ECSE 543: Numerical Methods

Assignment 2 Report

Wenjie Wei
260685967

November 11, 2018

Contents

Introduction

In this assignment, three numerical methods discussed in class were explored. The interpreter used for the Python codes is Python 3.6.

1 First Order Finite Element Problem

Figure 1 shows an illustration of the first order triangular finite element problem to be solved.

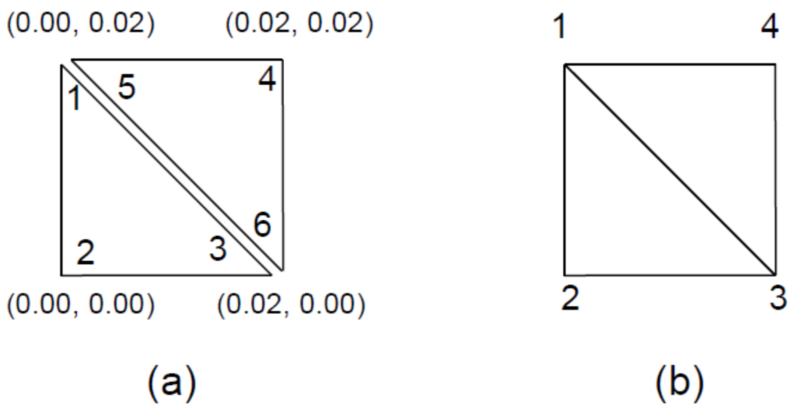

Figure 1: 1st Order Triangular FE Problem

Take the triangle with nodes 1, 2, and 3 as the beginning step. Firstly, interpolate the potential U as:

$$U = a + bx + cy$$

and at vertex 1, we can write an equation of potential as:

$$U_1 = a + bx_1 + cy_1$$

Thus, we can have a vector of potentials for vertex 1, 2, and 3 as follows:

$$\begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

and the terms a, b, c are acquired following:

$$U = \sum_{i=1}^{3} U_i \alpha_i(x, y) \tag{1}$$

and we can derive a general formula for α_i:

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

where A holds the value of the area of the triangle.

Following Equation 2, when the index i exceeds the top limit 3, it is wrapped around to 1. Now we can get the following calculations for α_1, α_2 and α_3:

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

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

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

With the expressions for α derived, we now go ahead and calculate the $S_{ij}^{(e)}$ matrices. The general formula below is used to calculate the $\boldsymbol{S}$ matrix:

$$S_{ij}^{(e)} = \int_{\Delta e} \nabla\alpha_i \nabla\alpha_j dS \tag{3}$$

Using the equation above, plug in the values provided in Figure 1, we can have the following calculations:

$$S_{11} = \frac{1}{4A}[(y_2 - y_3)^2 + (x_3 - x_2)^2] = \frac{1}{4 \times 2 \times 10^{-4}}[0 + 0.02^2] = 0.5$$

$$S_{12} = \frac{1}{4A}[(y_2 - y_3)(y_3 - y_1) + (x_3 - x_2)(x_1 - x_3)] = -0.5$$

$$S_{13} = \frac{1}{4A}[(y_2 - y_3)(y_1 - y_2) + (x_3 - x_2)(x_2 - x_1)] = 0$$

Before performing the calculation for the next row, we inspect the calculation rules of the entries of the $\boldsymbol{S}$ matrix, we can easily discover that $S_{ij} = S_{ji}$, since the flip of the orders of the operands in the parenthesis results in the same sign of the result. Therefore, the following statements can be made:

$$S_{21} = S_{12} = -0.5$$

$$S_{31} = S_{13} = 0$$

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

$$S_{23} = S_{32} = \frac{1}{4A}[(y_3 - y_1)(y_1 - y_2) + (x_1 - x_3)(x_2 - x_1)] = -0.5$$

$$S_{33} = \frac{1}{4A}[(y_1 - y_2)^2 + (x_2 - x_1)^2] = 0.5$$

From the calculation results above, we can come up with the $\boldsymbol{S}$ matrix for vertices 1, 2, and 3:

$$S^{(1)} = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{bmatrix} = \begin{bmatrix} 0.5 & -0.5 & 0 \\ -0.5 & 1 & -0.5 \\ 0 & -0.5 & 0.5 \end{bmatrix}$$

