

ECSE543 ASSIGNMENT 2 REPORT

Yi Zhu

260716006

First and partial of the second question of this assignment are written by hand and all programs are developed in Python language. Assignment has been discussed with Yuanzhe Gong.

Question 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.

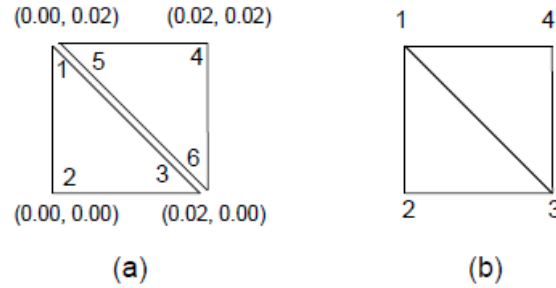


Figure 1

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

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

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

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

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

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

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

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

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

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

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

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

Since $S_{ij} = \nabla \alpha_i * \nabla \alpha_j * A$ where for example:

$$S^{(e)}_{12} = \frac{1}{4A} [(y_2 - y_3)(y_3 - y_1) + (x_3 - x_2)(x_1 - x_3)]$$

$$\text{And } A = \frac{0.02*0.02}{2} = 0.0002$$

$$S^{(1)} = \begin{bmatrix} 0.5 & -0.5 & 0 \\ -0.5 & 1 & -0.5 \\ 0 & -0.5 & 0.5 \end{bmatrix} \quad S^{(2)} = \begin{bmatrix} 1 & -0.5 & -0.5 \\ -0.5 & 0.5 & 0 \\ -0.5 & 0 & 0.5 \end{bmatrix}$$

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

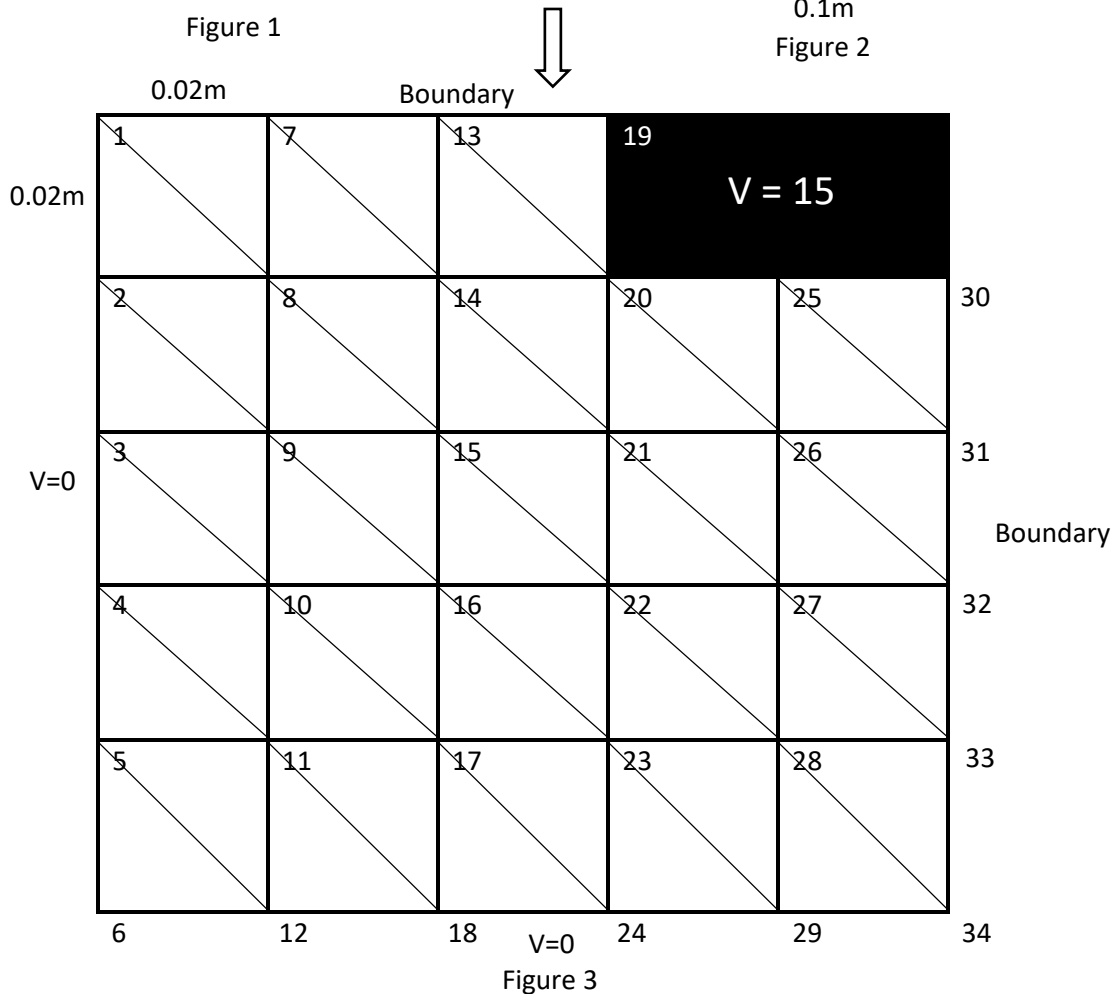
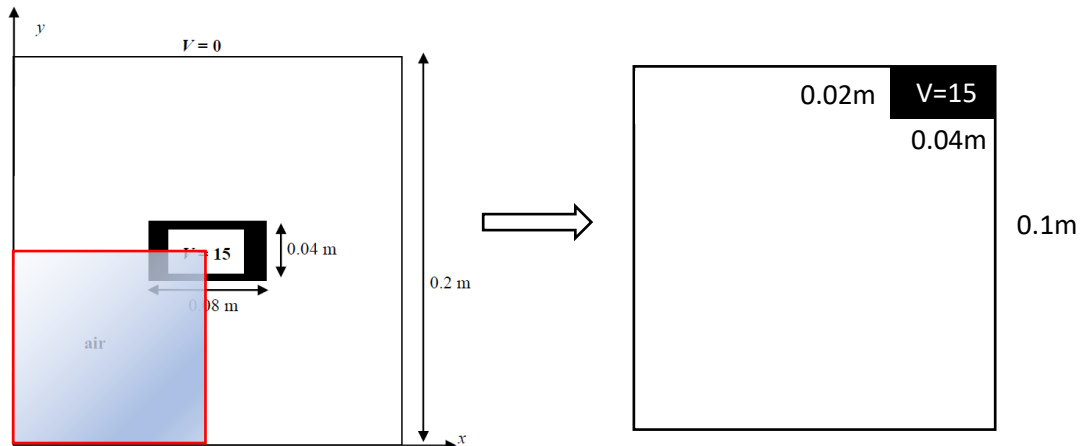
$$U_{dis} = C U_{con}$$

