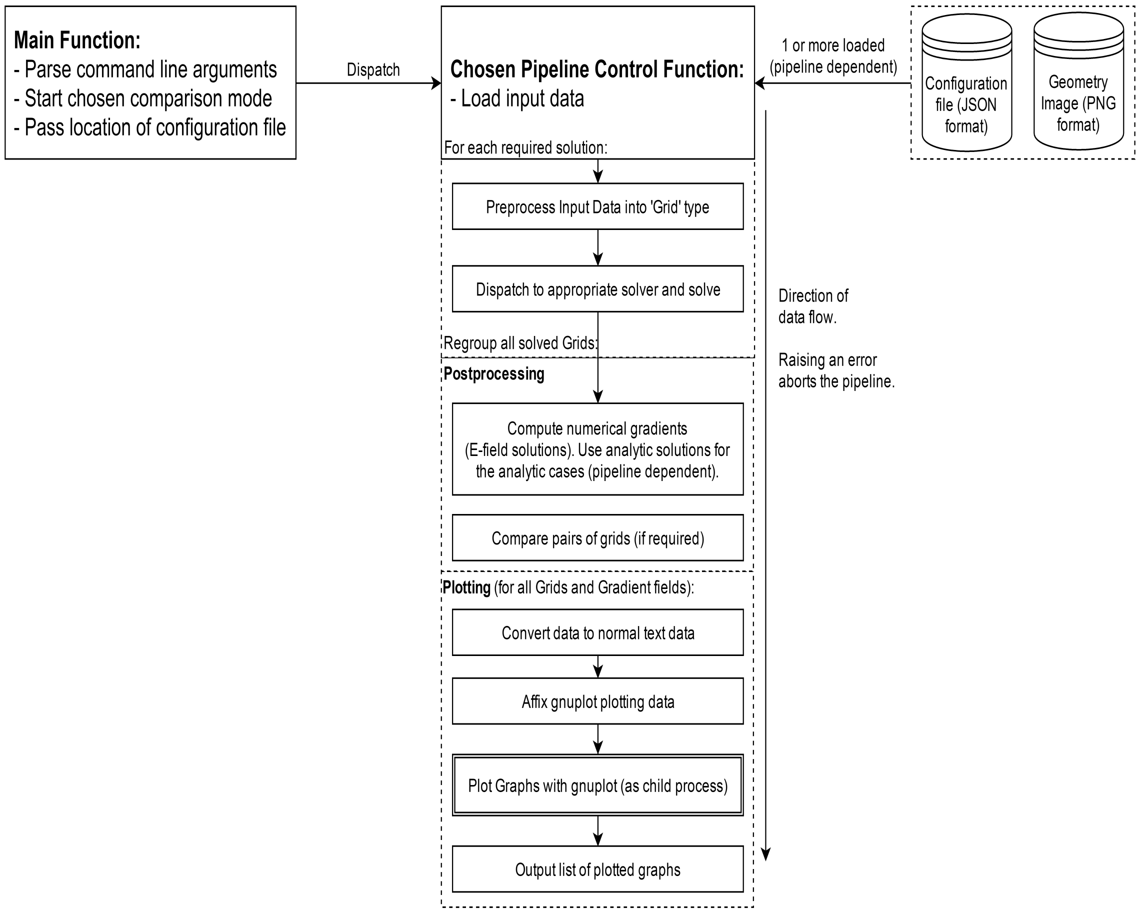

Main Function:
- Parse command line arguments
- Start chosen comparison mode
- Pass location of configuration file
Dispatch
Chosen Pipeline Control Function:
- Load input data
1 or more loaded (pipeline dependent)
Configuration file (JSON format)
Geometry Image (PNG format)
For each required solution:
Preprocess Input Data into 'Grid' type
Dispatch to appropriate solver and solve
Regroup all solved Grids:
Direction of data flow.
Raising an error aborts the pipeline.
Postprocessing
Compute numerical gradients (E-field solutions). Use analytic solutions for the analytic cases (pipeline dependent).
Compare pairs of grids (if required)
Plotting (for all Grids and Gradient fields):
Convert data to normal text data
Affix gnuplot plotting data
Plot Graphs with gnuplot (as child process)
Output list of plotted graphs

Figure 5: Program Flow Pipeline

5.2 Optimisation

The basic Jacobi finite difference method (with and without periodic boundary conditions) was the first method implemented. The performance of this method was much slower than desired and as such various optimisations were implemented.

