Delft University of Technology
Faculty of Civil Engineering and Geosciences

Bachelor Thesis
CTB3000

On the accuracy and convergence of the finite element method for truss and frame structures

Author:
Joep van Oostrum (4143787)

Supervisors:
Dr. R. Al-Khoury
Dr.ir. F.P. van der Meer

August 28, 2016

Contents

Chapter 1

Introduction

During applied mechanics courses for undergraduates, the words 'finite element' will be mentioned if a structural problem is so complex that it cannot be solved analytically or simply because it would require too much time to solve it by hand. The lecturer refers to a software package and explains how to work with it. Unfortunately, during these courses, no time is spend on the inner workings of this software. How does the program know, after virtually assembling the structural members and specifying the load, how the structure mechanically behaves? Another term that is mentioned in conjunction with the finite element method is 'approximation'; the method is not an exact method, only approximated values are obtained. The accuracy depends on the amount of elements and the type of element.

In this thesis the following question will be answered: what is the influence of the mesh size and the element type used in a finite element protocol for trusses and frames, on the accuracy of the displacements and forces and how does it affect the computation time? The goal of this thesis is also to get a better comprehension of the inner workings of a finite element program with a structural application. All this can be achieved by developing, analysing and using a finite element program created in the Matlab environment. To do this, prior literary study is required. Use is made of [1] and sub-paragraph 5.1.1 of [6].

A total of four programs are developed, one for trusses in 2D space, one for trusses in 3D space and two for frames in 2D space. Frames can namely be described by so-called linear and quadratic elements. What that means will be explained in paragraph 2.2. The structural models will be discretized and solved within the framework of Matlab matrix solutions.

The created finite element programs will be described and explained in the second chapter. After reading that part, it will be clear how the programs work and how to use them. In the chapter that follows, three elementary frame structures will be analysed by modelling them with different amount of elements and different element types. A comparison will be made with the analytical solution. Other aspects of the programs and how they compare to commercial finite element software is the focus of chapter 4. A geometrically complex structure like an arch will modelled by beam elements and analysed in the fifth chapter. The findings will be summarised and the central question will answered in chapter 6, the conclusion.

Chapter 2

Description of the programs

In this chapter the four programs are described, starting with the Matlab script for trusses in two dimensional space. Each program, i.e. for trusses in 2D and 3D space, for frames in 2D space modelled by linear elements and for frames in 2D space modelled by quadratic elements can be considered in this order as a stepping stone for the successive program. For that reason the scripts are discussed in this particular order. To become familiar with the input and output, a truss example will be used in the description of the programs for 2D and 3D trusses.

2.1 Trusses

2.1.1 Input

To understand the input file which describes the structure, an example is used as shown in figure 2.1.

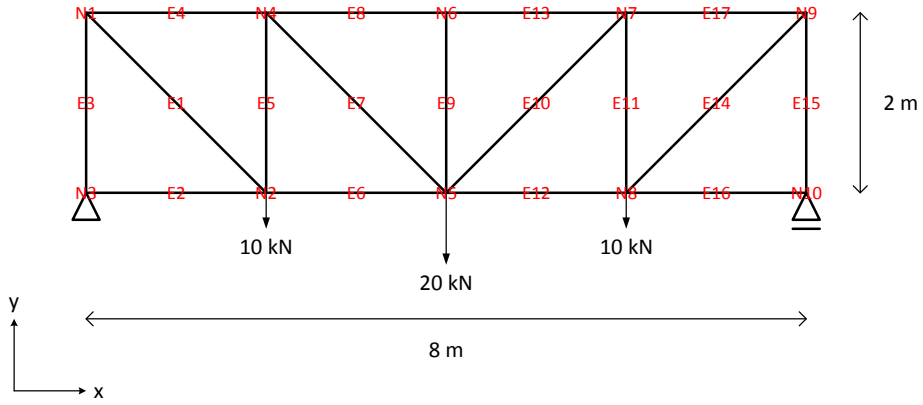

Figure 2.1: Truss example

A couple of things can be observed from this figure. First of all, the global coordinate system consists of an x- and y-axis which are oriented to the right and upwards respectively. The nodes and elements are centrically labelled in

red and the structure is suspended in node N3 by a hinge and in node N10 by a roller. As for the point loads, these are indicated with arrows and grab onto the structure in node N2, N5 and N8. To finish this observation, it is pointed out that in a truss structure the nodes between the bars are pin joints and only on these joints point loads can be applied.

This structure is described by a spreadsheet file. Figure 2.2 shows how this is done.

	A	B
1	0	2000
2	2000	0
3	0	0
4	2000	2000
5	4000	0
6	4000	2000
7	6000	2000
8	6000	0
9	8000	2000
10	8000	0
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

	A	B	C	D
1	1	2	210000	2000
2	2	3	210000	2000
3	3	1	210000	2000
4	1	4	210000	2000
5	4	2	210000	2000
6	2	5	210000	2000
7	5	4	210000	2000
8	4	6	210000	2000
9	6	5	210000	2000
10	5	7	210000	2000
11	7	8	210000	2000
12	8	5	210000	2000
13	7	6	210000	2000
14	8	9	210000	2000
15	9	10	210000	2000
16	10	8	210000	2000
17	9	7	210000	2000
18				
19				
20				

	A	B
1	nan	0
2	nan	0
3	nan	0
4	nan	-10000
5	0	nan
6	0	nan
7	nan	0
8	nan	0
9	nan	0
10	nan	-20000
11	nan	0
12	nan	0
13	nan	0
14	nan	0
15	nan	0
16	nan	-10000
17	nan	0
18	nan	0
19	nan	0
20	0	nan

nodes elements boundary_conditions

Figure 2.2: Descriptive file

As can be seen at the bottom, the file consists of three worksheets. The sheet 'nodes' is presented at the left and describes the position of the nodes in the x-y-plane; column A lists the x-coordinates and column B the y-coordinates, both in millimeters. The row index values correspond to the node numbering. This can be checked by looking at the node numbering in figure 2.1.

The middle worksheet contains the element information of which the first two columns describe from which node (column A) to which node (column B) the bar element goes. Again, the row index values correspond to the numbering. See figure 2.1. Column C and D contain the elastic moduli of the used material in $\mathrm{N/mm^2}$ and the cross-sectional areas of the bars in $\mathrm{mm^2}$ respectively.

Boundary conditions need to be given as well in order to let the program know which nodes has which degrees of freedom and at which node a point load is exerted. This is presented in the worksheet at the right where the first column describes the degrees of freedom; each successive row pair describes one node. The first row of each pair denotes the translation in the x-direction and the second row the translation in the y-direction. When these translations are unknown the nan value is inserted into the corresponding cell. In case of the suspensions, the degrees of freedom are (partly) known; the hinge at N3 is constrained in both directions and the roller at N10 only in the y-direction, this

is described by the null value. Column B shows at which nodes there are known and unknown point loads applied in units of newton and has the same structure as column A. Unknown point loads are present at the suspension points, i.e. the reaction forces, and known point loads are present at the other nodes of which the forces acting on node N2, N5 and N8 are pointing in the negative y-direction.

How this input is processed is explained in the following paragraph.

2.1.2 Trusses2d.m

The program, divided into ten sections, will be discussed from top to bottom, starting with the first section. Here the input data will be loaded and empty vectors and matrices will be set up. The spreadsheet which describes the structure is recognised in line 3 and the different worksheets are divided into separate matrices. Note the name of the descriptive file, the last number represents the amount of elements and the number before that represents the order of the interpolation polynomial. Because the number of nodes and elements will be used extensively throughout the program in different loops, these are set up in line 8 and 9 as numnode and numelem respectively. As for the empty matrix and vectors, these will be filled and used during the run of the program.

Listing 2.1: Section 1

```
1  %% SECTION 1
2  % Load input and setup empty vectors and matrices
3
4  structure = 'beam_truss_01_17.xlsx';
5
6  nodes = xlsread(structure,1);
7  elements = xlsread(structure,2);
8  boundaries = xlsread(structure,3);
9  numnode = size(nodes,1);
10 numelem = size(elements,1);
11
12 k = zeros(2*numnode,2*numnode); %global stiffness matrix
13 U = zeros(2*numnode,1); %global displacements of the nodes
14 epsilon = zeros(numelem,1);
15 sigma = zeros(numelem,1);
16 F_int = zeros(numelem,1);
```

In the next section the global stiffness matrix is constructed. This is done by adding the global stiffness matrix of each element to the empty matrix as defined in line 11 with the help of a loop which will be iterated numelem times. In the loop, the first thing that will be done is finding the x- and y-coordinate of both node i and node j. These values will be used as an input for the computation of the element length L and the orientation parameters l and m. To clarify the latter two, figure 2.3 presents the reasoning behind the derivation of the transformation matrix $[\lambda]$.

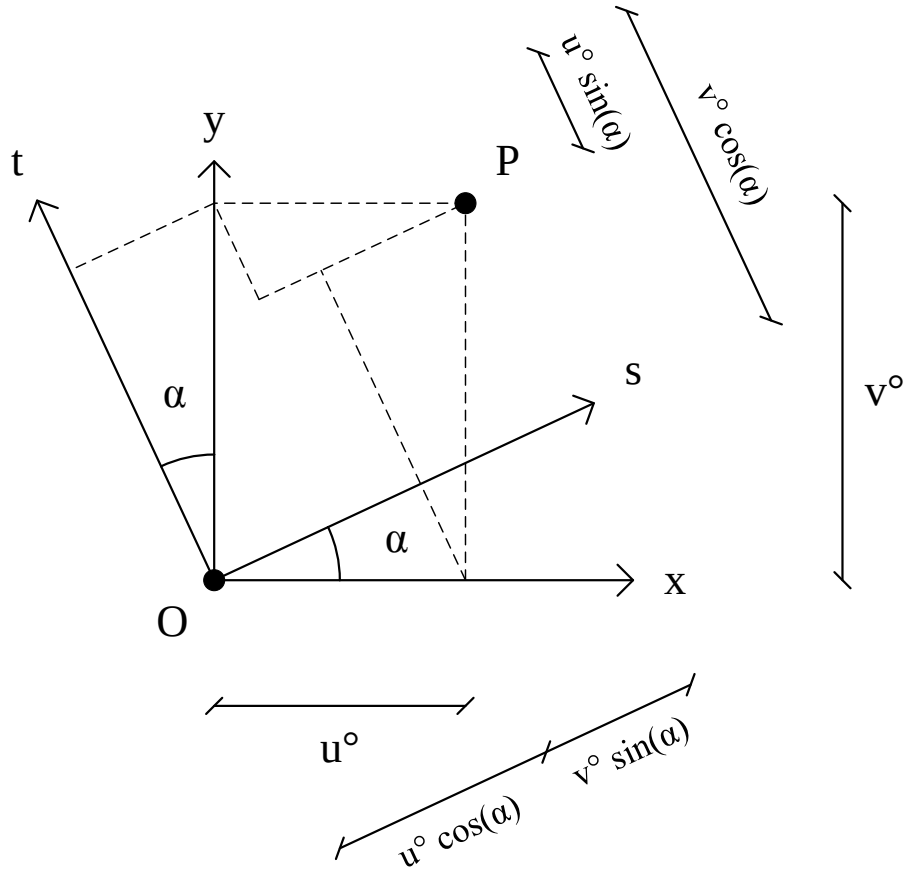

Figure 2.3: Node defined in global and local coordinate system

Because not every bar is parallel to the global coordinate system, a transformation needs to be performed in order to let the deformations of the bar be related to the local s-t coordinate system. The following can be derived from figure 2.3.

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \begin{Bmatrix} u^\circ \\ v^\circ \end{Bmatrix} = \begin{bmatrix} l & m \\ -m & l \end{bmatrix} \begin{Bmatrix} u^\circ \\ v^\circ \end{Bmatrix} \tag{2.1}$$

The circle as superscript means that the considered parameter is related to the global coordinate system. Because bar elements can solely undergo axial deformations, only the upper row of equation (2.1) is relevant. Dealing with two nodes per element, the following transformation matrix applies.

$$[\lambda] = \begin{bmatrix} l & m & 0 & 0 \\ 0 & 0 & l & m \end{bmatrix} \tag{2.2}$$

The entries vector is constructed next. This vector tells the program which entries of the global stiffness matrix need to be computed. In order to do this systematically the relation between the structure of the global stiffness matrix and the structure of the global displacement vector is used. Take element E1 for example. This element goes from node N1 to N2 and the resultant entries

vector is $[1;2;3;4]$. Equation (2.3) presents the general case for the global element displacement vector. Note the indices.

$$\left\{U^{\circ}\right\} = \begin{Bmatrix} u_i^{\circ} \\ v_i^{\circ} \\ u_j^{\circ} \\ v_j^{\circ} \end{Bmatrix} = \begin{Bmatrix} u_{2i-1}^{\circ} \\ u_{2i}^{\circ} \\ u_{2j-1}^{\circ} \\ u_{2j}^{\circ} \end{Bmatrix} \tag{2.3}$$

The final act within this loop is to assign numerical values to the according entries of the stiffness matrix $\left[k\right]$ of the entire structure. This matrix is defined by equation (2.4) and for the derivation reference is made to paragraph 5.2 (p.61-66) of [1].

$$\left[k\right] = \sum_{e=1}^{E} \left[\int_L \left[B^{(e)}\right]^T \left[D^{(e)}\right] \left[B^{(e)}\right] dx \right] = \sum_{e=1}^{E} \left[k^{(e)}\right] \tag{2.4}$$

For the upper limit E of the summation, the number of elements (i.e. numelem) is substituted. It will be discussed later on how the element stiffness matrix $\left[k^{(e)}\right]$ needs to be transformed by the matrix $\left[\lambda\right]$ in order to make it applicable for every arbitrary orientation of the bar element. First the matrix $\left[B^{(e)}\right]$ and $\left[D^{(e)}\right]$ will be defined.

The axial displacement field u is described by a first order polynomial $u = a + bx$ which can be expressed in the unknown displacements u_i and u_j by solving the equation for a and b. As a result u can be formulated as follows.

$$u = N_i u_i + N_j u_j \tag{2.5}$$

With $N_i = (x_j - x)/L$ and $N_j = (x - x_i)/L$. These functions are called shape functions and can be expressed in vector-matrix form: $u = \left[N^{(e)}\right] \left\{U^{(e)}\right\}$. $\left[B^{(e)}\right]$ is the derivative of $\left[N^{(e)}\right]$ to x, because the strain ε is defined as du/dx. Equation (2.6) shows the result.

$$\left[B^{(e)}\right] = \frac{1}{L} \begin{bmatrix} -1 & 1 \end{bmatrix} \tag{2.6}$$

Where the matrix $\left[B^{(e)}\right]$ holds the compatibility information (i.e. the relation between displacements and strains), the matrix $\left[D^{(e)}\right]$ holds the constitutive information:

$$\left[D^{(e)}\right] = EA \tag{2.7}$$

Substitution of (2.6) and (2.7) into (2.4) and assuming that the elastic modulus and the cross-sectional area are constant along the entire length of the bar, the stiffness matrix for the element can be defined as follows.

$$\left[k^{(e)}\right] = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.8}$$

To let this equation be compatible with any arbitrary orientation of the bar, the element stiffness matrix needs to be pre multiplied by the transpose of the lambda matrix and post multiplied by the lambda matrix. This is because the local and global displacement vector can be translated into each other with $\left\{U\right\} = \left[\lambda\right] \left\{U^{\circ}\right\}$ and the local and the global force vector with $\left\{F\right\} = \left[\lambda\right] \left\{F^{\circ}\right\}$.

By substituting these two equations into $\{F\} = [k]\{U\}$, followed by a rearrangement, one obtains: $[k^{\circ(e)}] = [\lambda]^T [k^{(e)}] [\lambda]$. Note that the stiffness matrix of the entire structure can now be called the 'global' stiffness matrix: $[k] \Rightarrow [k^{\circ}]$.

Listing 2.2: Section 2

```
18 %% SECTION 2
19 % Constructing k
20
21 for n = 1:numelem
22     x_i = nodes(elements(n,1),1);
23     y_i = nodes(elements(n,1),2);
24     x_j = nodes(elements(n,2),1);
25     y_j = nodes(elements(n,2),2);
26
27     L = sqrt((x_j-x_i)^2+(y_j-y_i)^2);
28     l = (x_j-x_i)/L;
29     m = (y_j-y_i)/L;
30
31     lambda = [l m 0 0; 0 0 l m];
32
33     entries = [2*elements(n,1)-1; 2*elements(n,1);...
34                2*elements(n,2)-1; 2*elements(n,2)];
35
36     k(entries,entries) = k(entries,entries)+elements(n,3)*...
37                          elements(n,4)/L*transpose(lambda)*...
38                          [1 -1; -1 1]*lambda;
39 end
```

As described by the descriptive file (figure 2.2), a distinction can be made between two kinds of known and unknown quantities; the known and unknown degrees of freedom and the known and unknown nodal forces. The knowns are either denoted by the null or a non-null value and the unknowns are denoted by the nan value. To solve the finite element equations, the program needs to recognise this distinction. That is done in the third section.

From line 44 down to 57, the code constructs an index vector of which the first part contains the row index values of the unknown degrees of freedom. These index values correspond to the known nodal forces (i.e. the external forces) as well. The second part contains the row index values of the known degrees of freedom/unknown nodal forces.

By using this index vector the global stiffness matrix, which is constructed in section 2, can be rearranged and partitioned into four sub-matrices.

Listing 2.3: Section 3

```
41 %% SECTION 3
42 % Rearranging and partitioning k
43 % [k_11 k_12; k_21 k_22]*{U_1; U_2} = {F_1; F_2}
44 % U_1 and F_2 -> unknown degrees of freedom and unknown nodal forces
45 % U_2 and F_1 -> known degrees of freedom and known nodal forces
46
47 vector_1 = zeros(2*numnode,1);
48 vector_2 = zeros(2*numnode,1);
49 counter = 0;
50 for n = 1:2*numnode
```

```
51      if boundaries(n,1) ~= 0
52          vector_1(n) = n;
53          counter = counter+1;
54      else
55          vector_2(n) = n;
56      end
57  end
58  vector_1 = vector_1(vector_1~=0);
59  vector_2 = vector_2(vector_2~=0);
60  vector_1(counter+1:2*numnode) = vector_2; %index vector
61
62  matrix = k(vector_1,vector_1); %rearranged stiffness matrix
63
64  k_11 = matrix(1:counter,1:counter);
65  k_12 = matrix(1:counter,counter+1:2*numnode);
66  k_21 = matrix(counter+1:2*numnode,1:counter);
67  k_22 = matrix(counter+1:2*numnode,counter+1:2*numnode);
```

These sub-matrices will be used in the section where the actual solution of the finite element equations will be computed, i.e. section 4. The final assembly of the equations are shown in (2.9). Here $\{U_1\}$ and $\{F_2\}$ are the unknowns and $\{U_2\}$ and $\{F_1\}$ are the knowns. The former two are computed in line 72 and 73 and the latter two are constructed in line 70 and 71. What remains is replacing the nan values as listed in the worksheet 'boundary_conditions' with the found solutions accordingly. The result is a vector U with the global nodal displacements and a vector F_ext with the external forces, i.e. the nodal forces.

$$\begin{bmatrix} [k_{11}] & [k_{12}] \\ [k_{21}] & [k_{22}] \end{bmatrix} \begin{Bmatrix} \{U_1\} \\ \{U_2\} \end{Bmatrix} = \begin{Bmatrix} \{F_1\} \\ \{F_2\} \end{Bmatrix} \tag{2.9}$$

Listing 2.4: Section 4

```
69  %% SECTION 4
70  % Determining U and F_ext
71  % [k_11]*{U_1}+[k_12]*{U_2} = {F_1}
72  % [k_21]*{U_1}+[k_22]*{U_2} = {F_2}
73
74  U_2 = zeros(2*numnode-counter,1);
75  F_1 = boundaries(1:end,2); F_1 = F_1(~isnan(F_1));
76  U_1 = k_11\(F_1-k_12*U_2); %[k]*{U} = {F} -> {U} = [k]\{F}
77  F_2 = k_21*U_1+k_22*U_2;
78
79  U(vector_1(1:counter)) = U_1;
80  F_ext = boundaries(1:end,2); F_ext(isnan(F_ext)) = F_2;
```

It is known from applied mechanics that the internal force within a prismatic bar that is axially loaded, is $N = A\sigma = AE\varepsilon$. This relation is used in section 5 of the script in order to find the normal force within each truss member. Starting with the axial strains, the deformations related to the local coordinate system need to be determined. To do this, firstly, the element length L and the orientation parameters l and m need to be recalculated. Secondly, the relevant global node displacements need to be retrieved from the vector U. The local displacements are then found by multiplying these displacements with the

transformation matrix as defined by equation (2.2). To determine the strains, the definition needs to be revisit: $\varepsilon = du/dx$. This can be described as $\varepsilon = \Delta u/L$. The relevant elastic modulus and cross-sectional area are retrieved from the descriptive file.

Listing 2.5: Section 5

```
%% SECTION 5
% Determining the strains, stresses and internal forces

for n = 1:numelem
    x_i = nodes(elements(n,1),1);
    y_i = nodes(elements(n,1),2);
    x_j = nodes(elements(n,2),1);
    y_j = nodes(elements(n,2),2);

    L = sqrt((x_j-x_i)^2+(y_j-y_i)^2);
    l = (x_j-x_i)/L;
    m = (y_j-y_i)/L;

    entries = [2*elements(n,1)-1; 2*elements(n,1);...
               2*elements(n,2)-1; 2*elements(n,2)];

    U_local = [l m 0 0; 0 0 l m]*U(entries);
    epsilon(n) = (U_local(2)-U_local(1))/L;
    sigma(n) = elements(n,3)*epsilon(n);
    F_int(n) = elements(n,4)*sigma(n);
end
```

In the three sections that follow two matrices are constructed; one with the coordinates of the non-displaced nodes and one with the coordinates of the displaced nodes. The former is a reassembly of the coordinates as described in the worksheet 'nodes' of the descriptive file. As for the displaced nodes, the coordinates are found by adding the global displacement components to the x- and y-coordinate of the non-displaced nodes accordingly. The matrix that follows is constructed in the same way.

Listing 2.6: Section 6, 7 and 8

```
%% SECTION 6
% Merger of 'nodes' and 'elements'
% Row-format: [x_i,y_i,x_j,y_j]

matrix_1 = zeros(numelem,4);
for p = 1:numelem
    matrix_1(p,[1 2]) = nodes(elements(p,1),:);
    matrix_1(p,[3 4]) = nodes(elements(p,2),:);
end

%% SECTION 7
% Coordinates of displaced nodes
% In format of 'nodes'

vector_dx = U(1:2:end)*10^3;
vector_dy = U(2:2:end)*10^3;

disnodes(1:numnode,1) = nodes(1:numnode,1) + vector_dx;
```

```
disnodes(1:numnode,2) = nodes(1:numnode,2) + vector_dy;

%% SECTION 8
% Merger of 'disnodes' and 'elements'
% Row-format: [x_i,y_i,x_j,y_j]

matrix_2 = zeros(numelem,4);
for p = 1:numelem
    matrix_2(p,[1 2]) = disnodes(elements(p,1),:);
    matrix_2(p,[3 4]) = disnodes(elements(p,2),:);
end
```

These matrices are put to use by plotting them and generating a graphical output as done in the ninth section. Each element and node is efficiently plotted in their position of the undistorted and distorted configuration of the structure by using loops. Adding labels and assigning helpful properties gives a graphic that is clear and easy to comprehend.

It needs to be noted that in section 7 the global displacements are multiplied by an amplification factor of 10^3 in order to observe the deformed structure easily in the graph.

Listing 2.7: Section 9

```
%% SECTION 9
% Graphical output

figure
hold on

for n = 1:numelem
    line_undef = line([matrix_1(n,1),matrix_1(n,3)],...
                      [matrix_1(n,2),matrix_1(n,4)]);
    line_undef.LineStyle = '--';
    line_undef.LineWidth = 0.75;
    line_undef.Color = 'k';
    text((matrix_1(n,1)+matrix_1(n,3))/2,...
         (matrix_1(n,2)+matrix_1(n,4))/2,...
         ['E #' num2str(n)],'Color','r');
end

for n = 1:numelem
    line_def = line([matrix_2(n,1),matrix_2(n,3)],...
                    [matrix_2(n,2),matrix_2(n,4)]);
    line_undef.LineStyle = '-';
    line_def.LineWidth = 0.75;
    line_def.Color = 'k';
end

for n = 1:numnode
    scatter(nodes(n,1),nodes(n,2),'k');
    scatter(disnodes(n,1),disnodes(n,2),'k');
    text(nodes(n,1),nodes(n,2),['N #' num2str(n)],'Color','r');
end

xlabel('x-coordinate [mm]');
ylabel('y-coordinate [mm]');
title('Deformed structure, amplification = 10^3');
grid on
```

```
169
170 hold off
```

What remains is generating a non-graphical output in which the different quantities are clearly displayed. This is done in the tenth section by tabulating the external forces and global displacements related to the nodes, as well as the strains, stresses and internal forces related to the elements. The units are expressed in N, mm and N/mm^2.

Listing 2.8: Section 10

```
172 %% SECTION 10
173 % Non-graphical output
174 % Nodal (external) forces, nodal displacements and internal forces
175
176 node = transpose(1:numnode);
177 F_x_ext = F_ext(1:2:end)*10^0;
178 F_y_ext = F_ext(2:2:end)*10^0;
179 dx = U(1:2:end);
180 dy = U(2:2:end);
181 table(node,F_x_ext,F_y_ext,dx,dy)
182
183 element = transpose(1:numelem);
184 sigma = sigma*10^0;
185 N = F_int*10^0;
186 table(element,epsilon,sigma,N)
```

2.1.3 Output

During the run of the program Matlab loads and stores the input values, computes the different vectors and matrices and generates the output. The vectors and matrices that are called and computed in the program are all stored and can be found in the Workspace window. It is also possible to find these quantities in the Command Window by calling them. Fortunately, the program presents the relevant non-graphical output automatically, as shown in figure 2.4. The graphical output is shown in figure 2.5.

As can be observed from figure 2.4, the program generates two answers; firstly, a table with the quantities that are related to the nodes and secondly, a table with the quantities that are related to the elements.

The second and third column of the first table presents the external forces per node in the x- and y-direction respectively in units of newton. As described in the descriptive file, the point loads exerted upon the structure in node 2, 5 and 8 are 10 kN, 20 kN and 10 kN in the negative y-direction respectively. From symmetry, one can easily conclude that the reaction force at node 3 and 10 are both 20 kN in the positive y-direction. This can be checked by looking at the according rows of the third column. Because no loads are working in the x-direction, the values in the second column are all zero. Why node 3 shows a negligible amount of reaction force leftwards is because the truss deforms a negligible amount rightwards at the lower right corner which is due to compatibility conditions.

The nodal displacement coordinates relative to the location of the non-displaced nodes are presented in the fourth and fifth column of the first table

in mm. As for the sign convention, this relates to the global coordinate system. For instance, the upper left and right node (N1 and N9) both displace inwards and down. This means that dx for N1 is positive and for N9 is negative and that dy for both is negative. Graphically, this can easily be checked by looking at figure 2.5.

Which elements elongate and which elements compress can be found in the second column of the second table. An elongation is indicated by a positive value and a compression is indicated by a negative value. This sign convention also applies to the third column where the stresses are presented in $\mathrm{N/mm^2}$ and the fourth column where the normal forces are presented in units of newton; tension is positive and pressure is negative. In this simple truss example, physical intuition tells that the upper horizontal elements compress and the lower horizontal elements elongate as well as the diagonals. This can be checked by looking at the signs in figure 2.4 or by looking at the deformed structure in figure 2.5. Apparently, element 16 compresses a negligible amount.

A comparison between the normal forces determined with Trusses2d.m and a commercial finite element program will be provided in paragraph 4.2.

```
>> Trusses2d

ans =
```

node	F_x_ext	F_y_ext	dx	dy
1	0	0	0.33333	-0.095238
2	0	-10000	4.6966e-17	-0.69795
3	-9.8628e-12	20000	0	0
4	0	0	0.2381	-0.74556
5	0	-20000	0.095238	-1.0231
6	0	0	0.095238	-1.0231
7	0	0	-0.047619	-0.74556
8	0	-10000	0.19048	-0.69795
9	0	0	-0.14286	-0.095238
10	0	20000	0.19048	0

```
ans =
```

element	epsilon	sigma	N
1	6.7344e-05	14.142	28284
2	2.3483e-20	4.9314e-15	9.8628e-12
3	-4.7619e-05	-10	-20000
4	-4.7619e-05	-10	-20000
5	-2.381e-05	-5	-10000
6	4.7619e-05	10	20000
7	3.3672e-05	7.0711	14142
8	-7.1429e-05	-15	-30000
9	0	0	0
10	3.3672e-05	7.0711	14142
11	-2.381e-05	-5	-10000
12	4.7619e-05	10	20000
13	-7.1429e-05	-15	-30000
14	6.7344e-05	14.142	28284
15	-4.7619e-05	-10	-20000
16	-1.3878e-20	-2.9143e-15	-5.8287e-12
17	-4.7619e-05	-10	-20000

Figure 2.4: Non-graphical output

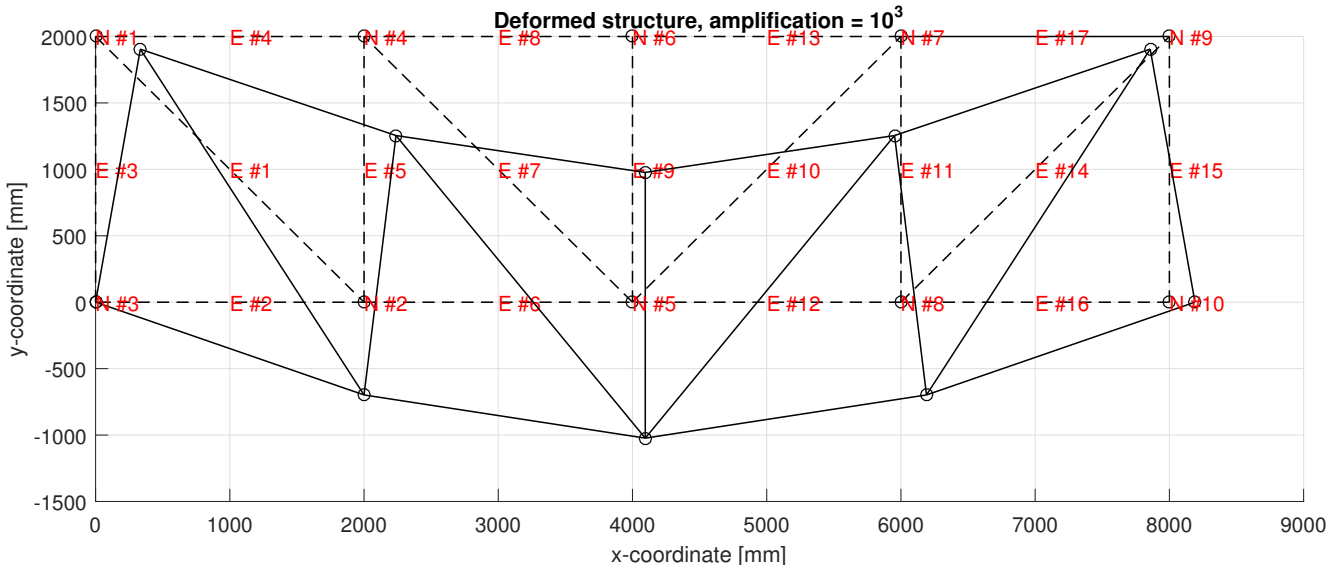

Figure 2.5: Graphical output

2.1.4 3D trusses

In the real world structures are three dimensional and can be modelled and analysed as such with the help of a finite element program. To demonstrate this a program for three dimensional trusses is written. Like in the previous program bar elements are used. Keep in mind that a third dimension can also be implemented in FE protocols that concern elements which are able to bend. Because both programs are very similar, only the input and output are presented in this paragraph. The full script 'Trusses3d.m' is presented in appendix A. One point that needs to be mentioned is that the added z-axis to the global coordinate system comes with a different transformation matrix $[\lambda]$ as defined by equation (2.10).

$$[\lambda] = \begin{bmatrix} l & m & n & 0 & 0 & 0 \\ 0 & 0 & 0 & l & m & n \end{bmatrix} \tag{2.10}$$

Just like in the program for two dimensional trusses, transformation parameter l and m can be implemented as $(x_j - x_i)/L$ and $(y_j - y_i)/L$ respectively. Transformation parameter n is implemented as $(z_j - z_i)/L$ and the global element stiffness matrices $[k^{\circ(e)}]$ are still computed by pre multiplication with the transpose of the lambda matrix and post multiplication with the lambda matrix. Equation (2.8) applies for bar elements in 3D space as well. After all, the bar element still deforms only along the local x-axis, i.e. the s-axis.

Input and output

As an example a cubicle truss of $3\times3\times3$ m^3, supported by four three dimensional hinges is described by a spreadsheet (figure 2.6). The structure derives its stability from the four vertical sides of which the corners are connected by diagonal bar elements. Point loads of 200 kN are exerted upon the structure in

the positive y-direction at node 7 and 8. The resultant deformation is shown in figure 2.8.

	A	B	C
1	0	0	0
2	3000	0	0
3	3000	3000	0
4	0	3000	0
5	0	0	3000
6	3000	0	3000
7	3000	3000	3000
8	0	3000	3000
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			

	A	B	C	D
1	1	5	210000	6000
2	5	6	210000	6000
3	6	7	210000	6000
4	7	8	210000	6000
5	8	5	210000	6000
6	2	6	210000	6000
7	3	7	210000	6000
8	4	8	210000	6000
9	1	6	210000	6000
10	2	5	210000	6000
11	2	7	210000	6000
12	3	6	210000	6000
13	3	8	210000	6000
14	4	7	210000	6000
15	1	8	210000	6000
16	4	5	210000	6000
17				
18				
19				
20				
21				
22				
23				
24				

	A	B
1	0	nan
2	0	nan
3	0	nan
4	0	nan
5	0	nan
6	0	nan
7	0	nan
8	0	nan
9	0	nan
10	0	nan
11	0	nan
12	0	nan
13	nan	0
14	nan	0
15	nan	0
16	nan	0
17	nan	0
18	nan	0
19	nan	0
20	nan	200000
21	nan	0
22	nan	0
23	nan	200000
24	nan	0

nodes elements boundary_conditions

Figure 2.6: Descriptive file

```
>> Trusses3d

ans =
```

node	F_x_ext	F_y_ext	F_z_ext	dx	dy	dz
1	-20284	-1.122e+05	-2e+05	0	0	0
2	20284	-1.122e+05	-2e+05	0	0	0
3	-25922	-87797	2e+05	0	0	0
4	25922	-87797	2e+05	0	0	0
5	0	0	0	0.024147	0.752	0.16075
6	0	0	0	-0.024147	0.752	0.16075
7	0	2e+05	0	0.03086	0.96104	-0.20543
8	0	2e+05	0	-0.03086	0.96104	-0.20543

```
ans =
```

element	epsilon	sigma	N
1	5.3582e-05	11.252	67513
2	-1.6098e-05	-3.3806	-20284
3	6.968e-05	14.633	87797
4	2.0573e-05	4.3204	25922
5	6.968e-05	14.633	87797
6	5.3582e-05	11.252	67513
7	-6.8477e-05	-14.38	-86281
8	-6.8477e-05	-14.38	-86281
9	2.2766e-05	4.7809	28686
10	2.2766e-05	4.7809	28686
11	0.00012594	26.446	1.5868e+05
12	-9.8543e-05	-20.694	-1.2416e+05
13	-2.9095e-05	-6.1099	-36660
14	-2.9095e-05	-6.1099	-36660
15	0.00012594	26.446	1.5868e+05
16	-9.8543e-05	-20.694	-1.2416e+05

Figure 2.7: Non-graphical output

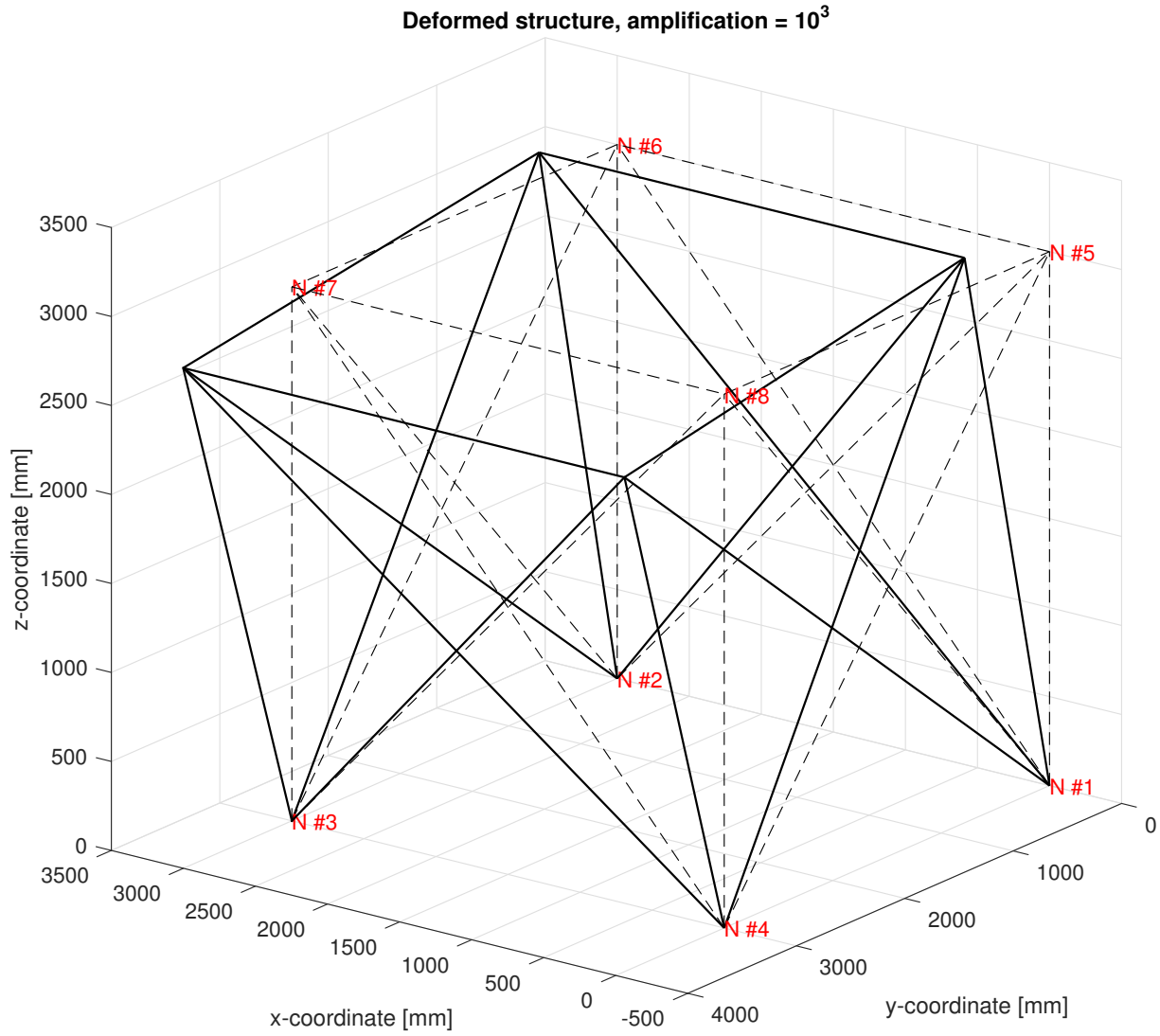

Figure 2.8: Graphical output

Signal tower

To make telecommunication possible, signal towers are a necessity. These structures are geometrically interesting and are perfect to analyse with a finite element program for 3D trusses. Imagine a conical truss tower with a triangular footprint. At the top, a dish is attached to one of the nodes in a vertical position. If a gust of wind catches the dish, the signal tower will deform. How this deformation looks, is presented in figure 2.9.

