ECSE 543 Assignment 2 Report

Code and report authored by Jake Peterson for the Fall 2022 semester of ECSE 543.

Report Structure

This report is divided into sections by questions. The subdivisions somewhat loosely correspond to the subparts of the questions and exceptions will be noted.

Problem Description

We are tasked with analyzing and simulating several parts of a conductor coax consisting of an outer conductor with square cross-section and side length 0.2 meters held at 0 V, a centred inner conductor with rectangular cross-section 0.08 by 0.06 meters held at 110 V, and an internal medium with free space permittivity (ϵ_0). The below is a cross-section diagram of this construction:

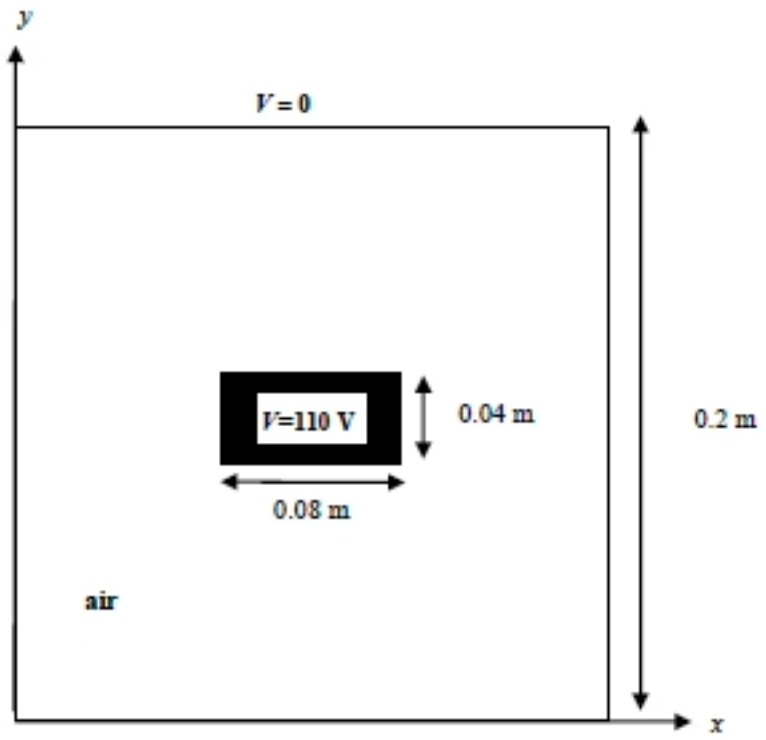
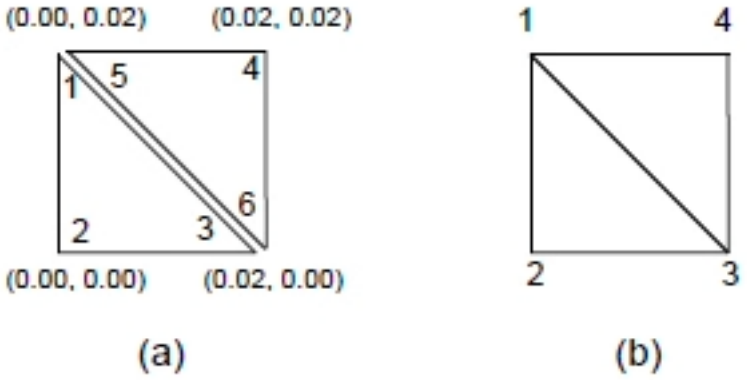


Figure 2.

Q1: Conjoint Triangular Finite Element S-Matrices

Triangular finite elements allow numerical simulations of more esoteric geometries outside of the square meshes we investigated in the last assignment. However, constructing methods to utilize them is nontrivial, often requiring that measures gained at triangular vertices is correctly paired to recognize that different matrices of different triangles may correspond to similar physical locations. We aim to analyze and form a method for using a grid of paired triangular finite elements with the following structure and node numbering:



By convention, these triangles have vertices that are numbered counterclockwise. We shall refer to this numbering scheme throughout this document where necessary, with deviations documented.

