Thomas Christinck
260601793

ECSE 543 Assignment II

1. First, we write the area of the triangles as:

$$A = \frac{1}{2}\Delta x \Delta y$$

$$A = \frac{1}{2}(0.02\ m)(0.02\ m)$$

$$A = 2 * 10^{-4}\ m^2$$

From the lecture, we want to minimize the total energy in each of the triangles with area A, given by (not considering ε_0):

$$W^{(e)} = \frac{1}{2}\int_{\Delta e} |\nabla U|^2\, dS$$

$$W^{(e)} = \frac{1}{2}\sum_{i=1}^{3}\sum_{j=1}^{3} U_i U_j \int_{\Delta e} \nabla\alpha_i \cdot \nabla\alpha_j\, dS$$

And, in matrix form:

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

Then, the entries of S are given as:

$$S_{i,j} = \int_{\Delta e} \nabla\alpha_i \cdot \nabla\alpha_j\ dS$$

$$S_{i,j} = A *\ \nabla\alpha_i \cdot \nabla\alpha_j$$

Where

$$\alpha_i(x_i, y_i) = \frac{1}{2A}[(x_{i+1}y_{i+2} -\ x_{i+2}y_{i+1}) + x(y_{i+1} - y_{i+2}) + y(x_{i+2} - x_{i+1})]$$

<u>Triangle (1,2,3)</u>

For $i = \{1,2,3\}$ we have:

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

$$\alpha_1(x_1, y_1) =\ 2500\ (0.02y)$$

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

$$\alpha_2(x_2, y_2) = 2500\,(0.0004 - 0.02x - 0.02y)$$

$$\alpha_3(x_3, y_3) = \frac{1}{2A}[(x_1y_2 - x_2y_1) + x(y_1 - y_2) + y(x_2 - x_1)]$$

$$\alpha_3(x_3, y_3) = 2500\,(-0.0004 + 0.02x)$$

Now, for triangle (1,2,3), we calculate the S-matrix element by element. An example calculation for $i = 1, j = 1$ is shown here.

$$S_{1,1} = (2 * 10^{-4})(\nabla(50y) \cdot \nabla(50y))$$

$$S_{1,1} = (2 * 10^{-4})(2500)$$

$$S_{1,1} = 0.5$$

We perform similar calculations for all other $i \in \{1,2,3\}$, $j \in \{1,2,3\}$. This gives us the following S-matrix for triangle (1,2,3):

$$\boldsymbol{S^{(1,2,3)}} = \begin{bmatrix} \mathbf{0.5} & \mathbf{-0.5} & \mathbf{0} \\ \mathbf{-0.5} & \mathbf{1.0} & \mathbf{-0.5} \\ \mathbf{0} & \mathbf{-0.5} & \mathbf{0.5} \end{bmatrix}$$

Triangle (4,5,6)

For $i = \{4,5,6\}$ we have:

$$\alpha_4(x_4, y_4) = \frac{1}{2A}[(x_5y_6 - x_6y_5) + x(y_5 - y_6) + y(x_6 - x_5)]$$

$$\alpha_4(x_4, y_4) = 2500\,(-0.004 + 0.02x + 0.02y)$$

$$\alpha_5(x_5, y_5) = \frac{1}{2A}[(x_6y_4 - x_4y_6) + x(y_6 - y_4) + y(x_4 - x_6)]$$

$$\alpha_2(x_2, y_2) = 2500\,(0.0004 - 0.02x)$$

$$\alpha_6(x_6, y_6) = \frac{1}{2A}[(x_4y_5 - x_5y_4) + x(y_4 - y_5) + y(x_5 - x_4)]$$

$$\alpha_6(x_6, y_6) = 2500\,(0.0004 - 0.02y)$$

Now, for triangle (4,5,6), we calculate the S-matrix element by element. An example calculation for $i = 1, j = 1$ is shown here.

$$S_{1,1} = (2 * 10^{-4})(\nabla(-1 + 50x + 50y) \cdot \nabla(-1 + 50x + 50y))$$

$$S_{1,1} = (2 * 10^{-4})(2500 + 2500)$$

$$S_{1,1} = 1.0$$

We perform similar calculations for all other $i \in \{4,5,6\},\ j \in \{4,5,6\}$. This gives us the following S-matrix for triangle (4,5,6):

$$\boldsymbol{S^{(4,5,6)}} = \begin{bmatrix} \mathbf{1.0} & \mathbf{-0.5} & \mathbf{-0.5} \\ \mathbf{-0.5} & \mathbf{0.5} & \mathbf{0} \\ \mathbf{-0.5} & \mathbf{0} & \mathbf{0.5} \end{bmatrix}$$

