

**ECSE 543: Numerical Methods**  
Assignment 2 Report

Wenjie Wei  
260685967

November 2, 2018

# Contents

<b>1</b>	<b>First Order Finite Difference Problem</b>	<b>2</b>
<b>2</b>	<b>Coaxial Cable Electrostatic Problem</b>	<b>4</b>
2.a	The Finite Element Mesh . . . . .	4
2.b	Potential Solved by SIMPLE2D.m . . . . .	5
2.c	Capacitance per Unit Length . . . . .	5
	<b>Appendix A Code Listings</b>	<b>6</b>

# 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.

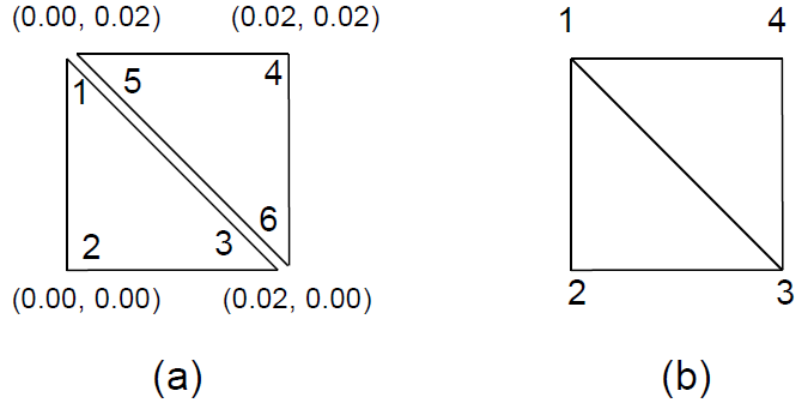


Figure 1: 1st Order Triangular FE Problem

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:

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

and we can derive a general formula for  $\alpha_i$ :

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

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  $\alpha_1, \alpha_2$  and  $\alpha_3$ :

$$\begin{aligned} \nabla \alpha_1 &= \nabla \frac{1}{2A} [(x_2y_3 - x_3y_2) + (y_2 - y_3)x + (x_3 - x_2)y] \\ \nabla \alpha_2 &= \nabla \frac{1}{2A} [(x_3y_1 - x_1y_3) + (y_3 - y_1)x + (x_1 - x_3)y] \end{aligned}$$

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

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

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

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  $\mathbf{S}$  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  $\mathbf{S}$  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:

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

which is also denoted as:

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

Use  $\mathbf{S}_{dis}$  to denote a  $6 \times 6$  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  $\mathbf{S}$  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.

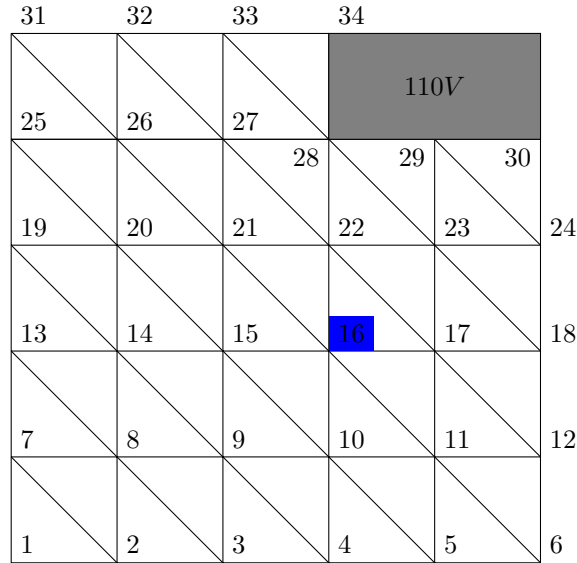


Figure 2: Organization of the Finite Element Mesh

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 L<sup>A</sup>T<sub>E</sub>X.