Use the similar approach for the other triangle and obtain S_{456}:

$$S^{(2)} = \begin{bmatrix} S_{44} & S_{45} & S_{46} \\ S_{54} & S_{55} & S_{56} \\ S_{64} & S_{65} & S_{66} \end{bmatrix} = \begin{bmatrix} 1 & -0.5 & -0.5 \\ -0.5 & 0.5 & 0 \\ -0.5 & 0 & 0.5 \end{bmatrix}$$

Add the triangles to get the energy of the whole system shown in (b) of Figure 1:

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

which is also denoted as:

$$U_{dis} = CU_{joint}$$

Use $\mathbf{S}_{dis}$ to denote a 6×6 matrix to represent the disjoint matrix:

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

Now the global $\boldsymbol{S}$ matrix will be calculated as:

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

which is the final result of this problem.

2 Coaxial Cable Electrostatic Problem

Use the triangular finite element model for the analysis of the coaxial cable problem seen in the previous assignment. We take the third quadrant for the analysis.

2.a The Finite Element Mesh

Listing 1 shows the implementation of the construction of the finite element mesh and the creation of the MATLAB input file.

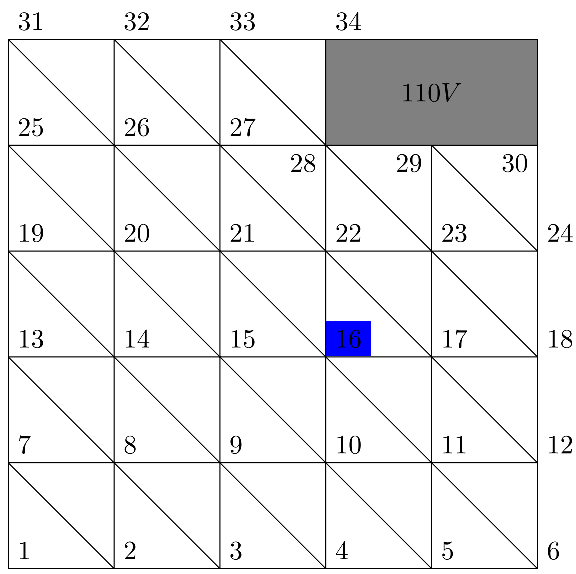

Figure 2: Organization of the Finite Element Mesh

Figure 2 shows the organization of the finite element mesh constructed by the program. The input file written by this program is shown in Listing 2. Note that the first number at the beginning of the lines are not an input to the MATLAB file, as it is the line number which is provided by the *minted* package in LaTeX.

2.b Potential Solved by SIMPLE2D.m

Use the input file generated in the previous section, we are able to use the MATLAB file to calculate the potential at every node we have specified. The output of the SIMPLE2D.m file is shown in Listing 3. The target node (0.06, 0.04) is highlighted in blue as Node 16, and from Listing 3 shows that the potential at this node is $40.527V$.

2.c Capacitance per Unit Length

To compute the capacitance, apply the fundamental Equation 4:

$$E = \frac{1}{2}CV^2 \tag{4}$$

Now apply the finite element method used in the previous section. Use U_{joint} to denote the potential vector shown in Listing 3. Use the S_{joint} calculated in the first question, we derive an equation to calculate the energy for each square finite element:

$$W = \frac{1}{2}U_{joint}^T S_{joint} U_{joint} \tag{5}$$

The value of the entries in the U matrix are the potential on the four corners of the square defined by the two finite element triangles.

Use the Python function implemented in Listing 1 which applies Equation 5, we are able to calculate the total energy contained in the cable. The energy contained in the cable is calculated to be $3.154 \times 10^{-7}J$. The following calculation is performed to find the total capacitance per unit length:

$$C = \frac{2E}{V^2} = \frac{2\varepsilon_0 W}{V^2} = 5.2137 \times 10^{-11}F/m$$

which equals to $52.137pF/m$.

3 Conjugate Gradient

3.a S. P. C Check

Use the Choleski decomposition implemented in the previous assignment to check if the matrix $\boldsymbol{A}$ is symmetric, positive and definite. The program fails with an exception when performing the symmetry check. Therefore, the matrix is not symmetric, positive definite.