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \end{bmatrix}_{dis} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix}_{con}$$

$$\begin{aligned} S = C^T S_{dis} C &= \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.5 & -0.5 & 0 & 0 & 0 & 0 \\ -0.5 & 1 & -0.5 & 0 & 0 & 0 \\ 0 & -0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -0.5 & -0.5 \\ 0 & 0 & 0 & -0.5 & 0.5 & 0 \\ 0 & 0 & 0 & -0.5 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & -0.5 & 0 & -0.5 \\ -0.5 & 1 & -0.5 & 0 \\ 0 & -0.5 & 1 & -0.5 \\ -0.5 & 0 & -0.5 & 1 \end{bmatrix} \end{aligned}$$

Question 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.



As we can see from figure 3 above, the mesh contains total 34 nodes and 46 triangle elements. Node 1, 2, 3, 4, 5, 6, 12, 18, 24, 29 and 34 are at 0 volts while node 19, 20, 25 and 30 are at 15 volts. Then we generate three input files “file1.dat” “file2.dat” and “file3.dat” for SIMPLE_2D program as shown below:

```

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

```

Figure 4 file1.dat

```

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

```

Figure 5 file2.dat

```

1 0.00
2 0.00
3 0.00
4 0.00
5 0.00
6 0.00
19 15.00
20 15.00
25 15.00
30 15.00
12 0.00
18 0.00
24 0.00
29 0.00
34 0.00

```

Figure 6 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.

Then I ran the SIMPLE2D program in Matlab2018b and results are shown in the screenshot below:

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

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

Figure 7

As we can see from the result labeled in red in figure 7, the potential at $(x,y) = (0.06, 0.04)$ is the potential at node 22 which is 5.5263v.

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

For this question, we know the energy equation is:

$$E = \frac{1}{2} CV^2 \dots\dots\dots (1)$$

So that we can calculate the capacitance by using equation:

$$C = 2 * \frac{E_{total}}{V^2} \dots\dots\dots (2)$$

Since we cut the coaxial cable into 4 equal parts, we can calculate E_{total} by 4 times E for each square. And we can get E from equation learned in class:

$$E = \frac{1}{2} * \epsilon_0 * U_{con}^T * S * U_{con} \dots\dots\dots (3)$$

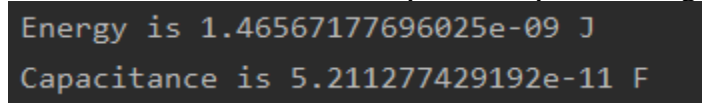
In my case, I will write the node number for each small square as same as the number labeled in question 1 figure 1. Thus, I can re-use the S from question 1. We also get the result of U_{con} at each node from part b. Then we can calculate the capacitance by plugging equation (3) into equation (2) and generated the new equation:

$$C = \frac{4 * \epsilon_0 * U_{con}^T * S * U_{con}}{V^2} \dots\dots\dots (4)$$

where $V = 15$ as mentioned in the question.

