Simulation of Electric Fields and Non-Neutral Plasmas

MPhys Dissertation

Thomas Lake
527562@swansea.ac.uk

May 24, 2012

Abstract

This project has looked at numerical methods of calculating the electric field for arbitrary arrangements of electrodes, with a view to using this data to calculate the distribution of non-neutral plasmas. These techniques could then be applied by other ongoing projects to calculate, for example, the effect of various electrode arrangements on the distribution of positrons within the positron accumulator and Penning trap in use in the department.

The results of this project are somewhat mixed. The electric field calculations performed by Dynamic_OR (one of the programs written for this project) compare very favourably to results obtained from a widely used piece of commercial software, SIMION. The difference between the two sets of results for a number of sample electrode arrangements is typically less than 1V and correspond to a percentage difference of $< 0.1\%$ in several cases.

The attempts to calculate the distribution of plasma within these electric fields using numerical methods has been less successful, with no reliable results obtained. The program written for this task, FLTE, tends to converge on a solution very similar to the electric field which yields plasma distributions clustered around electrodes rather than centred around the centre of the cylinder defined in the example used.

Simulation of Electric Fields and Non-neutral Plasmas

The general goal of this project has been to create a tool (or set of tools) to calculate the density distribution of a single component non-neutral plasma confined in an electric field. This was broken down into the following smaller tasks.

- Write a tool to calculate the electric field for given arrangements of electrodes
- Compare the results of this tool with output from SIMION
- Solve and compare sample electric fields in Mathematica
- Model the behaviour of charges/plasma in these fields
- Calculate the charge/density distribution for plasma in electric fields

This dissertation will outline the progress made in these areas over the course of the project, along with the necessary background information, analysis and ways that this project could be continued or expanded upon in the future.

There are a growing number of experiments working with matter that can be described as plasma of one variety or another, ranging from the anti-hydrogen experiments at ALPHA and ATRAP to work at ITER investigating nuclear fusion. All of these projects have a need to predict the behaviour of the plasmas that form part of the experiment, using analytical or numeric methods to calculate the distribution of the plasma at different points in the experiment at different times.

The work presented in this dissertation presents a very small set of examples covering the calculation of the electric fields around electrodes and moving on to look at the density distribution of thermal electron plasma.

1 Background Information

1.1 Electric fields

In physics, a field may be defined as an area where an object will experience a force proportional to the strength and direction of the field at that point and some property of that object. An object can be acted on by more than one force at a time, with the same effect as if the object were acted on by a single force equal to the vector sum of all of the forces acting on the object. Giving each force the index i, this can be written simply as $F_{net} = \sum_i F_i$. Given that that the magnitude of a field is proportional to the force, the principal of linear superposition of forces implies that a field due to multiple sources is equivalent to the sum of the fields due to each source individually.[1]

$$\vec{F} = kq_1q_2\frac{\vec{x}_1 - \vec{x}_2}{|\vec{x}_1 - \vec{x}_2|^3} \tag{1.1}$$
$$\text{where } k = \frac{1}{4\pi\epsilon_0}$$

In the case of an electric field, the force experienced by an object placed in the field is due to the Coulomb force given in 1.1[1], with the field being equal to the force acting per unit electric charge on the object. The electric field $\vec{E}$ due to a number of point charges can then be calculated using the vector sum of the force, $\vec{F}$ per charge q, denoting each charge using the index i as shown in equation 1.2.

$$\vec{E}(\vec{x}) = k\sum_i q_i\frac{\vec{x} - \vec{x}_i}{|\vec{x} - \vec{x}_i|^3} \tag{1.2}$$

For many small charges in a large volume, the sum can be rewritten as an integral over the volume, as shown below in equation 1.3, where $\rho(\vec{x})$ represents the charge density at point $\vec{x}$:

$$\vec{E}(\vec{x}) = k\int \rho(\vec{x}')\frac{\vec{x} - \vec{x}'}{|\vec{x} - \vec{x}'|^3}\,\mathrm{d}^3\vec{x}' \tag{1.3}$$

Electrodes

As the force acting on charged particles is dependent on the strength of the field at their current position, the strength and shape of this field can be chosen to manipulate the trajectory of the charged particles, allowing us to trap, focus and accelerate the particles as required. Treating electrodes as collections of charges and analysing the field around each would not be a sensible approach. It is easier to consider the potential of each electrode and the contribution of that potential to the electric field.

It can be shown that, for an arbitrary closed surface S, that the integral over the surface can be equated to the sum of charges enclosed by that surface (Eq. 1.4) or to the integral of the charge density ρ over the enclosed volume (Eq. 1.5). This is Gauss' law[1].

$$\oint_S \vec{E}\cdot\hat{n}\,\mathrm{d}a = \frac{1}{\epsilon_0}\sum_i q_i \tag{1.4}$$

$$\oint_S \vec{E}\cdot\hat{n}\,\mathrm{d}a = \frac{1}{\epsilon_0}\int_V \rho(\vec{x})\,\mathrm{d}^3x \tag{1.5}$$

Applying the divergence theorem (Shown for arbitrary vector field $\vec{A}$ in equation 1.6) to equation 1.5 yields the result shown in equation 1.7. Setting the integrand to zero, which is permissible due to the arbitrary volume V, yields equation 1.8, which states that the divergence of the electric field is proportional to the charge density, ρ.

$$\oint_S \vec{A}\cdot\hat{n}\,\mathrm{d}a = \int_V \nabla\cdot\vec{A}\,\mathrm{d}^3x \tag{1.6}$$

$$\int_V \left(\nabla\cdot\vec{E} - \frac{\rho}{\epsilon_0}\right)\mathrm{d}^3x = 0 \tag{1.7}$$

$$\nabla\cdot\vec{E} = \frac{\rho}{\epsilon_0} \tag{1.8}$$

In order to completely specify the vector field, we need to specify the curl at every point in space in addition to the divergence specified by 1.8[1]. The curl of an Electric field is given in Maxwell's equations as $\nabla\times\vec{E} = 0$. Applying this to equation 1.5 yields an equation for the electric field in terms of the gradient of a scalar function, shown in equation 1.9. Given that the curl of the gradient of a scalar function is zero, this immediately satisfies Maxwell's equation.

$$\vec{E}(\vec{x}) = k\,\nabla\int\frac{\rho(\vec{x}')}{|\vec{x}-\vec{x}'|}\,\mathrm{d}^3x' \tag{1.9}$$

We can name this function Ψ:

$$E = -\nabla\Psi \tag{1.10}$$

$$\Psi(\vec{x}) = k\int\frac{\rho(\vec{x}')}{|\vec{x}-\vec{x}'|}\,\mathrm{d}^3x' \tag{1.11}$$

Combining equations 1.8 and 1.10 gives us a partial differential equation for Ψ - Poisson's Equation (Eq. 1.12). If the charge density is zero in an area then Poisson's equation can be simplified to the form shown in equation 1.13 - this is known as Laplace's equation[1].

$$\nabla^2\Psi = \frac{-\rho}{\epsilon_0} \tag{1.12}$$

$$\nabla^2\Psi = 0 \tag{1.13}$$

Bessel Functions

As described in [2] and [1], the Laplace Equation in cylindrical coordinates can be solved using a particular family of mathematical functions called Bessel Functions. For an equation of form

$$\frac{\mathrm{d}^2R}{\mathrm{d}x^2} + \frac{1}{x}\frac{\mathrm{d}R}{\mathrm{d}x} + \left(1 - \frac{\nu^2}{x^2}\right)R = 0 \tag{1.14}$$

the solutions to this equation form the Bessel Functions, specifically the Bessel Functions of order ν[1]. The two Bessel Functions of the first kind of order $\pm\nu$ can be written as:

$$J_\nu(x) = \left(\frac{x}{2}\right)^\nu\sum_{j=0}^{\infty}\frac{(-1)^j}{j!\Gamma(j+\nu+1)}\left(\frac{x}{2}\right)^{2j} \tag{1.15}$$

$$J_{-\nu}(x) = \left(\frac{x}{2}\right)^{-\nu}\sum_{j=0}^{\infty}\frac{(-1)^j}{j!\Gamma(j-\nu+1)}\left(\frac{x}{2}\right)^{2j} \tag{1.16}$$

If $\nu \in \mathbb{Z}$ however, these two functions are not independent and a Bessel Function of the second kind is used:

$$N_\nu(x) = \frac{J_\nu(x)\cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)} \tag{1.17}$$

If, instead, the original equation (Eq. 1.14) takes the form

$$\frac{\mathrm{d}^2R}{\mathrm{d}x^2} + \frac{1}{x}\frac{\mathrm{d}R}{\mathrm{d}x} - \left(1 + \frac{\nu^2}{x^2}\right)R = 0 \tag{1.18}$$

then the solutions are referred to as Modified Bessel Functions of the first or second kind:

$$I_x(x) = i^{-\nu}J_\nu(ix) \tag{1.19}$$

$$K_\nu(x) = \frac{\pi}{2}i^{\nu+1}H_\nu^{(1)}(ix) \tag{1.20}$$

$$\text{Where } H_\nu^{(1)}(x) = J_\nu(x) + iN_\nu(x) \tag{1.21}$$

$$H_\nu^{(2)}(x) = J_\nu(x) - iN_\nu(x) \tag{1.22}$$

The two functions H_ν (1.21, 1.22) are the Hankel Functions[1], and form another set of solutions to equation 1.14.

1.2 Computational background

Using the information above, we have managed to break the difficult task of calculating the electric field for arbitrary arrangements of electrodes into a smaller task - solving a differential equation. In the case where we are calculating the electric field due to electrodes rather than a distribution of charges, the simpler Laplace equation can be used (Eq. 1.13).

In order to simplify the calculation of the appropriate potential, it is possible to approximate the solution by calculating the value of the potential at a number of discrete points. In this instance, the potential can be calculated at points on a uniform two or three dimensional grid using an iterative approach [3, 4]

SIMION is a commercial software suite that can be used to calculate the electric field resulting from an arbitrary arrangement of electrodes and simulate the trajectory of charged particles through that field[4]. It does this using an iterative finite difference method, which will be described in more detail below. As SIMION is widely used software, it will be used to calculate and compare results from code written for this project to calculate electric fields.

Iterative finite difference method

$$\Psi_{x,y} = \frac{1}{4}\left(\Psi_{x-1,y} + \Psi_{x+1,y} + \Psi_{x,y-1} + \Psi_{x,y+1}\right) \tag{1.23}$$

For a uniformly distributed two dimensional grid of discrete points, the Jacobi/Gauss-Seidal iteration for a point can be written as shown in 1.23[3]. Every value at each point on the grid is set to the average value of the points immediately adjacent to it in the $\pm x$ and $\pm y$ directions. In the three dimensional case, this would be extended to include the points immediately adjacent in the $\pm z$ directions. This is done for each point on the grid in every iteration of the calculation, with the calculation terminating when a certain convergence criteria is met.

The difference between these two methods is when the value at each point is changed. The Jacobi method changes the value at each point after calculating the new values for all the other points on the grid - this means that each point is calculated based on the state of the grid at the start of that iteration. The Gauss-Seidal iterative method changes the value at each point as soon as it has been calculated. This produces faster results than the Jacobi method but can be further improved by using an over-relaxation method[3], which will be described below. From a practical point of view, the Gauss-Seidal method requires less memory during computation, as there is no need to copy the array of points in order to have the original value of each point and its neighbours available throughout each iteration.

Over-relaxation methods allow faster convergence by making larger changes to the value at each point on the grid. Rather than setting the value at each point to the average of its neighbours as described above this average value is used to calculate the size of the change to be applied to the current value, as outlined in equation 1.24[3].

$$\Psi_{x,y}^{new} = \Psi_{x,y}^{current} + R \times \left(\frac{1}{4}\left(\Psi_{x-1,y} + \Psi_{x+1,y} + \Psi_{x,y-1} + \Psi_{x,y+1}\right)\right) \tag{1.24}$$

When using over-relaxation methods, it is necessary to choose a suitable value for the relaxation constant, R. Setting R to values less than 1 correspond to an under-relaxed iteration that will take longer to converge than the regular Gauss-Seidal iteration (for $R = 1$ the two $\Psi_{x,y}^{current}$ in equation 1.24 cancel to yield equation 1.23). Values of $R > 2$ are unstable[3], so it is necessary to use values of R that lie between 1 and 2. In the remainder of this document the convention from SIMION will be used, which defines an over-relaxation factor f as $R = 1 + f$, with $0 \leq f < 1$

A dynamic over-relaxation method

SIMION uses an over-relaxation method, as described above, with a dynamically adjusted over-relaxation factor, f. The over-relaxation factor is adjusted as described in figure 1.1, where mc represents the maximum change in value at a single point during the last iteration and $goal$ is the target precision - the over-relaxation method terminates when $mc < goal$ is true at the end of an iteration. This algorithm is designed to reduce the time required to converge on a solution for a given problem, while addressing some of the inaccuracies and errors that can arise with larger values of the over-relaxation factor[5].

For over-relaxation factor f_i, where i is the current iteration:

- $f_1 = 0.4$
- $f_i = 0.67$ for $2 \leq i \leq 10$
- $f_i = (f_{i-1} \times hist) + (1 - hist) \times \left(f_{max} - (f_{max} - 0.4) \times \frac{1}{1 + \frac{mc}{2 \times goal}} \right)$ for $10 < i$

The default values are $hist = 0.7$ and $f_{max} = 0.9$

Figure 1.1: Over-relaxation figure adjustment algorithm

2 Simple_OR and Dynamic_OR

As SIMION lacks the ability to simulate the collective behaviour of plasma, the first stage of this project involved creating a tool to load information from SIMION potential arrays (PA files) and to calculate the value of the field at points on the grid using the methods outlined in Appendix F of [4]. The results calculated using this tool can then be compared to the results obtained from SIMION as detailed below. Over the course of this part of the project, this tool became 3 seperate programs - Simple_OR , Dynamic_OR and PALoader, with a fourth tool named FLTE started to try and calculate the distribution of plasma within the electric field calculated using SIMION or Dynamic_OR .

Simple_OR and Dynamic_OR iterate over an array of potentials, adjusting the value at each point using an over-relaxation method. In Simple_OR , the over-relaxation factor is a constant value that is supplied when the program is executed. Initially these programs operated on a 2D 5×5 grid with electrode points in each corner, and were then extended to iterate over an $N \times N$ grid. The potential was fixed at $+5V$ in three corners and $-5V$ in the fourth. This ensured that there was a visible gradient in the resulting profile, which was useful during inital testing of the code. In both cases, the electrodes were fixed in the corners and the size of the grid was fixed at compile time. The results obtained by running Simple_OR using a 20×20 grid and varying over-relaxation factors are shown in 2.1.