In particular, we seek to apply these elements to the conductor problem stated previously, to account for the electrical energy stored in the configuration at electrostatic potential.

$$W = \frac{1}{2}\int_\Omega \underline{E}\cdot\underline{D}\,\mathrm{d}S = \frac{1}{2}\int_\Omega (-\nabla u)\cdot(-\varepsilon_0\nabla u)\,\mathrm{d}S = \frac{1}{2}\int_\Omega \varepsilon_0|\nabla u|^2\,\mathrm{d}S$$

where u is the electrostatic potential. As we do not start off with knowing u, we need to approximate it. The easiest way to do this in an appropriate manner is to make a linear approximation:

$$U(x,y) = a + bx + cy$$

where $a, b, c \in \mathbb{R}$. We further define three quantities U_1, U_2, U_3 which are the electrostatic potential evaluated at triangle vertices P_1, P_2, P_3 where $P_i = (x_i, y_i)$. We further define:

$$\underline{U} = [U_1, U_2, U_3]^T$$

where X^T refers to the real-valued transpose of X. We can construct a matrix system of equations

from this vector and the motivating approximation equation:

$$\underline{U} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

This system is most easily solved using Cramer's rule:

$$a = \frac{\begin{vmatrix} U_1 & x_1 & y_1 \\ U_2 & x_2 & y_2 \\ U_3 & x_3 & y_3 \end{vmatrix}}{\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}} = \frac{U_1(x_2y_y - x_3y_2) + x_1(U_3y_2 - U_2y_3) + y_1(U_2x_3 - U_3x_2)}{2A}$$

(**Context**: the denominator determinant is equal to twice the area of the triangle, which is denoted as A.)

$$b = \frac{\begin{vmatrix} 1 & U_1 & y_1 \\ 1 & U_2 & y_2 \\ 1 & U_3 & y_3 \end{vmatrix}}{2A} = \frac{(U_2y_3 - U_3y_2) + U_1(y_2 - y_3) + y_1(U_3 - U_2)}{2A}$$

$$c = \frac{\begin{vmatrix} 1 & x_1 & U_1 \\ 1 & x_2 & U_2 \\ 1 & x_3 & U_3 \end{vmatrix}}{2A} = \frac{(U_3x_2 - U_2x_3) + x_1(U_2 - U_3) + U_1(x_3 - x_2)}{2A}$$

We can substitute these equations back into our linear interpolation to arrive at a sum in terms of U_i:

$$U = \sum_{i=1}^{3} U_i \alpha_i(x, y)$$

Careful rearrangement of the constituent variables inside (a, b, c) reveal α_i:

$$\alpha_1(x, y) = \frac{1}{2A}\left[(x_2y_3 - x_3y_2) + (y_2 - y_3)x + (x_3 - x_2)y\right]$$
$$\alpha_2(x, y) = \frac{1}{2A}\left[(x_3y_1 - x_1y_3) + (y_3 - y_1)x + (x_1 - x_3)y\right]$$

$$\alpha_3(x,y) = \frac{1}{2A}\left[(x_1y_2 - x_2y_1) + (y_1 - y_2)x + (x_2 - x_1)y\right]$$

We can now turn our attention back to our energy equation. (**Remark**: ϵ is not written in the rest of this workup, as it is not necessary for minimization. It should not be ignored should this technique be used to calculate numerical quantities).