Global S-Matrix

To find the global S-matrix, we find S in the following equation:

$$S = C^T S_{dis} C$$

Here, *C* is a conjoining matrix that maps the disjoint numbering used to find $S^{(1,2,3)}$ and $S^{(4,5,6)}$ to a conjoint numbering that will be used to find the global S-matrix. The mesh has 6 local nodes and 4 global nodes, and so *C* is a 6 x 4 matrix, with local node 5 mapped to global node 1, and local node 6 mapped to global node 3. Thus, we have *C* (and by extension, C^T) as:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad C^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

S_{dis} is a 6 x 6 matrix consisting of $S^{(1,2,3)}$ in the upper left and $S^{(4,5,6)}$ in the lower right.

$$S_{dis} = \begin{bmatrix} 0.5 & -0.5 & 0 & & & \\ -0.5 & 1.0 & -0.5 & & 0 & \\ 0 & -0.5 & 0.5 & & & \\ & & & 1.0 & -0.5 & -0.5 \\ & 0 & & -0.5 & 0.5 & 0 \\ & & & -0.5 & 0 & 0.5 \end{bmatrix}$$

Finally, we obtain the global S-matrix as:

$$S = C^T S_{dis} C$$

$$\boldsymbol{S} = \begin{bmatrix} \mathbf{1.0} & \mathbf{-0.5} & \mathbf{0} & \mathbf{-0.5} \\ \mathbf{-0.5} & \mathbf{1.0} & \mathbf{-0.5} & \mathbf{0} \\ \mathbf{0} & \mathbf{-0.5} & \mathbf{1.0} & \mathbf{-0.5} \\ \mathbf{-0.5} & \mathbf{0} & \mathbf{-0.5} & \mathbf{1.0} \end{bmatrix}$$

2. (a) A program was written in MATLAB specifying the file to be given to the provided SIMPLE2D_m.m program. The program can be found under Appendix 1.1 as *write_mesh_file.m* (page 9). The generated file can be found under Appendix 1.2 as *toSimple2D.txt* (page 12) and has 46 elements as expected. The node numbering convention used is shown in Figure 1.

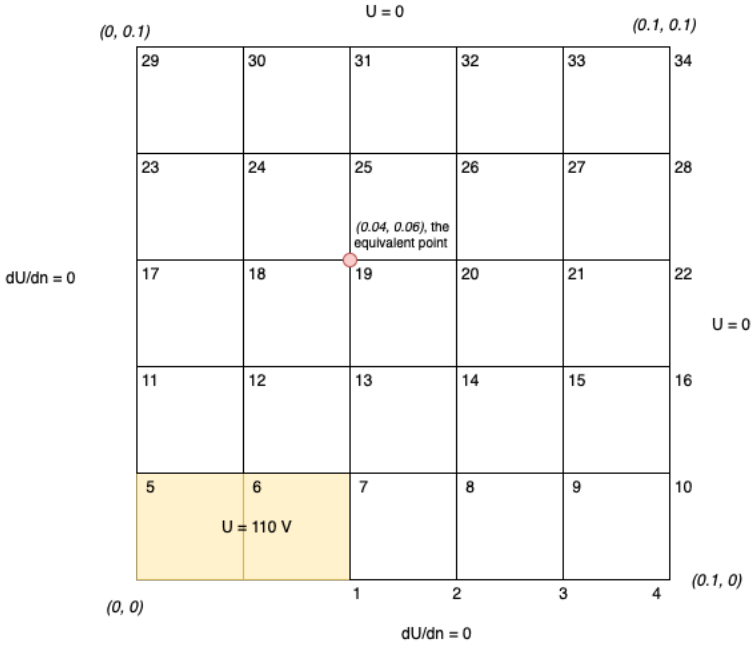

Figure 1 - Illustration showing the node numbering used to generate the input file for SIMPLE2D_m.m. Note that the point of interest (0.06, 0.04) is equivalent to (0.04, 0.06) in this quadrant due to symmetry.

