Tiffany Wang
260684152

ECSE 543 – Assignment 2

TABLE OF CONTENTS

Question 1

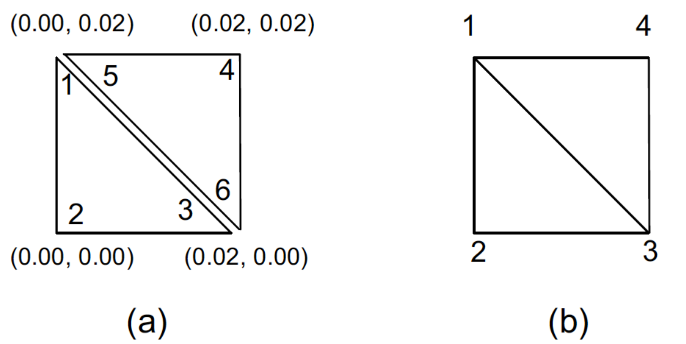


Figure 1. (a) disjoint node numbering (b) global node numbering

- α calculations:

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

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

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

- S calculation:

$$S_{ij} = \nabla\alpha_i \cdot \nabla\alpha_j \cdot A$$
$$e.g.: S_{12} = \frac{1}{4A}[(y_2 - y_3)(y_3 - y_1) + (x_3 - x_2)(x_1 - x_3)]$$
$$e.g.: S_{11} = \frac{1}{4A}[(y_2 - y_3)^2 + (x_3 - x_2)^2]$$

i	x_i	y_i
1	0.00	0.02
2	0.00	0.00
3	0.02	0.00

$$A = \frac{0.02 \cdot 0.02}{2} = 0.0002$$

$$S_{123} = \begin{bmatrix} 0.5 & -0.5 & 0 \\ -0.5 & 1 & -0.5 \\ 0 & -0.5 & 0.5 \end{bmatrix}$$

i	x_i	y_i
4	0.02	0.02
5	0.00	0.02
6	0.02	0.00

$$A = \frac{0.02 \cdot 0.02}{2} = 0.0002$$

$$S_{456} = \begin{bmatrix} 1 & -0.5 & -0.5 \\ -0.5 & 0.5 & 0 \\ -0.5 & 0 & 0.5 \end{bmatrix}$$

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

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

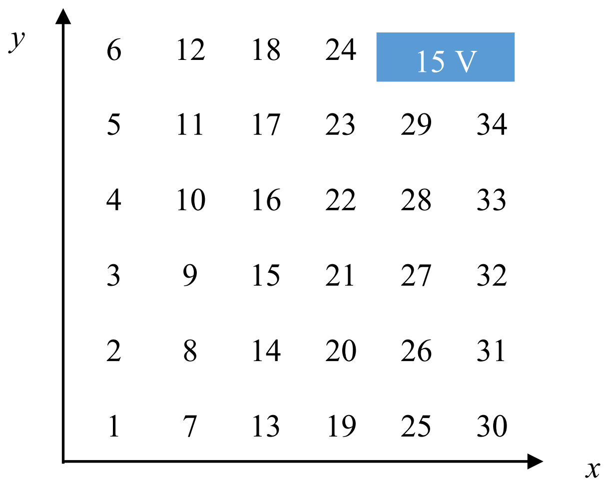
$$S_{global} = C^T S_{dis} C$$

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

$$= \begin{bmatrix} 1 & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{3}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & -\frac{1}{2} & 0 & 1 \end{bmatrix}$$

Question 2

I used the following node numbering to solve the problem



There is a total of 34 nodes and 46 elements (triangles).
Nodes 1, 2, 3, 4, 5, 6, 7, 13, 19, 25, 30 are fixed at 0V
Nodes 23, 24, 29, 34 are fixed at 15V

The columns on the following page are the inputs into the Matlab SIMPLE2D program.

