AEP 4380: Homework 6

Gregory Kaiser

October 30th, 2019

1 Problem Background and Solution Overview

This assignment asked for the electrostatic potential in a cylindrical region for a specific configuration of electrodes held at specific voltages. While the voltages at the electrodes and at the boundaries of the region were held constant, a finite difference method is used to calculate the potential inside the region. The electrostatic potential follows Laplace's equation in cylindrical coordinates:

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

To set up a finite difference that follows the above, the region was first tessellated into a two dimensional array of points. Because the specific configuration has cylindrical symmetry, it is enough to calculate the potential for a slice at given angle around the z axis, with variable radius, and variable z value. These points are at a distance h apart in both directions. The choice of h involves quite a bit of work, described in more detail in the Solution Description. Where $R_1 = 2mm$,

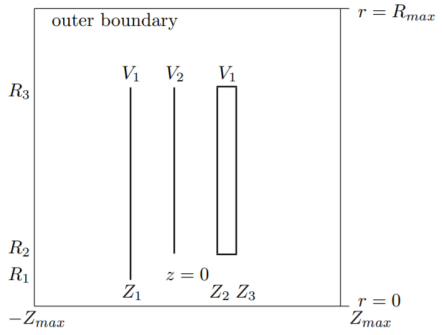


Figure 1: The electrode configuration we were asked to solve.
[2]

$R_2 = 3mm$, $R_3 = 10mm$, $R_{max} = 15mm$, $Z_1 = -2mm$, $Z_2 = 1.5mm$, $Z_3 = 3mm$, $Z_{max} = 15mm$ above.

In order to calculate this accurately, and to allow the potential to permeate the space being analyzed, successive over-relaxation (SOR) was used. Gauss-Siedel iteration essentially low-pass-filters each calculation of a new potential at a given spot, weighting the previous value and the new one

differently. Since electrostatic potentials in free space follow Laplace's equation, the finite difference calculated for a single point ends up resembling an average over its neighbors. The details of calculating this in cylindrical coordinates are described in more detail below.

Once the electrostatic potential is found at every point on this grid, other quantities can be found, plotted, and analyzed. The electric field, given by $\vec{E} = -\nabla\phi$, is of particular interest at $r = 0$. Since this configuration of cylindrical electrodes is meant to resemble an electron-focusing beam, that region is important to calculate for the manufacture of such devices. In addition, an integral can be performed over the electric field to find the overall capacitance of the plate configuration given. This value is also practically important to manufacturing and use of the potential device.

2 Solution Description

2.1 Laplace's Equation and Problem Initialization

To begin, Laplace's equation is broken up into approximate derivatives in the radial and axial directions using a central difference method for any given point with index i, j, where $z = jh$ and $r = ih$:

$$\phi_{i,j} = \frac{1}{4}\left[\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}\right] + \frac{1}{8i}\left[\phi_{i+1,j} - \phi_{i-1,j}\right] \tag{2}$$

With the special case that on the axis where $i = r = 0$:

$$\phi_{i,j} = \frac{1}{6}\left[4\phi_{1,j} + \phi_{0,j+1} + \phi_{0,j-1}\right] \tag{3}$$

[2] Note: My axes start at $(i, j) = (0, 0)$ at $(r, z) = (0, -Z_{max})$ By calculating a point's potential based on its neighbors, over many iterations this should reach the steady solution for that point.

The positions in millimeters of each plate in both the radial and axial directions are rounded to corresponding indices in the two dimensional array by dividing by a chosen h value. The set of values at each point is initialized as an arrayt$< float >$ type two-dimensional array of floats. The arrayt class given to us by Professor Kirkland was critical in debugging, since it allowed for out of bounds errors common to other languages with more safety built into their compilers.[3] It was initialized alongside an arrayt$< char >$ two-dimensional array of flag variables which allowed the boundary conditions of the system to be flagged as constant, unchangeable numbers.

Points not on a constant boundary are set to zero by default.