2. (b) The file mentioned in 2(a) was provided to the SIMPLE2D_M.m program, and the output data is included in Appendix 1.3 as *Simple2D Output* (page 13). Note that point (x, y) = *(0.06, 0.04)* lies in the lower right quadrant of the cross-section of the coaxial cable. Since we are considering only the upper right quadrant here, this point is equivalent to (x', y') = (0.04, 0.06) (Figure 2). From Appendix 1.3 we can see the potential at *(x,y) = (0.06, 0.04)* (i.e. *(x', y') = (0.04, 0.06)*) was found to be **40.526 V**.

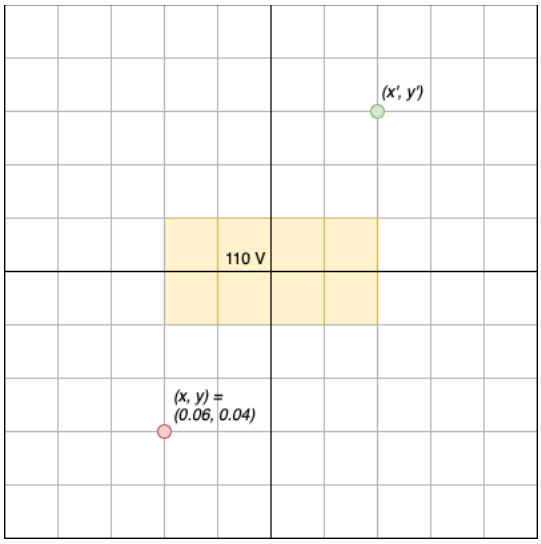

Figure 2 - Illustration showing that (x, y) = (0.06, 0.04) has the equivalent point (x', y') = (0.04, 0.06) in the upper-right quadrant

2. (c) Now, we can compute the capacitance per unit length of the system using the *SIMPLE2D_M.m* output results in Appendix 1.3. The energy in a capacitor with capacitance C and potential V can be written as:

$$E = \frac{1}{2}CV^2$$

From question 1, we have the energy in a potential field as:

$$W = \varepsilon_0 * \frac{1}{2}U^T SU$$

Now, equating the two expressions we have an expression for the capacitance per unit-length as:

$$C = \varepsilon_0 \frac{U^T SU}{V^2}$$

Here, *S* is the global *S*-matrix for the mesh, which we obtain from *SIMPLE2D_M.m*, and *U* is a vector $U = [U_1 \dots U_N]^T$, where the elements are the potentials at each of the *N* nodes in the mesh. A simple program was written to solve this equation and can be found in Appendix 1.4 under *compute_capacitance.m*, (page 14). Code for the matrix operations (multiplication, transposition) can be found in Appendix 4. The program was run and the capacitance per unit length of the cable was found to be **52.137 pF/m**.

3. A program to generate the matrix equation was written and can be found in Appendix 2.1 as *generate_matrix.m* (page 15). A program implementing the conjugate gradient method was written and can be found under Appendix 2.2 as *conjugate_gradient.m* on page 17.

3. (a) The generated matrix was tested using the Cholesky decomposition program written for Assignment 1 (see *cholesky.m* under Appendix 3.3 on page 21). However, the program finds that the matrix is not positive definite. We know that for a regular matrix A, A^TA is always positive definite. Thus, we multiply our equation by A^T and solve for x.

3. (b) The matrix equation was solved using the Cholesky decomposition program from Assignment 1. The program output is shown in Figure 3. The matrix equation was also solved using the conjugate gradient method. The program output is shown in Figure 4.

```
-------- Cholesky Decomposition Solution -------------

cholesky_x =

   66.6737
   31.1849
   62.7550
   29.0330
   77.3592
   75.4690
   67.8272
   45.3132
   22.1921
   48.4989
   46.6897
   40.5265
   28.4785
   14.4223
   23.2569
   22.2643
   19.1107
   13.6519
    7.0186
```

Figure 3- Program output for x using the Cholesky decomposition algorithm to solve Ax = b

```
-------- Conjugate Gradient Solution -------------

 Plotting graphs for L2 and Infinity Norms...

conjugate_grad_x =

   66.6737
   31.1849
   62.7550
   29.0330
   77.3592
   75.4690
   67.8272
   45.3132
   22.1921
   48.4989
   46.6897
   40.5265
   28.4785
   14.4223
   23.2569
   22.2643
   19.1107
   13.6519
    7.0186
```

Figure 4 - Program output for x using the conjugate gradient method to solve Ax = b.