To obtain a symmetric, positive definite matrix in order to pass the Choleski decomposition test, we will need to multiply A^T to both sides of the equation.

A Code Listings

Listing 1: Finite Element Mesh Implementation (`finite_element.py`).

```python
from matrix import Matrix
from finite_difference import Node

EPSILON = 8.854188e-12
HIGH_VOLTAGE = 110
LOW_VOLTAGE = 0
SPACING = 0.02
s_vec = [[1, -0.5, 0, -0.5],[-0.5, 1, -0.5, 0],[0, -0.5, 1, -0.5],[-0.5, 0, -0.5, 1]]
S = Matrix(s_vec, 4, 4)
f = open('SIMPLE2Dinput.dat', 'w')


class two_element(object):
    def __init__(self, x, y, bl_node, id):
        """
        This is the constructor of a two-triangle finite element
        the vertices are numbered from 0 to 5, replacing 1 - 6 in question 1

        :param x: x coord for the bottom-left corner
        :param y: y coord for the bottom-left corner
        """

        # vertices are put in the array
        # vertices 2&5, vertices 0&4 have the same properties
        self._vertex_array = [Node(0) for _ in range(6)]
        self._vertex_array[5] = self._vertex_array[2]
        self._vertex_array[4] = self._vertex_array[0]
        self._bl_x = x
        self._bl_y = y

        self._bl_node = bl_node
        self._tl_node = bl_node + 6
        self._br_node = bl_node + 1
        self._tr_node = bl_node + 7

        self._id = id

        if (self._bl_x + SPACING) > 0.1 or (self._bl_y + SPACING) > 0.1:
            raise ValueError("The finite elements cannot exceed the third quadrant!")

        if self._bl_y == 0:
            # configure node 1
            self._vertex_array[1].set_fixed()
            self._vertex_array[1].set_value(LOW_VOLTAGE)

            # configure node 2 and 5
            self._vertex_array[2].set_fixed()
            self._vertex_array[2].set_value(LOW_VOLTAGE)

            # configure node 3
            self._vertex_array[3].set_free()

            if self._bl_x == 0:
                # configure node 0 and 4
                self._vertex_array[0].set_fixed()
                self._vertex_array[0].set_value(LOW_VOLTAGE)
            else:
                self._vertex_array[0].set_free()
        elif self._bl_x >= 0.06 and self._bl_y == 0.06:
            # configure node 1
            self._vertex_array[1].set_free()

            # configure node 2 and 5
            self._vertex_array[2].set_free()

