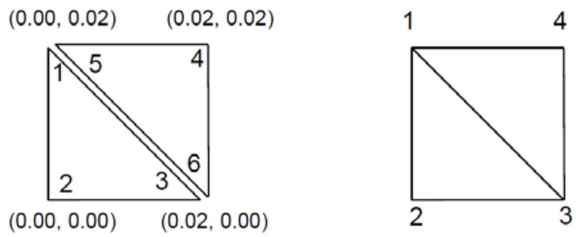
ASSIGNMENT 2: ECSE 543

Jorawar Singh Dham
260899593

NOTE: functions to perform matrix operations are in Appendix C and are used in other programs when necessary.

Q1.



$$\text{Area} = \mathrm{A} = \frac{1}{2} \times 0.02 \times 0.02 = 2 \times 10^{-4}\ \mathrm{m}^2$$

TRIANGLE-1

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

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

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

$$S_{11} = \frac{1}{4A}[(y_2 - y_3)^2 + (x_3 - x_2)^2] = \frac{10^{-4}}{8}[(0-0)^2 + (0.02)^2] = 0.5$$
$$S_{12} = \frac{1}{4A}[(y_2 - y_3)(y_3 - y_1) + (x_3 - x_2)(x_1 - x_3)] = \frac{10^{-4}}{8}[0 + (0.02 - 0)(0 - 0.02)] = -0.5$$
$$S_{13} = \frac{1}{4A}[(y_2 - y_3)(y_1 - y_2) + (x_3 - x_2)(x_2 - x_1)] = \frac{10^{-4}}{8}[0 + (0.02)(0)] = 0$$
$$S_{21} = S_{12} = -0.5$$
$$S_{22} = \frac{1}{4A}[(y_3 - y_1)^2 + (x_1 - x_3)^2] = \frac{10^{-4}}{8}[(-0.02)^2 + (0.02)^2] = 1$$
$$S_{23} = \frac{1}{4A}[(y_3 - y_1)(y_1 - y_2) + (x_1 - x_3)(x_2 - x_1)] = \frac{10^{-4}}{8}[(-0.02)(0.02) + (-0.02)(0)] = -0.5$$
$$S_{31} = S_{13} = 0$$
$$S_{33} = \frac{1}{4A}[(y_1 - y_2)^2 + (x_2 - x_1)^2] = \frac{10^{-4}}{8}[(0.02)^2 + (0)^2] = 0.5$$

This gives us the following S-matrix for triangle 1:

$$\boldsymbol{S_{\Delta 1}} = \begin{bmatrix} \mathbf{0.5} & \mathbf{-0.5} & \mathbf{0} \\ \mathbf{-0.5} & \mathbf{1} & \mathbf{-0.5} \\ \mathbf{0} & \mathbf{-0.5} & \mathbf{0.5} \end{bmatrix}$$

Similarly, for the second triangle:

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

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

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

$$S_{44} = \frac{1}{4A}[(y_5 - y_6)^2 + (x_6 - x_5)^2] = 1$$
$$S_{45} = \frac{1}{4A}[(y_5 - y_6)(y_6 - y_4) + (x_6 - x_5)(x_4 - x_6)] = -0.5$$
$$S_{46} = \frac{1}{4A}[(y_5 - y_6)(y_4 - y_5) + (x_6 - x_5)(x_4 - x_6)] = -0.5$$
$$S_{54} = S_{45} = -0.5$$
$$S_{55} = \frac{1}{4A}[(y_6 - y_4)^2 + (x_4 - x_6)^2] = 0.5$$
$$S_{56} = \frac{1}{4A}[(y_6 - y_4)(y_4 - y_5) + (x_4 - x_6)(x_5 - 4)] = 0$$
$$S_{64} = S_{46} = -0.5$$
$$S_{66} = \frac{1}{4A}[(y_4 - y_5)^2 + (x_5 - x_4)^2] = 0.5$$