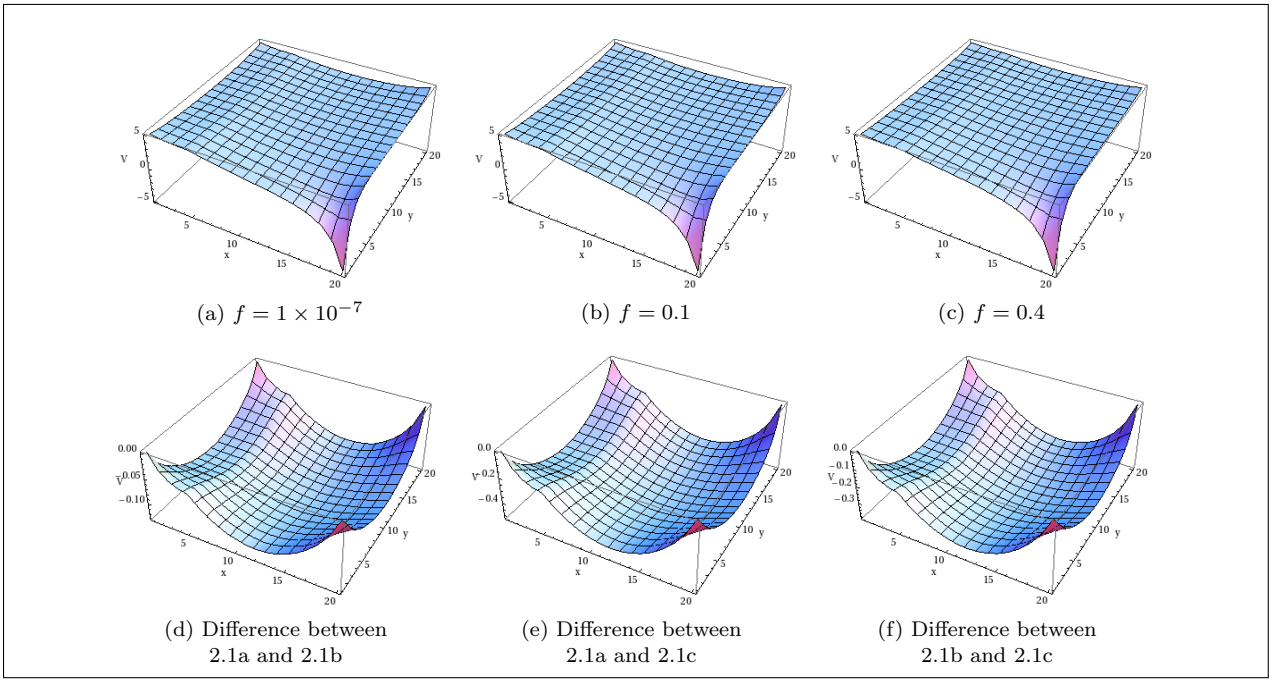
Although the results for the three different over-relaxation factors are visually similar at the scale shown, the differences between them can be more clearly seen in the figures 2.1d, 2.1e and 2.1f, with the largest variations occuring towards the center of the grid in the comparison between $f = 1 \times 10^{-7}$ and $f = 0.4$ (figure 2.1e).

As shown in table 2.1, the choice of over-relaxation factor (f), has a bearing on the number of iterations required. The large values of f result in a smaller number of iterations and a faster execution time, but potentially arriving at a less accurate result. The approach taken in SIMION and Dynamic_OR adjusts this over-relaxation factor during the programs execution in order to reach a result quickly, while maintaining an accurate result. This is done by using a larger over-relaxation factor when the solution is far from convergence, and reducing the value as the maximum difference between successive iterations over the grid decreases. The algorithm used is described earlier in this document, in figure 1.1.

ORF	Iterations	Time (s)
1×10^{-7}	225	0.012
0.1	195	0.007
0.4	124	0.004

Table 2.1: Time and iterations required to reach a stable solution for varying values of ORF

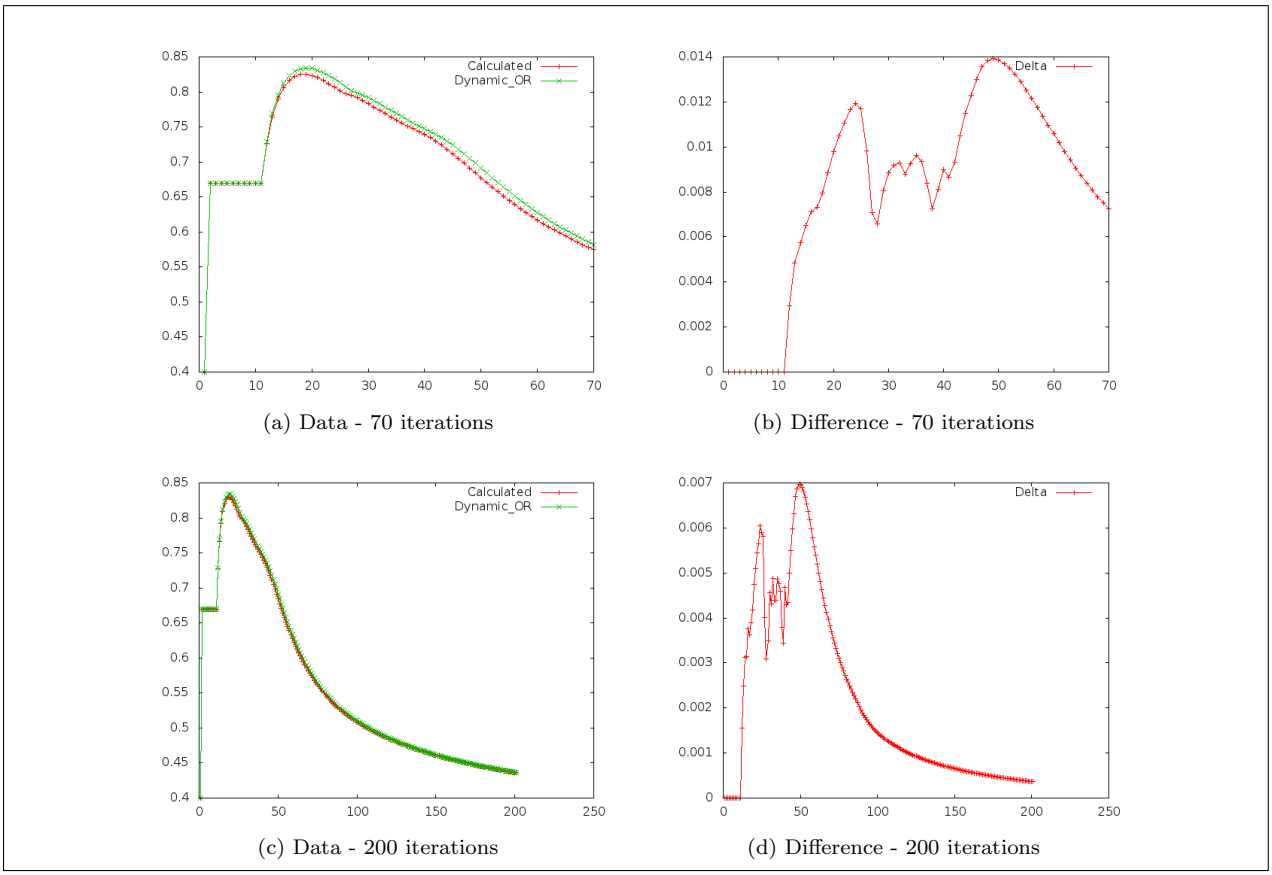
Dynamic_OR uses the dynamic over-relaxation algorithm to calculate the electric field. This should result in a faster calculation than in the Simple_OR case, while still giving accurate results. As the algorithm is the one published in [4], Dynamic_OR will be compared to SIMION and developed further. Figure 2.3 shows a plot of the results obtained from Dynamic_OR and SIMION, as well as a plot of the differences between them. In this example, the largest discrepancy between the SIMION and Dynamic_OR results is 0.284V, which occurs at the central 4 points of the plot - $(9, 9)$, $(9, 10)$, $(10, 9)$ and $(10, 10)$.



(a) $f = 1 \times 10^{-7}$ (b) $f = 0.1$ (c) $f = 0.4$

(d) Difference between 2.1a and 2.1b (e) Difference between 2.1a and 2.1c (f) Difference between 2.1b and 2.1c

Figure 2.1: Simple_OR results for a 20×20 grid.
The $-5V$ electrode is at the bottom right corner of the plots as shown with the remaining three at $+5V$

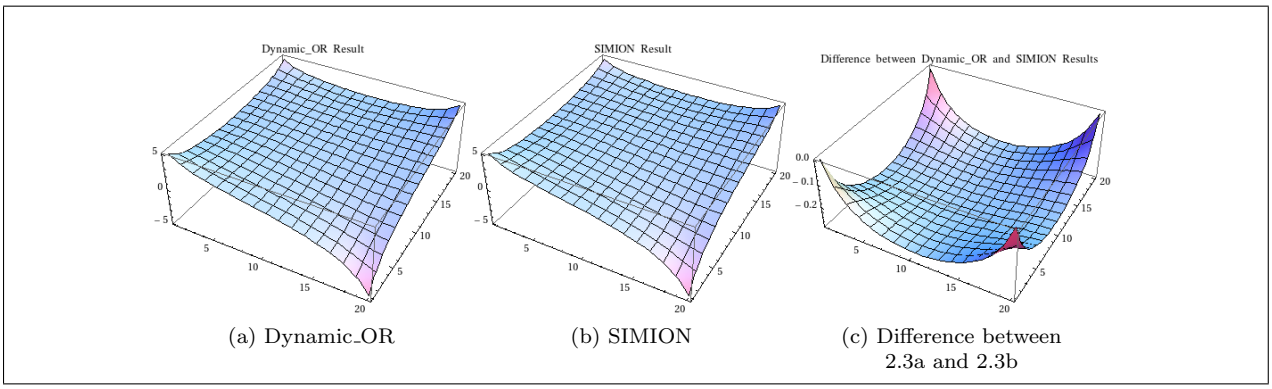


(a) Data - 70 iterations (b) Difference - 70 iterations

(c) Data - 200 iterations (d) Difference - 200 iterations

Figure 2.2: *ORF* values generated by Dynamic_OR and calculated independently for 70 and 200 iterations

In order to determine why this discrepancy exists, the generated over-relaxation factors were calculated using the algorithm shown in figure 1.1 and compared to the values used by Dynamic_OR . This showed that the over-relaxation factors used for the first 40 dynamic iterations (after iteration 11) were deviating from the expected value, with the values after that point slowly converging towards the values calculated. This convergence was checked by forcing a 200 iteration run, rather than allowing the program to terminate when the electric field solution converged around 70 iterations. The results of these runs can be seen in figure 2.2, with a comparison between Dynamic_OR and SIMION in figure 2.3.



(a) Dynamic_OR (b) SIMION (c) Difference between 2.3a and 2.3b

Figure 2.3: Simple example: Point electrodes in each corner, +5V and −5V (bottom right) with $N = 20$

After careful examination of the source code, no error could be located in the algorithm. As the largest difference between expected and generated over-relaxation factors was 0.006975 - approximately 1% - no further time was spent isolating this error. As can be seen later, the number of iterations required to reach a stable result is normally sufficient to ensure that the over-relaxation has moved beyond the peak in figures 2.2d and 2.2b.

2.1 PALoader

In order to make further comparisons between Dynamic_OR and SIMION, it becomes necessary to create a more usable way of defining a set of electrodes and potentials, as well as a more convenient method of extracting the data from SIMION's output. Both of these points can be handled by developing a library capable of reading files in the potential array format generated by SIMION (*PA Files*). Using the information in Appendix F of [4], code was written to read data from a specified PA file and load it into memory in a manner accessible to the rest of the code in Dynamic_OR . This code is contained in two files. `SimionPA.h` provides definitions for the file header and functions used to access those files, while `SimionPA.c` contains the code for those functions. The functions used in this project are `load_pa_file`, which loads the PA file into memory, and `free_points`, which frees the memory used by the data after the data is finished with.

These functions allow Dynamic_OR to load PA files and iterate over the grid defined in that file. It also allows us to read a PA file that has been refined in SIMION in order to easily compare the results to those obtained from Dynamic_OR . For this purpose, PALoader was written to read a PA file and output the data in a format compatible with Mathematica. This allowed both the unrefined and SIMION refined potential arrays to be plotted and compared to the Dynamic_OR results. Example output from PALoader is shown in figure 2.4

2.2 Programming Languages

This project has made use of C and Python for the various tools and scripts. Mathematica has also been used extensively to plot and analyse results. Simple_OR , Dynamic_OR , PALoader and FLTE are written in C, which was primarily done to take advantage of the speed of C. The speed of C based programs is offset by the limitations of the language, one of which relates to floating point mathematics. Floating point values are normally stored and manipulated as binary data, which can cause a loss of precision for some values - either those with a large number of significant figures or those which cannot be represented as non-recurring numbers in base 2. While libraries exist that permit the use and manipulation of arbitrarily precise numbers, they carry a penalty in terms of speed and complexity. In C a double precision floating point value corresponds to the IEEE definition 'binary64', which can safely store 15 significant figures and values in the range $2^{\pm 1023}$[6], which should be more than sufficient for the purposes of this project.

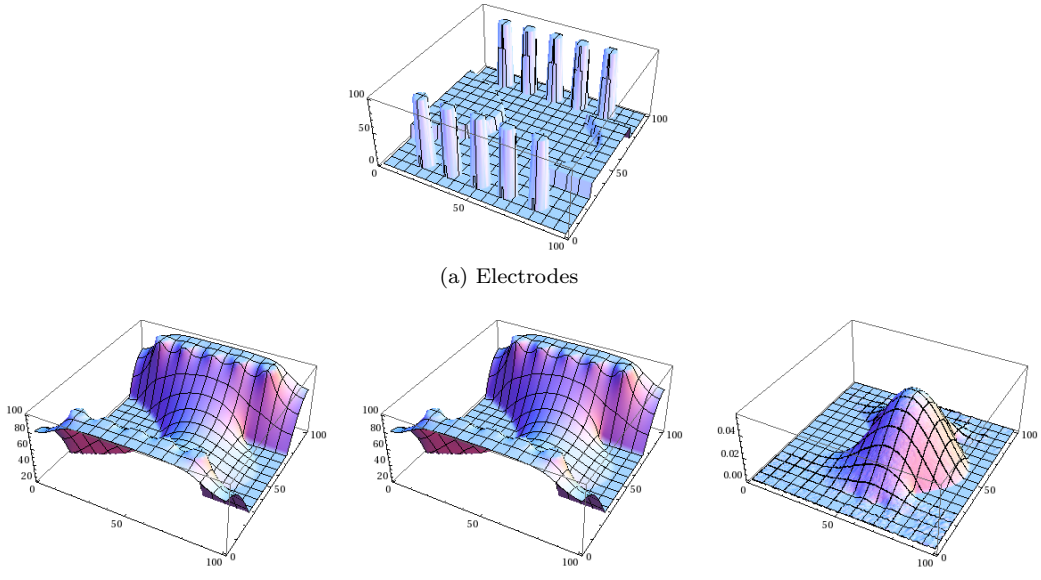
In addition to the main programs written in C, there are a number of python scripts associated with this project which generate the more complicated PA files used. Python was chosen for these scripts due to the available support for writing SIMION PA files using the SL libraries. A small selection of these scripts are included in appendix A.

```
Mode:          -1
Symmetry:       1
Max Voltage:    100000.000000
Nx:             5
Ny:             5
Nz:             1
Ng:             100
Mirror:         1600
Array has planar symmetry
----------------------------------
{
    {+5.000,+4.155,+3.771,+3.922,+5.000},
    {+4.155,+3.699,+3.240,+2.996,+3.071},
    {+3.771,+3.240,+2.493,+1.750,+1.221},
    {+3.922,+2.996,+1.750,+0.297,-1.161},
    {+5.000,+3.071,+1.221,-1.161,-5.000}
}
----------------------------------
```