2.2 Gauss-Siedel Iteration

A while loop with two nested for-loops control calculations at each given point in the system. A single point's potential is given as $\phi_{i,j}$, and the value calculated from its neighbors, following Laplace's equation in cylindrical coordinates as in section 2.1, is $\phi^{FD}_{i,j}$. The difference between these values is calculated for each i and j value:

$$|\Delta V|_{i,j} = \left|\phi_{i,j} - \phi^{FD}_{i,j}\right| \tag{4}$$

The maximum $|\Delta V|_{i,j}$ in one cycle through the matrix of points is stored. A tolerance on this difference is set before the while loop begins, in order to compare with the maximum difference found. If $|\Delta V|_{max}$ is less than the tolerance, we have found the solution to within some desired error bound (not necessary an error equivalent to that tolerance value). The new value for a given i and j is calculated as:

$$\phi_{i,j}^{new} = (1-\omega)\phi_{i,j} + \omega\phi_{i,j}^{FD} \tag{5}$$

where ω is the relaxation parameter which controls the weighting of $\phi_{i,j}$ and $\phi_{i,j}^{FD}$. This acts as a kind of low-pass filter, where the new calculated value is weighted with the value that already exists to slow the response and approach a steady state. This is important because this algorithm overwrites the values used on any given iteration in the same piece of memory. We are not storing a purely new and a purely old matrix, so we need to take each calculation with a grain of salt to make sure we aren't using too many newly set values to calculate a point which should've been found with purely old values surrounding it.

2.3 Finding Good Values for ω, h, and $|\Delta V|_{max}$

The value of ω is important because the algorithm would take many iterations to settle if we were taking any finite-difference calculation at face value. It might also take a long time if we weren't taking the new calculation into account much at all. Therefore, an ω somewhere between 1.0 and 2.0 must be chosen. An optimal value does happen on this range, so for a set value of $h = 0.1mm$, I varied ω and produced the following plot:

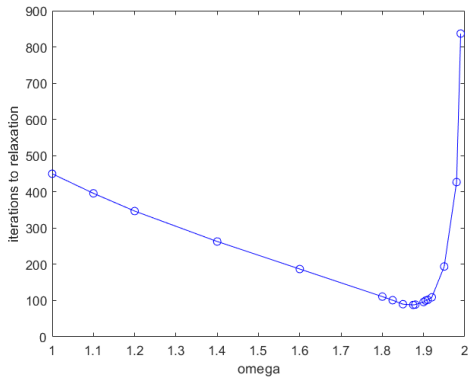


Figure 2: Number of iterations to relaxation for fixed h and $|\Delta V|_{max}$ as a function of relaxation parameter ω.

From this I concluded that at $h = 0.1mm$, $\omega_{opt} \approx 1.875$, which I lowered to $\omega \approx 1.85$ predicting that a lower h value would require some corresponding shift in omega.

Then, to find good values of h and $|\Delta V|_{max}$, each tolerance was varied over a few values until small changes in either parameter had little effect on the result. Since V was on the order of several hundred, I counted variations of a few Volts as negligible compared to the magnitude of V, and therefore stable. I also took into account the fact that, for $h < .02$, the algorithm would take a very long time to run.

V(h,$\|\Delta V\|_{max}$)			
$\|\Delta V\|_{max}$	h=.025mm	h=0.05mm	h=0.1mm
.0001		706.541	
.001	703.212	706.482	712.552
.01	698.572	705.766	712.472
.1	635.7	688.281	710.949
1	43.5061	551.521	678.282

Keeping speed up and keeping $V(r,z)$ relatively stable, I decided to stick with $h = .05$ and $|\Delta V|_{max} = .001$. Variations in $|\Delta V|_{max}$ of an order of magnitude in either direction caused almost no change in $V(r,z)$, and large variations in h caused changes as high as .9%. This means that these values are perfectly fine and keep my run time low enough for my sanity to remain intact.

2.4 Calculating $\vec{E}(r,z)$ and Capacitance

Once the potential $V(r,z,)$ is found, calculating $\vec{E}(r,z)$ was fairly straightforward. The electric field follows the following relation:

$$\vec{E}(r,z) = -\nabla\phi = -\frac{\partial\phi}{\partial r}\hat{r} - \frac{\partial\phi}{\partial z}\hat{z} \tag{6}$$

Which, on my two-dimensional array in cylindrical coordinates reduces to a central difference method derivative with respect to each coordinate:

$$E_r = -\frac{\phi_{i+1,j} - \phi_{i-1,j}}{2h}; E_z = -\frac{\phi_{i,j+1} - \phi_{i,j-1}}{2h} \tag{7}$$

As before, i represents the radial parameter, r, and j represents the axial parameter, z. Since $\vec{E}(r,z)$ is zero at fixed conductive boundaries, the central difference within this bound was easy to implement. Given the cylindrical symmetry of the system, $E_r(r=0) = 0$. See results section for plots of this information.