This gives us the following S-matrix for triangle 2:

$$\boldsymbol{S_{\Delta 2}} = \begin{bmatrix} \mathbf{1} & \mathbf{-0.5} & \mathbf{-0.5} \\ \mathbf{-0.5} & \mathbf{0.5} & \mathbf{0} \\ \mathbf{-0.5} & \mathbf{0} & \mathbf{0.5} \end{bmatrix}$$

GLOBAL S-MATRIX

$$\boldsymbol{S_{global}} = C^t S_{dis} C$$

$$S_{dis} = \begin{bmatrix} 0.5 & -0.5 & 0 & 0 & 0 & 0 \\ -0.5 & 1 & -0.5 & 0 & 0 & 0 \\ 0 & -0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -0.5 & -0.5 \\ 0 & 0 & 0 & -0.5 & 0.5 & 0 \\ 0 & 0 & 0 & -0.5 & 0 & 0.5 \end{bmatrix}$$

$$C \equiv \quad \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix}$$

$$\boldsymbol{S_{global}} = \begin{bmatrix} \mathbf{1} & \mathbf{-0.5} & \mathbf{0} & \mathbf{-0.5} \\ \mathbf{-0.5} & \mathbf{1} & \mathbf{-0.5} & \mathbf{0} \\ \mathbf{0} & \mathbf{-0.5} & \mathbf{1} & \mathbf{-0.5} \\ \mathbf{-0.5} & \mathbf{0} & \mathbf{-0.5} & \mathbf{1} \end{bmatrix}$$

Q2.

(a) The two element mesh from Q1 was used to create a finite element mesh for one quarter of the cable cross section (bottom left). The program **create_files.m** *(Appendix A.1)* generates the input file according to the specifications of the SIMPLE2D program. The file generated: **input_file.dat** *(Appendix A.2)* lists 34 nodes connected together to make a total of 46 triangular elements across the domain.

(b) **input_file.dat** *(Appendix A.2)* was passed to the SIMPLE2D_M.m program and the output data is shown in figure 1. The **node number 16** corresponds to $(x,y) = (0.06, 0.04)$ and the value of potential at this point is **5.5263 V**.

Command Window

```
>> create_files
>> Potential=SIMPLE2D_M('input_file.dat');
fx >>
```

Potential

PLOTS VARIABLE VIEW

34x4 double

	1	2	3	4	5
1	1	0	0	0	
2	2	0.0200	0	0	
3	3	0.0400	0	0	
4	4	0.0600	0	0	
5	5	0.0800	0	0	
6	6	0.1000	0	0	
7	7	0	0.0200	0	
8	8	0.0200	0.0200	0.9571	
9	9	0.0400	0.0200	1.8616	
10	10	0.0600	0.0200	2.6060	
11	11	0.0800	0.0200	3.0360	
12	12	0.1000	0.0200	3.1714	
13	13	0	0.0400	0	
14	14	0.0200	0.0400	1.9667	
15	15	0.0400	0.0400	3.8834	
16	16	0.0600	0.0400	5.5263	
17	17	0.0800	0.0400	6.3668	
18	18	0.1000	0.0400	6.6135	
19	19	0	0.0600	0	
20	20	0.0200	0.0600	3.0262	
21	21	0.0400	0.0600	6.1791	
22	22	0.0600	0.0600	9.2492	
23	23	0.0800	0.0600	10.2912	
24	24	0.1000	0.0600	10.5490	
25	25	0	0.0800	0	
26	26	0.0200	0.0800	3.9590	
27	27	0.0400	0.0800	8.5575	
28	28	0.0600	0.0800	15	
29	29	0.0800	0.0800	15	
30	30	0.1000	0.0800	15	
31	31	0	0.1000	0	
32	32	0.0200	0.1000	4.2525	
33	33	0.0400	0.1000	9.0919	
34	34	0.0600	0.1000	15	
35					

Figure 1: Output of SIMPLE2D_M.m

(c) The capacitance of the cross section of the cable is the capacitance per unit length of the cable. Considering the cross section of the cable as a capacitor, the total energy stored by a capacitor can be written as:

$$E = \frac{1}{2}CV^2 \; where \; V = 15 \; for \; the \; cross \; section$$

Now, the energy stored in one quarter of the cable cross section can be calculated as:

$$W = \frac{1}{2}\varepsilon_o U^t S U$$

Where U is the potential vector and S is the global connectivity matrix. The program SIMPLE2D_M.m was modified to also return the value of global connectivity matrix S, in addition to the output matrix of figure 1 in Q2 (b).

