Assignemnt 2
Numerical Methods

McGill University

Laith Mubaslat

November 14th, 2018

1) Figure 1 shows two first-order triangular finite elements used to solve the Laplace equation for electrostatic potential. Find a local **S**-matrix for each triangle and a global **S**-matrix for the mesh, which consists of just these two triangles. The local (disjoint) and global (conjoint) node-numberings are shown in Figure 1(a) and (b), respectively. Also, Figure 1(a) shows the (x, y)-coordinates of the element vertices in meters.

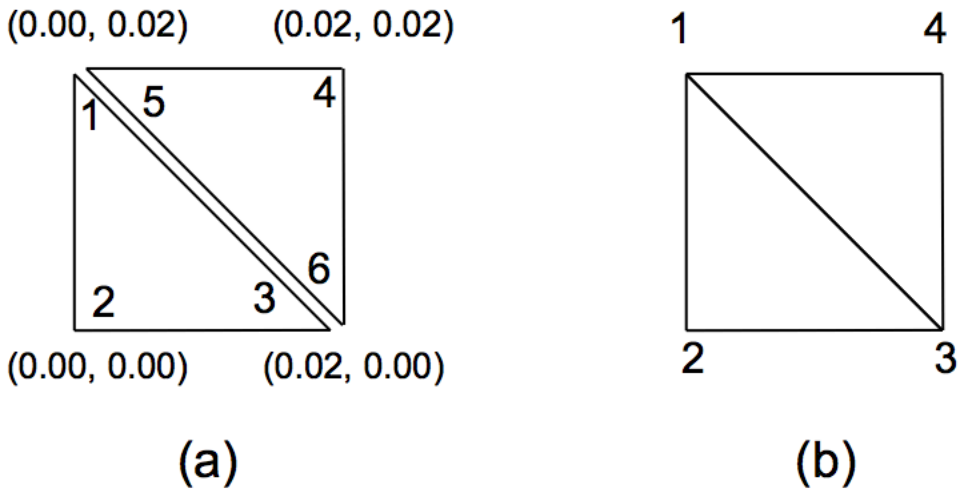


Figure 1

$$S_{11} = \int_{\Delta e} \nabla\alpha_1 \cdot \nabla\alpha_1 \, ds$$

~~$\nabla\alpha_1$~~

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

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

2

$$\nabla\alpha_1 \cdot \nabla\alpha_1 = \frac{1}{4A^2}\left[(y_2-y_3)^2 + (x_3-x_2)^2\right]$$

$$\int_{\nabla e} \nabla\alpha_1 \cdot \nabla\alpha_1 \, ds = \frac{1}{4A}\left[(y_2-y_3)^2 + (x_3-x_2)^2\right]$$

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

$$\int_{\nabla e} \nabla\alpha_2 \cdot \nabla\alpha_2 \, ds = \frac{1}{4A}\left[(y_3-y_1)^2 + (x_1-x_3)^2\right]$$

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

$$\int_{\nabla e} \nabla\alpha_3 \cdot \nabla\alpha_3 \, ds = \frac{1}{4A}\left[(y_1-y_2)^2 + (x_2-x_1)^2\right]$$

element 1 cont.

$$S_{12} = \frac{1}{4A}\left[(y_2 - y_3)(y_3 - y_1) + (x_3 - x_2)(x_1 - x_3)\right]$$

$$S_{21} = \frac{1}{4A}\left[(y_1 - y_3)(y_3 - y_2) + (x_3 - x_1)(x_2 - x_3)\right]$$

$$S_{13} = \frac{1}{4A}\left[(y_3 - y_2)(y_2 - y_1) + (x_2 - x_3)(x_1 - x_2)\right]$$

$$S_{31} = \frac{1}{4A}\left[(y_1 - y_2)(y_2 - y_3) + (x_2 - x_1)(x_3 - x_2)\right]$$

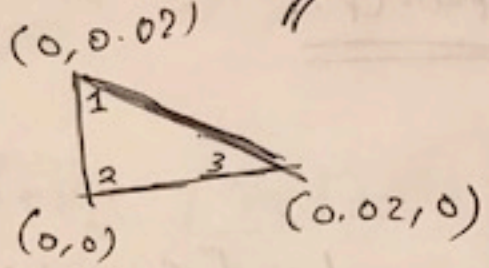
$$S_{23} = \frac{1}{4A}\left[(y_3 - y_1)(y_1 - y_2) + (x_1 - x_3)(x_2 - x_1)\right]$$

$$S_{32} = \frac{1}{4A}\left[(y_2 - y_1)(y_1 - y_3) + (x_1 - x_2)(x_3 - x_1)\right]$$

$$S_{11} = \frac{1}{4A}\left[(y_1 - y_2)(y_3 - y_1) + (x_3 - x_1)(x_1 - x_3)\right]$$

S_{22}

element 1



$X_1 = \emptyset \quad Y_1 = 0.02$

$X_2 = \emptyset \quad Y_2 = \emptyset$

$X_3 = 0.02 \quad Y_3 = \emptyset$

$$S_{12} = \frac{1}{4A}\left[(0-0)(Y_3 - Y_1) + (0.02)(-0.02)\right]$$

$$S_{21} = \frac{1}{4A}\left[(Y_1 - Y_3)(\emptyset) + (0.02)(-0.02)\right]$$

$$S_{13} = \frac{1}{4A}\left[(-0.02)(\emptyset) + (0.02)(\emptyset)\right]$$

$$S_{31} = \frac{1}{4A}\left[(0.02)(\emptyset) + (\emptyset)(0.02)\right]$$

$$S_{23} = \frac{1}{4A}\left[(-0.02)(0.02) + (-0.02)(\emptyset)\right]$$

$$S_{32} = \frac{1}{4A}\left[(\emptyset - 0.02)(0.02) + (\emptyset)(X_3 - X_1)\right]$$

$$S_{11} = \frac{1}{4A}\left[(\emptyset) + (0.02)^2\right]$$

$$S_{22} = \frac{1}{4A}\left[(-0.02)^2 + (-0.02)^2\right]$$

$$S_{33} = \frac{1}{4A}\left[(0.02)^2 + \emptyset\right]$$

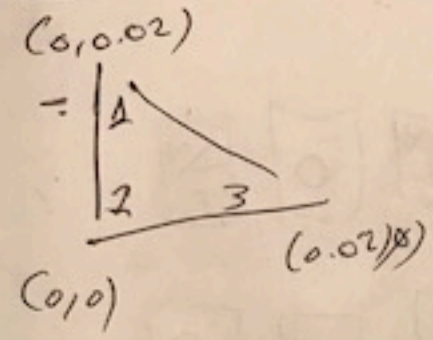
$$S_{11} = \frac{1}{4A}\left[(y_2 - y_3)^2 + (x_3 - x_2)^2\right]$$