The parts where the code spends a long time (namely the solvers) have been extensively profiled and optimised (primarily using the 'perf' profiler on Linux). The initial optimisation performed was to thread the main loop. Since all of the points depend only a previous grid (which is not modified during this process) all of their updated values can be calculated in parallel and placed into a new grid data structure.

The next optimisation came from noticing that the periodic boundary conditions (requiring branching to check if an iteration at the edge of the grid is being calculated) was a costly operation. Separating it out into a separate function that is only called when the image does not present completely fixed boundary conditions around the edges saves considerable time. Along the same lines as the previous optimisation, reducing the frequency at which the maximum relative change of the grid from one iteration to the next is calculated will significantly reduce the run-time of the algorithm.

Since this maximum relative error still needs to be checked, a solution was devised where the error is only checked every N iterations where N is a reasonably large number (500-1000). This is achieved by using compiler intrinsics present in gcc and clang to force a conditional to automatically continue with a selected branch (one that doesn't calculate the error). Then every N iterations there is a branch mis-prediction but this is on a slow branch on which the relative error change is calculated. Even if too many iterations are performed, testing has shown that removing the error calculation from the central loop increases its speed by a factor of 10^2. There would need to be a lot of additional iterations calculated to lose the gains of infrequently calculating the error.

An additional optimisation can also be performed from the above. Plotting the relative error against the number of iterations performed yields an approximately $\frac{1}{N^2}$ convergence rate. This can be used from a number of iterations with a known error quantity to predict the number of iterations required for the desired error. Instead of just calculating up to the calculated number of iterations the error is calculated several times in between to refine the required number of iterations. This is in part due to the method sometimes overshooting when a very small relative error is required.

All of the methods derived from the basic Jacobi finite difference method are built on the above model of optimisations presented above although only the Jacobi finite difference method uses the additional error prediction step since the convergence factors of the other methods were not calculated. Instead these methods infrequently calculate the error.

5.3 Note on the Matrix Inversion solver

Matrix inversion is the only method that is not derived from the finite difference method. This was implemented as described in Section 4.4 using the *Eigen* C++ library to do the matrix manipulations [5].

5.4 The Graphical User Interface

A graphical user interface (GUI) was also created for Gridle. This was built using Github's electron [6] allowing for it to be built using modern web technologies, namely HTML5 and JavaScript which are tools specifically designed for building web pages and thus user interfaces. The amount of code required to produce a good GUI with these technologies was significantly reduced relative to the amount required with normal C++ GUI libraries. The GUI runs as a separate process and spawns an instance of the C++ backend as a child process, communicating over a POSIX pipe. This avoids the need to replicate functionality present in the backend in the GUI program whilst also keeping the GUI optional. Screenshots of the GUI can be seen in Appendix C.

All of the code for Gridle is available to view and download at our public repository on github [7].

6 Results and Discussion

Due to the similarity of results between difference methods here only the result graphs for the method of red-black ordering are shown. Comparison between the methods is deferred until the next section.

6.1 Numerical Solutions

6.1.1 Concentric Cylinders

The result for potential for the set up of an inner and outer cylinder at 0V and +10V respectively shown in Figure 6 looks as expected and holds up well compared to the analytical solution as shown in Figure 7. This graph shows the absolute difference between the numerical solution grid and a grid filled with the analytical solution for an identical concentric cylinder set-up. The red regions around the inner cylinder are a consequence of the fact that the numerical solver is working on an image that consists of square pixels. It is of course impossible to form an exact circle from square pixels and as such there are discrepancies around the centre of the circle in the comparison to an analytical solution that has assumed a perfect circle. Regions of most extreme discrepancy represent a difference of only 0.45V which is of minor significance considering the outer cylinders is at potential of 10V.