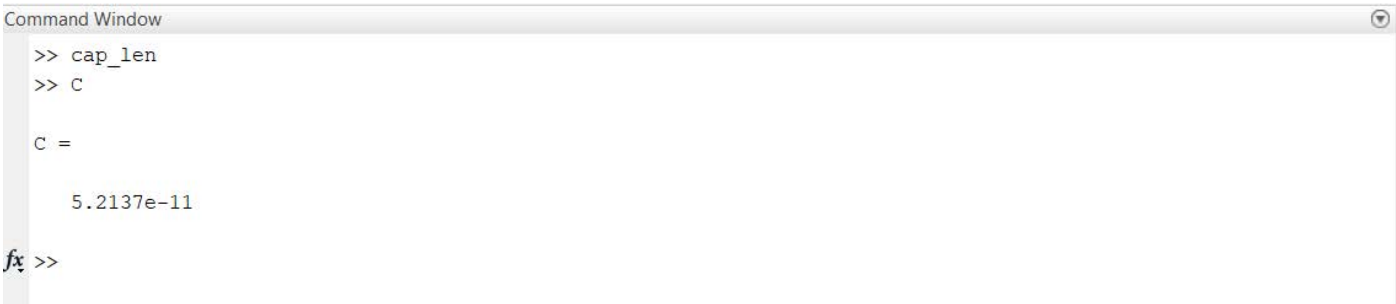
Now, for the entire cable:

$$W_{total} = 4 \times W = 2\varepsilon_o U^t S U$$

By equating the energy, we can calculate C as:

$$C = \frac{4\varepsilon_o U^t S U}{V^2}$$

A program was written to calculate C using this procedure and is listed as **cap_len.m** *(Appendix A.3)*. The capacitance per unit length of the cable was found to be **5.2137 e-11 F/m** or **52.137 pF/m.**


```
Command Window
>> cap_len
>> C

C =

   5.2137e-11

fx >>
```

Figure 2: output of cap_len.m to calculate capacitance per unit length

Q3. A program to implement un-preconditioned conjugate gradient method was written as **CG.m** *(Appendix B.3)*. A matlab script: **A_b.m** *(Appendix B.1)* was also written to create the matrices for the matrix equation corresponding to a finite difference node-spacing, h = 0.02m in x and y directions for the same one-quarter cross-section of the system as the one in Q2.

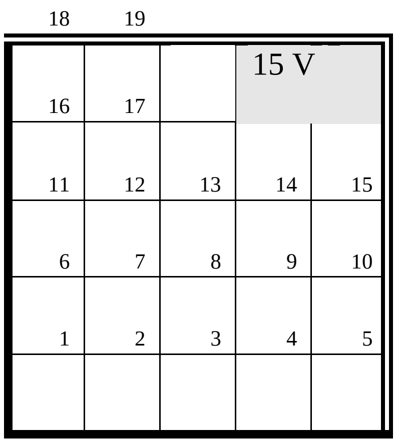


Figure 3:chosen node numbering scheme for the free nodes of mesh(h=0.02)

(a) The generated matrix equation was tested using the Cholesky decomposition program from assignment 1: **Cholesky.m** *(Appendix B.2)*. The matrix was found to not be symmetric positive definite (shown in figure 4). To modify the matrix equation to be symmetric positive definite we left multiply the matrix by the transpose of A. The equation now becomes:

$$A_{new} = A^t A, \qquad b_{new} = A^t b$$
$$A_{new} x = b_{new}$$

```
Command Window
>> A_b
>> [L,U,x] = Cholesky(A,b)

L =

  Columns 1 through 5

   0.0000 + 2.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 - 0.5000i   0.0000 + 1.9365i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 - 0.5164i   0.0000 + 1.9322i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.5175i   0.0000 + 1.9319i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 1.0353i   0.0000 + 1.7112i
   0.0000 - 0.5000i   0.0000 - 0.1291i   0.0000 - 0.0345i   0.0000 - 0.0092i   0.0000 - 0.0056i
   0.0000 + 0.0000i   0.0000 - 0.5164i   0.0000 - 0.1380i   0.0000 - 0.0370i   0.0000 - 0.0224i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.5175i   0.0000 - 0.1387i   0.0000 - 0.0839i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.5176i   0.0000 - 0.3132i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.5844i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
```