$$W = \frac{1}{2}\int_\Omega \varepsilon_0 |\nabla u|^2 \,\mathrm{d}S = \frac{1}{2}\sum_{i=1}^{3}\sum_{j=1}^{3} U_i U_j \int_{\Delta e} \nabla\alpha_i \nabla\alpha_j \,\mathrm{d}S = \frac{1}{2}\underline{U}^T \underline{\underline{S}}^{(e)} \underline{U}$$

where

$$S_{i,j} = \int_{\Delta e} \nabla\alpha_i \nabla\alpha_j \,\mathrm{d}S$$

It is useful to observe that $\int_{\Delta e} \mathrm{d}S = A$, particularly because $\nabla\alpha_i$ is constant in x and y, which makes constructing $\underline{\underline{S}}$ very easy given knowledge of $\nabla\alpha_i$. Furthermore, owing to the commutative property, $\underline{\underline{S}}$ is symmetric, which also reduces effort.

$$\nabla\alpha_1 = \frac{(y_2 - y_3)\hat{x} + (x_3 - x_2)\hat{y}}{2A}$$
$$\nabla\alpha_2 = \frac{(y_3 - y_1)\hat{x} + (x_1 - x_3)\hat{y}}{2A}$$
$$\nabla\alpha_3 = \frac{(y_1 - y_2)\hat{x} + (x_2 - x_1)\hat{y}}{2A}$$

Plugging these in:

$$\underline{\underline{S}} = \begin{bmatrix} S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} S_{1,1} & S_{1,2} & S_{1,3} \\ S_{1,2} & S_{2,2} & S_{2,3} \\ S_{1,3} & S_{2,3} & S_{3,3} \end{bmatrix}$$

$$S_{1,1} = \frac{(y_2 - y_3)^2 + (x_3 - x_2)^2}{4A}$$
$$S_{1,2} = \frac{(y_3 - y_1)(y_2 - y_3) + (x_1 - x_3)(x_3 - x_2)}{4A}$$
$$S_{1,3} = \frac{(y_1 - y_2)(y_2 - y_3) + (x_2 - x_1)(x_3 - x_2)}{4A}$$
$$S_{2,2} = \frac{(y_3 - y_1)^2 + (x_1 - x_3)^2}{4A}$$

$$S_{2,3} = \frac{(y_1 - y_2)(y_3 - y_1) + (x_2 - x_1)(x_1 - x_3)}{4A}$$

$$S_{3,3} = \frac{(y_1 - y_2)^2 + (x_2 - x_1)^2}{4A}$$

Having found the S-matrix for an arbitrary triangle, we must now find the conjoint S-matrix for the two triangles shown above. We start at the disjoint case, where we consider $\underline{U}_{\text{dis}} = [U_1\ U_2\ U_3\ U_4\ U_5\ U_6]^T$ at the previously defined vertices:

$$\underline{\underline{S}}_{\text{dis}} = \begin{bmatrix} S_{1,1} & S_{1,2} & S_{1,3} & 0 & 0 & 0 \\ S_{1,2} & S_{2,2} & S_{2,3} & 0 & 0 & 0 \\ S_{1,3} & S_{2,3} & S_{3,3} & 0 & 0 & 0 \\ 0 & 0 & 0 & S_{4,4} & S_{4,5} & S_{4,6} \\ 0 & 0 & 0 & S_{4,5} & S_{5,5} & S_{5,6} \\ 0 & 0 & 0 & S_{4,6} & S_{5,6} & S_{6,6} \end{bmatrix}$$

Observing that vertices (1, 5) and (3, 6) are physically equivalent, we can transform $\underline{\underline{S}}_{\text{dis}} \in \mathbb{R}^{6\times 6}$ into $\underline{\underline{S}}_{\text{conj}} \in \mathbb{R}^{4\times 4}$ by defining a connectivity matrix $\underline{\underline{C}} \in \mathbb{R}^{6\times 4}$:

$$\underline{\underline{S}}_{\text{conj}} = \underline{\underline{C}}^T \underline{\underline{S}}_{\text{dis}} \underline{\underline{C}}$$

where

$$\underline{\underline{C}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\therefore \underline{\underline{S}}_{\text{conj}} = \begin{bmatrix} (S_{1,1} + S_{5,5}) & S_{1,2} & (S_{1,3} + S_{5,6}) & S_{4,5} \\ S_{1,2} & S_{2,2} & S_{2,3} & 0 \\ (S_{1,3} + S_{5,6}) & S_{2,3} & (S_{3,3} + S_{6,6}) & S_{4,6} \\ S_{4,5} & 0 & S_{4,6} & S_{4,4} \end{bmatrix}$$