To find the capacitance of this system, I solved the relation for energy stored in a capacitor charged to voltage V_c:

$$\frac{1}{2}CV_c^2 = \frac{1}{2}\epsilon_0 \int \left|\vec{E}\right|^2 dv \tag{8}$$

[2]

This implies that:

$$C = \frac{\epsilon_0}{V_c^2}\int \left|\vec{E}\right|^2 dv = \frac{h^2 2\pi\epsilon_0}{V_c^2}\int \left|\vec{E}\right|^2 r dx dy \tag{9}$$

Since $dv = 2\pi r dx dy = 2\pi r h^2$, and since $r = ih$ at any point in the array, I summed $r\left|\vec{E}\right|^2$ at each point in space and then multiplied by the constants after the accumulation was finished:

$$C = \frac{h^3 2\pi\epsilon_0}{V_c^2}\sum_{i=0,noflag}^{rows-1}\sum_{j=0,noflag}^{cols-1}\left(E_i^2 + E_j^2\right)i \tag{10}$$

Which, after compensating for the unit conversion from mm to m, and using $\epsilon_0 = 8.8542\frac{pF}{m}$, yields $C \approx 3.34pF$. This value is fairly reasonable given that the capacitive plates are on the order of millimeters, and they are separated by distances comparable to their other dimensions. Capacitance is then on the order of ϵ_0.

3 Results and Interpretation

The region of interest for this device is along the axis at $r = 0$, since an electron beam would travel down the center path along that trajectory.

The voltage along that line increases to a peak at $z = 0$. $\vec{E}(r = 0, z)$ follows the negative slope of that curve, having sharpely negative value to the left of center, and strong positive peak towards the right:

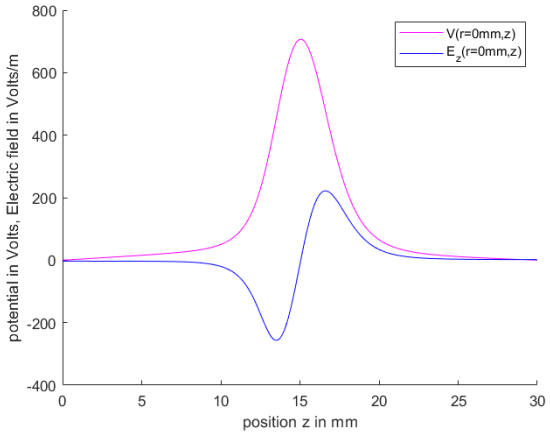
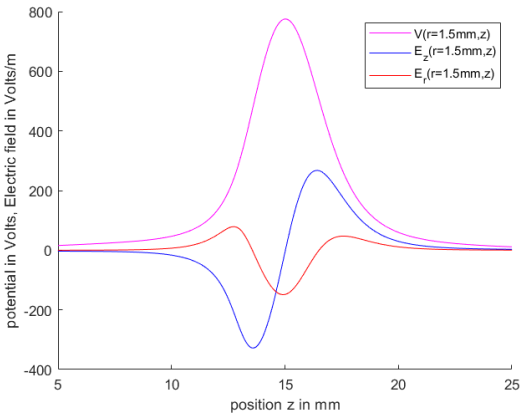


Figure 3: $E_z(r = 0, z)$ and $V(r = 0, z)$ from $-Z_{max}$ to Z_{max}. Note that my z-axis is shifted such that z=0 corresponds to z=15mm on my graphs.

This makes sense because particles on the axis should see the same potential everywhere radially, but see the set of electrodes along the axis. Therefore, you would expect that an electron would be attracted to that higher potential area near the center of the apparatus from along the axis. Due to the asymmetry along the axis however, the electron might change its velocity (go faster or slower) depending on the direction from which it approaches the center.

At a larger value of $r = 1.5mm$ there is some more interesting behavior of $V(r = 1.5mm, z)$ and $\vec{E}(r = 1.5mm, z)$:



tt

Figure 4: $E_z(r = 1.5mm, z)$, $E_r(r = 1.5mm, z)$, and $V(r = 1.5mm, z)$ from -10mm to 10mm. Note that my z-axis is shifted such that z=0 corresponds to z=15mm on my graphs.