Figure 2.4: Example output from PALoader

3 Comparing Dynamic_OR and SIMION

After writing the code that allows easy comparison between SIMION and Dynamic_OR, a number of test cases were constructed. The initial example in 2.3 showed that the difference between the two results was greatest at points furthest from the electrodes. It was uncertain whether this was a general property of Dynamic_OR, or an artifact of the relatively small grid size chosen for the demonstration. A more complicated arrangement of electrodes set in a larger grid was created as a PA file using the tools in SIMION. This file was then refined independently in SIMION and Dynamic_OR, and the results from each plotted as shown in figure 3.1



(a) Electrodes

(b) Dynamic_OR

(c) SIMION

(d) Difference between 3.1b and3.1c

Figure 3.1: SIMION and Dynamic_OR Comparison 1: All vertical axes in V

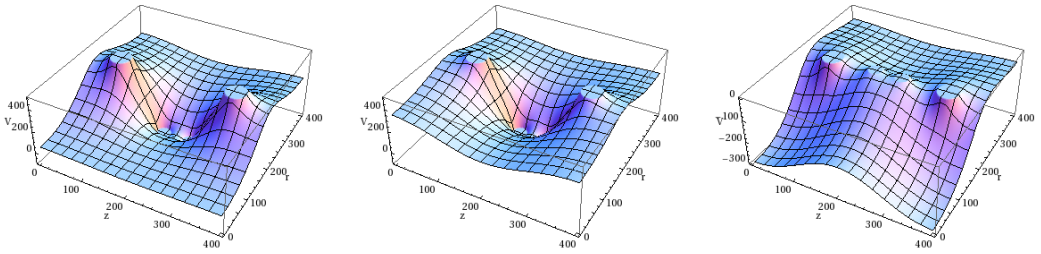
There are 4 sets of electrodes in this test pattern - 2 sets of 5 cylinders at a potential of 100V and two shaped plates at a potential of 20V. These potentials are shown in figure 3.1a. The difference between the SIMION and Dynamic_OR results is shown in figure 3.1d. As with the sample case shown in figure 2.3a, the difference between the two results is most pronounced at the furthest point from the electrodes. The difference at this point is 0.057V, $\sim 0.14\%$ of the value of the potential at that point.

3.1 Cylindrical Symmetry in Dynamic_OR

In addition to the planar arrays examined so far, SIMION is capable of calculating the electric field for cylindrically symmetric arrangements of electrodes. This is done by defining the potential array as a 2D plane in (r, z), with $r = 0$ corresponding to the axis of symmetry. The equations used to iterate over the field are modified slightly to allow for the relative distances between points once the array is revolved around the

$r = 0$ axis. By modifying Dynamic_OR to include these modified equations, cylindrically symmetric arrays can then be calculated using Dynamic_OR . Given that many experimental set ups involve ring electrodes around a beam line, this is a very useful scenario to be able to consider.

As a test, a potential array was set up in SIMION composed of two ring electrodes at a potential of 400V and refined seperately in Dynamic_OR and SIMION. This is shown in figure 3.2. As can be seen from the scale of figure 3.2c, there are significant problems with using Dynamic_OR for cylindrical arrays. The Dynamic_OR results shows the potential tending towards zero at the $r = 0$ axis, which is significantly different to the the SIMION result, which shows a much smaller drop in the potential towards $r = 0$. A peak difference of ~ 300V means that this result could not even be considered a reasonable approximation.



(a) Dynamic_OR (b) SIMION (c) Difference between 3.2a and 3.2b

Figure 3.2: Cylindrical symmetry test of Dynamic_OR using two ring electrodes at 400V

In order to eliminate programming error as the cause of the difference between these results, a number of changes were made to the code in Dynamic_OR used to generate cylindrical results, and these results compared against the original Dynamic_OR result and the SIMION result. A summary of these tests is noted in table 3.1. The test pattern used was a cross-section of a central ring electrode at 0V and two cylindrically symmetric end caps at 400V, as shown in figure 3.3. The results of these checks showed negligible changes in the results, which seems to indicate that the cause of the error does not lie in the implementation of the algorithm but in the algorithm or the way information is being passed to it.

Test	Comments
Reversed r iteration Reversed z iteration Iterated over X, Y rather than Y, X	Test if error is an artefact of iteration direction
Mirror data in $r = 0$	Although the algorithm given in [4] accounts for mirroring about $r = 0$, this test treated the array as being a cross-section from $-R$ to $+R$ rather than 0 to $+R$, effectively doubling the number of passes in the z direction
Forced long interation	Code altered to enforce a minimum number of iterations (150)

Table 3.1: Summary of tests on `WDCylinder.pa` test file

3D Cylindrical Electrodes in a Planar Array

As the tests in the preceding section showed no obvious cause for the difference between SIMION and Dynamic_OR, an alternative approach was taken. A three-dimensional cylinder was generated using a Python script and the `SIMION.PA` python module, with a central electrode and two grounded endcaps, in a similar manner to the electrodes described in section 4.

Refining the data seperately in Dynamic_OR and SIMION produced promisingly similar results. The maximum difference between Dynamic_OR and SIMION was 0.084V, which represents a 0.2% difference in terms of the potential at those points. Dynamic_OR and SIMION both dealt with the sample in similar lengths of time - Dynamic_OR taking 81 iterations and 0.451s and SIMION taking 90 iterations and 0.27s.

Repeating these tests with a larger array yielded similar results. In the second case, the maximum difference between the two results was 0.056V, which corresponds to 0.08% of the potential at that point. It took longer to generate the results for this larger array - 258 iterations in 14.044 seconds for Dynamic_OR and 250 iterations in 7.88 seconds for SIMION. The dimensions of the two cylinders are shown in table 3.2. Note that the Radius R is defined from the outer wall of the cylinder and that the electrodes are centered within the array, the array dimensions being large enough to contain the electrodes and a small number of grid points beyond the cylinder wall in all planes

As Dynamic_OR and SIMION appear to generate the same result for the cylinders in 3D, this method should be used to handle cylindrically symmetric arrangements of electrodes in Dynamic_OR rather than the cross-section method.

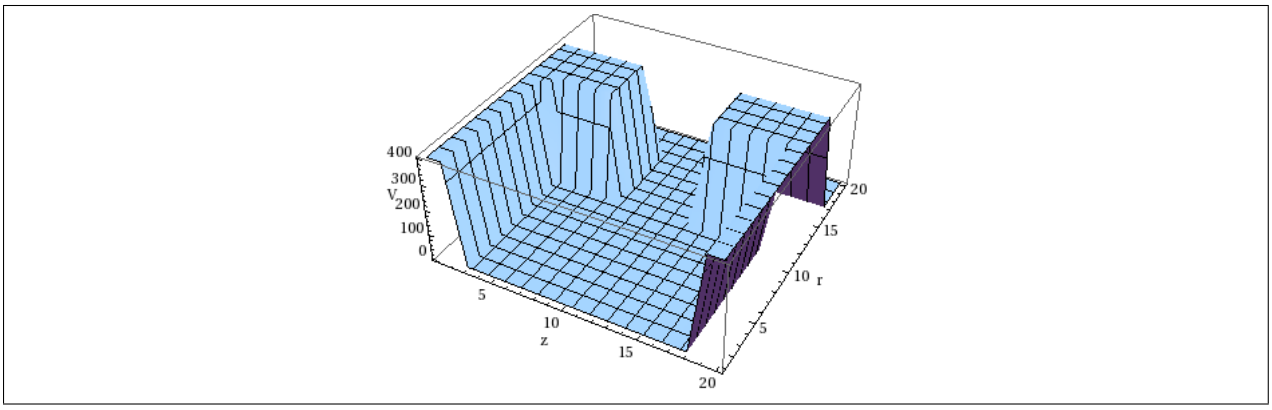
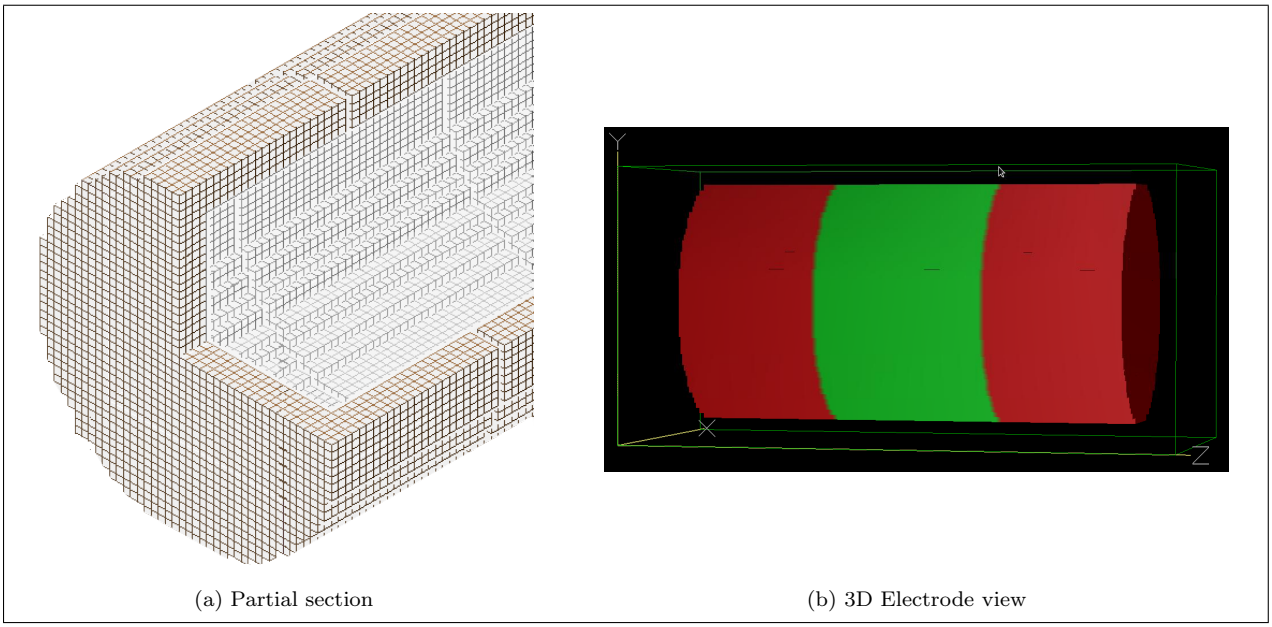


Figure 3.3: Cross-section of cylindrical electrodes in array `WDCylinder.pa`



(a) Partial section

(b) 3D Electrode view

Figure 3.4: Cross section and 3D views of the generated cylinder in SIMION

	Small	Large
Radius, R	22	40
Wall thickness, δ	5	10
Overall Length, $L1$	80	280
Central electrode length, $L2$	30	50
Array dimensions (X, Y, Z)	$50, 50, 100$	$100, 100, 300$

Table 3.2: Parameters for cylindrical test arrays

4 Harmonic Potentials

In this section results from SIMION and Dynamic_OR will be compared to analytical solutions for a harmonic potential. This class of electric fields is important, as they represent the fields for Penning type traps, which are widely used to trap charged particles.

A simple Penning trap can be constructed from 3 electrodes - a single cylindrical electrode and two endcaps. An externally applied homogeneous magnetic field is applied to confine particles radially, with the electric field forming a quadropole that confines the particles axially. This results in fast circular motion at the cyclotron frequency with a slower circular motion due at the magnetron.[7, 8]

An ideal Penning trap can be described as having the following arrangement of electric ($\vec{E}$) and magnetic ($\vec{B}$) fields[7]:

$$\vec{B} = B\hat{e}_z \tag{4.1}$$

$$\vec{E} = -\nabla\Phi \tag{4.2}$$

$$\Phi(x, y, z) = \frac{U}{2 \times z_0^2 + r_0^2}\left(2 \times z^2 - x^2 - y^2\right) \tag{4.3}$$

Using the relationship $\vec{F} = q\vec{E}$ and the principal of superposition the force on a particle can be written as the sum of two terms, where the first term is the force due to the electric potential and the second term is the force due to the magnetic field. This approach allows three equations of motion to be derived that accurately describe the motion of the trapped particle.

This analytical approach is fine for idealised, closed Penning traps, but would not necessarily be feasible for more complicated trap arrangements. A slightly more complex example is given in [9], which consists of a closed, hollow cylinder divided into three or five segments. The outer segments form the end caps, with the central electrode or electrodes being held at a given potential. It should be noted that the names of variables in this section are generally the same as those used in [9], with the cylinder described by radial coordinate ρ and length coordinate z as shown in the cross-section view in figure 4.1.

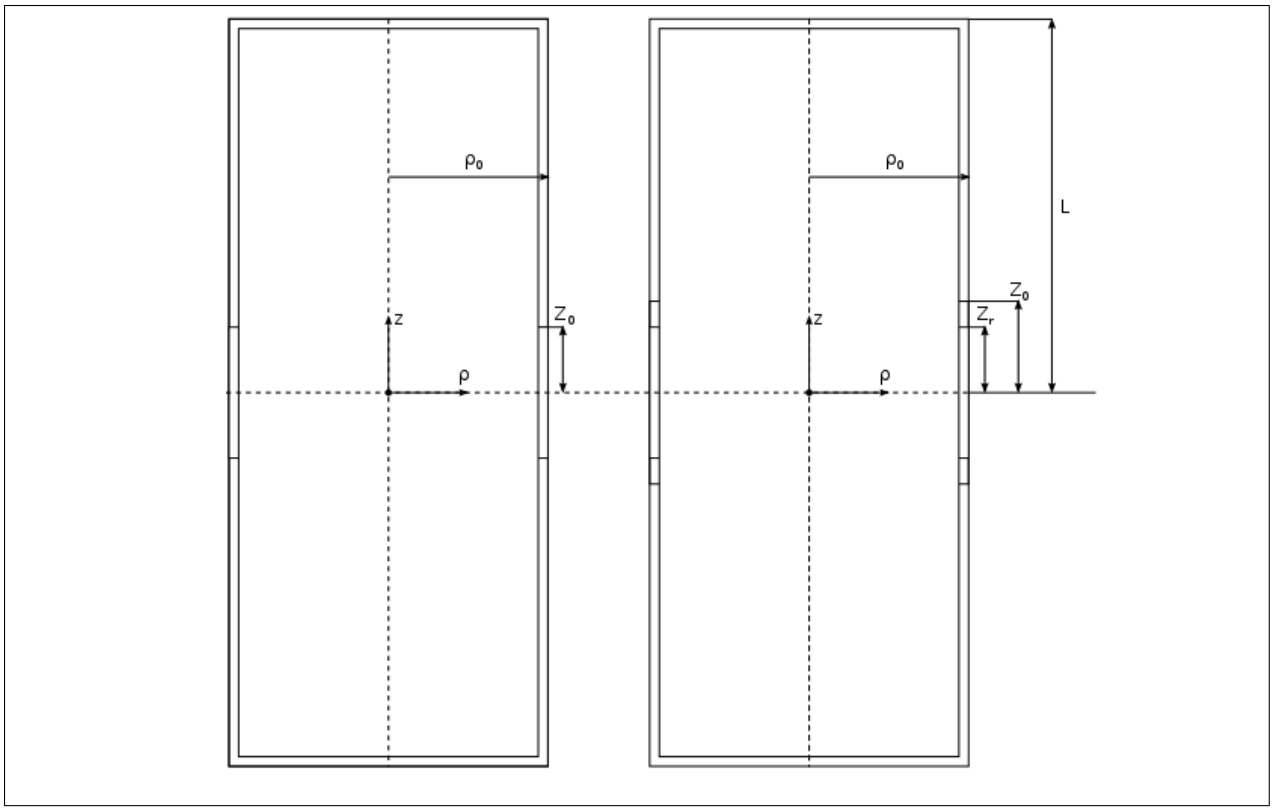


Figure 4.1: Cross section of cylindrical electrodes for a harmonic potential - 3 (L) and 5 (R) electrode cases

