Solving the Poisson Equation with Multigrid

M. L. Smedinghoff
Fermi National Accelerator Laboratory P.O. Box 500, Batavia, IL, USA

September 2, 2005

Abstract

I give a short explanation of how to use multigrid to solve the Poisson equation in cylindrical coordinates for a solid conducting pipe. I also explain how to use a non-uniform grid to optimize the problem.

1 Introduction

The Fermi National Accelerator Laboratory is the home of a circular accelerator known as the Booster. While working at the lab, my colleagues and I became interested in modeling the interactions between particles while they are being accelerated in the Booster. In other words we wanted to find potential as a function of space. The relevant equations is as follows where $\rho(r, \phi, z)$ is the charge density function

$$\nabla^2 \psi(r, \phi, z) = -4\pi\rho(r, \phi, z). \tag{1}$$

We decided to do the problem in terms of cylindrical coordinates since our model of the Booster is a cylindrical conducting pipe (we also developed code for the Cartesian case, but the cylindrical case is more interesting, so that is what we will discuss in this text). We also decided that the best way to approach the solving of Equation 1 was to use multigrid computing methods. In other words, we wanted to reduce Equation 1 to something of the form A$\vec{x}$=$\vec{b}$ and then let the computer do the work. In order to begin this process, we must first examine finite differences.

2 Finite Differences

Consider the following one-dimensional equation:

$$u(x) = f(x) + f'(x) \tag{2}$$

In a finite difference scheme, we simply substitute f'(x) with an approximation, yielding

$$u_i = f_i + \frac{f_{i+1} - f_{i-1}}{2h}. \tag{3}$$

Higher order derivatives are treated the same way. For example, the approximation of the second derivative would be

$$\frac{d^2 f}{dx^2} \approx \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}. \tag{4}$$

Once we obtain an equation in terms of f_i and u_i, we can create the matrix equation that we desire. Since matrix equations are easily solved using multigrid, our work is done.

$$\frac{1}{2h}\begin{pmatrix} 2h & 1 & & & & \\ -1 & 2h & 1 & & & \\ & -1 & 2h & 1 & & \\ & & & -1 & 2h & 1 \\ & & & & -1 & 2h \end{pmatrix}\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \\ f_{n-1} \\ f_n \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \\ u_{n-1} \\ u_n \end{pmatrix} \tag{5}$$

3 Solving the Poisson equation

Now that we understand the finite difference method, we can use or new technique to solve the Poisson equation. First we must express our equation in terms or derivatives. Keep in mind that we are using cylindrical coordinates.

$$-4\pi\rho = \nabla^2\psi \tag{6}$$

$$= \frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial \psi}{\partial r}) + \frac{1}{r^2}\frac{\partial^2 \psi}{\partial \phi^2} + \frac{\partial^2 \psi}{\partial z^2} \tag{7}$$

$$= \frac{1}{r}\frac{\partial \psi}{\partial r} + \frac{\partial^2 \psi}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2 \psi}{\partial \phi^2} + \frac{\partial^2 \psi}{\partial z^2} \tag{8}$$

Now we can use our finite difference methods. Note that h_r, h_ϕ, and h_z represent the step sizes in the r, ϕ, and z directions respectively.

$$= \frac{\psi(r_{i+1}, \phi_j, z_k) - \psi(r_{i-1}, \phi_j, z_k)}{2rh_r} + \frac{\psi(r_{i+1}, \phi_j, z_k) - 2\psi(r_i, \phi_j, z_k) + \psi(r_{i-1}, \phi_j, z_k)}{h_r^2} +$$
$$\frac{\psi(r_i, \phi_{j+1}, z_k) - 2\psi(r_i, \phi_j, z_k) + \psi(r_i, \phi_{j-1}, z_k)}{r^2 h_\phi^2} + \frac{\psi(r_i, \phi_j, z_{k+1}) - 2\psi(r_i, \phi_j, z_k) + \psi(r_i, \phi_j, z_{k-1})}{h_z^2} \tag{9}$$

Simplifying leads to our final result.