3. (c) Figure 5 shows the infinity norm of the residual vector versus the number of iterations for the conjugate gradient method. Figure 6 shows the L2 norm of the residual vector versus the number of iterations. Relevant code can be found under Appendix 2.3 under testq3.m (page 19).

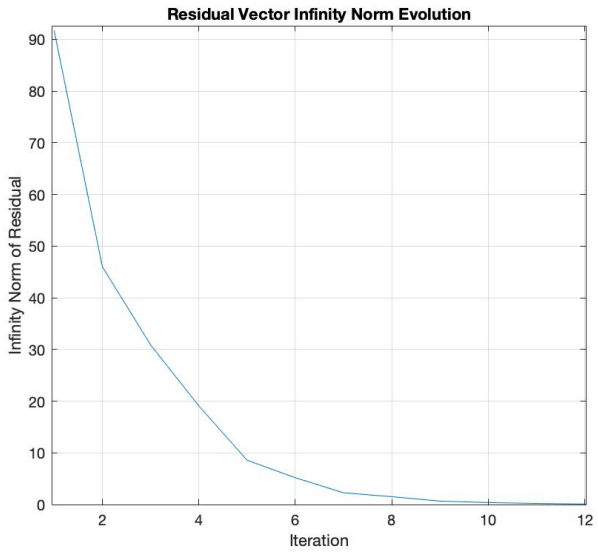

Figure 5 - The infinity norm of the residual vector versus the number of iterations of the conjugate gradient method.

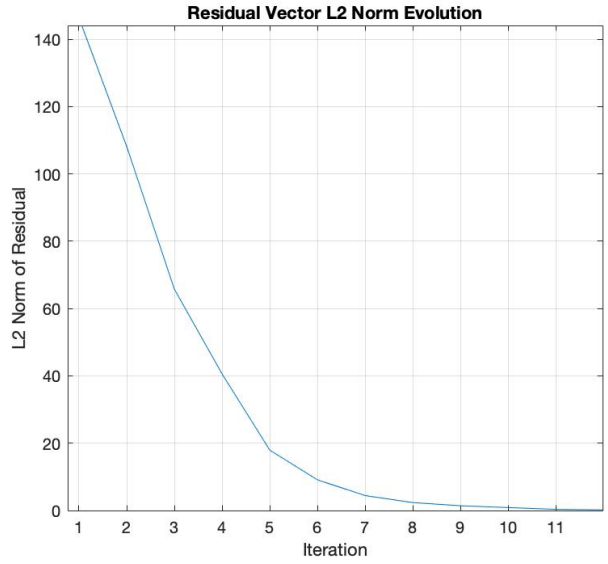

Figure 6 - The L2 norm of the residual vector versus the number of iterations of the conjugate gradient method.

3. (d) The potential at (x, y) = (0.06, 0.04) is found to be **40.526 V** using the conjugate gradient method. This matches the value of 40.526 V found using the Cholesky

decomposition method in (b). These potentials are obtained by observing the solution to our matrix problem at node 12 (this corresponds to node 19 when we re-number the mesh according to free nodes – see Figures 1, 3, and 4). The computed potentials and corresponding methods are tabulated in Table 1. Note that using SOR with the same node spacing (0.02), we find a potential that is less accurate. However, SOR will obtain a more accurate solution as the node spacing decreases.

Method	*Potential (V) at (0.06, 0.04)*
Conjugate Gradient	40.526
Cholesky Decomposition	40.526
SOR (grid size 0.02)	42.265

Table 1 - Calculated potential at (x, y) = (0.06, 0.04) for various methods

3. (e) The capacitance per unit length can be computed as:

$$C = \frac{Q}{V}$$

Here, we can find Q by approximating Gauss's Law as a discrete sum, about contour *L*, where *L* is the loop formed by the outer shell of the coaxial cable.

