Finite Difference Method for Partial Differential Equations

Kevin Juan (kj89)

October 24^{th} 2018

The Numerical Methods and Algorithms

The objective of this assignment was to implement a finite difference method on Laplace's Equation for a configuration of electrodes. Laplace's Equation in 2D Cartesian coordinates is shown below:

$$\nabla^2\phi(x,y) = 0 \tag{1}$$

$$\nabla^2\phi(x,y) = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0 \tag{2}$$

In 2D cylindrical coordinates, which will be used in the application of this method, Laplace's Equation can be written as follows:

$$\nabla^2\phi(z,r) = 0 \tag{3}$$

$$\nabla^2\phi(z,r) = \frac{\partial^2\phi}{\partial z^2} + \frac{\partial^2\phi}{\partial r^2} + \frac{1}{r}\frac{\partial\phi}{\partial r} = 0 \tag{4}$$

The finite difference method is one way to solve boundary value problems for PDEs, and is typically easier to program. The other method is the finite element method, which is more difficult to program, but could be more useful for odd shape boundaries or continuum mechanics. The finite difference method calculates the derivative of a point on grid that has been subdivided into many, typically equally spaced, points. To estimate the point (i, j), the values of the 4 nearest points, $(i+1, j)$, $(i-1, j)$, $(i, j+1)$, and $(i, j-1)$ are averaged to get the value of interest. As mentioned earlier, the gird can be subdivided into equally spaced points such that $\Delta x = \Delta y = h$. Due to the nature of a grid system with discrete points, it is important that the boundaries and, for our application, the electrodes line up with the grids for a good finite difference approximation. The boundaries and the potentials at the electrodes are held constant. The constrained boundaries makes the nature of the calculation more stable.

The finite difference equation can be viewed as a large sparse matrix, but with a large number of grid points in each direction, the matrix could easily cost an astronomical amount of memory, making it impractical for any purpose. Relaxation and Gauss-Seidel iteration can easily solve the problem without storing the entire matrix, making it an effective method to solving a finite difference problem. This method uses the smallest amount of memory and is easy to program, but can be slow. The algorithm goes as follows:

- $\phi_{i,j} = \phi(x = ih, y = jh)$
- Set $\phi_{i,j}$ to known constant value on the boundary or electrodes
- Set $\phi_{i,j}$ to some initial guess (typically $\phi_{i,j} = 0$) for all other points on the grid
- For all other points not on a boundary or on an electrode
  - $\phi^{\text{FD}}_{i,j} = \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1})$
  - $\phi^{\text{New}}_{i,j} = \phi^{\text{Old}}_{i,j} + \omega(\phi^{FD}_{i,j} + \phi^{\text{Old}}_{i,j})$
- If $\max|\Delta\phi_{i,j}| = \max|\phi^{\text{FD}}_{i,j} - \phi^{\text{Old}}_{i,j}| > \epsilon_{\text{Max}}$ and $N_{\text{Iter}} < N_{\text{Max}}$, then repeat

This method can be done using one array and overwriting it at each iteration. The parameter ω is called the relaxation parameter. If $\omega = 1$, then it is called relaxation. If $\omega > 1$, then it is called over-relaxation. If $\omega < 1$, then it is called under-relaxation. For this application, we will use over relaxation with repetition, which is also known as successive over-relaxation. To find the optimum over-relaxation parameter, we can track the number of iterations the simulation needs to converge for a large h and pick a value of ω that minimizes the number of iterations needed for convergence. For Laplace's Equation, $1 < \omega < 2$. The advantage of relaxation is that it requires only one location in memory for each grid point, thus reducing the memory requirements down to megabytes in size for 2D as compared to similarly sized problems using no relaxation.

For our application, we will use axial symmetry to reduce a 3D problem down to a 2D problem using cylindrical coordinates. Assuming no azimuthal dependence, we will get Laplace's Equation as shown in Equation 4. Our $\phi^{\text{FD}}_{i,j}$ must be split into two equations: one for the axis $j = 0$ and one of the axis $j > 0$. These are shown below:

$$\phi^{\text{FD}}_{i,j} = \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}) + \frac{1}{8j}(\phi_{i,j+1} - \phi_{i,j-1}); \text{for } j > 0 \tag{5}$$

$$\phi^{\text{FD}}_{i,j} = \frac{1}{6}(4\phi_{i,j=+1} + \phi_{i+1,j=0} + \phi_{i-1,j=0}); \text{for } j = 0 \tag{6}$$

More on finite difference method can be found in §20.0 and §20.5 in Press et al[1] and §17.1 – §17.9 in Paez et al[2].

Electric Fields and Potential

The assignment looks at a configuration of cylindrically symmetric set of electrodes. The following graphic details the electrode configuration in cylindrical coordinates.