$$S_{22} = \frac{1}{4A}\left[(y_3 - y_1)^2 + (x_1 - x_3)^2\right]$$

$$S_{33} = \frac{1}{4A}\left[(y_1 - y_2)^2 + (x_2 - x_1)^2\right]$$

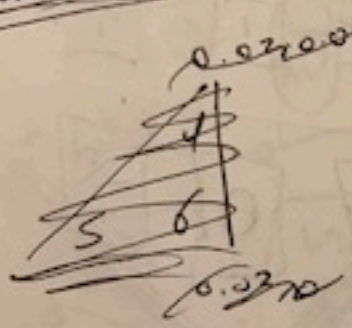
$A = 2 \times 10^{-4}$ 5 1 7

element 1

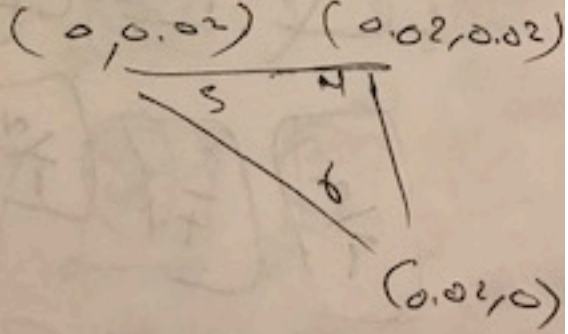


$$\begin{bmatrix} +1/2 & -1/2 & 0 \\ -1/2 & +1 & -1/2 \\ 0 & -1/2 & +1/2 \end{bmatrix}$$

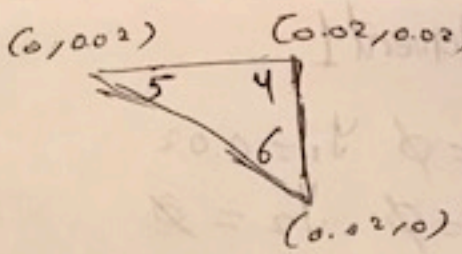
element 2



$$\begin{bmatrix} +1 & -1/2 & -1/2 \\ -1/2 & +1/2 & 0 \\ -1/2 & 0 & +1/2 \end{bmatrix}$$



element 2



$$S_{45} = \frac{1}{4A}\left[(0.02)(-0.02)+\emptyset\right]$$

$$S_{54} = \frac{1}{4A}\left[(0.02)(-0.02)+\emptyset\right]$$

$$S_{46} = \frac{1}{4A}\left[(-0.02)(\emptyset)+(-0.02)(0.02)\right]$$

$$S_{64} = \frac{1}{4A}\left[\emptyset+(-0.02)(0.02)\right]$$

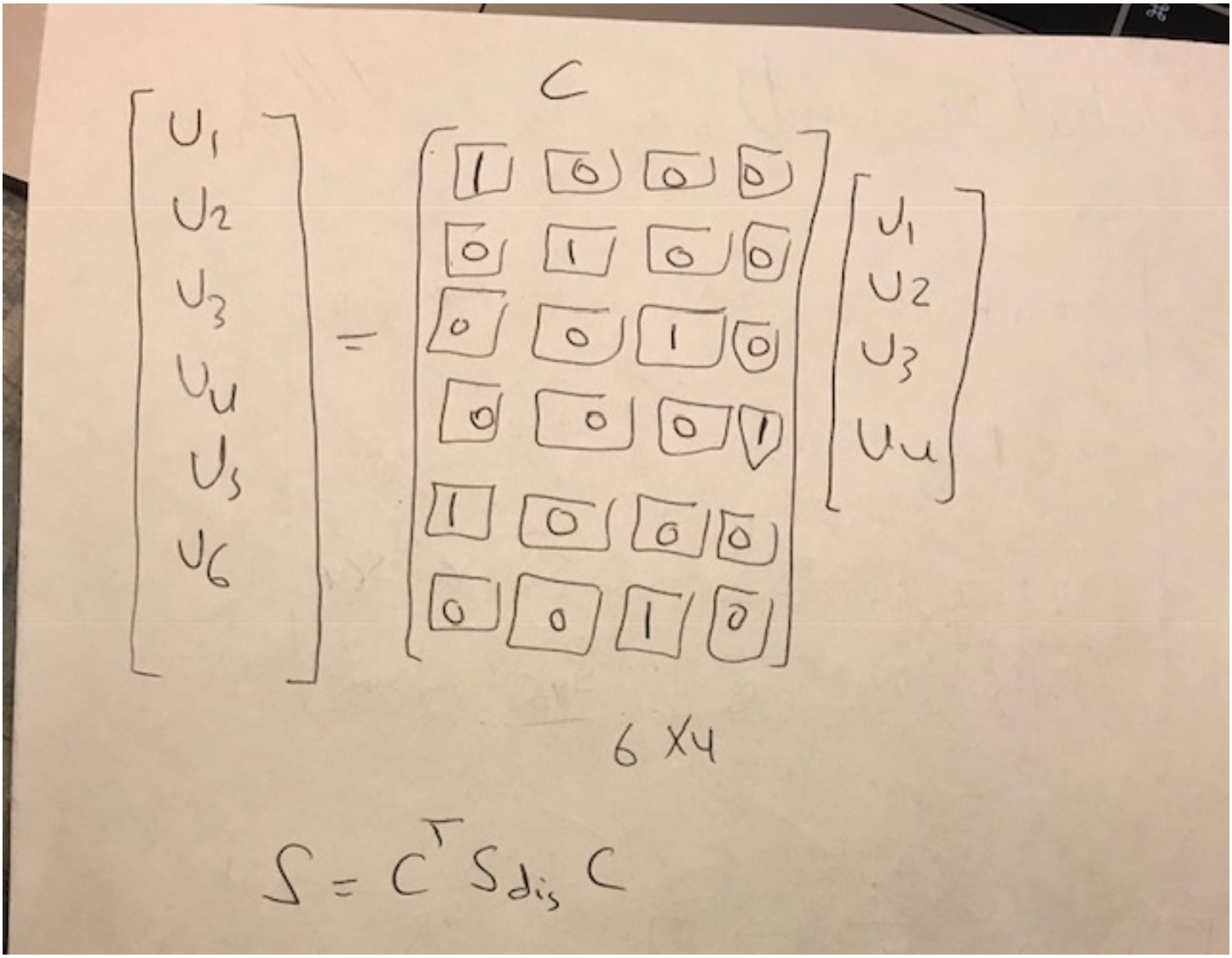
$$S_{56} = \frac{1}{4A}\left[(-0.02)(\emptyset)+\emptyset\right]$$

$$S_{65} = \frac{1}{4A}\left[\emptyset+\emptyset\right]$$

$$S_{44} = \frac{1}{4A}\left[(0.02)^2+(0.02)^2\right]$$

$$S_{55} = \frac{1}{4A}\left[(-0.02)^2+\emptyset\right]$$

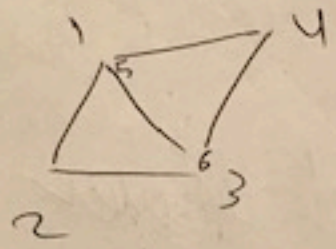
$$S_{66} = \frac{1}{4A}\left[(\emptyset)+(-0.02)^2\right]$$