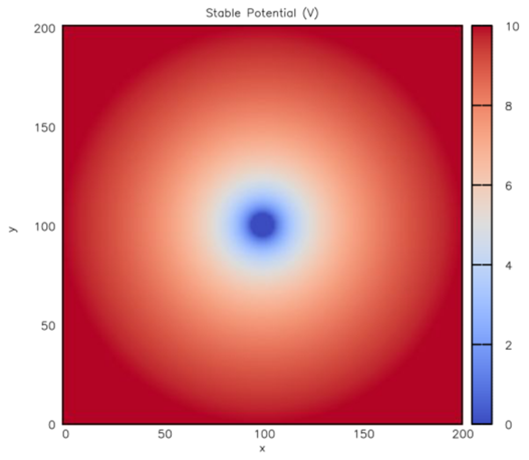


Figure 6: Potential

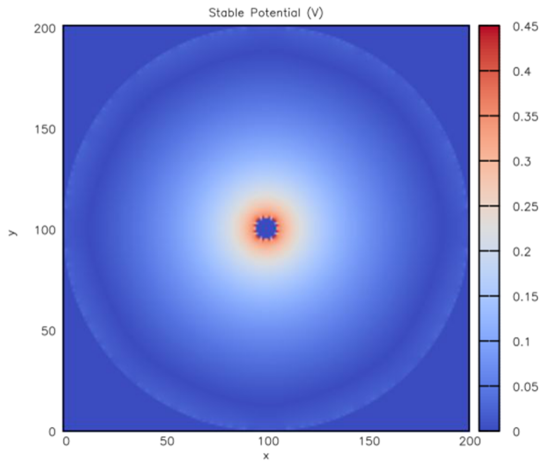


Figure 7: Difference vs Analytical

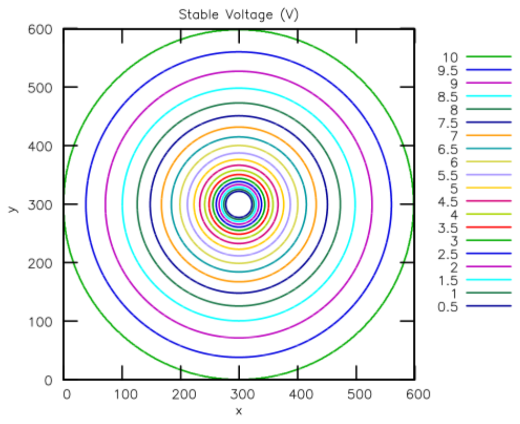


Figure 8: Lines of Equipotential

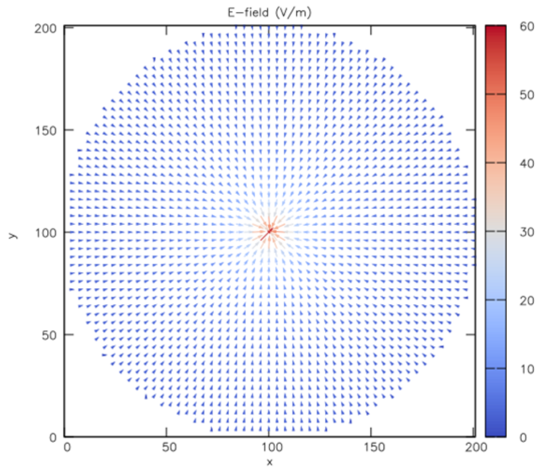


Figure 9: Electric Field Lines

Figures 8 and 9, showing Lines of Equipotential and Electric Field Lines, look as expected and are consistent with each other.

6.1.2 Ground Cylinder Between Parallel Plates

Figure 12 shows the solution for potential for a cylinder at ground centred between a plate at 10V to the left and a plate at -10V to the right. The fact that the line in the central region between the two plates (including of course the circumference of the cylinder) is at 0V is much easier to see from the lines of equipotential as shown in Figure 17.