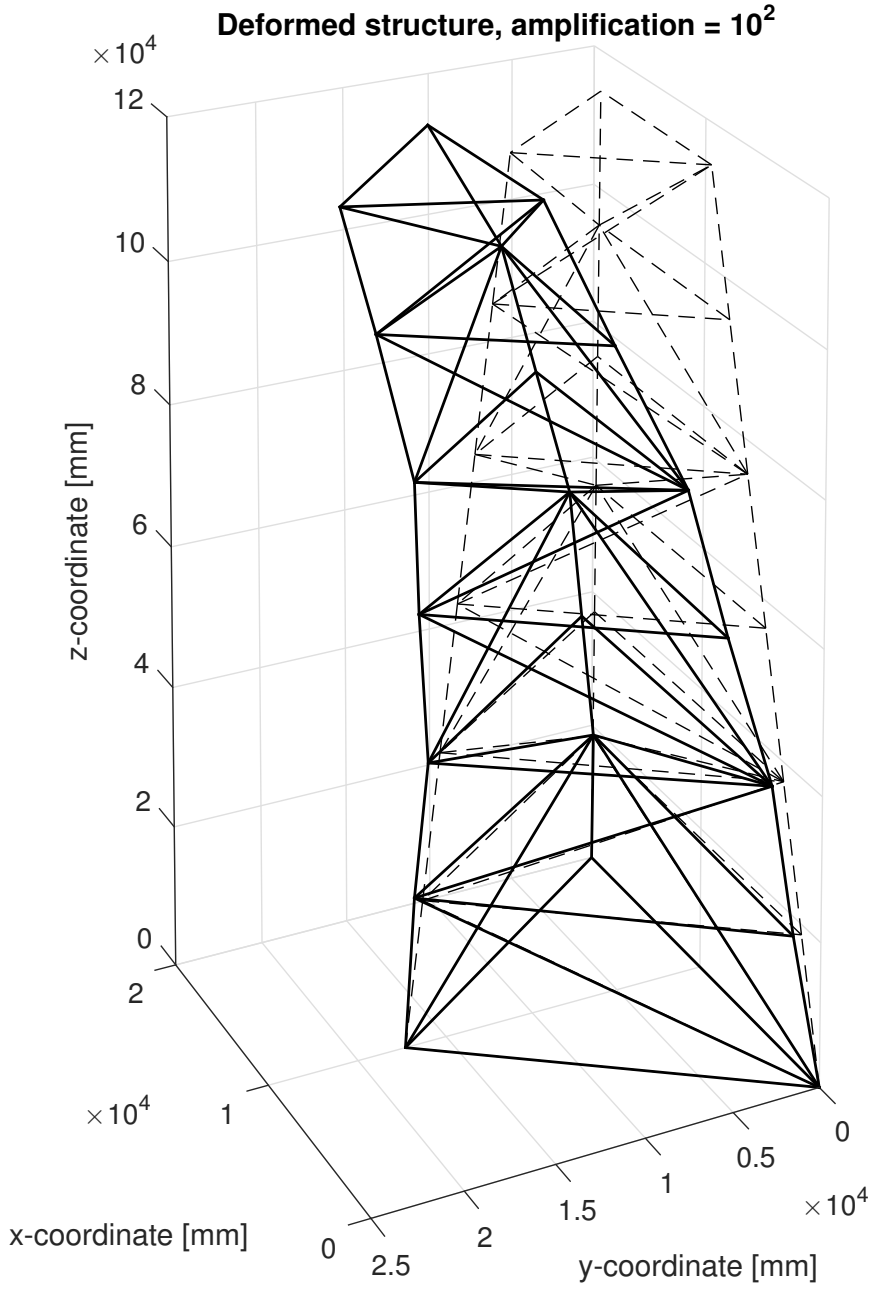

Figure 2.9: Signal tower

2.2 Frames

Where trusses which are an assembly of bar members, can only transfer loads to the suspension points by way of internal normal forces, frames also convey loads via internal shear forces and bending moments. The components of which these kind of structures consist are called beams.

In this paragraph two types of beam elements are discussed: linear and quadratic. These terms relate to the order of the polynomial which describes the displacement fields of the elements. Starting with the linear type, only the input and the script are presented and clarified; the output is analysed in depth in the chapters that follow.

2.2.1 Linear elements

Input

As an example, a simply supported beam of 10 meters is used with the cross-sectional profile HEA500. A uniformly distributed load of 15 kN/m is applied over the entire span in the negative y-direction. The corresponding descriptive file describes this structure by four linear beam elements. See figure 2.10. Among others, this structure will be analysed in the third chapter.

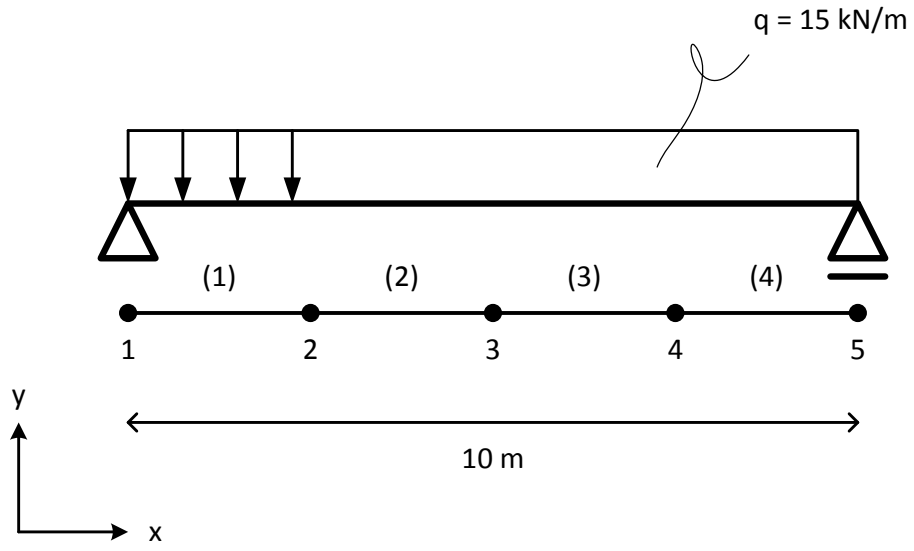

Figure 2.10: Simply supported beam described by linear elements

The set-up of the descriptive file is similar for frame structures. However, there are some additions. Firstly, a column E is added to the elements worksheet, containing the second moment of area. Secondly, regarding the same worksheet, column F and G are added, containing the uniformly distributed load in the x- and y-direction respectively, expressed in N/mm. Finally, a third degree of freedom per node is added to column A of the worksheet 'boundary conditions' which is the (un)known rotation around the axis parallel to the z-axis. In conjuction with this, a third force related quantity is added per node to column

B, i.e. the reaction moment or (un)known torque around the axis parallel to the z-axis. All this can be observed in figure 2.11. For example, row 3 and 15 show the new boundary conditions for the suspension points at both ends. Because these nodes cannot resist a bending moment, the rotational degree of freedom is unknown, as indicated by the nan value, and the corresponding force quantity will be either null or a non-null value; in this case null, because torque is not present.

	A	B
1	0	0
2	2500	0
3	5000	0
4	7500	0
5	10000	0
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		

	A	B	C	D	E	F	G
1	1	2	2E+05	19754	9E+08	0	-15
2	2	3	2E+05	19754	9E+08	0	-15
3	3	4	2E+05	19754	9E+08	0	-15
4	4	5	2E+05	19754	9E+08	0	-15
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							

	A	B
1	0	nan
2	0	nan
3	nan	0
4	nan	0
5	nan	0
6	nan	0
7	nan	0
8	nan	0
9	nan	0
10	nan	0
11	nan	0
12	nan	0
13	nan	0
14	0	nan
15	nan	0

nodes | elements | boundary_conditions

Figure 2.11: Descriptive file

FramesLinear2d.m

As remarked at the start of this chapter, each program can be considered as a steppingstone for the successive program; the complexity increases but the fundamental concepts remain the same. For that reason the scripts are similar and only those sections that need further explanation will be presented and discussed. This is done for frame structures described by linear elements in this part of the paragraph. The entire script can found in appendix B. Just like for trusses, the script for frames is divided into ten sections. Starting with the second section 'Constructing $\left[k^{\circ}\right]$ and $\left\{q^{\circ}\right\}$', the input is already loaded and the empty vectors and matrices are already set-up.

The global stiffness matrix is constructed by adding the global stiffness matrix of each element to the assigned empty matrix as defined in the first section. This will be done with the help of a loop which will be iterated numelem times. In the loop, the first thing that will be done is finding the x- and y-coordinate of both node i and node j. These values will be used as an input for the computation of the element length L and the orientation parameters l and m. The material and cross-sectional properties are defined next: E, A and I_{yy} are extracted from the descriptive file and the shear modulus G is related to E by $G = E/(2(1+\nu))$ in which Poisson's ration ν is set to 0.3.

What follows is the definition of the global element stiffness matrix $\left[k^{\circ(e)}\right]$ and the correct positioning of this matrix into the global stiffness matrix $\left[k^{\circ}\right]$. The matrix $\left[k\right]$ is defined by equation (2.4). Where a bar element has one displacement field, a beam element has three displacement fields: longitudinal

u, transverse v and rotational θ. In the case of a linear beam element, these displacement fields are described by a first order polynomial and can be defined, after solving for the constants a and b, by the following equations.

$$u = N_i u_i + N_j u_j \tag{2.11}$$
$$v = N_i v_i + N_j v_j \tag{2.12}$$
$$\theta = N_i \theta_i + N_j \theta_j \tag{2.13}$$

With $N_i = (x_j - x)/L$ and $N_j = (x - x_i)/L$. The element shape function matrix $\left[N^{(e)}\right]$ can be found by putting the equations into vector-matrix form.

$$\begin{Bmatrix} u \\ v \\ \theta \end{Bmatrix} = \begin{bmatrix} N_i & 0 & 0 & N_j & 0 & 0 \\ 0 & N_i & 0 & 0 & N_j & 0 \\ 0 & 0 & N_i & 0 & 0 & N_j \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ \theta_i \\ u_j \\ v_j \\ \theta_j \end{Bmatrix} \tag{2.14}$$

Where the product of the shape function matrix and the nodal displacement vector $\left\{U^{(e)}\right\}$ defines the displacements throughout the element, the product of the matrix $\left[B^{(e)}\right]$ and the nodal displacement vector defines the strains throughout the element. Equation (2.15) down to (2.17) presents the definition of the axial strain ε, the shear strain γ and the curvature κ.

$$\varepsilon = \frac{du}{dx} = \frac{dN_i}{dx} u_i + \frac{dN_j}{dx} u_j \tag{2.15}$$
$$\gamma = \frac{dv}{dx} - \theta = \frac{dN_i}{dx} v_i + \frac{dN_j}{dx} v_j - (N_i \theta_i + N_j \theta_j) \tag{2.16}$$
$$\kappa = \frac{d\theta}{dx} = \frac{dN_i}{dx} \theta_i + \frac{dN_j}{dx} \theta_j \tag{2.17}$$

After determining the derivative of the shape functions for linear beam elements ($dN_i/dx = -1/L$ and $dN_j/dx = 1/L$), the matrix $\left[B^{(e)}\right]$ can be found by putting the equations into vector-matrix form.

$$\begin{Bmatrix} \varepsilon \\ \gamma \\ \kappa \end{Bmatrix} = \frac{1}{L} \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & -N_i L & 0 & 1 & -N_j L \\ 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ \theta_i \\ u_j \\ v_j \\ \theta_j \end{Bmatrix} \tag{2.18}$$

The final step before equation (2.4) can be evaluated, is formulating the constitutive matrix $\left[D^{(e)}\right]$, i.e. the matrix which holds the relation between the stresses and strains. From applied mechanics, this relation is known for the normal force N, the shear force S and the bending moment M and can be presented in vector-matrix form as shown by equation (2.19).

$$\begin{Bmatrix} N \\ S \\ M \end{Bmatrix} = \begin{bmatrix} EA & 0 & 0 \\ 0 & GA & 0 \\ 0 & 0 & EI \end{bmatrix} \begin{Bmatrix} \varepsilon \\ \gamma \\ \kappa \end{Bmatrix} \tag{2.19}$$

Numerical integration is applied for the evaluation of $\left[k^{(e)}\right]$, because $\left[B^{(e)}\right]$ is a function of x. The numerical evaluation will be exact if the correct integration point is chosen. For a first order interpolation polynomial this is the mid-point of the element. A graphical clarification is presented in figure 2.12.

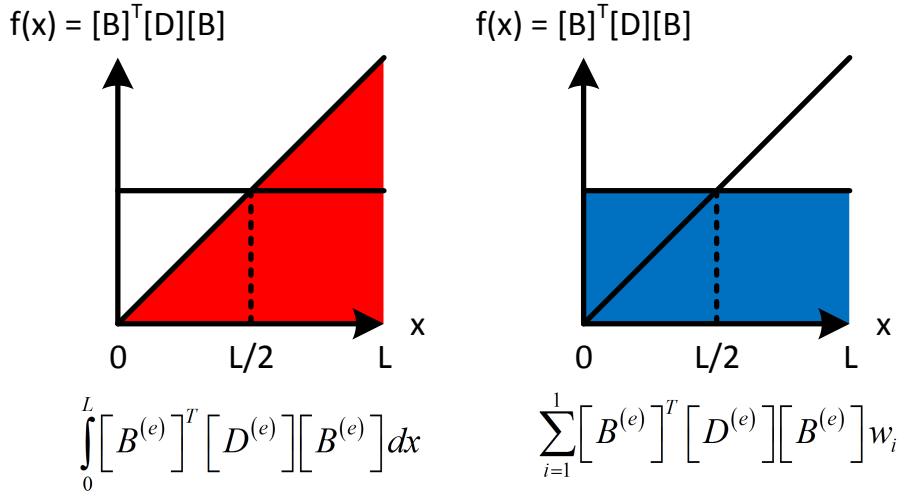

Figure 2.12: Numerical integration for linear beam elements

The left diagram represents the analytical solution and the right diagram represents the numerical solution. As a weight for the latter, $w_{i=1}$ is set to L and for the variable x in the matrix $\left[B^{(e)}\right]$ $L/2$ is substituted.

Up to this point, the element stiffness matrix only applies to beam elements which are orientated parallel to the global x-y coordinate system. In order to let $\left[k^{(e)}\right]$ apply to elements with an arbitrary orientation, a transformation matrix needs to be used in same way as is done for bar elements, i.e. the element stiffness matrix needs to be pre multiplied by the transpose of the lambda matrix and post multiplied by the lambda matrix.

$$\left[k^{\circ(e)}\right] = \left[\lambda\right]^T \left[k^{(e)}\right] \left[\lambda\right] \tag{2.20}$$

But how is this lambda matrix formulated? To answer this question, use is made, again, of figure 2.3 and equation (2.1). Unlike a bar element, a beam element can undergo shear deformation and rotational deformation as well. However the latter is not dependent on the orientation of the element, so for that reason an identity entry is used. Dealing with two nodes per element, the following transformation matrix applies.

$$\left[\lambda\right] = \begin{bmatrix} l & m & 0 & 0 & 0 & 0 \\ -m & l & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & l & m & 0 \\ 0 & 0 & 0 & -m & l & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.21}$$

In the program, this modification is implemented by post multiplying the matrix $\left[B^{(e)}\right]$ with the lambda matrix. See line 55 and 56 of the script. Because now

the local s-t coordinate system applies, the variable x is replaced by the variable s as shown in line 38 down to 43 which hold the relevant s-coordinates of the element and the shape functions. The element is shown in figure 2.13.

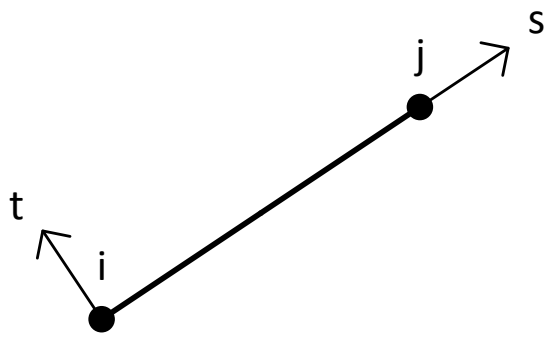

Figure 2.13: Linear beam element with arbitrary orientation

Now that the element stiffness matrix is defined, the positioning of each of these matrices into the global stiffness matrix needs to be performed in a systematic fashion. This is done, like in the program for trusses, by constructing the entries vector. See line 48 down to 50. The entries are identical to the renumbered indices of the elements in the global displacement vector. Equation (2.22) presents the general case for the global element displacement vector. Note the indices.

$$\left\{U^{\circ}\right\} = \begin{Bmatrix} u_i^{\circ} \\ v_i^{\circ} \\ \theta_i^{\circ} \\ u_j^{\circ} \\ v_j^{\circ} \\ \theta_j^{\circ} \end{Bmatrix} = \begin{Bmatrix} u_{3i-2}^{\circ} \\ u_{3i-1}^{\circ} \\ u_{3i}^{\circ} \\ u_{3j-2}^{\circ} \\ u_{3j-1}^{\circ} \\ u_{3j}^{\circ} \end{Bmatrix} \tag{2.22}$$

What remains is the construction of the global distributed force vector $\left\{q^{\circ}\right\}$ which is defined by equation (2.23). For the derivation, reference is made to paragraph 5.2 (p.61-66) of [1].

$$\left\{q^{\circ}\right\} = \sum_{e=1}^{E} \left[\int_L \left[N^{(e)}\right]^T \begin{Bmatrix} q_x \\ q_y \\ 0 \end{Bmatrix} ds \right] = \sum_{e=1}^{E} \left\{q^{\circ(e)}\right\} \tag{2.23}$$

The entry values for the component vector are retrieved from the descriptive file, the elements worksheet, column F and G. Why the last entry contains the null value is because there is no distributed equivalence for the bending moment. These components are parallel to the global coordinate system and can only be applied as such. In the fourth chapter it will be investigated how these kinds of loads can also be modelled parallel to the local coordinate system. After the component vector is pre multiplied by the transpose of the element shape function matrix, an integration along the local x-axis is performed. Numerically this is done by multiplying the product with a weight L and substituting for the variable s in the matrix $\left[N^{(e)}\right]^T$ the value $L/2$. This needs to be a halve times the element length in order to discretize the q-load equally at the outer nodes.

In other words, the distributed load is modelled as a pair of point loads $qL/2$ exerted upon both nodes of the considered element. The same entries vector is used to construct the global distributed force vector $\{q^{\circ}\}$ by substitution of the global element distributed force vectors $\{q^{\circ(e)}\}$ accordingly. See line 60. Continuing with the example structure of the input, figure 2.14 presents this graphically.

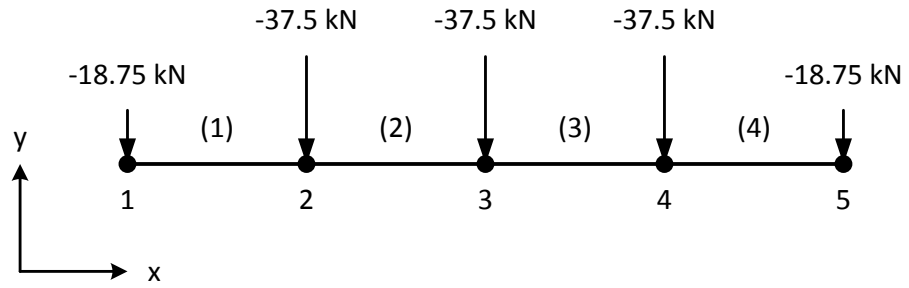

Figure 2.14: Model of the distributed load for linear elements

Listing 2.9: Section 2

```
20  %% SECTION 2
21  % Constructing k and q_glob
22
23  for n = 1:numelem
24      x_i = nodes(elements(n,1),1);
25      y_i = nodes(elements(n,1),2);
26      x_j = nodes(elements(n,2),1);
27      y_j = nodes(elements(n,2),2);
28
29      L = sqrt((x_j-x_i)^2+(y_j-y_i)^2);
30      l = (x_j-x_i)/L;
31      m = (y_j-y_i)/L;
32
33      E = elements(n,3);
34      G = E/(2*(1+0.3));
35      A = elements(n,4);
36      I = elements(n,5);
37
38      s_i = 0;
39      s_j = L;
40      s = L/2;
41
42      N_i = (s_j-s)/L;
43      N_j = (s-s_i)/L;
44
45      lambda = [l m 0 0 0 0; -m l 0 0 0 0; 0 0 1 0 0 0;...
46                0 0 0 l m 0; 0 0 0 -m l 0; 0 0 0 0 0 1];
47
48      entries = [3*elements(n,1)-2; 3*elements(n,1)-1;...
49                 3*elements(n,1); 3*elements(n,2)-2;...
50                 3*elements(n,2)-1; 3*elements(n,2)];
51
52      q = [elements(n,6); elements(n,7); 0];
53
54      N = [N_i 0 0 N_j 0 0; 0 N_i 0 0 N_j 0; 0 0 N_i 0 0 N_j];
```

```
55      B = 1/L*[-1 0 0 1 0 0; 0 -1 -L*N_i 0 1 -L*N_j;...
56               0 0 -1 0 0 1]*lambda;
57      D = [E*A 0 0; 0 G*A 0; 0 0 E*I];
58
59      k(entries,entries) = k(entries,entries)+transpose(B)*D*B*L;
60      q_glob(entries) = q_glob(entries)+transpose(N)*q*L;
61  end
```

After the global stiffness matrix is rearranged and partitioned, the global nodal displacements and the external forces can be determined. This is done in the fourth section of the script in the same way as for trusses. However, in order to deal with the q-load, some additions are implemented.

The first part of the index vector 'vector_a' is used to retrieve the known external forces from the worksheet that contains the boundary conditions. See line 101 of the script. Vector F_1 is completed after the values of the global distributed force vector q_glob are added. This can be done quite easily, because column B of the worksheet 'boundary_conditions' has the same ordering as q_glob. Special attention goes to the nodes where the structure is supported. Here the discretizised distributed load is transferred to the support directly. With respect to the example structure of the input, this concerns node 1 and 5. See figure 2.14. Because these entries of q_glob do not influence the behaviour of the structure, they simply need to be added to the reaction forces with an opposite sign. This is done in line 108.

Listing 2.10: Section 4

```
95  %% SECTION 4
96  % Determining U and F_ext
97  % [k_11]*{U_1}+[k_12]*{U_2} = {F_1}
98  % [k_21]*{U_1}+[k_22]*{U_2} = {F_2}
99
100 U_2 = zeros(3*numnode-counter,1);
101 F_1 = boundaries(vector_a,2)+q_glob(vector_a);
102 U_1 = k_11\(F_1-k_12*U_2); %[k]*{U} = {F} -> {U} = [k]\{F}
103 F_2 = k_21*U_1+k_22*U_2;
104
105 U(vector_a) = U_1;
106 boundaries(vector_a,2) = F_1;
107 F_ext = boundaries(1:end,2);
108 F_ext(isnan(F_ext)) = F_2-q_glob(vector_b);
```

Constructing the matrices which hold the coordinates of the non-displaced and the displaced nodes, enables generating a graphical output of the undistorted and distorted structure. This is done in section 5 down to 8. What follows is the section in which the internal forces are determined: the normal force N, the shear force S and the bending moment M_z.

The ninth section contains three loops of which the first computes the internal forces and the strains and also creates the graphical presentation of these quantities. Because this is done per element, the number of iterations is equal to numelem. As can be observed from listing 2.11, per loop iteration the global element stiffness matrix and the global element distributed force vector are computed, just like in the second section. But instead of substituting this matrix

and vector in their global equivalence, the former is multiplied with the global displacements of the relevant nodes in order to find the internal forces 'F_int'. See line 225.

$$F\_int = \begin{Bmatrix} F_{x,i} \\ F_{y,i} \\ M_{z,i} \\ F_{x,j} \\ F_{y,j} \\ M_{z,j} \end{Bmatrix} \tag{2.24}$$

In the lines that follow, down to line 233, the entries of the global element internal force vector with the same direction index, see (2.24), are isolated as three 2 by 1 matrices. To determine a normal force and a shear force per node, the matrices that hold the internal forces in the x- and y-direction are modified twice.

First, the uniformly distributed load per element is equally concentrated in the nodes as point loads and added to the aforementioned matrices accordingly. See line 227 down to 230. This is done because the discretizised distributed load at the support nodes are of no influence on the nodal displacements. So when the global element displacement vector ($\left\{U^{\circ(e)}\right\} = U\_elem$) is used to compute the internal quantities, these particular external forces will not 'show'. Obviously, this effect will become less when the mesh size is increased, but for linear elements the solution will become more accurate with this modification, regardless of the amount of elements. A closer look will be taken at this in the third chapter.

The second modification concerns the transformation of the internal forces related to the global x-y coordinate system, i.e. F_x and F_y, to the normal force N and the shear force S. This is done by pre multiplying the internal forces with the transformation matrix as defined by equation (2.1). See line 231 and 232.

Next, the strains are computed by multiplying $\left[B^{(e)}\right]$ with $\left\{U^{\circ(e)}\right\}$, after which the result is assigned to the empty vectors that are set-up in the first section. Note that the strains are computed at the location of the integration point. Generating the graphical and non-graphical output are the remaining acts within this loop.

In the loops that follow, the undistorted structure is generated as a graphical output along which the internal force related quantities are graphically presented.

Listing 2.11: Section 9

```
178 %% SECTION 9
179 % Graphical and non-graphical output: N, S and M_z_int
180 % [k_elem]*{U_elem} = {F_int} with {U_elem} = {U(entries)}
181
182 figure
183 hold on
184
185 for n = 1:numelem
186     x_i = nodes(elements(n,1),1);
187     y_i = nodes(elements(n,1),2);
188     x_j = nodes(elements(n,2),1);
189     y_j = nodes(elements(n,2),2);
```

```
190
191       L = sqrt((x_j-x_i)^2+(y_j-y_i)^2);
192       l = (x_j-x_i)/L;
193       m = (y_j-y_i)/L;
194
195       E = elements(n,3);
196       G = E/(2*(1+0.3));
197       A = elements(n,4);
198       I = elements(n,5);
199
200       s_i = 0;
201       s_j = L;
202       s = L/2;
203
204       N_i = (s_j-s)/L;
205       N_j = (s-s_i)/L;
206
207       lambda = [l m 0 0 0 0; -m l 0 0 0 0; 0 0 1 0 0 0;...
208                 0 0 0 l m 0; 0 0 0 -m l 0; 0 0 0 0 0 1];
209
210       entries = [3*elements(n,1)-2; 3*elements(n,1)-1;...
211                  3*elements(n,1); 3*elements(n,2)-2;...
212                  3*elements(n,2)-1; 3*elements(n,2)];
213
214       q = [elements(n,6); elements(n,7); 0];
215
216       N = [N_i 0 0 N_j 0 0; 0 N_i 0 0 N_j 0; 0 0 N_i 0 0 N_j];
217       B = 1/L*[-1 0 0 1 0 0; 0 -1 -L*N_i 0 1 -L*N_j;...
218                0 0 -1 0 0 1]*lambda;
219       D = [E*A 0 0; 0 G*A 0; 0 0 E*I];
220
221       k_elem = transpose(B)*D*B*L;
222       q_elem = transpose(N)*q*L;
223
224       U_elem = U(entries);
225       F_int = k_elem*U_elem;
226
227       F_x_int = (F_int(1:3:end)+diag(abs(q_elem(1:3:end)))*...
228                  sign(F_int(1:3:end)))*10^0;
229       F_y_int = (F_int(2:3:end)+diag(abs(q_elem(2:3:end)))*...
230                  sign(F_int(2:3:end)))*10^0;
231       N = [l*F_x_int(1)+m*F_y_int(1); l*F_x_int(2)+m*F_y_int(2)];
232       S = [-m*F_x_int(1)+l*F_y_int(1); -m*F_x_int(2)+l*F_y_int(2)];
233       M_z_int = F_int(3:3:end)*10^0;
234
235       strains = B*U_elem;
236       epsilon(n) = strains(1);
237       gamma(n) = strains(2);
238       kappa(n) = strains(3);
239
240       element = [n; n];
241       node = [elements(n,1); elements(n,2)];
242       table(element,node,N,S,M_z_int)
243
244       x_1 = x_i-m*S(1)*10^-2;
245       y_1 = y_i+l*S(1)*10^-2;
246       x_2 = x_1+(x_j-x_i);
247       y_2 = y_1+(y_j-y_i);
248
249       x_3 = x_i-m*M_z_int(1)*10^-5;
250       y_3 = y_i+l*M_z_int(1)*10^-5;
251       x_4 = x_j-m*abs(M_z_int(2))*sign(M_z_int(1))*10^-5;
```

```
252       y_4 = y_j+l*abs(M_z_int(2))*sign(M_z_int(1))*10^-5;
253
254       matrix_S = [x_i y_i x_1 y_1; x_1 y_1 x_2 y_2;
255                   x_2 y_2 x_j y_j]; %coordinates S-line
256       matrix_M = [x_i y_i x_3 y_3; x_3 y_3 x_4 y_4;
257                   x_4 y_4 x_j y_j]; %coordinates M-line
258
259       X_S = [matrix_S(:,1) matrix_S(:,3)];
260       Y_S = [matrix_S(:,2) matrix_S(:,4)];
261       plot(X_S,Y_S,'Color',[1 0.5 0],'LineWidth',2) %S-line
262       X_M = [matrix_M(:,1) matrix_M(:,3)];
263       Y_M = [matrix_M(:,2) matrix_M(:,4)];
264       plot(X_M,Y_M,'Color',[0.5 0 0.5],'LineWidth',2) %M-line
265   end
266
267   for n = 1:numelem
268       line_undef = line([matrix_1(n,1),matrix_1(n,3)],...
269                         [matrix_1(n,2),matrix_1(n,4)]);
270       line_undef.LineStyle = '--';
271       line_undef.LineWidth = 0.75;
272       line_undef.Color = 'k';
273       text((matrix_1(n,1)+matrix_1(n,3))/2,...
274            (matrix_1(n,2)+matrix_1(n,4))/2,...
275            ['E #' num2str(n)],'Color','r');
276   end
277
278   for n = 1:numnode
279       scatter(nodes(n,1),nodes(n,2),'k');
280       text(nodes(n,1),nodes(n,2),['N #' num2str(n)],'Color','r');
281   end
282
283   set(gca,'xtick',[])
284   set(gca,'ytick',[])
285   title('S-line and M-line')
286
287   hold off
```

In the last section, section 10, the non-graphical presentation of the nodal quantities are generated in the form of a table. The strains per element are tabulated here as well.

2.2.2 Quadratic elements

Input

For frame structures which are described by quadratic beam elements, the same example is used as for the description by linear beam elements to explain the input. As can be seen from observing figure 2.15, each element has one extra node in the middle. This is because the interpolation polynomial is of the second order which is accompanied by three unknown coefficients that need to be solved in order to determine the shape functions. The middle node is always located at the centre of the element, but does not have to be. It could have been placed at another point along the element, but that is not explored any further.

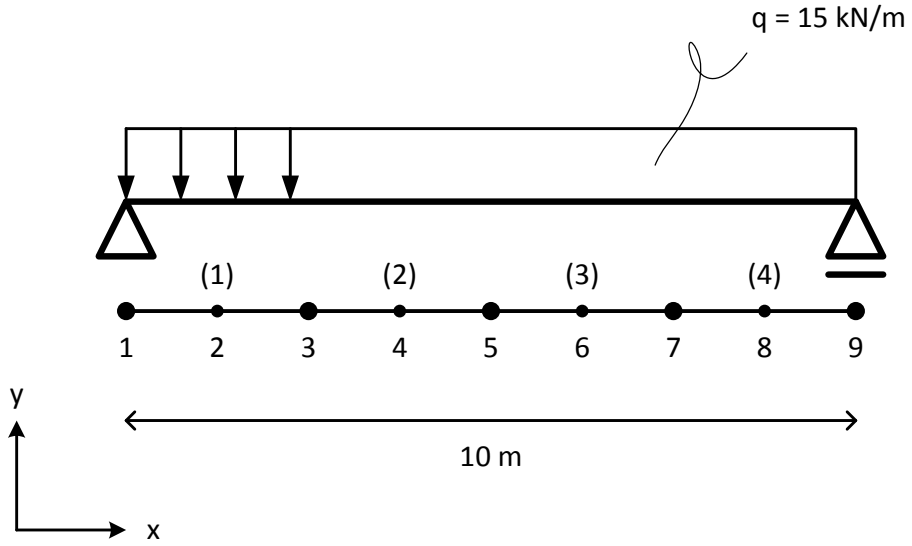

Figure 2.15: Simply supported beam described by quadratic elements

The spreadsheet that describes this structure is not any different from the descriptive file of its linear counterpart. However two remarks need to be made; each element takes two rows to be described in the corresponding worksheet. This is indicated in figure 2.16. Both sub-elements need to be grouped in this particular way, because the program reads the corresponding nodes from A_i to B_i to B_j. Also, the three degrees of freedom of the middle node are all unknown and the corresponding external nodal forces are known.

	A	B
1	0	0
2	1250	0
3	2500	0
4	3750	0
5	5000	0
6	6250	0
7	7500	0
8	8750	0
9	10000	0
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		

	A	B	C	D	E	F	G
1	1	2	210000	19754	8.7E+08	0	-15
2	2	3	210000	19754	8.7E+08	0	-15
3	3	4	210000	19754	8.7E+08	0	-15
4	4	5	210000	19754	8.7E+08	0	-15
5	5	6	210000	19754	8.7E+08	0	-15
6	6	7	210000	19754	8.7E+08	0	-15
7	7	8	210000	19754	8.7E+08	0	-15
8	8	9	210000	19754	8.7E+08	0	-15
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							

	A	B
1	0	nan
2	0	nan
3	nan	0
4	nan	0
5	nan	0
6	nan	0
7	nan	0
8	nan	0
9	nan	0
10	nan	0
11	nan	0
12	nan	0
13	nan	0
14	nan	0
15	nan	0
16	nan	0
17	nan	0
18	nan	0
19	nan	0
20	nan	0
21	nan	0
22	nan	0
23	nan	0
24	nan	0
25	nan	0
26	0	nan
27	nan	0

nodes elements boundary_conditions

Figure 2.16: Descriptive file

FramesQuadratic2d.m

Consisting of ten sections as well, only the second section of the script is discussed. The entire script can be found in appendix C.

As in the previous programs, the second section consists of a loop in which the global stiffness matrix $\left[k^{\circ}\right]$ is constructed. The loop starts with defining the x- and y-coordinate of the outer nodes, after which the element length L and the orientation parameters l and m can be computed. The material and cross-sectional properties are defined next.

