Chapter 1

METHODOLOGY and MODEL EQUATION

1.1 Coding Implementation

1	1	0	0	0	0	0	0
2	2	2	0	0	0	0	0
3	3	4	0	0	0	0	0
4	4	6	0	0	0	0	0
5	5	8	0	0	0	0	0
6	6	8	2.918	nan	nan	0.2	-0.1
7	7	6	2.1838	nan	nan	0	-0.1
8	8	4	1.4559	nan	nan	0	-0.1
9	9	2	0.7279	nan	nan	0	-0.1

Figure 1.1: node.csv

1	1	2	1	1	pi*0	1	2
2	2	2	1	1	pi*0	2	3
3	3	2	1	1	pi*0	3	4
4	4	2	1	1	pi*0	4	5
5	5	2.9118	1	1	pi*0.5	5	6
6	6	2.1284	1	1	pi*0.11111	7	6
7	7	2.1284	1	1	pi*0.11111	8	7
8	8	2.1284	1	1	pi*0.11111	9	8
9	9	2.1284	1	1	pi*0.11111	1	9
10	10	0.7279	1	1	pi*0.5	2	9
11	11	2.4738	1	1	pi*0.2	2	8
12	12	1.4559	1	1	pi*0.5	3	8
13	13	2.4738	1	1	pi*0.8	4	8
14	14	2.1838	1	1	pi*0.5	4	7
15	15	3.5325	1	1	pi*0.311111	4	6

Figure 1.2: element.csv

To feed the data to our program, we came up with the idea of using csv files so as to make the program available for general problems and for a easy to input interface.

Figure 1.1 shows the input format of nodes where each row represents a node and its attribues: node number, position on x and y-axis, its displacement on x and y-axis and the force applied on x and y-axis.

Similarly, Figure 1.2 shows the input format of the elements where each row represents a element and its attribues: element number, length , cross-section area , young's modulus , angle with the global x-axis and the nodes that it connects.

```python
def globalstiff(self):
    # dimension of global matrix
    dim = 2 * len(self.node_list)
    eles = self.ele_list
    # generate and store the element-wise stiffness matrices

    GK = np.zeros((dim, dim))

    for e in eles:
        i = 2 * e.node_a.num - 2
        j = 2 * e.node_a.num - 1
        k = 2 * e.node_b.num - 2
        l = 2 * e.node_b.num - 1
        e_stiff = e.stiff()

        index = [i, j, k, l]
        index2d = [(a, b) for a in index for b in index]
        d = {i: 0, j: 1, k: 2, l: 3}
        for p, q in index2d:
            GK[p][q] = GK[p][q] + e_stiff[d[p]][d[q]]
    self.GK = GK
```

Figure 1.3: Forming the global stiffness matrix

Formation of global stiffness matrix was a challenge and reducing the code to faster execution and simplicity was even bigger of a task. Finally the method applied is shown in the Figure 1.3.

The list of elements is iterated throughout for accessing each element. Then with the logic of how the values in local stiffness matrix is appended to the global matrix, we came up with an efficient algorithm that fits well to our purpose as well as works on all such general problems with slightest of tweaks.

```
del_row = []
index_list = list(range(0, 2 * len(self.node_list)))
for i in range(0, 2 * len(self.node_list)):
    if dis_list[i] == 0:
        del_row.append(i)
        index_list.remove(i)
GK_dis = np.delete(GK_dis, del_row, 0)
GK_dis = np.delete(GK_dis, del_row, 1)
f_dis = np.delete(f_dis, del_row, 0)

ans_dis = np.linalg.solve(GK_dis, f_dis)
```

Figure 1.4: Solution

After generating global stiffness matrix we now had 3 variables: the stiffness matrix , the displacement and the force vector. For simplifying the calculations and faster computation, for n^{th} element being 0 in the displacement vector we reduced the global matrix by removing coressponding n^{th} row and columns (Figure 1.4). After reduction simple linear algebra has been applied to find out the unknown/missing values of displacement of each node.

```
    for ele in self.ele_list:
        t.goto(ele.node_a.pos_x, ele.node_a.pos_y)
        t.pendown()
        t.goto(ele.node_b.pos_x, ele.node_b.pos_y)
        t.penup()

    t.speed(0)
    for node in self.node_list:
        t.penup()
        t.goto(node.pos_x, node.pos_y)
        t.pendown()
        t.dot(15, "red")
```

Figure 1.5:

```
for ele in self.ele_list:
    t.goto((ele.node_a.pos_x + ele.node_a.dis_x), (ele.node_a.pos_y + ele.node_a.dis_y))
    t.pendown()
    t.goto((ele.node_b.pos_x + ele.node_b.dis_x), (ele.node_b.pos_y + ele.node_b.dis_y))
    t.penup()

t.speed(0)
for node in self.node_list:
    t.penup()
    t.goto((node.pos_x + node.dis_x), (node.pos_y + node.dis_y))
    t.pendown()
    t.dot(15, "yellow")
turtle.Screen().exitonclick()
```

Figure 1.6:

For the last and final part of visualizing our problem and the visualizing the solution so as to make each bit of data understandable we used the Turtle module of Python. Initially we draw our problem provided the position of the nodes and elements connecting them (Figure 1.5). Then from the displacement data of each node we get after the computation, the same algorithm is applied to draw the final result/ shape of the truss. This time the position of node is the sum of its original position and its displacement in x and y-axis (Figure 1.6). Different tools have been used to make the visualized problem and result understandable.