$$= \frac{(h_r + 2r)\psi(r_{i+1}, \phi_j, z_k)}{2rh_r^2} + \frac{(-h_r + 2r)\psi(r_{i-1}, \phi_j, z_k)}{2rh_r^2} + \frac{\psi(r_i, \phi_{j+1}, z_k)}{r^2 h_\phi^2} +$$
$$\frac{\psi(r_i, \phi_{j-1}, z_k)}{r^2 h_\phi^2} + \frac{\psi(r_i, \phi_j, z_{k+1})}{h_z^2} + \frac{\psi(r_i, \phi_j, z_{k-1})}{h_z^2} + (\frac{-2}{h_r^2} - \frac{2}{r^2 h_\phi^2} - \frac{2}{h_z^2})\psi(r_i, \phi_j, z_k) \tag{10}$$

4 Boundary Conditions

Now that we have a formula that can easily be translated into a matrix, we need to consider boundary conditions. Obviously at r=0, there is a problem with the $\psi(r_{i-1},\phi_j,z_k)$ term. Should we simply ignore the $\psi(r_{i-1},\phi_j,z_k)$ term or should we replace it with something else? In our case, we must examine each direction separately.

First we will consider the $\hat{r}$ direction. In order to evaluate our equation, we will first need to divide the range of r into n different cells. Since r appears in the formula, we must find a way to calculate a value of r for each cell. We obviously cannot use the left side of each cell because we would have to evaluate our equation at r=0 which would result in dividing by zero. Instead, we will evaluate r at the middle of each cell. Now that we have a method for evaluating r, we can consider what should happen at the boundaries. At the inside boundary of r=0, we have no initial condition. Therefore $\psi(r_{i-1},\phi_j,z_k)$ will have no significance, and we can ignore the term. Now let us consider the outside boundary. In our model, the Booster is a conducting pipe, so we assume that the charge density is zero at the outside boundary. Hence, we can replace $\psi(r_{i+1},\phi_j,z_k)$ with zero as well when r reaches its maximum value.

Now we will consider the $\hat{\phi}$ direction. We do not have any boundary conditions for $\phi = 0$ or $\phi = 2\pi$, but we do know that ϕ should be periodic. In other words, $\psi(r_i,\phi_{j-1},z_k)$ evaluated in the leftmost cell should have the same value as $\psi(r_i,\phi_j,z_k)$ evaluated in the rightmost cell. This condition allows us to easily calculate terms that fall outside of the range of ϕ.

The $\hat{z}$ direction is similar to the $\hat{\phi}$ direction. There are no set boundary conditions, but our model dictates that z should be periodic. This condition takes care of all of the terms that fall outside the range of z.

Now that we have considered boundary conditions, we can create our matrix and solve the Poisson equation.

5 Example

Consider a beam of particles with uniform charge density of radius 0.2 in a beam pipe of radius 1. Since the charge distribution is uniform within the beam and the $\hat{\phi}$ and $\hat{z}$ direction have periodic boundary conditions, we can assume that the solution is independent of ϕ and z and can be treated as a one-dimensional problem. Now we have the equation

$$-4\pi\rho(r) = \nabla^2 f(r) = \frac{1}{r}\frac{d}{dr}(r\frac{df}{dr}). \tag{11}$$

This equation is simple enough that we can integrate it to obtain an exact solution.

$$f(r) = -\pi\rho(r)r^2 + a_0 ln(r) + a_1 \quad for\ 0 \leq r \leq .2$$

$$f(r) = a_2 ln(r) + a_3 \quad for\ 0.2 < r \leq 1$$

$$a_0 = 0$$
$$a_1 = 0.04\pi\rho(r)(-2ln(0.2) + 1)$$
$$a_2 = -0.08\pi\rho(r)$$
$$a_3 = 0$$

Now that we have an exact solution to our example, we can compare it to the multigrid solution. Figure 1 shows both curves. In this case, we used a grid size of 33 in each direction to generate the multigrid solution.