To define the global element stiffness matrix $\left[k^{\circ(e)}\right]$ by equation (2.4), the three displacement fields u, v and θ need to be defined first. The script is kept comprehensible by introducing natural coordinates which are a function of the local coordinates: $L_1 = (x - x_i)/L$ and $L_2 = (x_j - x)/L$ (if the local and global coordinate system are positioned in the same direction). All three displacement fields are described by a second order polynomial $u = a + bL_1 + cL_1^2$ that is a function of the first natural coordinate L_1. Solving for a, b and c, the displacement fields can be described in terms of the shape functions.

$$u = N_i u_i + N_j u_j + N_k u_k \tag{2.25}$$
$$v = N_i v_i + N_j v_j + N_k v_k \tag{2.26}$$
$$\theta = N_i \theta_i + N_j \theta_j + N_k \theta_k \tag{2.27}$$

$$N_i = 1 - 3L_1 + 2L_1^2 \tag{2.28}$$
$$N_j = 4L_1 - 4L_1^2 \tag{2.29}$$
$$N_k = -L_1 + 2L_1^2 \tag{2.30}$$

The element shape function matrix $\left[N^{(e)}\right]$ can be found by putting the equations into vector-matrix form.

$$\begin{Bmatrix} u \\ v \\ \theta \end{Bmatrix} = \begin{bmatrix} N_i & 0 & 0 & N_j & 0 & 0 & N_k & 0 & 0 \\ 0 & N_i & 0 & 0 & N_j & 0 & 0 & N_k & 0 \\ 0 & 0 & N_i & 0 & 0 & N_j & 0 & 0 & N_k \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ \theta_i \\ u_j \\ v_j \\ \theta_j \\ u_k \\ v_k \\ \theta_k \end{Bmatrix} \tag{2.31}$$

In order to obtain matrix $\left[B^{(e)}\right]$, the definition of the strains ε, γ and κ needs to be known.

$$\varepsilon = \frac{du}{dx} = \frac{dN_i}{dx}u_i + \frac{dN_j}{dx}u_j + \frac{dN_k}{dx}u_k \tag{2.32}$$

$$\gamma = \frac{dv}{dx} - \theta = \frac{dN_i}{dx}v_i + \frac{dN_j}{dx}v_j + \frac{dN_k}{dx}v_k - (N_i\theta_i + N_j\theta_j + N_k\theta_k) \tag{2.33}$$

$$\kappa = \frac{d\theta}{dx} = \frac{dN_i}{dx}\theta_i + \frac{dN_j}{dx}\theta_j + \frac{dN_k}{dx}\theta_k \tag{2.34}$$

The derivatives of the shape functions to x can be determined with the chain rule $dN/dx = dN/dL_1 \cdot dL_1/dx$.

$$\frac{dN_i}{dx} = (-3 + 4L_1)/L \tag{2.35}$$

$$\frac{dN_j}{dx} = (4 - 8L_1)/L \tag{2.36}$$

$$\frac{dN_k}{dx} = (-1 + 4L_1)/L \tag{2.37}$$

Matrix $\left[B^{(e)}\right]$ can be found by putting the equations into vector-matrix form.

$$\begin{Bmatrix} \varepsilon \\ \gamma \\ \kappa \end{Bmatrix} = \begin{bmatrix} N_i' & 0 & 0 & N_j' & 0 & 0 & N_k' & 0 & 0 \\ 0 & N_i' & -N_i & 0 & N_j' & -N_j & 0 & N_k' & -N_k \\ 0 & 0 & N_i' & 0 & 0 & N_j' & 0 & 0 & N_k' \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ \theta_i \\ u_j \\ v_j \\ \theta_j \\ u_k \\ v_k \\ \theta_k \end{Bmatrix} \tag{2.38}$$

What follows is the definition of the constitutive matrix $\left[D^{(e)}\right]$. This matrix does not change with respect to equation (2.19).

In order to evaluate the element stiffness matrix $\left[k^{(e)}\right]$ numerically, two integration points are required, i.e. $L_1 = (-1 + \sqrt{3})/(2\sqrt{3})$ and $L_1 = (1 + \sqrt{3})/(2\sqrt{3})$. A graphical clarification is presented in figure 2.17.

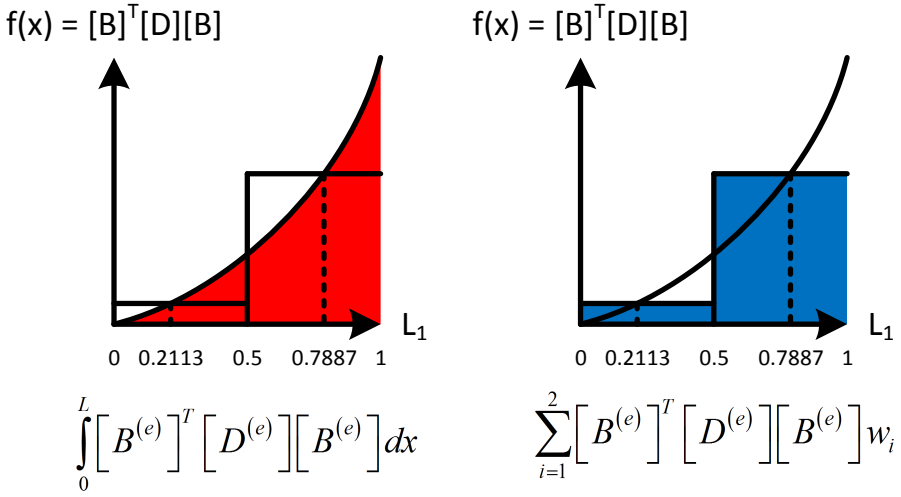

Figure 2.17: Numerical integration for quadratic beam elements

As a weight for both terms, $w_{i=1}$ and $w_{i=2}$ are set to $L/2$.

In order to let $\left[k^{(e)}\right]$ apply to elements with an arbitrary orientation, the lambda matrix needs to be used as shown in (2.20). Matrix $[\lambda]$ is simply extended with the same repetition as defined in equation (2.21).

$$[\lambda] = \begin{bmatrix} l & m & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -m & l & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & l & m & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -m & l & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & l & m & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -m & l & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.39}$$

Now that the local s-t coordinate system applies, the variable x can be replaced by the variable s as shown in line 38 down to 46 which hold the relevant coordinates of the element. Figure 2.18 presents this element.

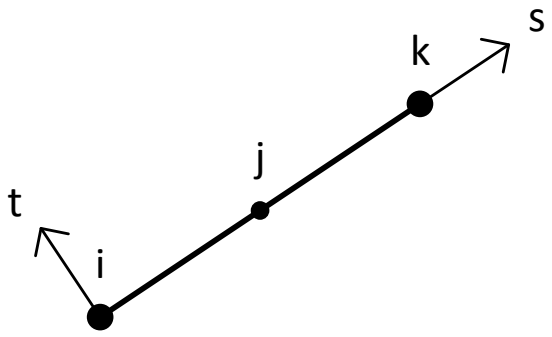

Figure 2.18: Quadratic beam element with arbitrary orientation

The entries vector that is used to substitute the global element stiffness matrix into the global stiffness matrix, has the following systematics.

$$entries = \begin{Bmatrix} 3i-2 \\ 3i-1 \\ 3i \\ 3j-2 \\ 3j-1 \\ 3j \\ 3k-2 \\ 3k-1 \\ 3k \end{Bmatrix} \tag{2.40}$$

Before the loop ends, the global distributed force vector $\{q^{\circ}\}$ is constructed. Equation (2.23) is used for this. Numerical evaluation of the global element distributed force vectors $\{q^{\circ(e)}\}$ is done by using the scheme that is presented in figure 2.17. The entries vector (2.40) is used to substitute the global element distributed force vectors in the global distributed force vector accordingly.

Unlike the program for linear beam elements, FramesQuadratic2d.m will divide the q-load unequally over the nodes i, j and k, i.e. 1/6:2/3:1/6. This is graphically presented in figure 2.19.

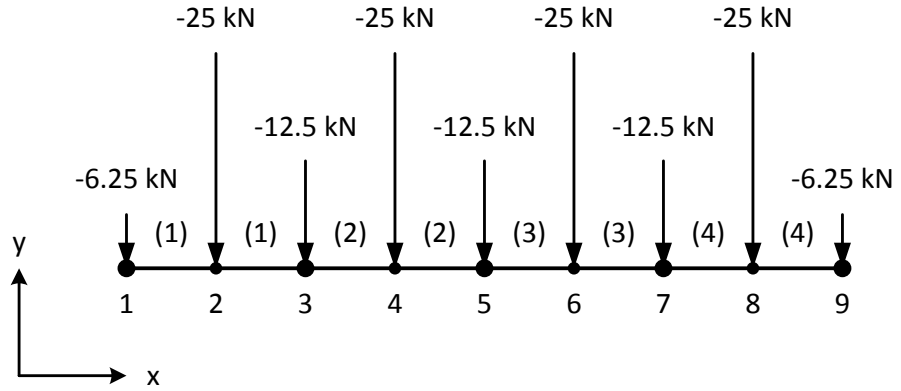

Figure 2.19: Model of the distributed load for quadratic elements

Listing 2.12: Section 2

```
20 %% SECTION 2
21 % Constructing k and q_glob
22
23 for n = 1:numelem/2
24     x_i = nodes(elements(2*n-1,1),1);
25     y_i = nodes(elements(2*n-1,1),2);
26     x_k = nodes(elements(2*n,2),1);
27     y_k = nodes(elements(2*n,2),2);
28
29     L = sqrt((x_k-x_i)^2+(y_k-y_i)^2);
30     l = (x_k-x_i)/L;
31     m = (y_k-y_i)/L;
```

```
32
33      E = elements(2*n-1,3);
34      G = E/(2*(1+0.3));
35      A = elements(2*n-1,4);
36      I = elements(2*n-1,5);
37
38      s_i = 0;
39      s_a = (-1+sqrt(3))/(2*sqrt(3))*L;
40  %   first integration point
41      s_b = (1+sqrt(3))/(2*sqrt(3))*L;
42  %   second integration point
43      L_1_a = (s_a-s_i)/L;
44  %   natural coordinate of first integration point
45      L_1_b = (s_b-s_i)/L;
46  %   natural coordinate of second integration point
47
48      N_i_a = 1-3*L_1_a+2*L_1_a^2;  N_i_b = 1-3*L_1_b+2*L_1_b^2;
49      N_j_a = 4*L_1_a-4*L_1_a^2;    N_j_b = 4*L_1_b-4*L_1_b^2;
50      N_k_a = -L_1_a+2*L_1_a^2;     N_k_b = -L_1_b+2*L_1_b^2;
51
52      dN_i_a = (-3+4*L_1_a)/L;      dN_i_b = (-3+4*L_1_b)/L;
53      dN_j_a = (4-8*L_1_a)/L;       dN_j_b = (4-8*L_1_b)/L;
54      dN_k_a = (-1+4*L_1_a)/L;      dN_k_b = (-1+4*L_1_b)/L;
55
56      lambda = zeros(9,9);
57      lambda(1:3,1:3) = [l m 0; -m l 0; 0 0 1];
58      lambda(4:6,4:6) = [l m 0; -m l 0; 0 0 1];
59      lambda(7:9,7:9) = [l m 0; -m l 0; 0 0 1];
60
61      entries = [3*elements(2*n-1,1)-2; 3*elements(2*n-1,1)-1;...
62                 3*elements(2*n-1,1);...
63                 3*elements(2*n-1,2)-2; 3*elements(2*n-1,2)-1;
64                 3*elements(2*n-1,2);...
65                 3*elements(2*n,2)-2; 3*elements(2*n,2)-1;...
66                 3*elements(2*n,2)];
67
68      q = [elements(2*n-1,6); elements(2*n-1,7); 0];
69
70      N_a = [N_i_a 0 0 N_j_a 0 0 N_k_a 0 0;...
71             0 N_i_a 0 0 N_j_a 0 0 N_k_a 0;...
72             0 0 N_i_a 0 0 N_j_a 0 0 N_k_a];
73      N_b = [N_i_b 0 0 N_j_b 0 0 N_k_b 0 0;...
74             0 N_i_b 0 0 N_j_b 0 0 N_k_b 0;...
75             0 0 N_i_b 0 0 N_j_b 0 0 N_k_b];
76
77      B_a = [dN_i_a 0 0 dN_j_a 0 0 dN_k_a 0 0;...
78             0 dN_i_a -N_i_a 0 dN_j_a -N_j_a 0 dN_k_a -N_k_a;...
79             0 0 dN_i_a 0 0 dN_j_a 0 0 dN_k_a]*lambda;
80      B_b = [dN_i_b 0 0 dN_j_b 0 0 dN_k_b 0 0;...
81             0 dN_i_b -N_i_b 0 dN_j_b -N_j_b 0 dN_k_b -N_k_b;...
82             0 0 dN_i_b 0 0 dN_j_b 0 0 dN_k_b]*lambda;
83
84      D = [E*A 0 0; 0 G*A 0; 0 0 E*I];
85
86      k(entries,entries) = k(entries,entries)+...
87                           transpose(B_a)*D*B_a*L/2+...
88                           transpose(B_b)*D*B_b*L/2;
89      q_glob(entries) = q_glob(entries)+...
90                        transpose(N_a)*q*L/2+...
91                        transpose(N_b)*q*L/2;
92  end
```

With regard to section 9, one remark need to be made. Where the internal forces are more accurate for linear elements by adding the discretizised q-load, for quadratic elements this artifice is not applied, because it does not contribute to the interpretation of the normal force distribution and the shear force distribution. A closer look will be taken at this in the third chapter.

Chapter 3

Analyses of elementary structures

Now that the programs are explained, it is time to use them and analyse the result. For that, three elementary frame structures are chosen; the simply supported beam, as presented in the previous chapter (see figure 2.10 and 2.15), a cantilever beam loaded by a point load at the unsupported end and a statically indeterminate beam. Starting with the simply supported beam, the output for the structure described by four elements (linear and quadratic) will be analysed first. What are the nodal displacements and how to interpret the internal forces N, S and M_z? Is there a difference between linear and quadratic elements? Also, how do these forces convergence when the mesh size, i.e. the amount of elements, is increased? Secondly, the influence of the mesh size and the element type on the accuracy of the displacements is investigated, as well as the influence on the computation time.

3.1 Simply supported beam

3.1.1 Output

The first set of answers that the frame programs give, after the input is processed and computations are made, are the internal forces per element in N and Nmm. This is presented in figure 3.1 and 3.2. Column one denotes which element it regards. The second column tells at the side of which node the internal forces apply. As expected, there are no normal forces present. Figure 3.3 and 3.4 show the tabulated nodal quantities. The second, third and fourth column present the nodal displacements in mm and radians, the fifth, sixth, seventh, eighth and ninth column the nodal loads in N and Nmm and the tenth, eleventh and twelfth column the reaction forces in N and Nmm. To interpret the interplay of forces more easily, a graphical presentation is given in figure 3.5 and 3.6. The sign convention corresponds to the local coordinate system which in this case is parallel to the global coordinate system. kN and kNm are the units.

```
>> FramesLinear2d

ans =

    element    node    N       S          M_z_int
    _______    ____    _    _______    ____________

    1          1       0      75000    -9.5367e-07
    1          2       0     -75000     1.4062e+08


ans =

    element    node    N       S          M_z_int
    _______    ____    _    _______    ____________

    2          2       0      37500    -1.4062e+08
    2          3       0     -37500      1.875e+08


ans =

    element    node    N       S          M_z_int
    _______    ____    _    _______    ___________

    3          3       0     -37500    -1.875e+08
    3          4       0      37500    1.4062e+08


ans =

    element    node    N       S          M_z_int
    _______    ____    _    _______    ____________

    4          4       0     -75000    -1.4063e+08
    4          5       0      75000    -4.7684e-07
```

Figure 3.1: Non-graphical output FramesLinear2d.m: internal forces

```
>> FramesQuadratic2d

ans =

    element    node    N       S        M_z_int
    _______    ____    _    _______    __________

    1          1       0     68750     2.3842e-07
    1          2       0    -25000     5.9605e-07
    1          3       0    -43750     1.4063e+08


ans =

    element    node    N       S        M_z_int
    _______    ____    _    _______    __________

    2          3       0     31250    -1.4063e+08
    2          4       0    -25000     3.5986e-07
    2          5       0     -6250      1.875e+08


ans =

    element    node    N       S        M_z_int
    _______    ____    _    _______    __________

    3          5       0     -6250     -1.875e+08
    3          6       0    -25000    -3.5763e-07
    3          7       0     31250     1.4063e+08


ans =

    element    node    N       S        M_z_int
    _______    ____    _    _______    __________

    4          7       0    -43750    -1.4063e+08
    4          8       0    -25000     2.3842e-07
    4          9       0     68750    -9.5367e-07
```

Figure 3.2: Non-graphical output FramesQuadratic2d.m: internal forces

ans =

node	u	v	theta	F_x	F_y	T_z	q_x	q_y	R_x	R_y	M_z_ext
1	0	0	-0.003208	0	0	0	0	-18750	0	75000	0
2	0	-6.9052	-0.0022456	0	0	0	0	-37500	0	0	0
3	0	-9.7416	2.1524e-18	0	0	0	0	-37500	0	0	0
4	0	-6.9052	0.0022456	0	0	0	0	-37500	0	0	0
5	0	0	0.003208	0	0	0	0	-18750	0	75000	0

Figure 3.3: Non-graphical output FramesLinear2d.m: nodal quantities

ans =

node	u	v	theta	F_x	F_y	T_z	q_x	q_y	R_x	R_y	M_z_ext
1	0	0	-0.0034219	0	0	0	0	-6250	0	75000	0
2	0	-4.1951	-0.0031278	0	0	0	0	-25000	0	0	0
3	0	-7.7072	-0.0023526	0	0	0	0	-12500	0	0	0
4	0	-10.002	-0.0012565	0	0	0	0	-25000	0	0	0
5	0	-10.811	7.0563e-17	0	0	0	0	-12500	0	0	0
6	0	-10.002	0.0012565	0	0	0	0	-25000	0	0	0
7	0	-7.7072	0.0023526	0	0	0	0	-12500	0	0	0
8	0	-4.1951	0.0031278	0	0	0	0	-25000	0	0	0
9	0	0	0.0034219	0	0	0	0	-6250	0	75000	0

Figure 3.4: Non-graphical output FramesQuadratic2d.m: nodal quantities

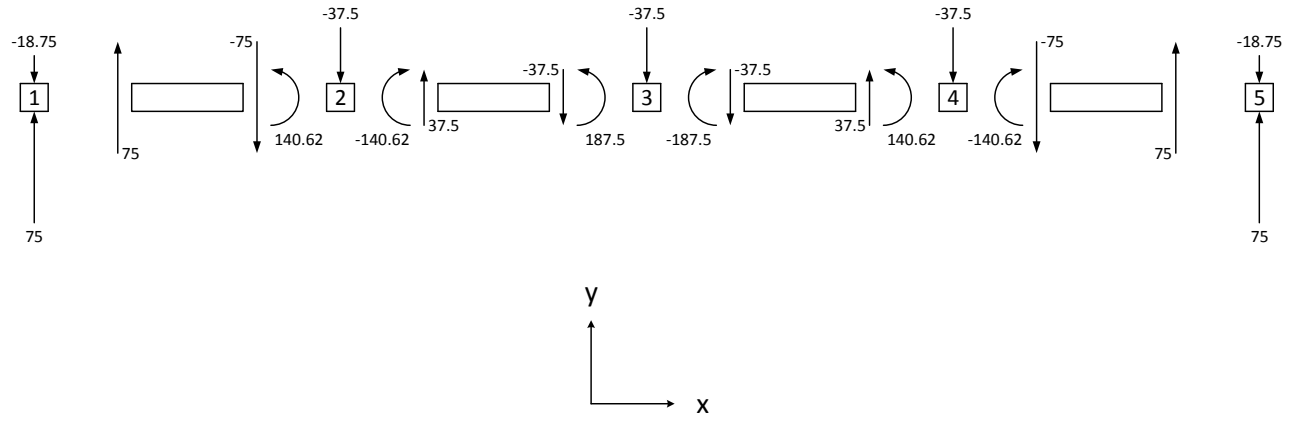

Figure 3.5: Graphical interpretation of figure 3.1 and 3.3

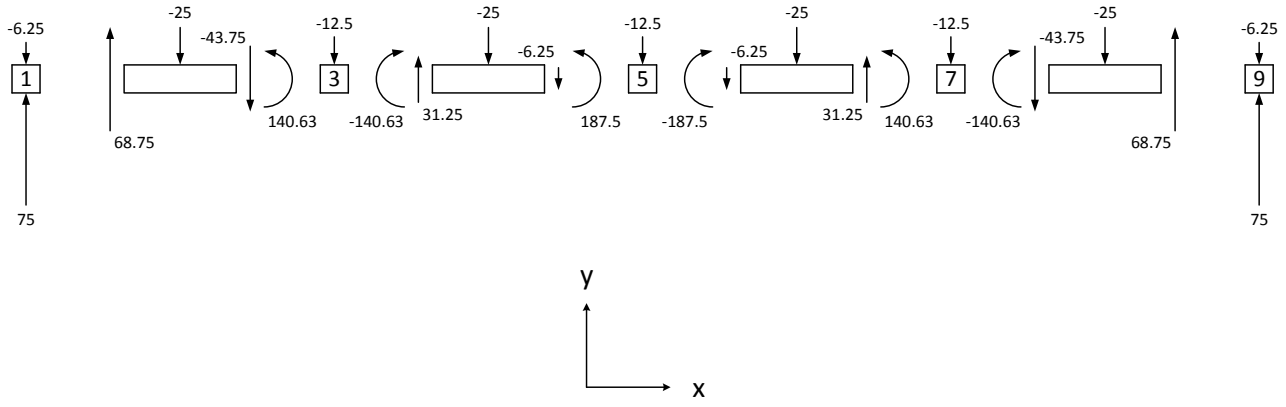

Figure 3.6: Graphical interpretation of figure 3.2 and 3.4

As mentioned in paragraph 2.2.1 in the discussion of the internal forces for linear elements, the equally concentrated q-load is added to the internal forces in order to come to a more accurate result. In paragraph 2.2.2 it was remarked that this artifice is not beneficial to the interpretation of the N- and S-distribution in quadratic elements. The reason for this is that the computed internal forces at the middle node of each element can be interpreted as the concentrated q-load on that element. Compare the shear forces at the middle nodes 2, 4, 6 and 8 as presented in figure 3.2 with the according concentrated q-loads as presented in figure 3.4. To make this more comprehensible, a graphical clarification is shown in figure 3.7.

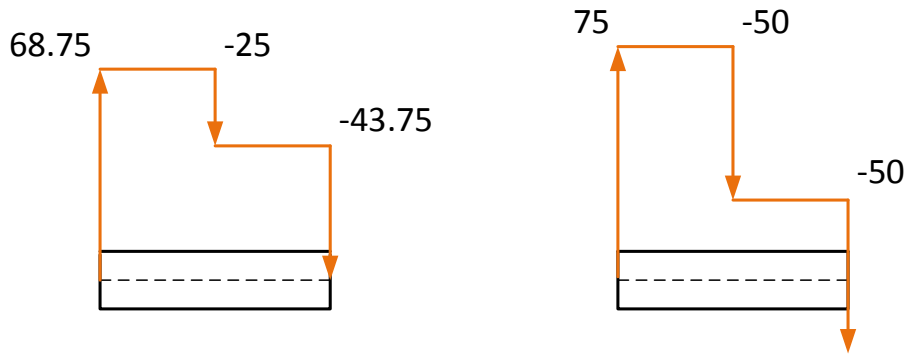

Figure 3.7: S-distribution, quadratic element 1, with (r) and without (l) artifice

The right graphic of figure 3.7 shows the shear force distribution in quadratic element 1 with the added concentrated q-load in the ratio of 1/6:2/3:1/6. As can be clearly seen, this does not give an enclosed polygon which is an incorrect interpretation. The correct interpretation is shown at the left where the polygon is enclosed.

Figure 3.8 and 3.9 present the shear force distribution within the entire structure modelled by linear and quadratic elements respectively. A red line

is added which indicates the analytical distribution of the shear force: $S = q/2(l - 2x)$. For the derivation, see chapter 11 of [3]. It needs to be noted that the global vertical axis in this derivation is pointing downwards. Increasing the mesh size, results in a better approximation of the red line. See figure 3.10 and 3.11.

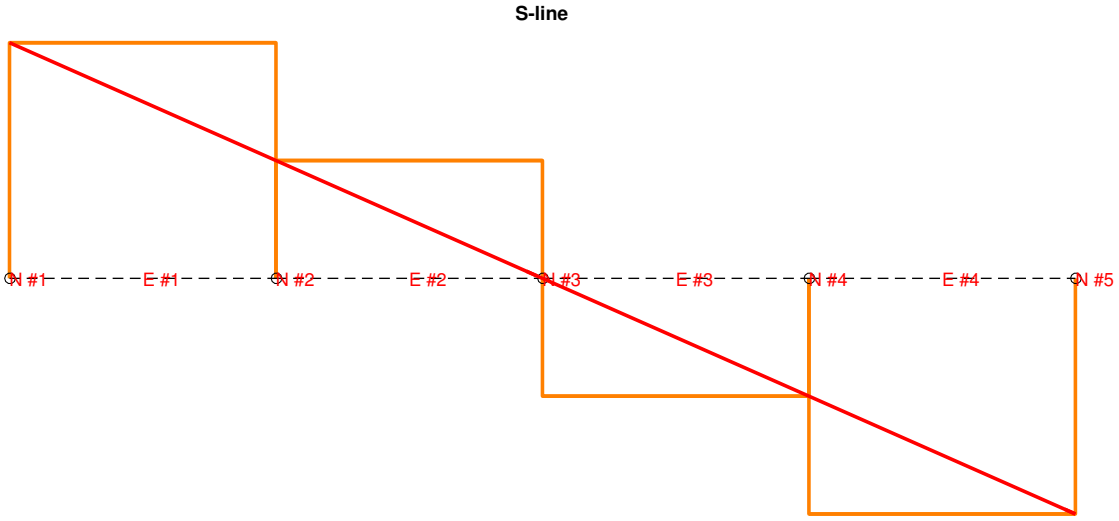

Figure 3.8: S-distribution, simply supported beam, 4 linear elements

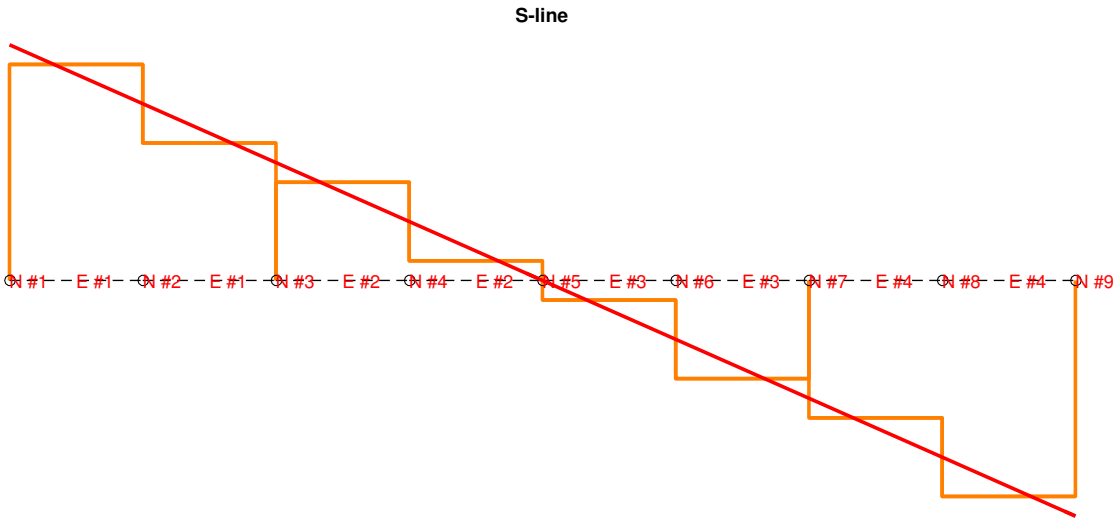

Figure 3.9: S-distribution, simply supported beam, 4 quadratic elements

For this structure, in the case of linear elements, the shear force at both ends are exact, unlike quadratic elements where S at the nodes deviates from the exact distribution. However, the exact shear force of the inner nodes can be found by taking the mean of the upper and lower value of the approximation line at the considered node.

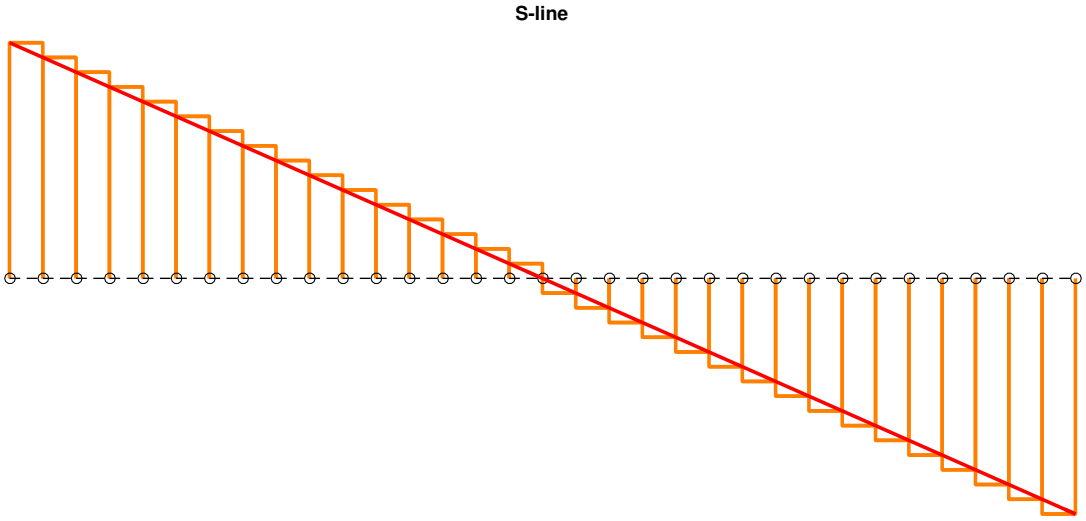

Figure 3.10: S-distribution, simply supported beam, 32 linear elements

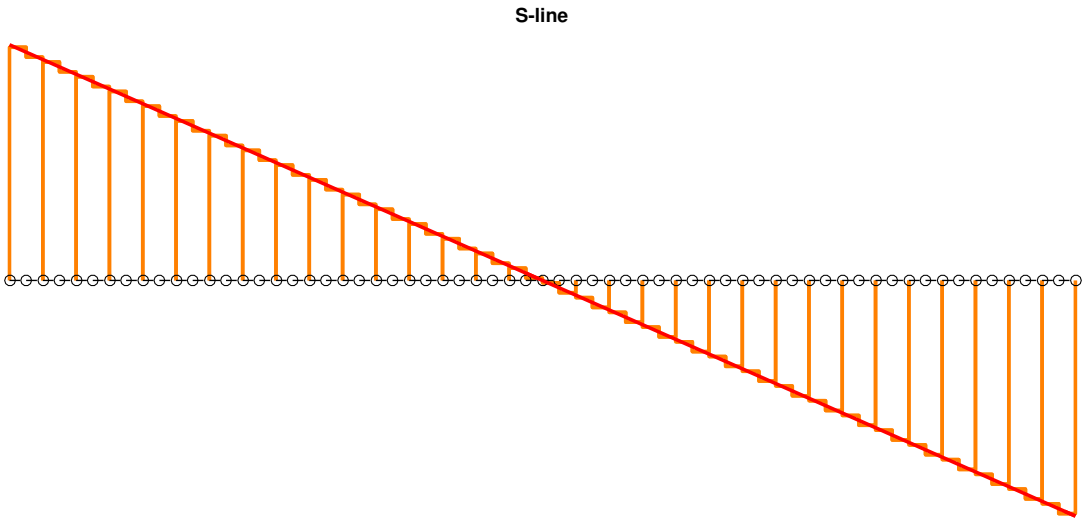

Figure 3.11: S-distribution, simply supported beam, 32 quadratic elements

Polygons can also give a graphical presentation of the bending moment distribution as shown in figure 3.12. This distribution is the same for both linear and quadratic elements, because the bending moment at the middle node of each quadratic element is negligible as can be seen in figure 3.2. The red line indicates the analytical distribution of the bending moment: $M_z = qx/2(l-x)$. For the derivation, see chapter 11 of [3]. A better approximation will be obtained when the mesh size is increased. See figure 3.13.

Unlike the shear force values, the bending moment values are exact at the outer nodes of quadratic elements.

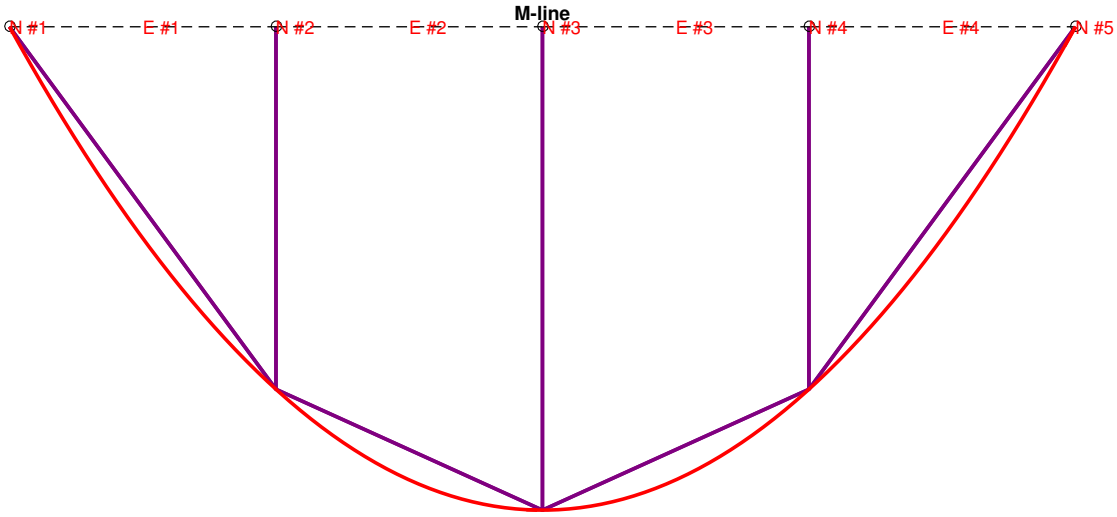

Figure 3.12: M-distribution, simply supported beam, 4 elements

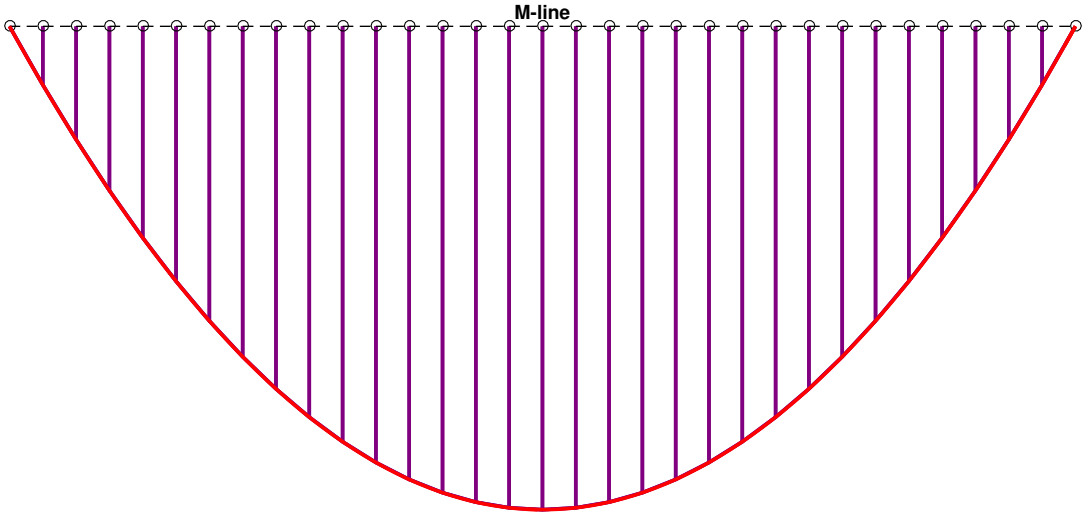

Figure 3.13: M-distribution, simply supported beam, 32 elements

Figure 3.14 and 3.15 present the strains for linear and quadratic elements respectively. Note that these strains are computed at the location of the integration points. Getting closer the middle of the span, one can observe that the shear strain γ decreases in absolute value. The reason for this can be found in equation (2.19); S is linearly related to γ by GA. Looking at the shear distribution in the previous figures, S decreases more and more when getting closer to the middle of the span. The same observation can be made for the bending moment M_z, only now the corresponding strain κ (i.e. the curvature in mm^{-1}) increases towards to middle of the span. From equation (2.19): M_z is linearly related to κ by EI.

```
ans =
```

element	epsilon	gamma	kappa
1	0	-3.5255e-05	3.8496e-07
2	0	-1.1752e-05	8.9825e-07
3	0	1.1752e-05	8.9825e-07
4	0	3.5255e-05	3.8496e-07

Figure 3.14: Strains of linear elements

ans =

element	point	epsilon	gamma	kappa
1	1	0	-4.204e-05	2.0548e-07
1	2	0	-2.847e-05	6.4999e-07
2	1	0	-1.8537e-05	8.6693e-07
2	2	0	-4.9668e-06	1.0151e-06
3	1	0	4.9668e-06	1.0151e-06
3	2	0	1.8537e-05	8.6693e-07
4	1	0	2.847e-05	6.4999e-07
4	2	0	4.204e-05	2.0548e-07

Figure 3.15: Strains of quadratic elements

A graphical presentation of the undeformed and deformed structure is presented in figure 3.16 and 3.17. The red line presents the analytical description of the deflection: $v = qx/(24EI)(x^3 - 2lx^2 + l^3)$. Paragraph 4.11 of [4] presents the differential equation from which this description is derived. One can conclude at a glance that the four quadratic elements approximate the deflection line more accurately than the four linear elements. This will be analysed in more detail in the next sub-paragraph.