## 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:

$$E = \frac{1}{2}CV^2 \quad (4)$$

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 total capacitance:

$$E = \frac{1}{2}\varepsilon_0 U_{joint}^T S_{joint} U_{joint} \quad (5)$$

## A Code Listings

*Listing 1: Finite Element Mesh Implementation (finite\_element.py).*

```
1  from matrix import Matrix
2  from finite_difference import Node
3
4  HIGH_VOLTAGE = 110
5  LOW_VOLTAGE = 0
6  SPACING = 0.02
7  f = open('SIMPLE2Dinput.dat', 'w')
8
9
10 class two_element(object):
11     def __init__(self, x, y, bl_node):
12         """
13         This is the constructor of a two-triangle finite element
14         the vertices are numbered from 0 to 5, replacing 1 - 6 in question 1
15
16         :param x: x coord for the bottom-left corner
17         :param y: y coord for the bottom-left corner
18         """
19
20         # vertices are put in the array
21         # vertices 2&5, vertices 0&4 have the same properties
22         self._vertex_array = [Node(0) for _ in range(6)]
23         self._vertex_array[5] = self._vertex_array[2]
24         self._vertex_array[4] = self._vertex_array[0]
25         self._bl_x = x
26         self._bl_y = y
27
28         self._bl_node = bl_node
29         self._tl_node = bl_node + 6
30         self._br_node = bl_node + 1
31         self._tr_node = bl_node + 7
32
33         if (self._bl_x + SPACING) > 0.1 or (self._bl_y + SPACING) > 0.1:
34             raise ValueError("The finite elements cannot exceed the third quadrant!")
35
36         if self._bl_y == 0:
37             # configure node 1
38             self._vertex_array[1].set_fixed()
39             self._vertex_array[1].set_value(LOW_VOLTAGE)
40
41             # configure node 2 and 5
42             self._vertex_array[2].set_fixed()
43             self._vertex_array[2].set_value(LOW_VOLTAGE)
44
45             # configure node 3
46             self._vertex_array[3].set_free()
47
48             if self._bl_x == 0:
49                 # configure node 0 and 4
50                 self._vertex_array[0].set_fixed()
51                 self._vertex_array[0].set_value(LOW_VOLTAGE)
52             else:
53                 self._vertex_array[0].set_free()
54         elif self._bl_x >= 0.06 and self._bl_y == 0.06:
55             # configure node 1
56             self._vertex_array[1].set_free()
57
58             # configure node 2 and 5
59             self._vertex_array[2].set_free()
60
61             # configure node 0 and 4
62             self._vertex_array[0].set_fixed()
63             self._vertex_array[0].set_value(HIGH_VOLTAGE)
64
65             # configure node 3
```

```

66         self._vertex_array[3].set_fixed()
67         self._vertex_array[3].set_value(HIGH_VOLTAGE)
68     elif self._bl_x == 0.04 and self._bl_y == 0.06:
69         # configure node 1
70         self._vertex_array[1].set_free()
71
72         # configure node 2 and 5
73         self._vertex_array[2].set_free()
74
75         # configure node 0 and 4
76         self._vertex_array[0].set_free()
77
78         # configure node 3
79         self._vertex_array[3].set_fixed()
80         self._vertex_array[3].set_value(HIGH_VOLTAGE)
81     elif self._bl_x == 0.04 and self._bl_y == 0.08:
82         # configure node 1
83         self._vertex_array[1].set_free()
84
85         # configure node 2 and 5
86         self._vertex_array[2].set_fixed()
87         self._vertex_array[2].set_value(HIGH_VOLTAGE)
88
89         # configure node 0 and 4
90         self._vertex_array[0].set_free()
91
92         # configure node 3
93         self._vertex_array[3].set_fixed()
94         self._vertex_array[3].set_value(HIGH_VOLTAGE)
95     elif self._bl_x == 0:
96         # configure node 1
97         self._vertex_array[1].set_fixed()
98         self._vertex_array[1].set_value(LOW_VOLTAGE)
99
100        # configure node 0 and 4
101        self._vertex_array[0].set_fixed()
102        self._vertex_array[0].set_value(LOW_VOLTAGE)
103
104        # configure node 3
105        self._vertex_array[3].set_free()
106
107        # configure node 2 and 5
108        self._vertex_array[2].set_free()
109    else:
110        for i in range(6):
111            self._vertex_array[i].set_free()
112
113    def print_two_element(self):
114        for i in range(6):
115            print("Vertex " + str(i) + " has value " + str(self._vertex_array[i].value) + ", free node: "
116                  + str(self._vertex_array[i].is_free))
117
118    @property
119    def bl_x(self):
120        return self._bl_x
121
122    @property
123    def bl_y(self):
124        return self._bl_y
125
126    @property
127    def bl_node(self):
128        return self._bl_node
129
130    @property
131    def tl_node(self):
132        return self._tl_node
133
134    @property
135    def br_node(self):