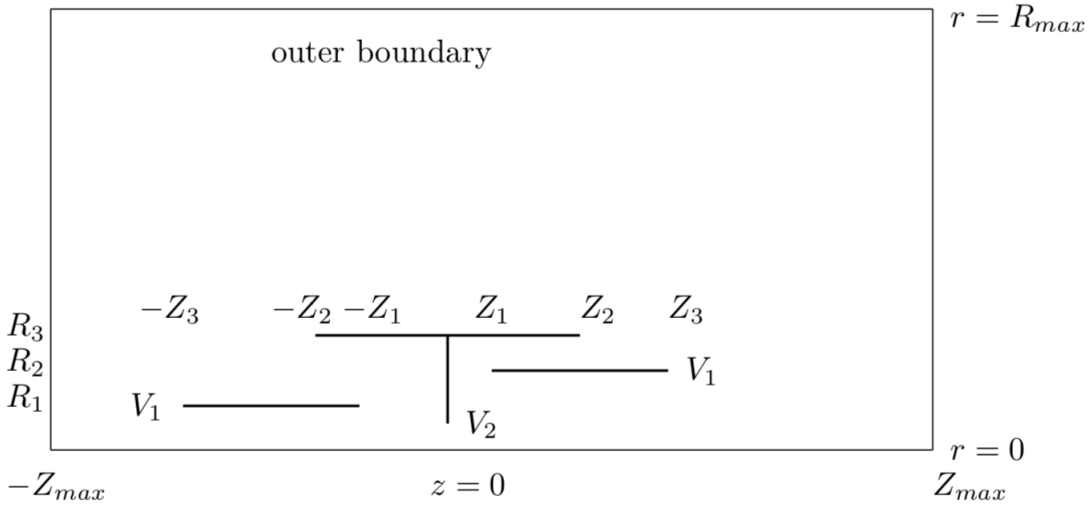

Figure 1: Electrode configuration for the problem of interest.

In the configuration, $V_1 = 0$ Volts and $V_2 = +1000$ Volts. One cylindrical electrode has a radius of R_1 and goes from $-Z_3$ to $-Z_1$. Another cylindrical electrode is at has a radius of R_2 and goes from Z_1 to Z_3. The third cylindrical electrode has a radius of R_3 and goes from $-Z_2$ to Z_2. There is also a disk electrode at $z = 0$ that goes from R_1 to R_3. A device like this could be used to focus charged particles, and is called an electron lens.

r in mm	z in mm
$R_1 = 3$	$Z_1 = 3.5$
$R_2 = 5$	$Z_2 = 8$
$R_3 = 7$	$Z_3 = 15$
$R_{\text{Max}} = 20$	$Z_{\text{Max}} = 25$

Table 1: Coordinate values for the electrode configuration.

By calculating the potential using finite difference method on Laplace's Equation, we can calculate the $\vec{E}$-field and its r and z components. The equation for an electric field is shown below:

$$\vec{E} = -\nabla\phi = -\frac{\partial\phi}{\partial r}\hat{r} - \frac{\partial\phi}{\phi z}\hat{z} \sim -\frac{\phi_{i,j+1} - \phi_{i,j-1}}{2h}\hat{r} - \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2h}\hat{z} \tag{7}$$

If the electrodes were being electronically driven, it would be useful to know the capacitance of the electrodes. After calculating the potential, the capacitance between the

electrodes at V_1 and V_2 and the fringe field can be calculated. The capacitance can be found through the total energy of the $\vec{E}$-field and the energy in the capacitor. The equation for the energy field is shown below:

$$W = \frac{1}{2}CV_C^2 = \frac{1}{2}\epsilon_0 \int |\vec{E}|^2 dv \tag{8}$$

The value of ϵ_0 is 8.8542 pico-Farads.

Implementation and Results

The program uses array class for constructing 2D arrays and bounds checking. At the start of the program, two arrays were initialized: one with the values of the potentials at their designated locations on the grid and a flag array of equal size. The purpose of the flag array is to create an array with static char value. The char values used were 0 and 1. A char value of 1 indicates the position of an electrode while a value of 0 indicates a grid point not occupied by an electrode. This allows the program to check the flag array to determine when the loops encounter an electrode rather than a large if-statement to ensure the value at the electrode position is unchanged by the program. To avoid changing the values of the boundaries, we can simply start loops later or end them earlier. Using char requires less memory than a float would, which is good for an array of static values that only serves to mark locations.

The first part of the assignment was to find an optimal value of ω for $h = 0.1$ mm, which will serve as the relaxation parameter for the rest of the simulations. The values of ω used were [1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.95, 1.99]. The following plot of N versus ω was obtained.

