ECSE 543: Numerical Methods
Assignment 2 Report

Wenjie Wei
260685967

November 2, 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 Difference Problem

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

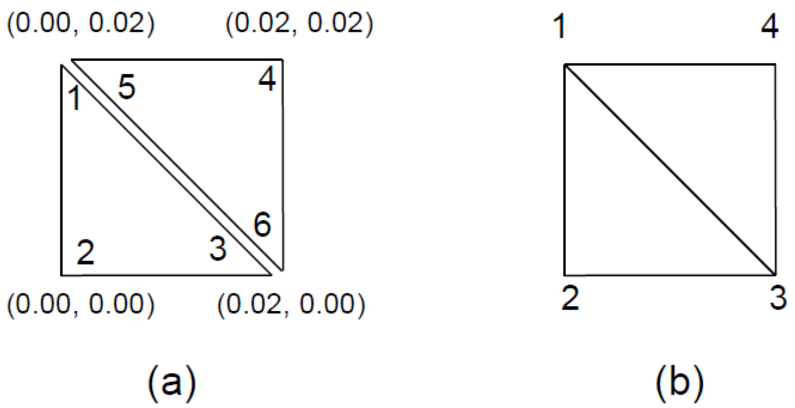

Figure 1: 1st Order Triangular FE Problem

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:

$$U_{dis} = CU_{con}$$

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:

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

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 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

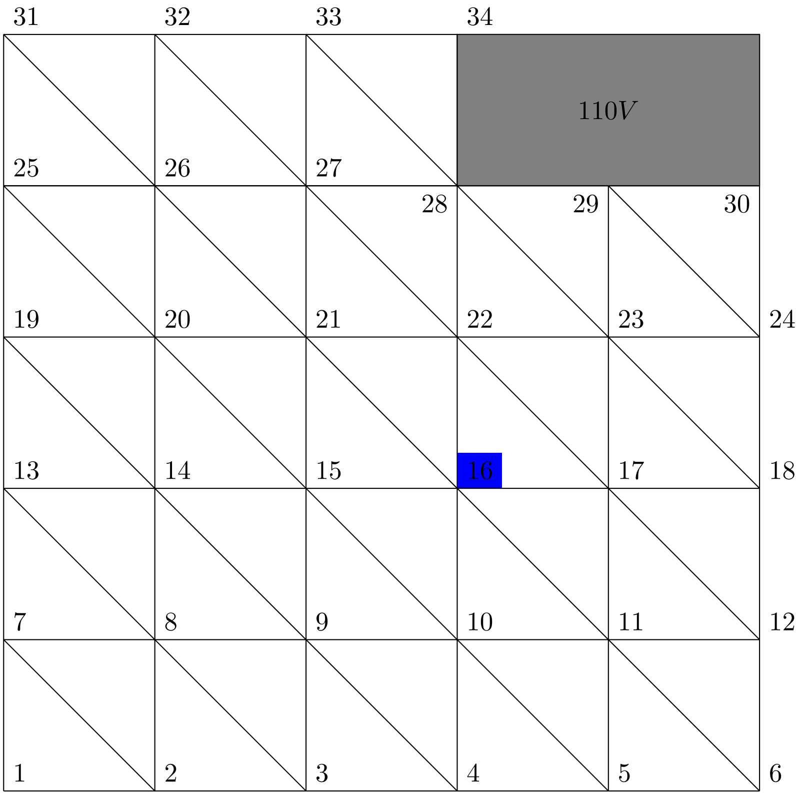

Figure 2: Organization of the Finite Element Mesh

A Code Listings

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

```
from matrix import Matrix
from finite_difference import Node

HIGH_VOLTAGE = 110
LOW_VOLTAGE = 0
SPACING = 0.02
f = open('SIMPLE2Dinput.dat', 'w')


class two_element(object):
    def __init__(self, x, y, bl_node):
        """
        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
```

```python
        self._br_node = bl_node + 1
        self._tr_node = bl_node + 7

        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()

            # 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]

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

    y_coord = 0
    count = 0

    print("Creating the mesh of the finite elements...")
    node_count = 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)
                fe_matrix[i][j] = temp_two_element
                node_count += 1
                count += 1

            x_coord += SPACING
        node_count += 1
        y_coord += SPACING

```

```python
    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')

    print("Writing boundary conditions")
    for i in range(4, -1, -1):
        for j in range(5):
            temp_two_element = fe_matrix[i][j]
```

```python
            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
```

Listing 2: Finite Element Mesh Input File

```
1 0.000 0.000
2 0.020 0.000
3 0.040 0.000
4 0.060 0.000
5 0.080 0.000
6 0.100 0.000
7 0.000 0.020
8 0.020 0.020
9 0.040 0.020
10 0.060 0.020
11 0.080 0.020
12 0.100 0.020
13 0.000 0.040
14 0.020 0.040
15 0.040 0.040
16 0.060 0.040
17 0.080 0.040
18 0.100 0.040
19 0.000 0.060
20 0.020 0.060
21 0.040 0.060
22 0.060 0.060
23 0.080 0.060
24 0.100 0.060
25 0.000 0.080
26 0.020 0.080
27 0.040 0.080
28 0.060 0.080
29 0.080 0.080
30 0.100 0.080
31 0.000 0.100
32 0.020 0.100
33 0.040 0.100
34 0.060 0.100

1 2 7 0.000
2 3 8 0.000
3 4 9 0.000
4 5 10 0.000
5 6 11 0.000
8 7 2 0.000
9 8 3 0.000
10 9 4 0.000
11 10 5 0.000
12 11 6 0.000
7 8 13 0.000
8 9 14 0.000
9 10 15 0.000
10 11 16 0.000
11 12 17 0.000
14 13 8 0.000
```

```
15 14 9 0.000
16 15 10 0.000
17 16 11 0.000
18 17 12 0.000
13 14 19 0.000
14 15 20 0.000
15 16 21 0.000
16 17 22 0.000
17 18 23 0.000
20 19 14 0.000
21 20 15 0.000
22 21 16 0.000
23 22 17 0.000
24 23 18 0.000
19 20 25 0.000
20 21 26 0.000
21 22 27 0.000
22 23 28 0.000
23 24 29 0.000
26 25 20 0.000
27 26 21 0.000
28 27 22 0.000
29 28 23 0.000
30 29 24 0.000
25 26 31 0.000
26 27 32 0.000
27 28 33 0.000
32 31 26 0.000
33 32 27 0.000
34 33 28 0.000

1 0.000
2 0.000
3 0.000
4 0.000
5 0.000
6 0.000
7 0.000
13 0.000
19 0.000
25 0.000
28 110.000
29 110.000
30 110.000
31 0.000
34 110.000
```