We can write the elements of this matrix in terms of the conjoint node indexing (recognizing that it is still symmetric to reduce our workload):

$$S_{\text{conj},1,1} = \frac{(y_2 - y_3)^2 + (x_3 - x_2)^2 + (y_3 - y_4)^2 + (x_4 - x_3)^2}{4A}$$

$$S_{\text{conj},1,2} = \frac{(y_3 - y_1)(y_2 - y_3) + (x_1 - x_3)(x_3 - x_2)}{4A}$$

$$S_{\text{conj},1,3} = \frac{(y_1 - y_2)(y_2 - y_3) + (x_2 - x_1)(x_3 - x_2) + (y_4 - y_1)(y_3 - y_4) + (x_1 - x_4)(x_4 - x_3)}{4A}$$

$$S_{\text{conj},1,4} = \frac{(y_3 - y_4)(y_1 - y_3) + (x_4 - x_3)(x_3 - x_1)}{4A}$$

$$S_{\text{conj},2,2} = \frac{(y_3 - y_1)^2 + (x_1 - x_3)^2}{4A}$$

$$S_{\text{conj},2,3} = \frac{(y_1 - y_2)(y_3 - y_1) + (x_2 - x_1)(x_1 - x_3)}{4A}$$

$$S_{\text{conj},2,4} = 0$$

$$S_{\text{conj},3,3} = \frac{(y_1 - y_2)^2 + (x_2 - x_1)^2 + (y_4 - y_1)^2 + (x_1 - x_4)^2}{4A}$$

$$S_{\text{conj},3,4} = \frac{(y_4 - y_1)(y_1 - y_3) + (x_1 - x_4)(x_3 - x_1)}{4A}$$

$$S_{\text{conj},4,4} = \frac{(y_1 - y_3)^2 + (x_3 - x_1)^2}{4A}$$

Given that our conjoint coordinates form a square, and $A = \frac{1}{2}h^2$, our S-matrix reduces very nicely to numeric terms:

$$\underline{\underline{S}}_{\text{conj}} = \begin{bmatrix} 1 & -0.5 & 0 & -0.5 \\ -0.5 & 1 & -0.5 & 0 \\ 0 & -0.5 & 1 & -0.5 \\ -0.5 & 0 & -0.5 & 1 \end{bmatrix}$$

Q2: Two-Element Meshes and `SIMPLE2D`

Returning our attention back to our conductor problem, we are tasked with building and using a mesh constructed from tiling our two-element setup from Q1.

2A: Constructing the mesh and `SIMPLE2D` input

It is very possible with step-size $h = 0.02$ to construct such a triangular matrix by hand, perhaps with the aid of spreadsheet software. I chose instead do so programmatically, borrowing code from Assignment 1, Q3 to construct the initial rectangular matrix on which the triangular mesh lattice is imposed.

- The code responsible for calculating this initial matrix is the code used for generating an SOR simulation matrix. This is somewhat inefficient (on a theoretical basis, with no perceivable penalty in wall time), but allows for a couple of clever optimizations when extracting necesary information for creating the input files for `SIMPLE2D`.
- Dirichlet boundaries are calculated independently for each conductor by finding nodes that are held 'fixed' at a voltage (determined by setting them as strictly integers within the code), and finding nodes under that condition whose neighbors are not held fixed. Nodes without such neighbors thus must be "inside" the conductor and are 'punched out' by setting them to the `NaN` floating-point constant. Nodes with such neighbors are on the dirichlet boundary, the detection of which is useful as explained later on.