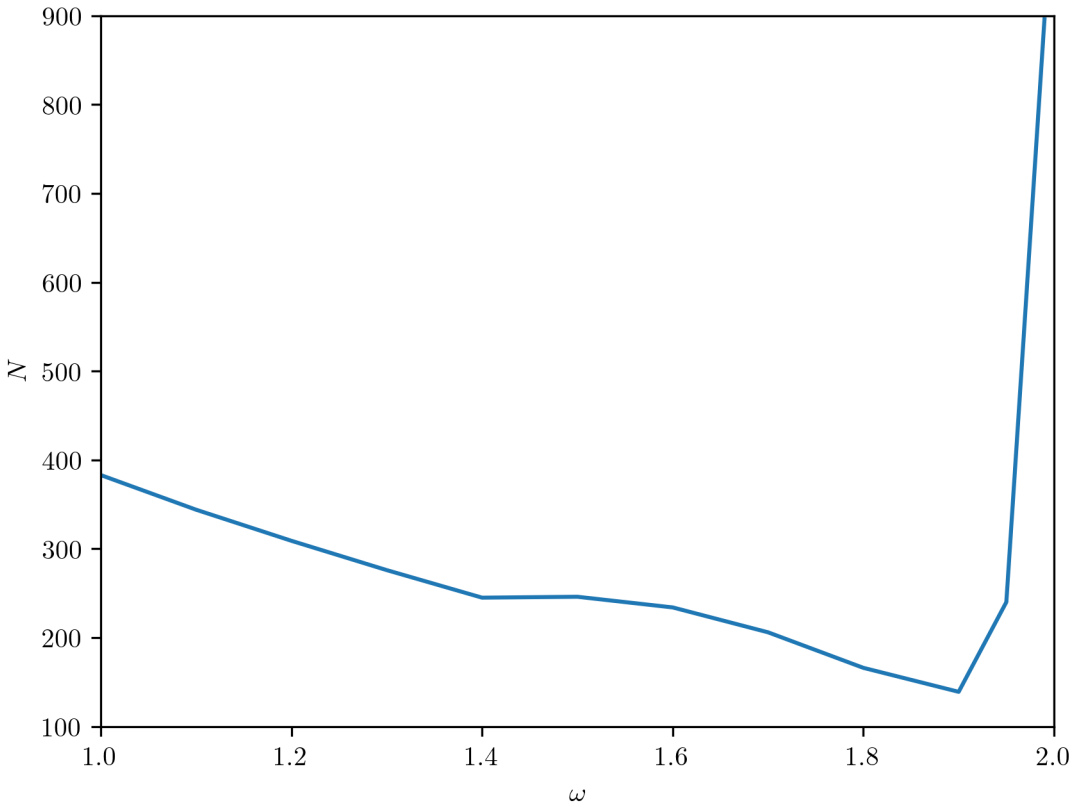

Figure 2: The number of iterations, N, needed for convergence for different values of ω.

The optimal value of ω, ω_{opt}, was found to be 1.9. After ω_{opt} was found, the values of h and the tolerance were changed until 3 or 4 significant digits accuracy was obtained. Throughout the testing, the potential at $r = 0$ and $z = 0$ was tracked. The table below shows the results of these trials.

h in mm	$\lvert\Delta V\rvert_{\text{Max}}$ in V	Tolerance	$V(0,0)$ in V
0.1	0.941063	1	653.505
0.1	0.0995636	0.1	668.292
0.1	0.00997162	0.01	668.321
0.05	0.00997162	1	443.217
0.05	0.0998535	0.1	665.416
0.05	0.00999451	0.01	668.276

Table 2: Trials showing how changing h and the tolerance level changed the accuracy.

During testing, it was found that h smaller than 0.05 took too long to converge, and tolerances lower than 0.01 sometimes did not converge. From this testing, $h = 0.05$ mm and

a tolerance of 0.01 was used throughout the simulation. A contour and surface plot were generated for $V(z, r)$ as shown below.

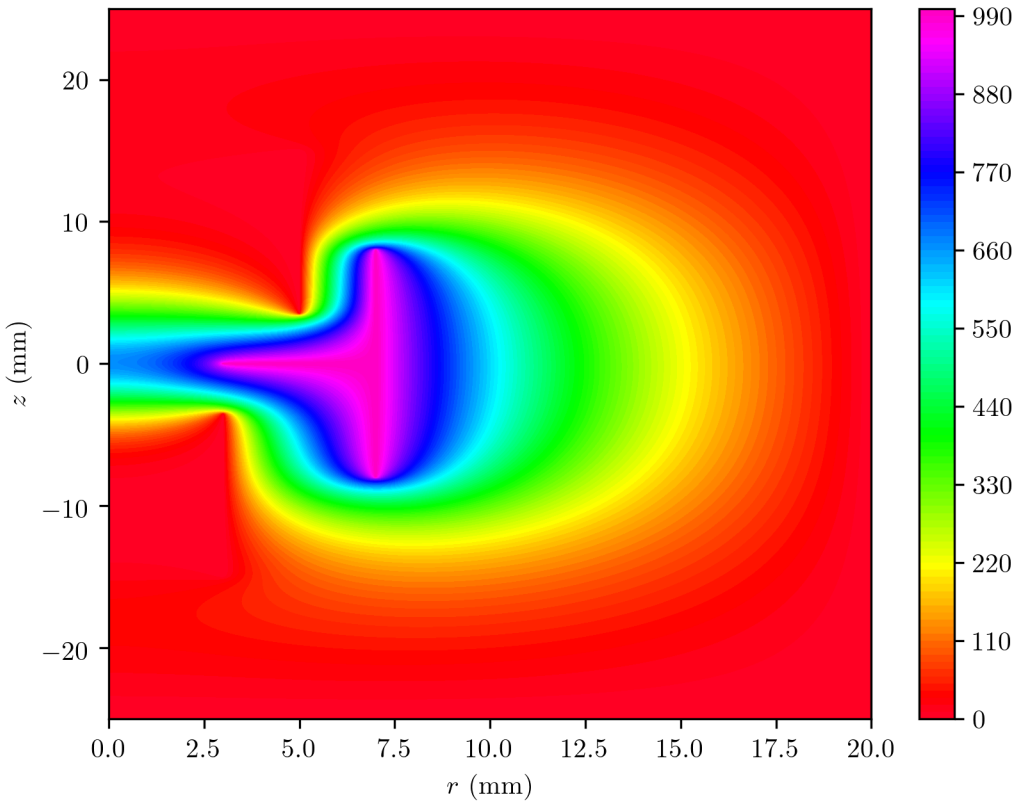

Figure 3: Contour plot of $V(z, r)$ for $h = 0.05$ mm and a tolerance of 0.01.