Then I developed a program named *Capacitance* to solve this question. First, I copy the result of voltage from Matlab in question 2 in a file named “result.dat”. Then the function *readResult(filename)* will read the result file and store those voltages in a vector U . Then function *generateMeshEnergy(U, Sg, uFix)* will take the vector U , the global $S(Sg)$ we calculated in question 1 and fixed voltage($uFix$) to calculate the total energy by using equations (3) listed above. To calculate the capacitance, a function named *capacitance(energy)* based equation (4) is developed.

Finally, by running the program, we get the energy and capacitance as shown in the screenshot below, where the capacitance per unit length is around 52.11pF.



```
Energy is 1.46567177696025e-09 J
Capacitance is 5.211277429192e-11 F
```

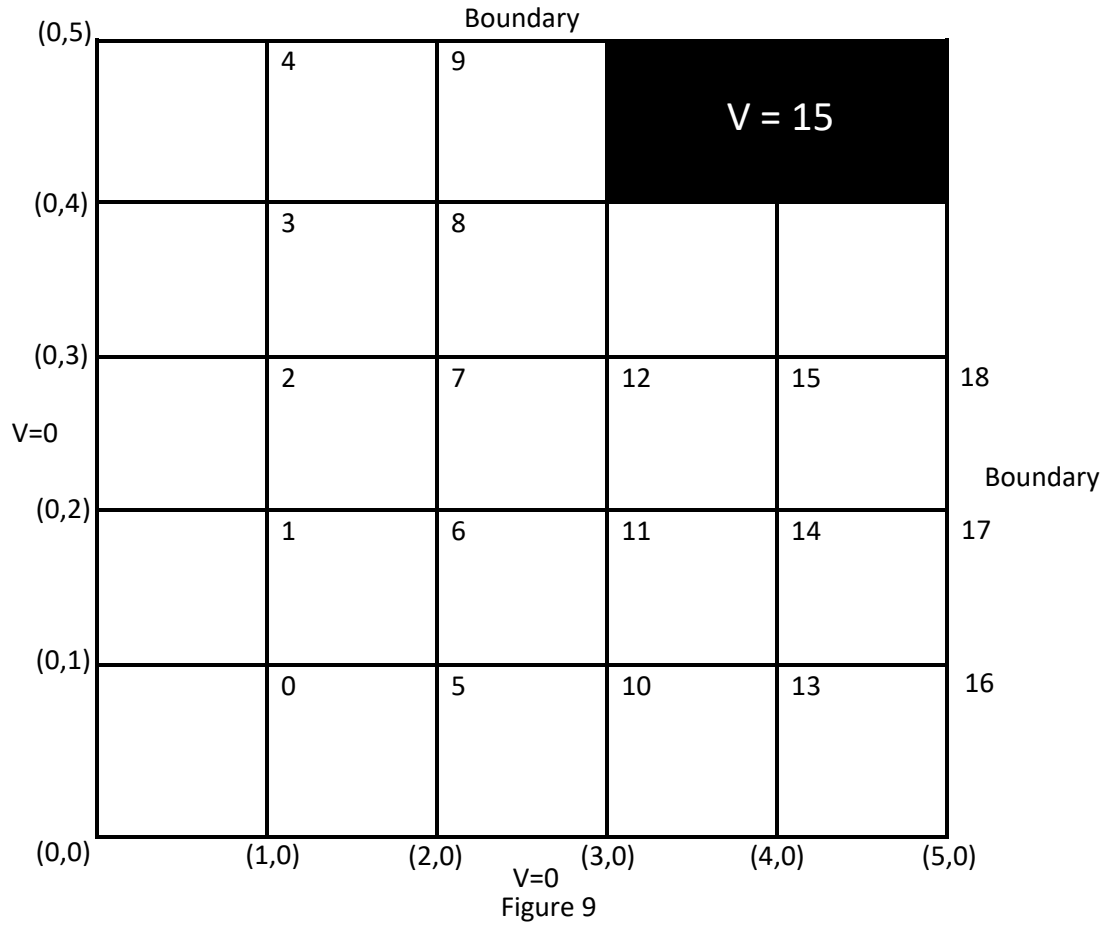
Figure 8

Question 3

Write a program implementing the conjugate gradient method (un-preconditioned). Solve 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 shown in Figure 2 that you 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.)

- (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.**

The one-quarter cross-section used in this question will be same as in Question 2. But in order to implement the five-points-difference method we learned in Finite Differences method class, the mesh has been re-numbered as shown below in Figure 9:



As we can see, now we have 19 free nodes. To generate the matrix A, we will use the five-points method where:

$$\varphi_{(i+1,j)} + \varphi_{(i-1,j)} + \varphi_{(i,j+1)} + \varphi_{(i,j-1)} - 4 * \varphi_{(i,j)} = 0 \dots \dots \dots (5)$$

We also need to consider the boundary conditions since we are making the use of symmetry. The matrix A we generated is shown below in figure 10:

```
[-4.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[1.0, -4.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 1.0, -4.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 1.0, -4.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 2.0, -4.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, -4.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, -4.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, -4.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, -4.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 2.0, -4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, -4.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, -4.0, 1.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, -4.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 1.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 2.0, -4.0]
```

Figure 10

The vector b that we will use to generate the equation $Ax = b$ is shown in figure 11:
 $[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -15, -15, 0.0, 0.0, -15, 0.0, 0.0, -15, 0.0, 0.0, -15]$

Figure 11

Then we can establish the matrix equation $Ax = b$ using matrix A and vector b we got above. However, the matrix A is not a positive definite matrix since it cannot pass the symmetric check in Choleski decomposition method developed in matrix class from assignment 1. An error message shown in figure 12 will pop out.