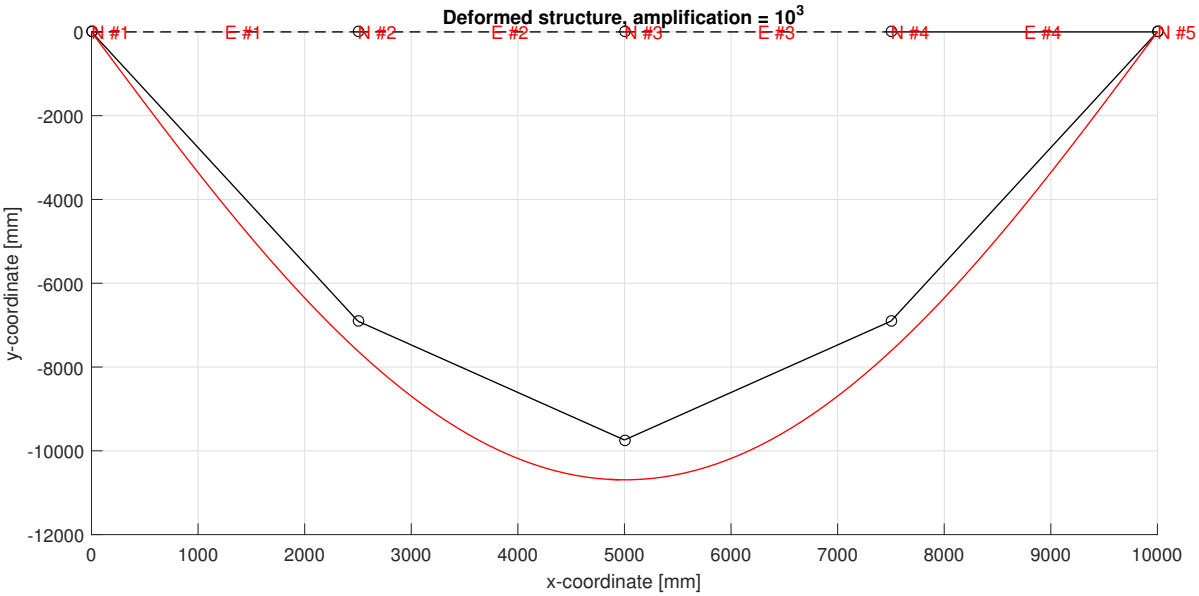

Figure 3.16: Deflection of the simply supported beam, 4 linear elements

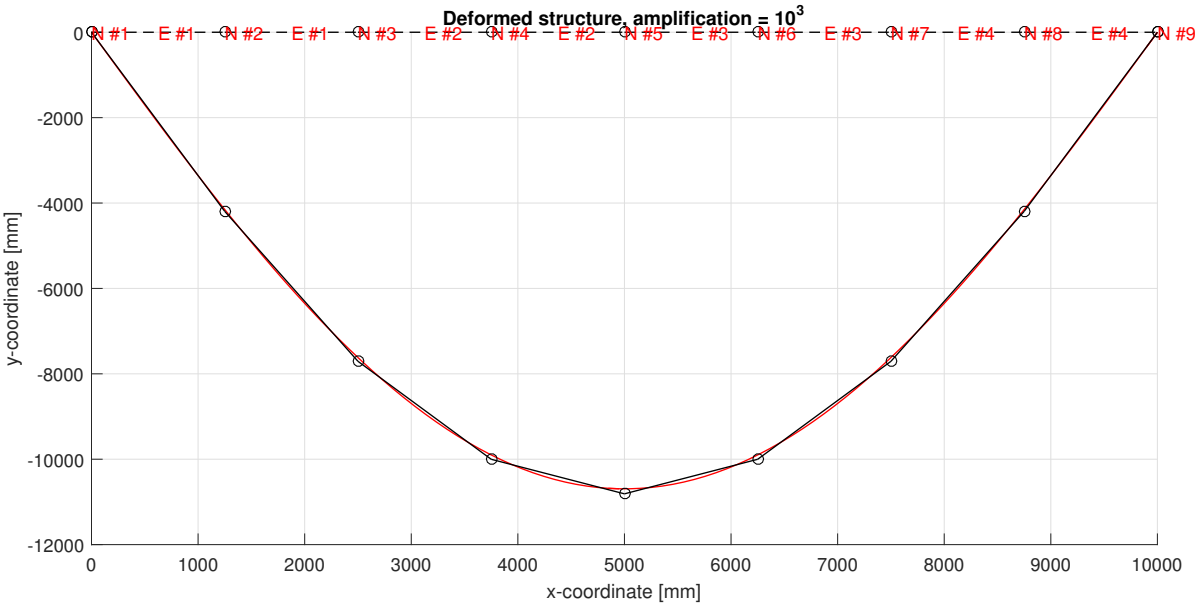

Figure 3.17: Deflection of the simply supported beam, 4 quadratic elements

3.1.2 Convergence behaviour

To say something quantitatively about the convergence behaviour of the simply supported beam, different descriptive files are evaluated by the programs; each successive file describing the structure by more elements. For both element types, the following mesh sizes are used: 2, 4, 8, 16, 32, 64, 128 and 256. Focussing on the middle node, the following deflections and computation durations are obtained.

	A	B	C	D	E
1		deflection at midspan (mm)		computation time (s)	
2	mesh size	linear	quadratic	linear	quadratic
3	2	-6.5336	-10.811	1.319	1.429
4	4	-9.7416	-10.811	1.409	1.548
5	8	-10.544	-10.811	1.535	1.776
6	16	-10.744	-10.811	1.863	2.325
7	32	-10.794	-10.811	2.431	3.319
8	64	-10.807	-10.811	3.651	5.264
9	128	-10.81	-10.811	5.724	8.827
10	256	-10.811	-10.811	10.122	17.162

Figure 3.18: Convergence of deflection at midspan and computation durations

Before this data is presented graphically, one important observation can be made already; it takes 128 to 256 linear elements to obtain the most accurate midspan deflection of -10.811 mm, whereas with quadratic elements it only takes two.

Based on the corresponding computation times of 5.724 to 10.122 seconds and 1.429 seconds respectively, it would be more economical to model the structure by two quadratic elements in order to obtain the deflection at midspan.

Because the quadratic deflection does not convergence any further, only the linear deflection at midspan is presented graphically. This is done in two different ways. First, dividing the linear approximation by the analytical value gives the graph as shown in figure 3.19. The analytical deflection value is obtained from the description as noted in the previous sub-paragraph: v_analytical = -10.6934 mm.

The linear approximation exceeds the analytical value from 16 elements. This phenomenon can also be observed at the other nodes. See figure 3.20; the black line hangs slightly under the red line with a maximum difference at midspan. Why is this?

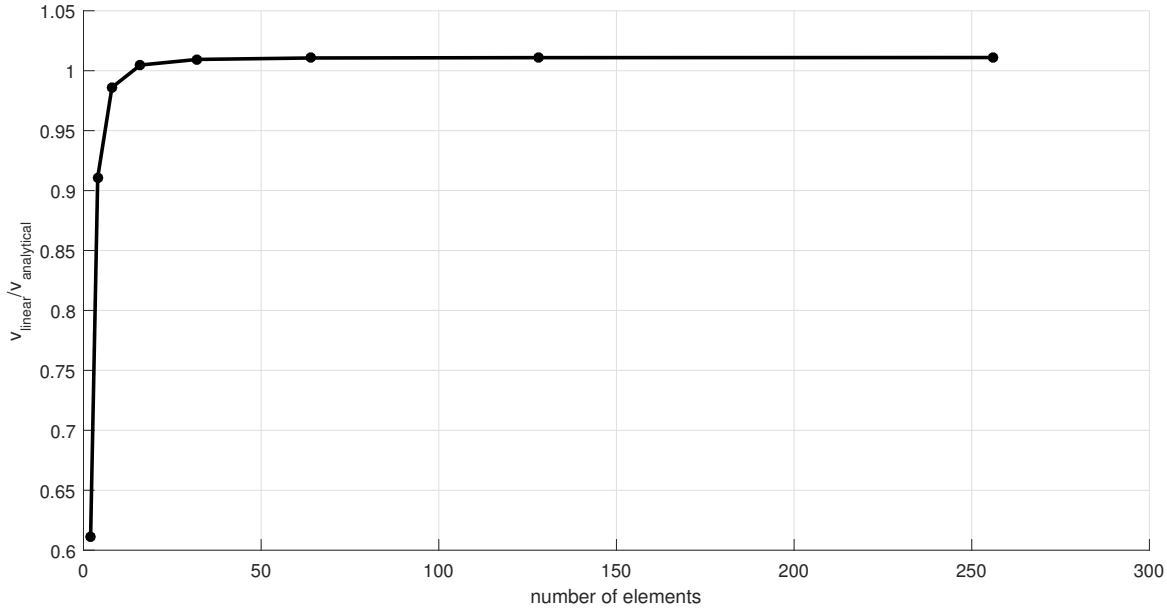

Figure 3.19: Convergence of FE to analytical (= 1) midspan deflection

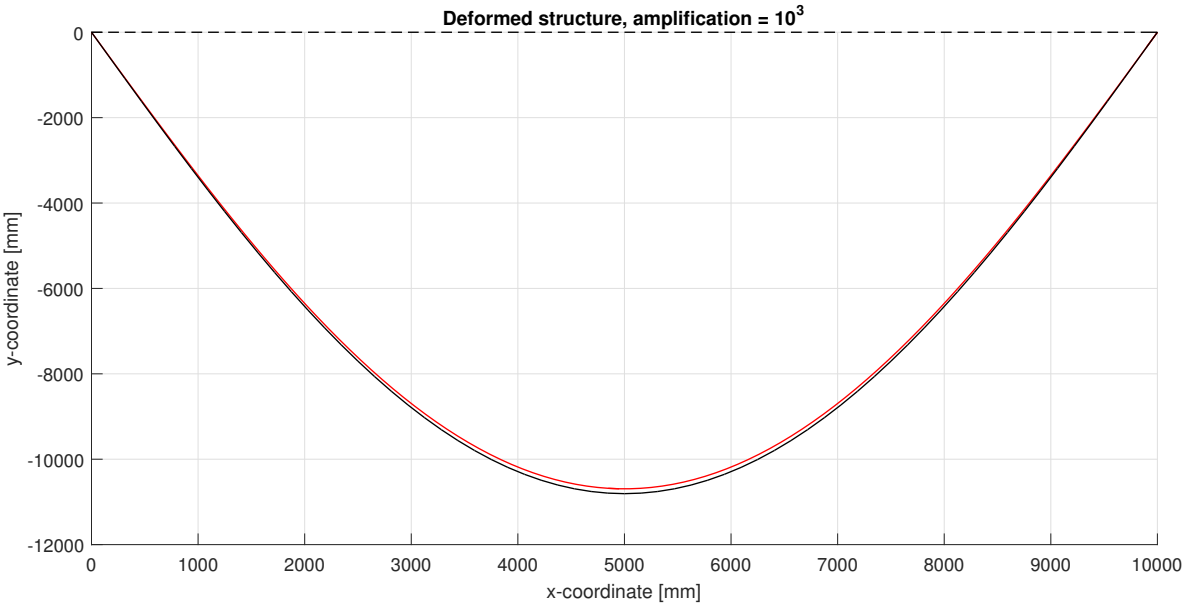

Figure 3.20: Deflection of the simply supported beam, 64 linear elements

To answer this question, reference is made to paragraph 11.2 of [2]. The reason for this discrepancy is the different underlying beam models. In the analytical description the Bernoulli-Euler beam is used which ignores the influence of the transverse shear stresses on the deformation, i.e. each cross section does not rotate around an axis that is parallel to the z-axis and passes through the centroid of the cross section. This relates to the kinematic assumption that is fundamental for the Bernoulli-Euler derivation: cross sections remain plane and normal to the deformed longitudinal axis. For the finite element programs the Timoshenko beam is applied which does take the shear deformation into account; cross sections do remain plane, but do not remain normal to the deformed longitudinal axis.

The requirement of normality and the corresponding neglected shear deformation in the Bernoulli-Euler model gives it a higher stiffness in comparison to the Timoshenko model. So, one would expect that by increasing the shear modulus G the deflection of the latter model converges to that of the former. This is indeed the case; G (= 80769 N/mm^2) is multiplied by 2, 4, 8, 16 and 32 and the corresponding quotients of the FE and analytical deflections are presented in figure 3.21. Figure 3.22 shows the graphical output.

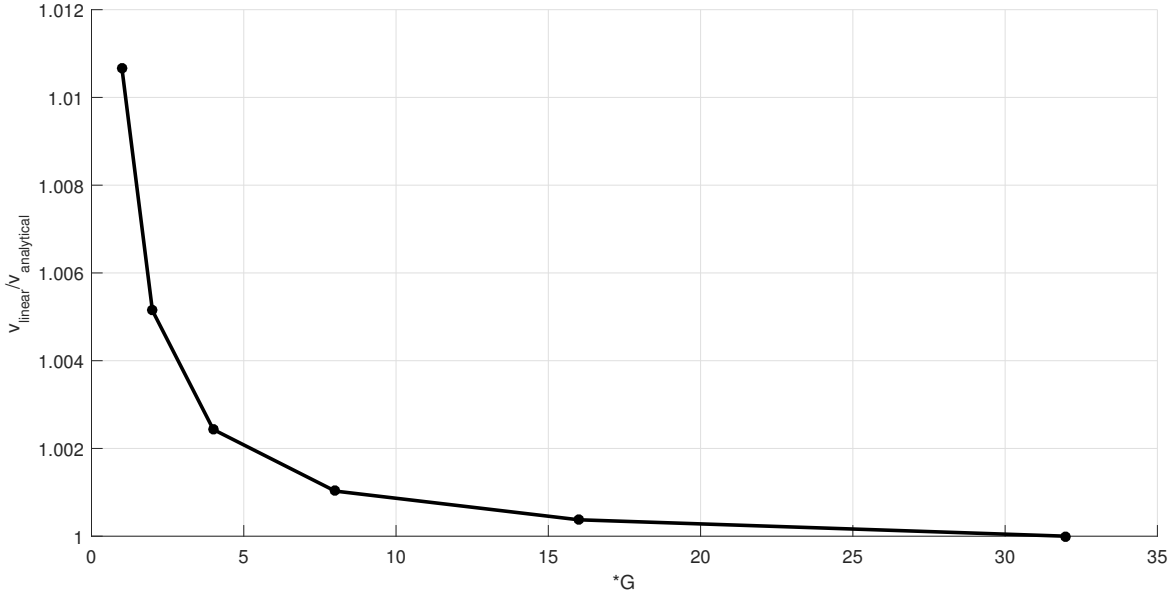

Figure 3.21: Influence of the shear modulus on midspan deflection

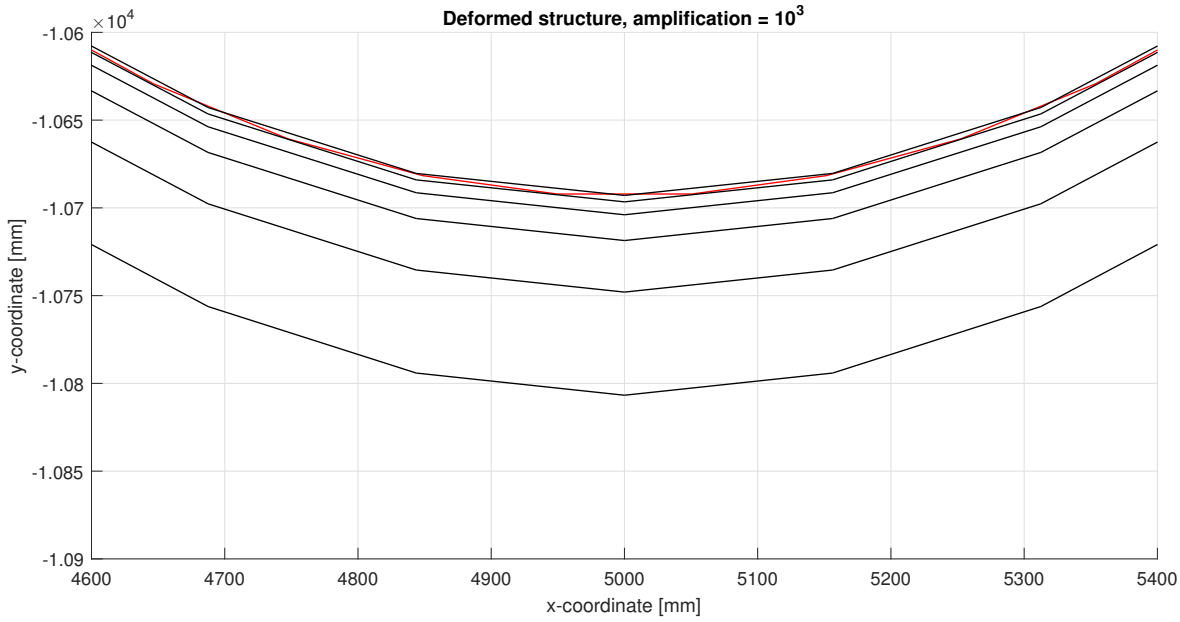

Figure 3.22: From bottom to top: $G/2G/4G/8G/16G/32G$, 64 linear elements

Another way to present the data of figure 3.18 is by calculating the absolute difference with the most accurate value of -10.811 mm and plotting the resultant decreasing error in logspace.

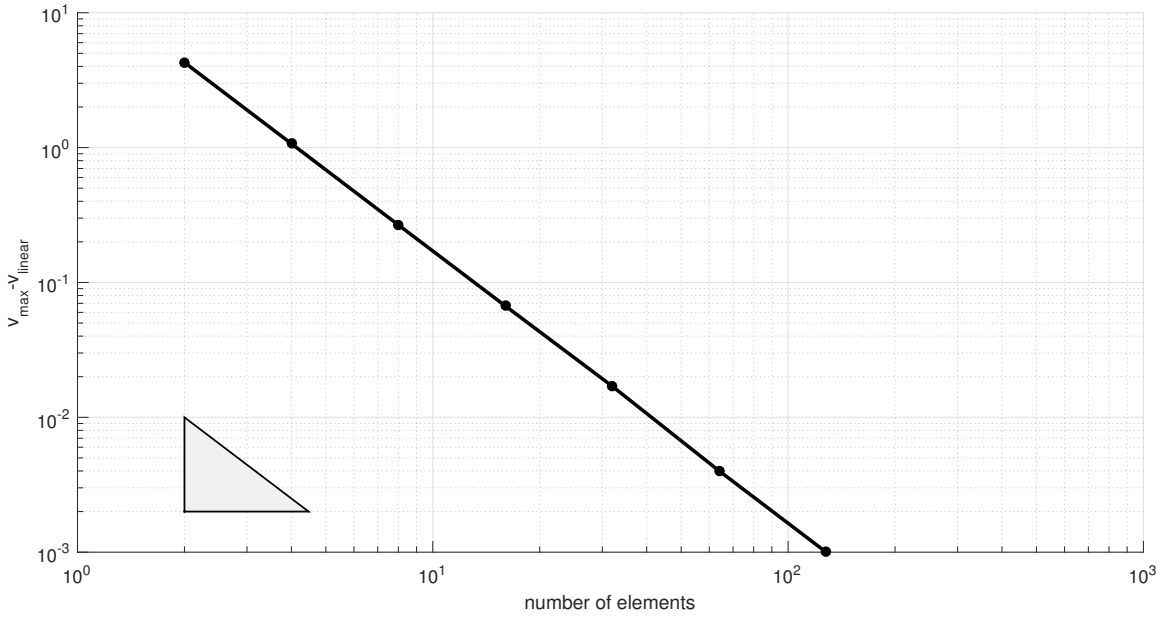

Figure 3.23: Error convergence

The tangent of this graph is a quantitative measure for the convergence rate, ranging between 0.0 and $-\infty$. In this case the slope is -2.0 which is emphasized by the grey polygon with the same tangent. Regarding the computation efficiency, one can state that the steeper the log/log-curve, the better.

3.2 Cantilever beam

The structure that is analysed next is a cantilever beam loaded by a point load of 100 kN in the negative y-direction, exerted at the unsupported end. See figure 3.24.

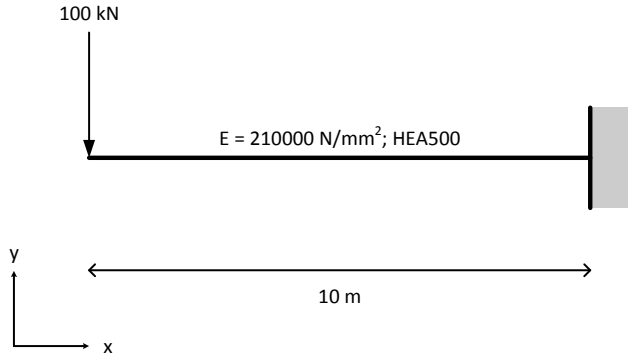

Figure 3.24: Cantilever beam

To determine the accuracy of the finite element approximation for the different quantities, the following analytical expressions are used of which the underlying differential equations can be found in chapter 11 of [3] and paragraph 4.11 of [4].

$$S = -F \tag{3.1}$$

$$M_z = -Fx \tag{3.2}$$

$$\theta = \frac{F}{2EI_{yy}}(x^2 - l^2) \tag{3.3}$$

$$v = \frac{F}{6EI_{yy}}(x^3 - 3l^2x + 2l^3) \tag{3.4}$$

Describing the structure by linear and quadratic elements approximates the internal forces as shown in the previous paragraph, i.e. the exact value of S and M_z at both nodes of a linear element and the exact value of M_z at the outer nodes of a quadratic element. However, there is one difference; now that the shear distribution is constant along the longitudinal axis of the structure, a quadratic element shows the exact S-value at the outer nodes as well.

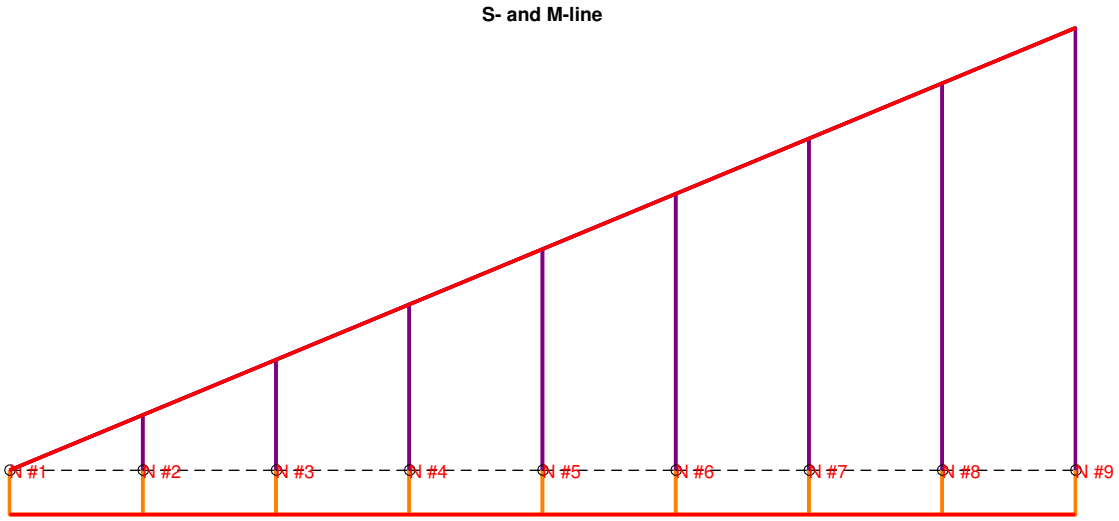

Figure 3.25: S- and M-distribution, cantilever beam, 8 linear elements

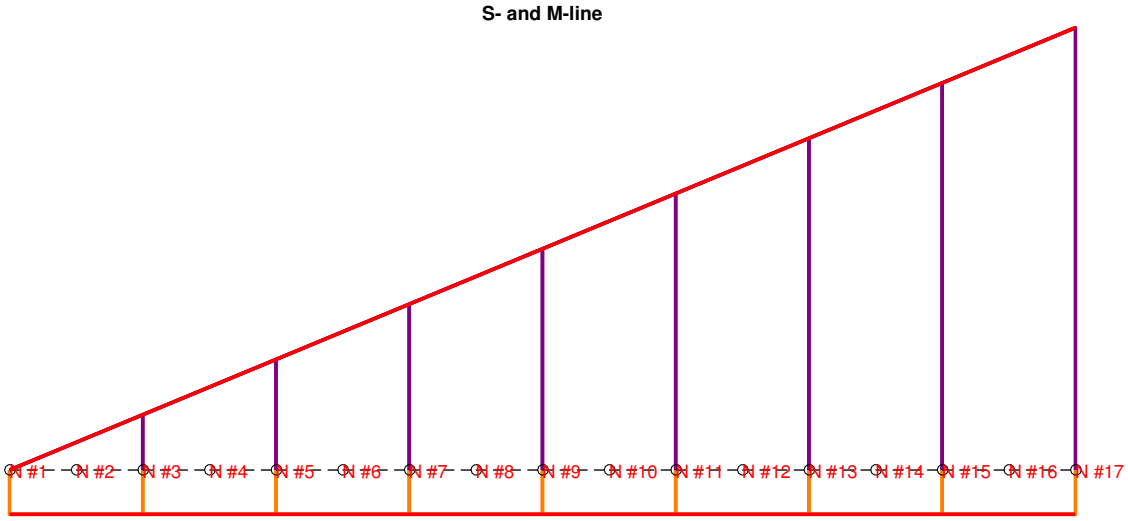

Figure 3.26: S- and M-distribution, cantilever beam, 8 quadratic elements

The next quantities that will be analysed are the angle of rotation θ and the deflection v at the unsupported end and at midspan. Figure 3.27 shows the obtained values in a spreadsheet. As can be observed right away, the quadratic elements give the most accurate approximation already with the smallest mesh size. This also applies for linear elements in the case of θ.

	A	B	C	D	E	F	G
1		deflection at unsupported end (mm)		rotation at unsupported end (radians)		computation time (s)	
2	mesh size	linear	quadratic	linear	quadratic	linear	quadratic
3	2	-171.72	-183.13	0.027375	0.027375	1.401	1.523
4	4	-180.28	-183.13	0.027375	0.027375	1.494	1.593
5	8	-182.41	-183.13	0.027375	0.027375	1.727	2.008
6	16	-182.95	-183.13	0.027375	0.027375	1.934	2.499
7	32	-183.08	-183.13	0.027375	0.027375	2.665	3.452
8	64	-183.12	-183.13	0.027375	0.027375	3.744	5.405
9	128	-183.12	-183.13	0.027375	0.027375	5.991	9.193
10	256	-183.13	-183.13	0.027375	0.027375	10.338	17.069
11		deflection at midspan (mm)		rotation at midspan (radians)			
12	mesh size	linear	quadratic	linear	quadratic		
13	2	-51.642	-57.345	0.020531	0.020531		
14	4	-55.919	-57.345	0.020531	0.020531		
15	8	-56.988	-57.345	0.020531	0.020531		
16	16	-57.256	-57.345	0.020531	0.020531		
17	32	-57.323	-57.345	0.020531	0.020531		
18	64	-57.339	-57.345	0.020531	0.020531		
19	128	-57.344	-57.345	0.020531	0.020531		
20	256	-57.345	-57.345	0.020531	0.020531		

Figure 3.27: Convergence of deformation quantities and computation durations

For this structure as well, the deflections approximated with the finite element programs exceed the analytical value of -182.5009 mm at the unsupported end and -57.0315 mm at midspan. By increasing the shear modulus, the most accurate finite element approximation approaches the analytical value. Figure 3.28 presents the deflection line modelled by 64 quadratic elements and a G-modulus of 80769 N/mm^2.

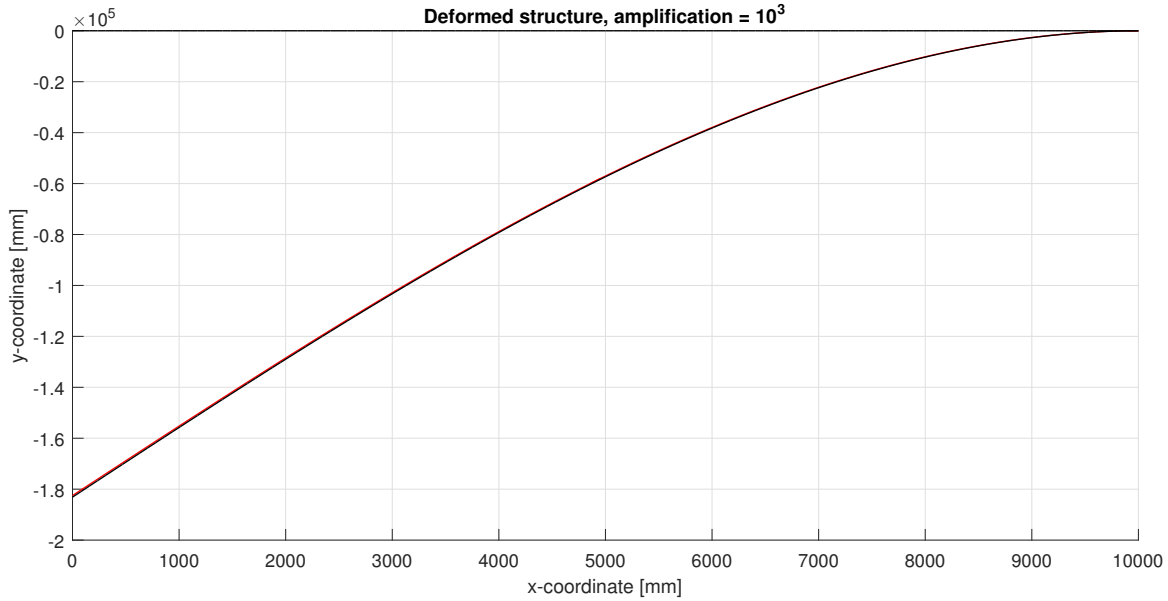

Figure 3.28: Deflection of the cantilever beam, 64 quadratic elements

Figure 3.29 presents the convergence behaviour of both deflections, modelled by linear elements. Normalising the error enables one to make a comparison which tells that the convergence proceeds similar, almost identical up to 64 linear elements, but shows an increasing difference when the mesh becomes finer. At such small values it is assumed that this has a computational reason, i.e. numerical precision. The tangent of the overlapping parts is -2.0.

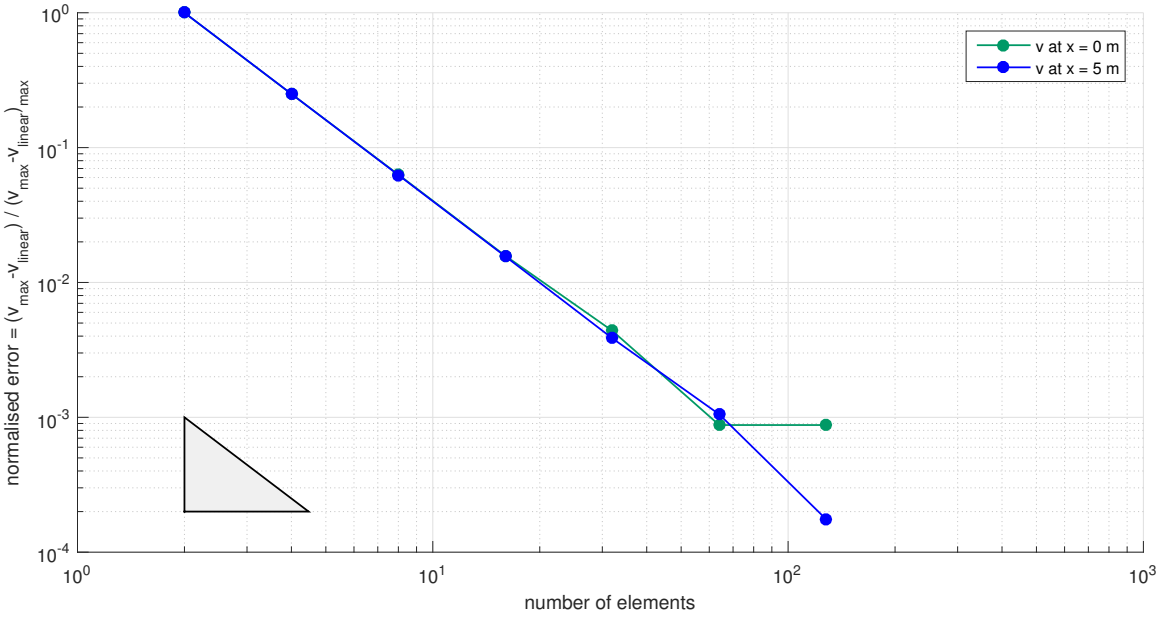

Figure 3.29: Error convergence of both deflections

The discrepancy between the finite element and analytical result does not apply to the angle of rotation. The FE approximation is, regardless of the mesh size and element type, identical to the analytical value; 0.027375 radians at the unsupported end and 0.020531 radians at midspan. Why is this?

Reference is made to paragraph 11.2 of [2] to answer this question. Depending on the used beam model, the calculated angle θ consists of either one or two components. In case of the Bernoulli-Euler beam, the rotation of the longitudinal axis ψ is equal to θ; the angle that one computes with the analytical description. An extra component is considered when the Timoshenko beam is applied: the shear distortion angle γ. This quantity describes the rotation of the cross-section relative to its non-displaced configuration. Apparently, for this particular structure, the value of γ is negligible to null; the cross-sections at both ends of each element remain parallel to the global vertical axis during shear deformation. A graphical presentation of θ along the entire structure is presented in figure 3.30.

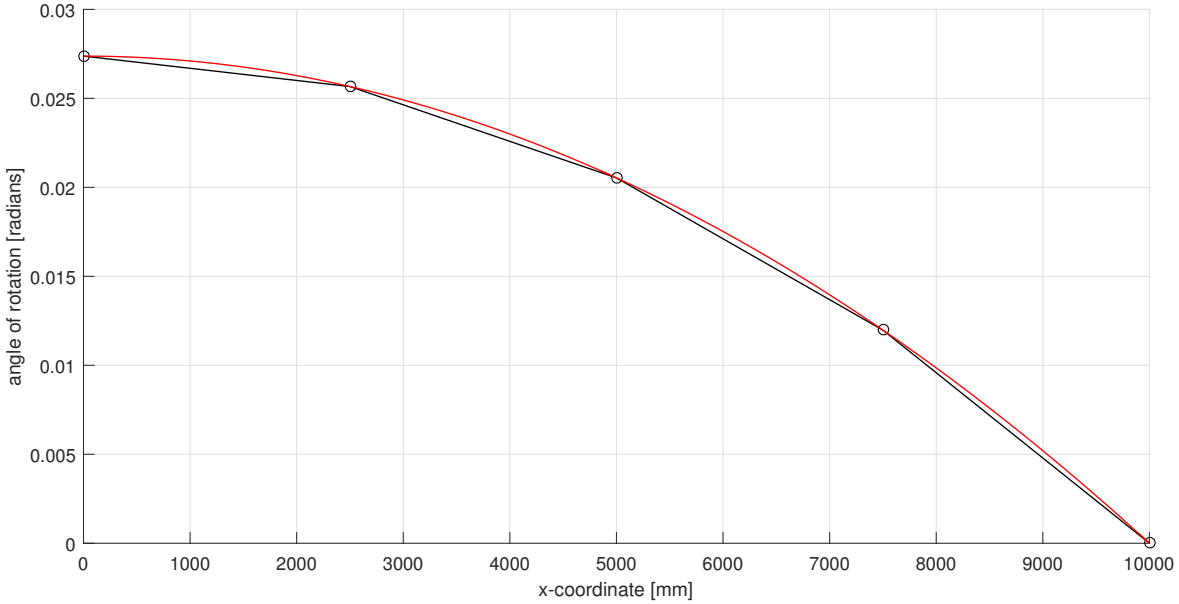

Figure 3.30: θ-diagram, 4 linear elements

3.3 Statically indeterminate beam

The final structure that will be analysed is a statically indeterminate beam, fixed at the left side and supported by a roller at the right where a couple of 250 kNm is applied clockwise. The cross-sectional and material properties are the same as in the previous frame structures.

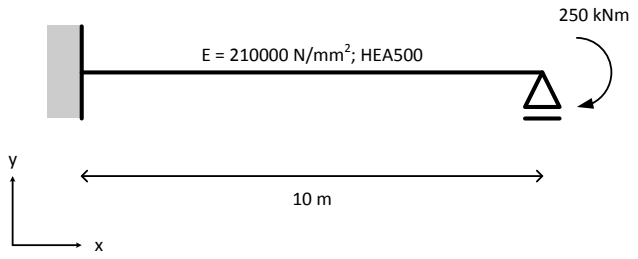

Figure 3.31: Statically indeterminate beam

To determine the accuracy of the finite element approximation for the different quantities, the following analytical expressions are used of which the underlying differential equations can be found in chapter 11 and paragraph 4.11 of [3] and

[4] respectively.

$$S = -\frac{3T_z}{2l} \tag{3.5}$$

$$M_z = \frac{T_z}{2l}(-3x + l) \tag{3.6}$$

$$\theta = \frac{T_z x}{4EI_{yy}l}(3x - 2l) \tag{3.7}$$

$$v = \frac{T_z x^2}{4EI_{yy}l}(x - l) \tag{3.8}$$

Starting with internal forces, both linear and quadratic elements generate the same output because of the S-distribution being constant along the longitudinal axis of the beam. There is one obscurity however to be observed from the diagram in figure 3.32; the M_z-value is slightly off at the left side of the structure, gradually overlapping the analytical distribution more and more when moving towards the right.

Figure 3.32: S- and M-distribution, statically indeterminate beam, 64 elements

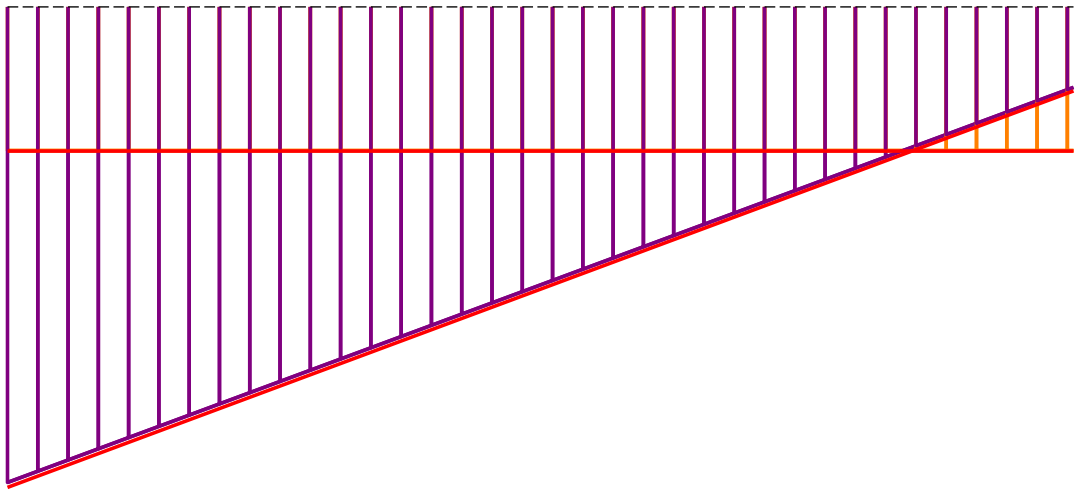

Figure 3.33: Discrepancy between FE and analytical distribution of M_z

Looking at the convergence of the reaction forces at the fixed end as presented in the spreadsheet of figure 3.34, one comes to the conclusion that this does not only apply to the bending moment, but also to the shear force; note the convergence of the vertical reaction force R_y in the spreadsheet. Why is this?