Figure 4:output of running Cholesky decomposition on A

(b) The modified matrix equations were solved using both Cholesky decomposition and the conjugate gradient methods. The output for both methods are shown in figure 5.

```
Command Window
>> A_b
>> A_new=mat_mul(mat_trans(A),A);
>> b_new=mat_mul(mat_trans(A),b);
>> [L,U,x] = Cholesky(A_new,b_new);
>> x

x =

    0.9571
    1.8616
    2.6060
    3.0360
    3.1714
    1.9667
    3.8834
    5.5263
    6.3668
    6.6135
    3.0262
    6.1791
    9.2492
   10.2912
   10.5490
    3.9590
    8.5575
    4.2525
    9.0919

fx >>
```

(i)

```
Command Window
>> A_b
>> A_new=mat_mul(mat_trans(A),A);
>> b_new=mat_mul(mat_trans(A),b);
>> x = CG(A_new,b_new)

x =

    0.9571
    1.8616
    2.6060
    3.0360
    3.1714
    1.9667
    3.8834
    5.5263
    6.3668
    6.6135
    3.0262
    6.1791
    9.2492
   10.2912
   10.5490
    3.9590
    8.5575
    4.2525
    9.0919

fx >>
```

(ii)

Figure 5: (i)Program output for x using the Cholesky Algorithm and (ii) Program output using the conjugate gradient method

(c) Part of the code in **CG.m** *(Appendix B.3)* plots the required graphs. Figure6 shows the plot of 2norm of the residual for each iteration. Similarly, Figure 7 shows the plot of infinity norm of the residual for each iteration.