1	0.00	0.00
2	0.00	0.02
3	0.00	0.04
4	0.00	0.06
5	0.00	0.08
6	0.00	0.10
7	0.02	0.00
8	0.02	0.02
9	0.02	0.04
10	0.02	0.06
11	0.02	0.08
12	0.02	0.10
13	0.04	0.00
14	0.04	0.02
15	0.04	0.04
16	0.04	0.06
17	0.04	0.08
18	0.04	0.10
19	0.06	0.00
20	0.06	0.02
21	0.06	0.04
22	0.06	0.06
23	0.06	0.08
24	0.06	0.10
25	0.08	0.00
26	0.08	0.02
27	0.08	0.04
28	0.08	0.06
29	0.08	0.08
30	0.10	0.00
31	0.10	0.02
32	0.10	0.04
33	0.10	0.06
34	0.10	0.08

Figure 2. Node numbering and coordinates

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

Figure 3. Triangle definition

1	0.00
2	0.00
3	0.00
4	0.00
5	0.00
6	0.00
24	15.00
23	15.00
29	15.00
34	15.00
30	0.00
25	0.00
19	0.00
13	0.00
7	0.00

Figure 4. Fixed node potentials

b) Potential at each of the 34 nodes, computed by Matlab SIMPLE_2D

```
>> SIMPLE2D_M('file.dat')

ans =

    1.0000         0         0         0
    2.0000         0    0.0200         0
    3.0000         0    0.0400         0
    4.0000         0    0.0600         0
    5.0000         0    0.0800         0
    6.0000         0    0.1000         0
    7.0000    0.0200         0         0
    8.0000    0.0200    0.0200    0.9571
    9.0000    0.0200    0.0400    1.9667
   10.0000    0.0200    0.0600    3.0262
   11.0000    0.0200    0.0800    3.9590
   12.0000    0.0200    0.1000    4.2525
   13.0000    0.0400         0         0
   14.0000    0.0400    0.0200    1.8616
   15.0000    0.0400    0.0400    3.8834
   16.0000    0.0400    0.0600    6.1791
   17.0000    0.0400    0.0800    8.5575
   18.0000    0.0400    0.1000    9.0919
   19.0000    0.0600         0         0
   20.0000    0.0600    0.0200    2.6060
   21.0000    0.0600    0.0400    5.5263
   22.0000    0.0600    0.0600    9.2492
   23.0000    0.0600    0.0800   15.0000
   24.0000    0.0600    0.1000   15.0000
   25.0000    0.0800         0         0
   26.0000    0.0800    0.0200    3.0360
   27.0000    0.0800    0.0400    6.3668
   28.0000    0.0800    0.0600   10.2912
   29.0000    0.0800    0.0800   15.0000
   30.0000    0.1000         0         0
   31.0000    0.1000    0.0200    3.1714
   32.0000    0.1000    0.0400    6.6135
   33.0000    0.1000    0.0600   10.5490
   34.0000    0.1000    0.0800   15.0000
```

Figure 5. Potential approximation obtained with Matlab SIMPLE_2D

The potential at (0.06, 0.04) is 5.53V, which is at node 21.

c) Since we have the nodes potential approximated in b, I will use the following formula to compute the energy inside the quarter of the coaxial cable:

$$S = C^T \cdot S_{dis} \cdot C$$
$$E = \frac{1}{2} {C_{on}}^T \cdot S \cdot C_{on} \cdot \varepsilon_o$$

I wrote a function which parse the file.dat file into the triangle coordinates, local and global S matrices. The code for the function can be found in the appendix.

Since since the coaxial cable is symmetrical, the total energy of the whole cross-section of the cable would be four times that of the quarter cable we studied in this problem.

$$E_{Total} = 4E = 2\, {C_{on}}^T \cdot S \cdot C_{on} \cdot \varepsilon_o$$

Finally, the capacitance per meter of the cable is found using: $C = \frac{2 \cdot E_{Total}}{V^2}$

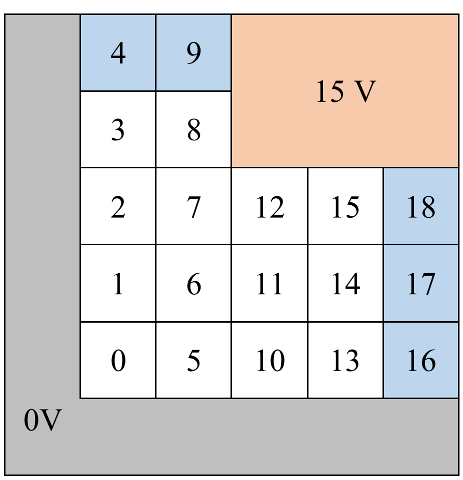
The capacitance per meter of the coaxial cable is found to be: `cap = 5.21127742919e-11`

$$C = 52.11\ pF/m$$

Question 3

The potential of the quarter coaxial cable will be approximated using the Conjugate Gradient method.