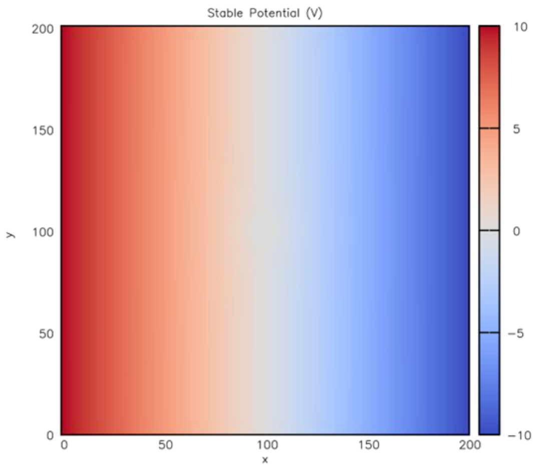


Figure 10: Potential

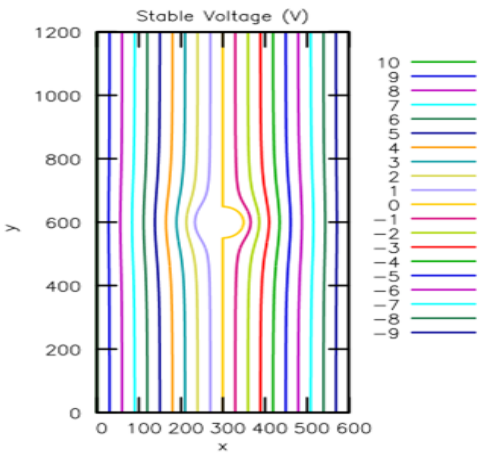


Figure 11: Lines of Equipotential

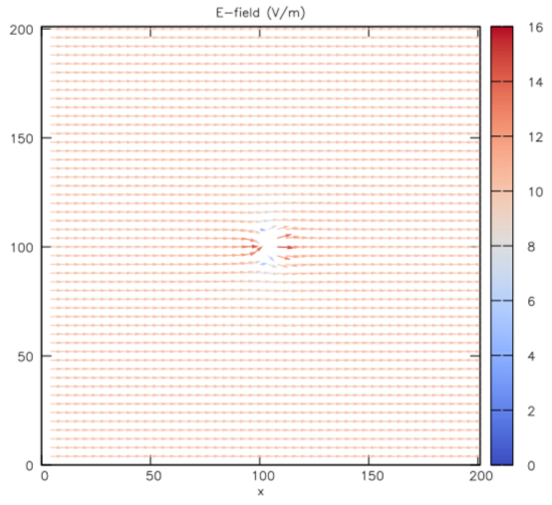


Figure 12: Electric Field Lines

The field lines shown in Figure 12 above converge to the ground on the left and diverge from the ground on the right as expected and confirm that one of the approximations in the analytical solution, i.e that the field behaves as if only the plates were present at further distances from the cylinder, is not inaccurate at least from inspection of the graph. The overshoot of the arrows into the central ground occurs because gnuplot draws an arrow of a certain size starting at the position of the point at which the value for electric field has been calculated. At points near the ground cylinder where the electric field is greatest these arrows are of a length such that their front ends overshoot into the ground cylinder region.

6.1.3 Edge Coupled Stripline

The results below are that for the edge coupled stripline. It is assumed that the edge coupled stripline is part of a repeating motif and as such the outer boundaries are unfixed and the solver wraps around the edges of the images as described in Section 4.1. As before, details of the solution become apparent in the graph of Lines of Equipotential, figure 14.

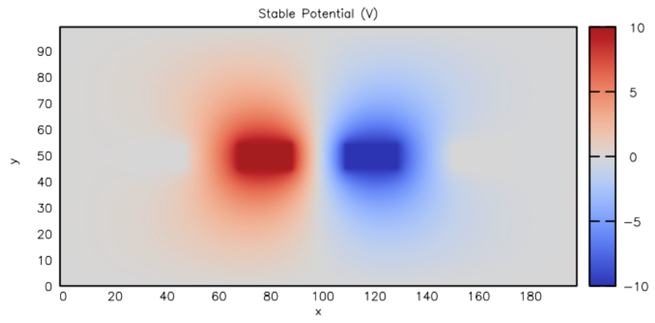


Figure 13: Potential

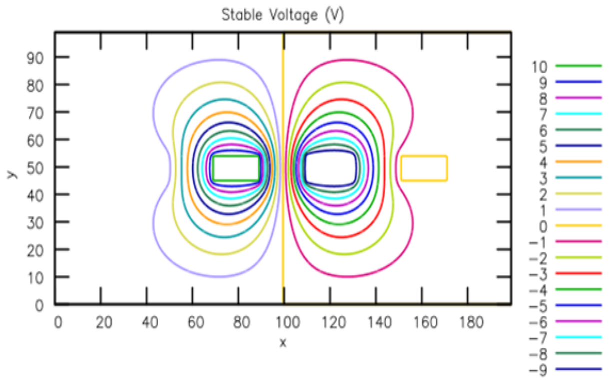


Figure 14: Lines of Equipotential

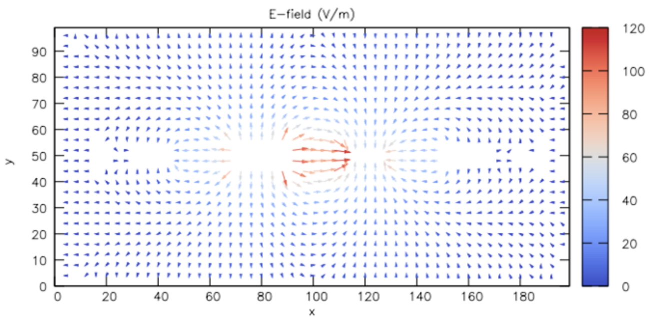


Figure 15: Electric Field Lines

The electric field lines shown above in Figure 18 look as expected, diverging from regions of higher potential and converging to regions of lower potential.