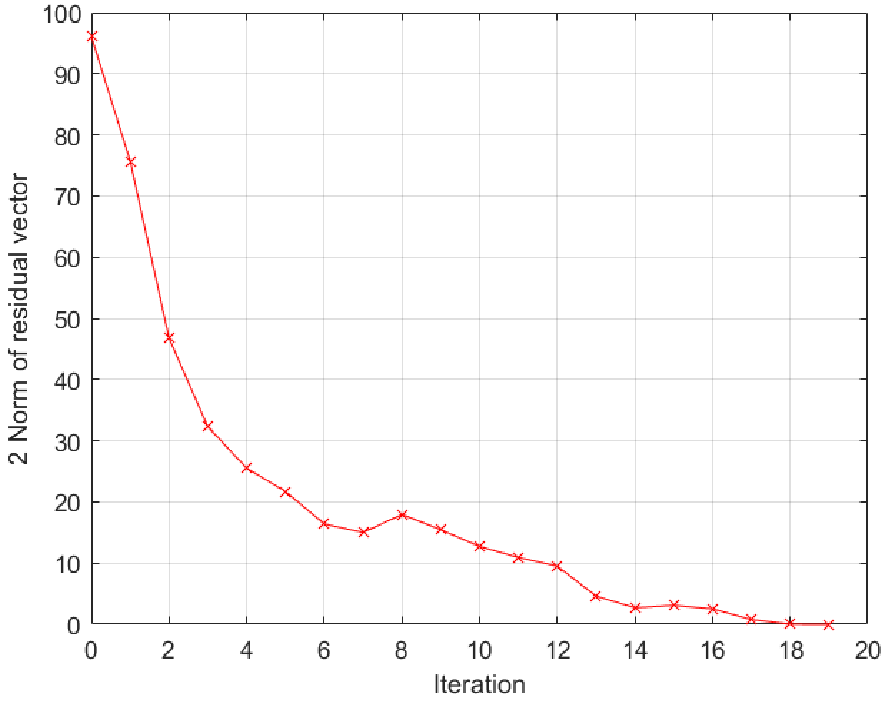


Figure 6: 2norm of the residual for each iteration

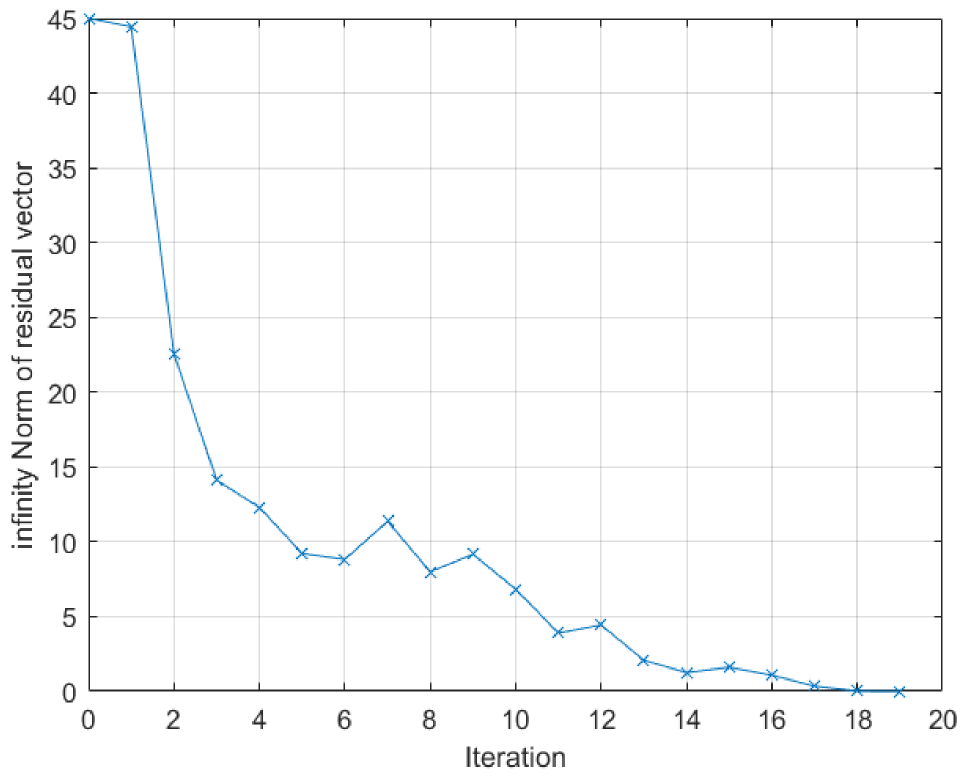


Figure 7: infinity norm of the residual for each iteration

(d) From figure 3, we can see that the node number 8 corresponds to (x, y) = (0.06, 0.04). Using the conjugate gradient method, we find that the value of potential at this point is **5.5263 V** [see figure 5 (ii)]. This matches the value found using the Cholesky decomposition method in (b). For a more accurate comparison of the outputs of the different methods, see the table below:

METHOD	POTENTIAL(V) AT (0.06, 0.04)
Cholesky	5.52634126516712
Conjugate Gradient	5.52634127307277
SOR (ω=1.35)	5.526337806822303

Table 1: Potential (V) at (x, y) = (0.06, 0.04) for various methods

(e) METHOD 1:

We can construct the potential vector as seen in Q2 (figure1) by including the potentials at the fixed nodes in the results obtained from applying the conjugate gradient method (figure 5 (ii)).Then the same procedure as Q2(c) can be applied to obtain the capacitance per unit length.