The fact that the structure is statically indeterminate is the reason why; such structures can have any internal force distribution if only equilibrium is considered. However, this is not the case; constitutive and kinematic conditions need to be met as well, i.e. the deformed configuration of the structure is of influence. So, the FE approximation of the quantities v and θ not corresponding exactly to their analytical counterparts means the FE approximation of the force quantities not corresponding exactly to their analytical values: R_y = -37.5 kN and $M_{z,ext}$ = -125 kNm. In accordance with the analyses of the simply supported beam, this can be demonstrated by increasing the shear modulus G for a fine mesh, let's take 128 quadratic elements. See figure 3.35.

	A	B	C	D	E	F	G
1		rotation at roller (radians)		deflection at midspan (mm)		computation time (s)	
2	mesh size	linear	quadratic	linear	quadratic	linear	quadratic
3	2	-0.0027775	-0.003457	3.4718	4.3213	1.417	1.521
4	4	-0.0032952	-0.003457	4.119	4.3213	1.516	1.679
5	8	-0.003417	-0.003457	4.2713	4.3213	1.617	1.911
6	16	-0.0034471	-0.003457	4.3088	4.3213	1.972	2.373
7	32	-0.0034545	-0.003457	4.3182	4.3213	3.047	3.373
8	64	-0.0034564	-0.003457	4.3205	4.3213	3.646	5.418
9	128	-0.0034569	-0.003457	4.3211	4.3213	5.918	9.216
10	256	-0.003457	-0.003457	4.3212	4.3213	10.38	16.829
11		R_y at fixed end (kN)		M_{z,ext} at fixed end (kNm)			
12	mesh size	linear	quadratic	linear	quadratic		
13	2	-39.854	-37.372	-148.54	-123.72		
14	4	-37.963	-37.372	-129.63	-123.72		
15	8	-37.518	-37.372	-125.18	-123.72		
16	16	-37.408	-37.372	-124.08	-123.72		
17	32	-37.381	-37.372	-123.81	-123.72		
18	64	-37.374	-37.372	-123.74	-123.72		
19	128	-37.372	-37.372	-123.72	-123.72		
20	256	-37.372	-37.372	-123.72	-123.72		

Figure 3.34: Convergence data and computation durations

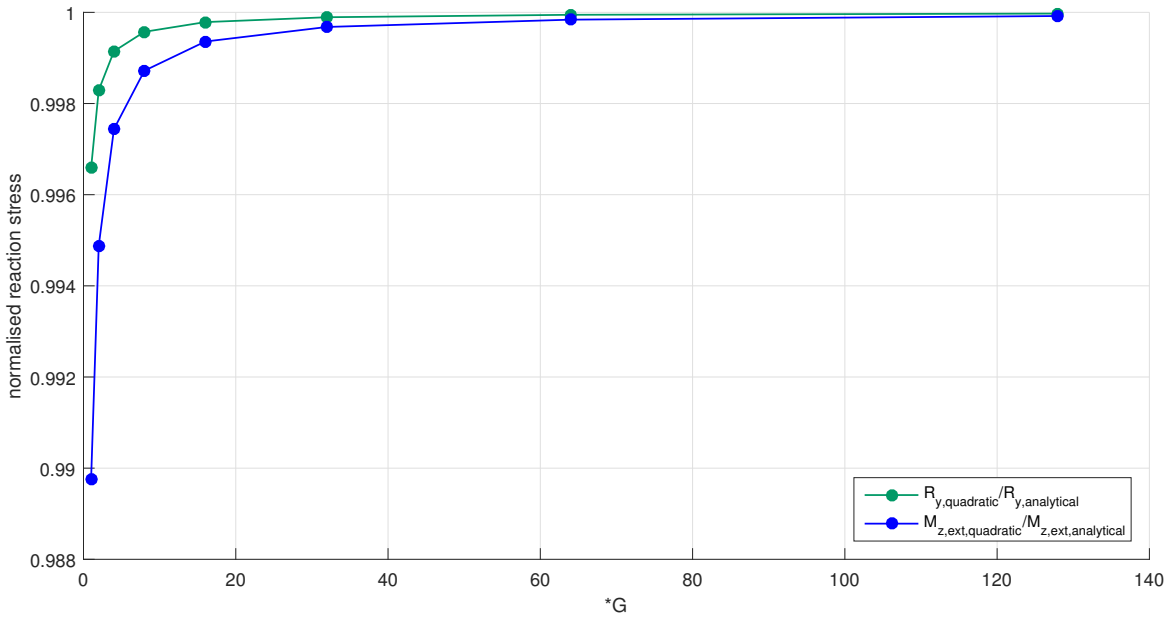

Figure 3.35: Convergence of reaction forces with increasing shear modulus

Continuing with the values of the spreadsheet, quadratic elements, again, give already the most accurate approximation with a mesh size of two. This applies for the deformation quantities as well. With linear elements, an amount of 256 is required to come to an equally accurate value as obtained with two quadratic elements. The analytical value of the deflection at midspan and the angle of rotation at the roller is 4.2774 mm and -0.003422 radians respectively.

Figure 3.38 presents the normalised error convergence of all four quantities. Just like the previous structure, all quantities converge with the same slope, but an increasing mesh size results in an increasing discrepancy due to numerical precision. The tangent of the overlapping parts is -2.0.

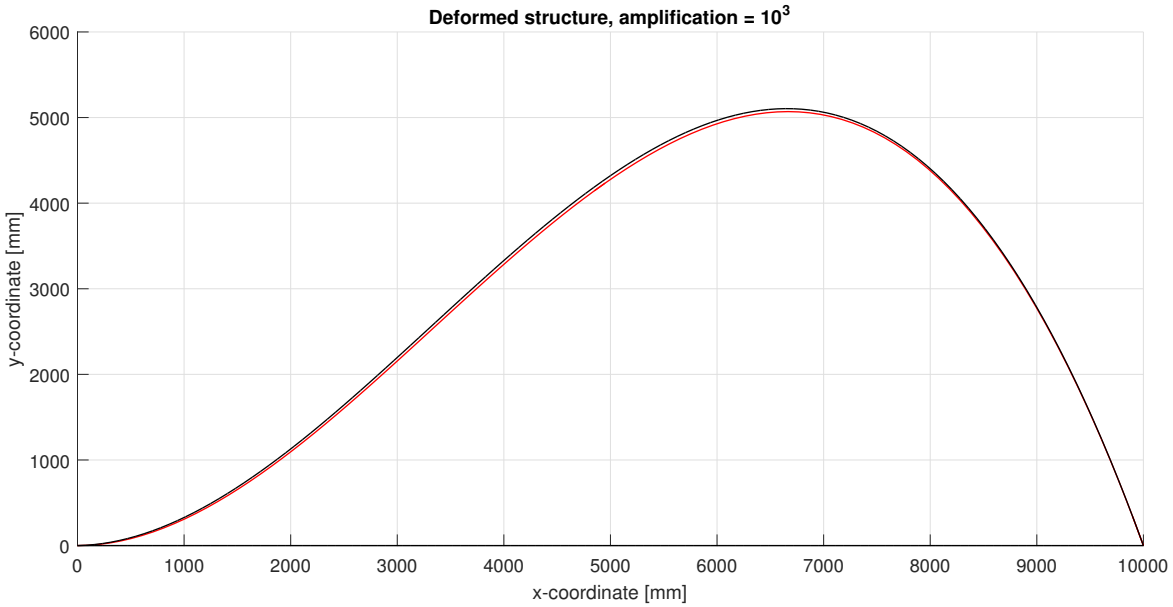

Figure 3.36: Deflection of the beam, 64 quadratic elements

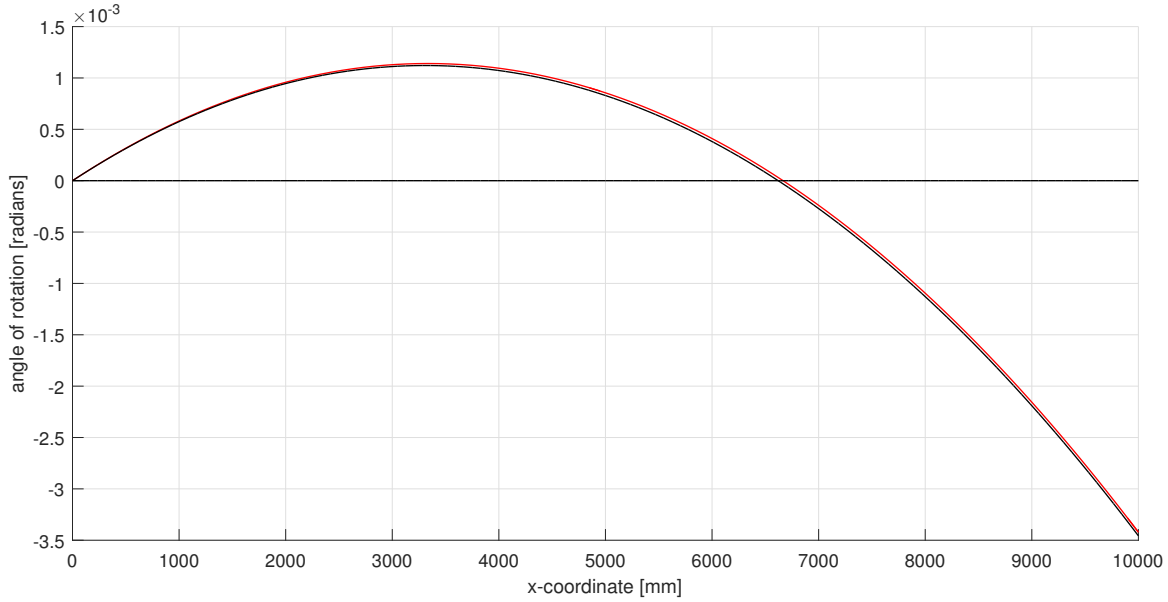

Figure 3.37: θ-diagram, 64 quadratic elements

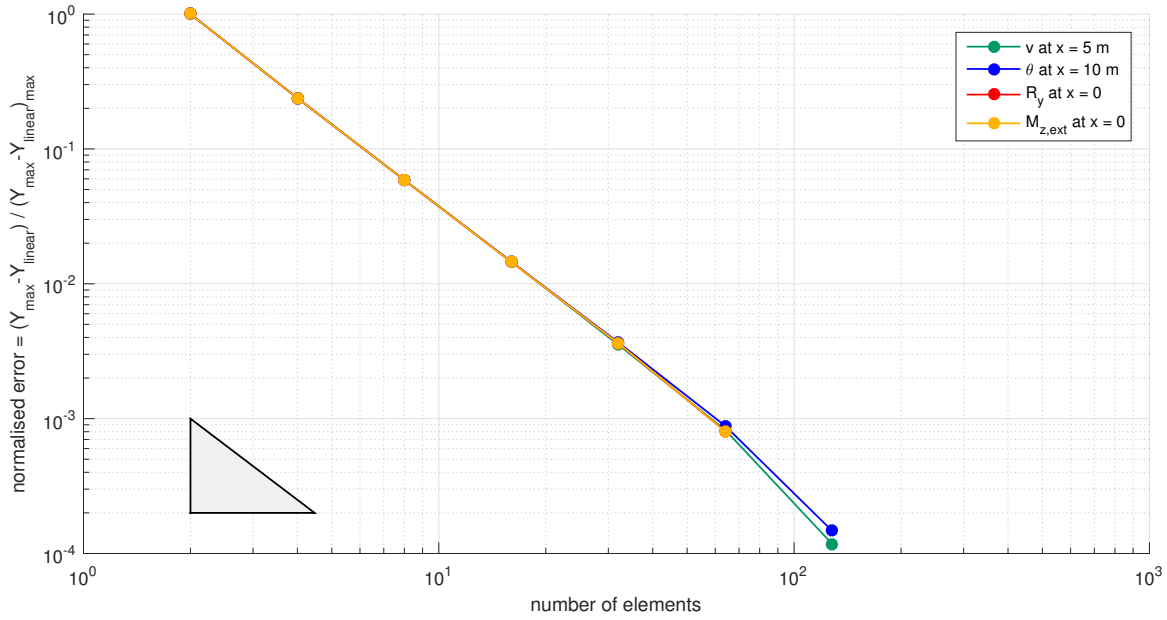

Figure 3.38: Error convergence of all four quantities

3.4 Linear versus quadratic

The question whether to use linear or quadratic elements can only be answered if the nodes of interest are known. For a limited amount of nodes, quadratic elements will suffice; the approximations are the most accurate and the computation time is limited. However, if the number of nodes becomes greater it will be more efficient to use linear elements, because of the equally accurate approximation (starting from 256 elements) and the relatively limited computation time.

A trend that can be observed at all three structures as analysed in the previous paragraphs, is that the duration increases linearly with increasing mesh size and that for quadratic elements the gradient has a higher value and is always located above the one of its linear counterpart. Figure 3.39 shows the progression of the computation time for both element types in the case of the cantilever beam.

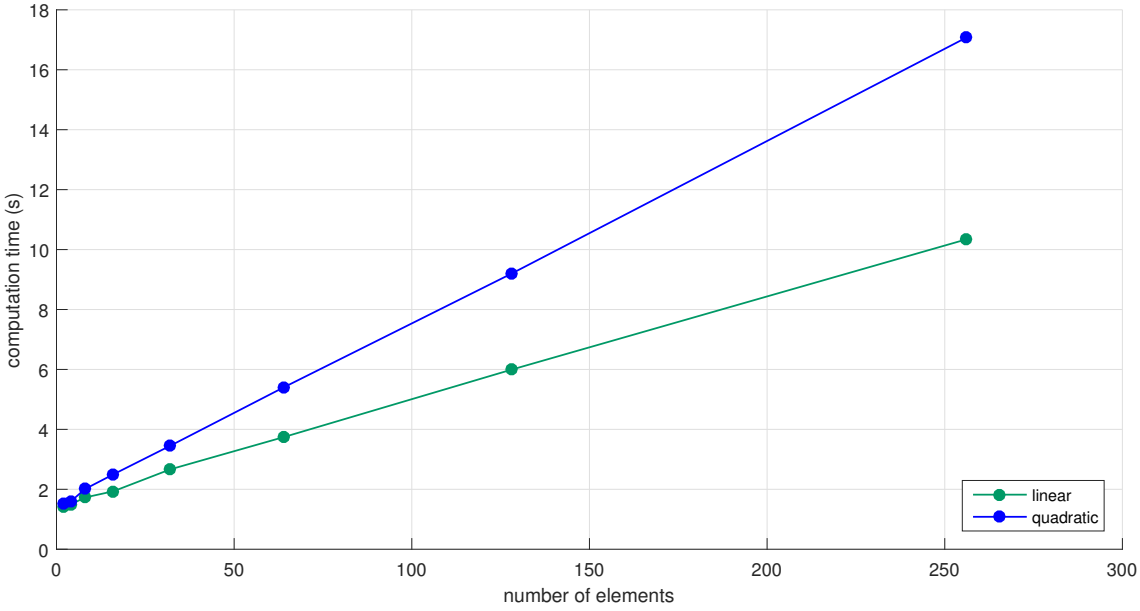

Figure 3.39: Progression of the computation time, cantilever beam

Chapter 4

Five compatible structures

The aesthetic is elevated by evaluating multiple member structures instead of single member structures. With a total of five, this chapter starts with a bridge structure upon which two different uniformly distributed loads are exerted. A comparison will be made with the commercial finite element software 'Matrixframe'[1]. For the second structure, the initial truss example will be revisited (figure 2.1). How do the internal forces change if the pin joints were to be replaced by joints that are able to transfer bending moments? In the third paragraph a portal structure is investigated by analysing the influence of the bending stiffness EI_{yy} on the M-distribution and the deflection line. This is followed by a simple roof structure which will be used to explain how a q-load can be related to the local s-t coordinate system. The chapter will end with an observation of how to program responds if a structure becomes kinematic indeterminate. Where needed, explanation will be provided.

4.1 Structure 1

Figure 4.1 presents the structure of which each member is modelled by 128 quadratic elements having an E-modulus of $2.1 \cdot 10^5$ N/mm^2, a cross-sectional area of $1.9754 \cdot 10^4$ mm^2 and a second moment of area of $8.6975 \cdot 10^8$ mm^4. The graphical output of both Matrixframe and FramesQuadratic2d.m will be shown, i.e. the S-line, the M-line and the deflection line. Also, a comparison will be made between the programs with regard to the quantities that are related to the node which connects all four members. The resultant discrepancy is due to the chosen mesh size. See figure 4.9.

[1]Downloaded from: `http://www.matrix-software.com/`

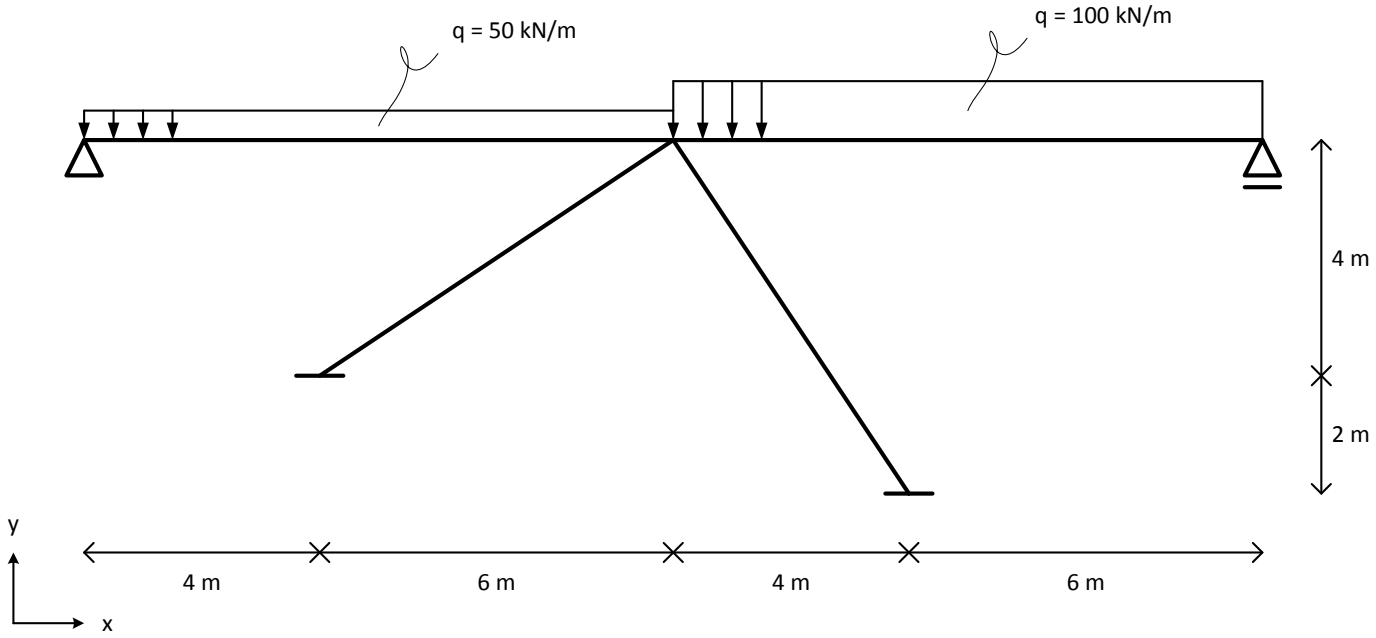

Figure 4.1: Bridge structure

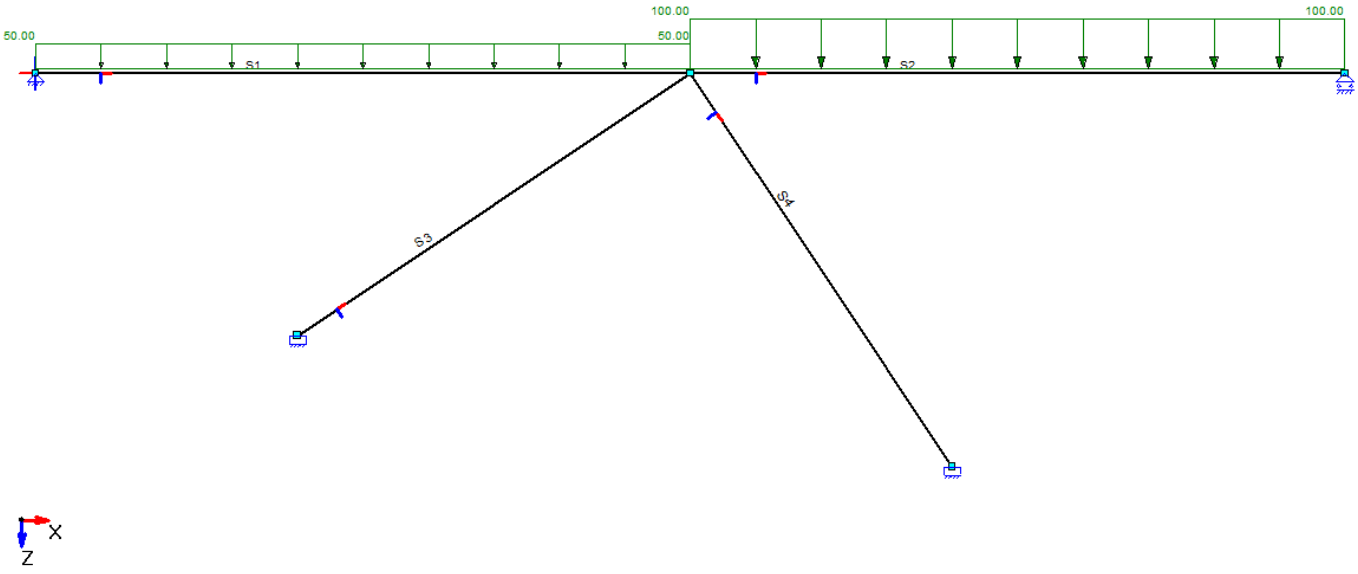

Figure 4.2: Matrixframe model

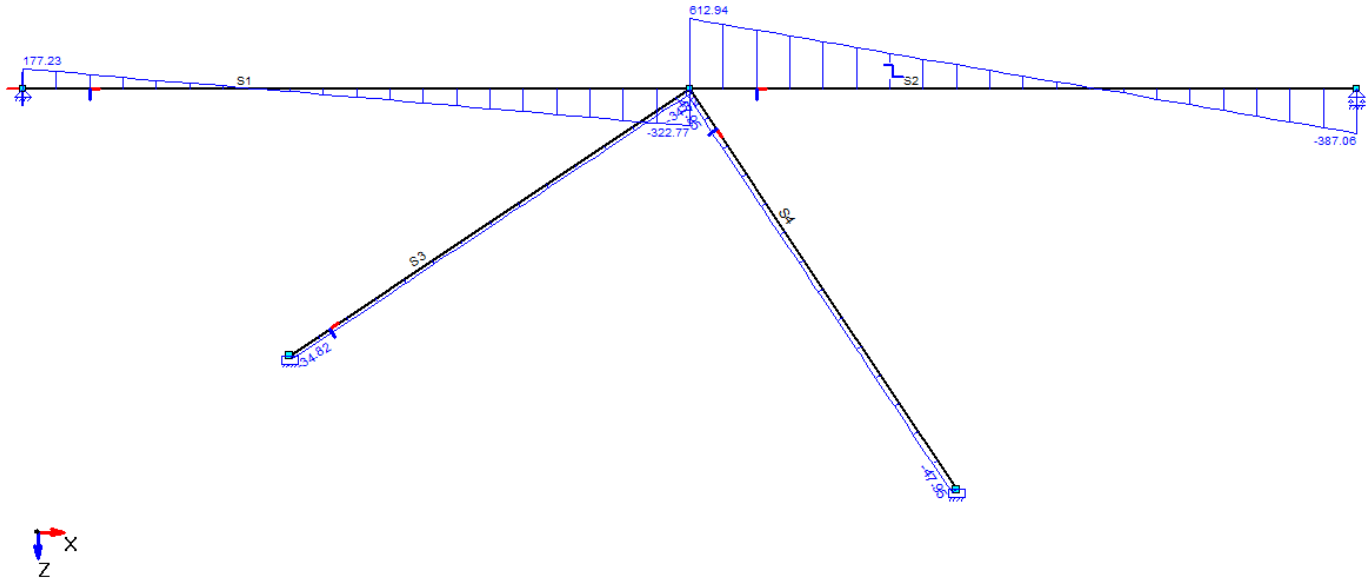

Figure 4.3: S-distribution according to Matrixframe

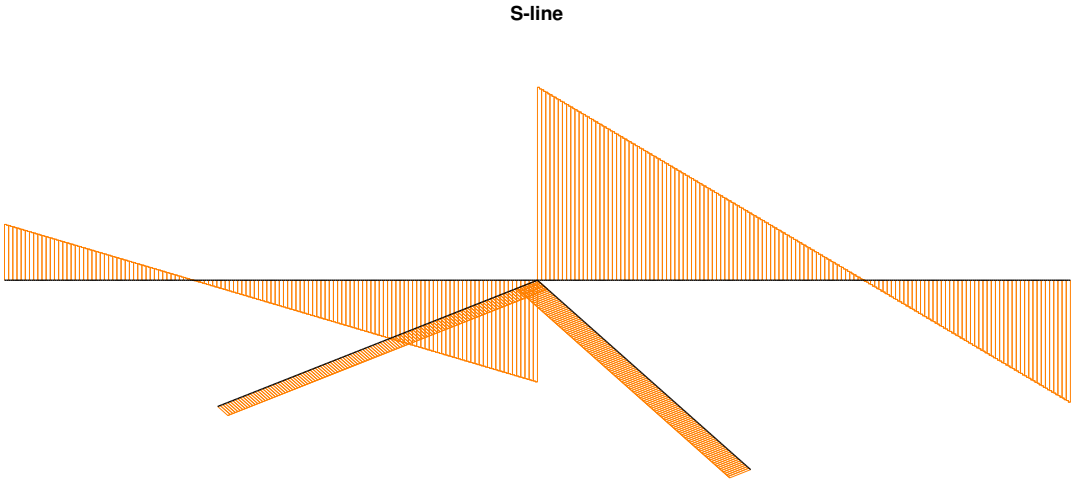

Figure 4.4: S-distribution according to FramesQuadratic2d.m

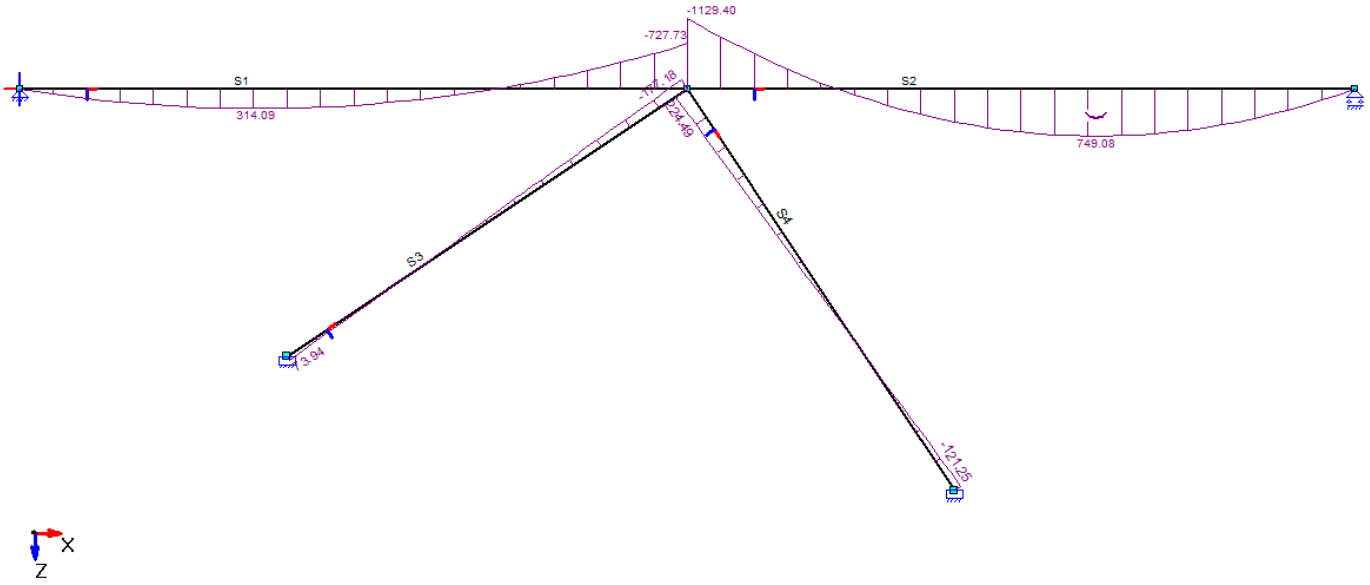

Figure 4.5: M-distribution according to Matrixframe

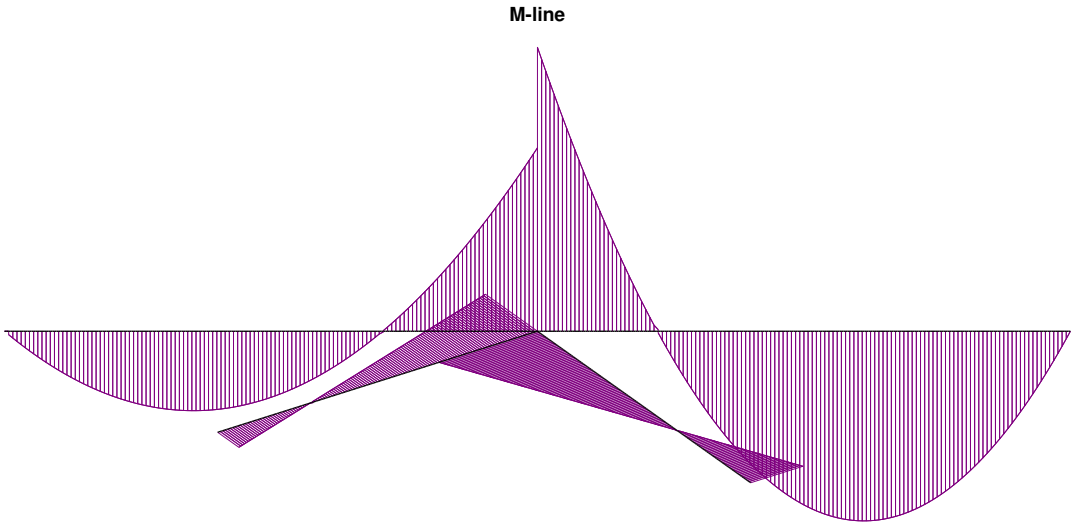

Figure 4.6: M-distribution according to FramesQuadratic2d.m

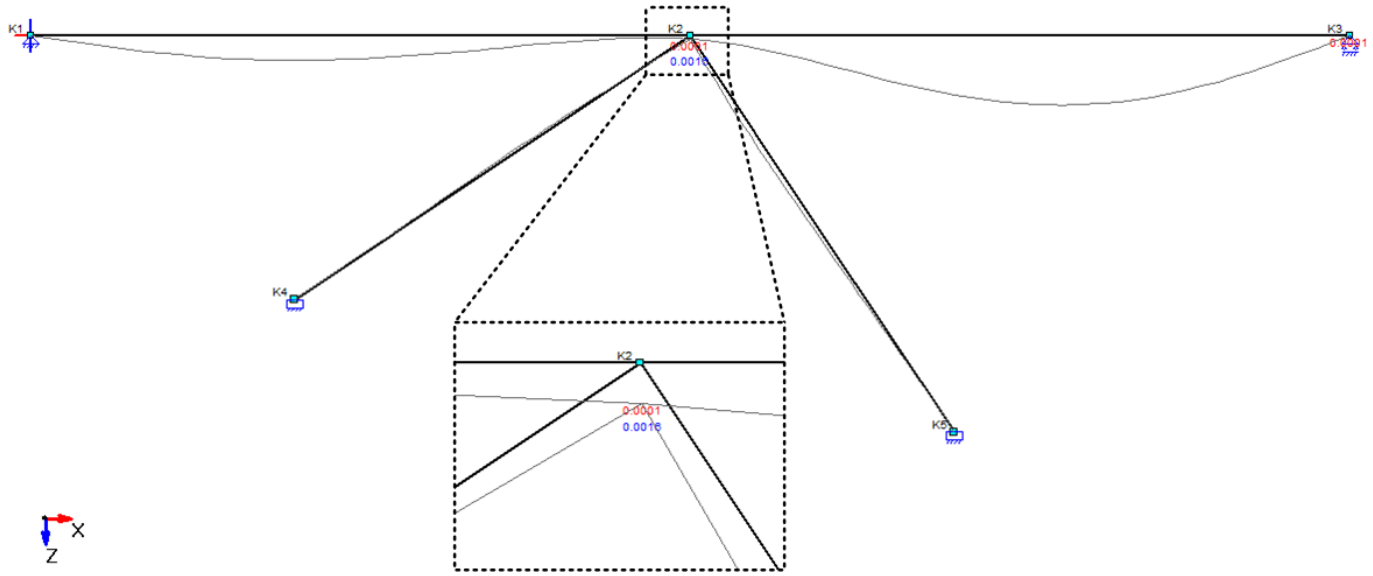

Figure 4.7: Deflection according to Matrixframe

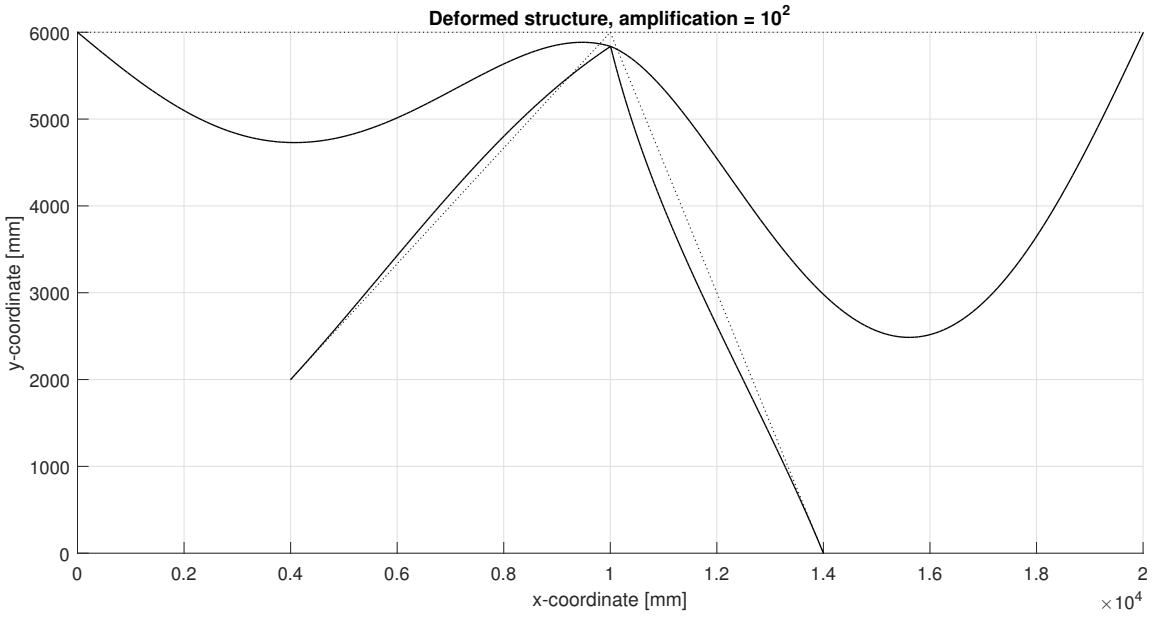

Figure 4.8: Deflection according to FramesQuadratic2d.m

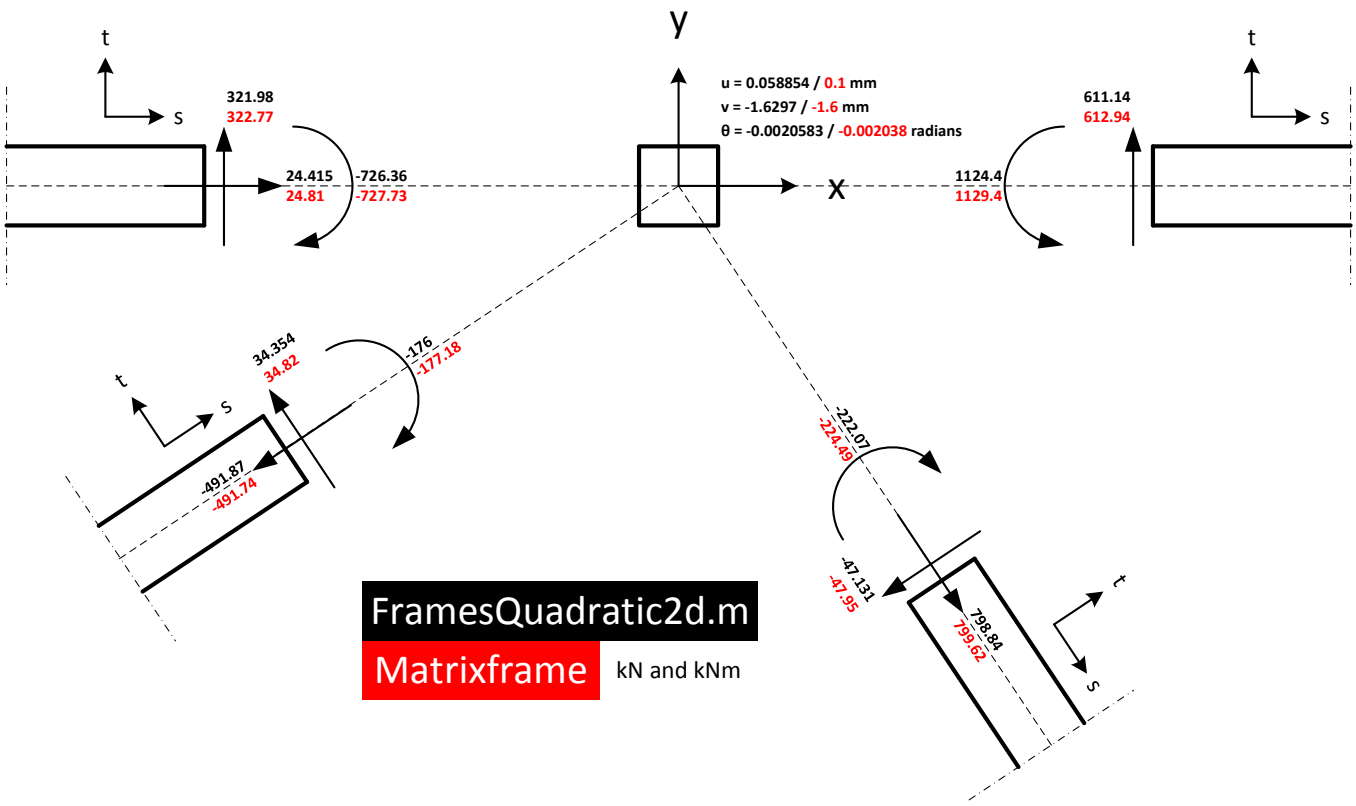

Figure 4.9: FramesQuadratic2d.m versus Matrixframe: mid node

4.2 Structure 2

The structure of figure 2.1 will be evaluated by the frame programs in order to investigate what happens to the internal forces if the members are able to transfer shear forces and bending moments as well. In other words, what happens to the internal force distribution if the members are modelled as beams instead of bars? The material and cross-sectional properties are the same with addition of the second moment of area: $I_{yy} = 4.12 \cdot 10^6$ mm^4.