6.2 Comparison Between Methods

As previously stated each method, provided that it is able to be used for the image in question, arrives at a very similar result. The graphs below show the absolute difference between the solution returned by the red-black method and the solutions returned by the other three methods for the concentric cylinders set up.

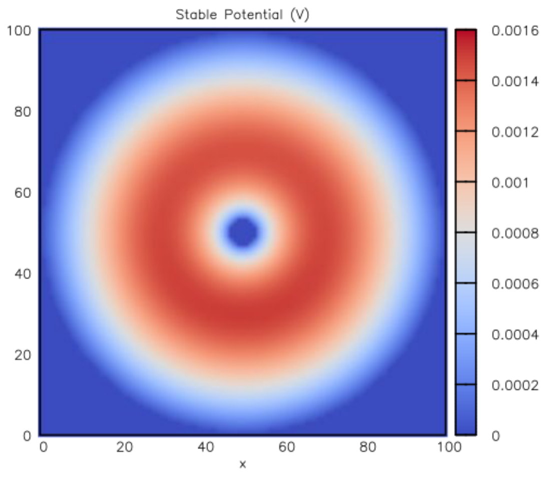


Figure 16: vs Standard Finite Difference Method

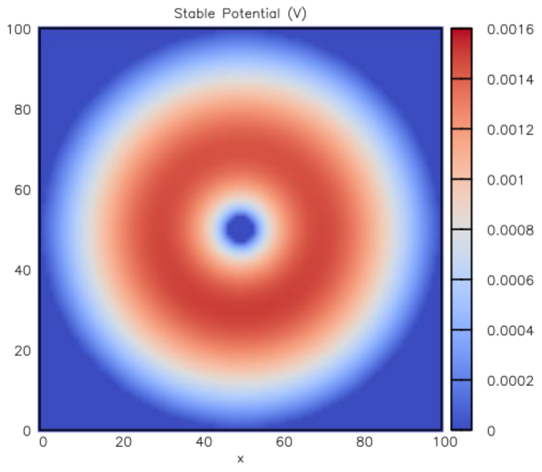


Figure 17: vs. Gauss-Seidel Method

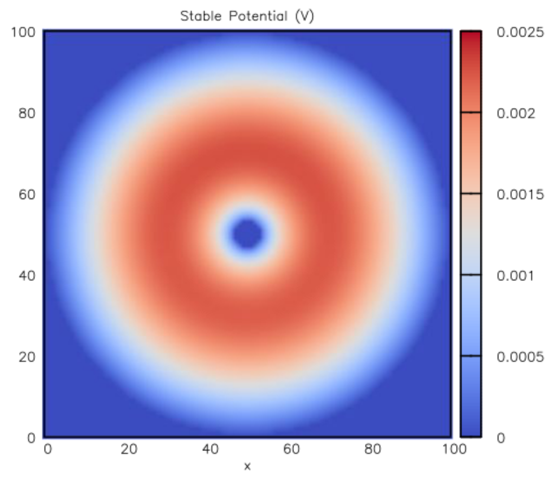


Figure 18: vs Standard Finite Difference Method

Comparisons are for a specified maximum relative error of 10^{-6}. Maximum error is defined such that the program continues to iterate until the error is smaller than the specified value, where the error is defined as:

$$RelError = \frac{NewVal - OldVal}{NewVal} \tag{16}$$

where $NewVal$ and $OldVal$ refer to values of potential at a point across two successive iterations. On the whole the solutions returned by different methods are incredibly similar. It is notable that the difference between solutions of different methods coincide less further from the boundary conditions, however even at its most extreme the difference is 3 orders of magnitude less than the magnitudes of the voltages of the outer cylinder.

Now the performances of the methods solving for the edge coupled stripline are compared. The matrix inversion method is omitted from comparisons because the method is carried out in one step (the inversion of a matrix) as opposed to over a number of iterations. Additionally it is only suitable for situations represented by a perfectly square image and as such is not useful for general set-ups such as that of the edge coupled stripline.