C
$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}$
6 X 4
$\Sigma = C^T S_{dis} C$

Global (cojoint) without formula 7/11

$$\begin{bmatrix} S_{11}+S_{55} & S_{12} & S_{13}+S_{56} & S_{54} \\ S_{21} & S_{22} & S_{23} & \emptyset \\ S_{31}+S_{65} & S_{32} & S_{33}+S_{66} & S_{64} \\ S_{45} & \emptyset & S_{46} & S_{44} \end{bmatrix}$$

$$\begin{bmatrix} +1 & -1/2 & \emptyset & -1/2 \\ -1/2 & +1 & -1/2 & \emptyset \\ \emptyset & -1/2 & +1 & -1/2 \\ -1/2 & \emptyset & -1/2 & +1 \end{bmatrix}$$



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

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

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \end{bmatrix} = C \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \quad S = C^T S_{dis} C$$

2) Figure 2 shows the cross-section of an electrostatic problem with translational symmetry: a rectangular coaxial cable. The inner conductor is held at 110 volts and the outer conductor is grounded. (This is similar to the system considered in Question 3, Assignment 1.)

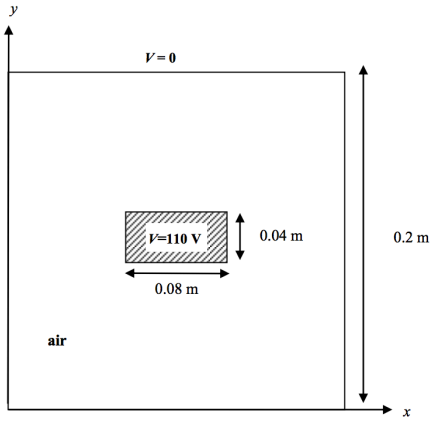


Figure 2.

(a) Use the two-element mesh shown in Figure 1(b) as a "building block" to construct a finite element mesh for one-quarter of the cross-section of the coaxial cable. Specify the mesh, including boundary conditions, in an input file following the format for the **SIMPLE2D** program as explained in the course notes. (Hint: Your mesh should consist of 46 elements.)

Answer:
The finite element mesh consisting of 46 elements can be shown in the following 3 figures which show the values for the 3 input .dat files file1.dat, file2.dat and file3.dat used to describe each node's xy coordinates, the elements of the finite element mesh and the boundary conditions for the problem.

ans =

34×3 table

Var1	Var2	Var3
1	0	0.1
2	0	0.08
3	0	0.06
4	0	0.04
5	0	0.02
6	0	0
7	0.02	0.1
8	0.02	0.08
9	0.02	0.06
10	0.02	0.04
11	0.02	0.02
12	0.02	0
13	0.04	0.1
14	0.04	0.08
15	0.04	0.06
16	0.04	0.04
17	0.04	0.02
18	0.04	0
19	0.06	0.1
20	0.06	0.08
21	0.06	0.06
22	0.06	0.04
23	0.06	0.02
24	0.06	0
25	0.08	0.08
26	0.08	0.06
27	0.08	0.04
28	0.08	0.02
29	0.08	0
30	0.1	0.08
31	0.1	0.06
32	0.1	0.04
33	0.1	0.02
34	0.1	0

Figure 3: 'File1.dat'

ans =

46×4 table

Var1	Var2	Var3	Var4
6	12	5	0
12	11	5	0
5	11	4	0
11	10	4	0
4	10	3	0
10	9	3	0
3	9	2	0
9	8	2	0
2	8	1	0
8	7	1	0
12	18	11	0
18	17	11	0
11	17	10	0
17	16	10	0
10	16	9	0
16	15	9	0
9	15	8	0
15	14	8	0
8	14	7	0
14	13	7	0
18	24	17	0
24	23	17	0
17	23	16	0
23	22	16	0
16	22	15	0
22	21	15	0
15	21	14	0
21	20	14	0
14	20	13	0
20	19	13	0
24	29	23	0
29	28	23	0
23	28	22	0
28	27	22	0
22	27	21	0
27	26	21	0
21	26	20	0
26	25	20	0
29	34	28	0
34	33	28	0
28	33	27	0
33	32	27	0
27	32	26	0
32	31	26	0
26	31	25	0
31	30	25	0

Figure 4: 'File2.dat'

ans =

15×2 table

Var1	Var2
1	0
2	0
3	0
4	0
5	0
6	0
12	0
18	0
24	0
29	0
34	0
19	110
20	110
25	110
30	110

Figure 5: 'File3.dat'

(b) Use the **SIMPLE2D** program with the mesh from part (a) to compute the electrostatic potential solution. Determine the potential at $(x,y) = (0.06, 0.04)$ from the data in the output file of the program.

Answer:
With an input of the three input files shown in part a the following command was used :

"Potential = SIMPLE2D_M('file1.dat','file2.dat','file3.dat')"

The following output voltages were obtained:

```
Potential =

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

Figure 6: Finite Element Method Simple2D program output

From the output shown above it can be clearly seen that the voltage at the point (0.06,0.04) is 40.5265V

(c) Compute the capacitance per unit length of the system using the solution obtained from **SIMPLE2D.**

Answer:

The energy at each element was determined. Following that the total energy of the region under study was calculated as the sum of the aforementioned values. That value was then multiplied by 4 in order to encompass the energy in the entire system, which is valid considering the region under study is symmetric to the remaining 3 parts. Following that the formula for energy storage in a capacitor (E=1/2 C V^2) was used in order to determine the capacitance taking into consideration the permittivity of free space and the relative permittivity of air (i.e. 1). The value obtained is in Farad per m considering how the region under study is only a cross section and does not account for the depth of the system. The value was found to be the following (F/m):