A:= five points difference matrix
x:= approximated potential at the nodes



blue: $\frac{\partial \Phi}{\partial x} = 0$

The *b* value is equal to -1 x fixed potential directly neighboring the corresponding node. In this case, every value in *b* is *0*, except for index 8, 9, 12, 15 where *b[index] = -15*

```
A:
[-4, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, -4, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, -4, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, -4, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 2, -4, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, -4, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 1, -4, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 1, -4, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 1, -4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 2, -4, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, -4, 1, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, -4, 1, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, -4, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, -4, 1, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, -4, 1, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, -4, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, -4, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, -4, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, -4]

b:
[0, 0, 0, 0, 0, 0, 0, 0, -15, -15, 0, 0, -15, 0, 0, -15, 0, 0, -15]
```

Figure 6. A and b generated for the conjugate gradient

a) A is not a symmetric matrix: A[4][3] ≠ A[3][4]
   In order to obtain a singular symmetric positive definite matrix, we will multiple both side of the equation by A^T
   We know that $A^T \cdot A = B$ where B is a symmetric matrix.

The resulting equation to be solved is: $A^T \cdot Ax = A^T b$

$A^T \cdot A$:

```
[18, -8, 1, 0, 0, -8, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[-8, 19, -8, 1, 0, 2, -8, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[1, -8, 19, -8, 1, 0, 2, -8, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 1, -8, 22, -12, 0, 0, 2, -8, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, -12, 18, 0, 0, 0, 3, -8, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[-8, 2, 0, 0, 0, 19, -8, 1, 0, 0, -8, 2, 0, 1, 0, 0, 0, 0, 0]
[2, -8, 2, 0, 0, -8, 20, -8, 1, 0, 2, -8, 2, 0, 1, 0, 0, 0, 0]
[0, 2, -8, 2, 0, 1, -8, 20, -8, 1, 0, 2, -8, 0, 0, 1, 0, 0, 0]
[0, 0, 2, -8, 3, 0, 1, -8, 22, -12, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 3, -8, 0, 0, 1, -12, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, -8, 2, 0, 0, 0, 19, -8, 1, -8, 2, 0, 1, 0, 0]
[0, 1, 0, 0, 0, 2, -8, 2, 0, 0, -8, 20, -8, 2, -8, 2, 0, 1, 0]
[0, 0, 1, 0, 0, 0, 2, -8, 1, 0, 1, -8, 19, 0, 2, -8, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, -8, 2, 0, 22, -8, 1, -12, 3, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, -8, 2, -8, 23, -8, 3, -12, 3]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, -8, 1, -8, 22, 0, 3, -12]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, -12, 3, 0, 18, -8, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3, -12, 3, -8, 19, -8]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3, -12, 1, -8, 18]
```

$A^T A$ is a singular symmetric positive definite matrix.

$A^T b =$ `[0, 0, 0, -15, -15, 0, 0, -30, 30, 45, 0, -15, 45, 0, -15, 15, 0, -15, 45]`

b) The results obtained from Choleski and Conjugate Gradient methods are really identical to two decimal places precision.

```
potential using Choleski:
[0.96, 1.97, 3.03, 3.96, 4.25, 1.86, 3.88, 6.18, 8.56, 9.09, 2.61, 5.53, 9.25, 3.04, 6.37, 10.29, 3.17, 6.61, 10.55]
potential at (0.06, 0.04) using Choleski:
5.53
potential at (0.06, 0.04): 5.53

potential using Conjugate Gradient:
[0.96, 1.97, 3.03, 3.96, 4.25, 1.86, 3.88, 6.18, 8.56, 9.09, 2.61, 5.53, 9.25, 3.04, 6.37, 10.29, 3.17, 6.61, 10.55]
potential at (0.06, 0.04) using Conjugate Gradient: 5.53
```

Figure 8. Potential approximation obtained using Choleski and Conjugate Gradient with residuals

c)