Once we have our initial rectangles matrix, we must find three things, all of which are needed as input tables for `SIMPLE2D`:

1. A list of all free nodes, which are not part of a conductor/dirichlet boundary. This is given as a table of entries, with columns corresponding to node index and position (x, y).
2. A list of triangles whose vertices are gathered from those free nodes. The formerly-introduced constraints of counter-clockwise indexing apply here, and the scheme with which these nodes are mapped to vertices within a two-element mesh matches that present in Figure 1. This is given as a table of entries, with columns corresponding to three node indices (our triangle vertices, in correct order)) and a current density, which is 0 given no current flow specified.
3. A list of all nodes lying on dirichlet boundaries. This is given as a table of entries, with columns corresponding to node index and fixed voltage level.

The output generated from this scheme is presented in raw form below:

```
1   0.000   0.000
2   0.000   0.020
3   0.000   0.040
4   0.000   0.060
5   0.000   0.080
6   0.000   0.100
7   0.020   0.000
8   0.020   0.020
9   0.020   0.040
10  0.020   0.060
11  0.020   0.080
12  0.020   0.100
13  0.040   0.000
```

```
14  0.040   0.020
15  0.040   0.040
16  0.040   0.060
17  0.040   0.080
18  0.040   0.100
19  0.060   0.000
20  0.060   0.020
21  0.060   0.040
22  0.060   0.060
23  0.060   0.080
24  0.060   0.100
25  0.080   0.000
26  0.080   0.020
27  0.080   0.040
28  0.080   0.060
29  0.080   0.080
30  0.100   0.000
31  0.100   0.020
32  0.100   0.040
33  0.100   0.060
34  0.100   0.080

2  1  7     0.000
8  2  7     0.000
3  2  8     0.000
9  3  8     0.000
4  3  9     0.000
10 4  9     0.000
5  4  10    0.000
11 5  10    0.000
6  5  11    0.000
12 6  11    0.000
8  7  13    0.000
14 8  13    0.000
9  8  14    0.000
15 9  14    0.000
10 9  15    0.000
16 10 15    0.000
```

```
11 10 16    0.000
17 11 16    0.000
12 11 17    0.000
18 12 17    0.000
14 13 19    0.000
20 14 19    0.000
15 14 20    0.000
21 15 20    0.000
16 15 21    0.000
22 16 21    0.000
17 16 22    0.000
23 17 22    0.000
18 17 23    0.000
24 18 23    0.000
20 19 25    0.000
26 20 25    0.000
21 20 26    0.000
27 21 26    0.000
22 21 27    0.000
28 22 27    0.000
23 22 28    0.000
29 23 28    0.000
26 25 30    0.000
31 26 30    0.000
27 26 31    0.000
32 27 31    0.000
28 27 32    0.000
33 28 32    0.000
29 28 33    0.000
34 29 33    0.000

1   0.000
2   0.000
3   0.000
4   0.000
5   0.000
6   0.000
7   0.000
```

```
13  0.000
19  0.000
25  0.000
30  0.000
23  110.000
24  110.000
29  110.000
34  110.000
```

Remark: There are 46 triangles specified in this output.

2B: SIMPLE2D solution.

Providing the above data to SIMPLE2D as input produces the following electrostatic mesh solution (grabbed perhaps unceremoniously from Microsoft Excel):

	0	0.02	0.04	0.06	0.08	0.1
0	0	0	0	0	0	0
0	0	7.018554	13.65193	19.11068	22.26431	23.25687
0	0	14.42229	28.47848	40.5265	46.68967	48.49886
0	0	22.19212	45.31319	67.82718	75.46902	77.35922
0	0	29.03301	62.75498	110	110	110
0	0	31.18494	66.67372	110		

Remark: The above representation of the SIMPLE2D output was rearranged by hand. SIMPLE2D will output this information as a column attached to the table of nodes by physical location.

The node corresponding to (0.06, 0.04) has a potential of 40.5265 V.

