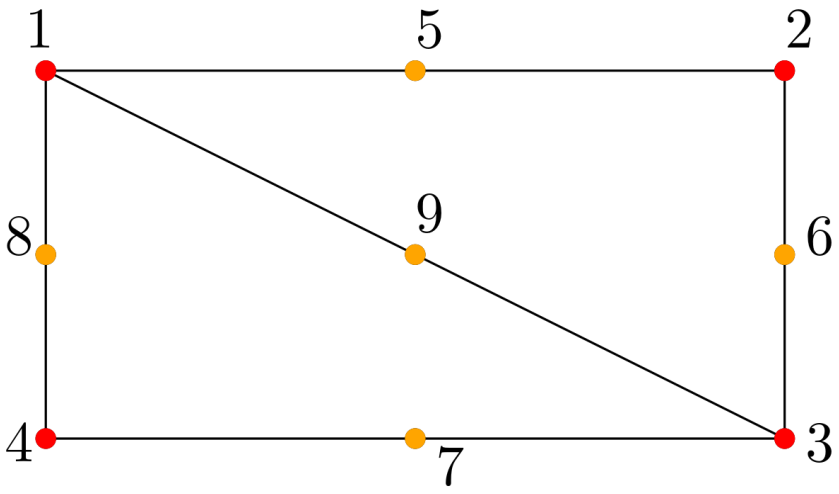
How to structure your FE code



Consider this rectangle where we have 4 main nodes, in red, and we decide to put 2 elements per beam, so that we add the orange nodes. We need several things in the code:

- **elemList**. It contains, for each element, the start node and end node of the element. At first, in this case, we basically need to write the main beams of our structure as

[1 2;
2 3;
3 4;
4 1;
1 3]

Then, we simply add the nodes in between in a loop.

elemList =
[1 5;
5 2;
2 6;
6 3;
3 7;
7 4;
4 8;
8 1;
1 9;
9 3]

- **dofList**. It contains the list of all the degrees of freedom in the model. In our case, it looks like this:

dofList =
[1 2 3 4 5 6;

7 8 9 10 11 12;
13 14 15 16 17 18;
19 20 21 22 23 24;
25 26 27 28 29 30;
31 32 33 34 35 36;
37 38 39 40 41 42;
43 44 45 46 47 48;
49 50 51 52 53 54]

Basically, we will need as many lines as the number of nodes in our model and this is easily done in a loop.

- **nodeList**. It is a list of all the coordinates for each node. We first need to define the main nodes 1, 2, 3 and 4. In this example, let us consider the coordinates:

[0 0 0;
2 0 0;
2 -1 0;
0 -1 0]

Then, as we decided to have intermediate nodes, as a function of the number of elements per beam, we just add the coordinates in between and put them in the same order as our numbering. This can be done in a loop. We finally get:

nodeList =
[0 0 0;
2 0 0;
2 -1 0;
0 -1 0;
1 0 0;
2 -0.5; 0;
1 -1 0;
0 -0.5 0;
1 -0.5 0]

- **locel**. It is a matrix which says which dof is involved for each element. We can create it easily using our elemList. For each element *i*, we need to access the line index of the dofList given by the i^{th} line and corresponding column of the elemList, while taking all the columns. In other words, the i^{th} line of locel is given by locel(i, :) = [dofList(elemList(i, 1), :), dofList(elemList(i, 2), :)].

Here, the result is:

locel =
[1 2 3 4 5 6 25 26 27 28 29 30;
25 26 27 28 29 30 7 8 9 10 11 12;
7 8 9 10 11 12 31 32 33 34 35 36;
31 32 33 34 35 36 13 14 15 16 17 18;
13 14 15 16 17 18 37 38 39 40 41 42;
37 38 39 40 41 42 19 20 21 22 23 24;

19 20 21 22 23 24 43 44 45 46 47 48;
43 44 45 46 47 48 1 2 3 4 5 6;
1 2 3 4 5 6 49 50 51 52 53 54;
49 50 51 52 53 54 13 14 15 16 17 18]

- If needed (for instance, for the elementary matrices), we can determine the length of each element *i* by first accessing the coordinates of the nodes of this element, in a similar fashion as what was done for the locel, but this time using nodeList instead of dofList. The length is later found by using the formula for the Euclidian distance.

Please note that all these lists should be done **once and for all** before any assembly or computation. Then, the assembly can be done thanks the elementary matrices and the locel using a loop on every element.

Example for element *i*: K(locel(i, :), locel(i, :) = K(locel(i, :), locel(i, :) + K_elementary

Please note that we should assemble the **ALREADY ROTATED** elementary matrices, so that they are not the same for each element. To this end, we need to set the local axes of each beam in order to be consistent with their inertia properties. We first need to define two points, which are the start node and end node of the beam. Then, a third point is needed to define the local axes. In the context of circular beams, we do not care about the coordinate of this third point. Indeed, when the cross-section is circular, the inertias I_yy and I_zz are the same regardless of the radial direction on the cross-section, so that the y- and z-axes can be rotated freely on the cross-section if the x-axis is correctly defined between point 1 and 2.

Finally, we might need to add boundary conditions. Let us consider that, just for the sake of our example, we clamp node 1, meaning that all the degrees of freedom for this node are blocked, and we put a roller in the x-direction for node 4. Basically, we need to remove the lines and columns in the mass and stiffness matrices for the dofs we wish to constraint. Here, we will remove lines and columns 1 2 3 4 5 6 because of the campling of node 1, and we will also remove lines and columns 20 and 21, which correspond to the number of the dofs Y and Z for node 4 (we can check with the dofList). To remove the i^{th} line of a matrix A, we can use A(i, :) = []. We can of course do the same for a column. I would advise to begin from the higher number and to finish with the lowest when doing this in a loop (so from 21 to 1). This will avoid the line numbers to remove to constantly change.