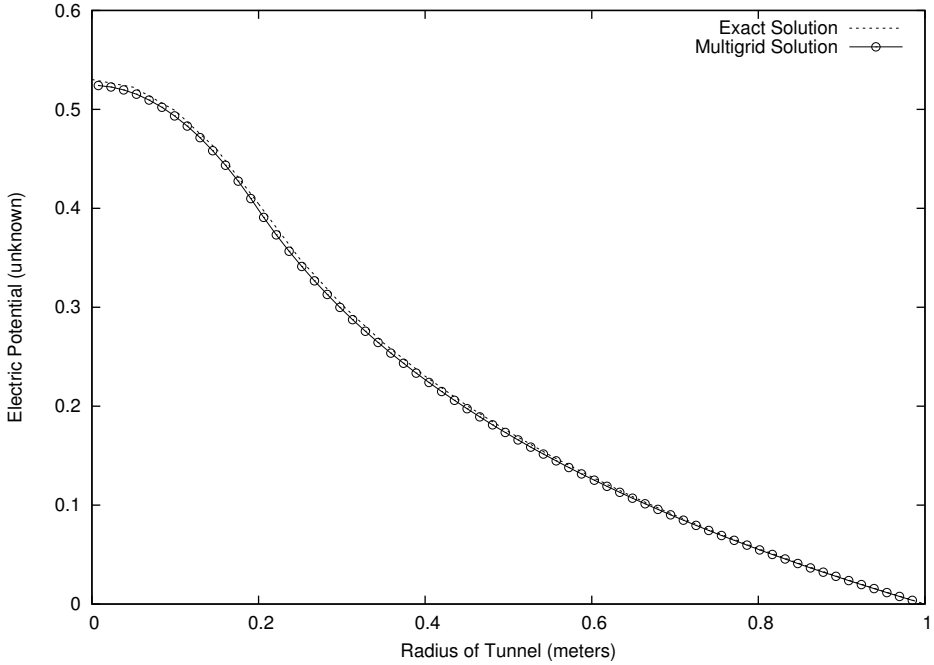

Figure 1: The exact solution and the multigrid solution to the Poisson Equation in which the charge density is a solid cylinder of radius 0.2

6 Non-Uniform Grid

In the previous example, we saw that our multigrid approach to solving the Poisson equation in cylindrical coordinates provides a close approximation to the exact solution given a fairly modest grid size. Is there any way we can optimize this method? In our case, there is. The beam inside of the Booster occupies the center of the beam pipe but not the entire space. Therefore, if we make our grid very fine on the part of the beam pipe where the beam is and

coarser in the other part, we can achieve a closer approximation to the solution since we can eliminate many zeros from our matrix.

The first step to implementing a non-uniform grid is to decide where the the grid should change from fine to coarse. In our case, the full width at half max seems to be optimal. After deciding the value of r where we want to switch, we must also find the cell number where we want to switch. In my experiments, I usually devoted 75 percent of my cells to the fine grid and the other 25 percent to the coarse grid. After making these two decisions, we load the particles into our $\rho(r, \phi, z)$ vector. To illustrate this process, let r_b be the cutoff value of r, r_{max} be the radius of the beam pipe, g_b be the cutoff cell of the grid, and g_{max} be the number of grid cells. To put a particle into the $\rho(r, \phi, z)$ vector, we first check to see if its r coordinate is less than r_b. If it is, we treat the problem as if there are only g_b cells that represent r's ranging from 0 to r_b and we put our new particles in the proper place. If our r coordinate is greater than r_b, we pretend there are only g_{max} - gc cells that range from r_b to r_{max}. We find the proper cell number for our new particle and add g_b to that number.

Now that we have our new $\rho(r, \phi, z)$ vector, we need to adjust our matrix. As in the case of the $\rho(r, \phi, z)$ vector, cells 1 through g_b will represent the fine grid and cells g_b+1 through g_{max} will represent the coarse grid. Calculating the coefficients of the matrix is easy. We simply need to replace the variable h with the h value from either the fine grid or the coarse grid depending on which cell we are in. The only complication arises at the cells directly to the left and right of r_b. Let h_f be the step size for the fine grid and h_c be the step size for the coarse grid. Since we calculate r at the center of each cell, the value to the left of r_b will be $h_f/2$ away from r_b, and the value to the right of r_b will be $h_c/2$ away from r_b. Therefor, the step size between these two cells will be neither h_f nor h_c, and these cases will have to be treated separately from the rest of the cells.