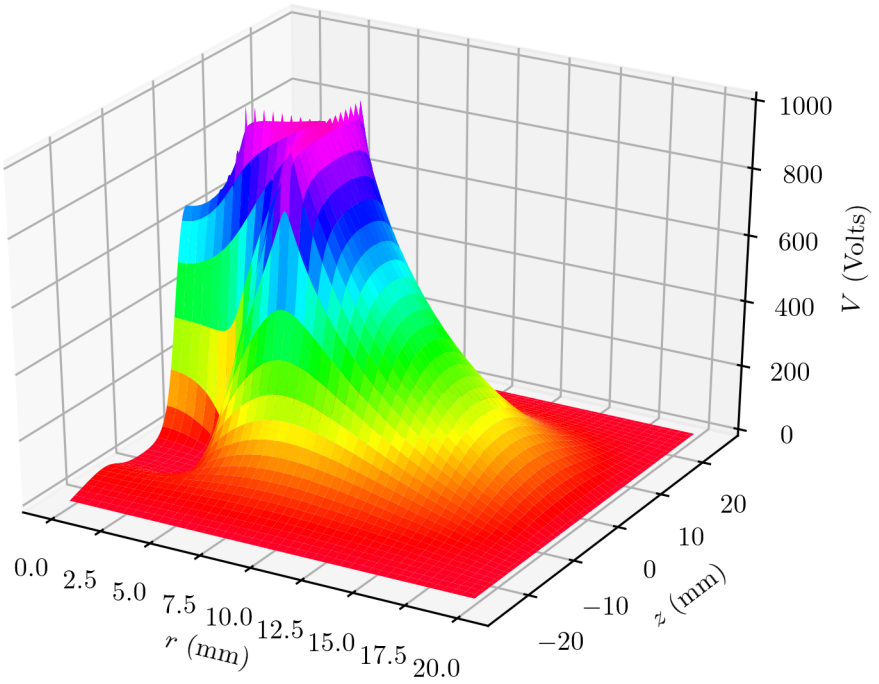

Figure 4: Surface plot of $V(z, r)$ for $h = 0.05$ mm and a tolerance of 0.01.

Maintaining the same spacing and tolerance level, $V(z, r)$ and E_z as a function of z along $r = 0$ mm. These plots are shown below:

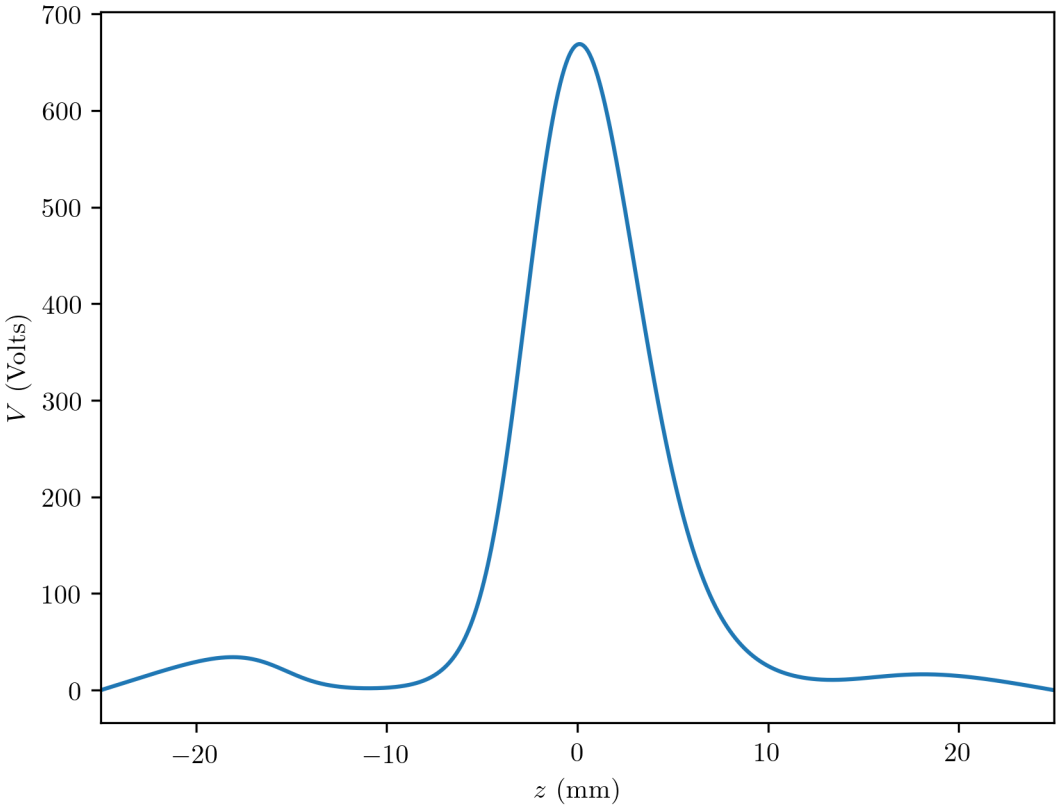

Figure 5: $V(z, r)$ as a function of z along $r = 0$ mm.