4.1 Three electrodes

Taking the electrodes as shown in figure 4.1, the central electrode will be assigned a potential V_0 and the two end caps grounded ($V = 0$). These conditions can be expressed in cylindrical coordinates as shown below, where ρ is the radial coordinate, z is the axial coordinate and ρ_0, z_0, Z_r and L are defined as shown in figure 4.1.

$$V(\rho = \rho_0) = \begin{cases} -V_0 & |z| < z_0 \\ 0 & |z| > z_0 \end{cases} \tag{4.4}$$

$$V(z = \pm L) = 0 \tag{4.5}$$

This yields a potential that can be written as an infinite sum of modified Bessel $I_n()$ functions of order n, in the following form[9]:

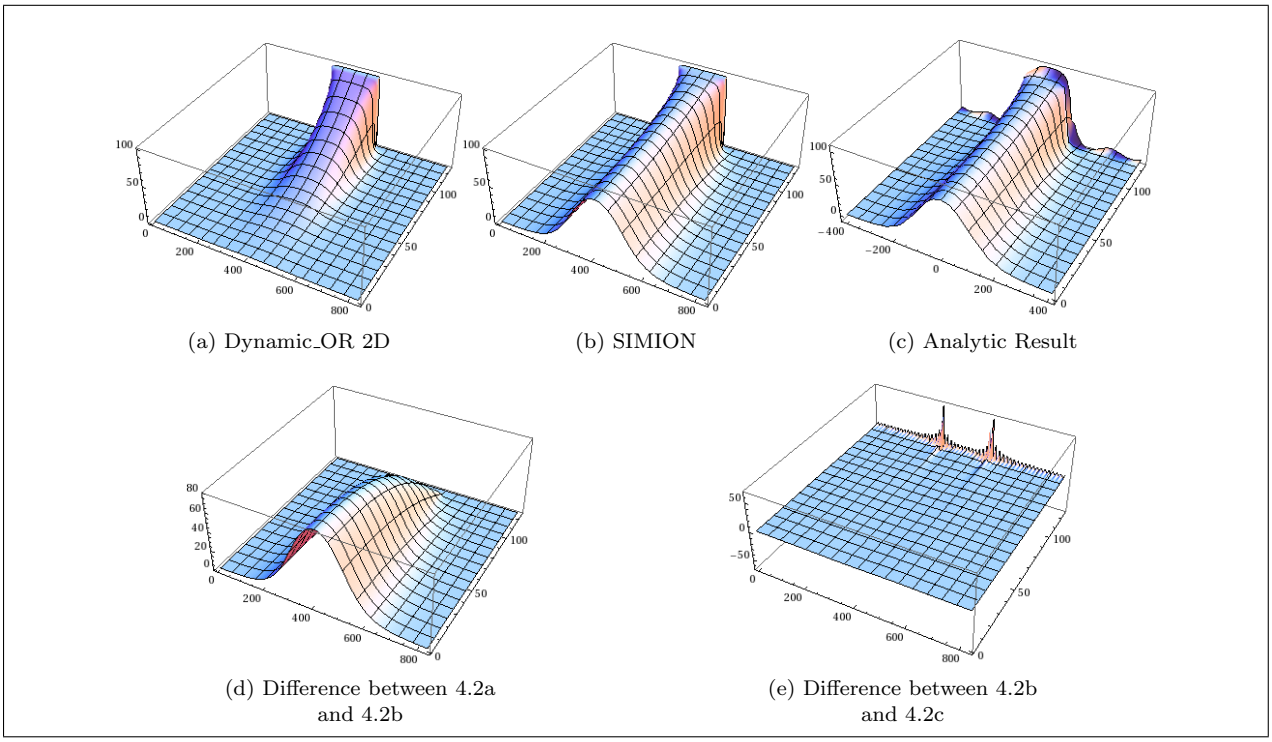
$$\Phi = \Phi_0 \sum_{n=0}^{\infty} A_n I_0(k_n \rho) \cos(k_n z) \tag{4.6}$$

$$\text{Where } A_n = \frac{2}{k_n \times L} \frac{\sin(k_n z_0)}{I_0(k_n \rho_0)} \tag{4.7}$$

$$\text{and } k_n = \frac{\left(n + \frac{1}{2}\right)\pi}{L} \tag{4.8}$$

Using the values given in [9], a two dimensional potential array was constructed using a python script, using 200 grid points per centimetre to convert physical dimensions into an array size. This resulted in a large potential array, 830×125 points, which was then refined independently in SIMION and Dynamic_OR and a corresponding solution generated in Mathematica using the the analytical solution given in equation 4.6. Based on table 2.1 of [9], the value of L was set to $4Z_0$ as there was little difference between the results of the $L = 4Z_0$ and $L = 20Z_0$ cases discussed there, and the smaller L value allowed for faster calculations.

The results shown in figure 4.2 show that, as expected, Dynamic_OR does not yield a realistic result when iterating over a cross-section of a cylinder in this manner. The results from SIMION however compare very well to the analytical results obtained from Mathematica. Note that while the horizontal scale in figure 4.2c runs from -400 to 400, the calculations correspond to the 0 to 800 range specified for the other plots. This is a side effect of the different locations for the origin in SIMION and Dynamic_OR compared to the analytical approach. This does not change any of the results shown. The largest difference between the SIMION and analytical results is 66V seen as the large peaks in 4.2e - the early termination of the summations gives rise to the 'ripples' observed in figure 4.2c, which in turn gives the large differences where the sharper edges of the SIMION solution coincide with the smoother profile of the analytical result. The mean difference between the two sets of results is much lower - 0.03V.



(a) Dynamic_OR 2D

(b) SIMION

(c) Analytic Result

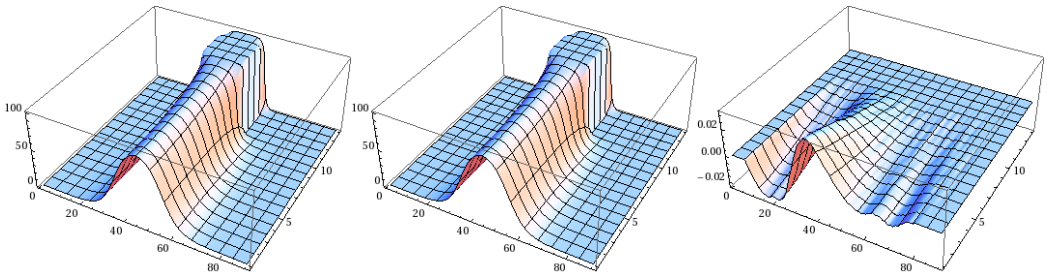
(d) Difference between 4.2a and 4.2b

(e) Difference between 4.2b and 4.2c

Figure 4.2: Results for the 3 Electrode harmonic potential and comparison between them

3D comparison

As discussed above, Dynamic_OR does not deal with cylindrically symmetric potential arrays correctly. This can be worked around by generating a PA containing the full cylinder rather than a 2D cross-section. Attempting to create a 3D PA using the same scale as the 2D cross-section fails, as the number of points is too large for SIMION to deal with. For the purposes of this test, a revised scale of 20 points/cm was used, giving a $90 \times 25 \times 25$ PA with a 12 point radius cylinder, 3 point thick walls and 85 points long. Visualising the electric field inside the entire volume is difficult, so the potential on a 2D radial slice is shown in figure 4.3 for both Dynamic_OR and SIMION. Comparing SIMION and Dynamic_OR over the entire PA shows that the maximum difference between the two sets of results was 0.036V



(a) Dynamic_OR (b) SIMION (c) Difference between 4.3a and 4.3b

Figure 4.3: Radial slice through cylindrically symmetric arrangement of 3 electrodes

4.2 Five electrodes

In the five electrode arrangement an additional ring electrode is located to either side of the central electrode, with these three electrodes being surrounded by a pair of end caps as before. The additional pair of electrodes are held at a fraction of the potential of the central electrode, yielding the following modified boundary conditions [9]:

$$V_1\left(\rho = \rho_0\right) = \begin{cases} V_0 - V_c & |z| < z_0 \\ 0 & |z| > z_0 \end{cases} \tag{4.9}$$

$$V_2\left(\rho = \rho_0\right) = \begin{cases} V_c & |z| < z_0 \\ 0 & |z| > z_0 \end{cases} \tag{4.10}$$

$$V\left(z = \pm L\right) = 0 \tag{4.11}$$

$$V = V_1 + V_2 \tag{4.12}$$

$$\Phi = \Phi_1 + \Phi_2 \tag{4.13}$$

The new scalar potential Φ is the sum of the 3 electrode potential Φ, now referred to as Φ_1, and an additional potential Φ_2. The two potentials are very similar mathematically - the difference between the two lies in the A_n values, which are related to the size of the electrodes.

$$\Phi_1 = \Phi_0^{(1)} \sum_{n=0}^{\infty} A_n^{(1)} I_0\left(k_n \rho\right) \cos\left(k_n z\right) \tag{4.14}$$

$$\Phi_2 = \Phi_0^{(2)} \sum_{n=0}^{\infty} A_n^{(2)} I_0\left(k_n \rho\right) \cos\left(k_n z\right) \tag{4.15}$$

$$A_n^{(1)} = \frac{2}{k_n \times L} \frac{\sin\left(k_n z_r\right)}{I_0\left(k_n \rho_0\right)} \tag{4.16}$$

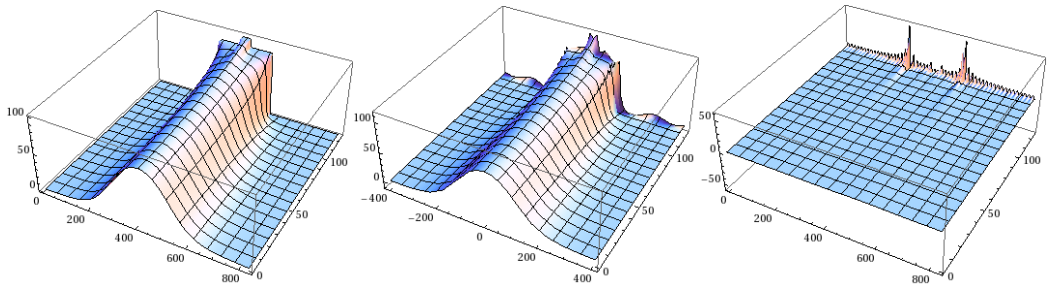
$$A_n^{(2)} = \frac{2}{k_n \times L} \frac{\sin\left(k_n z_0\right)}{I_0\left(k_n \rho_0\right)} \tag{4.17}$$

$$k_n = \frac{\left(n + \frac{1}{2}\right)\pi}{L} \tag{4.18}$$

This new potential can then be plotted and compared to the results from SIMION, as shown in figure 4.4. The largest difference between the SIMION and analytical result is 60V, which can be seen as the large peaks in 4.4c.

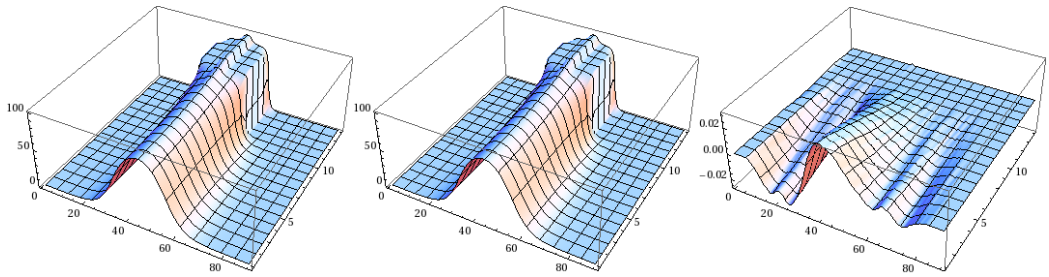
The advantage of the numerical approach used in SIMION and Dynamic_OR in general become apparent when the time required to run the programs is compared. Using figures from the two dimensional three electrode case, refining the potential array in SIMION required 1520 iterations in 3.08 seconds, or 270 iterations in 0.12 seconds using a technique called skipped-point refining, while Mathematica needed 718

seconds to calculate the value at each point on the grid for comparison. For large three dimensional arrays, the time requirements of Mathematica would be even higher. It should be pointed out that Mathematica is capable of plotting the potential in less time than it requires to evaluate it at all points on the grid however, using its own algorithms to analyse the functions and calculate sufficient values to create the plot.



(a) SIMION (b) Analytic Result (c) Difference between 4.4a and 4.4b

Figure 4.4: Results for the 5 Electrode harmonic potential and comparison between them



(a) Dynamic_OR (b) SIMION (c) Difference between 4.5a and 4.5b

Figure 4.5: Radial slice through cylindrically symmetric arrangement of 5 electrodes

5 Plasma density profile

A plasma is a state of matter composed of free, charged particles. Plasmas can be classified as neutral or non-neutral depending on their overall charge, although in both cases charged particles are able to move independently. This differs from gases and other states of matter where, for example, electrons are bound to atomic nuclei and are not as free to move. There are a number of other ways to group and classify plasmas, for example thermal - all constituents in thermal equilibrium - or non-thermal, single- or multi-species, fully or partially ionised. In all of these cases, the behaviour and distribution of the plasma is affected by any externally applied electric or magnetic field as well as the fields generated by the distribution and motion of the particles that make up the plasma.