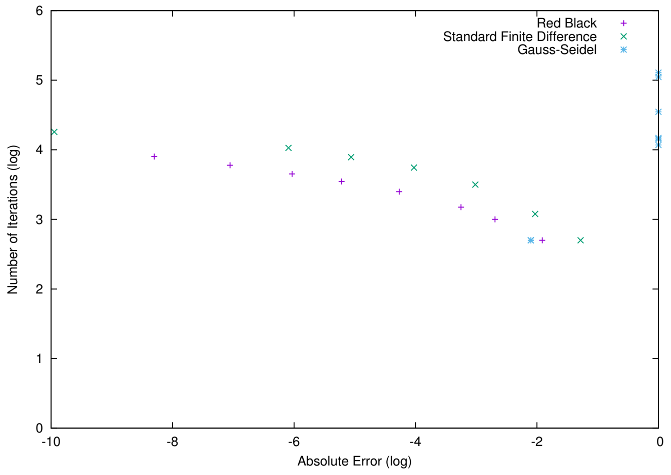


Figure 19: log of Number of Iterations vs log of Absolute Error

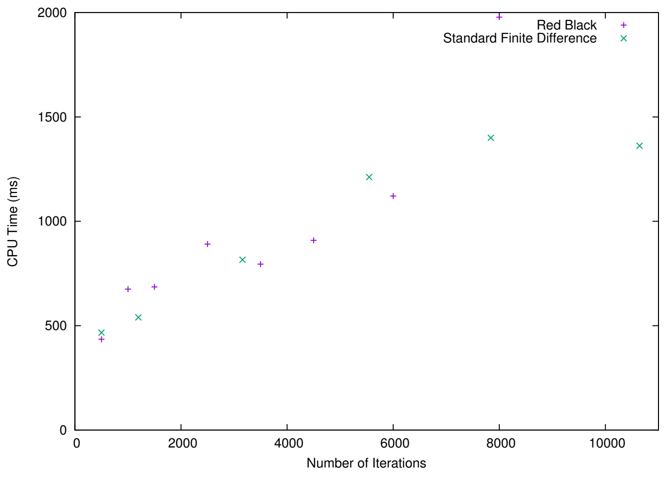


Figure 20: Number of Iterations vs CPU Time

Figure 19 displays the performance of each method in terms of iterations taken to converge. The trends approximately follows the expected $\frac{1}{N^2}$ relation. The matrix inversion method is not graphed as it takes incomparably longer than the other methods due to the size of the matrix that is to be inverted (size of the matrix scales as N^4 where n is the number of pixels in the image. The red-black method in general takes fewer iterations than the other methods to converge as expected.

In terms of CPU time there is no clear winner between the red-black and Jacobi methods in a simulation for the image we use to describe the edge coupled stripline (See Appendix B). The benefits of red-black outlined in Section 4.3 suggest that it should be faster in terms of CPU time which is not the case for this specific simulation.

That red-black is not faster in this instance is due to the algorithm's internal overhead. The image used for the stripline is not a high resolution image and as such there are not a great number of points in the grid. The Jacobi method splits the points in the grid into chunks of 10k so that each thread has a linear region to work on. In this case a maximum of 2 threads would be used regardless of how many are available. Each thread then has a linear, contiguous chunk of 10k points to iterate over.

In the red-black case the same chunking mechanic is used to limit the size of the thread pool however actually looping over the points is not as optimised. Instead of the linear region given to each thread the grid is looped over in parallel. The difference is that the threads don't necessarily get contiguous points to work on and unlike the standard finite difference method case the switch to a new point incurs a significantly greater penalty than rolling over to the next index of the loop. As the number of points increases the context switching penalty tends to average out - for a higher resolution image representation of the concentric cylinders situation the red-black Method takes half the CPU time compared to the Jacobi method.

The CPU time for the Gauss-Seidel method is not graphed because it is not run on multiple cores and such a comparison would be an unfair one. This falls down to a data race issue in that when the grid is divided up amongst cores there is no guarantee that the $x-1$ and $y-1$ points for a point at the boundary of one of these divisions has been updated already during this iteration. The method will still converge but there is no guarantee that the Gauss-Seidel Method has been used properly for every point. If the $x-1$ and/or the $y-1$ points have not been updated this iteration then the Gauss-Seidel method reduces to the standard finite difference method (or even a combination of the two methods if one of the points has been updated already and one has not) and as such convergence information is not guaranteed to be accurate for the method.