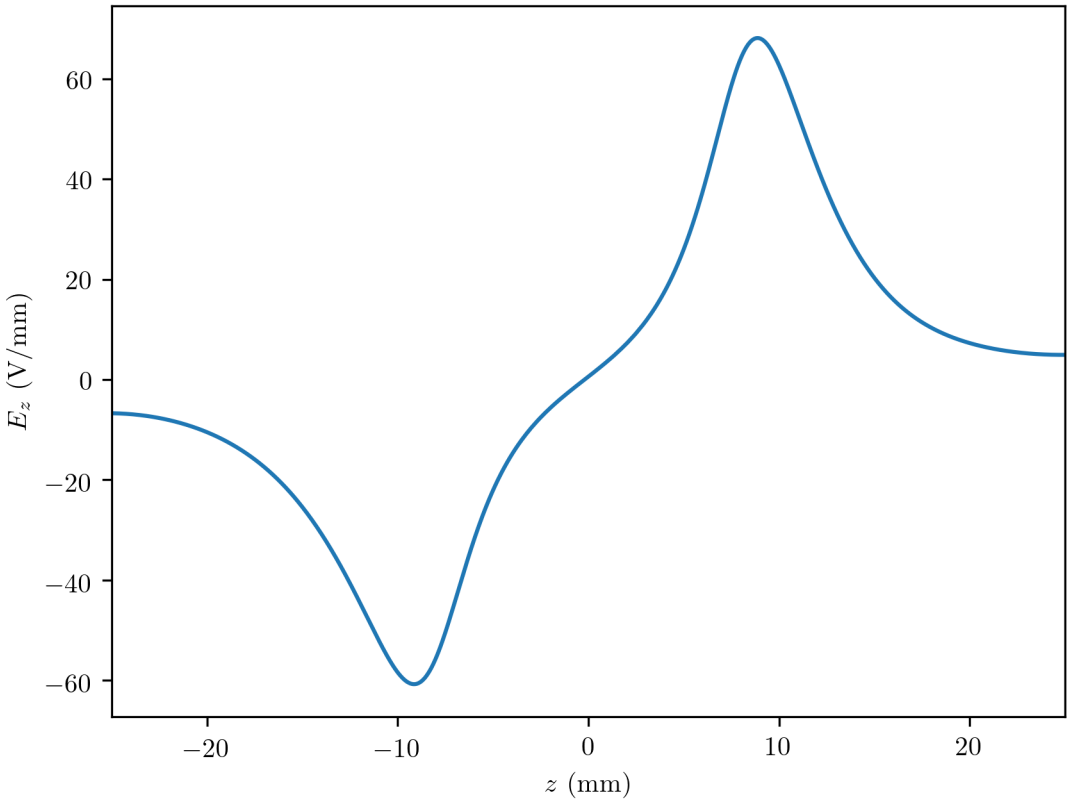

Figure 6: E_z as a function of z along $r = 0$ mm.

Plots for $V(z, r)$, E_z, and E_r were generated at $r = 2.5$ mm for the z-domain of $-10 \leq z \leq +10$. These are shown below:

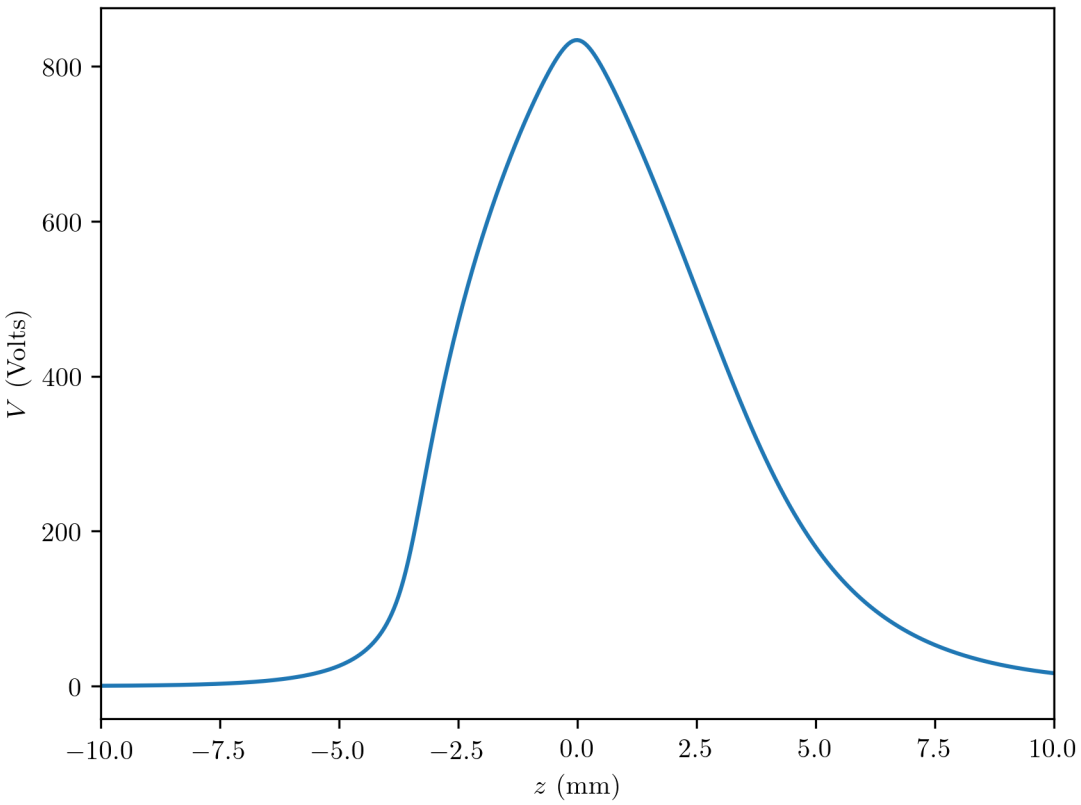

Figure 7: $V(z, r)$ at $r = 2.5$ mm plotted as a function of z.

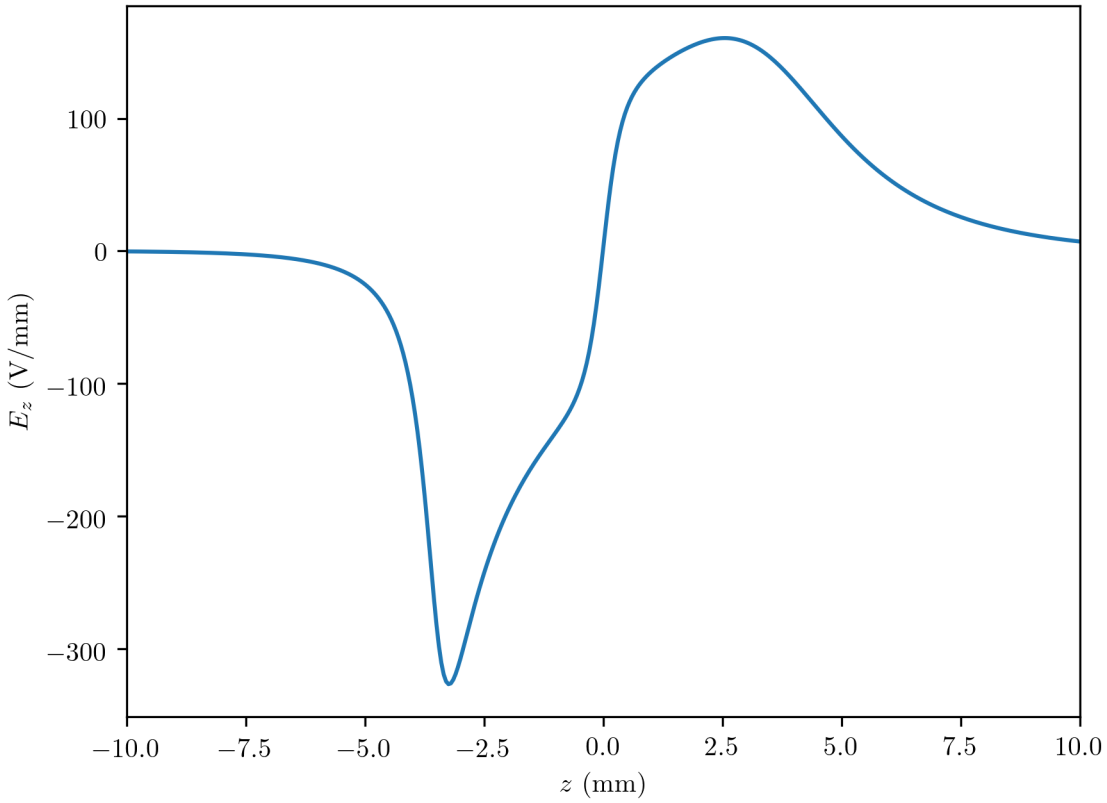

Figure 8: E_z at $r = 2.5$ mm plotted as a function of z.