6.1 Calculations for the Cell to the Left of r_b

First let us consider the cell to the left of r_b. The $\psi(r_i, \phi_{j+1}, z_k)$, $\psi(r_i, \phi_{j-1}, z_k)$, $\psi(r_i, \phi_j, z_{k+1})$, and $\psi(r_i, \phi_j, z_{k-1})$ terms will all remain the same. We do need to recalculate all of the other terms, though. We can do this by recalculating the first and second partial derivatives in terms of r. Below are the recalculations of the derivatives and the new coefficients.

$$\frac{1}{r}\frac{\partial \psi}{\partial r} = \frac{\psi(r_{i+1}, \phi_j, z_k) - \psi(r_{i-1}, \phi_j, z_k)}{r(h_f + \frac{h_f + h_c}{2})} = \frac{2\psi(r_{i+1}, \phi_j, z_k) - 2\psi(r_{i-1}, \phi_j, z_k)}{r(3h_f + h_c)}$$

$$\frac{\partial^2 \psi}{\partial r^2} = \frac{\frac{\partial \psi(r_{i+.5}, \phi_j, z_k)}{\partial r} - \frac{\partial \psi(r_{i-.5}, \phi_j, z_k)}{\partial r}}{3h_f + h_c}$$

$$= \frac{8h_f\psi(r_{i+1}, \phi_j, z_k) - 8h_f\psi(r_i, \phi_j, z_k) - 4(h_f + h_c)\psi(r_i, \phi_j, z_k) + 4(h_f + h_c)\psi(r_{i-1}, \phi_j, z_k)}{h_f(h_f + h_c)(3h_f + h_c)}$$

$$\psi(r_{i+1}, \phi_j, z_k) \text{ coefficient} = \frac{8rh_f + 2h_f(h_f + h_c)}{rh_f(h_f + h_c)(3h_f + h_c)}$$

$$\psi(r_{i-1},\phi_j,z_k) \text{ coefficient} = \frac{4rh_f + 4rh_c - 2h_f(h_f+h_c)}{rh_f(h_f+h_c)(3h_f+h_c)}$$

$$\psi(r_i,\phi_j,z_k) \text{ coefficient} = \frac{-12h_f - 4h_c}{h_f(h_f+h_c)(3h_f+h_c)} + \frac{1}{r^2h_\phi^2} + \frac{1}{h_z^2}$$

6.2 Calculations for the Cell to the Right of r_b

The calculations for the cell to the right of r_b are very similar to those of the cell to the left of r_b.

$$\psi(r_{i+1},\phi_j,z_k) \text{ coefficient} = \frac{4rh_f + 4rh_c + 2h_c(h_f+h_c)}{rh_c(h_f+h_c)(h_f+3h_c)}$$

$$\psi(r_{i-1},\phi_j,z_k) \text{ coefficient} = \frac{8rh_c - 2h_c(h_f+h_c)}{rh_c(h_f+h_c)(h_f+3h_c)}$$

$$\psi(r_i,\phi_j,z_k) \text{ coefficient} = \frac{-4h_f - 12h_c}{h_c(h_f+h_c)(h_f+3h_c)} + \frac{1}{r^2h_\phi^2} + \frac{1}{h_z^2}$$

7 Example of a Non-Uniform Grid

The first step towards implementing a non-uniform grid is deciding where the grid will change from fine to coarse. In our case, we determined the cutoff point experimentally. We ran the program with several different cutoff points and compared the solution to a uniform solution on a much larger grid (the solution on the uniform grid was treated as the exact solution). We determined the error by the following formula:

$$error = \sqrt{\sum_i \frac{(x_i - x_i^{exact})^2}{(x_i^{exact})^2}}$$

Table 1 shows the data we collected. We decided to measure the cutoff in terms of r/σ in order to be better able to gage the cutoff point for future problems. Figure 2 shows a graph of that data as well as a line representing the error calculated from a uniform grid of the same size. Our hope was that the non-uniform grid would improve the accuracy of our solution, and Figure 2 clearly shows that we accomplished this goal.

Figure 3 shows a plot of a non-uniform case compared to the very fine (exact) uniform solution. In this case, the cutoff point is $r = .008$, which is just under one standard deviation above the mean. The non-uniform solution lies almost on top of the uniform solution, even though it was calculated on a much coarser grid. Again we find that a non-uniform grid can improve accuracy.