7 Conclusion and Outlook

Gridle fulfils the aims of the project in that it is able to return solutions for both potential and electric field for a given geometry of electrodes. Returned solutions for potential are compared to and found to be consistent with the known analytical solution for the concentric cylinders setup. Of the various numerical methods included in the software the red-black method converges in fewest iterations and shortest CPU runtime provided that the resolution of images is high enough. For lower resolution images the Jacobi method may converge quicker in terms of runtime. As such red-black is the recommended method for all situations except perhaps for the case of lower resolution images in a situation where CPU runtime is of importance. The Gauss-Seidel method and matrix inversion methods are far slower in terms of CPU runtime with the latter being by far the least useful of the methods in that it requires a pixel perfect square geometry to return a solution.

The project provides a good example of the power of scientific computing, namely solving differential equations such as Laplace's equation when they cannot be solved analytically.

The most obvious extension to the project would involve making the red-black method consistently the most efficient solver for all problems such that it could be implemented as the exclusive method. This could be achieved by forcing it to run single threaded for any image smaller than an image size at which the Jacobi method is faster. Additionally the matrix inversion method could be improved using the SparseMatrix class in *Eigen* to represent the coefficient matrices that are mostly zero that are to be inverted. This might bring the runtime into the region of the finite difference methods but of course the requirement that the input image be a perfect square is still present.

As a final point, the interface could easily be made available online so as to make it an open source tool that can be used remotely and extended by anyone who may wish to do so.

References

[1] Eklund, L. Theoretical Physics 3 Project Description. 2016

[2] http://tclap.sourceforge.net (MIT license)

[3] https://wiki.haskell.org/Maybe

[4] https://github.com/nothings/stb (Public Domain)

[5] http://Eigen.tuxfamily.org/index.php?title=MainPage (MozillaPublicLicense2)

[6] https://github.com/atom/electron (MIT license)

[7] https://github.com/Goobley/TheoryGroupProject

A Analytical Solutions

In order to solve for our situation (Figure 2) that demonstrates circular symmetry Laplace's equation is first converted to polar form:

$$\vec{\nabla}^2\phi = \frac{\partial^2\phi}{\partial r^2} + \frac{1}{r}\frac{\partial\phi}{\partial r} + \frac{1}{r^2}\frac{\partial^2\phi}{\partial\theta^2} = 0 \tag{17}$$

As there is no θ dependence for this situation the term containing the second derivative of ϕ with respect to θ disappears and the differential equation, no longer partial, becomes:

$$\vec{\nabla}^2\phi = \frac{\partial^2\phi}{\partial r^2} + \frac{1}{r}\frac{\partial\phi}{\partial r} = 0 \tag{18}$$

and is now easily solvable by first introducing a function to reduce order:

$$\lambda(r) = \frac{\partial\phi}{\partial r} \tag{19}$$

to rewrite the differential equation as follows:

$$\lambda' + \frac{1}{r}\lambda = 0. \tag{20}$$

This is trivial to solve for λ and thus first derivative of potential, i.e:

$$\lambda(r) = \frac{\partial\phi}{\partial r} = ke^{-\ln|r|} = \frac{k}{r} \tag{21}$$

where k is some constant.

$$\frac{\partial\phi}{\partial r} = \frac{k}{r} \tag{22}$$

The first order differential equation (11) is now trivial to separate and solve and we arrive at a general solution for potential:

$$\phi = k\ln|r| + C \tag{23}$$

Using the fact that potential is 0 at R$_i$ and V at R$_o$ as boundary conditions gives the particular form of potential as (6).

B Sample Input Images and Key

Colour	Description
Red	+V
Blue	-V
Black	Ground
White	Region to be solved for
Green	Boundaries

Figure 21: Table describing the colour coding of the diagrams

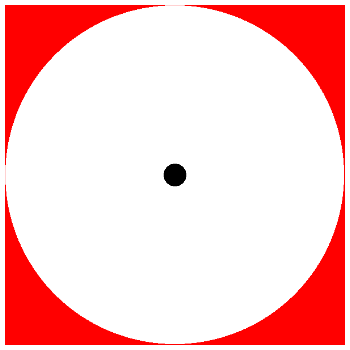


Figure 22: The input image for concentric cylinders.