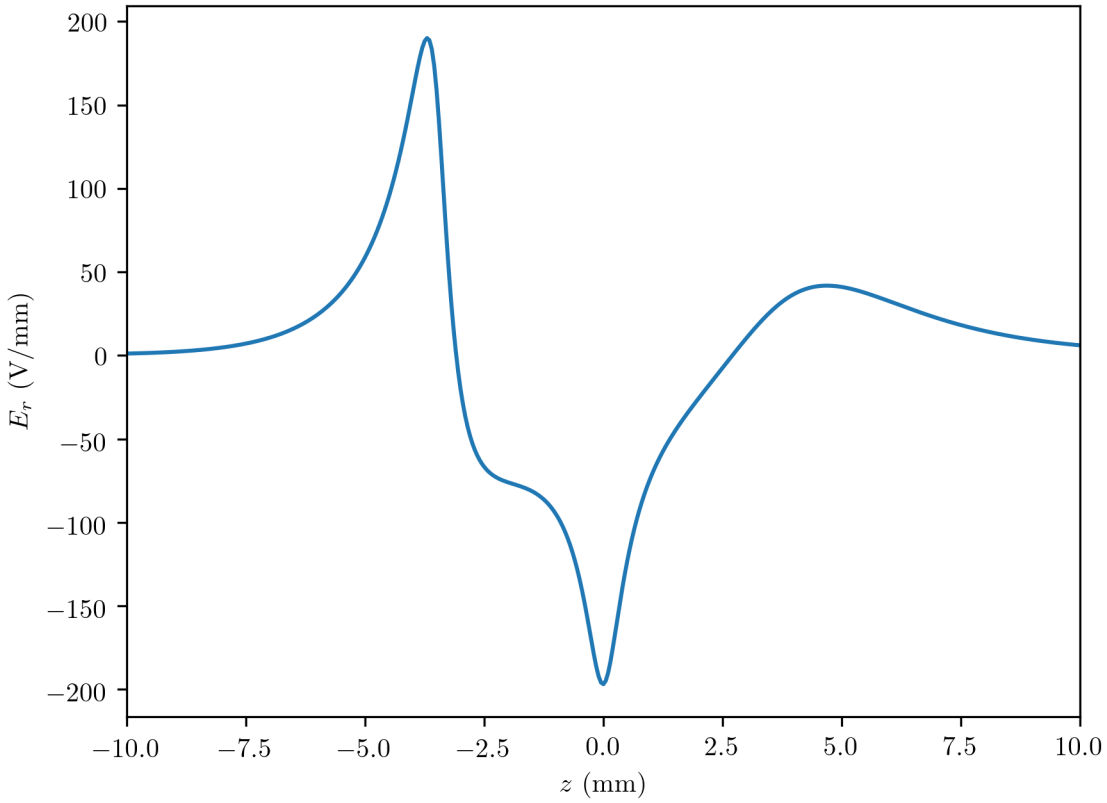

Figure 9: E_r at $r = 2.5$ mm plotted as a function of z.

The value for the capacitance was found to be 3.04743 pF.