Input matrix must be a symmetric matrix!

Figure 12

To solve this problem, we can multiple a transpose of matrix A on both side of the equation $Ax = b$, Thus the equation will become $A^T Ax = A^T b$, where the result matrix of $A^T A$ is a positive-definite matrix as shown in figure 13:

$[18.0, -8.0, 1.0, 0.0, 0.0, -8.0, 2.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],$
 $[-8.0, 19.0, -8.0, 1.0, 0.0, 2.0, -8.0, 2.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],$
 $[1.0, -8.0, 19.0, -8.0, 1.0, 0.0, 2.0, -8.0, 2.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],$
 $[0.0, 1.0, -8.0, 22.0, -12.0, 0.0, 0.0, 2.0, -8.0, 3.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],$
 $[0.0, 0.0, 1.0, -12.0, 18.0, 0.0, 0.0, 0.0, 3.0, -8.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],$
 $[-8.0, 2.0, 0.0, 0.0, 0.0, 19.0, -8.0, 1.0, 0.0, 0.0, -8.0, 2.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0],$
 $[2.0, -8.0, 2.0, 0.0, 0.0, -8.0, 20.0, -8.0, 1.0, 0.0, 2.0, -8.0, 2.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0],$
 $[0.0, 2.0, -8.0, 2.0, 0.0, 1.0, -8.0, 20.0, -8.0, 1.0, 0.0, 2.0, -8.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0],$
 $[0.0, 0.0, 2.0, -8.0, 3.0, 0.0, 1.0, -8.0, 22.0, -12.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],$
 $[0.0, 0.0, 0.0, 3.0, -8.0, 0.0, 0.0, 1.0, -12.0, 18.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],$
 $[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, -8.0, 2.0, 0.0, 0.0, 0.0, 19.0, -8.0, 1.0, -8.0, 2.0, 0.0, 1.0, 0.0, 0.0],$
 $[0.0, 1.0, 0.0, 0.0, 0.0, 2.0, -8.0, 2.0, 0.0, 0.0, -8.0, 20.0, -8.0, 2.0, -8.0, 2.0, 0.0, 1.0, 0.0, 0.0],$
 $[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 2.0, -8.0, 1.0, 0.0, 1.0, -8.0, 19.0, 0.0, 2.0, -8.0, 0.0, 0.0, 1.0, 0.0],$
 $[0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, -8.0, 2.0, 0.0, 22.0, -8.0, 1.0, -12.0, 3.0, 0.0, 0.0],$
 $[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 2.0, -8.0, 2.0, -8.0, 23.0, -8.0, 3.0, -12.0, 3.0, 0.0],$
 $[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 2.0, -8.0, 1.0, -8.0, 22.0, 0.0, 3.0, -12.0],$
 $[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, -12.0, 3.0, 0.0, 18.0, -8.0, 1.0],$
 $[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 3.0, -12.0, 3.0, -8.0, 19.0, -8.0],$
 $[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 3.0, -12.0, 1.0, -8.0, 18.0]$

Figure 13

$A^T b$ then becomes:

$[0.0, 0.0, 0.0, -15.0, -15.0, 0.0, 0.0, -30.0, 30.0, 45.0, 0.0, -15.0, 45.0, 0.0, -15.0, 15.0, 0.0, -15.0, 45.0]$

Figure 14

- (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.

After the modification, the result vector x can be then solved by using the Choleski decomposition and Forward/Backward substitution. The result is shown below in figure 15:

The result vector x solved by using the Choleski Decomposition is:
 $[0.9571, 1.9667, 3.0262, 3.959, 4.2525, 1.8616, 3.8834, 6.1791, 8.5575, 9.0919, 2.606, 5.5263, 9.2492, 3.036, 6.3668, 10.2912, 3.1714, 6.6135, 10.549]$

Figure 15

For the Conjugate Gradient program, I developed a function named *ConjugateGradient(A, b)* where it will take the matrix A and right-hand-vector b as inputs and use the formula of Conjugate Gradient method to calculate the result x . The algorithm of the Conjugate Gradient method learned in class is given in the next page:

Initial vectors:

$$\begin{aligned}x^{(0)} &= 0 \\r^{(0)} &= b - Ax^{(0)} \\p^{(0)} &= r^{(0)}\end{aligned}$$