```
In [819]: Cpul
Out[819]: 3.522390890285829e-10
```

Figure 7: Capacitance per unit length

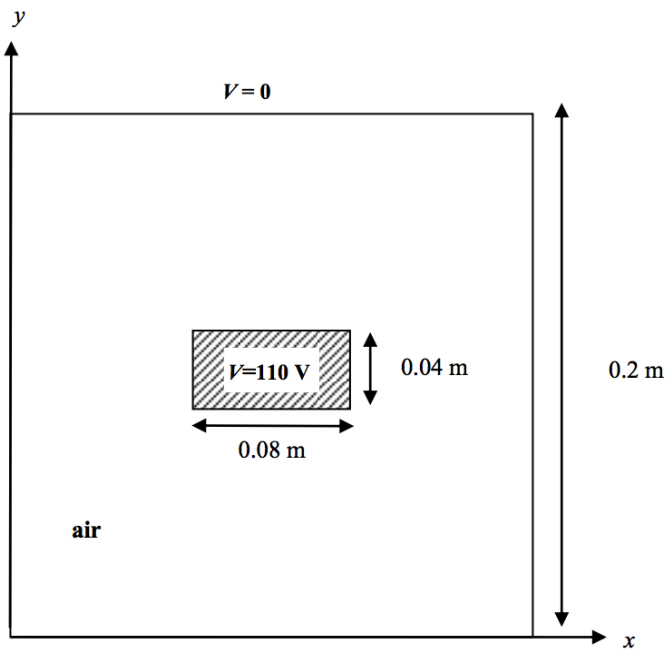


Figure 2.

3) Write a program implementing the conjugate gradient method (un-preconditioned). Solve the matrix equation corresponding to a finite difference node-spacing, $h = 0.02$m in x and y directions for the same one-quarter cross-section of the system shown in Figure 2 that considered in Question 2 above. Use a starting solution of zero. (Hint: The program you wrote for Question 3 of Assignment 1 may be useful for generating the matrix equation.)

The program can be seen in the appendix.

(a) Test your matrix using your Choleski decomposition program that you wrote for Question 1 of Assignment 1 to ensure that it is positive definite. If it is not, suggest how you could modify the matrix equation in order to use the conjugate gradient method for this problem.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	−4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	−4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	−4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	−4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	−4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	−1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	−4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
7	0	1	0	0	0	0	1	−4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
8	0	0	1	0	0	0	0	1	−4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	1	0	0	0	0	1	−4	1	0	0	0	0	1	0	0	0	0	0	0	0	0
10	0	0	0	0	1	0	0	0	0	1	−4	1	0	0	0	0	1	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	−1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	1	0	0	0	0	0	−4	1	0	0	0	0	1	0	0	0	0	0
13	0	0	0	0	0	0	0	1	0	0	0	0	1	−4	1	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	1	0	0	0	0	1	−4	1	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	−4	1	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	−4	1	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−1	0	1	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	−4	1	1	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	−4	0	1	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	−4	1	1	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	−4	0	1
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−1	0	0	0	1	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−1	0	0	0	1

Figure 8: Original Matrix A