Source Code

```
/* AEP 4380 Homework #6

Finite difference method using successive over relaxation
Gauss-Seidel is used
Tested on a potential field with electrodes

Run on a core i7 using clang 1000.11.45.2 on macOS Mojave

Kevin Juan 24 October 2018
*/
```

```
#include <cstdlib>  // plain C
#include <cmath>    // use math package
#include <iostream> // stream IO
#include <fstream>  // stream file IO
#include <iomanip>  // to format the output
#define ARRAYT_BOUNDS_CHECK
#include "arrayt.hpp" // to use arrays

using namespace std;

int main()
{
    float h, w, phiFD, delVMax = 1000.0, tol, dPhi, V2 = 1000.0, V1 = 0.0;
    float totalE = 0.0, ErSq, EzSq, C, VC = V2 - V1;
    int zBound = 50, rBound = 20, i, j, iter = 0, maxIter = 100000;
    const static float e0 = 8.8542 * 1.0e-3;

    // aks user for inputs for w, h, and tolerance
    cout << "What relxation parameter should be used?" << endl;
    cin >> w;
    cout << "What h value should be used?" << endl;
    cin >> h;
    cout << "What level of tolerance should be used?" << endl;
    cin >> tol;

    int zMax = (int)(zBound / h + 1.5), rMax = (int)(rBound / h + 1.5);
    int i25mm = (int)(25 / h + 0.5), j3mm = (int)(3 / h + 0.5);
    int j7mm = (int)(7 / h + 0.5), j5mm = (int)(5 / h + 0.5);
    int i17mm = (int)(17 / h + 0.5), i33mm = (int)(33 / h + 0.5);
    int i10mm = (int)(10 / h + 0.5), i40mm = (int)(40 / h + 0.5);
    int i21p5mm = (int)(215 / (10 * h) + 0.5);
    int i28p5mm = (int)(285 / (10 * h) + 0.5);
    arrayt<float> phi(zMax, rMax);
    arrayt<char> flag(zMax, rMax);
    arrayt<float> Ez(zMax, rMax);
    arrayt<float> Er(zMax, rMax);

    ofstream fpz;
    fpz.open("z_vals.dat");
    if (fpz.fail())
    {
        cout << "cannot open file" << endl;
        return (EXIT_SUCCESS);
    }
    ofstream fpr;
```

```
fpr.open("r_vals.dat");

if (fpr.fail())
{
    cout << "cannot open file" << endl;
    return (EXIT_SUCCESS);
}

// write z and r values to file
for (i = 0; i < zMax; i++)
{
    fpz << setw(15) << i * h - zBound / 2 << endl;
}
for (j = 0; j < rMax; j++)
{
    fpr << setw(15) << j * h << endl;
}
fpz.close();
fpr.close();

// initialize flag array and grid with boundaries and electrodes
for (i = 0; i < zMax; i++)
{
    for (j = 0; j < rMax; j++)
    {
        phi(i, j) = 0.0;
        flag(i, j) = '0';
        if (i == i25mm && (j >= j3mm && j <= j7mm))
        {
            phi(i, j) = V2;
            flag(i, j) = '1';
        }
        else if (j == j7mm && (i >= i17mm && i <= i33mm))
        {
            phi(i, j) = V2;
            flag(i, j) = '1';
        }
        else if (j == j3mm && (i >= i10mm && i <= i21p5mm))
        {
            phi(i, j) = V1;
            flag(i, j) = '1';
        }
        else if (j == j5mm && (i >= i28p5mm && i <= i40mm))
        {
            phi(i, j) = V1;
```

```
            flag(i, j) = '1';
        }
    }
} // end phi and flag initialization

// begin finite difference method with successive over-relaxation
do
{
    delVMax = -1.0;
    for (i = 1; i < zMax - 1; i++)
    {
        for (j = 0; j < rMax - 1; j++)
        {
            if (flag(i, j) == '0')
            {
                if (j == 0)
                {
                    phiFD = (4.0 * phi(i, 1) + phi(i + 1, 0) +
                    phi(i - 1, 0)) / 6.0;
                    dPhi = phiFD - phi(i, j);
                    if (abs(dPhi) > delVMax)
                    {
                        delVMax = abs(dPhi);
                    }
                    phi(i, j) = phi(i, j) + w * dPhi;
                }
                else
                {
                    phiFD = (phi(i, j + 1) + phi(i, j - 1) +
                    phi(i + 1, j) + phi(i - 1, j)) / 4.0 +
                    (phi(i, j + 1) - phi(i, j - 1)) / (8.0 * j);
                    dPhi = phiFD - phi(i, j);
                    if (abs(dPhi) > delVMax)
                    {
                        delVMax = abs(dPhi);
                    }
                    phi(i, j) = phi(i, j) + w * dPhi;
                }
            }
        }
    }
    iter++;
    cout << "Iterations: " << iter << " w: " << w << " delVMax: "
    << delVMax << " V(0,0): " << phi(i25mm, 0) << endl;
} while (iter < maxIter && delVMax > tol);
```

```
// end finite difference method

// write phi values to file
ofstream fpDat;
fpDat.open("phi_vals.dat");
if (fpDat.fail())
{
    cout << "cannot open file" << endl;
    return (EXIT_SUCCESS);
}
for (i = 0; i < zMax; i++)
{
    for (j = 0; j < rMax; j++)
    {
        fpDat << setw(15) << phi(i, j);
    }
    fpDat << endl;
}
fpDat.close();

ofstream fpEz;
fpEz.open("Ez_vals.dat");
if (fpEz.fail())
{
    cout << "cannot open file" << endl;
    return (EXIT_SUCCESS);
}

ofstream fpEr;
fpEr.open("Er_vals.dat");
if (fpDat.fail())
{
    cout << "cannot open file" << endl;
    return (EXIT_SUCCESS);
}

// begin electric field calculations noting that edges are lost
for (j = 0; j < rMax; j++)
{
    Ez(0, j) = 0;
    Ez(zMax - 1, j) = 0;
    for (i = 1; i < zMax - 1; i++)
    {
        Ez(i, j) = -(phi(i + 1, j) - phi(i - 1, j)) / (2.0 * h);
    }
```

```
    }
    for (i = 0; i < zMax; i++)
    {
        Er(i, 0) = 0;
        Er(i, rMax - 1) = 0;
        for (j = 1; j < rMax - 1; j++)
        {
            Er(i, j) = -(phi(i, j + 1) - phi(i, j - 1)) / (2.0 * h);
        }
    } // end electric field calculation

    // write electric field values to files
    for (i = 0; i < zMax; i++)
    {
        for (j = 0; j < rMax; j++)
        {
            fpEz << setw(15) << Ez(i, j);
        }
        fpEz << endl;
    }
    for (i = 0; i < zMax; i++)
    {
        for (j = 0; j < rMax; j++)
        {
            fpEr << setw(15) << Er(i, j);
        }
        fpEr << endl;
    } // end write to file
    fpEz.close();
    fpEr.close();

    // calculate capacitance
    for (i = 0; i < zMax; i++)
    {
        for (j = 0; j < rMax; j++)
        {
            EzSq = Ez(i, j) * Ez(i, j);
            ErSq = Er(i, j) * Er(i, j);
            totalE += j * (ErSq + EzSq);
        }
    }
    cout << totalE << endl;
    C = 2 * h * h * h * M_PI * totalE * e0 / (VC * VC);
    cout << "Capacitance in picoFarads: " << C << " pF" << endl;
} //end calculate capacitance
```

References

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, editors. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK ; New York, 3rd ed edition, 2007. OCLC: ocn123285342.

[2] Rubin H. Landau, Jose Paez, and Christian C. Bordeianu. *A Survey of Computational Physics: Introductory Computational Science*. Princeton University Press, Princeton, 2008.