$$Q = \varepsilon_0 \sum_{i=1}^{N} U_i * l$$

Note l here is equivalent to *h = 0.02*. Thus, we can write Q as a sum of node potentials multiplied by the permittivity of free space. The nodes of interest here (re-numbered considering only free nodes) are *i = {2, 4, 9, 14, 15, 16, 17, 18, 19}.* To get *Q* for the entire cable, we scale node 19 by a factor of 2, and nodes 2 and 15 by a factor of 0.5. Then, we multiply the sum by a factor of 4. This gives:

$$C = 4 * \varepsilon_0 \frac{\frac{1}{2}U_2 + U_4 + U_9 + U_{14} + \frac{1}{2}U_{15} + U_{16} + U_{17} + U_{18} + 2U_{19}}{V}$$

This expression was evaluated in testq3.m (Appendix 2.3 on page 19). As in question 2(c), the capacitance per unit length was found to be **52.136 pF/m**.

APPENDIX 1

1.1 write_mesh_file.m

```
function mesh_file = write_mesh_file()

	% Writes a file 'toSimple2D.txt' describing the mesh in problem 2, that can be interpreted by SIMPLE2D_M.m

	output_file = fopen('toSimple2D.txt','w');

	y_nodes = 6;

	x_nodes = 5;

	potential = 110.0;

	bounds(1:(y_nodes - 1)) = 0;

	for i = 1:(y_nodes - 1)

		bounds(i) = i;

	end;

	mesh(1:x_nodes, 1:y_nodes) = 0;

	for i = 1:x_nodes

		for j = 1:y_nodes

			mesh(i,j) = ((j + 4) + (i-1) * y_nodes);

		end;

	end;

	%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

	% Section one of input file (node #, x coordinate, y coordinate)

	%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

	for i = 1:(y_nodes - 2)

		nbytes = fprintf(output_file, '%5.2f %5.2f %5.2f \n', bounds(i), (i + 1) * 0.02, i * 0.0);

	end;

	for x = 1:x_nodes

		for y = 1:y_nodes

			nbytes = fprintf(output_file, '%5.2f %5.2f %5.2f \n', mesh(x,y), (y - 1) * 0.02, (x) * 0.02);

		end;
```

```
end;

nbytes = fprintf(output_file, '\n');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Section two of input file (mesh node 1 -> node 2 -> node 3 - 0):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:(y_nodes - 3)
        nbytes = fprintf(output_file, '%5d %5d %5d %6.2f \n', bounds(i), bounds(i + 1), mesh(1, i + 2), 0.0);
        nbytes = fprintf(output_file, '%5d %5d %5d %6.2f\n', bounds(i + 1), mesh(1, i + 3), mesh(1, i + 2), 0.0);
end;
for x = 1:(x_nodes - 1)
        for y = 1:(y_nodes - 1)
                nbytes = fprintf(output_file, '%5d %5d %5d %6.2f \n', mesh(x,y), mesh(x, y + 1), mesh(x + 1, y), 0.0);
                nbytes = fprintf(output_file, '%5d %5d %5d %6.2f \n', mesh(x, y + 1), mesh(x + 1, y + 1), mesh(x + 1,
                \       y), 0.0);
        end;
end;
nbytes = fprintf(output_file, '\n');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Section three of input file (nodes with boundary conditions):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get boundary conditions outside the rectangular mesh
nbytes = fprintf(output_file,'%5d %5d \n', 1, potential);
nbytes = fprintf(output_file,'%5d %5d \n', 4, 0.00);
for i = 1:y_nodes
        if mesh(1,i) < 8
                nbytes = fprintf(output_file,'%5d %5d \n', mesh(1,i), potential);
        end;
 end;
```

```
for i = 1:y_nodes
        nbytes = fprintf(output_file,'%5d %5.2f \n', mesh(x_nodes, i), 0.00);
end;
for i = 1:x_nodes
        nbytes = fprintf(output_file,'%5d %5.2f \n', mesh(i, y_nodes), 0.00);
end;
fclose(output_file);
```