```
In [782]: CholskeyL(A)
Cholskey Decomp. Not Possible; Matrix is not Positive Definite
```

Figure 9: Choleskey Test: Not PD

The matrix was found to be not positive definite as can be seen in figure 9 and as can be seen by inspecting figure 8. The solution used for this problem was multiplying the matrix by It's transpose which would thus produce a symmetric positive definite matrix. However, in order for the equation to remain valid the vector b had to be multiplied by the transpose of A as well which produced a new vector b used for solving the new matrix equation which both choleskey and CG methods are capable of solving. Figure 10 shows a part of the lower triangular matrix L produced by performing Choleskey Decomposition on the new PD matrix:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	4.243	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1	−1.886	3.930	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.236	−1.923	3.905	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.254	−1.923	4.029	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.256	−1.863	3.803	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	−1.052	0.945	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
6	−1.886	−0.396	−0.081	−0.014	−0.001	−0.001	3.909	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
7	0.471	−1.809	−0.407	−0.080	−0.012	−0.013	−2.011	3.505	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.509	−1.798	−0.394	−0.072	−0.080	0.269	−2.084	3.453	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	0.000	0.000	0.512	−1.741	−0.362	−0.402	0.004	0.305	−2.082	3.604	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
10	0.000	0.000	0.000	0.496	−1.860	−1.012	0.001	0.002	0.285	−2.115	3.117	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
11	0.000	0.000	0.000	0.000	0.263	0.293	0.000	0.002	0.014	0.067	−0.988	0.930	0.000	0.000	0.000	0.000	0.000	0.000	0.000
12	0.236	0.113	0.041	0.013	0.003	0.004	−1.920	−0.499	−0.145	−0.039	−0.011	−0.007	3.869	0.000	0.000	0.000	0.000	0.000	0.000
13	0.000	0.254	0.125	0.044	0.013	0.014	0.540	−1.825	−0.531	−0.147	−0.045	−0.033	−2.066	3.423	0.000	0.000	0.000	0.000	0.000
14	0.000	0.000	0.256	0.122	0.043	0.047	0.006	0.607	−1.802	−0.505	−0.157	−0.133	0.263	−2.171	3.197	0.000	0.000	0.000	0.000
15	0.000	0.000	0.000	0.248	0.122	0.135	0.001	0.007	0.617	−1.717	−0.503	−0.497	0.004	0.305	−2.279	3.285	0.000	0.000	0.000
16	0.000	0.000	0.000	0.000	0.263	0.293	0.000	0.002	0.014	0.622	−1.894	−1.147	−0.001	−0.009	0.264	−2.414	2.580	0.000	0.000
17	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.321	0.341	0.002	0.008	0.035	0.124	−1.051	0.813	0.000
18	0.000	0.000	0.000	0.000	0.000	0.000	0.256	0.147	0.069	0.027	0.012	0.009	−1.919	−0.524	−0.182	−0.071	−0.045	−0.039	3.989
19	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.285	0.172	0.075	0.035	0.029	0.561	−1.815	−0.909	−0.446	−0.312	−0.303	−2.044
20	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.258	0.156	0.085	0.044	0.033	0.030	−1.856
21	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.292	0.198	0.111	0.084	0.080	0.553
22	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Figure 10: Choleskey Test: Not PD

(b) Once you have modified the problem, if necessary, so that the matrix is positive definite, solve the matrix equation first using the Choleski decomposition program from Assignment 1, and then the conjugate gradient program written for this assignment.

Answer:

Figure 11 shows the node placement for both problems and Figures 12 and 13 show the output for both the Choleski and the CG methods for solving the modified problem.

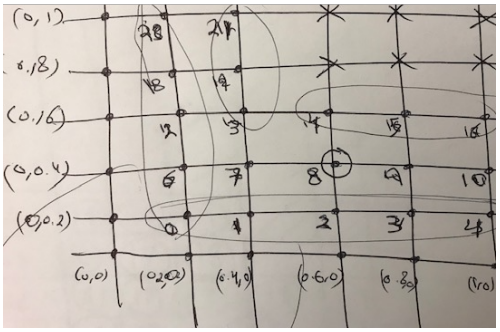


Figure 11: Node Placement

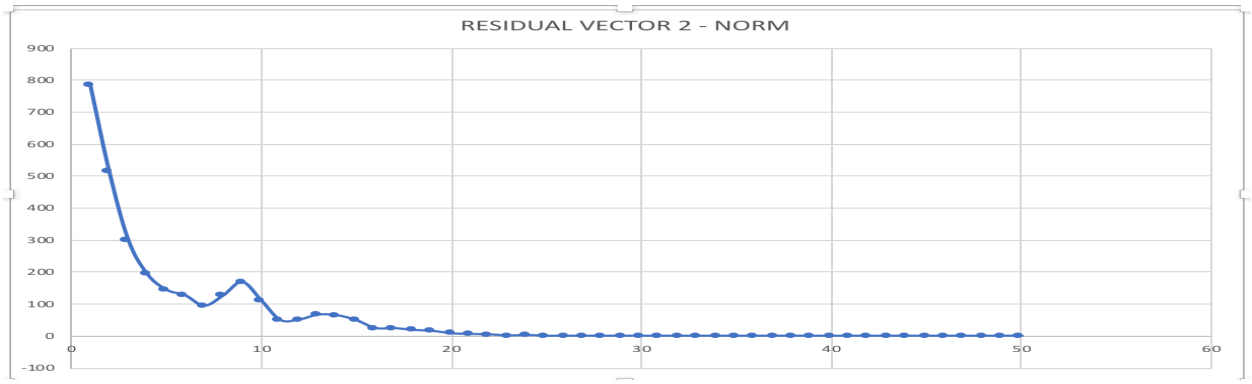
ıdɛ ▲	Type	Size	
0	list	1	[7.018554351943977]
1	list	1	[13.65192901361166]
2	list	1	[19.110684345944417]
3	list	1	[22.26430575894032]
4	list	1	[23.25686747625886]
5	list	1	[22.264305758940328]
6	list	1	[14.422288394164257]
7	list	1	[28.478477356558255]
8	list	1	[40.526502611225666]
9	list	1	[46.68967121355796]
10	list	1	[48.49885838715477]
11	list	1	[46.68967121355799]
12	list	1	[22.192121868154793]
13	list	1	[45.313189407231434]
14	list	1	[67.82717752884204]
15	list	1	[75.46901809691107]
16	list	1	[77.35922364524424]
17	list	1	[75.4690180969111]
18	list	1	[29.033009671223475]
19	list	1	[62.75498087537056]
20	list	1	[31.184935941368565]
21	list	1	[66.67372442302741]
22	list	1	[29.033009671223436]
23	list	1	[62.75498087537055]

Figure 12: Choleski Output

ıdɛ ▲	Type	Size	
0	list	1	[7.018554351943977]
1	list	1	[13.651929013611657]
2	list	1	[19.110684345944403]
3	list	1	[22.264305758940292]
4	list	1	[23.256867476258826]
5	list	1	[22.26430575894027]
6	list	1	[14.422288394164253]
7	list	1	[28.478477356558248]
8	list	1	[40.526502611225666]
9	list	1	[46.68967121355795]
10	list	1	[48.49885838715475]
11	list	1	[46.68967121355797]
12	list	1	[22.192121868154793]
13	list	1	[45.31318940723141]
14	list	1	[67.82717752884204]
15	list	1	[75.46901809691106]
16	list	1	[77.35922364524423]
17	list	1	[75.46901809691109]
18	list	1	[29.033009671223493]
19	list	1	[62.75498087537058]
20	list	1	[31.184935941368607]
21	list	1	[66.67372442302744]
22	list	1	[29.033009671223503]
23	list	1	[62.75498087537056]

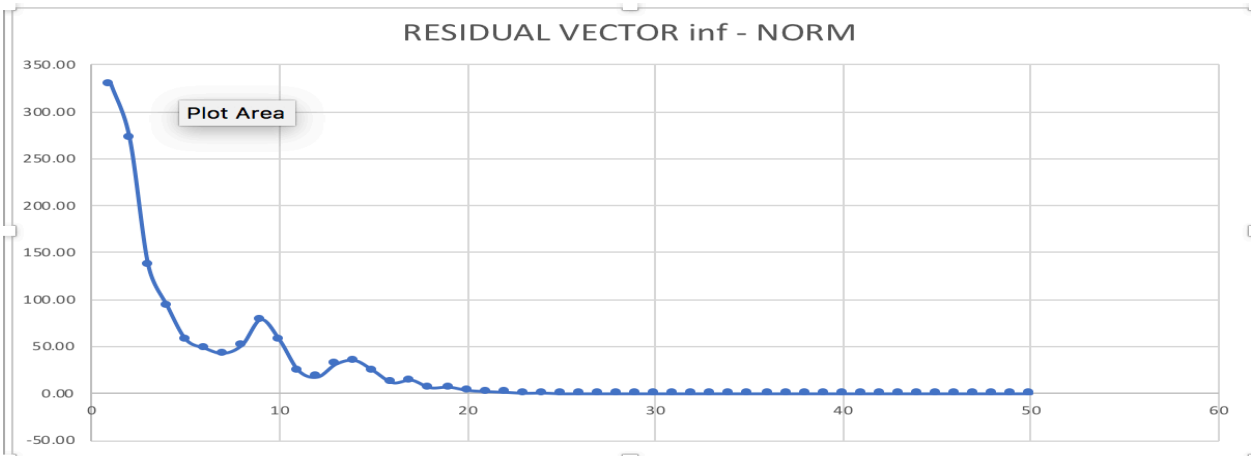
Figure 13: CG Output

(c) Plot a graph of the infinity norm and the 2-norm of the residual vector versus the number of iterations for the conjugate program.



Graph 1: residual vector 2-norm

Number of iterations



Number of iterations

Graph 2: residual vector inf-norm

It can be clearly seen how both the 2 and infinity norms for the residual vector are converging to a low value with a decreasing ripple the higher the number of iteration gets.

(d) What is the potential at $(x,y) = (0.06, 0.04)$, using the Choleski decomposition and the conjugate gradient programs, and how do they compare with the value you computed in Question 2(b) above. How do they compare with the value at the same (x,y) location and for the same node spacing that you computed in Assignment 1 using SOR.

Method	SOR	Choleskey	CG
Potential at point (0.06, 0.04)	40.5265	40.5265	40.5265

Table 1: Comparison between voltage value at point (0.06,0.04) obtained by SOR,Choleskey and CG

Further comparing these values to the value of 40.5265 v obtained in part b we can see how they are completely identical as far as the result is concerned which does make sense considering all of these methods tackles the same problem with the same characteristics and specifications (e.g a spacing h=0.02)

(e) Suggest how you could compute the capacitance per unit length of the system from the finite difference solution.

The same program of part c could be used to compute the solution by matching up the nodes in the finite difference solution to their counterpart in the finite element solution which is possible considering how the geometry of the triangular elements used allows the construction of the elements used for the FD solution. Following that the voltages computed by FD are mapped and then used in the program of part c to compute the capacitance per unit length in the same procedure. The code is provided in the appendix.

	0	1	2	3	4	5
0	0.000	0.000	0.000	0.000	0.000	0.000
1	0.000	7.019	14.422	22.192	29.033	31.185
2	0.000	13.652	28.478	45.313	62.755	66.674
3	0.000	19.111	40.526	67.827	110.000	110.000
4	0.000	22.264	46.690	75.469	110.000	110.000
5	0.000	23.257	48.499	77.359	110.000	110.000

Figure 14: FD output and node allocation

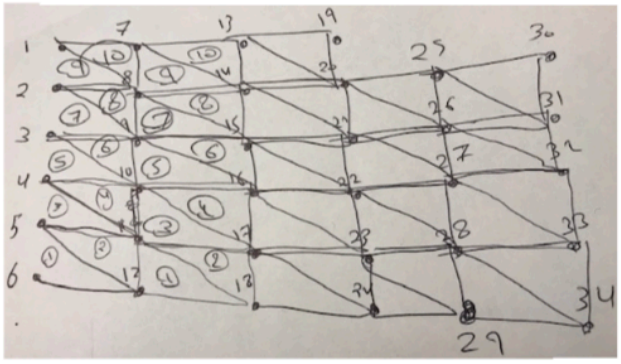


Figure 15: FE node allocation