Cutoff Point (r/σ)	Error	Time to Run (seconds)
2.364066194	0.00055121	6.868
2.955082742	0.00018577	6.022
3.546099291	0.00013550	6.284
4.137115839	0.00011496	6.156
4.432624113	0.00011026	6.226
4.728132388	0.00010888	6.201
5.023640662	0.00011093	6.033
5.082742317	0.00011183	6.116
5.141843972	0.00011289	5.957
5.200945626	0.00011409	6.069
5.260047281	0.00011548	6.103
5.319148936	0.00011697	6.092
5.378250591	0.00011867	6.067
5.437352246	0.00012055	6.087
5.496453901	0.00012254	6.036
5.555555556	0.00012468	6.128
5.61465721	0.00012696	6.037
5.910165485	0.00014099	6.174
6.205673759	0.00015910	5.990
6.501182033	0.00018157	6.040
7.092198582	0.00023944	5.912
7.68321513	0.00031596	5.962
8.274231678	0.00041242	5.866
8.865248227	0.00053004	6.138

Table 1: The error calculated as described in Section 7 for several different cutoffs between the fine section and coarse section in a non-uniform grid.

8 A Few Words about PETSc

In order to implement our finite difference scheme to solve the Poisson Equation, we needed data structures that implemented multigrid. We chose to use the Portable, Extensible Toolkit for Scientific Computing (PETSc) created by Argonne National Laboratory. PETSc is a set of data structures that allow for parallel processing of solutions to partial differential equations. More information can be found at http://www-unix.mcs.anl.gov/petsc/petsc-as/index.html.

9 Conclusion

One effective way of solving the Poisson equation in cylindrical coordinates is to convert the differential equation into a matrix equation and solve it using multigrid. As the grid size increases, the multigrid solution will approach the

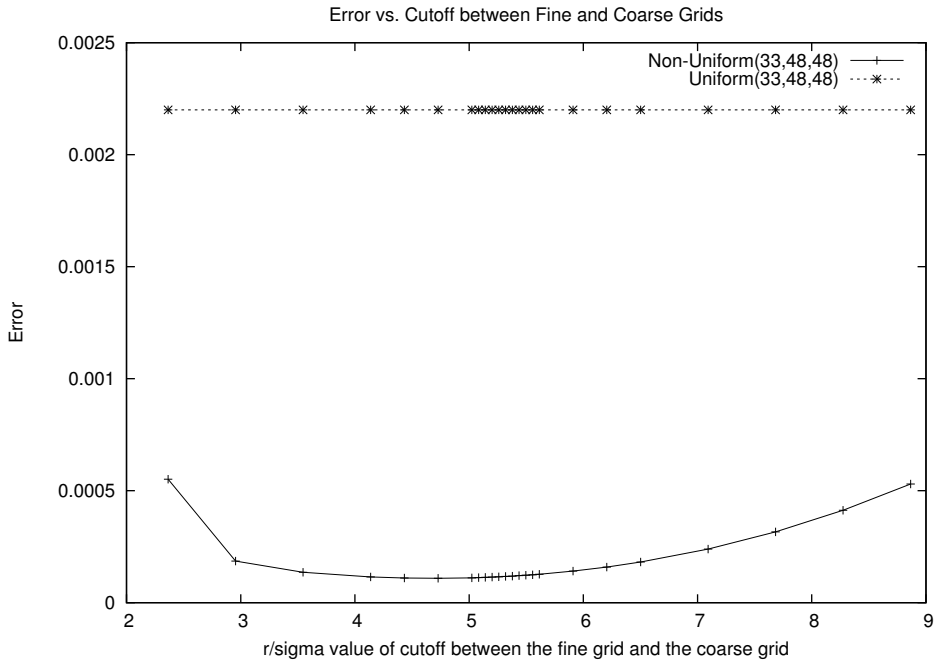

Figure 2: The lower of the two curves represents the error measured for non-uniform grids with different cutoffs between the fine portion and the coarse portion. The upper curve represents the error measured for a uniform grid of the same size.

exact solution very quickly. One way to optimize the multigrid approach is to create a non-uniform grid in which the important parts of the problem are solved on a fine grid while the less important parts are solved on a coarser grid. This optimization often results in closer approximations without sacrificing time.

References

[B] W. L. Briggs. *A Multigrid Tutorial*, Center for Applied Scientific Computing, Lawrence Libermore National Laboratory, http://www.llnl.gov/casc/people/henson/mgtut/ps/mgtut.pdf

[L] M. Lai. *A note on finite difference discretizations for Poisson equation on a disk*, Chung cheng University, Minghsiung, Chiayi 621, Taiwan, ?.

[P] S. Balay, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. Smith, and H. Zhang. *PETSc Users Manual*, Argonne National Laboratory, Argonne, IL, 2005.