2C: Calculating Capacitance per Unit Length

Calculating Capacitance per Unit Length is possible knowing the cross-section Charge and Voltage of a capacitor.

$$W = \frac{1}{2}CV^2 \implies C = \frac{2W}{V^2}$$

We can calculate the charge by finding a sum over the electrostatic potential grid. The charge of every square in the grid can be found from the groundwork we laid in addressing the first question:

$$W = \frac{1}{2}\underline{U}^T \underline{\underline{S}}^{(e)} \underline{U}$$

This equation becomes useful to us by considering the potentials at the corners of the squares (numbered counterclockwise, starting at the upper right corner) and the conjoint S-Matrix. Once we do this, the total cross-section energy is equal to the sum of all square charges.

$$W = \sum_{\square \in \Omega} W^{(\square)}$$

Remarks:

1. Our workup did not include ϵ, so we should divide the above sum by our permittivity constant (in this case, ϵ_0) to get the total charge.
2. We have only run this summation over a quarter of the conductor, so we should multiply this result by four to get the charge for the whole conductor.

Doing this work with a voltage of 110V, we find a total charge of 0.61 µJ and a unit length capacitance of 0.1 nF/m.

Q3: Conjugate Gradient Method

We are now tasked with writing an implementation of the Conjugate Gradient Method to apply to our conductor simulation problem.

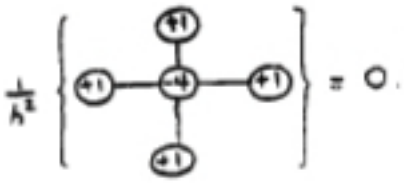
3A: Test the matrix

The first step is to make sure we can use the Conjugate Gradient Method on our system. We construct a matrix by finding the free nodes in our system, indexing their unknown potentials $\phi_{i,j}$ into a vector $\underline{\phi}_c$, and constructing a matrix based on their occurence within a a system

$$\underline{\underline{A}}\,\underline{\phi}_c = \underline{b}$$

where each consituent equation in this system is the first order finite difference approximation to

$$\nabla^2 \phi|_{i,j} = 0$$



When considering simple simulations with no Neumann boundaries, this produces diagonally dominant symmetric matrices that, when multiplied by -1, are positive-definite. Unfortunately, with the introduction of free nodes on Neumann boundaries (such as where axes of symmetry are taken advantage of to reduce matrix sizes), the matrices produced are not symmetric, but multiplying them by their transpose produces positive definite matrices.

Both the Cholesky and non-Cholesky `is_positive_definite` methods were used to analyze these matrices, producing identical results.

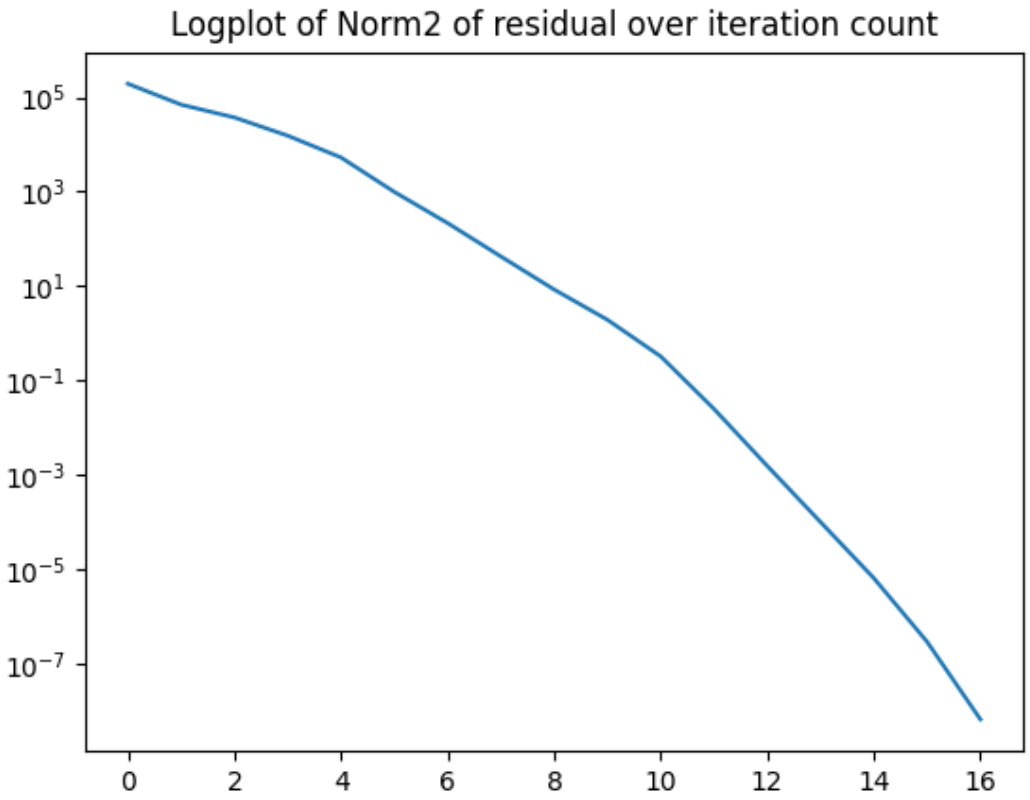
An alternative implementation that simulates the full conductor was also written as a sanity check.