In the loop for $k = 0, 1 \dots \dots \text{size of } b$:

$$\begin{aligned}\alpha^{(k)} &= \frac{p^{(k)T} r^{(k)}}{p^{(k)T} A p^{(k)}} \\x^{(k+1)} &= x^{(k)} + \alpha^{(k)} p^{(k)} \\r^{(k+1)} &= b - A x^{(k+1)} \\\beta^{(k)} &= -\frac{p^{(k)T} A r^{(k+1)}}{p^{(k)T} A p^{(k)}} \\p^{(k+1)} &= r^{(k+1)} + \beta^{(k)} p^{(k)}\end{aligned}$$

The result vector x solved by using the Conjugate Gradient method is shown below:

The result vector x solved by using the Conjugate Gradient method is:
[0.9571, 1.9667, 3.0262, 3.959, 4.2525, 1.8616, 3.8834, 6.1791, 8.5575, 9.0919, 2.606, 5.5263, 9.2492, 3.036, 6.3668, 10.2912, 3.1714, 6.6135, 10.549]

Figure 16

As we can compare figure 16 and figure 15, the result vector calculated by using Conjugate Gradient method is same as the result we get by using the Choleski decomposition and Forward/Backward substitution method.

(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.

Iteration	Infinite norm	2-norm
0	45	96.0469
1	44.4373	75.6998
2	22.536	46.7838
3	14.1089	32.2778
4	12.2942	25.5219
5	9.2095	21.7203
6	8.8128	16.3987
7	11.3686	15.0198
8	7.9873	17.972
9	9.1604	15.4563
10	6.8282	12.7232
11	3.8933	10.9194
12	4.4415	9.5137
13	2.0753	4.6026
14	1.2522	2.713
15	1.6066	3.1026
16	1.0778	2.5253
17	0.3373	0.7709
18	0.0089	0.021
19	0	0

Chart 1

To calculate the infinity norm and the 2-norm of Conjugate Gradient method, the *ConjugateGradient*(A, b) function has been slightly modified and can return the result vectors of infinity norm and the 2-norm. The formulas used for calculating the infinity norm and 2-norm of the residual vector are shown below:

$$\text{2-norm: } \sqrt{\sum_{i=1}^n |res_i|^2} \dots\dots\dots (6)$$

$$\text{Infinity norm: } \max(|res_j|) \text{ where } j = 0, 1 \dots \dots n \dots (7)$$

The results of infinity norm and the 2-norm of the residual vector and the number of iterations is shown in the chart 1 on the left. Based on this chart, I plotted the graph of the infinity norm and the 2-norm of the residual vector versus the number of iterations in below.

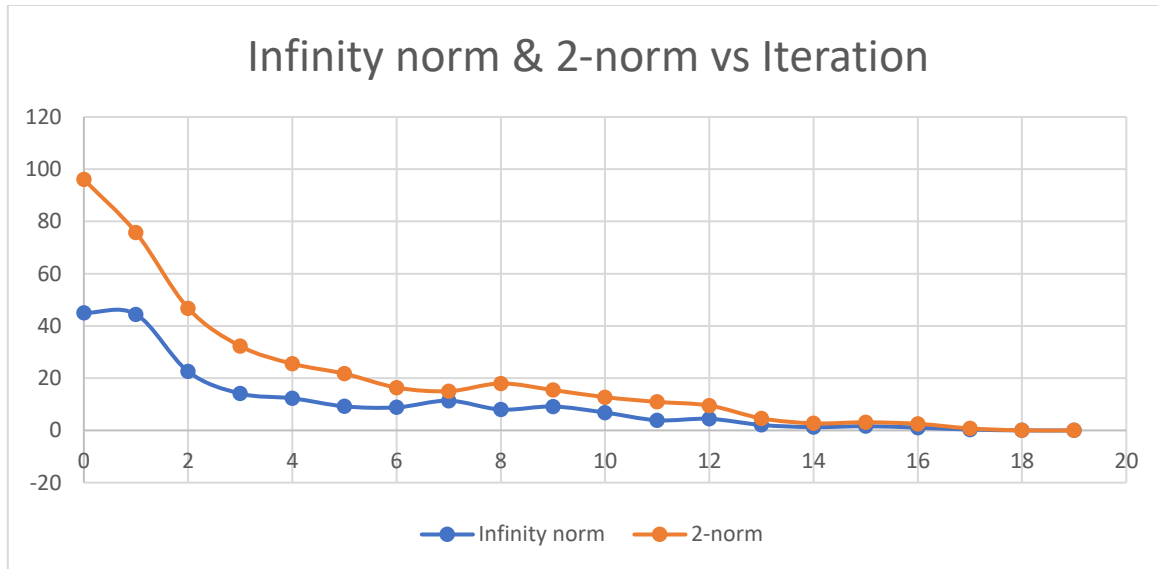


Chart 2

As we can see in chart 2, the 2-norm is almost doubled the infinity norm when the iteration starts, while both of them die out into zero in 19 iterations.

- (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.