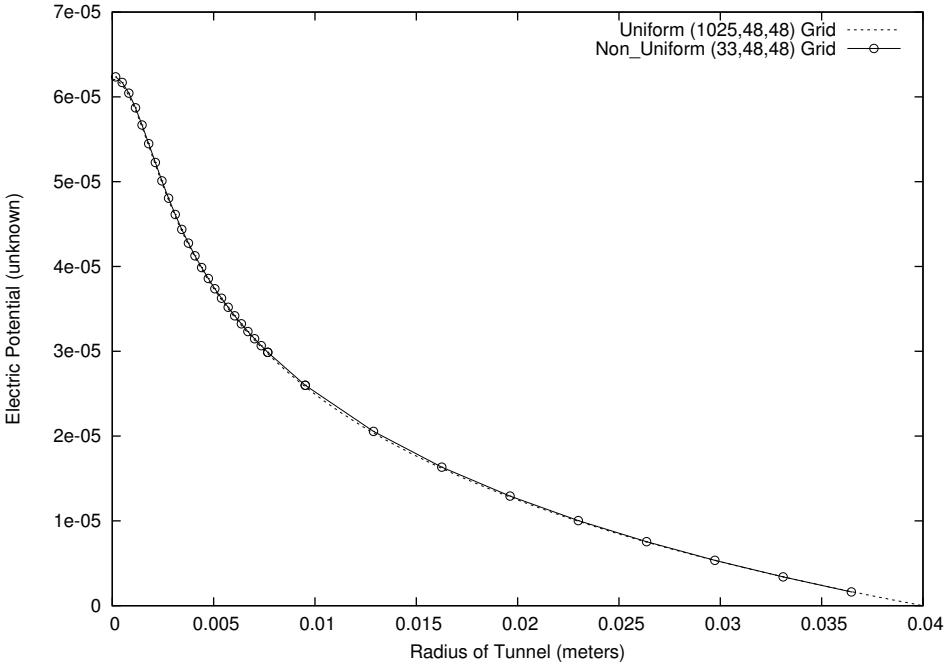

Figure 3: A uniform solution on a (1025,48,48) grid compared with a non-uniform solution on a (33,48,48) grid.

APPENDIX A

The following is a summary of all of the files that were used in this project. All files are located at /home/cepa01/msmeding/strawman/. There are also additional copies of many of these files located on lqcdib at /home/msmeding/strawman/. Many of the lqcdib files have been modified slightly to run on the cluster. A copy of this paper is located at /home/cepa01/msmeding/latex/.

C++ Files

• strawman1.cc - strawman1.cc is designed to solve the Poisson Equation using PETSc libraries. To solve the Poisson Equation, you need a right hand side and a Jacobian. strawman1.cc has several options for the right hand side and the Jacobian. The two options for the Jacobian are a Cartesian version and a cylindrical version depending on which coordinates system you are using. There are several different options for the right hand side. The options for a Cartesian coordinate system are a solid sphere of randomly generated particles, a solid cylinder of randomly generated particles, and a solid Gaussian cylinder of randomly generated particles. The three options for a cylindrical coordinate

system are a solid cylinder, a cylinder with a Gaussian in the r direction, and a right hand side generated from the particles in a .h5 file.
The program is called as follows:
strawman1 < $sphere|cylinder|gaussian|cylinder\_c|gaussian\_c|h5file$ >. There are several other optional arguments that I've found useful. They include -dmmg_nlevels n which lets you set the number of levels of refinement in the multigrid process. Some others are -pc_mg_type kaskade, -pc_mg_galerkin, -pc_type mg, and -ksp_converged_reason. One other variable that you might want to set is the grid size. This variable must be set manually by editing the main function of the program.
strawman1.cc automatically produces files called rho.dat and soln.dat which are Octave representations of the right hand side and solution respectively. In addition, there is a function called VecView_vtk which produces structuredPoints vtk files that can be plotted using plotVtk.py. There is another function called getField which produces a vtk file containing the electric field. There are also cylindrical versions of both of these routines.
• num.cc - num.cc is very similar to strawman1.cc, the main difference being that num.cc solves the Poisson Equation on a non-uniform grid. In other words, the important part of the equation is solved on a fine grid while the less important part is solved on a coarser grid. The cutoff between the fine grid and the coarse grid must be set manually in the main function. Note also that the routines that produce vtk files have not been tested for num.cc and probably don't work correctly.