METHOD 2:

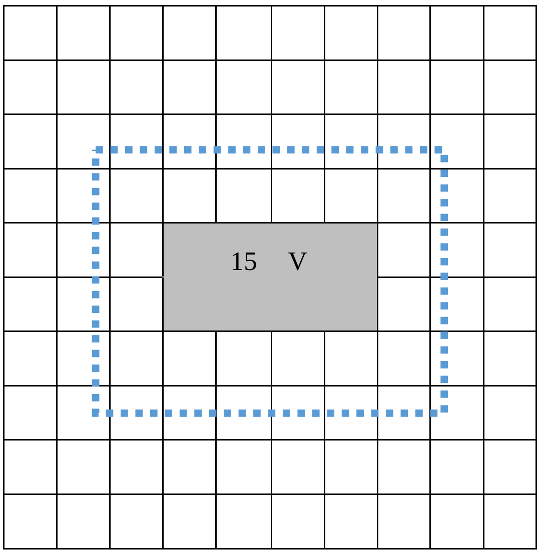


Figure 8: Cable cross section (showing a closed contour around core)

We can find the capacitance per unit length of this cable by using the charge per unit length on the inner conductor: $\frac{C}{L} = \frac{Q_{inner}}{VL}$ where V=15

Using Gauss Law: $\frac{Q_{inner}}{\varepsilon_o} = \oiint E.ds$

The Gaussian surface is a cylinder (rectangular cross-section) of length L in the z-direction. Hence:

$$Q_{inner} = \oint \varepsilon_o E_n dl \int_0^L dz = L \oint \varepsilon_o E_n dl$$

$$\frac{Q_{inner}}{L} = \oint \varepsilon_o E_n dl$$

Here the line integral is over a closed contour surrounding the core as shown in figure 8 and E_n is the component of electric field normal to the contour.

We know that $E = -\nabla U$. Hence we can use the potential at pairs of nodes which lie on either side of the contour to approximate the normal component of the electric field:

$$E_{n_i} = \frac{U_{node\ inside\ contour} - U_{node\ outside\ contour}}{h_x}$$

Using this expression, we can approximate the integration using a summation:

$$\int_{segment} \varepsilon_o E_n dl = \sum_{i=1}^{N} \varepsilon_o E_{n_i} h_y$$

where N represents the number of node pairs that along that segment of the contour.

Repeat this summation for all four segments to get $\oint \varepsilon_o E_n dl = \frac{Q_{inner}}{L}$

Finally use the value of $\frac{Q_{inner}}{L}$ to calculate $\frac{C}{L}$

APPENDIX

A. Code for Question 2

A.1 create_files.m

```
%Create the input file for the SIMPLE2D program.
clear()
f= fopen('input_file.dat','w');
for i=1:6 %create 1st section of file: position of each node
    for j=1:6
        node_pos=j+6*(i-1);
        x=(j-1)*0.02;
        y=(i-1)*0.02;
        if node_pos<35
            fprintf(f, '%5d %5.2f %5.2f \n', node_pos, x, y);
        end
    end
end

fprintf(f,'\n');%add a blank row

for i=1:5 %create 2nd section of file: form triangular elements from the nodes
    for j=1:5
       node_pos=j+6*(i-1);
       if node_pos<28
            fprintf(f, '%5d %5d %5d %5.2f \n', node_pos,node_pos+1 , node_pos+6 , 0);
            fprintf(f, '%5d %5d %5d %5.2f \n', node_pos+1,node_pos+7 , node_pos+6 , 0);
       end
    end
end

fprintf(f,'\n');%add a blank row

for i=1:6%create 3rd section of file: specifiy prescribed nodes on dirichlet boundaries
    for j=1:6
        node_pos=j+6*(i-1);
        if i==1 || j==1
            fprintf(f,'%5d %5.2f \n', node_pos,0);
        elseif node_pos==28 || node_pos==29 || node_pos==30 || node_pos==34
            fprintf(f,'%5d %5.2f \n', node_pos,15);
        end

    end
end
fclose(f);
```

A.2 input_file.dat

```
    1  0.00  0.00
    2  0.02  0.00
    3  0.04  0.00
    4  0.06  0.00
    5  0.08  0.00
    6  0.10  0.00
    7  0.00  0.02
    8  0.02  0.02
    9  0.04  0.02
   10  0.06  0.02
```