```
In [817]: CpulFD
Out[817]: 3.522390373593319e-10
```

Figure 16: FD Capacitance per unit length

APPENDIX

```
##### BasicFunctions
from math import sqrt


def VerticalVector(HorizontalVector,dimension):
    n=dimension
    V=[0]*(n*n)
    for i in range (n*n):
        if ((i%n)==0): V[i]=HorizontalVector[i//n]

    VV=Matrix(V,n,1)

    return VV
#function to create a matrix
def Matrix(dataList,rows,columns ):
    Matrix = []
    counter=0
    for i in range(rows):
        rowList = []
        for j in range(columns):
            # you need to increment through dataList here, like this:
            rowList.append(dataList[counter])
            counter=counter+1
        Matrix.append(rowList)

    return Matrix
def SqMatrix(dataList,n):
    Matrix = []
    for i in range(n):
        rows = []
        for j in range(n):
            # you need to increment through dataList here, like this:
            rows.append(dataList[n * i + j])
        Matrix.append(rows)

    return Matrix
def rowdimension(s):
    return 1 + rowdimension(s[1:]) if s else 0
def columndimension(s):
    x=s[0]
    return  rowdimension(x)
#searches for largest value in square matrix of size n returns index of t
def TransposeSquare(rows,columns,M):
    MTinit=[0]*(n*n)
    MT=Matrix(MTinit,rows,columns)
    for i in range (rows):
        for j in range (columns):

                MT[i][j]=M[j][i]
    return MT
def Transpose(rows,columns,M):
```

```
    MTinit=[0]*(rows*columns)
    MT=Matrix(MTinit,columns,rows)
    for j in range (columns):
        for i in range (rows):

                MT[j][i]=M[i][j]
    return MT
def Multiplication (M1,M2):
    r1=rowdimension(M1)
    c1=columndimension(M1)

    r2=rowdimension(M2)
    c2=columndimension(M2)
    Productzeros=[0]*(r1*c2)
    Product=Matrix(Productzeros,r1,c2)
    if (c1!=r2):
        print("Matrix Multiplication is Invalid: Column count of first do
        return
    for i in range (r1):
        for j in range (c2):
            for k in range (c1):
                Product[i][j] = Product [i][j]+ M1[i][k]*M2[k][j]
    return Product
def MatrixVectorMultiplication(M1,V): #MV1=V2
    r1=rowdimension(M1)
    c1=columndimension(M1)
    r2=rowdimension(V)
    c2=1
    Productzeros=[0]*(c1*r2)
    Product=VerticalVector(Productzeros,r1*c2)
    if (c1!=r2):
        print("Matrix Multiplication is Invalid: Column count of first do
        return
    for i in range (r1):
            for k in range (c1):
                    Product[i][0] = Product [i][0]+ M1[i][k]*V[k]
    return Product
```

q2c

```
#Energy

#### export elements/coordinates/ potential

#elements

ElementsVector= [0,0,0,

                 6, 12, 5,12, 11, 5, 5, 11, 4,11, 10, 4,11, 10, 4,4, 10,
                 12, 18, 11, 18, 17, 11, 11, 17, 10,  17, 16, 10,  10, 16
                 18, 24, 17, 24, 23, 17,  17, 23, 16, 23, 22, 16, 16, 22,
                 24, 29, 23,  29, 28, 23,  23, 28, 22,  28, 27, 22, 22, 2
                 29, 34, 28,  34, 33, 28,  28, 33, 27,  33, 32, 27,  27,



                 ]

Elements= Matrix(ElementsVector,47,3)


#points
XYCoordinatesperpointvector=[0,0,0, 0.1,  0, 0.08 , 0, 0.06 , 0, 0.04 , 0

 0.02, 0.1 , 0.02, 0.08 ,  0.02, 0.06 , 0.02, 0.04 , 0.02, 0.02 ,  0.02,

 0.04, 0.1 , 0.04, 0.08 ,  0.04, 0.06 , 0.04, 0.04 , 0.04, 0.02 ,  0.04,

 0.06, 0.1 , 0.06, 0.08 ,  0.06, 0.06 , 0.06, 0.04 , 0.06, 0.02 ,  0.06,


 0.08, 0.08 , 0.08, 0.06 ,  0.08, 0.04 , 0.08, 0.02 , 0.08, 0 ,


 0.1 ,0.08 , 0.1 ,0.06 ,  0.1 ,0.04 , 0.1, 0.02 , 0.1, 0 ,

   ]

XYCor=Matrix(XYCoordinatesperpointvector,35,2)


#potential
potentialvector=[0,

0,
0,
0,
0,
0,
0,
31.1849359413686,
```

```
29.0330096712235,
22.1921218681548,
14.4222883941643,
7.01855435194398,
0,
66.6737244230275,
62.7549808753706,
45.3131894072314,
28.4784773565582,
13.6519290136117,
0,
110,
110,
67.8271775288420,
40.5265026112257,
19.1106843459444,
0,
110,
75.4690180969110,
46.6896712135579,
22.2643057589403,
0,
110,
77.3592236452442,
48.4988583871547,
23.2568674762588,
0]
```

```
#### first calculate S per element (C12)

A= 0.01*0.02
term=1/(4*A)
epsilon0= 8.854187817 * (10**(-12 ))

Wsum=0
for i in range (1,47):
    element=Elements[i]
    point1=element[0]
```

```
point2=element[1]
point3=element[2]


V1=potentialvector[point1]
V2=potentialvector[point2]
V3=potentialvector[point3]

point1coord=XYCor[point1]
point2coord=XYCor[point2]
point3coord=XYCor[point3]

x1=point1coord[0]
y1=point1coord[1]

x2=point2coord[0]
y2=point2coord[1]

x3=point3coord[0]
y3=point3coord[1]

S11= term * (  (y2-y3)**2        +       (x3-x2)**2        )

S22= term * (  (y2-y1)**2        +       (x1-x2)**2        )


S33= term * (  (y1-y2)**2        +       (x2-x3)**2        )

S12= term * (  (y2-y3) * (y3-y1)     +       (x3-x2) * (x1-x2)
S21=S12

S13= term * (  (y2-y3) * (y1-y2)     +       (x3-x2) * (x2-x1)


S31=S13


S23= term * (  (y3-y1) * (y1-y2)     +       (x1-x3) * (x2-x1)