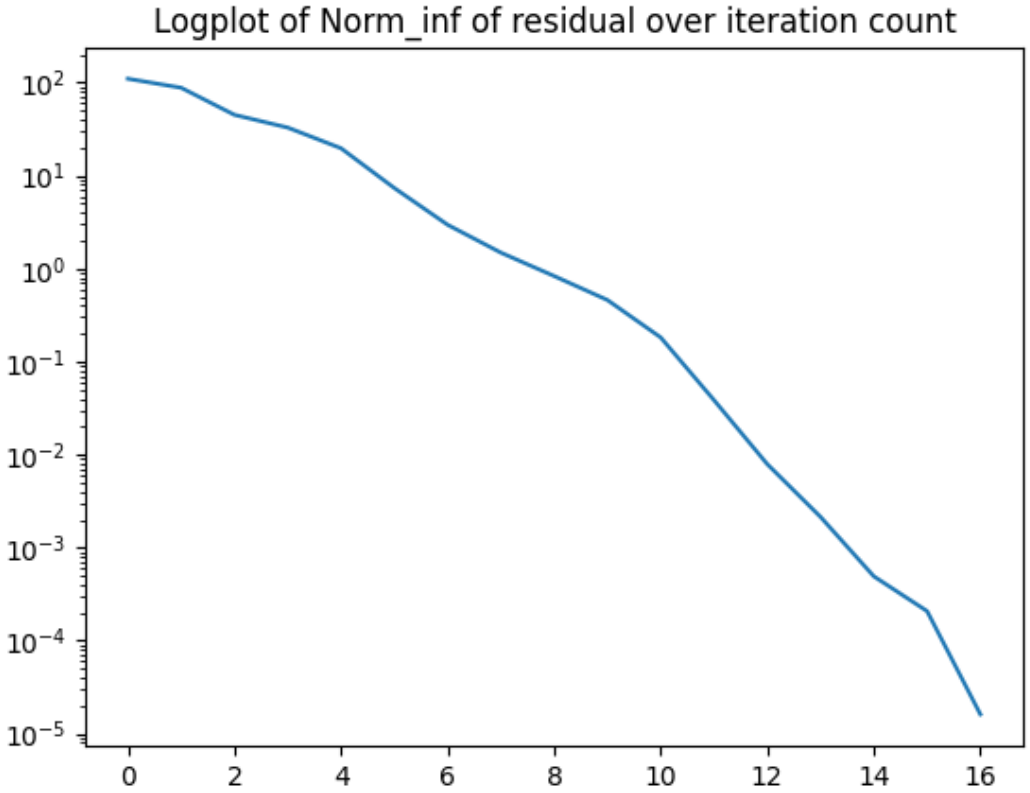
3B: Cholesky and Conjugate Gradient Method solutions

> This section addresses Question 3b), 3c), and 3d). At the time of writing, the main implementation of 3A did not correctly account for Neumann boundaries, so the below was written based on the full conductor alternate implementation.