11	0.08	0.02
12	0.10	0.02
13	0.00	0.04
14	0.02	0.04
15	0.04	0.04
16	0.06	0.04
17	0.08	0.04
18	0.10	0.04
19	0.00	0.06
20	0.02	0.06
21	0.04	0.06
22	0.06	0.06
23	0.08	0.06
24	0.10	0.06
25	0.00	0.08
26	0.02	0.08
27	0.04	0.08
28	0.06	0.08
29	0.08	0.08
30	0.10	0.08
31	0.00	0.10
32	0.02	0.10
33	0.04	0.10
34	0.06	0.10

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
4	5	10	0.00
5	11	10	0.00
5	6	11	0.00
6	12	11	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
10	11	16	0.00
11	17	16	0.00
11	12	17	0.00
12	18	17	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
16	17	22	0.00
17	23	22	0.00
17	18	23	0.00
18	24	23	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
22	23	28	0.00
23	29	28	0.00

```
23    24    29  0.00
24    30    29  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

 1  0.00
 2  0.00
 3  0.00
 4  0.00
 5  0.00
 6  0.00
 7  0.00
13  0.00
19  0.00
25  0.00
28 15.00
29 15.00
30 15.00
31  0.00
34 15.00
```

A.3 cap_len.m

```
%To calculate capacitance per unit length of the cable
clear()
[U,s]=SIMPLE2D_M('input_file.dat');
V=U(:,4);
W=8.85418782e-12*0.5*mat_mul(mat_trans(V),mat_mul(s,V));%calculate energy of one quarter of the cable
E=4*W;%total enerfy stored in cross section of cable
C=(2*E)/225;%equate total energy to 1/2CV^2 and calculate C (V=15).
```

B. Code for Question 3

B.1 A_b.m

```
%generate the matrices A and b for the matrix equation
clear()
for i=1:19%initialise the matrices
    for j=1:19
        A(i,j)=0;
        b(i,1)=0;
    end
end

for i=1:19%fill the matrix according to location of node
    A(i,i)=-4;
    if i==1
        A(i,i+1)=1;
        A(i,i+5)=1;
    elseif i==2 || i==3 || i==4
        A(i,i-1)=1;
        A(i,i+1)=1;
        A(i,i+5)=1;
    elseif i==5
        A(i,i-1)=2;
        A(i,i+5)=1;
    elseif i==6 || i==11
        A(i,i+1)=1;
        A(i,i-5)=1;
        A(i,i+5)=1;
    elseif i==13 || i==14
        A(i,i-1)=1;
        A(i,i+1)=1;
        A(i,i-5)=1;
        b(i,1)=-15;
    elseif i==10
        A(i,i-1)=2;
        A(i,i-5)=1;
        A(i,i+5)=1;
    elseif i==15
        A(i,i-1)=2;
        A(i,i-5)=1;
        b(i,1)=-15;
    elseif i==16
        A(i,i+1)=1;
        A(i,i+2)=1;
        A(i,i-5)=1;
    elseif i==17
        A(i,i-1)=1;
        A(i,i-5)=1;
        A(i,i+2)=1;
        b(i,1)=-15;
    elseif i==18
        A(i,i+1)=1;
        A(i,i-2)=2;
    elseif i==19
        A(i,i-2)=2;
        A(i,i-1)=1;
        b(i,1)=-15;
    else
        A(i,i-1)=1;
        A(i,i+1)=1;
```

```
        A(i,i+5)=1;
        A(i,i-5)=1;
    end
end
```

B.2 Cholesky.m

```
function [L,U,x] = Cholesky(A,b)
%%Function to solve Ax=b using Choleski algorithm :takes A,b as input
    n=size(A,1);
    for j=1:n %to decompose A into L and transpose of L (= U)
        for i=1:n
            temp_sum=0;
            if(i<j)
                L(i,j)=0;
                U(j,i)=0;
            elseif(i==j)
                for m = 1:j-1
                    temp_sum = temp_sum + (L(j,m)^2);
                end
                L(j,j)= sqrt(A(j,j)-temp_sum);
                U(j,j)=L(j,j);
            else
                for m= 1:j-1
                    temp_sum = temp_sum + (L(i,m)*L(j,m));
                end
                L(i,j)= (A(i,j)-temp_sum)/L(j,j);
                U(j,i)=L(i,j);
            end
        end
    end

    for i=1:n %FORWARD ELIMINATION(Put Ux=y and solve Ly=b for y)
        temp_sum=0;
        for j=1:i-1
            temp_sum = temp_sum + L(i,j)*y(j,1);
        end
        y(i,1) = (b(i,1)-temp_sum)/L(i,i);
    end
    for i=n:-1:1%BACKWARD ELIMINATION(solve Ux=y for x)
        temp_sum=0;
        for j=n:-1:i+1
            temp_sum = temp_sum + U(i,j)*x(j,1);
        end
        x(i,1) = (y(i,1)-temp_sum)/U(i,i);
    end
end
```