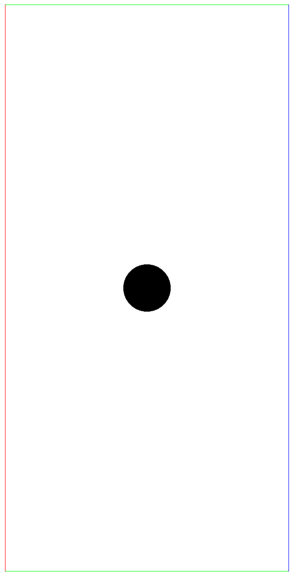


Figure 23: The input image for ground cylinder between parallel plates.

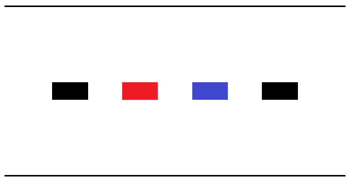
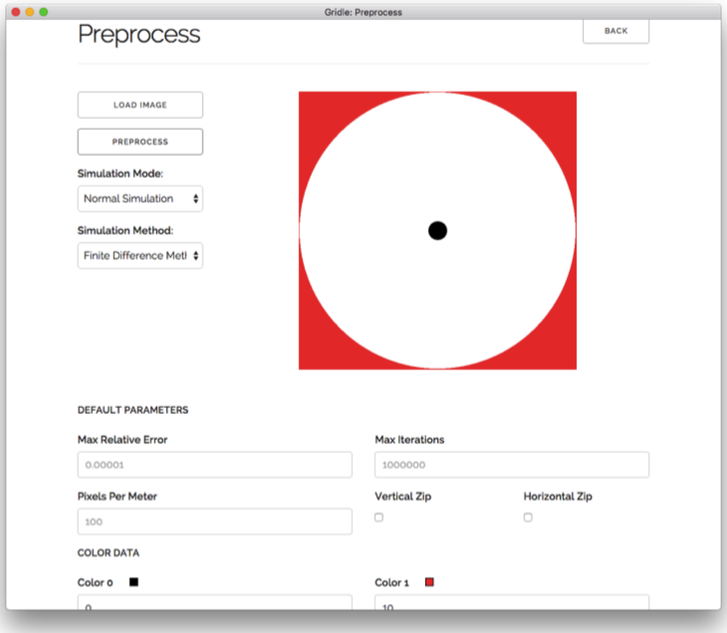


Figure 24: The set up for the edge coupled stripline. The absence of boundaries instruct the program to wrap around from one side to the other.

C Graphical User Interface Screenshots



Simulation Mode: Normal Simulation

Simulation Method: Finite Difference Metl

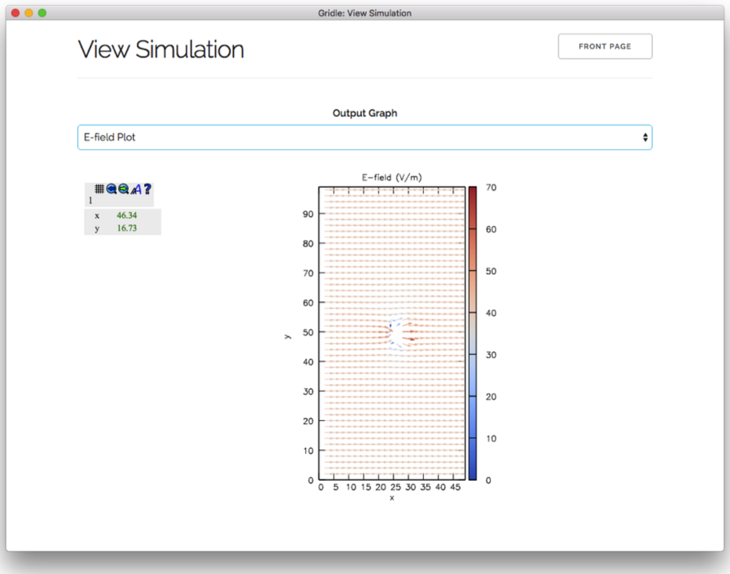
DEFAULT PARAMETERS

Max Relative Error	0.00001
Max Iterations	1000000
Pixels Per Meter	100
Vertical Zip	☐
Horizontal Zip	☐

COLOR DATA

Color 0 ■	0
Color 1 🟥	10

Figure 25: Preprocess Screen



Output Graph: E-field Plot

Figure 26: Sample Output