The potential at $(x,y) = (0.06, 0.04)$ we calculated by using the Choleski decomposition and the Conjugate Gradient method is given in the screenshot below. As we can see, we got the same result from two methods where the potential is 5.5263v.

```
The potential at (x,y) = (0.06, 0.04) solved by using the Choleski Decomposition is:
5.5263v
The potential at (x,y) = (0.06, 0.04) solved by using the Conjugate Gradient method is:
5.5263v
```

Figure 17

From Assignment 1, the potential at $(x,y) = (0.06, 0.04)$ calculated by using the SOR with the same node spacing $h = 0.2$ is 5.59168557v (screenshot shown below in figure 18). Which is about 1.18% higher than the result we got from using Choleski, Conjugate Gradient and SIMPLE_2D method. From the chart 3 below we can see that the potential at $(x,y) = (0.06, 0.04)$ we got by using Choleski, Conjugate Gradient and SIMPLE_2D method is the same, which is 5.5263v.

```
[ [15.      15.      15.      9.38368664  4.47994059  0.      ]
  [15.      15.      15.      8.67111995  4.05613511  0.      ]
  [10.74457853 10.38357155 9.30497872 6.24465853 3.07347998 0.      ]
  [ 6.85017333 6.48472831 5.59168557 3.92905542 1.99312623 0.      ]
  [ 3.32121807 3.11348175 2.64797976 1.8867512 0.96996935 0.      ]
  [ 0.         0.         0.         0.         0.         0.      ]]
```

Figure 18

Node	Choleski	Conjugate Gradient	SIMPLE_2D	SOR
V at (0.06, 0.04)	5.5263v	5.5263v	5.5263v	5.5917v

Chart 3

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

We can re-use the formula (1) and (2) of capacitance calculation in question 2 (c) above that $E = \frac{1}{2} CV^2$, $C = 2 * \frac{E_{total}}{V^2}$ to compute the capacitance per unit length. Now the E_{total} will become 4 times the sum of all node voltages in the one-quarter cross-section we found by using the finite difference method and Conjugate Gradient method or Choleski decomposition method programs developed in previous questions.

Appendix

(See following pages of codes)

```

1 from Matrix import matrix
2
3 # Author@Yi Zhu
4 # ID@260716006
5
6 matrix = matrix()
7
8
9 class Capacitance(object):
10     def __init__(self):
11         pass
12
13     S = [[1, -0.5, 0, -0.5], [-0.5, 1, -0.5, 0], [0, -0.5, 1, -0.5], [-0.5, 0, -0.5, 1
14 ]]
15     # Function generating the energy matrix using voltage at free nodes, fixed nodes
    and SgGlobal
16     def generateMeshEnergy(self, U, Sg, uFix):
17         n = 0
18         m = 0
19         sum = 0
20         while (n < 28):
21             if (n == 5 or n == 11 or n == 17 or n == 18 or n == 23):
22                 n += 1
23             if (n < 17):
24                 uV = self.part1(U, n)
25                 w = matrix.matrixVectorMultiplication(Sg, uV)
26                 e = matrix.vectorMultiplication(uV, w)
27                 sum += 0.5 * e
28                 n += 1
29                 m += 1
30             elif (n > 18 and n < 28):
31                 uV = self.part2(U, n)
32                 w = matrix.matrixVectorMultiplication(Sg, uV)
33                 e = matrix.vectorMultiplication(uV, w)
34                 sum += 0.5 * e
35                 n += 1
36                 m += 1
37
38             wFix = matrix.matrixVectorMultiplication(Sg, uFix)
39             eFix = matrix.vectorMultiplication(uFix, wFix)
40             sum += 2 * 0.5 * eFix
41             sum = sum * 8.85 * pow(10, -12)
42             return sum
43
44     # Stamp free node voltage from U to nodes number <17
45     def part1(self, U, n):
46         uVector = []
47         a = U[n]
48         b = U[n + 1]
49         c = U[n + 7]
50         d = U[n + 6]
51         uVector.append(a)
52         uVector.append(b)
53         uVector.append(c)
54         uVector.append(d)
55         return uVector
56
57     # Stamp free node voltage from U to nodes number <28
58     def part2(self, U, n):
59         uVector = []
60         a = U[n]
61         b = U[n + 1]
62         c = U[n + 6]
63         d = U[n + 5]
64         uVector.append(a)
65         uVector.append(b)