Each particle in the plasma can be described by its mass (m), charge (q), position ($\vec{x}$), and velocity ($\vec{v}$). For N particles, the state of the system can be represented as a point in a $6N$-dimensional space. This can then be simplified to a six dimensional system using a distribution function $f(\vec{x}, \vec{v}, t)$[10]. This function represents the average number of particles at point $\vec{x}$ with velocity $\vec{v}$ at time t. Integrating the distribution over all velocities yields the number or spatial density distribution:

$$n(\vec{x}, t) = \int_{-\infty}^{\infty} f(\vec{x}, \vec{v}, t)\ \mathrm{d}^3\vec{v} \tag{5.1}$$

We can then define a velocity distribution as the quotient of f and n:

$$f'(\vec{x}, \vec{v}, t) = \frac{f(\vec{x}, \vec{v}, t)}{n(\vec{x}, t)} \tag{5.2}$$

If the plasma thermalizes due to collisions within the plasma then the velocity distribution will tend towards a Maxwellian distribution[10]:

$$f'_M = \left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} \times \exp\left(-\frac{v^2}{v_T^2}\right) \tag{5.3}$$

$$\text{With } v_T = \left(\frac{2k_B T}{m}\right)^{\frac{1}{2}} \tag{5.4}$$

5.1 Thermal equilibria for a pure electron plasma

A pure electron plasma is able to come to thermal equilibrium while confined by electric and magnetic fields[11], using a similar arrangement of electrodes to that described above in Section 4 (Harmonic Potentials). If the plasma is confined near the centre of the electrodes then it can be described using the following distribution:

$$f = n_0 \left(\frac{m}{2\pi T}\right)^{\frac{3}{2}} \times \exp\left(\frac{-\left(H - \omega p_\theta\right)}{T}\right) \tag{5.5}$$

$$\text{With } H = \frac{m\left|\vec{v}\right|^2}{2} - q\varphi\left(r, z\right) \tag{5.6}$$

$$p_\theta = mv_\theta r - \frac{q}{c} A_\theta\left(r\right) r \tag{5.7}$$

$$A_\theta\left(r\right) = \frac{Br}{2} \tag{5.8}$$

By taking Poisson's Equation (Eq. 1.12) and scaling it, the problem can be reduced to four parameters, γ, ρ_c, ζ_c and $\frac{e\Delta V}{T}$. The variable substitutions shown below allow Poisson's Equation to be rewritten as shown in equation 5.15. Note that ρ in this section refers to radius, not charge density.

$$\psi = \frac{e\varphi}{T} - \frac{m\omega(\Omega - \omega)r^2}{2T} \tag{5.9}$$

$$\rho = \frac{r}{\lambda_D} \tag{5.10}$$

$$\zeta = \frac{z}{\lambda_D} \tag{5.11}$$

$$\lambda_D^2 = \frac{T}{4\pi n_0 e^2} \tag{5.12}$$

$$\gamma = \frac{2m\omega(\Omega - \omega)}{4\pi n_0 e^2} - 1 \tag{5.13}$$

$$\Omega = \frac{eB}{mc} \tag{5.14}$$

$$\frac{1}{\rho}\frac{\partial}{\partial\rho}\rho\frac{\partial\psi}{\partial\rho} + \frac{\partial^2\psi}{\partial\zeta^2} = \left(e^\psi - 1\right) - \gamma \tag{5.15}$$

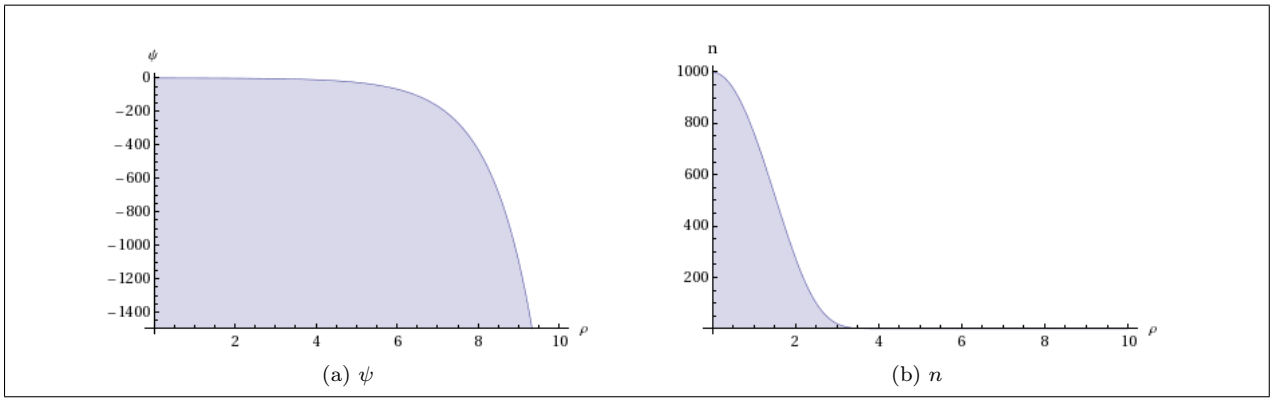
The number density is then given as $n\left(\rho, \zeta\right) = n_0 \exp\left(\psi\left(\rho, \zeta\right)\right)$.

If we model the plasma as an infinitely long column then this problem is simplified further by eliminating ζ and replacing $\exp(\psi) - 1$ with ψ in equation 5.15[11]. For sample parameters shown in table 5.1, this gives the plots shown in figure 5.1. Figure 5.1b represents the number density of the column, with the origin of the figure corresponding to the centre of the plasma column. As this calculation is for an infinitely long column, the density profile doesn't vary along the axis of the column.

This gives us a first approximation of the density in a plasma in any given electric field, φ, provided that field can be represented algebraically. We can then also use this result to compare against results for a finite column, which will be discussed further below.

m	9.11×10^{-31}
c	2.9×10^{8}
e	1.6×10^{-19}
n_0	1000
B	1
ω	1.6×10^{-25}

Table 5.1: Summary of parameters used to generate figure 5.1



(a) ψ (b) n

Figure 5.1: ψ and density against radius for an infinite column of electron plasma

For a finite plasma column this analytical approach does not hold, and a numerical iterative approach can instead be used to solve equation 5.15, which is equivalent to Poisson's Equation. The program written for this task, FLTE, uses a Jacobi iterative method as described in section 1.2. The calculation is slightly modified to solve Poisson's Equation as follows. We can start with Laplace's Equation and its corresponding iterative calculation.

$$\nabla^2 \Psi = 0$$
$$\Psi^{i+1}_{x,y} = \frac{1}{4}\left(\Psi^i_{x-1,y} + \Psi^i_{x+1,y} + \Psi^i_{x,y-1} + \Psi^i_{x,y+1}\right)$$

Poisson's Equation is slightly different. Rather than $\nabla^2\Psi$ being equal to zero, it is equal to some number. In the case of an electric field in cartesian coordinates, this is commonly written and rearranged as follows:

$$\nabla^2 \vec{E} = \frac{-\rho}{\epsilon_0}$$
$$\nabla^2 \vec{E} + \frac{\rho}{\epsilon_0} = 0$$

Ψ calculated using the iterative calculation in the Laplace case has to satisfy $\nabla^2\Psi = 0$, which allows us to write:

$$\text{If } \nabla^2\Psi - f = 0 \text{ then:} \tag{5.16}$$
$$\Psi^{i+1}_{x,y} = \frac{1}{4}\left(\Psi^i_{x-1,y} + \Psi^i_{x+1,y} + \Psi^i_{x,y-1} + \Psi^i_{x,y+1}\right) - f \tag{5.17}$$

This modified form calculation should allow for equation 5.15 to be solved, which in turn allows the number density $n(\rho, \zeta)$ to be calculated.

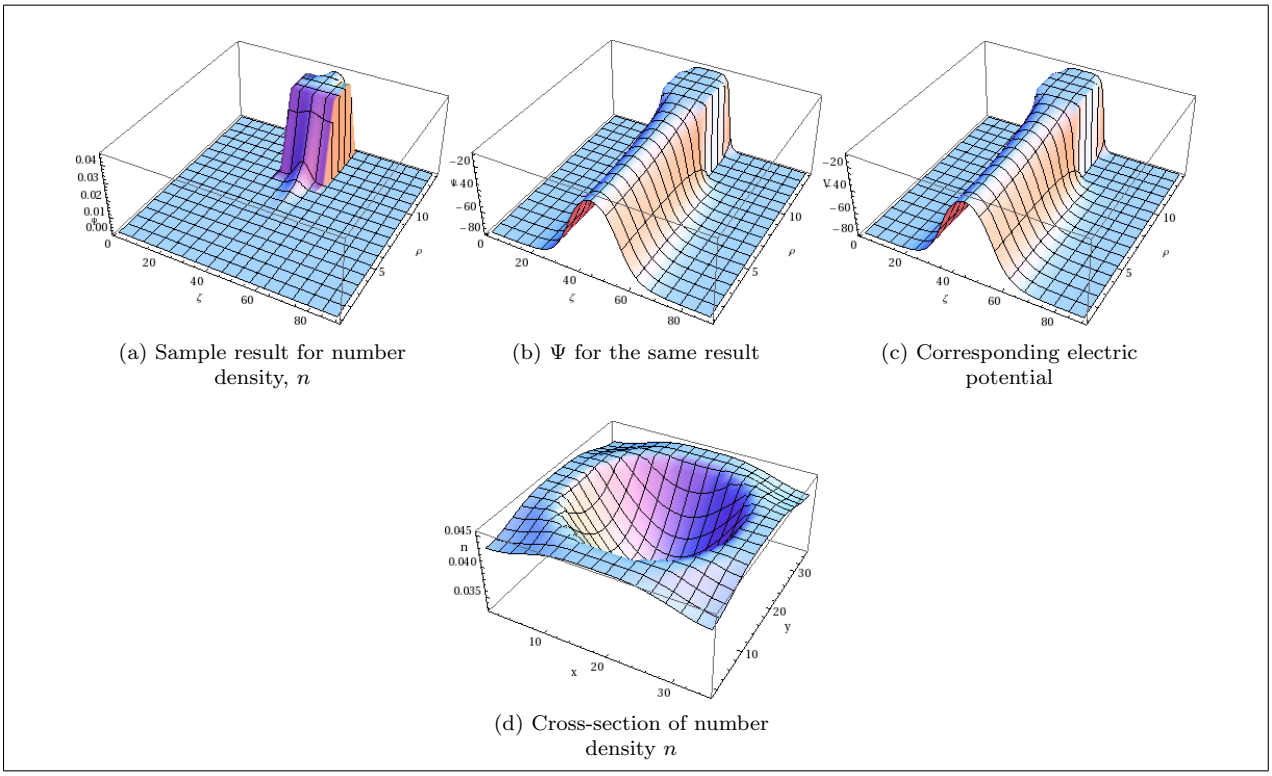
Implementations

The general procedure given in [11] starts with an estimated distribution based on equation 5.9 and requires iterating over the distribution as outlined in Section 1.2, applying the following boundary conditions:

$$\Psi = \frac{eV}{T} - (\gamma + 1)\frac{\rho_0^2}{4} \qquad \text{For points on central electrode} \tag{5.18}$$
$$\Psi = \frac{eV}{T} - \frac{\Delta V}{T} - (\gamma + 1)\frac{\rho_c^2}{4} \qquad \text{For points on endcaps} \tag{5.19}$$

Applying the algorithm above using the boundary conditions given in [11] converges on an answer for Ψ approximately equal to the potential, which is not the expected result. The expected result would show a variation in density in the radial direction similar to the number density shown in figure 5.1b, extending along the axis of the cylinder and tailing towards zero at the endcaps. The density within the plasma away from its edges should be approximately uniform. Figure 5.2 shows the results obtained for n and Ψ using the code produced for this project.

The code was run for a variety of values of the parameters, but the results varied only slightly between runs and did not display the expected results. A number of other changes were also tested in a similar manner to the tests conducted trying to narrow down the errors in Dynamic_OR for the 2D cylindrical cross-section arrays. These changes including adding an additional boundary condition ($\Psi(\rho = 0, \zeta = 0)$) and starting with a completely empty distribution rather than the estimate given in equation 5.9. The parameters were initially set using the example figures given in [11], which were $\gamma = 0.0003$ and $\frac{e\Delta V}{T} = 100$ with $\rho_c = 16$ and $\zeta_c = 64$



(a) Sample result for number density, n

(b) Ψ for the same result

(c) Corresponding electric potential

(d) Cross-section of number density n

Figure 5.2: Example of erroneous profile calculated for Plasma

6 Conclusions and recommendations

While this project may not have been able to achieve all of its original goals - namely that it is not possible to calculate the charge or number densities for plasma using the tools produced for this project - progress was made which can provide a framework for future development. Using Dynamic_OR, it is now possible to load a two- or three-dimensional array of points containing definitions of electrodes (a potential array), and calculate the value of the electric field at every point within that array. Despite the comparatively short amount of development time available, Dynamic_OR takes only a little longer to make these calculations than SIMION and produces results which differ from the SIMION results by very small amounts. These results can then be plotted and analysed in Mathematica, alongside results extracted from SIMION using another of the programs produced for this project - PALoader.

Once Dynamic_OR had been written and tested, research moved towards investigating the behaviour of plasma - principally finite columns of electron plasmas. While some results from [11] were able to be reproduced for the infinite column case, there has been little success finding the problems that prevent the system converging towards the desired results. The tests conducted have included altering the parameters to try and eliminate parameter values as the cause of the erroneous results, as well as changes to the electric field input and the associated boundary conditions. As none of these tests showed any shift towards the expected results (as approximated in 5.1b and discussed extensively in [11]), it will be necessary to further investigate the method used to solve Poisson's Equation. Other iterative methods may be more successful, such as multiple grid solvers or enhancements to the existing modified Jacobi iteration using staggered grids or non-uniform grids.

Despite its limitations, Dynamic_OR does offer some advantages over SIMION. It provides an open framework for further modification to perform similar calculations, as shown by the modifications made to create the FLTE Poisson solver. Dynamic_OR could be extended to handle different file formats, which may allow it to work more readily with other software in a lab. This would be particularly useful if Dynamic_OR were to be adapted for other physics problems that can be modelled using Laplace's Equation or Poisson's Equation. These include problems related to heat and fluid flow, gravitation or indeed any other problem which can be reduced to the investigation of a scalar potential function.

Another modification that may be worth further investigation would be to alter Dynamic_OR to use a variation on the skipped-point refining technique employed by SIMION. Iterating over a $\frac{1}{2^n}$ scale copy of the potential array, doubling it in size and interpolating for the new points before continuing as normal can reduce the time taken to calculate the field for a particular array. This can reduce the iterative finite difference method from $O\left(n^2\right)$, that is to say that the time taken to perform the calculation scales according to the square of the number of grid points, to approximately $O\left(n\right)$, or a linear increase in time required with

respect to the number of grid points (depending on the shape of the array)[4, 5]. It may also be possible to split the array into sections and iterate over them in parallel, taking advantage of the multiprocessor nature of most modern computers. One difficulty that may occur is the calculation of appropriate boundary conditions for each section, with one approach being to use the values of neighbouring points in other sections at the end of the previous iteration. An alternative approach would be to split the array at electrodes in order to provide fixed boundary conditions. This would probably be of greater benefit for large arrays, where the computational effort of splitting the array and determining the boundary conditions is more likely to be offset by the increased performance and reduced execution time of the process as a whole.

6.1 Recommendations

To summarise, the following points are recommended for further development of this project:

- Investigate iterative solutions to Poisson's equation
- Obtain and verify results for distribution of charges within a plasma
- Integrate Dynamic_OR and plasma calculations and investigate ways to improve its performance
- Investigate ways of parallelising the iteration process as described above
- Examine other input output/formats
- Look into other problems where the tools used in this project can be applied

Acknowledgments

First and foremost, thanks are due to Dr. Dirk van der Werf and Dr. Aled Isaacs for their help and assistance over the course of this project and over the rest of my degree. I hope that the results of this project are of at least some use to you with your ongoing work

I'd also like to thank Aston French for helping to proofread this document and acting as a sounding board for ideas when nothing would work.

Thanks are also due to Swansea University Computer Society for encouraging and supporting my computing interests over the past few years, which was has been of use to me during this project. SUCS will also be hosting the code from this project, which should remain available for download from http://sucs.org/~tswsl1989/Dissertation/ along with a copy of this dissertation in PDF format.