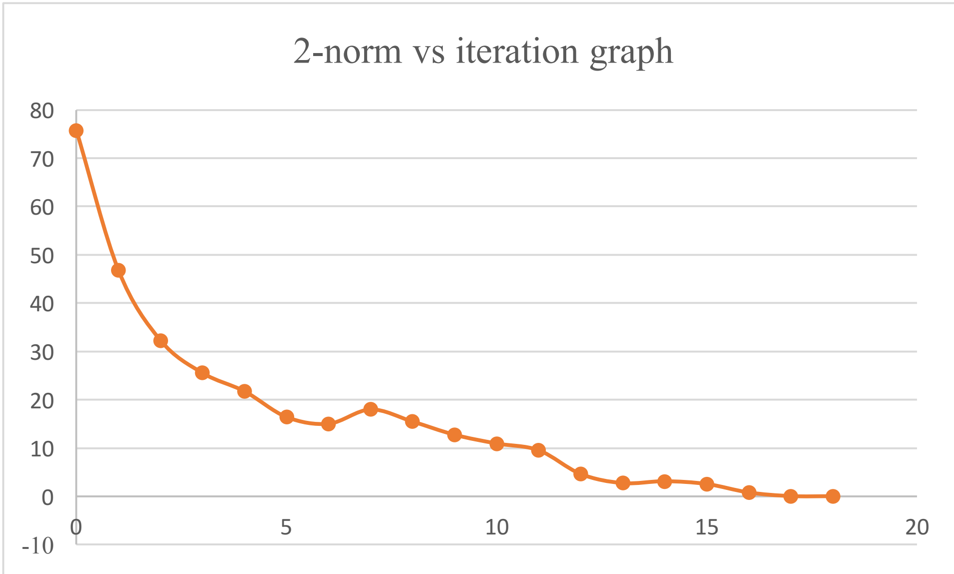
```
iteration 0:      2norm: 75.7      infinity_norm: 44.437
iteration 1:      2norm: 46.784      infinity_norm: 22.536
iteration 2:      2norm: 32.278      infinity_norm: 14.109
iteration 3:      2norm: 25.522      infinity_norm: 12.294
iteration 4:      2norm: 21.72      infinity_norm: 9.209
iteration 5:      2norm: 16.399      infinity_norm: 8.813
iteration 6:      2norm: 15.02      infinity_norm: 11.369
iteration 7:      2norm: 17.972      infinity_norm: 7.987
iteration 8:      2norm: 15.456      infinity_norm: 9.16
iteration 9:      2norm: 12.723      infinity_norm: 6.828
iteration 10:      2norm: 10.919      infinity_norm: 3.893
iteration 11:      2norm: 9.514      infinity_norm: 4.442
iteration 12:      2norm: 4.603      infinity_norm: 2.075
iteration 13:      2norm: 2.713      infinity_norm: 1.252
iteration 14:      2norm: 3.103      infinity_norm: 1.607
iteration 15:      2norm: 2.525      infinity_norm: 1.078
iteration 16:      2norm: 0.771      infinity_norm: 0.337
iteration 17:      2norm: 0.021      infinity_norm: 0.009
iteration 18:      2norm: 0.0      infinity_norm: 0.0
```

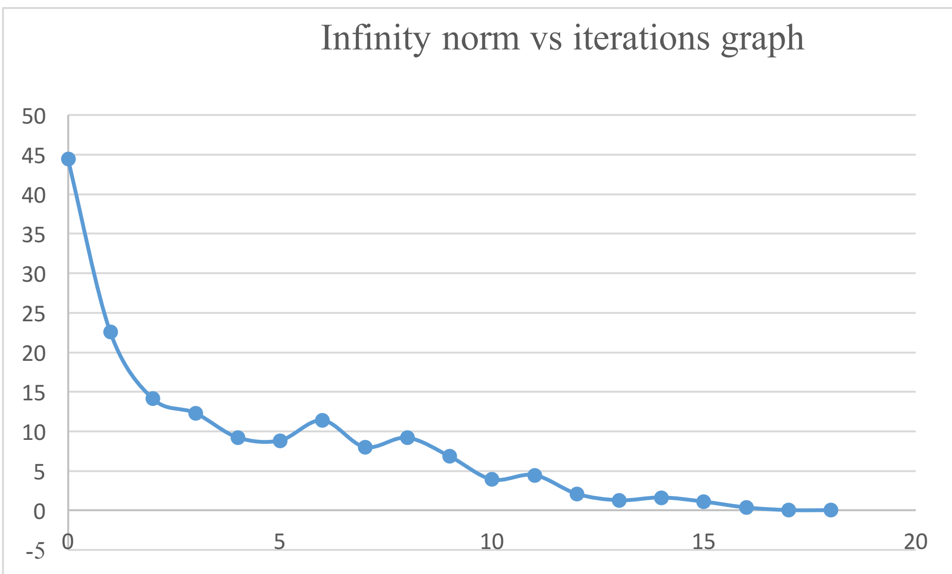
In order to obtain more accurate results, I programmed the method to compute new approximations until the residue is smaller than a threshold of 0.005. However, from the results, we can observe convergence of the approximation with *O(n)* complexity – 19 iterations for 19 nodes. This is consistent with the theoretical expectation of *O(n)* complexity.