1.2 toSimple2D.txt

Section 1:

```
 1.00  0.04  0.00
 2.00  0.06  0.00
 3.00  0.08  0.00
 4.00  0.10  0.00
 5.00  0.00  0.02
 6.00  0.02  0.02
 7.00  0.04  0.02
 8.00  0.06  0.02
 9.00  0.08  0.02
10.00  0.10  0.02
11.00  0.00  0.04
12.00  0.02  0.04
13.00  0.04  0.04
14.00  0.06  0.04
15.00  0.08  0.04
16.00  0.10  0.04
17.00  0.00  0.06
18.00  0.02  0.06
19.00  0.04  0.06
20.00  0.06  0.06
21.00  0.08  0.06
22.00  0.10  0.06
23.00  0.00  0.08
24.00  0.02  0.08
25.00  0.04  0.08
26.00  0.06  0.08
27.00  0.08  0.08
28.00  0.10  0.08
29.00  0.00  0.10
30.00  0.02  0.10
31.00  0.04  0.10
32.00  0.06  0.10
33.00  0.08  0.10
34.00  0.10  0.10
```

Section 2:

```
 1    2    7   0.00
 2    8    7   0.00
 2    3    8   0.00
 3    9    8   0.00
 3    4    9   0.00
 4   10    9   0.00
 5    6   11   0.00
 6   12   11   0.00
 6    7   12   0.00
 7   13   12   0.00
 7    8   13   0.00
 8   14   13   0.00
 8    9   14   0.00
 9   15   14   0.00
 9   10   15   0.00
10   16   15   0.00
11   12   17   0.00
12   18   17   0.00
12   13   18   0.00
13   19   18   0.00
13   14   19   0.00
14   20   19   0.00
14   15   20   0.00
15   21   20   0.00
15   16   21   0.00
16   22   21   0.00
17   18   23   0.00
18   24   23   0.00
18   19   24   0.00
19   25   24   0.00
19   20   25   0.00
20   26   25   0.00
20   21   26   0.00
21   27   26   0.00
21   22   27   0.00
22   28   27   0.00
23   24   29   0.00
24   30   29   0.00
24   25   30   0.00
25   31   30   0.00
25   26   31   0.00
26   32   31   0.00
26   27   32   0.00
27   33   32   0.00
27   28   33   0.00
28   34   33   0.00
```

Section 3:

```
 1  110
 4    0
 5  110
 6  110
 7  110
29 0.00
30 0.00
31 0.00
32 0.00
33 0.00
34 0.00
10 0.00
16 0.00
22 0.00
28 0.00
34 0.00
```

1.3 Simple2D Output

```
ans =

    1.0000    0.0400         0  110.0000
    2.0000    0.0600         0   66.6737
    3.0000    0.0800         0   31.1849
    4.0000    0.1000         0         0
    5.0000         0    0.0200  110.0000
    6.0000    0.0200    0.0200  110.0000
    7.0000    0.0400    0.0200  110.0000
    8.0000    0.0600    0.0200   62.7550
    9.0000    0.0800    0.0200   29.0330
   10.0000    0.1000    0.0200         0
   11.0000         0    0.0400   77.3592
   12.0000    0.0200    0.0400   75.4690
   13.0000    0.0400    0.0400   67.8272
   14.0000    0.0600    0.0400   45.3132
   15.0000    0.0800    0.0400   22.1921
   16.0000    0.1000    0.0400         0
   17.0000         0    0.0600   48.4989
   18.0000    0.0200    0.0600   46.6897
   19.0000    0.0400    0.0600   40.5265
   20.0000    0.0600    0.0600   28.4785
   21.0000    0.0800    0.0600   14.4223
   22.0000    0.1000    0.0600         0
   23.0000         0    0.0800   23.2569
   24.0000    0.0200    0.0800   22.2643
   25.0000    0.0400    0.0800   19.1107
   26.0000    0.0600    0.0800   13.6519
   27.0000    0.0800    0.0800    7.0186
   28.0000    0.1000    0.0800         0
   29.0000         0    0.1000         0
   30.0000    0.0200    0.1000         0
   31.0000    0.0400    0.1000         0
   32.0000    0.0600    0.1000         0
   33.0000    0.0800    0.1000         0
   34.0000    0.1000    0.1000         0
```

Figure i. - SIMPLE2D_m.m output. Note that at (x', y') = (0.04, 0.06) we have 40.526 V

1.4 compute_capacitance.m

Note that matrix manipulation functions (transpose, multiply) can be found in Appendix 3.

```
function C = compute_capacitance(file)

    % Function to compute capacitance per unit length. SIMPLE2D_M has been

    % modified to return the global S-matrix, which is used in determining the

    % total energy in the mesh.

    [Potential, S] = SIMPLE2D_M(file);

    % Potential pot = 110 V, U vector consists of potential at all nodes

    pot = 110;

    U = Potential(:, 4);

    W = 0.5 * matrix_multiply(mat_transpose(U), matrix_multiply(S, U));


    % Calculates the Capacitance per length (we multiply by 4 because there

    % are 4 quadrants)

    e0 = 8.854187817620e-12;

    V2 = pot * pot;

    C = e0 *(2 * W / V2) * 4;
```