Bibliography

[1] J. D. Jackson, *Classical Electrodynamics*. Wiley, 3rd ed., 1998. ISBN:978-047-130932-1.

[2] C. A. Isaac, *Axialisation of Particles in a Penning-type Trap by the Application of a Rotating Dipole Electric Field and its Application to Positron Accumulation*. PhD thesis, Swansea University, School of Physical Sciences, 2010.

[3] A. Bonderson, T. Rylander, and P. Ingelström, *Computational Electromagnetics*. Texts in Applied Mathematics, Springer, 2005. ISBN: 0-387-26158-3.

[4] D. Manura and D. Dahl, *SIMION (R) 8.0 User Manual*. Scientific Instrument Services, Inc, Ringoes, NJ 08551, 8.0 ed., January 2008. http://simion.com/.

[5] D. A. Dahl, "SIMION for the personal computer in reflection," *International Journal of Mass Spectrometry*, vol. 200, pp. 3–25, December 2000.

[6] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1 –58, 29 2008.

[7] M. Kretzschmar, "Particle motion in a penning trap," *European Journal of Physics*, vol. 12, pp. 240–246, Sept. 1991.

[8] C. J. Foot, *Atomic Physics*. Oxford Master Series in Physics, Oxford University Press, 2005. ISBN: 019850696.

[9] J. K. Estrada, *Cold Trapped Positrons and Progress to Cold Antihydrogen*. PhD thesis, Massachusetts Institute of Technology, January 2002. http://hdl.handle.net/1721.1/16797.

[10] R. Dendy, ed., *Plasma Phyiscs: An introductory course*. Cambridge University Press, 1993. ISBN: 0581484529.

[11] S. Prasad and T. O'Neil, "Finite length thermal equilibria of a pure electron plasma column," *Physics of Fluids*, vol. 22, p. 278, 1979.

Appendix A

Code Samples

There were a number of scripts used to create and analyse the information presented in this dissertation. Many of them are very similar to one another, having been copied and adjusted to suit a particular example.

```
# generate.py
# Based on swirl.py, provided in the SIMION examples

from SIMION.PA import *
from math import *


# create new potential array
pa = PA(nx=5, ny=5, nz=1)

# iterate over all points
for x in range(0, pa.nx()):
    for y in range(0, pa.ny()):
        # compute point value
        is_electrode1 = ( (x==0 and y==0) or (x==4 and y==0) or  (x==0 and y==4))
        is_electrode2 = (x==4 and y==4)
        # set point value
        if is_electrode1: pa.point(x, y, 0, 1, 5)
        if is_electrode2: pa.point(x, y, 0, 1, -5)

# write file
pa.save("5x5_+++-5V.pa")
```

Listing A.1: Source Code: Two dimensional PA generator used for figure 2.1

```
# generate.py
# Based on swirl.py, provided in the SIMION examples

from SIMION.PA import *
from math import *


# create new potential array
pa = PA(nx=830, ny=125, nz=1)
pa.mirror('y')
pa.symmetry('cylindrical')

# iterate over all points
for x in range(0, pa.nx()):
    for y in range(0, pa.ny()):
        # compute point value
        is_electrode1 = ((((x >= 3 and x <= 5) or (x >= 817 and x <=819)) and y <= 123) or (
            (y >= 121 and y <= 123) and ((x >= 3 and x <= 294) or (x >= 528 and x <= 820)))
            )
        is_electrode2 = (((x >= 294 and x <= 392) or (x >= 430 and x <= 528)) and (y >= 121
            and y <= 123))
        is_electrode3 = ((x >= 392 and x <= 430) and (y >= 121 and y<= 123))
        # set point value
        if is_electrode1: pa.point(x, y, 0, 1, 0)
        if is_electrode2: pa.point(x, y, 0, 1, 88.1)
        if is_electrode3: pa.point(x, y, 0, 1, 100)

# write file
pa.save("5-electrode.pa")
```

Listing A.2: Source Code: Cylindrical cross-section PA generator for 5 electrodes

```
# generate.py

from SIMION.PA import *
from math import *


# create new potential array
pa = PA(nx=90, ny=25, nz=25)

delta = 3 #Wall thickness
R=12
r=R-delta
L1=82
L2=23
L3=4

offset1=(pa.nx()-L1)/2
offset2=(pa.nx()-L2)/2
offset3=(pa.nx()-L3)/2

centreZ=pa.nz()/2
centreY=pa.ny()/2

V1 = 0
V2 = 88.1
V3 = 100

# iterate over all points
for x in range(0, pa.nx()):
    for y in range(0, pa.ny()):
        for z in range(0,pa.nz()):
            # compute point value
            radius=sqrt(pow(z-centreZ,2)+pow(y-centreY,2))
            if (offset3 <= x < (offset3 + L3)) and (r <= radius < R):
                pa.point(x, y, z, 1, V3)
            elif (offset2 <= x < (offset2 + L2)) and (r <= radius < R):
                #Set Central Electrode Potential
                pa.point(x, y, z, 1, V2)
            elif ((offset1 <= x < offset2) or ((offset2 + L2) <= x < (offset1 + L1))) and (r
                <= radius < R):
                #Set Outer Electrode potential (cap wall)
                pa.point(x, y, z, 1, V1)
            elif ((offset1 <= x < (offset1 + delta)) or ((offset1 + L1 - delta) <= x < (
                offset1 + L1))) and (radius <= r):
                #Set Outer Electrode potential (cap face)
                pa.point(x, y, z, 1, V1)


# write file
pa.save("5-electrode3D.pa")
```

Listing A.3: Source Code: Three dimensional cylinder PA generator for 5 electrodes

```
/*****************************************************************************
 The definitions below are based on those printed in appendix F of the SIMION
 8.0 manual. While the definitions provided in that section  are not part of
 the SL Libraries, this code may still fall under section 1(g) of the SIMION
 License and should only be  used in conjunction with a valid license for
 SIMION 8 or a license  for the SL Libraries
*****************************************************************************/
#ifndef SIMIONPA_H
#define SIMIONPA_H

#include <stdint.h>

#define SIMION_PLANAR_SYMMETRY 1
#define SIMION_CYLINDRICAL_SYMMETRY 0

/* SIMION_MIRROR_AXIS - Mirror in AXIS */
#define SIMION_MIRROR_X 1
#define SIMION_MIRROR_Y 2
#define SIMION_MIRROR_Z 4

/**
 Magnetic potential array, rather than an
 electrosotatic array
**/
#define SIMION_MAGNETIC 8

struct simion_pa_header {
        int32_t mode ; /* = -1 */
        int32_t symmetry; /* = SIMION_CYLINDRICAL_SYMMETRY */
        double max_voltage; /* = 10000 */

        int32_t nx; /* = 50*/
        int32_t ny; /*= 50*/
        int32_t nz; /*= 50*/
/**
 SIMION works with magnetic potentials, which can be calculated from a measured
 gradient using the following defintions:

 Array Magnetic Potential = Array Gradient
 Magnetic Flux = Array MP * ng
 gauss = Array Gradient * ng

 ng is arbitrary and can be adjusted as required for an individual array
**/
        int32_t ng; /*= 100*/
/**
 The value stored in the mirror variable is defined as:
 (SIMION_MIRROR_X | SIMION_MIRROR_Y  | SIMION_MIRROR_Z | SIMION_MAGNETIC)
  + (ng << 4)
 Where the values SIMION_MIRROR etc are either zero or the value defined above
**/
        int32_t mirror; /* = 0 */
};

typedef struct simion_pa_header PAHeader;

/**
 A PA file is simion_pa_header followed by double points[nz][ny][nx]
 Each point is the value of the potential at that point. For non-electrodes,
 the value at that point is stored as is. For electrodes, the value at that
 point is added to 2 * max_voltage
**/

double*** load_pa_file(char *c, PAHeader *head, double ***points);
int save_pa_file(char *c, PAHeader *head, double ***points);
void free_points(PAHeader head, double ***p);
#endif
```

Listing A.4: Source Code: SimionPA.h

```
1
2  (*
3  Cylinder properties
4  Cylinder assumed to be open ended, of finite length and made of 3 \
5  segments - a central piece and two ends, with the 2 ends being \
6  identical
7  *)
8  L = 10; (*Length of central segment @ potential V *)

10 rc = 5; (*Radius of central segment *)

12 V = 100; (*Potential of central segment *)

14 V2 = 90; (* Potential of end segments *)

16 deltaV = V - V2
17 Zc = L/2;

19 (* Electric field *)
20 terms = 50;
21 A = Table[
22     2/((n + 1/2)*Pi) Sin[((n + 1/2)*Pi*Zc)/L]/
23      BesselI[0, ((n + 1/2)*Pi*rc)/L], {n, 0, terms}];
24 Phi[r_, z_] :=
25   N[Sum[A[[n + 1]]*BesselI[0, ((n + 1/2)*Pi)/L*r]*
26       Cos[((n + 1/2)*Pi)/L*z], {n, 0, terms}]];

28 (* Magnetic field *)
29 B = 1;

31 (* particle properties *)
32 m = 9.11*10^-31;
33 c = 2.9*10^8;
34 e = 1.6*10^-19;
35 w = 1.6*10^-26;
36 Omega = (e*B)/(m*c);
37 lambda = Sqrt[T/(4*Pi*n0*e^2)];


40 Psi[rho, zeta] := (e*Phi[rho, zeta])/T - (m*w (Omega -w)*rho^2)/(2*T);

42 (* rho=r/lambda;
43 zeta=z/lambda;
44 Gamma=(2*m*w*(Omega-w))/(4*Pi*n0*e^2)-1; *)

46 n[rho_, zeta_] := n0*Exp[Psi[rho, zeta]]
```

Listing A.5: Mathematica code used to try and plot number density of a finite length thermal plasma

```
1  dynSlice = dor53D[[12]][[12 ;; 25]];
2  simSlice = sim53D[[12]][[12 ;; 25]];
3  sliceDiff = dynSlice - simSlice;
4  Export["5-Electrode3D_D.png",
5   ListPlot3D[dynSlice, PlotRange -> Full]]
6  Export["5-Electrode3D_S.png",
7   ListPlot3D[simSlice, PlotRange -> Full]]
8  Export["5-Electrode3D_DSdiff.png",
9   ListPlot3D[dor53D[[12]][[12 ;; 25]] - sim53D[[12]][[12 ;; 25]],
10   PlotRange -> Full]]
11 sliceMaxPos = Position[sliceDiff, Max[sliceDiff]]
12 pcDiff = (Extract[sliceDiff, sliceMaxPos]/
13     Extract[dynSlice, sliceMaxPos])*100
```

Listing A.6: Mathematica code snippet used to plot a slice of a 3D array

Appendix B

Progress Report 1

Simulation of Non-Neutral Plasmas - Progress Report 1

B.1 Introduction

The goal of this project is to create and use various methods to calculate the electric field for a given arrangement of charges or electrodes and then simulate the behaviour of plasma in that electric field. The level of detail involved in these simulations will depend on the time available and difficulties encountered during the project.

The initial phase of the project involved some background research into algebraic solutions to penning-trap type magnetic fields and the equations of motion of charged particles and plasma in those fields, principally from [2]. As background information for numerical methods of calculating electric fields, extensive use has been made of the SIMION manual and software[4]. The SIMION software has also been used to provide a comparison for solutions generated by the code used in this project.

B.2 Progress

B.2.1 SIMION

SIMION is a piece of software designed to simulate the trajectories of particles in an electric field. The electric field can be input to the program from an external source, or calculated based on an arrangement of electrodes. Analytically, the field can be found by solving Laplace's equation[1, 1.7]:

$$\nabla^2 V = 0 \tag{B.2.1}$$

For scalar potential V, we can also relate the potential to the field $\vec{E}$:

$$\nabla^2 V = \nabla \vec{E} \tag{B.2.2}$$

$$\nabla^2 V = \frac{\partial^2 V_x}{\partial x^2} + \frac{\partial^2 V_y}{\partial y^2} + \frac{\partial^2 V_z}{\partial z^2} \tag{B.2.3}$$

$$\nabla \vec{E} = \frac{\partial E_x}{\partial x} + \frac{\partial E_y}{\partial y} + \frac{\partial E_z}{\partial z} \tag{B.2.4}$$

Solving a field analytically is non-trivial, but it is possible to approximate the result using simpler, numberical methods. The principle method used by SIMION is an over-relaxation method.

B.2.2 Over-Relaxation Method

Rather than try and solve the Laplace equation (Equation B.2.1) in an analytical fashion, we can approximate a solution at a finite number of points. To first approximation, the value at each point is the average of it's neigbouring points. By iterating repeatedly over the points and setting each one equal to the average of it's nearest neighbours, the values converge towards the solution to the Laplace equation. This algorithm is outlined in figure B.2.1

Solving the problem by iteration can be a long process. In this instance, we can stop the iterations once the difference between two successive iterations falls below a certain threshold. We can also reduce the time taken by changing the algorithm slightly, as shown in figure B.2.2. By adjusting the over-relaxation factor f between 1 and 2 (reducing it as we near convergence), we can reduce the time taken to converge.

1. Set initial values
2. For all points:
   - If point is an electrode, move on to next point
   - Otherwise, set the value of this point to the average of the 6 nearest points
3. If any point changed by more than *Goal*, repeat
4. Otherwise, finish

Figure B.2.1: Outline iterative approach for solving Laplace's equation

1. Set initial values
2. For all points:
   - If point is an electrode, move on to next point
   - Otherwise:
     - Let X be the the average of the 6 nearest points
     - Let Y be the current value of this point
     - Let Δ be $X - Y$
     - Set the value of this point to $Y + f\Delta$
3. If any point changed by more than *Goal*, repeat
4. Otherwise, finish

Figure B.2.2: Over-Relaxation approach for solving Laplace's equation

B.3 Work to date

B.3.1 Proofs of concept

As a first step, two simple programs were written in order to test the algorithms outlined above. These programs both operated on a statically defined 2Darray with electrodes defined in each corner, with the size defined at compile time. The first program, Simple_OR , iterated over the grid with a fixed over-relaxation factor and the second, Dynamic_OR , adjusts the over-relaxation factor based on the algorithm outlined in figure B.3.1.