The residue vectors norms are calculated using:

$$2 - norm = \sqrt{\sum |res_i{}^2|}$$

$$infinity - norm = \max\left(\{|res_1|, |res_2|, |res_3| \ldots |res_N|\}\right)$$





We observe a general decreasing trend of the residue norms. The residue (error) eventually reaches 0 as the iteration count reaches N (19) as expected. The best fitting trend line would be that of an inverse function.

```
[0.0, 2.6, 5.54, 15.0, 15.0, 15.0]
[0.0, 3.13, 7.3, 15.0, 15.0, 15.0]
[0.0, 2.6, 5.54, 8.77, 9.51, 8.77]
[0.0, 1.73, 3.49, 5.02, 5.52, 5.02]
[0.0, 0.85, 1.66, 2.3, 2.53, 2.3]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Figure 9. Potential approximation results using SOR (from Assignment 1)

The results were really different compared to the results obtained with both Choleski and Conjugate Gradient.

The results from Choleski, SIMPLE2D and Conjugate Gradient are identical up to a 2-decimal place rounding for all nodes in the quarter coaxial cable. The results are however higher than those obtained from SOR. For instance, the SOR potential at (0.06, 0.04) is 10% lower than the 5.53V.

Potential approximation	Choleski	Conjugate Gradient	SIMPLE2D	SOR
(0.02, 0.02)	0.96	0.96	0.96	0.85
(0.02, 0.04)	1.97	1.97	1.97	1.73
(0.02, 0.06)	3.03	3.03	3.03	2.60
(0.02, 0.08)	3.96	3.96	3.96	3.13
(0.02, 0.10)	4.25	4.25	4.25	2.60
(0.04, 0.02)	1.86	1.86	1.86	1.66
(0.04, 0.04)	3.88	3.88	3.88	3.49
(0.04, 0.06)	6.18	6.18	6.18	5.54
(0.04, 0.08)	8.56	8.56	8.56	7.30
(0.04, 0.10)	9.09	9.09	9.09	5.54
(0.06, 0.02)	2.61	2.61	2.61	2.30
(0.06, 0.04)	5.53	5.53	5.53	5.02
(0.06, 0.06)	9.25	9.25	9.25	8.77
(0.06, 0.08)	15.0	15.0	15.0	15.0
(0.06, 0.10)	15.0	15.0	15.0	15.0
(0.08, 0.02)	3.04	3.04	3.04	2.53
(0.08, 0.04)	6.37	6.37	6.37	5.52
(0.08, 0.06)	10.29	10.29	10.29	9.51
(0.08, 0.08)	15.0	15.0	15.0	15.0
(0.08, 0.10)	15.0	15.0	15.0	15.0
(0.10, 0.02)	3.17	3.17	3.17	2.30
(0.10, 0.04)	6.61	6.61	6.61	5.02
(0.10, 0.06)	10.55	10.55	10.55	8.77
(0.10, 0.08)	15.0	15.0	15.0	15.0
(0.10, 0.10)	15.0	15.0	15.0	15.0

Table 1. Potential approximation using Choleski, Conjugate Gradient, SIMPLE2D and SOR

d) The capacitance per meter of the coaxial cable would be computed using $C = \frac{2W}{V^2}$

The total energy stored can be calculated using $W = \frac{1}{2}\iiint \vec{D}\cdot\vec{E}\,dv = \frac{1}{2}\iiint \vec{E}^2\cdot\varepsilon_o\,dv$
Since we are only interested in the capacitance per unit length, we will use
$W/m = \frac{1}{2}\iint \vec{E}^2\cdot\varepsilon_o\,dA$
Knowing that $\vec{E} = -\nabla V = (V_{x-1} - V_{x+1})\vec{x} + (V_{y-1} - V_{y+1})\vec{y}$

$$\frac{W}{m} = \frac{1}{2}\iint ((V_{x-1} - V_{x+1})\vec{x} + (V_{y-1} - V_{y+1}))\vec{y}^2\cdot\varepsilon_o\,dA$$
$$\frac{W}{m} = \frac{1}{2}((V_{x-1} - V_{x+1})^2 + (V_{y-1} - V_{y+1})^2)\cdot\varepsilon_o\cdot 0.0004$$

With the same idea as in Question 2 c., the total energy of the coaxial cable is four times that of the quarter cross section.

$$C/m = \frac{2\times 4W/m}{V^2} = \frac{4((V_{x-1} - V_{x+1})^2 + (V_{y-1} - V_{y+1})^2)\cdot\varepsilon_o\cdot 0.0004}{15^2}$$

Appendix

Please refer to matrix functions written for Assignment 1 in its appendix.

Matrix function:

- Vector addition