APPENDIX 2

2.1 generate_matrix.m

```
function [A, b] = generate_matrix(num_free_nodes)

        % Generates a finite difference mesh with the given number of free nodes.

        % This program is not portable and will only work for Problems in this

        % assignment.

         A(1:num_free_nodes, 1:num_free_nodes) = 0;

         b(1:num_free_nodes, 1) = 0;

        for i = 1:(num_free_nodes)

                A(i, i) = -4;

                % Look for nodes connected on the right

                if mod(i + 1, 5) ~= 0 && i ~= 2 && i ~= 19

                        A(i + 1, i) = 1;

                end;

                % Nodes connected on the top

                if i >= 3 && i <=14

                        A(i + 5, i) = 1;

                elseif i == 1 || i == 2

                        A(i + 2, i) = 1;

                end;

                % Nodes connected on the bottom

                if i >= 8

                        A(i - 5, i) = 1;

                elseif i == 3 || i == 4

                        A(i - 2, i) = 2;

                end;

                % Nodes connected on the left

                if mod(i, 5) ~= 0 && i ~= 3 && i ~= 1

                        A(i - 1, i) = 1;
```

```
        end;
        if mod(i - 1, 5) == 0 && i ~= 1
                A(i - 1, i) = 2;
        end;
        if i == 1 || i == 3 || i == 5 || i == 6 || i == 7
                b(i) = -110.0;
        end;
end;
```

2.2 conjugate_gradient.m

```
function [x] = conjugate_gradient(A, b)
        % Find x in Ax = b using the conjugate gradient method
        size_b = size(b);
        x(1:size_b(1), 1) = 0;
        tolerance = 1.0e-6;
        r = b - matrix_multiply(A, x);
        p = r;
        r_old = matrix_multiply(mat_transpose(r), r);
        r_new = r_old;
         iteration = 1;
        % Set up storage for norms
        npoints = 10000;
        inf_norm_list = cell(npoints, 2);
         l2_norm_list = cell(npoints, 2);
        while sqrt(r_new) > tolerance
                Ap = matrix_multiply(A, p);
                alpha = r_old / matrix_multiply(mat_transpose(p), Ap);
                x = x + p * alpha;
                r = r - Ap * alpha;
                r_new = matrix_multiply(mat_transpose(r), r);
                p = r + p * (r_new / r_old);
                r_old = r_new;
                size_r = size(r);
                squares_sum = 0;
                inf_norm = 0;
                for i = 1:size_r(1)
                        % Update norms
                        squares_sum = squares_sum + r(i)^2;
                        inf_norm = max(abs(r(i)), inf_norm);
```

```
            end;
            l2_norm = sqrt(squares_sum);
            inf_norm_list(iteration, :) = {iteration, inf_norm};
            l2_norm_list(iteration, :) = {iteration, l2_norm};
            iteration = iteration + 1;
end;
inf_norm_list(iteration:end, :) = [ ];
l2_norm_list(iteration:end, :) = [ ];
l2_norms = cell2mat(l2_norm_list);
inf_norms = cell2mat(inf_norm_list);
% Plot graphs for norms
fprintf('\n Plotting graphs for L2 and Infinity Norms... \n ')
figure(1)
plot(l2_norms(:, 1), l2_norms(:,2))
xlabel('Iteration')
ylabel('L2 Norm of Residual')
grid
figure(2)
plot(inf_norms(:, 1), inf_norms(:,2))
xlabel('Iteration')
ylabel('Infinity Norm of Residual')
grid
```