Based on the output, it appears that a small part of the load will be transferred to the support as shear force and bending moment. This can also be solely observed from the normal forces as computed by Trusses2d.m and FramesQuadratic2d.m; showing values that are slightly different. See figure 4.10.

Linear beam elements do not present an accurate approximation as can be observed from the evaluation by Matrixframe. See column C of figure 4.10 and the right column as shown in figure 4.11. From the same figures it can be concluded that quadratic elements give an equally accurate result as Matrixframe.

	A	B	C	D
1	element	Trusses2d.m	FramesLinear2d.m	FramesQuadratic2d.m
2	1	28.284	11.291	27.589
3	2	9.86E-15	5.9623	0.22408
4	3	-20	-13.998	-19.754
5	4	-20	-14.009	-19.739
6	5	-10	-1.3937	-9.636
7	6	20	22.352	19.985
8	7	14.142	7.5766	13.842
9	8	-30	-27.451	-29.769
10	9	0	3.7081	0.19157
11	10	14.142	7.5766	13.842
12	11	-10	-1.3937	-9.636
13	12	20	22.352	19.985
14	13	-30	-27.451	-29.769
15	14	28.284	11.291	27.589
16	15	-20	-13.998	-19.754
17	16	-5.83E-15	-5.9623	-0.22408
18	17	-20	-14.009	-19.739

Figure 4.10: Normal forces (kN) determined by the three programs

Staaf	Nxmax
S1	28.28
S2	-0.00
S3	-20.00
S4	-20.00
S5	-10.00
S6	20.00
S7	14.14
S8	-30.00
S9	0.00
S10	14.14
S11	-10.00
S12	20.00
S13	-30.00
S14	28.28
S15	-20.00
S16	0.00
S17	-20.00

Staaf	Nxmax
S1	27.58
S2	0.23
S3	-19.75
S4	-19.74
S5	-9.63
S6	19.99
S7	13.84
S8	-29.77
S9	0.19
S10	13.84
S11	-9.63
S12	19.99
S13	-29.77
S14	27.58
S15	-19.75
S16	0.23
S17	-19.74

Figure 4.11: Normal forces (kN) determined by Matrixframe: truss and frame

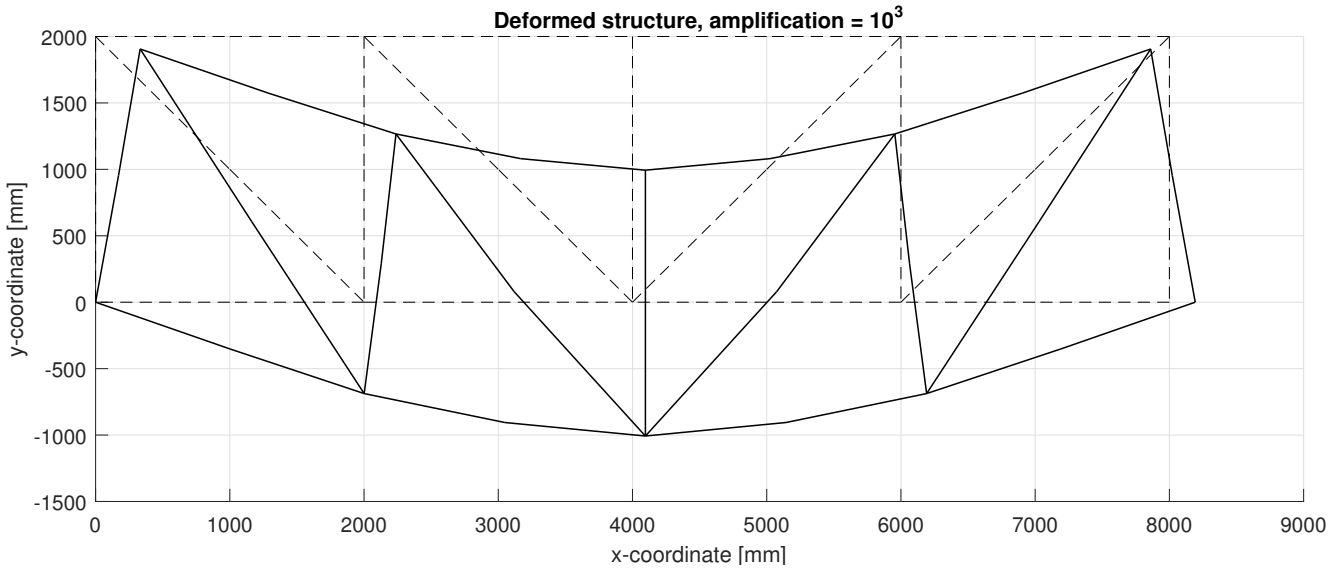

Figure 4.12: Deformation of the structure as frame, 17 quadratic elements

4.3 Structure 3

A portal structure with an alternating bending stiffness ratio between its members, results in varying mechanical behaviour. Loaded by a uniformly distributed load of 15 kN/m on the beam member, the structure as shown in figure 4.13 will be investigated. More specifically, the influence of EI_{yy} of the left column on the internal bending moment at B and E and the horizontal displacement at B will be investigated quantitatively. This is done by plotting the normalised values. Example 4 in paragraph 7.2 of [5] derives the obtained relation analytically. See equation (4.1) to (4.3).

Each member is modelled by 128 quadratic elements having an E-modulus of $2.1 \cdot 10^5$ N/mm^2, a cross-sectional area of $1.9754 \cdot 10^4$ mm^2 and a second moment of area of $8.6975 \cdot 10^8$ mm^4.

$$\frac{M_B}{M_{B,prismatic}} = \frac{5n}{4n+1} \tag{4.1}$$

$$\frac{M_E}{M_{E,prismatic}} = \frac{5(2n+1)}{3(4n+1)} \tag{4.2}$$

$$\frac{u_B}{ql^4/EI_{yy}} = \frac{n-1}{24(4n+1)} \tag{4.3}$$

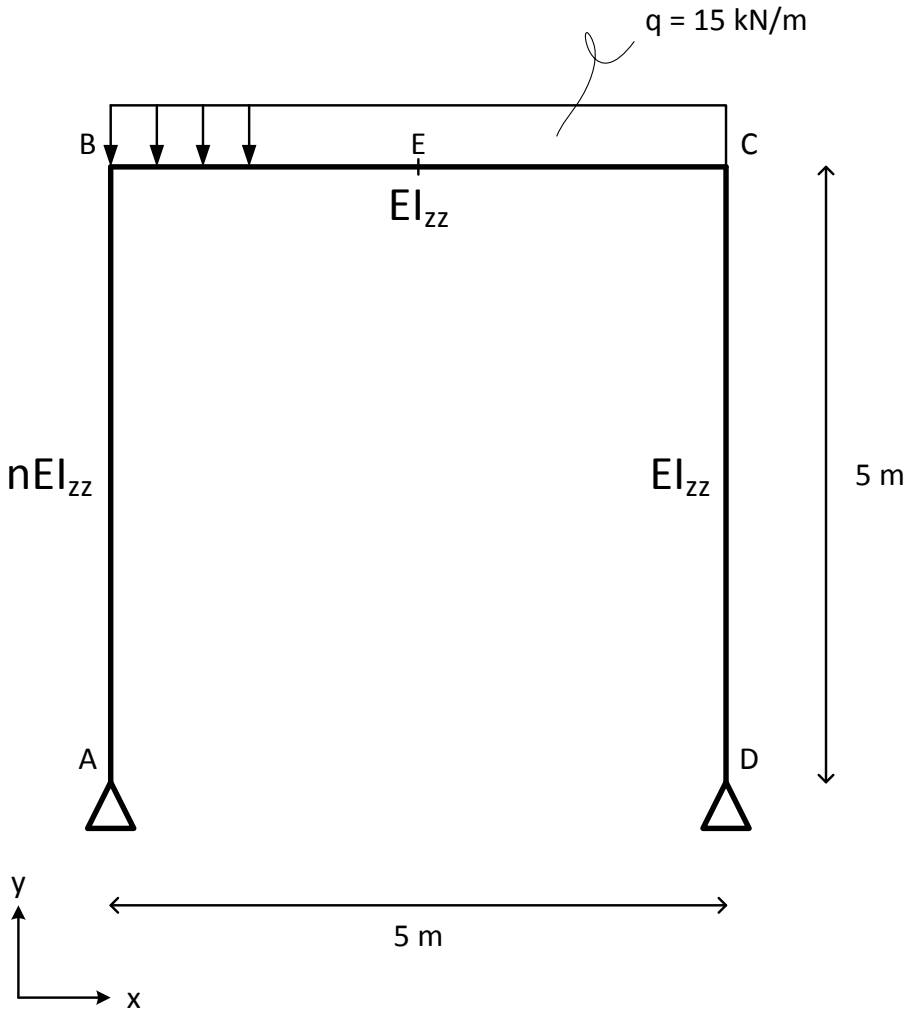

Figure 4.13: Portal structure

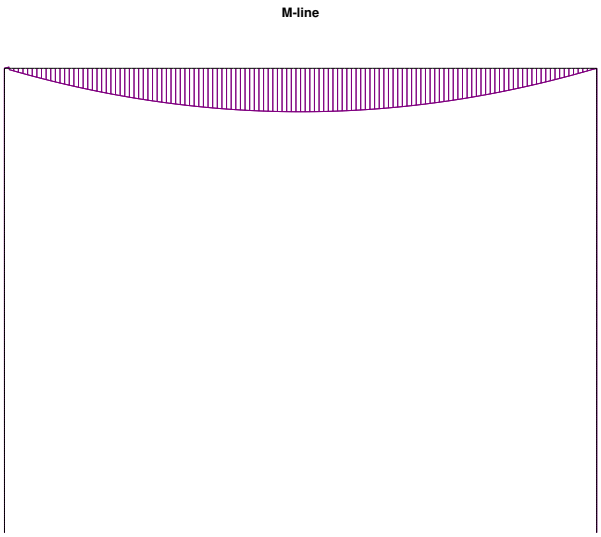

Figure 4.14: M-distribution if n = 0.001

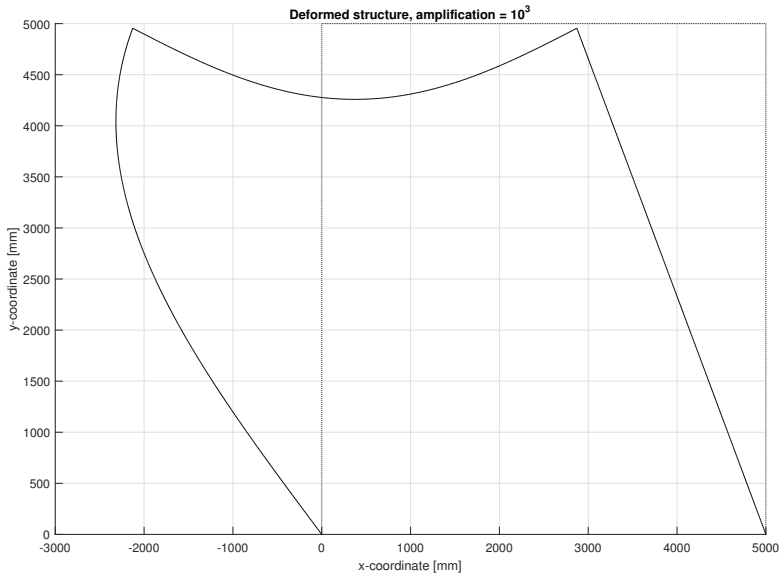

Figure 4.15: Deflection if n = 0.001

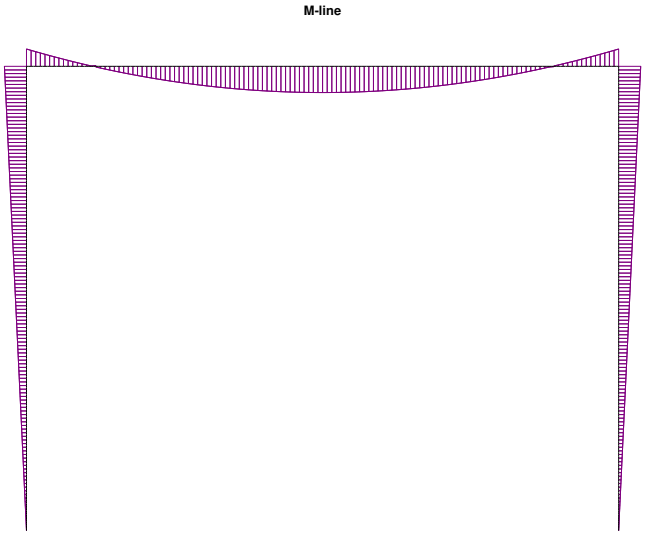

Figure 4.16: M-distribution if n = 1

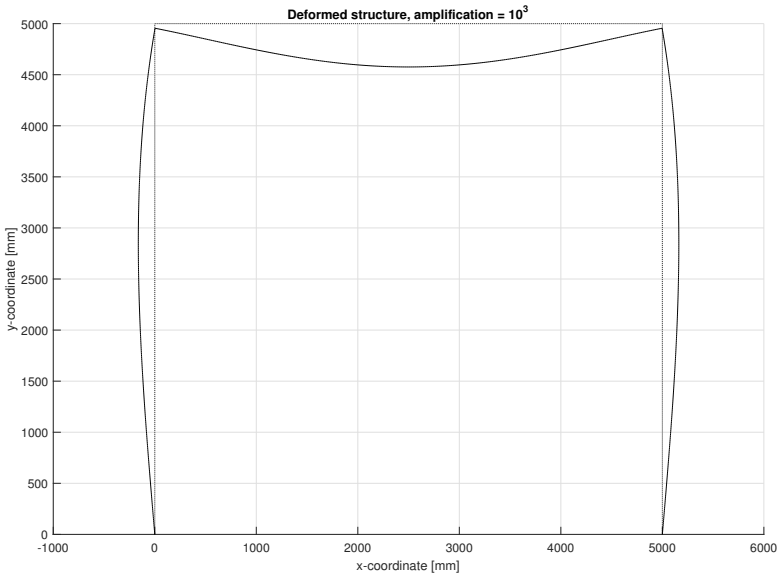

Figure 4.17: Deflection if n = 1

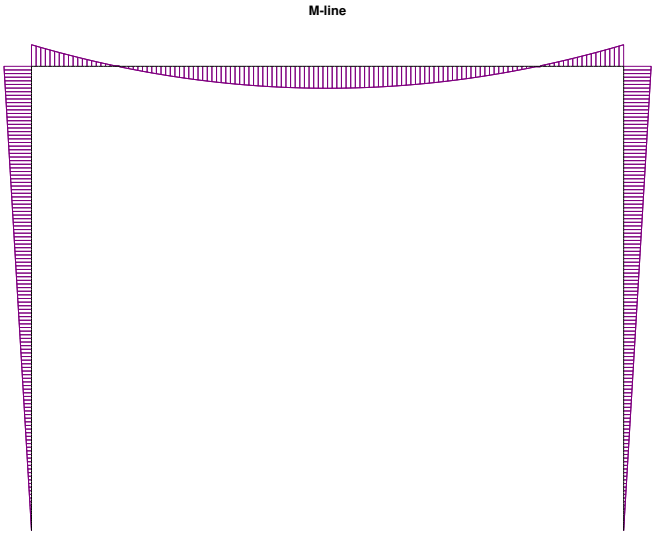

Figure 4.18: M-distribution if n = 1000

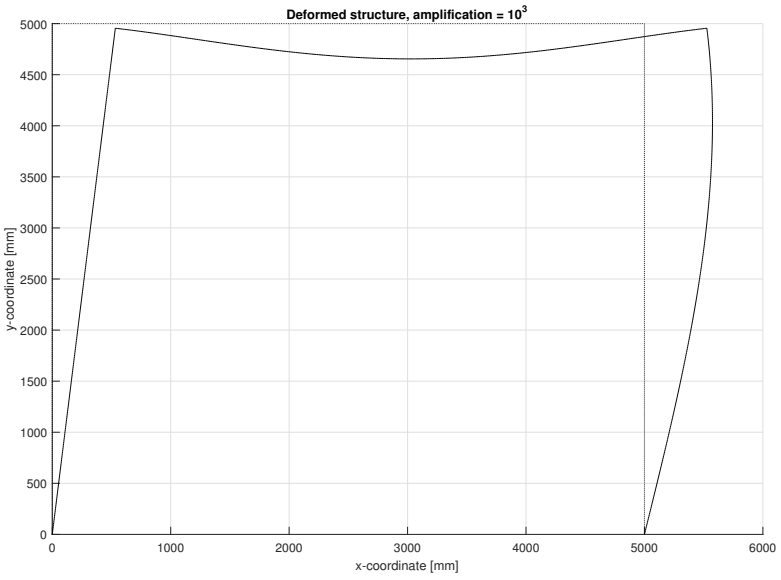

Figure 4.19: Deflection if n = 1000

The left column having a bending stiffness such that the limit of n goes to 0, results in the structure having an M-distribution which is similar to that of the simply supported beam as seen in chapter 3; a parabolic distribution with a minimum of null at the corners B and C and a maximum at midspan. Increasing the value for n leads to an increase of the corner moments and a decrease of the midspan moment which corresponds to an increase of the M-distribution along both columns, because the connections between the members are considered stiff. A graphical presentation of this observation is given in figure 4.20. As for the distribution of the shear force, this will remain unchanged for the beam member, but will increase in both columns with an increasing value for n. This is because the gradient of the M-line is not dependent on the parameter n, only in case of the horizontal member.

For the horizontal displacement of B a similar graph can be made as shown in figure 4.21. Depending on which column has a relative higher bending stiffness, the structure deflects either to the left (n $<$ 1) or to the right (n $>$ 1), considering the columns respectively as roller bearings. Why the deflection line is not symmetric as the bending moment distribution is because the curvature 'kappa' is linearly related to M by EI_{yy}.

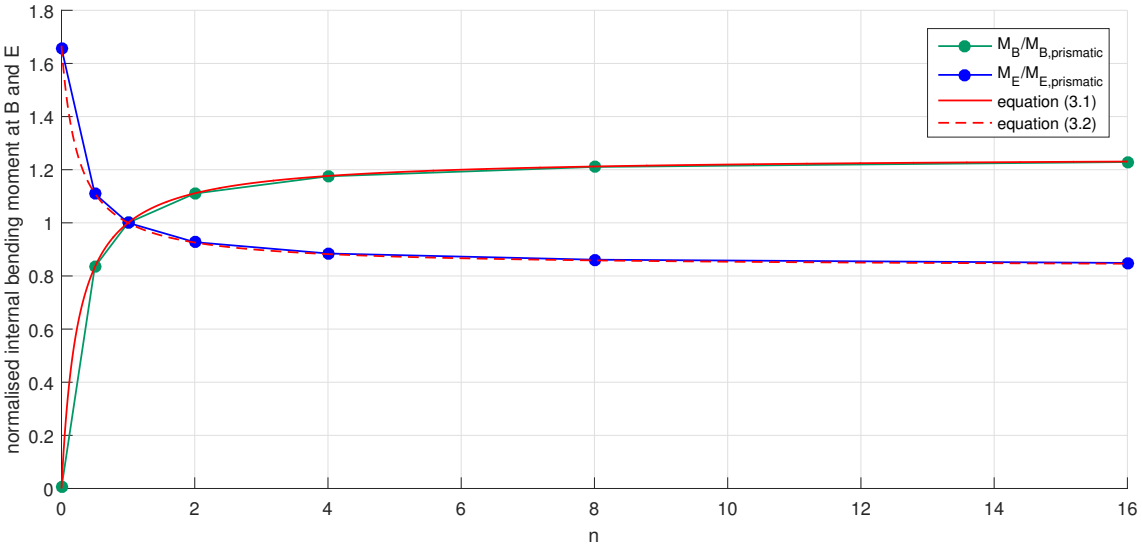

Figure 4.20: Normalised M_B and M_E as function of n

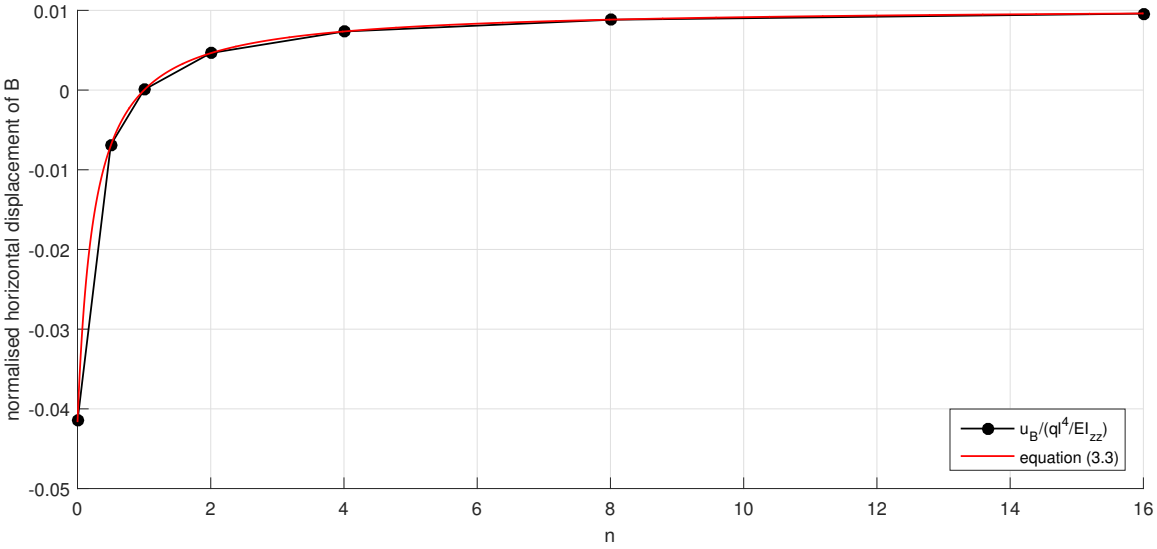

Figure 4.21: Normalised u_B as function of n

4.4 Structure 4

The structure as presented in figure 4.22 is used to demonstrate how a uniformly distributed load can be rotated such that it is exerted perpendicular on a structural member with any arbitrary orientation. Modelling a wind load is a perfect example for the necessity of this requirement to the program; idealised as a uniformly distributed load exerted on one leg as a pressure and on the other as a tensile stress, depending from which side the wind blows. See figure 4.26. But first the nodal quantities are evaluated in case the roof structure is loaded by snow which is idealised as a q-load exerted along the longitudinal axis of the member and orientated parallel to the global vertical axis.

Each leg is modelled by four quadratic elements having an E-modulus of $2.1 \cdot 10^5\,\mathrm{N/mm^2}$, a cross-sectional area of $1.9754 \cdot 10^4\,\mathrm{mm^2}$ and a second moment of area of $8.6975 \cdot 10^8\,\mathrm{mm^4}$.

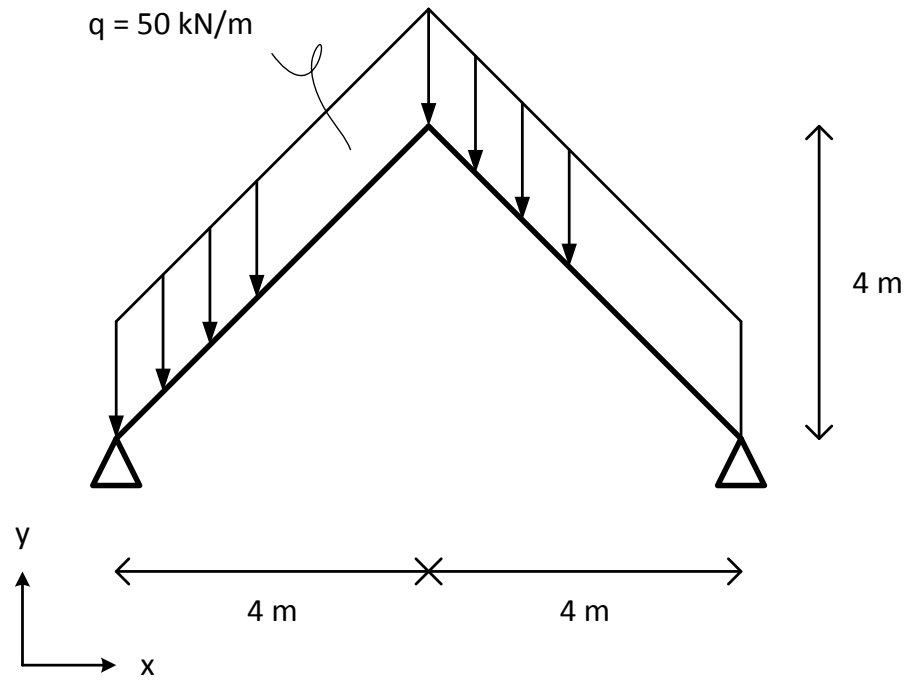

Figure 4.22: Simple roof structure loaded by snow

Note that the q-load due to snow is ten times greater than what is considered realistic in Dutch engineering practice; assuming a specific mass of 200 kg/m3 and a strip of 2.5 meter width. This is done to calculate distinct displacement values for the nodes.

A discrepancy can be observed between the nodal displacements computed by FramesQuadratic2d.m (figure 4.24) and Matrixframe (figure 4.25). This is because of the numerical accuracy of the latter program; values are rounded off to a tenth of a millimetre. The difference is of the order of one percent. As for the discretized q-load, again, a distribution ratio per quadratic element of 1/6:2/3:1/6 can be observed. See sub-paragraph 2.2.2.

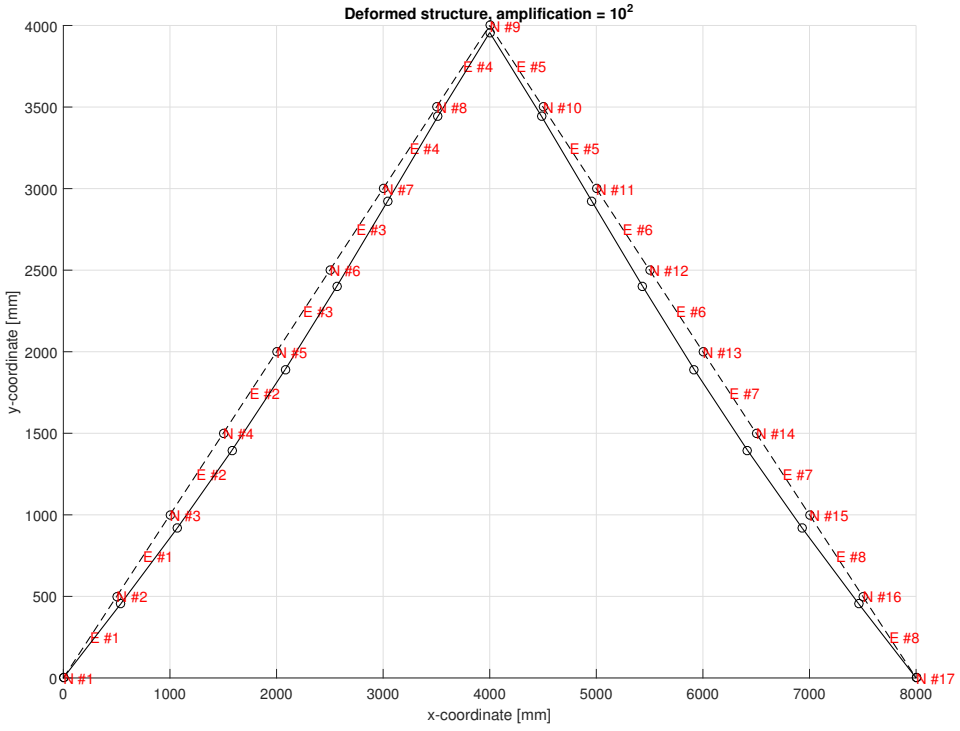

Figure 4.23: Deformed roof structure due to an amplified snow load

ans =

node	u	v	theta	F_x	F_y	T_z	q_x	q_y	R_x	R_y	M_z_ext
1	0	0	-0.00083333	0	0	0	0	-11785	1.7511e+05	2.8284e+05	0
2	0.38172	-0.45677	-0.00074046	0	0	0	0	-47140	0	0	0
3	0.67716	-0.82122	-0.00050749	0	0	0	0	-23570	0	0	0
4	0.82767	-1.0347	-0.00020286	0	0	0	0	-47140	0	0	0
5	0.82024	-1.0843	0.00010501	0	0	0	0	-23570	0	0	0
6	0.66465	-0.97962	0.00034766	0	0	0	0	-47140	0	0	0
7	0.41632	-0.77621	0.00045667	0	0	0	0	-23570	0	0	0
8	0.15349	-0.55226	0.0003636	0	0	0	0	-47140	0	0	0
9	8.0832e-15	-0.43163	1.5503e-17	0	0	0	0	-23570	0	0	0
10	-0.15349	-0.55226	-0.0003636	0	0	0	0	-47140	0	0	0
11	-0.41632	-0.77621	-0.00045667	0	0	0	0	-23570	0	0	0
12	-0.66465	-0.97962	-0.00034766	0	0	0	0	-47140	0	0	0
13	-0.82024	-1.0843	-0.00010501	0	0	0	0	-23570	0	0	0
14	-0.82767	-1.0347	0.00020286	0	0	0	0	-47140	0	0	0
15	-0.67716	-0.82122	0.00050749	0	0	0	0	-23570	0	0	0
16	-0.38172	-0.45677	0.00074046	0	0	0	0	-47140	0	0	0
17	0	0	0.00083333	0	0	0	0	-11785	-1.7511e+05	2.8284e+05	0

Figure 4.24: Nodal quantities, FramesQuadratic2d.m, first load case

Knoop	X	Z
K1	-0.0000	0.0000
K2	0.0004	0.0004
K3	0.0006	0.0008
K4	0.0008	0.0010
K5	0.0007	0.0010
K6	0.0006	0.0009
K7	0.0004	0.0007
K8	0.0001	0.0005
K9	0.0000	0.0004
K10	-0.0001	0.0005
K11	-0.0004	0.0007
K12	-0.0006	0.0009
K13	-0.0007	0.0010
K14	-0.0008	0.0010
K15	-0.0006	0.0008
K16	-0.0004	0.0004
K17	0.0000	0.0000

Figure 4.25: Nodal displacements (m), Matrixframe, first load case

In the next load case wind is presented as a static uniformly distributed load that exerts pressure at the left leg and tensile stress at the right leg, perpendicular to each member. Proper modelling of this load starts with the descriptive file. One option is to calculate both the x- and y-component manually and insert these values in the columns F and G of the elements worksheet respectively. However, this is rather tedious and a better option is to make use of the lambda matrix.

The local equivalence of the global force vector $\left\{F^{\circ}\right\}$ and the global displacement vector $\left\{U^{\circ}\right\}$ is obtained by pre-multiplication with the lambda matrix. This applies to the global distributed force vector $\left\{q^{\circ}\right\}$ as well. If the values in columns F and G of the elements worksheet are considered as components in the s- and t-direction instead of the x- and y-direction, one can obtain their global equivalence by pre-multiplication with the inverse, i.e. the transpose, of the lambda matrix. Compare (2.23) with (4.4). This is an efficient way to deal with q-loads that are exerted perpendicular to the member. However, it is paramount to know how the local coordinate system is orientated for each element, i.e. from which node to which node the element is defined, in order to make sure whether a pressure or a tensile stress is applied.

$$\left\{q^{\circ}\right\} = \sum_{e=1}^{E} \left[\left[\lambda\right]^{T} \int_{L} \left[N^{(e)}\right]^{T} \begin{Bmatrix} q_s \\ q_t \\ 0 \end{Bmatrix} ds \right] = \sum_{e=1}^{E} \left[\left[\lambda\right]^{T} \left\{q^{(e)}\right\} \right] \tag{4.4}$$

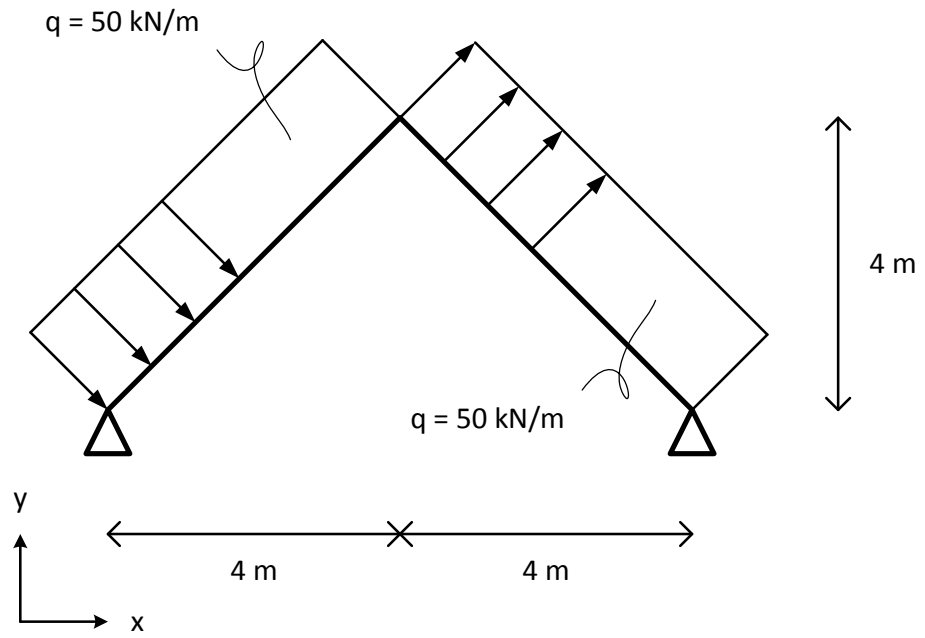

Figure 4.26: Simple roof structure loaded by wind

From the eight and ninth column of the table as presented in figure 4.28, it can be checked that the discretized distributed force has the correct orientation with respect to the global coordinate system and that in this direction as well a distribution ratio per quadratic element of 1/6:2/3:1/6 is applied.

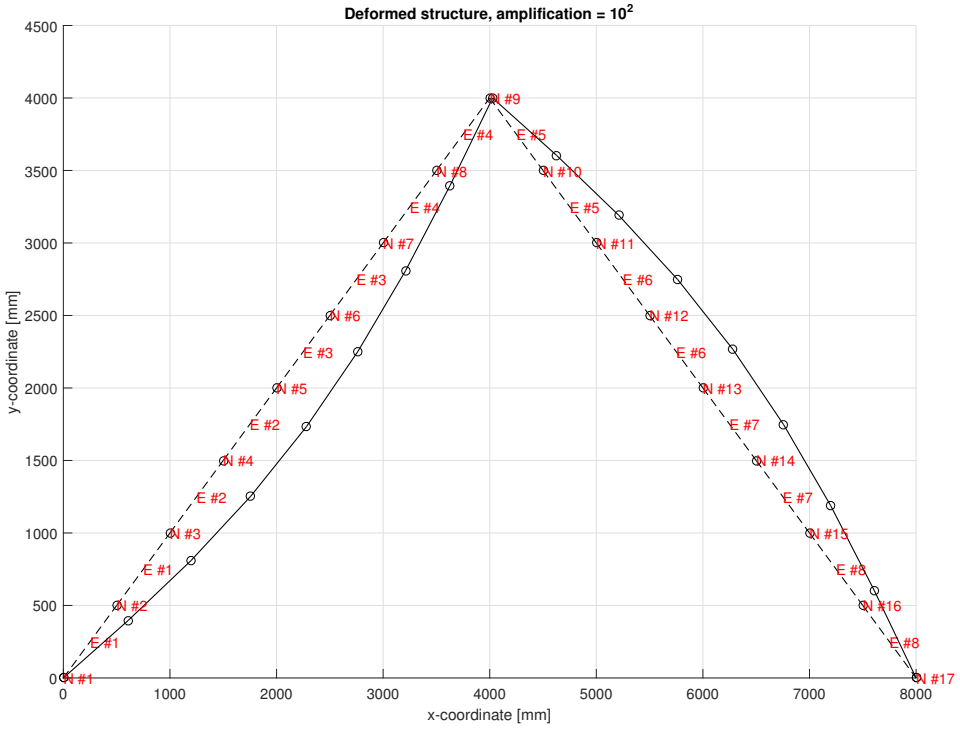

Figure 4.27: Deformed roof structure due to wind

ans =

node	u	v	theta	F_x	F_y	T_z	q_x	q_y	R_x	R_y	M_z_ext
1	0	0	-0.0020989	0	0	0	8333.3	-8333.3	-2e+05	-6.4319e-09	0
2	1.073	-1.0389	-0.0019214	0	0	0	33333	-33333	0	0	0
3	1.9736	-1.9054	-0.0014536	0	0	0	16667	-16667	0	0	0
4	2.5728	-2.4705	-0.00079225	0	0	0	33333	-33333	0	0	0
5	2.806	-2.6696	-3.4091e-05	0	0	0	16667	-16667	0	0	0
6	2.6409	-2.4705	0.00072406	0	0	0	33333	-33333	0	0	0
7	2.11	-1.9054	0.0013854	0	0	0	16667	-16667	0	0	0
8	1.2775	-1.0389	0.0018532	0	0	0	33333	-33333	0	0	0
9	0.27273	7.6199e-15	0.0020307	0	0	0	16667	7.7307e-12	0	0	0
10	1.2775	1.0389	0.0018532	0	0	0	33333	33333	0	0	0
11	2.11	1.9054	0.0013854	0	0	0	16667	16667	0	0	0
12	2.6409	2.4705	0.00072406	0	0	0	33333	33333	0	0	0
13	2.806	2.6696	-3.4091e-05	0	0	0	16667	16667	0	0	0
14	2.5728	2.4705	-0.00079225	0	0	0	33333	33333	0	0	0
15	1.9736	1.9054	-0.0014536	0	0	0	16667	16667	0	0	0
16	1.073	1.0389	-0.0019214	0	0	0	33333	33333	0	0	0
17	0	0	-0.0020989	0	0	0	8333.3	8333.3	-2e+05	9.6989e-09	0

Figure 4.28: Nodal quantities, FramesQuadratic2d.m, second load case

Knoop	X	Z
K1	0.0000	0.0000
K2	0.0010	0.0010
K3	0.0019	0.0018
K4	0.0025	0.0024
K5	0.0027	0.0026
K6	0.0026	0.0024
K7	0.0020	0.0018
K8	0.0012	0.0010
K9	0.0003	0.0000
K10	0.0012	-0.0010
K11	0.0020	-0.0018
K12	0.0026	-0.0024
K13	0.0027	-0.0026
K14	0.0025	-0.0024
K15	0.0019	-0.0018
K16	0.0010	-0.0010
K17	0.0000	0.0000

Figure 4.29: Nodal displacements (m), Matrixframe, second load case

4.5 Structure 5

Up until now, only frames that are both statically and kinematic determinate were evaluated, as well as frame structures that are statically indeterminate. But how does the program respond when a 2D configuration of beams and columns is not sufficiently constrained? That is, how does the program deal with kinematic indeterminate frames? The frame structure presented in figure 4.30 is used to demonstrate this.