I can see how this makes sense also, since electrons at some $r > 0$ will feel a force from the electric field radially outward towards the high potential electrode when they near the center of the device. However, due to the asymmetry across that region, the paths of the electrons will end up being focused to a narrow path that crosses at some focal point. Since the peak of E_r is a bit higher on the left, I think this means that the beam is bent a bit stronger towards the axis on the left. This leads me to believe that if a mostly straight beam comes from the right, it will be bent towards a focal point on the left. Perhaps my logic is reversed, but regardless, this asymmetry can be very useful in applications which require careful deposit (or a tight high energy beam) of electrons.

The final potential plot looks like the classic capacitor plate imagery that I expected, with a high potential at electrode 2, and fringe fields billowing out of the edge of the grounded outer plates:

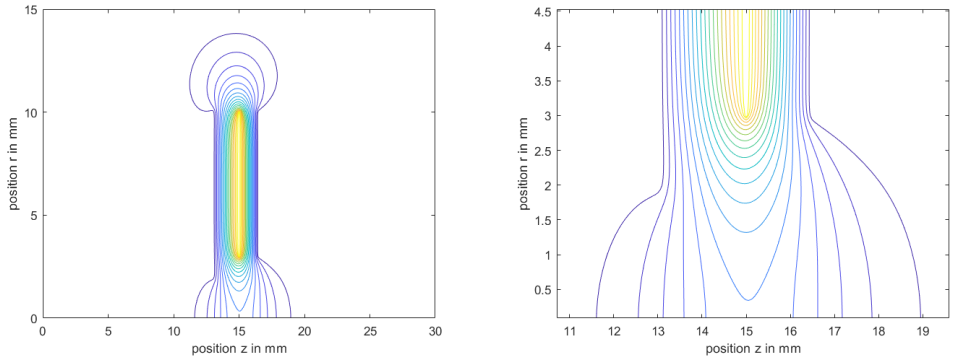


Figure 5: $V(r, z)$ plotted with MATLAB contour plot. Left: Full view with 40 contours. Right: Zoomed detail on axis plot.

These curving lines are analogous to a lens' surface, since the electric field, and therefore applied force, will follow the gradient (or negative gradient, for positively charged species) of these contours. Seeing the curvature higher on the left indicates to me that that is a location of higher lensing power, than the curvature on the right. It makes sense that the characteristics of this device depend on that asymmetry of potential.

For more numerical detail and error checking, fewer contours with printed labels were somewhat useful:

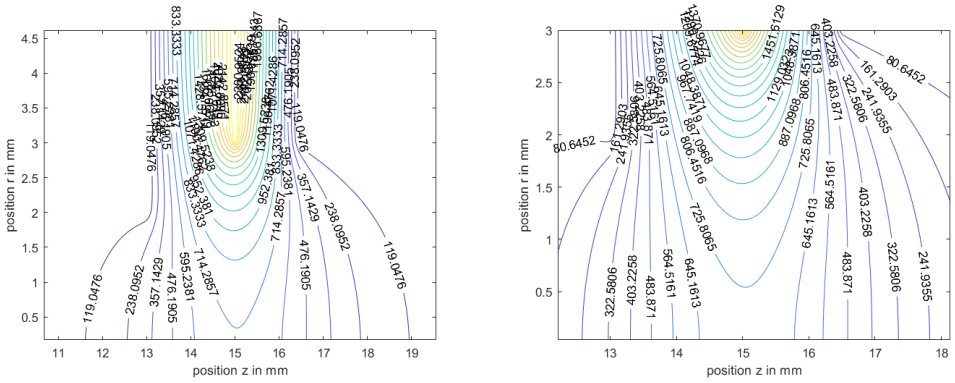


Figure 6: $V(r, z)$ plotted with MATLAB contour plot. Left: Displayed potential values. The center point had voltage hovering around 700V depending on grid size and $|\Delta V|_{max}$. Right: Zoomed detail on axis.

Overall I found this method of solution very interesting, since it incorporated a variety of topics that I've studied previously, including finite difference methods (through graphics) and electromagnetism. In particular, I think this could be an interesting method of solving heat conduction equations in room geometries. Using whatever driving sources exist like windows, lights, or people, one could calculate the steady state temperature of a room based on conduction between discrete sections of the walls, convection to volume cubes of air, and radiation from one surface piece to another. I know that solutions to lighting scenarios are done on graphics hardware like this, as mentioned briefly in class, but incorporating energy of other kinds could add an extra challenge.