```

```
66         uVector.append(c)
67         uVector.append(d)
68         return uVector
69
70     # Calculate capacitance per unit length using formula  $C=2E/V^2$ 
71     def capacitance(self, energy):
72         return 2 * 4 * energy / (15 * 15)
73
74     # Read the potential from output file of SIMPLE_2D method
75     def readResult(self, filename):
76         U = []
77         file = open(filename, "r")
78         potential = file.readlines()
79         for row in potential:
80             U.append(float(row.split("\n")[0].split(" ")[-1]))
81         return U
82
```

```

1 from Matrix import matrix
2 from math import sqrt
3 import copy
4
5 # Author@Yi Zhu
6 # ID@260716006
7
8 m = matrix()
9
10
11 class ConjugateGradient(object):
12     def __init__(self):
13         pass
14
15     # Generate a mesh Matrix A containing voltages at free node using five-points
method
16     def generateMeshMatrix(self):
17         A = [[0.0] * 19 for i in range(19)]
18         for i in range(19):
19             A[i][i] = -4
20             if (i < 10):
21                 if (i < 8):
22                     A[i][i + 5] = 1
23                 if (i != 0 and i != 5):
24                     A[i][i - 1] = 1
25                 if (i + 1 == 5 or i + 1 == 10):
26                     A[i][i - 1] += 1
27             else:
28                 A[i][i + 1] = 1
29             if i > 4:
30                 A[i][i - 5] = 1
31
32         elif (i < 13):
33             A[i][i - 5] = 1
34             A[i][i + 3] = 1
35             if (i != 10):
36                 A[i][i - 1] = 1
37             if (i != 12):
38                 A[i][i + 1] = 1
39         else:
40             A[i][i - 3] = 1
41             if (i < 16):
42                 A[i][i + 3] = 1
43             else:
44                 A[i][i - 3] += 1
45             if (i != 15) and (i != 18):
46                 A[i][i + 1] = 1
47             if (i != 13) and (i != 16):
48                 A[i][i - 1] = 1
49         return A
50
51     # Generate a vector b containing voltages at fixed node
52     def generateBVector(self):
53         b = [0.0] * 19
54         for i in range(19):
55             if i == 8 or i == 9 or i == 12 or i == 15 or i == 18:
56                 b[i] = -15
57         return b
58
59     # Function solve Ax=b using Conjugate Gradient method and calculate infinity norm
and 2-norm
60     # Return result x, infinity norm and 2-norm
61     def ConjugateGradient(self, A, b):
62         infNorm = []
63         twoNorm = []
64         nCol = len(b)
65         x = [0 for i in range(nCol)]

```

```

66     r = m.vectorSubtraction(b, m.matrixVectorMultiplication(A, x))
67     p = copy.deepcopy(r)
68
69     infiniteNorm = max([abs(j) for j in r])
70     infNorm.append(infiniteNorm)
71     res = 0
72     norm = 0
73     for k in r:
74         res += k ** 2
75         norm = sqrt(res)
76     twoNorm.append(norm)
77
78     for i in range(nCol):
79         alpha = (m.vectorMultiplication(p, r)) / (m.vectorMultiplication(m.
vectorMatrixMultiplication(p, A), p))
80         x = m.vectorAddition(x, [alpha * j for j in p])
81         r = m.vectorSubtraction(b, m.matrixVectorMultiplication(A, x))
82         beta = -1 * ((m.vectorMultiplication(p, m.matrixVectorMultiplication(A, r
))) / (
83             m.vectorMultiplication(p, m.matrixVectorMultiplication(A, p))))
84         p = m.vectorAddition(r, [beta * j for j in p])
85
86         infiniteNorm = max([abs(k) for k in r])
87         infNorm.append(infiniteNorm)
88         res = 0
89         norm = 0
90         for l in r:
91             res += l ** 2
92             norm = sqrt(res)
93         twoNorm.append(norm)
94
95     return x, infNorm, twoNorm
96

```

```

1 import math, copy
2 #Author@Yi Zhu
3 #ID@260716006
4
5 class matrix(object):
6
7     # Method to solve Ax=b using Choleski and fwd/bwd elimination
8     def solveMatrix(self, matrix, b):
9         self.checkSym(matrix)
10        self.checkDet(matrix)
11        L = self.choleskiDecompose(matrix)
12        Y = self.forwardElm(L, b)
13        Lt = self.matrixTranspose(L)
14        X = self.backElm(Lt, Y)
15        return X
16
17    # Method to solve Ax=b using Choleski and fwd/bwd elimination and sparse matrix
18    def sparseSolveMatrix(self, matrix, b, band):
19        self.checkSym(matrix)
20        self.checkDet(matrix)
21        L = self.sparseCholeskiDecompose(matrix, band)
22        Y = self.forwardElm(L, b)
23        Lt = self.matrixTranspose(L)
24        X = self.backElm(Lt, Y)
25        return X
26
27    # Choleski decomposition using look ahead method return L
28    def choleskiDecompose(self, matrix):
29        A = copy.deepcopy(matrix)
30        n = len(A)
31        L = [[0.0] * n for i in range(n)]
32        for j in range(n):
33            if A[j][j] < 0:
34                exit('Input matrix must be a positive-definite matrix!')
35            L[j][j] = math.sqrt(A[j][j])
36            for i in range(j + 1, n):
37                L[i][j] = A[i][j] / L[j][j]
38                for k in range(j + 1, i + 1):
39                    A[i][k] = A[i][k] - L[i][j] * L[k][j]
40        return L
41
42    # Choleski decomposition using look ahead method and sparse matrix return L
43    def sparseCholeskiDecompose(self, matrix, b):
44        A = copy.deepcopy(matrix)
45        n = len(A)
46        L = [[0.0] * n for i in range(n)]
47        for j in range(n):
48            if A[j][j] < 0:
49                exit('Input matrix must be a positive-definite matrix!')
50            L[j][j] = math.sqrt(A[j][j])
51            for i in range(j + 1, min(j + 1 + b, n)):
52                L[i][j] = A[i][j] / L[j][j]
53                for k in range(j + 1, min(j + 1 + b, i + 1)):
54                    A[i][k] = A[i][k] - L[i][j] * L[k][j]
55        return L
56
57    # Forward elimination return Y
58    def forwardElm(self, lMatrix, bVector):
59        L = copy.deepcopy(lMatrix)
60        b = copy.deepcopy(bVector)
61        n = len(b)
62        Y = []
63        for j in range(n):
64            b[j] = b[j]/L[j][j]
65            Y.append(b[j])
66            for i in range(j+1, n):
67                b[i] = b[i] - L[i][j]*b[j]