```
def vectorAddition(self, vector1, vector2):
    result = []
    for i in range(len(vector2)):
        result.append(vector1[i] + vector2[i])
    return result
```

- Vector subtraction (Vector1 – Vector2)

```
def vectorDifference(self, vector1, vector2):
    result = []
    for i in range(len(vector2)):
        result.append(vector1[i] - vector2[i])
    return result
```

- Vector Dot Product (A · B)

```
def vectorMultiplication(self, vector1, vector2):
    sum = 0
    for i in range(len(vector2)):
        sum+= vector1[i] * vector2[i]
    return sum
```

- Vector · Matrix

```
def vectorMatrixMultiplication(self, vector, matrix):
    result = []
    for i in range(len(matrix)):
        sum = 0
        for j in range(len(vector)):
            sum += vector[j] * matrix[i][j]

        result.append(sum)

    return result
```

- Scale Matrix (a · Matrix)

```
def scaleMatrix(self, scale, matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            matrix[i][j] *= scale
def scaleVector(self, scale, vector):
    return [scale * i for i in vector]
```

- Transpose Matrix (result = A^T)

```
def matrixTranspose(self, matrix):
    row_size = len(matrix)
    column_size = len(matrix[0])
    transpose = []
    for i in range(column_size):
        row = []
        for j in range(row_size):
            row.append(matrix[j][i])

        transpose.append(row)

    return transpose
```

First Order Class

Includes: Local S matrix generator, Disjoint S matrix generator, conjoint S matrix generator, Energy calculator, Capacitance calculator

```
class FirstOrder(object):
    def __init__(self):
        pass

    def Slocal(self, coord):
        Area = 1.0/2.0 * abs((coord[1][1]-coord[0][1])*(coord[2][0]-coord[0][0]) -
            (coord[1][0]-coord[0][0])*(coord[2][1]-coord[0][1]))
        S = []
        for i in range(3):
            Si = []
            for j in range(3):
                y = (coord[(i+1) % 3][1] - coord[(i+2) % 3][1])*(coord[(j+1) % 3][1] - coord[(j+2) % 3][1])
                x = (coord[(i+2) % 3][0] - coord[(i+1) % 3][0])*(coord[(j+2) % 3][0] - coord[(j+1) % 3][0])
                Sij = 1.0 / 4.0 / Area * (y + x)
                Si.append(Sij)
            S.append(Si)

        return S

    def Sdis(self, triangles):
        S = [[0 for i in range(len(triangles) * 3)] for i in range(len(triangles) * 3)]
        for i in range(len(triangles)):
            local = self.Slocal(triangles[i])
            for j in range(3):
                for k in range(3):
                    S[3 * i + j][3 * i + k] = local[j][k]
        return S


    def Sglobal(self, Sdis, C):
        CT = m.matrixTranspose(C);
        SdisC = m.matrixMultiplication(Sdis, C)
        return m.matrixMultiplication(CT, SdisC)

    def Energy(self, S, Ucon):
        SUcon = m.matrixVectorMultiplication(S, Ucon)
        E2 = m.vectorMultiplication(Ucon, SUcon)
        return E2 * 0.5

    def capacitance(self, energy):
        return 2 * energy / 15 / 15 * 8.85 * pow(10, -12)
```

- Five points difference matrix generator

(for the specific quarter coaxial cable)