```



```

136         return self._br_node
137
138     @property
139     def tr_node(self):
140         return self._tr_node
141
142     @property
143     def vertex(self, i):
144         return self._vertex_array[i]
145
146 if __name__ == "__main__":
147     fe_vec = [[None for _ in range(5)] for _ in range(5)]
148     fe_matrix = Matrix(fe_vec, 5, 5)
149
150     y_coord = 0
151     count = 0
152
153     print("Creating the mesh of the finite elements...")
154     node_count = 1
155     for i in range(4, -1, -1):
156         x_coord = 0
157         for j in range(5):
158             if x_coord >= 0.06 and y_coord == 0.08:
159                 break
160             else:
161                 temp_two_element = two_element(x_coord, y_coord, node_count)
162                 fe_matrix[i][j] = temp_two_element
163                 node_count += 1
164                 count += 1
165
166             x_coord += SPACING
167             node_count += 1
168             y_coord += SPACING
169
170     print("Finite elements created: " + str(count * 2))
171
172     # Now write the input file for SIMPLE2D.m
173     print("Writing node information...")
174     # write the bottom row
175     i = 4
176     for j in range(5):
177         temp_two_element = fe_matrix[i][j]
178         f.write('%d %.3f %.3f\n' % (temp_two_element.bl_node, temp_two_element.bl_x,
↪ temp_two_element.bl_y))
179         if j == 4:
180             f.write('%d %.3f %.3f\n' % (temp_two_element.br_node,
181                                     temp_two_element.bl_x + SPACING, temp_two_element.bl_y))
182
183     # write the general rows
184     for i in range(4, -1, -1):
185         for j in range(5):
186             temp_two_element = fe_matrix[i][j]
187             if temp_two_element is not None:
188                 if i != 0 and j != 4:
189                     f.write('%d %.3f %.3f\n' %
190                             (temp_two_element.tl_node, temp_two_element.bl_x, temp_two_element.bl_y
↪ + SPACING))
191                 elif i != 0 and j == 4:
192                     f.write('%d %.3f %.3f\n' %
193                             (temp_two_element.tl_node, temp_two_element.bl_x, temp_two_element.bl_y
↪ + SPACING))
194                     f.write('%d %.3f %.3f\n' %
195                             (temp_two_element.tr_node, temp_two_element.bl_x + SPACING,
196                             temp_two_element.bl_y + SPACING))
197                 else:
198                     if j != 2:
199                         f.write('%d %.3f %.3f\n' %
200                                 (temp_two_element.tl_node, temp_two_element.bl_x, temp_two_element.bl_y
↪ + SPACING))
201                     else:

```

```

202         f.write('%d %.3f %.3f\n' %
203                 (temp_two_element.tl_node,
204                  temp_two_element.bl_x, temp_two_element.bl_y + SPACING))
205         f.write('%d %.3f %.3f\n' %
206                 (temp_two_element.tr_node, temp_two_element.bl_x + SPACING,
207                  temp_two_element.bl_y + SPACING))
208     else:
209         break
210
211 f.write('\n')
212 # Now write the triangle connection
213 print("Writing triangle information...")
214 for i in range(4, -1, -1):
215     for j in range(5):
216         temp_two_element = fe_matrix[i][j]
217         if temp_two_element is not None:
218             f.write('%d %d %d %.3f\n' %
219                     (temp_two_element.bl_node, temp_two_element.br_node, temp_two_element.tl_node, 0))
220         else:
221             break
222     for j in range(5):
223         temp_two_element = fe_matrix[i][j]
224         if temp_two_element is not None:
225             f.write('%d %d %d %.3f\n' %
226                     (temp_two_element.tr_node, temp_two_element.tl_node, temp_two_element.br_node, 0))
227         else:
228             break
229
230 f.write('\n')
231
232 print("Writing boundary conditions")
233 for i in range(4, -1, -1):
234     for j in range(5):
235         temp_two_element = fe_matrix[i][j]
236         if temp_two_element is not None:
237             if i == 4 and j != 4:
238                 f.write('%d %.3f\n' % (temp_two_element.bl_node, LOW_VOLTAGE))
239             elif i == 4 and j == 4:
240                 f.write('%d %.3f\n' % (temp_two_element.bl_node, LOW_VOLTAGE))
241                 f.write('%d %.3f\n' % (temp_two_element.br_node, LOW_VOLTAGE))
242             elif i == 3 and j == 0:
243                 f.write('%d %.3f\n' % (temp_two_element.bl_node, LOW_VOLTAGE))
244                 f.write('%d %.3f\n' % (temp_two_element.tl_node, LOW_VOLTAGE))
245             elif j == 0 and i != 3 and i != 4:
246                 f.write('%d %.3f\n' % (temp_two_element.tl_node, LOW_VOLTAGE))
247             elif j >= 2 and i <= 1:
248                 f.write('%d %.3f\n' % (temp_two_element.tr_node, HIGH_VOLTAGE))
249         else:
250             break

```

*Listing 2: Finite Element Mesh Input File*

```

1  1 0.000 0.000
2  2 0.020 0.000
3  3 0.040 0.000
4  4 0.060 0.000
5  5 0.080 0.000
6  6 0.100 0.000
7  7 0.000 0.020
8  8 0.020 0.020
9  9 0.040 0.020
10 10 0.060 0.020
11 11 0.080 0.020
12 12 0.100 0.020
13 13 0.000 0.040
14 14 0.020 0.040
15 15 0.040 0.040
16 16 0.060 0.040
17 17 0.080 0.040
18 18 0.100 0.040
19 19 0.000 0.060
20 20 0.020 0.060
21 21 0.040 0.060
22 22 0.060 0.060
23 23 0.080 0.060
24 24 0.100 0.060
25 25 0.000 0.080
26 26 0.020 0.080
27 27 0.040 0.080
28 28 0.060 0.080
29 29 0.080 0.080
30 30 0.100 0.080
31 31 0.000 0.100
32 32 0.020 0.100
33 33 0.040 0.100
34 34 0.060 0.100
35
36 1 2 7 0.000
37 2 3 8 0.000
38 3 4 9 0.000
39 4 5 10 0.000
40 5 6 11 0.000
41 8 7 2 0.000
42 9 8 3 0.000
43 10 9 4 0.000
44 11 10 5 0.000
45 12 11 6 0.000
46 7 8 13 0.000
47 8 9 14 0.000
48 9 10 15 0.000
49 10 11 16 0.000
50 11 12 17 0.000
51 14 13 8 0.000
52 15 14 9 0.000
53 16 15 10 0.000
54 17 16 11 0.000
55 18 17 12 0.000
56 13 14 19 0.000
57 14 15 20 0.000
58 15 16 21 0.000
59 16 17 22 0.000
60 17 18 23 0.000
61 20 19 14 0.000
62 21 20 15 0.000
63 22 21 16 0.000
64 23 22 17 0.000
65 24 23 18 0.000
66 19 20 25 0.000
67 20 21 26 0.000

```

```

68 21 22 27 0.000
69 22 23 28 0.000
70 23 24 29 0.000
71 26 25 20 0.000
72 27 26 21 0.000
73 28 27 22 0.000
74 29 28 23 0.000
75 30 29 24 0.000
76 25 26 31 0.000
77 26 27 32 0.000
78 27 28 33 0.000
79 32 31 26 0.000
80 33 32 27 0.000
81 34 33 28 0.000
82
83 1 0.000
84 2 0.000
85 3 0.000
86 4 0.000
87 5 0.000
88 6 0.000
89 7 0.000
90 13 0.000
91 19 0.000
92 25 0.000
93 28 110.000
94 29 110.000
95 30 110.000
96 31 0.000
97 34 110.000

```

*Listing 3: MATLAB File Outputs*

```

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

```