References

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes, The Art of Scientific Computing*. (3rd edit.), Cambridge Univ. Press, 2007, (ISBN 978-0-521-88068- 8, QA297 .N866 2007)

[2] Kirkland, Earl. *Boundary Value Problems and Relaxation.* AEP 4380 Fall 2019 Assignment 6. https://courses.cit.cornell.edu/aep4380/secure/hw06f19.pdf

[3] Kirkland, Earl. *Array Class Objects in C/C++ for Vectors and Matrices* AEP 4380 Fall 2019. https://courses.cit.cornell.edu/aep4380/secure/arrayt.hpp

Source Code

```
/*  AEP 4380 Assignment #6
    Boundary Value Problems and Relaxation

    Run on Windows core i7 with gcc version 8.2.0 (MinGW.org GCC-8.2.0-3)

    Gregory Kaiser October 30th 2019

*/
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <iomanip>

#define ARRAYT_BOUNDS_CHECK
#include "arrayt.hpp"//from https://courses.cit.cornell.edu/aep4380/secure/arrayt.hpp

using namespace std; //makes writing code easier
int i,j;//loop stuff
double pi = 4.0*atan(1.0);
double eps = 8.8542e-12;//Farads

int main(){

    ofstream fp; //output file
    fp.precision(7);

    fp.open("hw6_1.dat");
    if(fp.fail()){
        cout<<"cannotopenfile"<<endl;
        return(EXIT_SUCCESS); //defined by standard library
    }

     ofstream fp2; //output file
    fp2.precision(7);

    fp2.open("hw6_r.dat");
    if(fp2.fail()){
        cout<<"cannotopenfile"<<endl;
        return(EXIT_SUCCESS); //defined by standard library
    }

     ofstream fp3; //output file
    fp3.precision(7);

    fp3.open("hw6_z.dat");
    if(fp3.fail()){
        cout<<"cannotopenfile"<<endl;
        return(EXIT_SUCCESS); //defined by standard library
    }

    ofstream fp4; //output file
    fp4.precision(7);

    fp4.open("hw6_wopt.dat");
    if(fp4.fail()){
```

```
    cout<<"cannotopenfile"<<endl;
    return(EXIT_SUCCESS); //defined by standard library
}

ofstream fp5; //output file
fp5.precision(7);

fp5.open("hw6_Er.dat");
if(fp5.fail()){
    cout<<"cannotopenfile"<<endl;
    return(EXIT_SUCCESS); //defined by standard library
}

ofstream fp6; //output file
fp6.precision(7);

fp6.open("hw6_Ez.dat");
if(fp6.fail()){
    cout<<"cannotopenfile"<<endl;
    return(EXIT_SUCCESS); //defined by standard library
}

//HW Assignment
//------------electrode constants-------------
double r1=2,r2=3,r3=10,rmax=15,z1=-2,z2=1.5,z3=3,zmax=15,v1=0,v2=2500;
//to hit these values exactly, h spacing must be in 0.5mm/integer increments
double h=.05;//deltax=deltay=h
int rows = (int) rmax/h+1;
int cols = (int) 2*zmax/h+1;
arrayt<float> phi(rows,cols);
arrayt<char> flag(rows,cols);
int iter=0,max_iter=10000;

//------------initialization---------

//round electrode values to points in the array based on h choice
int e1_rmin = (int) (r1/h);
int e1_rmax = (int) (r3/h);
int e1_z = (int) ((z1+zmax)/h);
int e2_rmin = (int) (r2/h);
int e2_rmax = e1_rmax;
int e2_z = (int) ((zmax)/h);
int e3_rmin = e2_rmin;
int e3_rmax = e1_rmax;
int e3_zmin = (int) ((z2+zmax)/h);
int e3_zmax = (int) ((z3+zmax)/h);

//a bit of double counting, fill initial condition
for(i=0;i<rows;i++){//from r to rmax
    for(j=0;j<cols;j++){//from -zmax to zmax
        if(j==e1_z&&i>=e1_rmin&&i<=e1_rmax){//electrode 1
            phi(i,j)=v1;
            flag(i,j)=1;
        }
        else if(j==e2_z&&i>=e2_rmin&&i<=e2_rmax){//electrode 2
            phi(i,j)=v2;
            flag(i,j)=1;
        }
        else if(j>=e3_zmin&&j<=e3_zmax&&i>=e3_rmin&&i<=e3_rmax){//thick electrode 3
```

```
            phi(i,j)=v1;
            flag(i,j)=1;
        }
        else{
            phi(i,j) = 0;
            flag(i,j) = 0;
        }
    }
}
//boundary
for(i=0;i<rows;i++){//from r to rmax
    flag(i,0) = 1;
    flag(i,cols-1) = 1;
}
for(j=0;j<cols;j++){//from -zmax to zmax