```

```python
            # configure node 0 and 4
            self._vertex_array[0].set_fixed()
            self._vertex_array[0].set_value(HIGH_VOLTAGE)

            # configure node 3
            self._vertex_array[3].set_fixed()
            self._vertex_array[3].set_value(HIGH_VOLTAGE)
        elif self._bl_x == 0.04 and self._bl_y == 0.06:
            # configure node 1
            self._vertex_array[1].set_free()

            # configure node 2 and 5
            self._vertex_array[2].set_free()

            # configure node 0 and 4
            self._vertex_array[0].set_free()

            # configure node 3
            self._vertex_array[3].set_fixed()
            self._vertex_array[3].set_value(HIGH_VOLTAGE)
        elif self._bl_x == 0.04 and self._bl_y == 0.08:
            # configure node 1
            self._vertex_array[1].set_free()

            # configure node 2 and 5
            self._vertex_array[2].set_fixed()
            self._vertex_array[2].set_value(HIGH_VOLTAGE)

            # configure node 0 and 4
            self._vertex_array[0].set_free()

            # configure node 3
            self._vertex_array[3].set_fixed()
            self._vertex_array[3].set_value(HIGH_VOLTAGE)
        elif self._bl_x == 0:
            # configure node 1
            self._vertex_array[1].set_fixed()
            self._vertex_array[1].set_value(LOW_VOLTAGE)

            # configure node 0 and 4
            self._vertex_array[0].set_fixed()
            self._vertex_array[0].set_value(LOW_VOLTAGE)

            # configure node 3
            self._vertex_array[3].set_free()

            # configure node 2 and 5
            self._vertex_array[2].set_free()
        else:
            for i in range(6):
                self._vertex_array[i].set_free()

    def print_two_element(self):
        for i in range(6):
            print("Vertex " + str(i) + " has value " + str(self._vertex_array[i].value) + ", free node: "
                  + str(self._vertex_array[i].is_free))

    @property
    def bl_x(self):
        return self._bl_x

    @property
    def bl_y(self):
        return self._bl_y

    @property
    def bl_node(self):
        return self._bl_node

    @property
```

```
    def tl_node(self):
        return self._tl_node

    @property
    def br_node(self):
        return self._br_node

    @property
    def tr_node(self):
        return self._tr_node

    @property
    def vertex(self, i):
        return self._vertex_array[i]

    @property
    def id(self):
        return self._id


def calc_energy(fe_matrix):
    file = open('potentials.dat', mode='r', encoding='utf-8-sig')
    lines = file.readlines()
    file.close()
    potentials = [0 for _ in range(len(lines))]

    energy = 0

    count = 0
    for line in lines:
        line = line.split(' ')
        line = [i.strip() for i in line]
        potentials[count] = line[3]
        count += 1

    for i in range(4, -1, -1):
        for j in range(5):
            temp_two_element = fe_matrix[i][j]

            if temp_two_element is not None:
                u_vec = [[0] for _ in range(4)]
                U = Matrix(u_vec, 4, 1)

                U[0][0] = float(potentials[temp_two_element.bl_node + 5])
                U[1][0] = float(potentials[temp_two_element.bl_node - 1])
                U[2][0] = float(potentials[temp_two_element.bl_node])
                U[3][0] = float(potentials[temp_two_element.bl_node + 6])

                energy += 0.5 * EPSILON * U.T.dot_product(S.dot_product(U))[0][0]

    return energy


def write_mesh(fe_matrix):
    y_coord = 0
    count = 0

    print("Creating the mesh of the finite elements...")
    node_count = 0
    row = 1
    for i in range(4, -1, -1):
        x_coord = 0
        for j in range(5):
            if x_coord >= 0.06 and y_coord == 0.08:
                break
            else:
                temp_two_element =two_element(x_coord, y_coord, node_count + row, node_count)
                fe_matrix[i][j] = temp_two_element
                count += 1
                node_count += 1
```

```python

            x_coord += SPACING
        y_coord += SPACING
        row += 1

    print("Finite elements created: " + str(count * 2))

    # Now write the input file for SIMPLE2D.m
    print("Writing node information...")
    # write the bottom row
    i = 4
    for j in range(5):
        temp_two_element = fe_matrix[i][j]
        f.write('%d %.3f %.3f\n' % (temp_two_element.bl_node, temp_two_element.bl_x, temp_two_element.bl_y))
        if j == 4:
            f.write('%d %.3f %.3f\n' % (temp_two_element.br_node,
                                        temp_two_element.bl_x + SPACING, temp_two_element.bl_y))

    # write the general rows
    for i in range(4, -1, -1):
        for j in range(5):
            temp_two_element = fe_matrix[i][j]
            if temp_two_element is not None:
                if i != 0 and j != 4:
                    f.write('%d %.3f %.3f\n' %
                                  (temp_two_element.tl_node, temp_two_element.bl_x, temp_two_element.bl_y + SPACING))
                elif i != 0 and j == 4:
                    f.write('%d %.3f %.3f\n' %
                                  (temp_two_element.tl_node, temp_two_element.bl_x, temp_two_element.bl_y + SPACING))
                    f.write('%d %.3f %.3f\n' %
                                  (temp_two_element.tr_node, temp_two_element.bl_x + SPACING,
                                   temp_two_element.bl_y + SPACING))
                else:
                    if j != 2:
                        f.write('%d %.3f %.3f\n' %
                                  (temp_two_element.tl_node, temp_two_element.bl_x, temp_two_element.bl_y + SPACING))
                    else:
                        f.write('%d %.3f %.3f\n' %
                                      (temp_two_element.tl_node,
                                       temp_two_element.bl_x, temp_two_element.bl_y + SPACING))
                        f.write('%d %.3f %.3f\n' %
                                      (temp_two_element.tr_node, temp_two_element.bl_x + SPACING,
                                       temp_two_element.bl_y + SPACING))
            else:
                break

    f.write('\n')
    # Now write the triangle connection
    print("Writing triangle information...")
    for i in range(4, -1, -1):
        for j in range(5):
            temp_two_element = fe_matrix[i][j]
            if temp_two_element is not None:
                f.write('%d %d %d %.3f\n' %
                    (temp_two_element.bl_node, temp_two_element.br_node, temp_two_element.tl_node, 0))
            else:
                break
        for j in range(5):
            temp_two_element = fe_matrix[i][j]
            if temp_two_element is not None:
                f.write('%d %d %d %.3f\n' %
                    (temp_two_element.tr_node, temp_two_element.tl_node, temp_two_element.br_node, 0))
            else:
                break

    f.write('\n')
```

```python

    print("Writing boundary conditions")
    for i in range(4, -1, -1):
        for j in range(5):
            temp_two_element = fe_matrix[i][j]
            if temp_two_element is not None:
                if i == 4 and j != 4:
                    f.write('%d %.3f\n' % (temp_two_element.bl_node, LOW_VOLTAGE))
                elif i == 4 and j == 4:
                    f.write('%d %.3f\n' % (temp_two_element.bl_node, LOW_VOLTAGE))
                    f.write('%d %.3f\n' % (temp_two_element.br_node, LOW_VOLTAGE))
                elif i == 3 and j == 0:
                    f.write('%d %.3f\n' % (temp_two_element.bl_node, LOW_VOLTAGE))
                    f.write('%d %.3f\n' % (temp_two_element.tl_node, LOW_VOLTAGE))
                elif j == 0 and i != 3 and i != 4:
                    f.write('%d %.3f\n' % (temp_two_element.tl_node, LOW_VOLTAGE))
                elif j >= 2 and i <= 1:
                    f.write('%d %.3f\n' % (temp_two_element.tr_node, HIGH_VOLTAGE))
            else:
                break


if __name__ == "__main__":
    fe_vec = [[None for _ in range(5)] for _ in range(5)]
    fe_matrix = Matrix(fe_vec, 5, 5)

    write_mesh(fe_matrix)

    energy = 4 * calc_energy(fe_matrix)
    capacitance = 2 * energy / (HIGH_VOLTAGE * HIGH_VOLTAGE)
    print("Energy enclosed between the conductors is " + str(energy) + "J")
    print("The calculated capacitance is " + str(capacitance) + "F")
```