For over-relaxation factor f_i, where i is the current iteration:

- $f_1 = 0.4$
- $f_i = 0.67$ for $2 \leq i \leq 10$
- $f_i = (f_{i-1} \times hist) + (1 - hist) \times \left(f_{max} - (f_{max} - 0.4) \times \frac{1}{1 + \frac{mc}{2 \times goal}} \right)$ for $10 < i$

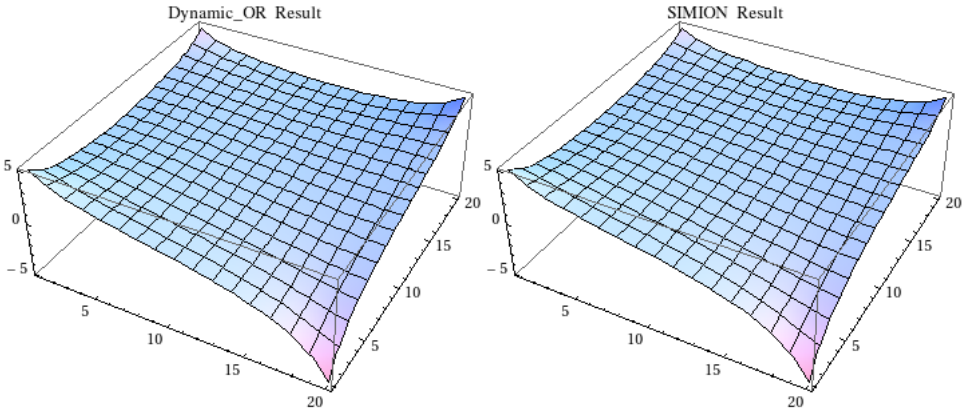
The default values are $hist = 0.7$ and $f_{max} = 0.9$

Figure B.3.1: Over-relaxation figure adjustment algorithm

The output from Dynamic_OR compared favourably to data from SIMION. In order to compare the results, the output of each was plotted in Mathematica using the `ListPlot3D` command. The results from Dynamic_OR and SIMION are shown in figure B.3.2, and are visually very similar. A plot showing the difference between the two sets of data is shown in figure B.3.2c. The difference is 0 in the corners, as these points were treated as electrodes. The largest difference between the Dynamic_OR results and those from SIMION is -0.287, which occurs at 4 points - $(9, 9)$, $(9, 10)$, $(10, 9)$ and $(10, 10)$ - the 4 central points of the grid. It can be seen in further comparisons that the discrepancy between SIMION and Dynamic_OR for planar arrays increases with distance from the nearest electrode. The reason for this is currently unknown.

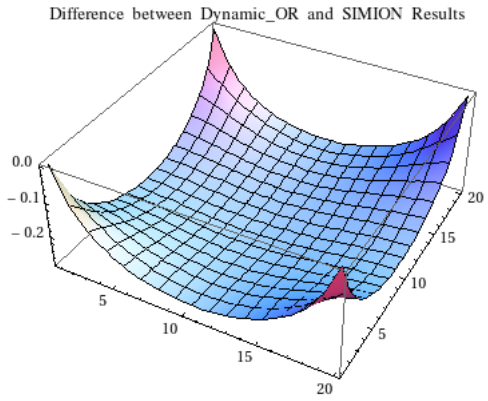
B.3.2 PALoader

In order to easily compare SIMION and Dynamic_OR , a tool was written using the information contained in [4, Appendix F] to read SIMION PA files. This allows PA files to be defined using the SIMION editor,



(a) Dynamic_OR

(b) SIMION



(c) Difference between B.3.2a and B.3.2b

Figure B.3.2: Simple example: Point electrodes in each corner, +5V and −5V (bottom right)

and then refined seperately in both SIMION and Dynamic_OR . To help develop the code used to read PA files, a tool called PALoader was written that reads PA files and outputs the value at each point in a format that can be used in Mathematica. This also makes it easier to compare the results.

B.3.3 Dynamic_OR

The earlier version of Dynamic_OR has now been rewritten to read PA files and then refine them, before printing the result in a format suitable for input into Mathematica. A number of different arrays were used to test the program, and the results generated compared to the results from SIMION. As with the simple test case illustrated in figure B.3.2, the difference between SIMION and Dynamic_OR increases away from the electrodes. A sample result from Dynamic_OR can be seen in figure B.3.3, with a maximum difference between Dynamic_OR and SIMION of 0.057V

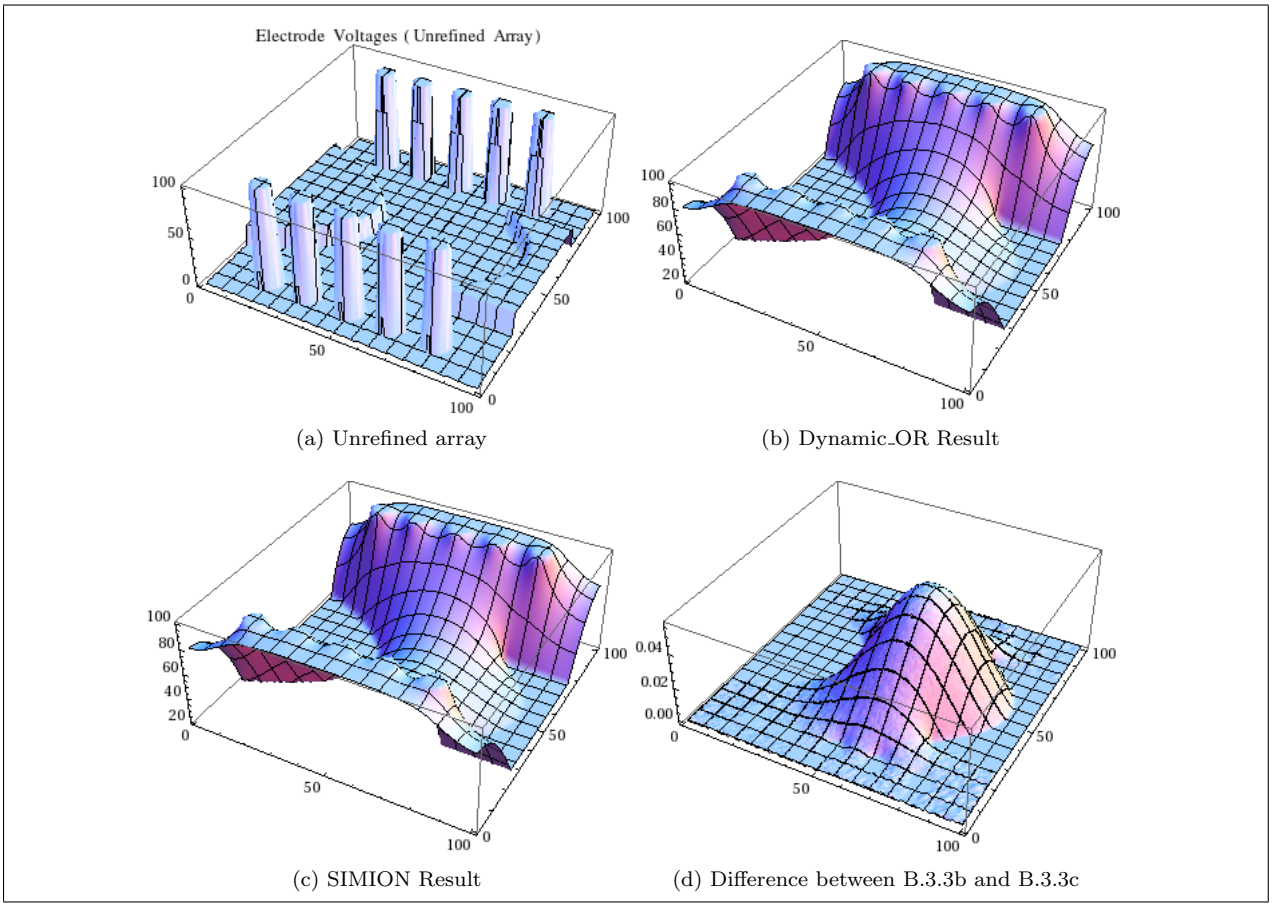
B.4 Ongoing Work

B.4.1 Dynamic_OR

Currently, Dynamic_OR does not give proper results for cylindrically symmetric arrays - arguably the variety most likely to be used for this project. This needs to be fixed if Dynamic_OR is going to be used for further simulations. The SIMION and Dynamic_OR results will be compared to an analytical approach given in [9]. Mathematica to calculate the results for the analytical method.

B.4.2 Mathematica models

A Mathematica notebook needs to be written to calculate results following the approach used in [9] in order to compare these answers to SIMION and Dynamic_OR . It should be possible to implement a Mathematica based model to solve for arbitrary arrangements of electrodes, but it may not be feasible to implement in a robust manner. The algorithm used in Dynamic_OR can be rewritten in Mathematica, which may help show where the discrepancies between Dynamic_OR and SIMION arise, although it is likely that a naïve reimplementation in Mathematica would yield a slower result.



(a) Unrefined array

(b) Dynamic_OR Result

(c) SIMION Result

(d) Difference between B.3.3b and B.3.3c

Figure B.3.3: Results for a sample arrangement of electrodes in Dynamic_OR and SIMION

B.5 Planned Work

After the ongoing work has been completed, further research is required to understand the physical properties and behaviours of plasma. This will enable a simulation of the behaviour of plasma in an electric field. This simulation should be able to predict the position of the plasma and it's density or charge distribution. By combining this with the completed and ongoing work, it should become possible to simulate the behaviour of plasma in an electric field created by an arbitrary arrangement of electrons.

B.5.1 Draft Timetable

- Mathematica models
  - Complete first draft - by 27/02/2012
  - Refine model and compare to Dynamic_OR and SIMION - by 05/03/2012
- Further research
  - Initial summary - by 12/03/2012, then ongoing
  - Expected results for sample electric fields - by 23/04/2012
- Plasma behaviour simulation
  - Initial proof of concept - by 19/03/2012
  - Further development and integration is dependent on results of Mathematica models and Dynamic_OR improvements
  - Initial functional version - by 02/04/2012
- Refine and improve models
  - Description of known errors and results of improvements made - by 27/04/2012, and ongoing as appropriate
- Next progress report due 27/04/2012

By the next progress report, sufficient progress should have been made to generate plausible results for plasma behaviour in at least one electric field, preferably more. Determining how plausible these results are can be aided by researching the expected/observed results from appropriate experiments.

Appendix C

Progress Report 2

C.1 Introduction

The goal of this project is to create and use various methods to calculate the electric field for a given arrangement of charges or electrodes and then simulate the behaviour of plasma in that electric field. The level of detail involved in these simulations will depend on the time available and difficulties encountered during the project.

Since the last progress report has been made, some refinements have been made to the code written prior to submission of the last report and a couple of new areas have been explored.

C.2 Progress

C.2.1 Harmonic Potentials

Mathematica models for harmonic potentials arising from two different arrangements of electrodes were created using the information in [9]. The two arrangements tested involve 3 and 5 cylindrical electrodes respectively, arranged symmetrically about the middle electrode, as shown in figure C.2.1

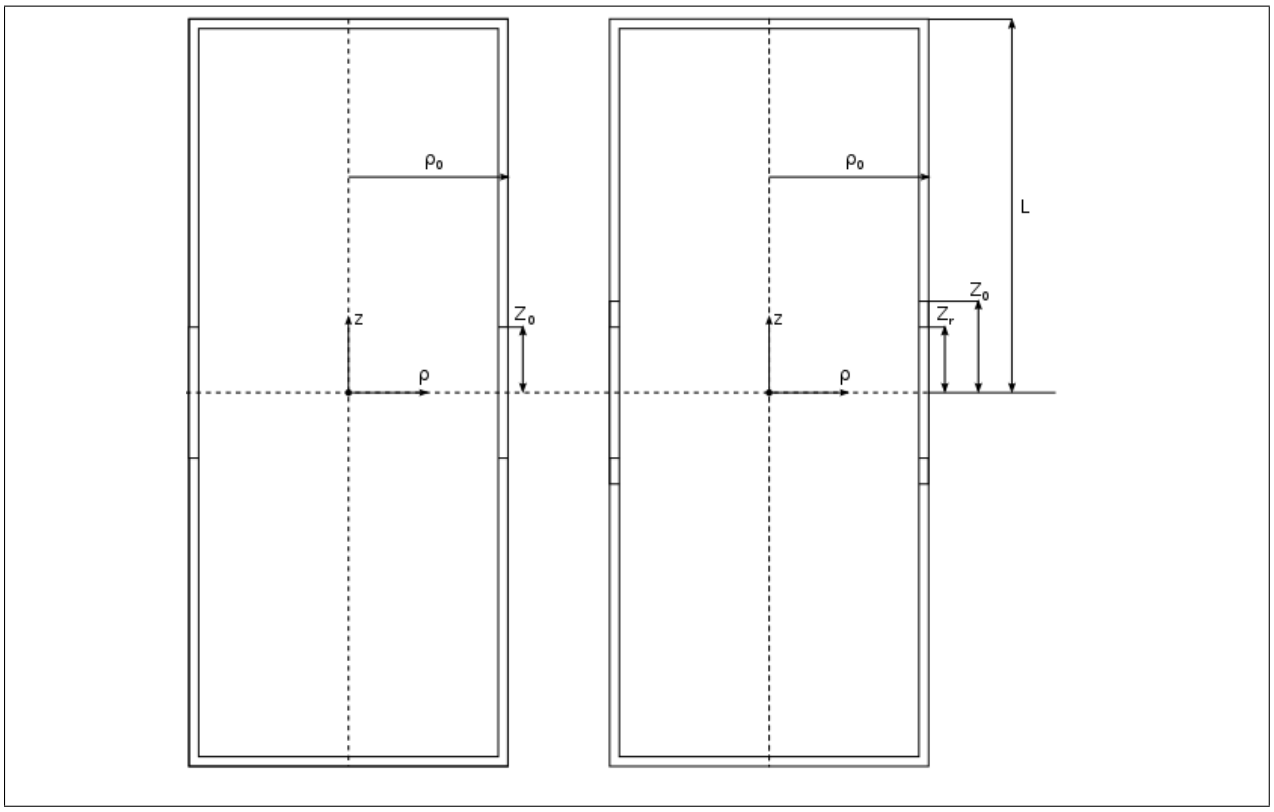


Figure C.2.1: Cross section of cylindrical electrodes for a harmonic potential - 3 (L) and 5 (R) electrode cases

3 Electrodes

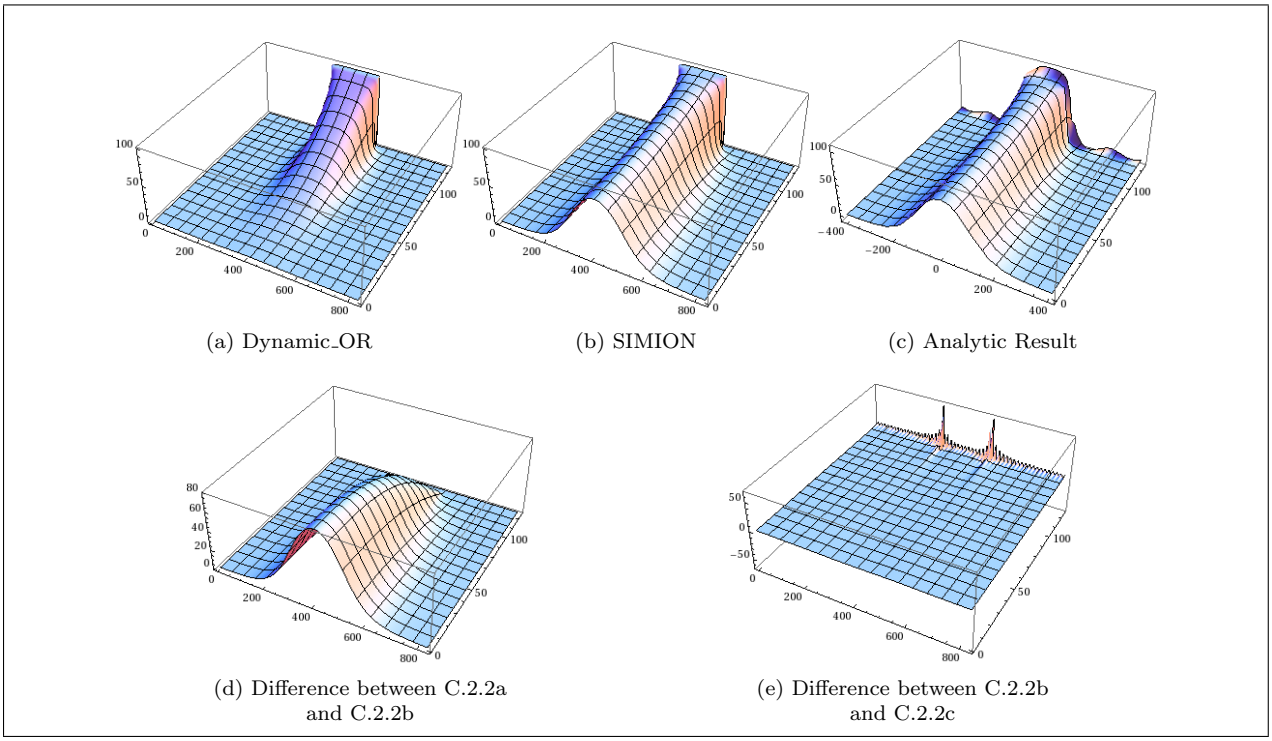
For the 3 electrode case, the outer two electrodes are grounded and the central ring electrode is held at a potential of 100V. The electrode has a length $2Z_0$, with the endcaps having a length $L - Z + 0$. Based on table 2.1 of [9], the value of L was set to $4Z_0$ as there was little difference between the $L = 4Z_0$ and $L = 20Z + 0$ cases discussed, and the smaller L value allowed for faster calculations.