    flag(rows-1,j) = 1;//outer cylinder surface
}
double tempfd=0;
double errortemp=0;
double tol = .001;//tolerance on change per iteration
double deltamax=tol;//change between iterations of the relaxation method
//setting it to tol just gets me into the first loop
//it gets set higher than tol on the first go around of each calculation
double w = 1.85;//relaxation parameter
while(deltamax>=tol&&iter<max_iter){
    //relax_step(phi, flag, deltamax);
    //reset delta max since this is a new relaxation step
    deltamax=0;
    for(i=0;i<rows-1;i++){//from r to rmax
        for(j=1;j<cols-1;j++){//from -zmax to zmax
            if(!flag(i,j)){//can modify
                if(i==0){//on axis special case
                    tempfd = (4.0*phi(1,j)+phi(0,j+1)+phi(0,j-1))/6.0;
                }
                else{
             tempfd = .25*(phi(i+1,j)+phi(i-1,j)+phi(i,j+1)+phi(i,j-1))+(phi(i+1,j)-phi(i-1,j))/(8*i);
                }
                //each i,j is unique position, so here i can calculate what the new value should be
                //as well as what the change was
                errortemp = fabs(tempfd-phi(i,j));
                if (errortemp>deltamax){
                    deltamax = errortemp;//find max difference (not really error)
                }
                phi(i,j)=(1-w)*phi(i,j)+w*tempfd;//replace phi new with weighted value
            }
        }
    }
    iter++;
}
//prinouts used in optimizing omega, h and the max difference
cout<<iter<<endl;
cout<<deltamax<<endl;
cout<<phi(0,e2_z)<<endl;//probe the center of the apparatus
cout<<h<<endl;

//--------Calculate E field------------------
float E_r_temp=0;
float E_z_temp=0;
float cap_integ_sum = 0;
```

```
for(i=0;i<rows;i++){
    for(j=0;j<cols;j++){
        if(!flag(i,j)){//non electrode,non boundary point
            if(i==0){//on axis special case, symmetry so E_r is zero
                E_r_temp = 0;
                E_z_temp = -(phi(i,j+1)-phi(i,j-1))/(2*h);//central difference along z
            }
            else{
                E_r_temp= -(phi(i+1,j)-phi(i-1,j))/(2*h);//central difference along r
                E_z_temp = -(phi(i,j+1)-phi(i,j-1))/(2*h);//central difference along z
            }
            cap_integ_sum+=(E_r_temp*E_r_temp+E_z_temp*E_z_temp)*i;
            //integral over E^2 times 2pi(r)dxdy
        }
        else{//E=0 on the electrodes and on the boundary
            E_r_temp= 0;
            E_z_temp = 0;
        }
        fp5<<E_r_temp<<setw(15);
        fp6<<E_z_temp<<setw(15);
        //specific plot at r=1.5mm
        // if(i==(int)z2/h){//since the spacing is the same, i==e3_zmin gives i=1.5mm
        //for that set of plots I want
        //     fp5<<E_r_temp<<setw(15);
        //     fp6<<E_z_temp<<setw(15);
        // }
    }
    fp5<<endl;
    fp6<<endl;
}
//moved common factors out of the loop, and multiply by .001 to account for the
//units of eps in m^-1
cap_integ_sum=cap_integ_sum*2*pi*h*h*h*eps*.001/(v2*v2);
//constant values scaling the integral correctly
cout<<"capacitance: "<<cap_integ_sum<<endl;
//----------------output phi-----------------------
//print the solution
for(i=0;i<rows;i++){//from r to rmax
    for(j=0;j<cols;j++){//from -zmax to zmax
        fp<<phi(i,j)<<setw(15);
        //specific plot at r=1.5mm
        // if(i==(int)z2/h){
        //     fp<<phi(i,j)<<setw(15);
        // }
    }
    fp<<endl;
    //cout<<endl;
}
//print the solution spacing in r
for(i=0;i<rows;i++){
    fp2<<i*h<<setw(15);
}
fp2<<endl;
//print the solution spacing in z
for(j=0;j<cols;j++){
    fp3<<j*h<<setw(15);
}
fp3<<endl;
```

```
        fp.close();
        fp2.close();
        fp3.close();
        fp4.close();
        fp5.close();
        return(EXIT_SUCCESS);
    } //end main
```