Listing 2: Finite Element Mesh Input File

Listing 3: MATLAB *File Outputs*

```
1 0.000000 0.000000 0.000000
2 0.020000 0.000000 0.000000
3 0.040000 0.000000 0.000000
4 0.060000 0.000000 0.000000
5 0.080000 0.000000 0.000000
6 0.100000 0.000000 0.000000
7 0.000000 0.020000 0.000000
8 0.020000 0.020000 7.018554
9 0.040000 0.020000 13.651929
10 0.060000 0.020000 19.110684
11 0.080000 0.020000 22.264306
12 0.100000 0.020000 23.256867
13 0.000000 0.040000 0.000000
14 0.020000 0.040000 14.422288
15 0.040000 0.040000 28.478477
16 0.060000 0.040000 40.526503
17 0.080000 0.040000 46.689671
18 0.100000 0.040000 48.498858
19 0.000000 0.060000 0.000000
20 0.020000 0.060000 22.192122
21 0.040000 0.060000 45.313189
22 0.060000 0.060000 67.827178
23 0.080000 0.060000 75.469018
24 0.100000 0.060000 77.359224
25 0.000000 0.080000 0.000000
26 0.020000 0.080000 29.033010
27 0.040000 0.080000 62.754981
28 0.060000 0.080000 110.000000
29 0.080000 0.080000 110.000000
30 0.100000 0.080000 110.000000
31 0.000000 0.100000 0.000000
32 0.020000 0.100000 31.184936
33 0.040000 0.100000 66.673724
34 0.060000 0.100000 110.000000
```