At this point, it was necessary to set a scale between the measurements given in [9] and the grid spacing used in SIMION and Dynamic_OR . It was decided that 200points/cm allowed for a reasonably large grid over which to compare data, while not being unreasonably slow to run through SIMION or Dynamic_OR . The results from SIMION, Dynamic_OR and Mathematica are shown in figure C.2.2. The data shows a good correlation between the results generated by SIMION and Mathematica, with SIMION producing the results significantly faster - while Mathematica can generate a graphical representation of the analytical solution in a reasonably short period of time, it can take up to 15 minutes to generate the information in a format suitable to compare to SIMION style arrays. The results generated by Dynamic_OR diverge from the SIMION and Mathematica results in a manner consistent with the problems described in section 4.1 of the previous progress report, the attempts to resolve these problems will be described later in this report. It should also be noted that the oscillations seen in figures C.2.2c and C.2.2e are related of the number of terms used to generate the graph, and can be reduced by increasing the number of terms in equation C.2.1. As with the choice of number of points per cm made earlier, the number of terms used has selected to balance the time required to calculate the result against the precision required. The difference between the SIMION result and the Mathematica result tends towards zero as the number of terms used increases

$$V = V_0 \times \sum_{n=0}^{\text{terms}} \frac{2}{k_n \times L} \frac{\sin(k_n \times z_0)}{I_0(k_n \rho_0)} \times I_0(k_n \rho) \cos(k_n \times z) \quad \text{(C.2.1)}$$

$$k_n = \frac{\left(n + \frac{1}{2}\right)\pi}{L} \quad \text{(C.2.2)}$$

These are equations 2.18 and 2.17 from [9]



(a) Dynamic_OR (b) SIMION (c) Analytic Result

(d) Difference between C.2.2a and C.2.2b (e) Difference between C.2.2b and C.2.2c

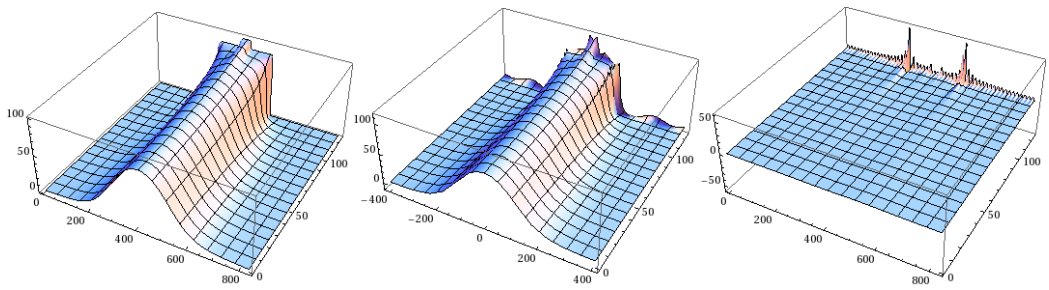
Figure C.2.2: Results for the 3 Electrode harmonic potential and comparison between them

Improving the speed of the Mathematica code

As stated above, using Mathematica to calculate the potential at each point in the array is a slow process, which can take up to 15 minutes depending on the version of Mathematica in use and available hardware. Mathematica supports compiling specific functions in order to gain speed, while sacrificing the arbitrary precision available from 'pure' Mathematica code. Support for this is particularly effective in version 8, which allows the calculations to be parallelised. In Mathematica 8, using `Compile[]` reduced the time required for this calculation from around 320 seconds to around 230 seconds. Comparing the compiled and 'pure' versions showed that the largest difference between the two sets of results was 4.31×10^{-13} on a 32-bit machine. This is small enough to be acceptable for the increased calculation speed.

5 Electrodes

In the 5 electrode case, the central ring electrode with length $2Z_r$ is, in this case, held at a potential of 100V, with a second ring electrode either side at 88.1V with length $2(Z_0 - Z_r)$ and then the two grounded endcaps, as in the 3 electrode case. In this case, $L = 2Z_e$ was chosen using the same reasoning as in the 3 electrode case above. The conversion between distances and grid points was kept at 200points/cm. The SIMION and Mathematica results are shown in C.2.3



(a) SIMION (b) Analytic Result (c) Difference between C.2.3a and C.2.3b

Figure C.2.3: Results for the 5 Electrode harmonic potential and comparison between them

C.2.2 Dynamic_OR Improvements

As mentioned in the previous progress report, Dynamic_OR does not generate reliable results for cylindrical arrays, with the refined arrays tending towards zero along $\rho = 0$. Since that report, a number of attempts have been made to eliminate this error, with little success:

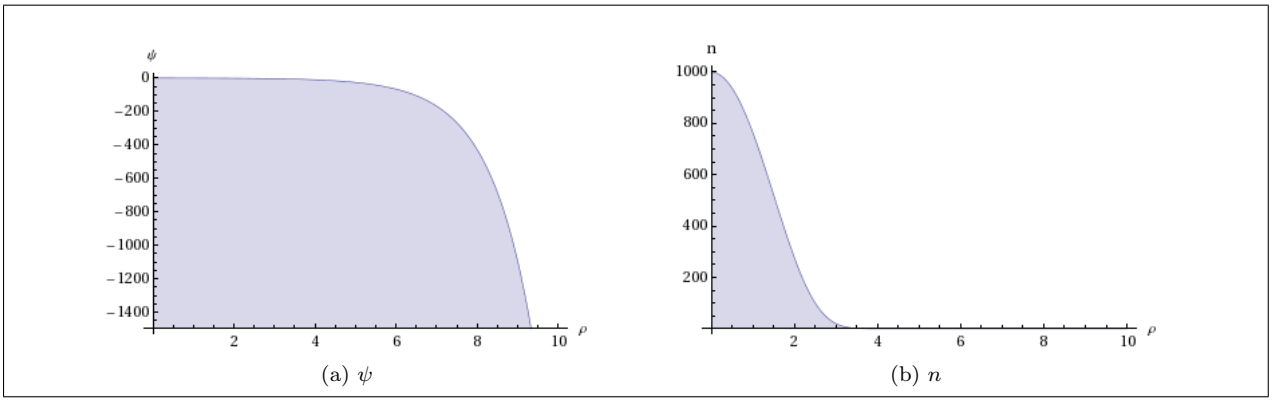
- Change direction of iterations over each axis
- Iterate over array forwards and backwards in each pass
- Artificially mirror array about $\rho = 0$ before iterating
- Force a higher number of iterations before terminating
- Set all non-electrode points to a new potential in an attempt to converge on a different solution

None of these attempts resulted in a noticeable improvement in the data generated by Dynamic_OR . As it is possible to extract data from refined SIMION PA files, this failure is disappointing but not critical. If time permits during the remaining weeks of this project, a 3D planar array containing an example cylindrical array will be created and tested. If this test produces favourable results, then this would be a suitable work around for calculating the field about a cylindrical electrode or set of electrodes. If time permits, the algorithm used to iterate over the cylindrical arrays will be re-derived in order to confirm that the correct coefficients are being used.

C.2.3 Thermal Equilibria of plasma columns

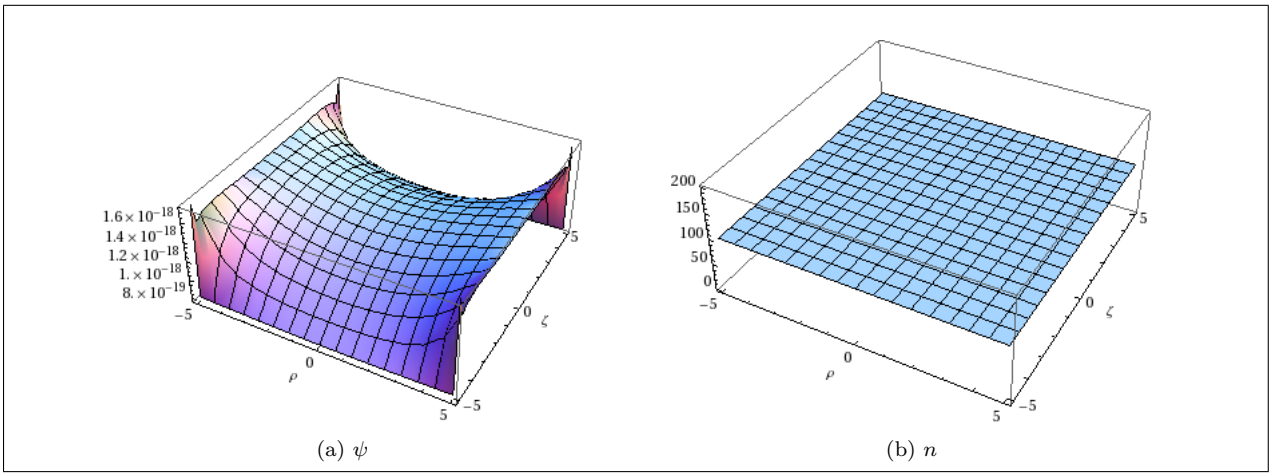
A new part of this project has concerned calculating the density profile for pure electron plasma columns, principally using information contained in [11]. The paper describes the calculations required to plot the density profiles of these plasmas for both the infinite and finite length cases. Plots showing the number density, n, and distribution function ψ for the infinite length column are shown in figure C.2.4, and the corresponding plots for a finite length plasma can be found in figure C.2.5. The infinite length results show the expected result - a plasma which is dense in the centre and thins out away from the origin, with this spread being affected primarily by the angular momentum of the electrons in the plasma.

The finite length results do not produce such neat plots. While a reasonable profile is obtained for Ψ in figure C.2.5a, the overall scale of this result yields a uniformly dense plasma with no fall off in density at any point within the walls of the confining electrodes in the model, as shown in figure C.2.5b. Plotting the dependence of Ψ and n against $\log_1 0(\omega)$ and T shows that at higher temperatures the plasma distribution at the centre seems to decrease. The overall scale of Ψ results in a uniformly dense plasma when the density is plotted. It is possible that this scale difference is caused by incorrect units being used for the constants in the equations. One of the papers cited in [11] has been requested in an attempt to verify the units intended by the paper's authors.



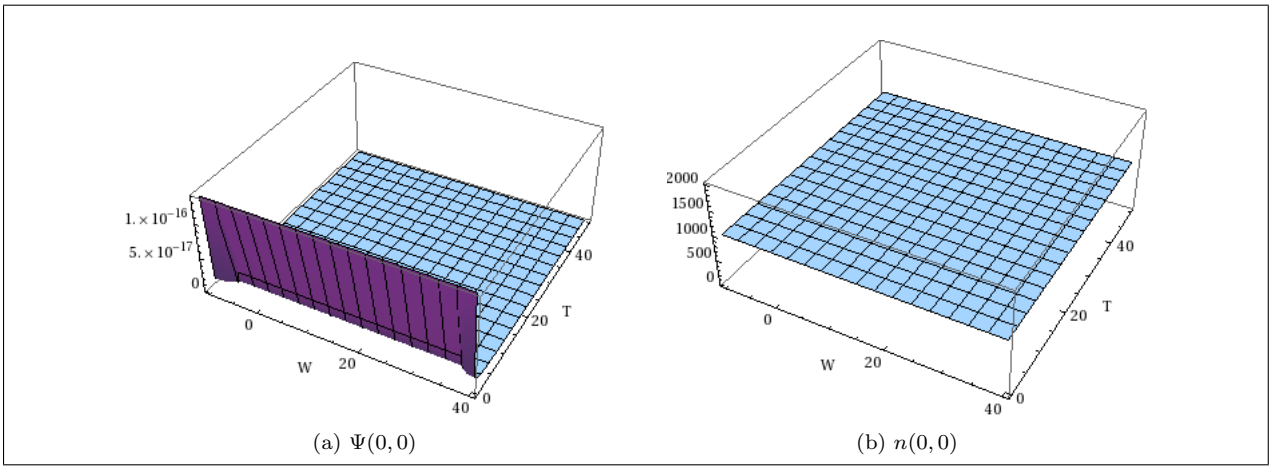
(a) ψ (b) n

Figure C.2.4: ψ and density against radius for an infinite column of electron plasma



(a) ψ (b) n

Figure C.2.5: ψ and density against radius (ρ) and length (ζ) for a finite length plasma



(a) $\Psi(0,0)$ (b) $n(0,0)$

Figure C.2.6: Plots showing dependnce of Ψ and density at the centre and electrode surface on ω and T

C.2.4 Timetable - as at 27/02/12

- Mathematica models:
  Complete first draft - by 27/02/2012, Refine model and compare to Dynamic_OR and SIMION - by 05/03/2012
  This work was done and is described in section C.2.1

- Further research:
  Initial summary - by 12/03/2012, Expected results for sample electric fields - by 23/04/2012
  This referred to the work done so far in section C.2.3. No written summary has been produced, but background reading was done as part of the work completed so far. Some of this will be incorporated into the final write up for this project.

- Plasma behaviour simulation Work has not started to simulate plasma for arbitrary electronic fields, although some arrangements could be approximated using the results for the finite length column at thermal equilibrium

- Refine and improve models
  Description of known errors and results of improvements made - by 27/04/2012, and ongoing as appropriate
  The current state of Dynamic_OR is described in section C.2.2, and further work and improvements will be made as time allows over the remainder of the project

C.3 Planned Work

At time or writing, there are 19 working days remaining for this project. By the end of this time it is planned to (in order of priority):

- Try to correct the errors in the code being used to calculate the finite length column densities

- Test Dynamic_OR against cylinders generated into a 3D potential array and compare to previously generated results

- Research suitable approximations to allow plasma densities to be calculated in an iterative manner

- Rederive cylindrical symmetry algorithm used in Dynamic_OR in an attempt to fix errors

  And of course:

- Complete final write up of project