S32=S23


Svector=[S11,S12,S13,S21,S22,S23,S13,S23,S33]



Se=Matrix(Svector,3,3)


Vvector=[V1,V2,V3]
```

```
    V=Matrix(Vvector,3,1)
    VT=Matrix(Vvector,1,3)

    firstmult=Multiplication(VT,Se)
    secondmult=Multiplication(firstmult,V)

    We= (1/2)*  epsilon0* secondmult[0][0]

    Wsum=Wsum+We



Wtotal=Wsum*4 #due to symmetry
V=110
Cpul= 2 * Wtotal / (V*V) #considering this Wtotal is per unit length
```

#*

```
### calculate W(e)

### sum W(e) > W total

### 1/2 C(V)^2
```

q3a

```
numberofnodes=24

zeros=[0]*(numberofnodes*numberofnodes)

A=Matrix(zeros,numberofnodes,numberofnodes)

for i in range (24):
    A[i][i]=-4

for i in range (22,24):
     A[i][i]=1


A[22][18]=-1
A[23][19]=-1


i=5
A[i][i]=1
A[i][3]=-1
i=11
A[i][i]=1
A[i][9]=-1
i=17
A[i][i]=1
A[i][15]=-1




A[0][1]=1
A[0][6]=1


for i in range (1,5):
    A[i][i+1]=1
    A[i][i-1]=1
    A[i][i+6]=1

i=6
A[i][i+1]=1
A[i][i+6]=1
A[i][i-6]=1

i=12
A[i][i+1]=1
A[i][i+6]=1
A[i][i-6]=1


i=18
```

```
A[i][i+2]=1
A[i][i+1]=1
A[i][i-6]=1

i=20
A[i][i+1]=1
A[i][i+2]=1
A[i][i-2]=1


for i in range (7,11):
    A[i][i+1]=1
    A[i][i+6]=1
    A[i][i-6]=1
    A[i][i-1]=1

i=13
A[i][i+1]=1
A[i][i+6]=1
A[i][i-6]=1
A[i][i-1]=1


for i in range (14,17):
    A[i][i+1]=1

    A[i][i-6]=1
    A[i][i-1]=1


i=19
A[i][i+2]=1

A[i][i-6]=1
A[i][i-1]=1


i=21
A[i][i+2]=1

A[i][i-1]=1
A[i][i-2]=1




i=13
A[i][i+1]=1
A[i][i+6]=1
```

```
A[i][i-6]=1
A[i][i-1]=1


Verify=np.array(A)


##########################################################################
##########################################################################

b=[0]*numberofnodes


for i in range(14,17):
    b[i]=-110

i=19
b[i]=-110
i=21
b[i]=-110
b=Matrix(b,numberofnodes,1)
```

```
def CholskeyL(A):

    #1st step: Check if A is square

    if(rowdimension(A)!=columndimension(A)):
        print ("matrix not square; Cholskey Decomp. Not Possible")
        return




    n=rowdimension(A)
    zeros=[0] * (n*n)
    L=Matrix(zeros,n,n)
    lam=Matrix(zeros,n,n)







    for i in range (n):
        for j in range (n):
            lam[i][j]=A[i][j]


    for j in range (n):
        if (lam[j][j]<=0):
            print ("Cholskey Decomp. Not Possible; Matrix is not Positive
            return


    for i in range (n):
        for j in range (n):
            if (A[i][j]!= A[j][i]):
                print ("matrix not symmetric; Cholskey Decomp. Not Possib
                return

    for j in range (n):
        L[j][j]=sqrt(A[j][j])

        for i in range (j,n) :
```

```
            L[i][j]=A[i][j]/L[j][j]

            for k in range (j,i+1): ########? +!+!
                A[i][k]=A[i][k]-L[i][j]*L[k][j]




    return L
```

```
L=CholskeyL(A)
```

```
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
# problem to be solved

AT=Transpose(numberofnodes,numberofnodes,A)
APD=Multiplication(AT,A)
bnew=Multiplication(AT,b )

A=APD
b=bnew
```

q3b

```
L=CholskeyL(A)

y=Lyb(L,b)


xx=LTxY(L,y)
```

```
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
zeros = [0]*24
x0=Matrix(zeros,24,1)


placeholder=Matrix(zeros,24,1)


xold=Matrix(zeros,24,1)
Pold=Matrix(zeros,24,1)
rold=Matrix(zeros,24,1)


Pnew=Matrix(zeros,24,1)
rnew=Matrix(zeros,24,1)
xnew=Matrix(zeros,24,1)


##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
# r0 = b-Ax0 (intial r0; initial p0=r0; r0=rold; p0=Pold)
term2= Multiplication(A,xold)
term1= b
for i in range (24):
    rold[i][0]=term1[i][0]-term2[i][0]
    Pold[i][0]=rold[i][0]




#iterative process
k=200  #number of iterations


for i in range (k):

    PoldTranspose= Transpose(24,1,Pold)

    term1=Multiplication(PoldTranspose,rold)
    term2prov=Multiplication(A,Pold)
    term2=Multiplication(PoldTranspose,term2prov)
```

```
    alpha= term1[0][0] / term2[0][0]

    for i in range (24):
        xnew[i][0]=xold[i][0]+alpha*Pold[i][0]



    term3=Multiplication(A,xnew)

    for i in range (24):
        rnew[i][0]= b[i][0] - term3[i][0]



    term4prov=Multiplication(A,rnew)
    term4=Multiplication(PoldTranspose,term4prov)

    term5prov=Multiplication(A,Pold)
    term5=Multiplication(PoldTranspose,term5prov)

    Beta= - (term4[0][0]) / (term5[0][0])

    for i in range (24):
        Pnew[i][0]=rnew[i][0]+Beta*Pold[i][0]


    for i in range (24):
        xold[i][0]=xnew[i][0]
        Pold[i][0]=Pnew[i][0]
        rold[i][0]=rnew[i][0]




#visualizing the answer


visual=Matrix(xold,6,6)
visualx=np.array(visual)
```

q3c

```
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
def norminf (x,n):
    maxabs = 0

    for i in range (n):
        if (x[i][0]<0) :
            a=-x[i][0]
        else:
            a=x[i][0]

        if (a>maxabs):
            maxabs=a


    return maxabs




def norm2 (x,n):
    sqsum=0
    for i in range (n):
        sqsum=x[i][0]*x[i][0]+sqsum

    root=sqrt(sqsum)

    return root
```

```
zeros = [0]*24
x0=Matrix(zeros,24,1)


placeholder=Matrix(zeros,24,1)


xold=Matrix(zeros,24,1)
Pold=Matrix(zeros,24,1)
rold=Matrix(zeros,24,1)


Pnew=Matrix(zeros,24,1)
rnew=Matrix(zeros,24,1)
xnew=Matrix(zeros,24,1)