B.3 CG.m

```
function x = CG(A,b)
%Conjugate Gradient method to solve Ax=b
n=length(b);
for i=1:n
x(i,1)=0;%initial guess
end
```

```
r=mat_sub(b,mat_mul(A,x));
p=r;
norm_2(1)=sqrt(mat_mul(mat_trans(r),r));
norm_inf(1)=0;
for i=1:n
    if abs(r(i,1))> norm_inf(1)
        norm_inf(1)= abs(r(i,1));
    end
end
itr(1)=0;
i=1;
while norm_2(i)>1e-5%iterate while 2 norm is < 1e-5 (similar to SOR)
    alpha = mat_mul(mat_trans(p),r)/mat_mul(mat_trans(p),mat_mul(A,p));
    x = mat_sum(x,s_mul(alpha,p));
    r = mat_sub(b,mat_mul(A,x));
    beta = -mat_mul(mat_trans(p),mat_mul(A,r))/mat_mul(mat_trans(p),mat_mul(A,p));
    p = mat_sum(r,s_mul(beta,p));
    i=i+1;
    norm_2(i)=sqrt(mat_mul(mat_trans(r),r));
    norm_inf(i)=0;
    for j=1:n
        if abs(r(j,1))> norm_inf(i)
            norm_inf(i)=abs(r(j,1));
        end
    end
    itr(i)=i-1;
end
%plot 2 norm
figure(1)
plot(itr,norm_2,'x-')
xlabel('Iteration')
ylabel('2 Norm of residual vector')
grid
%plot inf norm
figure(2)
plot(itr,norm_inf,'x-')
xlabel('Iteration')
ylabel('infinity Norm of residual vector')
grid
end
```

C. General Matrix Operations

C.1 s_mul.m

```
function M = s_mul(a,A)%function to multiply  matrix by a scaler
[n,m]=size(A);
for i=1:n
    for j=1:m
        A(i,j)=a*A(i,j);
    end
end
M=A;
end
```

C.2 mat_trans.m

```
function At = mat_trans(A)
%function to transpose a matrix

[n,m]=size(A);
for i=1:n
    for j=1:m
        At(j,i)=A(i,j);
    end
end
end
```

C.3 mat_sum.m

```
function A = mat_sum(B,C)%function to add two matrices
[n1,m1]=size(B);
[n2,m2]=size(C);
if m1==m2 && n1==n2
    for i=1:n1
        for j=1:m1
            A(i,j)=B(i,j)+C(i,j);
        end
    end
else
    disp('matrices are not the same size');
    A=nan;
    return;
end
end
```

C.4 mat_sub.m

```
function A = mat_sub(B,C)%function to subtract two matrices
[n1,m1]=size(B);
[n2,m2]=size(C);
if m1==m2 && n1==n2
    for i=1:n1
```

```
        for j=1:m1
            A(i,j)=B(i,j)-C(i,j);
        end
    end
else
    disp('matrices are not the same size');
    A=nan;
    return;
end
end
```

C.5 mat_mul.m

```
function M = mat_mul(A,B)%function to multiply two matrices
[n1,m1]=size(A);
[n2,m2]=size(B);
    if m1~=n2
        disp('size mismatch');
        M=nan;
        return;
    else
        for i=1:n1
            for j=1:m2
                M(i,j)=0;
                for t=1:m1
                    M(i,j)=M(i,j)+ A(i,t)*B(t,j);
                end
            end
        end
    end
end
```