Each member is modelled by two quadratic elements having an E-modulus of $2.1 \cdot 10^5$ N/mm^2, a cross-sectional area of $1.9754 \cdot 10^4$ mm^2 and a second moment of area of $8.6975 \cdot 10^8$ mm^4.

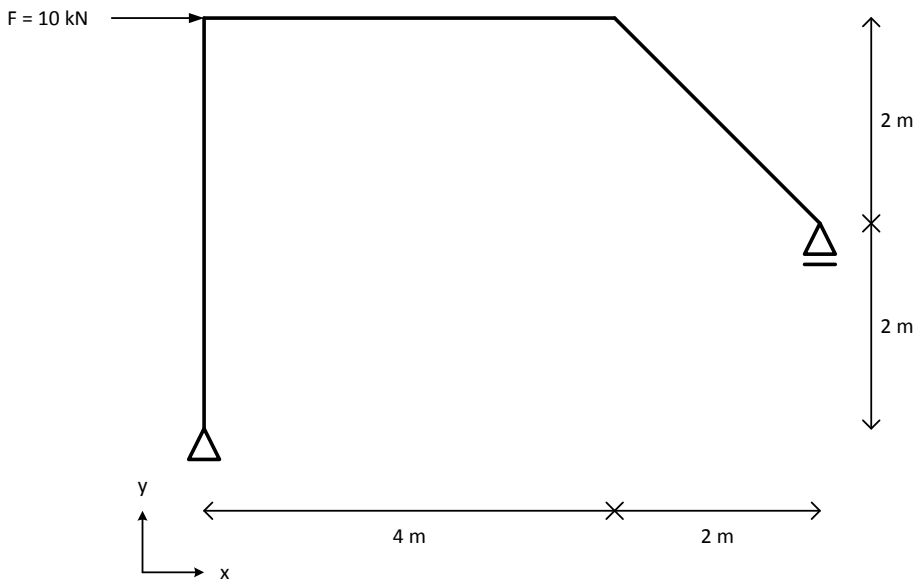

Figure 4.30: Kinematic determinate frame structure

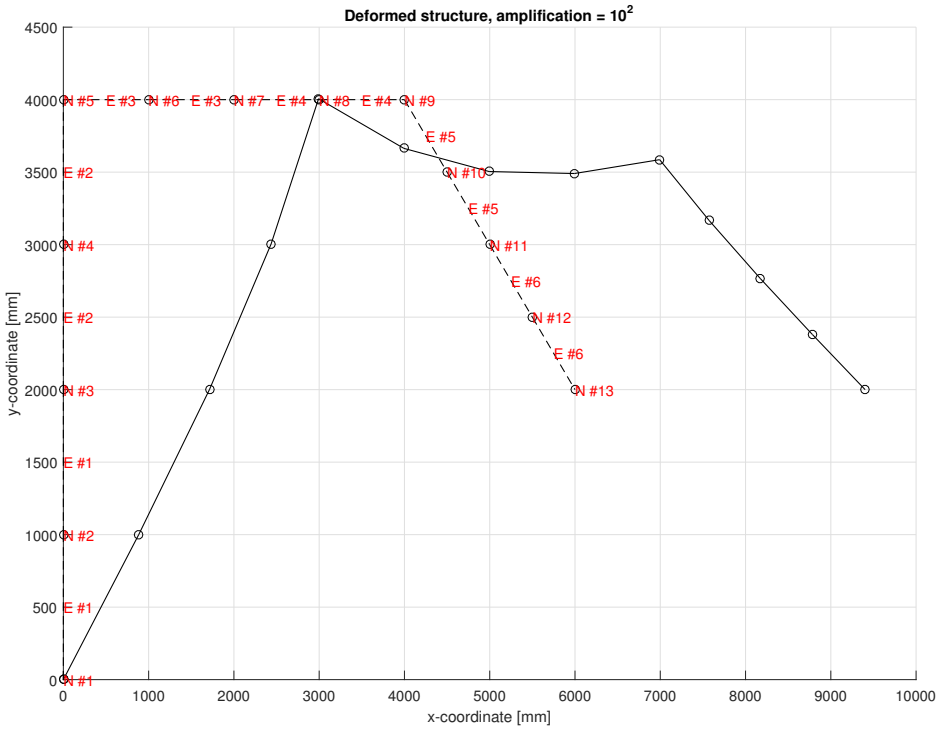

Figure 4.31: Deformed kinematic determinate frame structure

If the left hinge is replaced by a roller bearing, the structure will become kinematic indeterminate and Matlab will give the following warning.

```
>> FramesQuadratic2d
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  1.177473e-22.
> In FramesQuadratic2d (line 133)
```

In order to compute the unknown nodal displacements $\{U_1\}$ in the fourth section of the program, the relevant part of the stiffness matrix $[k_{11}]$ needs to be inverted. However, it appears that when the frame is kinematic indeterminate its corresponding relevant part of the stiffness matrix will be singular or almost singular ('badly scaled'), i.e. the matrix cannot be inverted at all or with enough accuracy depending on how close the determinant is to null. This is quantitatively described by the reciprocal condition number 'RCOND'; the closer this value is to 1.0, the higher the accuracy of the solution. RCOND of $[k_{11}]$ is $1.1775 \cdot 10^{-22}$ and $1.1335 \cdot 10^{-10}$ for the kinematic indeterminate and determinate frame structure respectively. The displacement is shown in figure 4.32. Note the values along the x-axis.

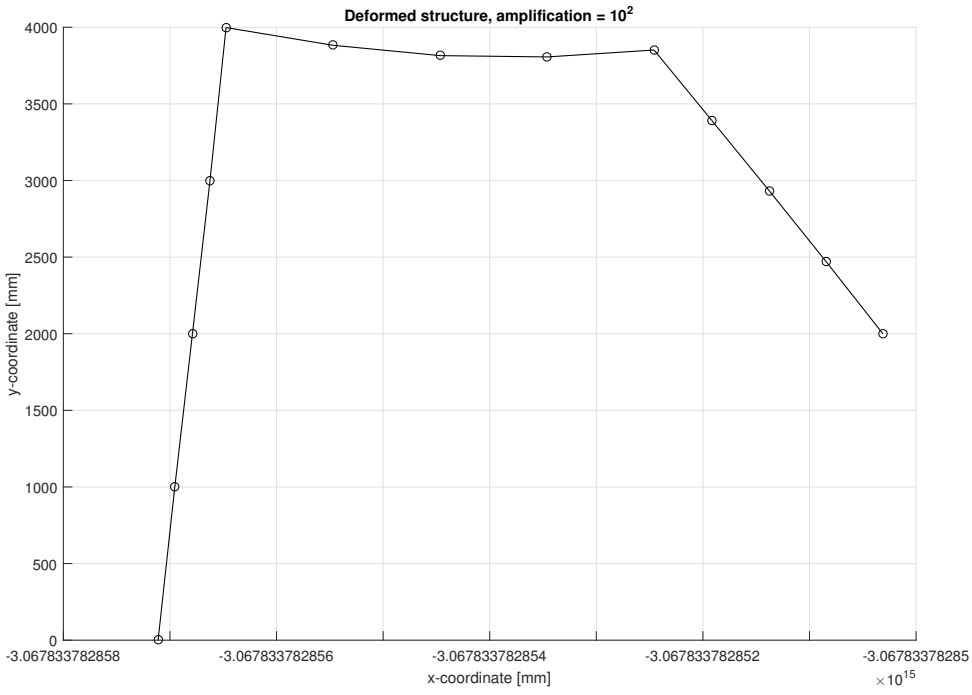

Figure 4.32: Deformed and displaced kinematic indeterminate frame structure

Chapter 5

An arch structure

Arches were and are still used as architectural and structural elements in buildings and other civil engineering applications. That's why it is important to determine the mechanical behaviour under a certain load for these kind of structures as well. In this chapter a semicircle is modelled by beam elements in 2D space and evaluated by different mesh sizes. Depending on the application, this conceptual model will suffice more or less; it is clear that a structure like the Gateway Arch[2] -located in the American city of St. Louis, Missouri- is more susceptible to this model than a barrel vault, let alone a dome structure. The latter two require 2D or even 3D elements to come to a more reliable model.

Figure 5.1 presents the structure loaded by a uniformly distributed load of 50 kN/m in the negative y-direction. Each element has an E-modulus of $2.1 \cdot 10^5\,\mathrm{N/mm^2}$, a cross-sectional area of $1.9754 \cdot 10^4\,\mathrm{mm^2}$ and a second moment of area of $8.6975 \cdot 10^8\,\mathrm{mm^4}$. The way each successive mesh size is created, can be seen in figure 5.2; going from 2 to 1024 elements, the corresponding nodes are divided equally over the semicircle. With help from the function as presented in listing 5.1, the coordinates of these nodes can be found efficiently.

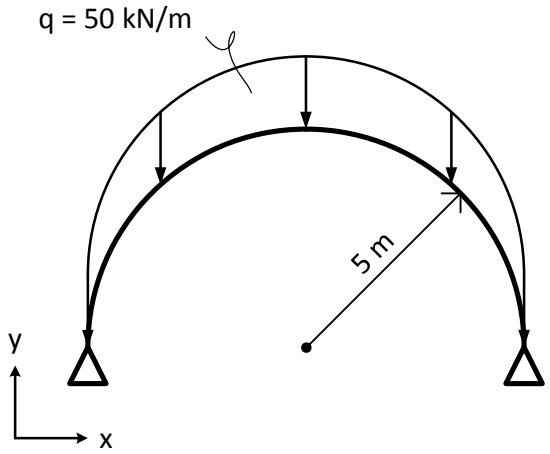

Figure 5.1: Semicircle arch structure

[2]Eero Saarinen, 1963-1965

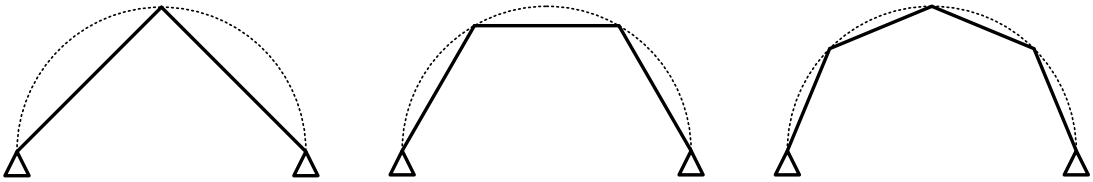

Figure 5.2: Geometry convergence

Listing 5.1: Function to find node coordinates

```
function [coordinates] = NodesSemicircle(numelem,radius,order)

angle = zeros(numelem+1,1);
for n = 1:numelem
    angle(n+1) = pi*n/numelem;
end

coordinates = zeros(numelem+1,2);
for n = 1:numelem+1
    coordinates(n,1) = radius*cos(angle(n))+radius;
    coordinates(n,2) = radius*sin(angle(n));
end

if order == 2
    matrix = zeros(2*numelem+1,2);
    matrix(1:2:end,:) = coordinates(:,:);
    for n = 1:numelem
        matrix(2*n,1) = (matrix(2*n-1,1)+matrix(2*n+1,1))/2;
        matrix(2*n,2) = (matrix(2*n-1,2)+matrix(2*n+1,2))/2;
    end
    coordinates = matrix;
end

coordinates = flipud(coordinates);
```

The focus of this chapter lies on the convergence behaviour of five quantities, i.e. the deflection of the top, the rotation at the left support, the horizontal reaction force at the left support, the vertical reaction force at the left support and the internal bending moment at the top. Figure 5.6 presents the obtained values. By plotting the corresponding normalised errors in logspace, one obtains a clear graphical presentation of the convergence behaviour as done in chapter 3. These plots are shown in figure 5.7 to 5.11. The slope values are tabulated in table 5.1.

But first, have a look at figure 5.3 to 5.5 which depicts the deflection line, the S-line and the M-line. As expected, the arch dents in at the top and flexes outwards at the left and right. This corresponds to a decrease respectively an increase of the curvature as can be checked by observing the moment distribution. After all, M_z is linearly related to κ by EI_{yy}.

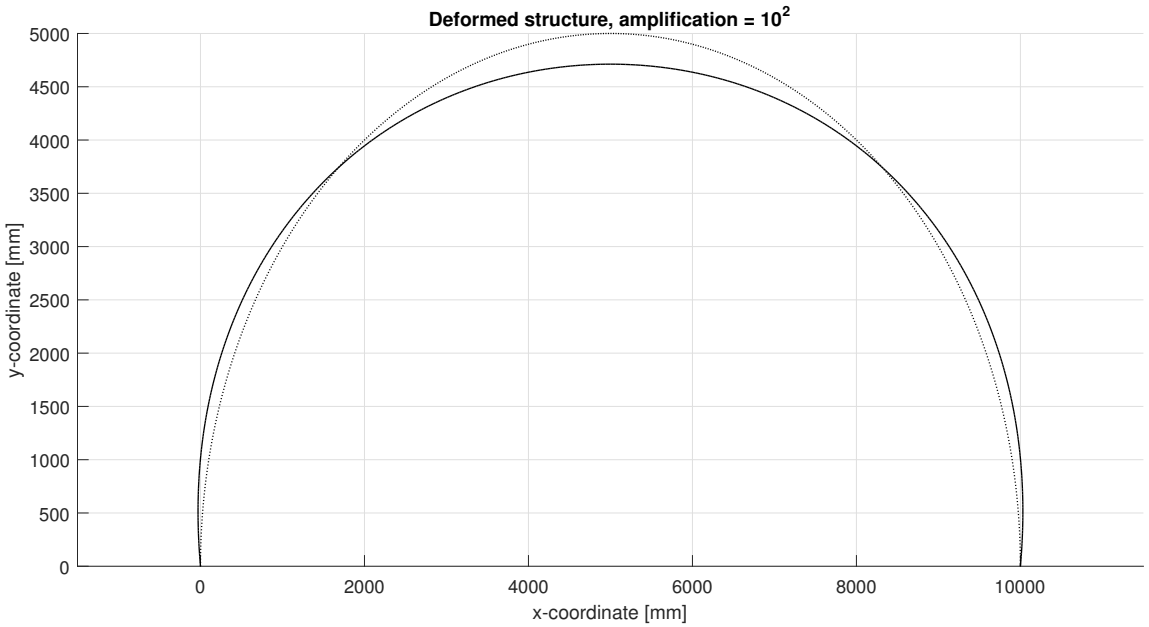

Figure 5.3: Deflection, 512 linear elements

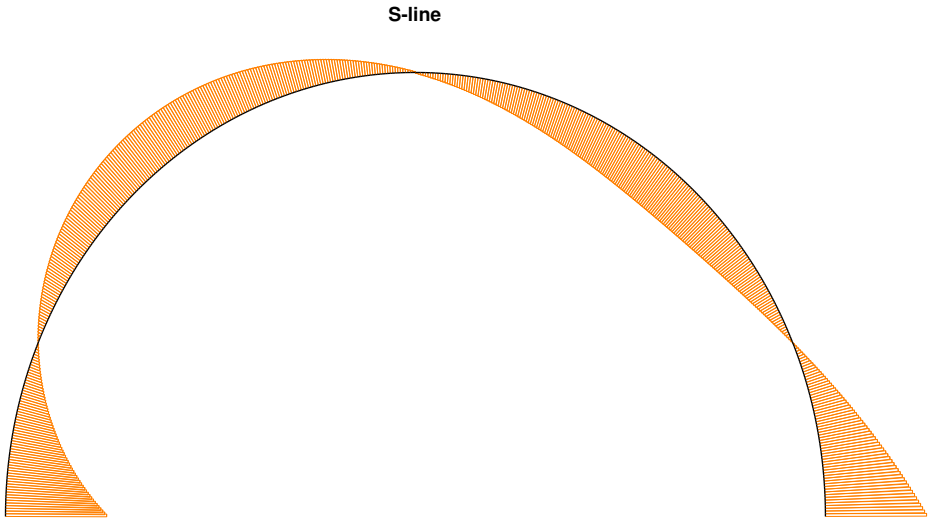

Figure 5.4: S-distribution, 512 linear elements

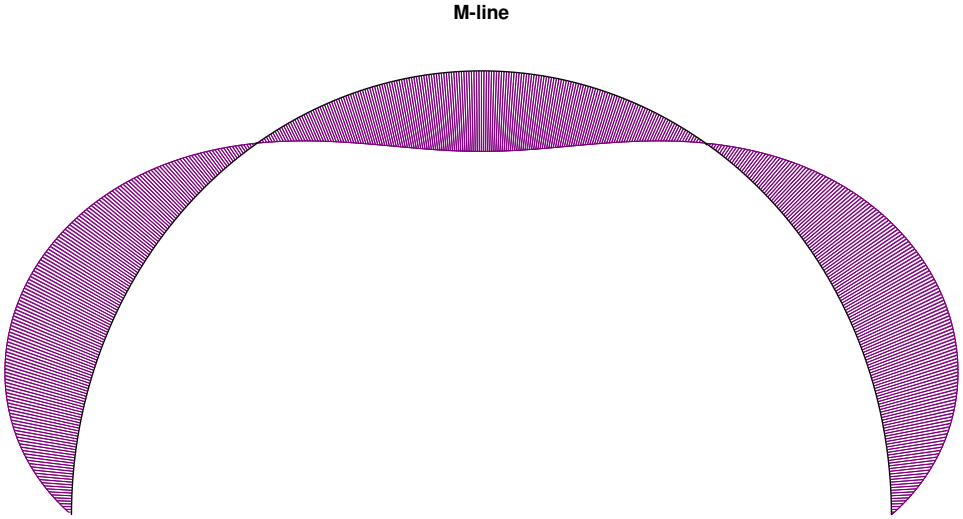

Figure 5.5: M-distribution, 512 linear elements

	A	B	C	D	E	F	G
1		deflection at top (mm)		rotation at left support (radians)		computation time (s)	
2	mesh size	linear	quadratic	linear	quadratic	linear	quadratic
3	2	-0.60055	-0.6757	-0.00011902	-0.0015556	1.331	1.42
4	4	-0.93897	-1.6717	0.00028605	0.00018843	1.414	1.547
5	8	-2.2759	-2.5163	0.00083035	0.00082274	1.578	1.78
6	16	-2.7212	-2.7851	0.00099719	0.00099634	1.832	2.272
7	32	-2.84	-2.8562	0.0010409	0.0010407	2.414	3.198
8	64	-2.8702	-2.8743	0.0010519	0.0010519	3.428	5.009
9	128	-2.8778	-2.8788	0.0010547	0.0010547	5.55	8.71
10	256	-2.8797	-2.8799	0.0010554	0.0010554	9.797	16.362
11	512	-2.8801	-2.8802	0.0010556	0.0010556	18.389	31.522
12	1024	-2.8803	-2.8803	0.0010556	0.0010556	38.109	66.046
13		R_x at left support (kN)		R_y at left support (kN)		M_{z,int} at top (kNm)	
14	mesh size	linear	quadratic	linear	quadratic	linear	quadratic
15	2	175.55	219.63	353.55	353.55	6.1487	-214.26
16	4	134.74	147.18	382.68	382.68	84.844	22.657
17	8	126.99	130.16	390.18	390.18	89.995	74.11
18	16	125.16	125.96	392.07	392.07	90.557	86.566
19	32	124.71	124.91	392.54	392.54	90.654	89.655
20	64	124.6	124.65	392.66	392.66	90.676	90.426
21	128	124.57	124.58	392.69	392.69	90.681	90.619
22	256	124.56	124.57	392.7	392.7	90.683	90.667
23	512	124.56	124.56	392.7	392.7	90.683	90.679
24	1024	124.56	124.56	392.7	392.7	90.683	90.682

Figure 5.6: Convergence data and computation durations

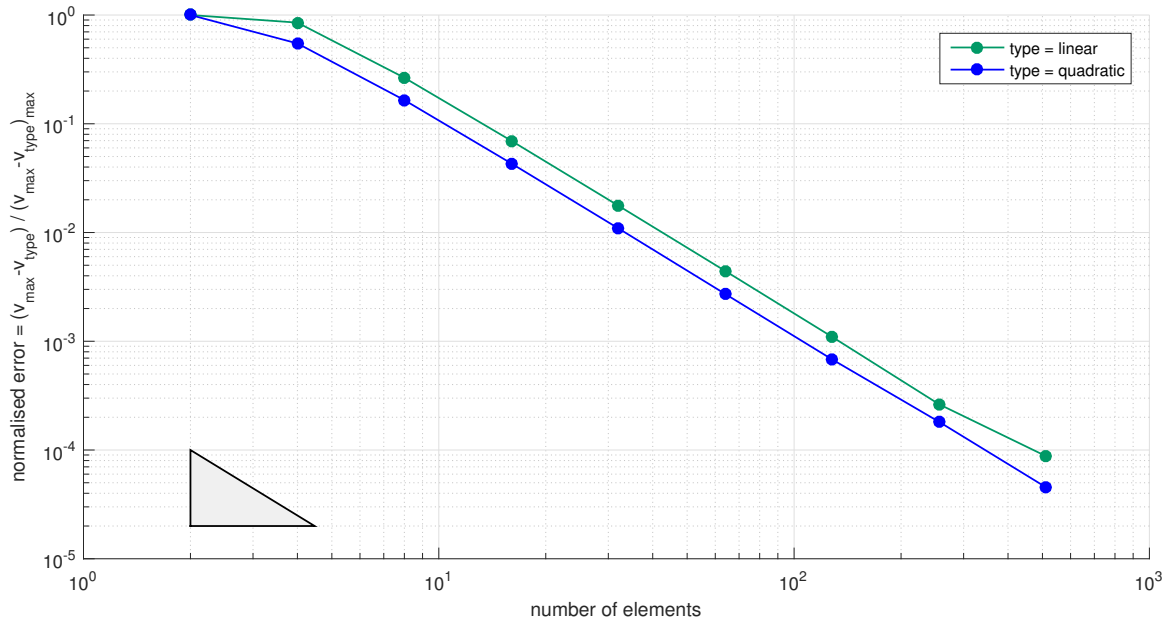

Figure 5.7: Error convergence of deflection at the top

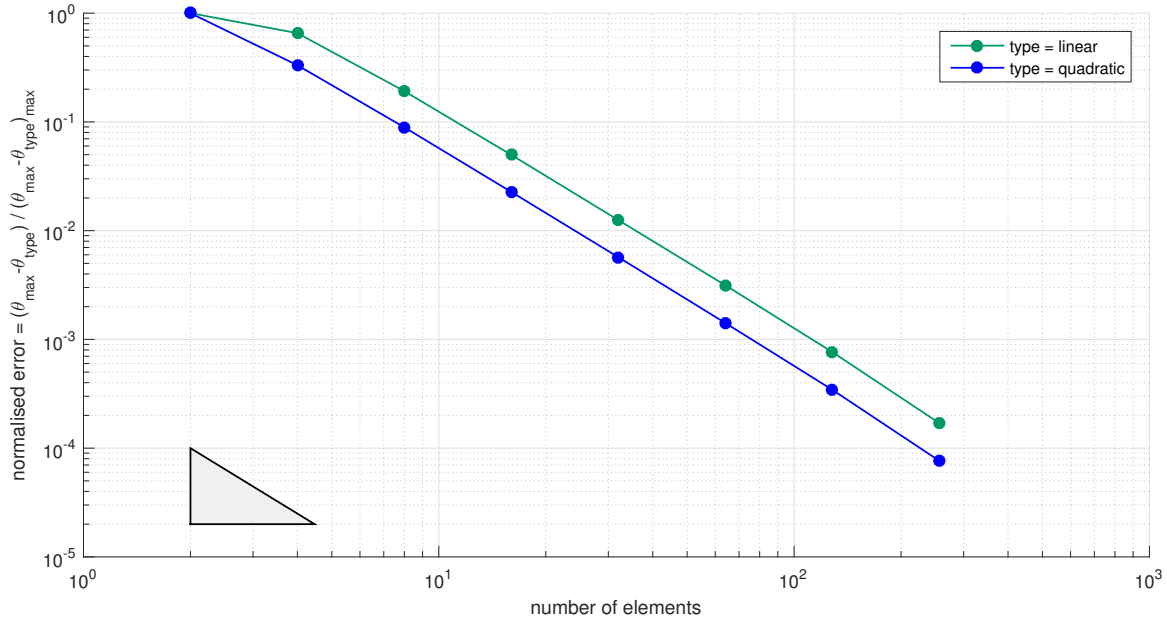

Figure 5.8: Error convergence of rotation at the left support

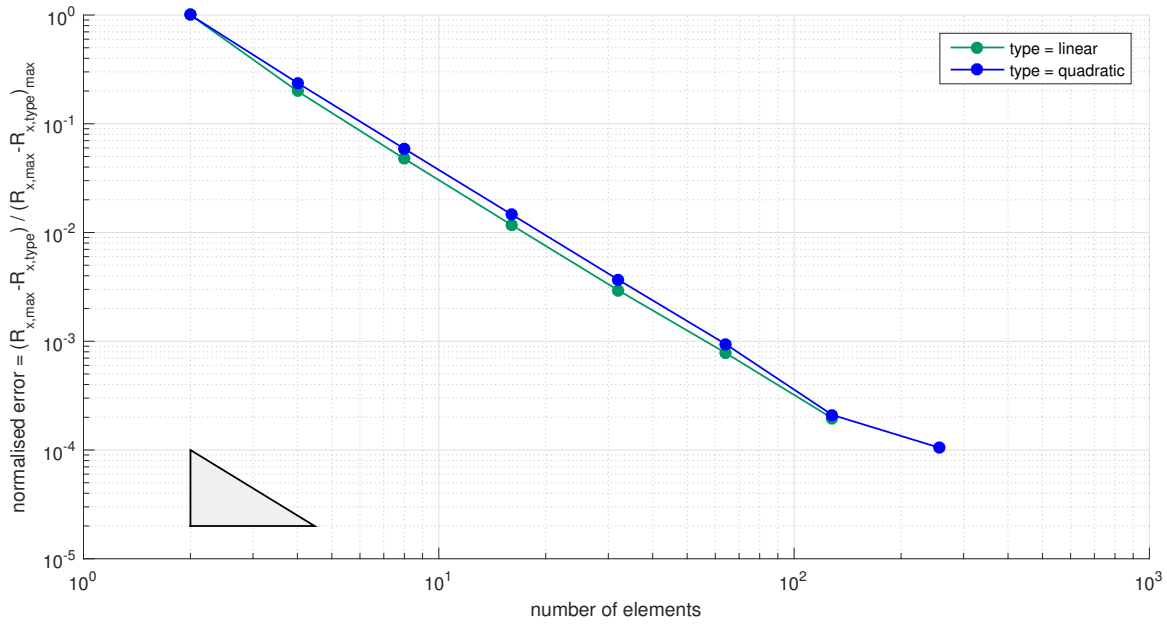

Figure 5.9: Error convergence of R_x at the left support

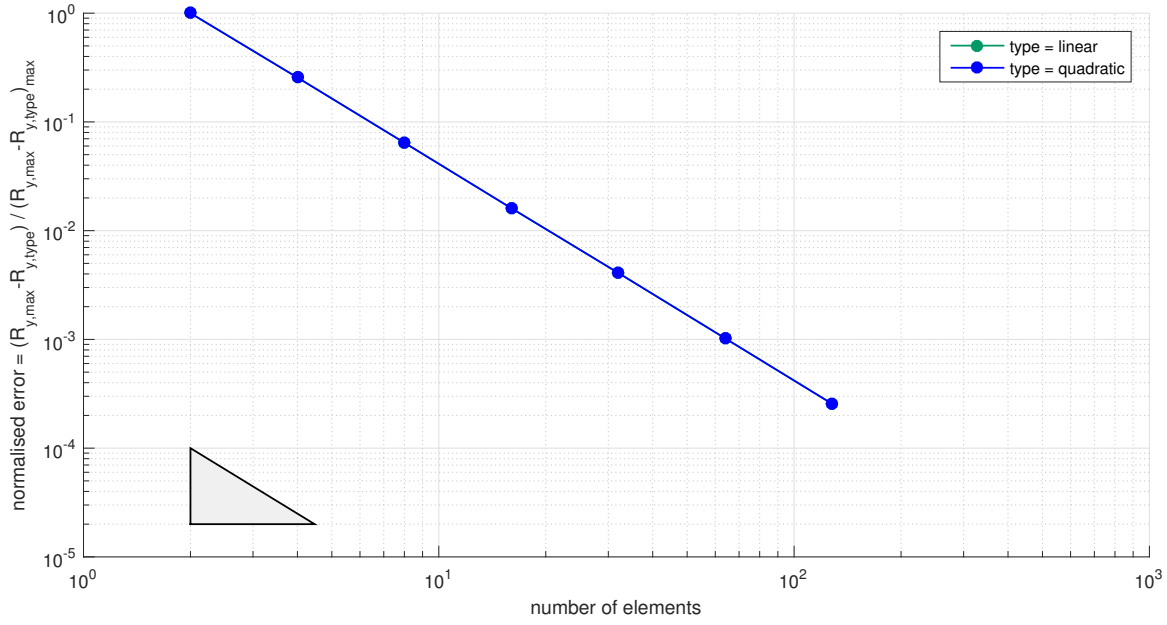

Figure 5.10: Error convergence of R_y at the left support

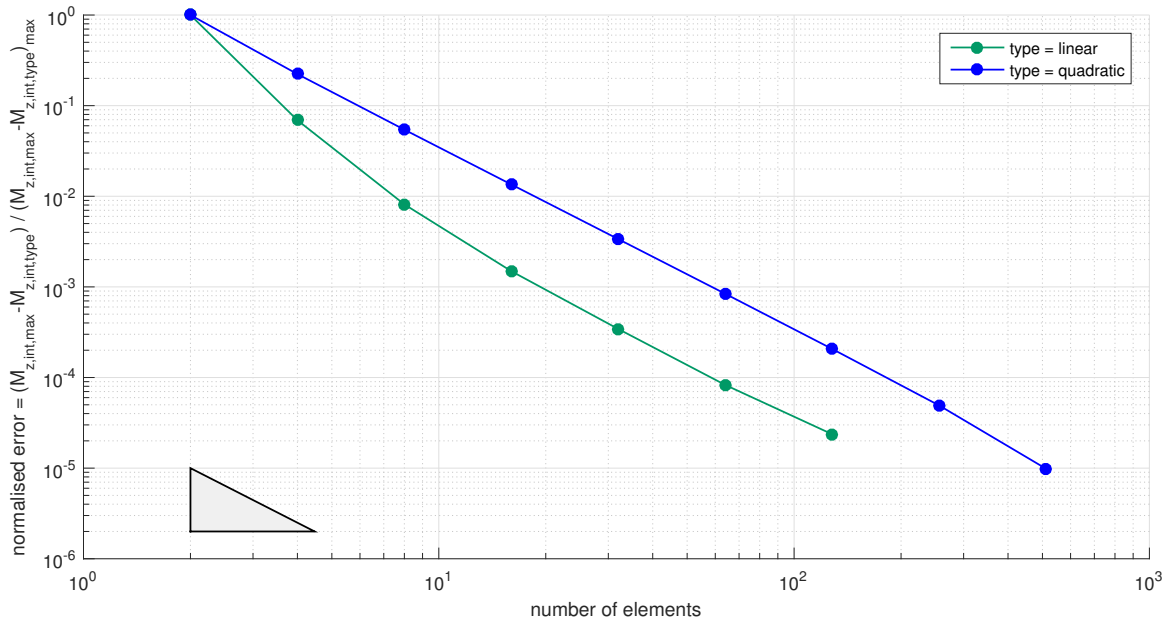

Figure 5.11: Error convergence of $M_{z,int}$ at the top

	linear	quadratic
v	-1.9	-1.9
θ	-2.0	-2.0
R_x	-2.1	-2.0
R_y	-2.0	-2.0
$M_{z,int}$	-3.5/-2.0	-2.1

Table 5.1: Slope values

Based on the spreadsheet data, a prominent distinction between the error convergence of the arch structure and the structures investigated in chapter 3 can be observed right away; it takes an equal amount of elements to obtain the most accurate approximation of the mechanical quantities of the arch, regardless of the element type. Indeed, in case of the horizontal reaction force at the left support and the internal bending moment at the top, an even smaller amount of linear elements is required. A value for R_x of 124.56 kN can be obtained with either 128 to 256 linear elements or 256 to 512 quadratic elements. As for $M_{z,int}$, the best approximation of 90.683 kNm can be found with either 128 to 256 linear elements or more than 1024 quadratic elements. Looking at the corresponding computation durations, it can be concluded that it would be more efficient to model the arch by linear elements.

The reason for this is the geometry convergence. With every iteration that the mesh size increases, the configuration alters and with that the structure as a whole; a gable roof typology like the one investigated in paragraph 4.4 is different than a gambrel roof typology. See figure 5.2. This translates in a similar convergence behaviour of the normalised error for linear and quadratic elements as can be observed from the diagrams and the corresponding tangents.

It appears that the slope values are the same as the ones determined in chapter 3, i.e. around -2.0. However, the accuracy of the quantity values can be increased by using curved quadratic elements. This is not explored any further in this text. As for the relative positioning of the convergence graphs, the linear line lying above the quadratic line in case of v and θ is because the maximum difference between the best and worst approximation is smaller for linear elements. The opposite can be observed for R_x and $M_{z,int}$.

As a final observation, the initial slope value of $M_{z,int}$ is discussed. Where the green line at the end has a similar slope as the blue line, at the beginning, a steeper trend can be observed. The cause of this can be assigned to the discrepancy between the deformation quantities for linear and quadratic elements in case of mesh sizes between 2 and 8. See row 3 to 5 of the spreadsheet. Because the value of these quantities is of influence on the internal force distribution, a different error development occurs. After all, we're dealing with a statically indeterminate structure.

Chapter 6

Conclusion

The goal of this project was twofold. First, gaining a better comprehension of the inner workings of a finite element program with a structural application. Second, answering the question: what is the influence of the mesh size and the element type used in a finite element protocol for trusses and frames, on the accuracy of the displacements and forces and how does it affect the computation time? Both these goals are achieved by developing, analysing and using four finite element programs created in the Matlab environment. That is, a program for trusses in 2D space, a program for trusses in 3D space and two programs for frames in 2D space (linear and quadratic). Also, a lot of emphasis is put on comparing the output of the programs with that of analytical computations and commercial finite element software. The discrepancy which can be observed during these comparisons results from the usage of different beam elements; where a Timoshenko element considers shear deformation, a Bernoulli-Euler beam does not. The result will be that a FE analyses gives an absolute higher deflection value than an analyses based on theory, because of a lower stiffness. As for the discrepancy with commercial finite element software, this is generally less and can presumably be explained by numerical precision. In the next three subsections the central question of this thesis will be answered.

An increase of the mesh size will lead to an accuracy increase of the values of the displacement quantities (u, v and θ) and force quantities (N, S, $M_{z,int}$, R_x, R_y and $M_{z,ext}$). For linear elements, the slope of the (normalised) error convergence, plotted in logspace, will have a value of around -2.0. This is whether or not the structure is statically indeterminate and/or needs to geometrically converge. As for quadratic elements, it takes a minimum amount of two per member to come to the most accurate approximation, regardless of the static determinacy. However, this does not apply when the geometry of the structure needs to converge as well. In that case the convergence of the normalised error will be the same for both linear and quadratic elements, i.e. a tangent of around -2.0. By applying curved quadratic elements the geometry of the structure will converge more rapidly and with that the convergence of the error as well. In general, one can state with regard to the slope of the log/log-curve: the steeper, the better; a steeper line indicates a more rapid convergence which is an indication of higher computation efficiency.

Sticking to straight elements, as argued in paragraph 3.4, it will be more time efficient to model a structure by linear elements if the required amount becomes

more than 256 per member. For instance, when the different quantities need be known at all 257 nodes per member or because a more accurate graphical presentation of the stress distributions is required. However, when the quantities at a limited amount of nodes need to be known, i.e. less than 257 nodes, it would be more efficient to use quadratic elements. Dealing with a curved structure like an arch, it is recommended to use linear elements, regardless of the number of elements.