```



```

68         return Y
69
70     # Backward elimination return X
71     def backElm(self, LtMatrix, yVector):
72         Lt = copy.deepcopy(LtMatrix)
73         Y = copy.deepcopy(yVector)
74         n = len(Y)
75         X = []
76         for i in range(n)[::-1]:
77             sum = 0
78             for j in range(n)[i:-1]:
79                 sum += X[n - j - 1] * Lt[i][j]
80             X.append((Y[i] - sum) / Lt[i][i])
81         return X[::-1]
82
83     # Check if the matrix is symmetric
84     def checkSym(self, matrix):
85         n = len(matrix)
86         for i in range(n):
87             for j in range(i + 1, n):
88                 if matrix[i][j] != matrix[j][i]:
89                     exit('Input matrix must be a symmetric matrix!')
90
91     # Check if the determinant of the matrix is zero
92     def checkDet(self, matrix):
93         n = len(matrix)
94         det = 1
95         for i in range(n):
96             det *= matrix[i][i]
97         if det <= 0:
98             exit('Input matrix must be a positive definite matrix!')
99
100    # Check if A * x equals b
101    def checkSol(self, A, X, b):
102        n = len(A)
103        for i in range(n):
104            res = self.matrixVectorMultiplication(A,X)
105            if abs(b[i] - res[i]) > 0.01:
106                return False
107        return True
108
109    # Matrix transpose return the transpose of a matrix
110    def matrixTranspose(self, matrix):
111        A = copy.deepcopy(matrix)
112        nRow = len(A)
113        nCol = len(A[0])
114        res = []
115        for i in range(nCol):
116            row = []
117            for j in range(nRow):
118                row.append(A[j][i])
119            res.append(row)
120        return res
121
122    # Matrix multiply by a vector return the result vector
123    def matrixVectorMultiplication(self, A, b):
124        res = []
125        nRow = len(A)
126        nCol = len(b)
127        for i in range(nRow):
128            sum = 0
129            for j in range(nCol):
130                sum += A[i][j] * b[j]
131            res.append(sum)
132        return res
133
134    # Matrix multiply by another matrix return the result matrix

```

```

135     def matrixMultiplication(self, A, B):
136         nRow = len(A)
137         nCol = len(B[0])
138         nB = len(B)
139         res = [0.0] * nRow
140         for i in range(nRow):
141             res[i] = [0] * nCol
142         for i in range(nRow):
143             for j in range(nCol):
144                 for k in range(nB):
145                     res[i][j] += A[i][k] * B[k][j]
146         return res
147
148     # Vector multiply by another matrix return the result vector
149     def vectorMatrixMultiplication(self, b, A):
150         res = []
151         nRow = len(A)
152         nCol = len(b)
153         for i in range(nRow):
154             sum = 0
155             for j in range(nCol):
156                 sum += b[j] * A[i][j]
157             res.append(sum)
158         return res
159
160     # Two vectors adding up return the result vector
161     def vectorAddition(self, a, b):
162         n = len(a)
163         res = []
164         for i in range(n):
165             res.append(a[i] + b[i])
166         return res
167
168     # Two vectors subtracting return result vector
169     def vectorSubtraction(self, a, b):
170         n = len(a)
171         res = []
172         for i in range(n):
173             res.append(a[i] - b[i])
174         return res
175
176     # Two vectors multiplying return result value
177     def vectorMultiplication(self, a, b):
178         nb = len(b)
179         res = 0
180         for i in range(nb):
181             res += a[i] * b[i]
182         return res
183
184     # Two matrix subtraction return result matrix
185     def matrixSubtract(self, A, B):
186         res = []
187         nRow = len(A)
188         nCol = len(A[0])
189         for i in range(nRow):
190             row = []
191             for j in range(nCol):
192                 row.append(A[i][j]-B[i][j])
193             res.append(row)
194         return res
195
196

```