Both the Cholesky and CGM methods give equivalent results (barring some floating-point variance, accurate to > 6 significant figures for threshold == 10e-8 for CGM).

The reported Voltages at (0.06, 0.04) for both Conjugate Gradient and Cholesky are 40.5265 V. For context, the SOR voltage was 42.2654 V.


Logplot of Norm2 of residual over iteration count
10^{5}
10^{3}
10^{1}
10^{-1}
10^{-3}
10^{-5}
10^{-7}
0
2
4
6
8
10
12
14
16



3C: Suggestions for Calculating Capacitance over Length

The output from the CGM matches the `SIMPLE2D` output closely. Thus, the same technique used there for finding capacitance over length applies here. The necessary steps would be to:

1. Construct the (rectangular) mesh problem, marking free nodes and accounting for dirichlet fixed voltages and Neumann boundaries.
2. Index the free nodes and construct a Finite Difference matrix equation.
3. Solve for potential using CGM or another solver of choice (SOR, Cholesky, etc...)
4. Use the free nodes to construct a triangular mesh as in 1A/2.
5. Use the potentials to solve for the two-element Energy terms.
6. Sum the terms over the mesh. If the mesh covers only a section of the conductor, multiply that work to cover the entire conductor. (i.e., if only a quarter was meshed, multiply by 4.)
7. Plug the energy sum into the equation for unit length capacitance.

Acknowledgements

Figures on residual norms were generated using matplotlib. This is the only library dependency this project has outside of python for the codebase, and pandoc for report generation.

Submission contents

This project is meant to be run on Python >= 3.10 with matplotlib installed, with the aid of MATLAB for running `SIMPLE2D_M`.

The submission code is organized with the following hierarchy:

- `sturdy-bassoon` contains the entirety of code used for assignment 1. Some modifications have been made in the interim, but none of them are especially significant to the development of Assignment 2 outside of quality-of-life. (There is no particular reasoning behind this name; this project has been labelled "`eloquent-baritone`" for similar lack of reasoning.)

> **This folder must also be added to the python path for this project to run. If running in PyCharm, this folder should be marked as a Python Source Folder.**

- `cjm_conductor_simulation.py` contains methods for constructing conductor (rectangular) meshes for Question 3. It is also meant to generate results used in this report for that question outside of the matplotlib diagrams.
- `conjugate_gradient.py` contains the Conjugate Gradient Method implementation used for Question 3.
- `construct_simple2d_input.py` contains code geared towards writing `simple2d`-format input to disk generated from the conductor problem.
- `generate_simple2d_voltages.m` is a three-line MATLAB script for generating and writing the `SIMPLE2D_M` output to disk
- `graph_conjugate_gradient.py` contains code for generating the matplotlib diagrams seen in the report. This is the only file that depends on third-party libraries and is otherwise auxiliary.
- `poetry.lock`/`pyproject.toml` are version control files.
- `utilize_simple2d_output` contains code for estimating the unit length capacitance of the problem within the context of question 2. It depends on `generate_simple2d_voltages.m` having been run.
- `symbolic_wkspce.m` contains some s

> Remark: The main `cjm_conductor_simulation` implementation is still broken. The alternate configuration also present in that file gives the correct results at the expense of simulating the entire conductor cross-section.

References

Tools used:

- Course notes were used as reference for techniques implemented herein.
- Pandoc was used to generate this report PDF.
- Microsoft Excel was used for some manual data analysis (and the makeshift heatmap diagram).
- matplotlib was used **only** to generate some diagrams related to CGM residuals.
- MATLAB was used to run `SIMPLE2D_M`.

There are no other external references to report. The submission archive additionally contains the writeup used to construct the answer for Q1.