Data Files
• field.dat - A file generated by strawman1.cc and num.cc. This file contains the electric field of the solution to the Poisson Equation. There is code to generate field.dat in both Cartesian and cylindrical coordinates, but the cylindrical code is untested. This file is intended to be plotted by plotField.py to produce a 3d VTK plot.
• rho.dat - A file generated by strawman1.cc and num.cc. This file contains the right hand side of the Poisson Equation. rho.dat is generated for both Cartesian and cylindrical coordinates and has been thoroughly tested in both coordinate systems. This file is intended to be plotted in Octave.
• rho.vtk - A file generated by strawman1.cc and num.cc. This file also contains the right hand side of the Poisson Equation. rho.vtk is a structuredPoints vtk file and is intended to be used with plotVtk.py to create a 3d volume rendering of the right hand side of the Poisson Equation. Note that the cylindrical version of this file produces lower quality plots that the Cartesian version.
• soln.dat - A file generated by strawman1.cc and num.cc. This file is the solution to Poisson's Equation. It is similar to rho.dat in that it has been thoroughly tested for both Cartesian and cylindrical coordinates. soln.dat is intended to be plotted in Octave.
• soln.vtk - A file generated by strawman1.cc and num.cc. This file is also the

solution to the Poisson Equation. Similar to rho.vtk, soln.vtk is a structured-Points vtk file that is intended to be plotted using plotVtk.py to create a 3d volume rendering of the solution. Again, note that the cylindrical version of this file produces lower quality plots that the Cartesian version.

Gnuplot Files

• timings/num.gp - This file produces the plot seen in Figure 2. The data file that this script uses for input is timings/num.data. num.gp is currently set to produce a postscript plot, but a png plot can be produced instead by uncommenting the appropriate lines.

Octave Files

• anger.m - This script plots the uniform solution stored in soln.bak and three non-uniform solutions stored in soln1.dat, soln2.dat, and soln3.dat. For anger.m to work properly, you must manually set the radius of the beam pipe (max) and the r-values where your non-uniform solutions go from fine to coarse (brk1, brk2, brk3).

• index.m - index.m is a function that takes three inputs, an r-index, a ϕ-index, and a z-index. index.m returns the corresponding index in a one-dimensional array.

• lai.m - This script plots soln.dat, the solution produced when strawman1.cc is run in cylinder_c mode. lai.m also plots the exact solution to the Poisson Equation when the input is a rigid cylinder. There are two arguments to lai.m. They are the grid size in r and the grid size in ϕ respectively. If you uncomment the last couple of lines, you can get a postscript version of the plot.

• ndx.m - ndx.m performs the same function as index.m but for the two-dimensional case where there is only an r-index and a ϕ-index.

• num.m - This script takes an integer argument. An argument of 0 plots the right hand side of a uniform grid stored in rho.bak as well as the non-uniform right hand side stored in rho.dat. An argument of 1 does the same as 0 except that it plots the solutions in soln.bak and soln.dat instead of the right hand sides. An argument of 2 plots the solutions to two uniform grids. The finer grid should be stored is soln.bak and the coarser should be in soln.dat. An argument of 3 produces the same as 1 except that it adds a second non-uniform grid stored in soln.data.

Note that you have to manually set the cutoffs between coarse and fine grids in brk. Note also that num.m calculates the error function discussed in Section 7 for all arguments with the exception of 3. You can also produce a postscript version of the plot by uncommenting the last couple lines in the file.

Python Files

• plotField.py - plotField.py is intended to plot the electric field of the solution

to the Poisson Equation using vtk. This script has one mandatory argument and two optional arguments. The required argument is the name of the file that contains the field you want to plot (probably field.dat produced by strawman1.cc). The first optional argument is a file containing the solution to the Poisson Equation (probably soln.vtk produced by strawman1.cc). The other optional argument is a png file where you want the plot to be stored.

• plotVtk.py - plotVtk.py is intended to produce a 3-dimensional volume rendering of a structuredPoints vtk file. This script has one optional and one required argument. The required argument is the vtk structuredPoints file (probably rho.vtk or soln.vtk produced by strawman1.cc). The optional argument is the png file where you want the plot to be stored.