2.3 testq3.m

```
% ECSE 543 Assignment 2

% Question 3

% Script for Conjugate Gradient Node Voltage solver (for testing)

% Generate the matrix problem

[A,b] = generate_matrix(19)

% UNCOMMENT TO CHECK POSITIVE DEFITENESS OF A

%disp("\n---------Positive Definite Check ----------\n")

%cholesky_x = cholesky(A,  b);

disp("-------- Cholesky Decomposition Solution ----------")

% We first take the matrix and make it positive definite and symmetric by multiplying

% both sides by A^T.

cholesky_x = cholesky(matrix_multiply(mat_transpose(A), A),  matrix_multiply(mat_transpose(A), b))

disp("-------- Conjugate Gradient Solution ----------")

conjugate_grad_x = conjugate_gradient(A, b)

disp("-------- Error ----------")

error = (abs(conjugate_grad_x - cholesky_x))

fprintf("Value at (0.06, 0.04) (Cholesky): %6.4f V \n", (cholesky_x(12, 1)))

fprintf("Value at (0.06, 0.04)  (CG): %6.4f V \n", (conjugate_grad_x(12, 1)))

% Compute the capacitance.

U_sum = conjugate_grad_x(2, 1) / 2;

U_sum = U_sum + conjugate_grad_x(4, 1) + conjugate_grad_x(9, 1) + conjugate_grad_x(14, 1);

U_sum = U_sum + conjugate_grad_x(19, 1) * 2;

U_sum = U_sum + conjugate_grad_x(16, 1) + conjugate_grad_x(17, 1) + conjugate_grad_x(18, 1);

U_sum = U_sum + conjugate_grad_x(15, 1) / 2;U_sum = U_sum *  4;

Q = 8.854e-12 * U_sum;

C = Q / 110.0;

fprintf("Total capacitance: %5.3f pF/m \n", C * 1e12)
```

APPENDIX 3

3.1 matrix_multiply.m

```
function[C] = matrix_multiply(A, B)

        %   Function that multiplies two matricies A and B

        %   The number of columns in matrix A must equal the number of rows in matrix B. The

        %   product matrix, C, is returned.

        size_A = size(A);

        size_B = size(B);

        rows_A = size_A(1);

        cols_A = size_A(2);

        rows_B = size_B(1);

        cols_B = size_B(2);

        if cols_A == rows_B

                C(1:rows_A,1:cols_B)=0;

                for i = 1:rows_A

                for j = 1:cols_B

                                for k = 1:cols_A

                                        C(i,j) = C(i,j) + (A(i, k) * B(k,j));

                                end;

                end;

                end;

        else

                error("Cannot multiply the two matrices. Cols_A must equal Rows_B")

        end;
```

3.2 mat_transpose.m

```
function[A_T] = mat_transpose(A)

        %   Function that transposes a matrix A and returns the matrix's transpose  A_T

        %   Author: Thomas Christinck, 2018.

        size_A = size(A);

        rows_A = size_A(1);

        cols_A = size_A(2);


        A_T(1:cols_A,1:rows_A)=0;

        for i = 1:cols_A

                for j = 1:rows_A

                        A_T(i,j) = A(j,i);

                end;

        end;
```

3.3 cholesky.m

```
function [x] = cholesky(A, b)

        %   Cholesky decomposition algorithm - decomposes and then solves using forward elimination

        %   and back substitution. [x] = cholesky(A,b) solves the equation Ax = b. A must be a

        %   symmetric, positive-definite, real matrix.

        size_A = size(A);

        length = size_A(1);

        L(1:length,1:length)=0;

        if length ==1

                L = A;

        else

                for i = 1:length

                        for k = 1:i

                                sum = 0;
```

```
                    if A(i,k) ~= A(k,i)
                        error("The matrix needs to be symmetric")
                    end;
                    for j = 1:k
                        sum = sum + (L(i,j) * L(k,j));
                    end;
                    if (i == k)
                        if ((A(i,i) - sum)) < 0
                            error("The matrix needs to be positive definite")
                        end;
                        L(i,k) = sqrt(abs(A(i,i) - sum));
                    else
                        L(i,k) = (A(i,k) - sum) / L(k,k);
                    end;
                end;
            end;
        end;
        size_L = size(L);
        length = size_L(1);
        y(1:length)=0;
        for i = 1:length
            sum = 0;
            for j = 1:i
                sum = sum + (L(i, j) * y(j));
            end;
            y(i) = (b(i) - sum) / L(i, i);
         end;
        x(1:length, 1)=0;
        for i = length:-1:1
            sum = 0;
```

```
        for j = i:length
                sum = sum + (L(j, i) * x(j));
        end;
        x(i) = (y(i) - sum) / L(i,i);
end;
```