##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
# r0 = b-Ax0 (intial r0; initial p0=r0; r0=rold; p0=Pold)
term2= Multiplication(A,xold)
term1= b
for i in range (24):
    rold[i][0]=term1[i][0]-term2[i][0]
    Pold[i][0]=rold[i][0]




#iterative process
k=200  #number of iterations

rinfnorm=[0]*k
r2norm=[0]*k

rinfnorm=Matrix(rinfnorm,k,1)
r2norm=Matrix(r2norm,k,1)

n=24 #number of elements in r vector

for i in range (k):
    m=norminf(rold,n)
    mm=norm2(rold,n)
```

```
rinfnorm[i][0]=m
r2norm[i][0]=mm

PoldTranspose= Transpose(24,1,Pold)

term1=Multiplication(PoldTranspose,rold)
term2prov=Multiplication(A,Pold)
term2=Multiplication(PoldTranspose,term2prov)

alpha= term1[0][0] / term2[0][0]

for i in range (24):
    xnew[i][0]=xold[i][0]+alpha*Pold[i][0]



term3=Multiplication(A,xnew)

for i in range (24):
    rnew[i][0]= b[i][0] - term3[i][0]



term4prov=Multiplication(A,rnew)
term4=Multiplication(PoldTranspose,term4prov)

term5prov=Multiplication(A,Pold)
term5=Multiplication(PoldTranspose,term5prov)

Beta= - (term4[0][0]) / (term5[0][0])

for i in range (24):
    Pnew[i][0]=rnew[i][0]+Beta*Pold[i][0]









for i in range (24):
    xold[i][0]=xnew[i][0]
    Pold[i][0]=Pnew[i][0]
    rold[i][0]=rnew[i][0]
```

```
#visualizing the answer


visual=Matrix(xold,6,6)
visualx=np.array(visual)
```

q3e

```
#Energy

#### export elements/coordinates/ potential

#elements

ElementsVector= [0,0,0,

                 6, 12, 5,12, 11, 5, 5, 11, 4,11, 10, 4,11, 10, 4,4, 10,
                 12, 18, 11, 18, 17, 11, 11, 17, 10,  17, 16, 10,  10, 16
                 18, 24, 17, 24, 23, 17,  17, 23, 16, 23, 22, 16, 16, 22,
                 24, 29, 23,  29, 28, 23,  23, 28, 22,  28, 27, 22, 22, 2
                 29, 34, 28,  34, 33, 28,  28, 33, 27,  33, 32, 27,  27,



                 ]

Elements= Matrix(ElementsVector,47,3)


#points
XYCoordinatesperpointvector=[0,0,0, 0.1,  0, 0.08 , 0, 0.06 , 0, 0.04 , 0

 0.02, 0.1 , 0.02, 0.08 ,  0.02, 0.06 , 0.02, 0.04 , 0.02, 0.02 ,  0.02,

 0.04, 0.1 , 0.04, 0.08 ,  0.04, 0.06 , 0.04, 0.04 , 0.04, 0.02 ,  0.04,

 0.06, 0.1 , 0.06, 0.08 ,  0.06, 0.06 , 0.06, 0.04 , 0.06, 0.02 ,  0.06,


 0.08, 0.08 , 0.08, 0.06 ,  0.08, 0.04 , 0.08, 0.02 , 0.08, 0 ,


 0.1 ,0.08 , 0.1 ,0.06 ,  0.1 ,0.04 , 0.1, 0.02 , 0.1, 0 ,

   ]

XYCor=Matrix(XYCoordinatesperpointvector,35,2)


#potential / Z is the node potential output of FD algo.
potentialvector=[5]*35

potentialvector[1]=Z[0][5]
potentialvector[2]=Z[0][4]
potentialvector[3]=Z[0][3]
potentialvector[4]=Z[0][2]
potentialvector[5]=Z[0][1]
potentialvector[6]=Z[0][0]
```

```
potentialvector[7]=Z[1][5]
potentialvector[8]=Z[1][4]
potentialvector[9]=Z[1][3]
potentialvector[10]=Z[1][2]
potentialvector[11]=Z[1][1]
potentialvector[12]=Z[1][0]



potentialvector[13]=Z[2][5]
potentialvector[14]=Z[2][4]
potentialvector[15]=Z[2][3]
potentialvector[16]=Z[2][2]
potentialvector[17]=Z[2][1]
potentialvector[18]=Z[2][0]



potentialvector[19]=Z[3][5]
potentialvector[20]=Z[3][4]
potentialvector[21]=Z[3][3]
potentialvector[22]=Z[3][2]
potentialvector[23]=Z[3][1]
potentialvector[24]=Z[3][0]



potentialvector[25]=Z[4][4]
potentialvector[26]=Z[4][3]
potentialvector[27]=Z[4][2]
potentialvector[28]=Z[4][1]
potentialvector[29]=Z[4][0]



potentialvector[30]=Z[5][4]
potentialvector[31]=Z[5][3]
potentialvector[32]=Z[5][2]
potentialvector[33]=Z[5][1]
potentialvector[34]=Z[5][0]
```

```
#### first calculate S per element (C12)

A= 0.01*0.02
term=1/(4*A)
epsilon0= 8.854187817 * (10**(-12 ))

Wsum=0
for i in range (1,47):
    element=Elements[i]
    point1=element[0]
    point2=element[1]
    point3=element[2]


    V1=potentialvector[point1]
    V2=potentialvector[point2]
    V3=potentialvector[point3]

    point1coord=XYCor[point1]
    point2coord=XYCor[point2]
    point3coord=XYCor[point3]

    x1=point1coord[0]
    y1=point1coord[1]

    x2=point2coord[0]
    y2=point2coord[1]

    x3=point3coord[0]
    y3=point3coord[1]

    S11= term * (  (y2-y3)**2        +       (x3-x2)**2        )

    S22= term * (  (y3-y1)**2        +       (x1-x3)**2        )


    S33= term * (  (y1-y2)**2        +       (x2-x3)**2        )

    S12= term * (  (y2-y3) * (y3-y1)     +       (x3-x2) * (x1-x2)
    S21=S12

    S13= term * (  (y2-y3) * (y1-y2)     +       (x3-x2) * (x2-x1)
```

```
    S31=S13

    S23= term * (  (y3-y1) * (y1-y2)     +       (x1-x3) * (x2-x1)

    S32=S23

    Svector=[S11,S12,S13,S21,S22,S23,S13,S23,S33]


    Se=Matrix(Svector,3,3)

    Vvector=[V1,V2,V3]
    V=Matrix(Vvector,3,1)
    VT=Matrix(Vvector,1,3)

    firstmult=Multiplication(VT,Se)
    secondmult=Multiplication(firstmult,V)


    We= (1/2)*  epsilon0* secondmult[0][0]

    Wsum=Wsum+We



Wtotal=Wsum*4 #due to symmetry
V=110

CpulFD= 2 * Wtotal / (V*V) #considering this Wtotal is per unit length
```

#*

```
### calculate W(e)

### sum W(e) > W total

### 1/2 C(V)^2
```