```python
def matrixGenerator(self):
    A = [[0 for i in range(19)] for j in range(19)]
    for i in range(19):
        if i < 10:
            if(i < 8): A[i][i+5] = 1
            if i%5 != 0: A[i][i-1] = 1
            if (i+1)% 5 == 0: A[i][i-1] += 1
            else: A[i][i+1] = 1
            if i > 4: A[i][i-5] = 1

        elif i < 13:
            A[i][i-5] = 1
            A[i][i+3] = 1
            if i != 10: A[i][i-1] = 1
            if i != 12: A[i][i+1] = 1

        else:
            A[i][i-3] = 1
            if i < 16: A[i][i+3] = 1
            else: A[i][i-3] += 1
            if i%3 != 0: A[i][i+1] = 1
            if i%3 != 1: A[i][i-1] = 1
        A[i][i] = -4
    return A
```

- resulting b vector generator

(for the specific quarter coaxial cable)

```python
def bGenerator(self):
    b = []
    list = [8, 9, 12, 15, 18]
    for i in range(19):
        if i in list: b.append(-15)
        else: b.append(0)
    return b
```

Conjugate Gradient approximation method

```
def ConjugateGradient(self, A, b, residual):
    norm2 = []
    infinity_norm = []
    x = [0 for i in range(len(b))]
    r = self.residue(A, x, b)
    p = copy.deepcopy(r)
    residue = 1
    while(residue < len(A)):
        alpha = self.alpha(A, r, p)
        x = self.newGuess(x, alpha, p)
        r = self.residue(A, x, b)
        beta = self.beta(A, r, p)
        p = self.newOrientation(r, beta, p)
        maxRes = 0
        norm = 0
        for res in r:
            res = abs(res)
            residue += res
            norm += res*res
            if abs(res) > maxRes : maxRes = res
        residue = math.sqrt(norm)
        norm2.append(math.sqrt(norm))
        infinity_norm.append(maxRes)
    return (x, norm2, infinity_norm)
```

- first arrangement coefficient (alpha)

```
#pTr/pTAp
def alpha(self, A, r, p):
    pTr = m.vectorMultiplication(p, r)
    pTA = m.vectorMatrixMultiplication(p, A)
    pTAp = m.vectorMultiplication(pTA, p)
    return float(pTr) / float(pTAp)
```

- Residue vector r calculator

```
def residue(self, A, x, b):
    Ax = m.matrixVectorMultiplication(A, x)
    return m.vectorDifference(b, Ax)
```

- Next approximation (new guess) calculator

```
def newGuess(self, x, alpha, p):
    alphap = [alpha * i for i in p]
    return m.vectorAddition(x, alphap)
```

- New rearrangement coefficient calculator

```
#-pTAr/pTAp
def beta(self, A, r, p):
    Ar = m.matrixVectorMultiplication(A, r)
    pTAr = m.vectorMultiplication(p, Ar)
    Ap = m.matrixVectorMultiplication(A, p)
    pTAp = m.vectorMultiplication(p, Ap)
    return -1 * float(pTAr)/float(pTAp)
```

- new orientation vector generator

```
#newR + bp
def newOrientation(self, r, beta, p):
    bp = m.scaleVector(beta, p)
    return m.vectorAddition(r, bp)
```

- File reader to parse element (coordinates, triangle, fixed potentials) into lists

```python
def parseElementFile(self, filename):
    file = open(filename, "r")
    element = file.readlines()
    coord = []
    triangles = []
    fixedPotential = []
    C = []

    i = 0
    for line in element:
        temp = line.split("\n")[0].split("\t")
        if len(line) == 1:
            i+=1
        elif i == 0:
            coord.append(map(float, (temp[1], temp[2])))
        elif i == 1:
            temp = map(int, temp[0].split(" "))
            triangles.append((coord[temp[0] - 1], coord[temp[1] - 1], coord[temp[2] - 1]))
            for count in range(3):
                tempC = [0 for j in range(len(coord))]
                tempC[temp[count] - 1] = 1
                C.append(tempC)
        else:
            fixedPotential.append(map(float, temp));

    return triangles, C
```