Based on the 2D truss structure of paragraph 2.1 and 4.2, it only requires one linear bar element per member to come to a normal force distribution that is equally accurate as the one obtained with the commercial finite element software.

References

[1] M.J. Fagan. *Finite Element Analysis*. Addison-Wesley, 1992.

[2] C.A. Felippa. Web-posted Lectures on Nonlinear Finite Element Methods. `http://www.colorado.edu/engineering/cas/courses.d/NFEM.d/Home.html`. Accessed 5-August-2016.

[3] C. Hartsuijker. *Mechanica: Evenwicht*. Academic Service, 2015.

[4] C. Hartsuijker. *Mechanica: Spanningen, vervormingen, verplaatsingen*. Academic Service, 2016.

[5] C. Hartsuijker and J.W. Welleman. *Mechanica: Statisch onbepaalde constructies en bezwijkanalyse*. Academic Service, 2014.

[6] F.P. van der Meer. *Stability of Structures: CIE5144 Lecture Notes*. 2016.

Appendix A

Trusses3d.m

```
%% SECTION 1
% Load input and setup empty vectors and matrices

structure = 'cube_01_16.xlsx';

nodes = xlsread(structure,1);
elements = xlsread(structure,2);
boundaries = xlsread(structure,3);
numnode = size(nodes,1);
numelem = size(elements,1);

k = zeros(3*numnode,3*numnode); %global stiffness matrix
U = zeros(3*numnode,1); %global displacements of the nodes
epsilon = zeros(numelem,1);
sigma = zeros(numelem,1);
F_int = zeros(numelem,1);

%% SECTION 2
% Constructing k

for p = 1:numelem
    x_i = nodes(elements(p,1),1);
    y_i = nodes(elements(p,1),2);
    z_i = nodes(elements(p,1),3);
    x_j = nodes(elements(p,2),1);
    y_j = nodes(elements(p,2),2);
    z_j = nodes(elements(p,2),3);

    L = sqrt((x_j-x_i)^2+(y_j-y_i)^2+(z_j-z_i)^2);
    l = (x_j-x_i)/L;
    m = (y_j-y_i)/L;
    n = (z_j-z_i)/L;

    eta = [l^2 l*m l*n; m*l m^2 m*n; n*l n*m n^2];

    entries = [3*elements(p,1)-2; 3*elements(p,1)-1;...
               3*elements(p,1); 3*elements(p,2)-2;...
               3*elements(p,2)-1; 3*elements(p,2)];

    k(entries,entries) = k(entries,entries)+elements(p,3)*...
                         elements(p,4)/L*[eta -eta; -eta eta];
end
```

```
43
44  %% SECTION 3
45  % Rearranging and partitioning k
46  % [k_11 k_12; k_21 k_22]*{U_1; U_2} = {F_1; F_2}
47  % U_1 and F_2 -> unknown degrees of freedom and unknown nodal forces
48  % U_2 and F_1 -> known degrees of freedom and known nodal forces
49
50  vector_1 = zeros(3*numnode,1);
51  vector_2 = zeros(3*numnode,1);
52  counter = 0;
53  for p = 1:3*numnode
54      if boundaries(p,1) ~= 0
55          vector_1(p) = p;
56          counter = counter+1;
57      else
58          vector_2(p) = p;
59      end
60  end
61  vector_1 = vector_1(vector_1~=0);
62  vector_2 = vector_2(vector_2~=0);
63  vector_1(counter+1:3*numnode) = vector_2; %index vector
64
65  matrix = k(vector_1,vector_1); %rearranged stiffness matrix
66
67  k_11 = matrix(1:counter,1:counter);
68  k_12 = matrix(1:counter,counter+1:3*numnode);
69  k_21 = matrix(counter+1:3*numnode,1:counter);
70  k_22 = matrix(counter+1:3*numnode,counter+1:3*numnode);
71
72  %% SECTION 4
73  % Determining U and F_ext
74  % [k_11]*{U_1}+[k_12]*{U_2} = {F_1}
75  % [k_21]*{U_1}+[k_22]*{U_2} = {F_2}
76
77  U_2 = zeros(3*numnode-counter,1);
78  F_1 = boundaries(1:end,2); F_1 = F_1(~isnan(F_1));
79  U_1 = k_11\(F_1-k_12*U_2); %[k]*{U} = {F} -> {U} = [k]\{F}
80  F_2 = k_21*U_1+k_22*U_2;
81
82  U(vector_1(1:counter)) = U_1;
83  F_ext = boundaries(1:end,2); F_ext(isnan(F_ext)) = F_2;
84
85  %% SECTION 5
86  % Determining the strains, stresses and internal forces
87
88  for p = 1:numelem
89      x_i = nodes(elements(p,1),1);
90      y_i = nodes(elements(p,1),2);
91      z_i = nodes(elements(p,1),3);
92      x_j = nodes(elements(p,2),1);
93      y_j = nodes(elements(p,2),2);
94      z_j = nodes(elements(p,2),3);
95
96      L = sqrt((x_j-x_i)^2+(y_j-y_i)^2+(z_j-z_i)^2);
97      l = (x_j-x_i)/L;
98      m = (y_j-y_i)/L;
99      n = (z_j-z_i)/L;
100
101     entries = [3*elements(p,1)-2; 3*elements(p,1)-1;...
102                3*elements(p,1); 3*elements(p,2)-2;...
103                3*elements(p,2)-1; 3*elements(p,2)];
104
```

```
105     U_local = [l m n 0 0 0; 0 0 0 l m n]*U(entries);
106     epsilon(p) = (U_local(2)-U_local(1))/L;
107     sigma(p) = elements(p,3)*epsilon(p);
108     F_int(p) = elements(p,4)*sigma(p);
109 end
110
111 %% SECTION 6
112 % Merger of 'nodes' and 'elements'
113 % Row-format: [x_i,y_i,z_i,x_j,y_j,z_j]
114
115 matrix_1 = zeros(numelem,6);
116 for p = 1:numelem
117     matrix_1(p,1:3) = nodes(elements(p,1),:);
118     matrix_1(p,4:6) = nodes(elements(p,2),:);
119 end
120
121 %% SECTION 7
122 % Coordinates of displaced nodes
123 % In format of 'nodes'
124
125 vector_dx = U(1:3:end)*10^3;
126 vector_dy = U(2:3:end)*10^3;
127 vector_dz = U(3:3:end)*10^3;
128
129 disnodes(1:numnode,1) = nodes(1:numnode,1) + vector_dx;
130 disnodes(1:numnode,2) = nodes(1:numnode,2) + vector_dy;
131 disnodes(1:numnode,3) = nodes(1:numnode,3) + vector_dz;
132
133 %% SECTION 8
134 % Merger of 'disnodes' and 'elements'
135 % Row-format: [x_i,y_i,z_i,x_j,y_j,z_j]
136
137 matrix_2 = zeros(numelem,6);
138 for p = 1:numelem
139     matrix_2(p,1:3) = disnodes(elements(p,1),:);
140     matrix_2(p,4:6) = disnodes(elements(p,2),:);
141 end
142
143 %% SECTION 9
144 % Graphical output
145
146 figure
147 hold on
148
149 for n = 1:numelem
150     line_undef = line([matrix_1(n,1),matrix_1(n,4)],...
151                       [matrix_1(n,2),matrix_1(n,5)],...
152                       [matrix_1(n,3),matrix_1(n,6)]);
153     line_undef.LineStyle = '--';
154     line_undef.LineWidth = 0.5;
155     line_undef.Color = 'k';
156 end
157
158 for n = 1:numelem
159     line_def = line([matrix_2(n,1),matrix_2(n,4)],...
160                     [matrix_2(n,2),matrix_2(n,5)],...
161                     [matrix_2(n,3),matrix_2(n,6)]);
162     line_def.LineStyle = '-';
163     line_def.LineWidth = 1;
164     line_def.Color = 'k';
165 end
166
```

```
167 for p = 1:numnode
168     text(nodes(p,1),nodes(p,2),nodes(p,3),...
169     ['N #' num2str(p)],'Color','r');
170 end
171
172 xlabel('x-coordinate [mm]');
173 ylabel('y-coordinate [mm]');
174 zlabel('z-coordinate [mm]');
175 title('Deformed structure, amplification = 10^3');
176 % set(gca,'xtick',[])
177 % set(gca,'ytick',[])
178 % set(gca,'ztick',[])
179 grid on
180
181 hold off
182
183 %% SECTION 10
184 % Non-graphical output
185 % Nodal (external) forces, nodal displacements and internal forces
186
187 node = transpose(1:numnode);
188 F_x_ext = F_ext(1:3:end)*10^0;
189 F_y_ext = F_ext(2:3:end)*10^0;
190 F_z_ext = F_ext(3:3:end)*10^0;
191 dx = U(1:3:end);
192 dy = U(2:3:end);
193 dz = U(3:3:end);
194 table(node,F_x_ext,F_y_ext,F_z_ext,dx,dy,dz)
195
196 element = transpose(1:numelem);
197 sigma = sigma*10^0;
198 N = F_int*10^0;
199 table(element,epsilon,sigma,N)
```

Appendix B

FramesLinear2d.m

```
%% SECTION 1
% Load input and setup empty vectors and matrices

structure = 'beam_01_04.xlsx';

nodes = xlsread(structure,1);
elements = xlsread(structure,2);
boundaries = xlsread(structure,3);
numnode = size(nodes,1);
numelem = size(elements,1);

k = zeros(3*numnode,3*numnode); %global stiffness matrix
U = zeros(3*numnode,1); %global displacements of the nodes
q_glob = zeros(3*numnode,1); %global distributed force vector

epsilon = zeros(numelem,1); %axial strain
gamma = zeros(numelem,1); %shear strain
kappa = zeros(numelem,1); %bending strain

%% SECTION 2
% Constructing k and q_glob

for n = 1:numelem
    x_i = nodes(elements(n,1),1);
    y_i = nodes(elements(n,1),2);
    x_j = nodes(elements(n,2),1);
    y_j = nodes(elements(n,2),2);

    L = sqrt((x_j-x_i)^2+(y_j-y_i)^2);
    l = (x_j-x_i)/L;
    m = (y_j-y_i)/L;

    E = elements(n,3);
    G = E/(2*(1+0.3));
    A = elements(n,4);
    I = elements(n,5);

    s_i = 0;
    s_j = L;
    s = L/2;

    N_i = (s_j-s)/L;
```

```
43      N_j = (s-s_i)/L;
44
45      lambda = [l m 0 0 0 0; -m l 0 0 0 0; 0 0 1 0 0 0;...
46                0 0 0 l m 0; 0 0 0 -m l 0; 0 0 0 0 0 1];
47
48      entries = [3*elements(n,1)-2; 3*elements(n,1)-1;...
49                 3*elements(n,1); 3*elements(n,2)-2;...
50                 3*elements(n,2)-1; 3*elements(n,2)];
51
52      q = [elements(n,6); elements(n,7); 0];
53
54      N = [N_i 0 0 N_j 0 0; 0 N_i 0 0 N_j 0; 0 0 N_i 0 0 N_j];
55      B = 1/L*[-1 0 0 1 0 0; 0 -1 -L*N_i 0 1 -L*N_j;...
56               0 0 -1 0 0 1]*lambda;
57      D = [E*A 0 0; 0 G*A 0; 0 0 E*I];
58
59      k(entries,entries) = k(entries,entries)+transpose(B)*D*B*L;
60      q_glob(entries) = q_glob(entries)+transpose(N)*q*L;
61  end
62
63  %% SECTION 3
64  % Rearranging and partitioning k
65  % [k_11 k_12; k_21 k_22]*{U_1; U_2} = {F_1; F_2}
66  % U_1 and F_2 -> unknown degrees of freedom and
67  %                unknown nodal stresses
68  % U_2 and F_1 -> known degrees of freedom and
69  %                known nodal stresses
70
71  vector = zeros(3*numnode,1); %index vector
72  vector_a = zeros(3*numnode,1); %first part of the index vector
73  vector_b = zeros(3*numnode,1); %second part of the index vector
74  counter = 0;
75  for n = 1:3*numnode
76      if boundaries(n,1) ~= 0
77          vector_a(n) = n;
78          counter = counter+1;
79      else
80          vector_b(n) = n;
81      end
82  end
83  vector_a = vector_a(vector_a~=0);
84  vector(1:counter) = vector_a;
85  vector_b = vector_b(vector_b~=0);
86  vector(counter+1:3*numnode) = vector_b;
87
88  matrix = k(vector,vector); %rearranged stiffness matrix
89
90  k_11 = matrix(1:counter,1:counter);
91  k_12 = matrix(1:counter,counter+1:3*numnode);
92  k_21 = matrix(counter+1:3*numnode,1:counter);
93  k_22 = matrix(counter+1:3*numnode,counter+1:3*numnode);
94
95  %% SECTION 4
96  % Determining U and F_ext
97  % [k_11]*{U_1}+[k_12]*{U_2} = {F_1}
98  % [k_21]*{U_1}+[k_22]*{U_2} = {F_2}
99
100 U_2 = zeros(3*numnode-counter,1);
101 F_1 = boundaries(vector_a,2)+q_glob(vector_a);
102 U_1 = k_11\(F_1-k_12*U_2); %[k]*{U} = {F} -> {U} = [k]\{F}
103 F_2 = k_21*U_1+k_22*U_2;
104
```

```
105 U(vector_a) = U_1;
106 boundaries(vector_a,2) = F_1;
107 F_ext = boundaries(1:end,2);
108 F_ext(isnan(F_ext)) = F_2-q_glob(vector_b);
109
110 %% SECTION 5
111 % Merger of 'nodes' and 'elements'
112 % Row-format: [x_i,y_i,x_j,y_j]
113
114 matrix_1 = zeros(numelem,4);
115 for p = 1:numelem
116     matrix_1(p,[1 2]) = nodes(elements(p,1),:);
117     matrix_1(p,[3 4]) = nodes(elements(p,2),:);
118 end
119
120 %% SECTION 6
121 % Coordinates of displaced nodes
122 % In format of 'nodes'
123
124 vector_dx = U(1:3:end)*10^3;
125 vector_dy = U(2:3:end)*10^3;
126
127 disnodes(1:numnode,1) = nodes(1:numnode,1) + vector_dx;
128 disnodes(1:numnode,2) = nodes(1:numnode,2) + vector_dy;
129
130 %% SECTION 7
131 % Merger of 'disnodes' and 'elements'
132 % Row-format: [x_i,y_i,x_j,y_j]
133
134 matrix_2 = zeros(numelem,4);
135 for p = 1:numelem
136     matrix_2(p,[1 2]) = disnodes(elements(p,1),:);
137     matrix_2(p,[3 4]) = disnodes(elements(p,2),:);
138 end
139
140 %% SECTION 8
141 % Graphical output: deformed structure
142
143 figure
144 hold on
145
146 for n = 1:numelem
147     line_undef = line([matrix_1(n,1),matrix_1(n,3)],...
148                       [matrix_1(n,2),matrix_1(n,4)]);
149     line_undef.LineStyle = '--';
150     line_undef.LineWidth = 0.75;
151     line_undef.Color = 'k';
152     text((matrix_1(n,1)+matrix_1(n,3))/2,...
153          (matrix_1(n,2)+matrix_1(n,4))/2,...
154          ['E #' num2str(n)],'Color','r');
155 end
156
157 for n = 1:numelem
158     line_def = line([matrix_2(n,1),matrix_2(n,3)],...
159                     [matrix_2(n,2),matrix_2(n,4)]);
160     line_undef.LineStyle = '-';
161     line_def.LineWidth = 0.75;
162     line_def.Color = 'k';
163 end
164
165 for n = 1:numnode
166     scatter(nodes(n,1),nodes(n,2),'k');
```

```
    scatter(disnodes(n,1),disnodes(n,2),'k');
    text(nodes(n,1),nodes(n,2),['N #' num2str(n)],'Color','r');
end

xlabel('x-coordinate [mm]');
ylabel('y-coordinate [mm]');
title('Deformed structure, amplification = 10^3');
grid on

hold off

%% SECTION 9
% Graphical and non-graphical output: N, S and M_z_int
% [k_elem]*{U_elem} = {F_int} with {U_elem} = {U(entries)}

figure
hold on

for n = 1:numelem
    x_i = nodes(elements(n,1),1);
    y_i = nodes(elements(n,1),2);
    x_j = nodes(elements(n,2),1);
    y_j = nodes(elements(n,2),2);

    L = sqrt((x_j-x_i)^2+(y_j-y_i)^2);
    l = (x_j-x_i)/L;
    m = (y_j-y_i)/L;

    E = elements(n,3);
    G = E/(2*(1+0.3));
    A = elements(n,4);
    I = elements(n,5);

    s_i = 0;
    s_j = L;
    s = L/2;

    N_i = (s_j-s)/L;
    N_j = (s-s_i)/L;

    lambda = [l m 0 0 0 0; -m l 0 0 0 0; 0 0 1 0 0 0;...
              0 0 0 l m 0; 0 0 0 -m l 0; 0 0 0 0 0 1];

    entries = [3*elements(n,1)-2; 3*elements(n,1)-1;...
               3*elements(n,1); 3*elements(n,2)-2;...
               3*elements(n,2)-1; 3*elements(n,2)];

    q = [elements(n,6); elements(n,7); 0];

    N = [N_i 0 0 N_j 0 0; 0 N_i 0 0 N_j 0; 0 0 N_i 0 0 N_j];
    B = 1/L*[-1 0 0 1 0 0; 0 -1 -L*N_i 0 1 -L*N_j;...
             0 0 -1 0 0 1]*lambda;
    D = [E*A 0 0; 0 G*A 0; 0 0 E*I];

    k_elem = transpose(B)*D*B*L;
    q_elem = transpose(N)*q*L;

    U_elem = U(entries);
    F_int = k_elem*U_elem;

    F_x_int = (F_int(1:3:end)+diag(abs(q_elem(1:3:end)))*...
               sign(F_int(1:3:end)))*10^0;
```

```
229     F_y_int = (F_int(2:3:end)+diag(abs(q_elem(2:3:end)))*...
230                sign(F_int(2:3:end)))*10^0;
231     N = [l*F_x_int(1)+m*F_y_int(1); l*F_x_int(2)+m*F_y_int(2)];
232     S = [-m*F_x_int(1)+l*F_y_int(1); -m*F_x_int(2)+l*F_y_int(2)];
233     M_z_int = F_int(3:3:end)*10^0;
234
235     strains = B*U_elem;
236     epsilon(n) = strains(1);
237     gamma(n) = strains(2);
238     kappa(n) = strains(3);
239
240     element = [n; n];
241     node = [elements(n,1); elements(n,2)];
242     table(element,node,N,S,M_z_int)
243
244     x_1 = x_i-m*S(1)*10^-2;
245     y_1 = y_i+l*S(1)*10^-2;
246     x_2 = x_1+(x_j-x_i);
247     y_2 = y_1+(y_j-y_i);
248
249     x_3 = x_i-m*M_z_int(1)*10^-5;
250     y_3 = y_i+l*M_z_int(1)*10^-5;
251     x_4 = x_j-m*abs(M_z_int(2))*sign(M_z_int(1))*10^-5;
252     y_4 = y_j+l*abs(M_z_int(2))*sign(M_z_int(1))*10^-5;
253
254     matrix_S = [x_i y_i x_1 y_1; x_1 y_1 x_2 y_2;
255                 x_2 y_2 x_j y_j]; %coordinates S-line
256     matrix_M = [x_i y_i x_3 y_3; x_3 y_3 x_4 y_4;
257                 x_4 y_4 x_j y_j]; %coordinates M-line
258
259     X_S = [matrix_S(:,1) matrix_S(:,3)];
260     Y_S = [matrix_S(:,2) matrix_S(:,4)];
261     plot(X_S,Y_S,'Color',[1 0.5 0],'LineWidth',2) %S-line
262     X_M = [matrix_M(:,1) matrix_M(:,3)];
263     Y_M = [matrix_M(:,2) matrix_M(:,4)];
264     plot(X_M,Y_M,'Color',[0.5 0 0.5],'LineWidth',2) %M-line
265 end
266
267 for n = 1:numelem
268     line_undef = line([matrix_1(n,1),matrix_1(n,3)],...
269                       [matrix_1(n,2),matrix_1(n,4)]);
270     line_undef.LineStyle = '--';
271     line_undef.LineWidth = 0.75;
272     line_undef.Color = 'k';
273     text((matrix_1(n,1)+matrix_1(n,3))/2,...
274          (matrix_1(n,2)+matrix_1(n,4))/2,...
275          ['E #' num2str(n)],'Color','r');
276 end
277
278 for n = 1:numnode
279     scatter(nodes(n,1),nodes(n,2),'k');
280     text(nodes(n,1),nodes(n,2),['N #' num2str(n)],'Color','r');
281 end
282
283 set(gca,'xtick',[])
284 set(gca,'ytick',[])
285 title('S-line and M-line')
286
287 hold off
288
289 %% SECTION 10
290 % Non-graphical output: displacements, stresses and strains
```

```
291
292 node = reshape(1:numnode,numnode,1);
293 element = reshape(1:numelem,numelem,1);
294
295 u = U(1:3:end);
296 v = U(2:3:end);
297 theta = U(3:3:end);
298
299 F_point = boundaries(1:end,2)-q_glob; F_point(isnan(F_point)) = 0;
300 F_x = F_point(1:3:end)*10^0;
301 F_y = F_point(2:3:end)*10^0;
302 T_z = F_point(3:3:end)*10^0;
303
304 q_x = q_glob(1:3:end)*10^0;
305 q_y = q_glob(2:3:end)*10^0;
306
307 R = zeros(3*numnode,1); R(vector_b) = F_ext(vector_b);
308 R_x = R(1:3:end)*10^0;
309 R_y = R(2:3:end)*10^0;
310 M_z_ext = R(3:3:end)*10^0;
311
312 table(node,u,v,theta,F_x,F_y,T_z,q_x,q_y,R_x,R_y,M_z_ext)
313 table(element,epsilon,gamma,kappa)
```

Appendix C

FramesQuadratic2d.m

```
%% SECTION 1
% Load input and setup empty vectors and matrices

structure = 'beam_02_04.xlsx';

nodes = xlsread(structure,1);
elements = xlsread(structure,2);
boundaries = xlsread(structure,3);
numnode = size(nodes,1);
numelem = size(elements,1);

k = zeros(3*numnode,3*numnode); %global stiffness matrix
U = zeros(3*numnode,1); %global displacements of the nodes
q_glob = zeros(3*numnode,1); %global distributed force vector

epsilon = zeros(numelem,1); %axial strain
gamma = zeros(numelem,1); %shear strain
kappa = zeros(numelem,1); %bending strain

%% SECTION 2
% Constructing k and q_glob

for n = 1:numelem/2
    x_i = nodes(elements(2*n-1,1),1);
    y_i = nodes(elements(2*n-1,1),2);
    x_k = nodes(elements(2*n,2),1);
    y_k = nodes(elements(2*n,2),2);

    L = sqrt((x_k-x_i)^2+(y_k-y_i)^2);
    l = (x_k-x_i)/L;
    m = (y_k-y_i)/L;

    E = elements(2*n-1,3);
    G = E/(2*(1+0.3));
    A = elements(2*n-1,4);
    I = elements(2*n-1,5);

    s_i = 0;
    s_a = (-1+sqrt(3))/(2*sqrt(3))*L;
%   first integration point
    s_b = (1+sqrt(3))/(2*sqrt(3))*L;
%   second integration point
```

```
43      L_1_a = (s_a-s_i)/L;
44  %   natural coordinate of first integration point
45      L_1_b = (s_b-s_i)/L;
46  %   natural coordinate of second integration point
47
48      N_i_a = 1-3*L_1_a+2*L_1_a^2;  N_i_b = 1-3*L_1_b+2*L_1_b^2;
49      N_j_a = 4*L_1_a-4*L_1_a^2;    N_j_b = 4*L_1_b-4*L_1_b^2;
50      N_k_a = -L_1_a+2*L_1_a^2;     N_k_b = -L_1_b+2*L_1_b^2;
51
52      dN_i_a = (-3+4*L_1_a)/L;      dN_i_b = (-3+4*L_1_b)/L;
53      dN_j_a = (4-8*L_1_a)/L;       dN_j_b = (4-8*L_1_b)/L;
54      dN_k_a = (-1+4*L_1_a)/L;      dN_k_b = (-1+4*L_1_b)/L;
55
56      lambda = zeros(9,9);
57      lambda(1:3,1:3) = [l m 0; -m l 0; 0 0 1];
58      lambda(4:6,4:6) = [l m 0; -m l 0; 0 0 1];
59      lambda(7:9,7:9) = [l m 0; -m l 0; 0 0 1];
60
61      entries = [3*elements(2*n-1,1)-2; 3*elements(2*n-1,1)-1;...
62                 3*elements(2*n-1,1);...
63                 3*elements(2*n-1,2)-2; 3*elements(2*n-1,2)-1;
64                 3*elements(2*n-1,2);...
65                 3*elements(2*n,2)-2; 3*elements(2*n,2)-1;...
66                 3*elements(2*n,2)];
67
68      q = [elements(2*n-1,6); elements(2*n-1,7); 0];
69
70      N_a = [N_i_a 0 0 N_j_a 0 0 N_k_a 0 0;...
71             0 N_i_a 0 0 N_j_a 0 0 N_k_a 0;...
72             0 0 N_i_a 0 0 N_j_a 0 0 N_k_a];
73      N_b = [N_i_b 0 0 N_j_b 0 0 N_k_b 0 0;...
74             0 N_i_b 0 0 N_j_b 0 0 N_k_b 0;...
75             0 0 N_i_b 0 0 N_j_b 0 0 N_k_b];
76
77      B_a = [dN_i_a 0 0 dN_j_a 0 0 dN_k_a 0 0;...
78             0 dN_i_a -N_i_a 0 dN_j_a -N_j_a 0 dN_k_a -N_k_a;...
79             0 0 dN_i_a 0 0 dN_j_a 0 0 dN_k_a]*lambda;
80      B_b = [dN_i_b 0 0 dN_j_b 0 0 dN_k_b 0 0;...
81             0 dN_i_b -N_i_b 0 dN_j_b -N_j_b 0 dN_k_b -N_k_b;...
82             0 0 dN_i_b 0 0 dN_j_b 0 0 dN_k_b]*lambda;
83
84      D = [E*A 0 0; 0 G*A 0; 0 0 E*I];
85
86      k(entries,entries) = k(entries,entries)+...
87                           transpose(B_a)*D*B_a*L/2+...
88                           transpose(B_b)*D*B_b*L/2;
89      q_glob(entries) = q_glob(entries)+...
90                        transpose(N_a)*q*L/2+...
91                        transpose(N_b)*q*L/2;
92  end
93
94  %% SECTION 3
95  % Rearranging and partitioning k
96  % [k_11 k_12; k_21 k_22]*{U_1; U_2} = {F_1; F_2}
97  % U_1 and F_2 -> unknown degrees of freedom and
98  %                unknown nodal stresses
99  % U_2 and F_1 -> known degrees of freedom and
100 %                known nodal stresses
101
102 vector = zeros(3*numnode,1); %index vector
103 vector_a = zeros(3*numnode,1); %first part of the index vector
104 vector_b = zeros(3*numnode,1); %second part of the index vector
```

```
counter = 0;
for n = 1:3*numnode
    if boundaries(n,1) ~= 0
        vector_a(n) = n;
        counter = counter+1;
    else
        vector_b(n) = n;
    end
end
vector_a = vector_a(vector_a~=0);
vector(1:counter) = vector_a;
vector_b = vector_b(vector_b~=0);
vector(counter+1:3*numnode) = vector_b;

matrix = k(vector,vector); %rearranged stiffness matrix

k_11 = matrix(1:counter,1:counter);
k_12 = matrix(1:counter,counter+1:3*numnode);
k_21 = matrix(counter+1:3*numnode,1:counter);
k_22 = matrix(counter+1:3*numnode,counter+1:3*numnode);

%% SECTION 4
% Determining U and F_ext
% [k_11]*{U_1}+[k_12]*{U_2} = {F_1}
% [k_21]*{U_1}+[k_22]*{U_2} = {F_2}

U_2 = zeros(3*numnode-counter,1);
F_1 = boundaries(vector_a,2)+q_glob(vector_a);
U_1 = k_11\(F_1-k_12*U_2); %[k]*{U} = {F} -> {U} = [k]\{F}
F_2 = k_21*U_1+k_22*U_2;

U(vector_a) = U_1;
boundaries(vector_a,2) = F_1;
F_ext = boundaries(1:end,2);
F_ext(isnan(F_ext)) = F_2-q_glob(vector_b);

%% SECTION 5
% Merger of 'nodes' and 'elements'
% Row-format: [x_i,y_i,x_j,y_j]

matrix_1 = zeros(numelem,4);
for p = 1:numelem
    matrix_1(p,[1 2]) = nodes(elements(p,1),:);
    matrix_1(p,[3 4]) = nodes(elements(p,2),:);
end

%% SECTION 6
% Coordinates of displaced nodes
% In format of 'nodes'

vector_dx = U(1:3:end)*10^3;
vector_dy = U(2:3:end)*10^3;

disnodes(1:numnode,1) = nodes(1:numnode,1) + vector_dx;
disnodes(1:numnode,2) = nodes(1:numnode,2) + vector_dy;

%% SECTION 7
% Merger of 'disnodes' and 'elements'
% Row-format: [x_i,y_i,x_j,y_j]

matrix_2 = zeros(numelem,4);
for p = 1:numelem
```

```
167     matrix_2(p,[1 2]) = disnodes(elements(p,1),:);
168     matrix_2(p,[3 4]) = disnodes(elements(p,2),:);
169 end
170
171 %% SECTION 8
172 % Graphical output: deformed structure
173
174 figure
175 hold on
176
177 label = zeros(numelem,1);
178 label(1:2:end) = transpose(1:numelem/2);
179 label(2:2:end) = transpose(1:numelem/2);
180
181 for n = 1:numelem
182     line_undef = line([matrix_1(n,1),matrix_1(n,3)],...
183                       [matrix_1(n,2),matrix_1(n,4)]);
184     line_undef.LineStyle = '--';
185     line_undef.LineWidth = 0.75;
186     line_undef.Color = 'k';
187     text((matrix_1(n,1)+matrix_1(n,3))/2,...
188          (matrix_1(n,2)+matrix_1(n,4))/2,...
189          ['E #' num2str(label(n))],'Color','r');
190 end
191
192 for n = 1:numelem
193     line_def = line([matrix_2(n,1),matrix_2(n,3)],...
194                     [matrix_2(n,2),matrix_2(n,4)]);
195     line_def.LineStyle = '-';
196     line_def.LineWidth = 0.75;
197     line_def.Color = 'k';
198 end
199
200 for n = 1:numnode
201     scatter(nodes(n,1),nodes(n,2),'k');
202     scatter(disnodes(n,1),disnodes(n,2),'k');
203     text(nodes(n,1),nodes(n,2),['N #' num2str(n)],'Color','r');
204 end
205
206 xlabel('x-coordinate [mm]');
207 ylabel('y-coordinate [mm]');
208 title('Deformed structure, amplification = 10^3');
209 grid on
210
211 hold off
212
213 %% SECTION 9
214 % Graphical and non-graphical output: N, S and M_z_int
215 % [k_elem]*{U_elem} = {F_int} with {U_elem} = {U(entries)}
216
217 figure
218 hold on
219
220 for n = 1:numelem/2
221     x_i = nodes(elements(2*n-1,1),1);
222     y_i = nodes(elements(2*n-1,1),2);
223     x_k = nodes(elements(2*n,2),1);
224     y_k = nodes(elements(2*n,2),2);
225
226     L = sqrt((x_k-x_i)^2+(y_k-y_i)^2);
227     l = (x_k-x_i)/L;
228     m = (y_k-y_i)/L;
```

```
229
230     E = elements(2*n-1,3);
231     G = E/(2*(1+0.3));
232     A = elements(2*n-1,4);
233     I = elements(2*n-1,5);
234
235     s_i = 0;
236     s_a = (-1+sqrt(3))/(2*sqrt(3))*L;
237 %   first integration point
238     s_b = (1+sqrt(3))/(2*sqrt(3))*L;
239 %   second integration point
240     L_1_a = (s_a-s_i)/L;
241 %   natural coordinate of first integration point
242     L_1_b = (s_b-s_i)/L;
243 %   natural coordinate of second integration point
244
245     N_i_a = 1-3*L_1_a+2*L_1_a^2;  N_i_b = 1-3*L_1_b+2*L_1_b^2;
246     N_j_a = 4*L_1_a-4*L_1_a^2;    N_j_b = 4*L_1_b-4*L_1_b^2;
247     N_k_a = -L_1_a+2*L_1_a^2;     N_k_b = -L_1_b+2*L_1_b^2;
248
249     dN_i_a = (-3+4*L_1_a)/L;      dN_i_b = (-3+4*L_1_b)/L;
250     dN_j_a = (4-8*L_1_a)/L;       dN_j_b = (4-8*L_1_b)/L;
251     dN_k_a = (-1+4*L_1_a)/L;      dN_k_b = (-1+4*L_1_b)/L;
252
253     lambda = zeros(9,9);
254     lambda(1:3,1:3) = [l m 0; -m l 0; 0 0 1];
255     lambda(4:6,4:6) = [l m 0; -m l 0; 0 0 1];
256     lambda(7:9,7:9) = [l m 0; -m l 0; 0 0 1];
257
258     entries = [3*elements(2*n-1,1)-2; 3*elements(2*n-1,1)-1;...
259                3*elements(2*n-1,1);...
260                3*elements(2*n-1,2)-2; 3*elements(2*n-1,2)-1;
261                3*elements(2*n-1,2);...
262                3*elements(2*n,2)-2; 3*elements(2*n,2)-1;...
263                3*elements(2*n,2)];
264
265     q = [elements(2*n-1,6); elements(2*n-1,7); 0];
266
267     N_a = [N_i_a 0 0 N_j_a 0 0 N_k_a 0 0;...
268            0 N_i_a 0 0 N_j_a 0 0 N_k_a 0;...
269            0 0 N_i_a 0 0 N_j_a 0 0 N_k_a];
270     N_b = [N_i_b 0 0 N_j_b 0 0 N_k_b 0 0;...
271            0 N_i_b 0 0 N_j_b 0 0 N_k_b 0;...
272            0 0 N_i_b 0 0 N_j_b 0 0 N_k_b];
273
274     B_a = [dN_i_a 0 0 dN_j_a 0 0 dN_k_a 0 0;...
275            0 dN_i_a -N_i_a 0 dN_j_a -N_j_a 0 dN_k_a -N_k_a;...
276            0 0 dN_i_a 0 0 dN_j_a 0 0 dN_k_a]*lambda;
277     B_b = [dN_i_b 0 0 dN_j_b 0 0 dN_k_b 0 0;...
278            0 dN_i_b -N_i_b 0 dN_j_b -N_j_b 0 dN_k_b -N_k_b;...
279            0 0 dN_i_b 0 0 dN_j_b 0 0 dN_k_b]*lambda;
280
281     D = [E*A 0 0; 0 G*A 0; 0 0 E*I];
282
283     k_elem = transpose(B_a)*D*B_a*L/2+...
284              transpose(B_b)*D*B_b*L/2;
285     q_elem = transpose(N_a)*q*L/2+...
286              transpose(N_b)*q*L/2;
287
288     U_elem = U(entries);
289     F_int = k_elem*U_elem;
290
```

```
    F_x_int = F_int(1:3:end)*10^0;
%   +diag(abs(q_elem(1:3:end)))*sign(F_int(1:3:end));
%   Does not contribute to the interpretation of
%   the shear force distribution!
    F_y_int = F_int(2:3:end)*10^0;
%   +diag(abs(q_elem(2:3:end)))*sign(F_int(2:3:end));
%   Does not contribute to the interpretation of
%   the shear force distribution!
    N = [l*F_x_int(1)+m*F_y_int(1); l*F_x_int(2)+m*F_y_int(2);...
         l*F_x_int(3)+m*F_y_int(3)];
    S = [-m*F_x_int(1)+l*F_y_int(1); -m*F_x_int(2)+l*F_y_int(2);...
         -m*F_x_int(3)+l*F_y_int(3)];
    M_z_int = F_int(3:3:end)*10^0;

    strains_a = B_a*U_elem;
    strains_b = B_b*U_elem;
    epsilon(2*n-1) = strains_a(1);  epsilon(2*n) = strains_b(1);
    gamma(2*n-1) = strains_a(2);    gamma(2*n) = strains_b(2);
    kappa(2*n-1) = strains_a(3);    kappa(2*n) = strains_b(3);

    element = [n; n; n];
    node = [elements(2*n-1,1); elements(2*n-1,2); elements(2*n,2)];
    table(element,node,N,S,M_z_int)

    x_1 = x_i-m*S(1)*10^-2; y_1 = y_i+l*S(1)*10^-2;
    x_2 = x_1+(x_k-x_i)/2;  y_2 = y_1+(y_k-y_i)/2;
    x_3 = x_2-m*S(2)*10^-2; y_3 = y_2+l*S(2)*10^-2;
    x_4 = x_3+(x_k-x_i)/2;  y_4 = y_3+(y_k-y_i)/2;

    x_5 = x_i-m*M_z_int(1)*10^-5;
    y_5 = y_i+l*M_z_int(1)*10^-5;
    x_6 = x_k-m*abs(M_z_int(3))*sign(M_z_int(1))*10^-5;
    y_6 = y_k+l*abs(M_z_int(3))*sign(M_z_int(1))*10^-5;

    matrix_S = [x_i y_i x_1 y_1; x_1 y_1 x_2 y_2;
                x_2 y_2 x_3 y_3; x_3 y_3 x_4 y_4;
                x_4 y_4 x_k y_k]; %coordinates S-line
    matrix_M = [x_i y_i x_5 y_5; x_5 y_5 x_6 y_6;
                x_6 y_6 x_k y_k]; %coordinates M-line

    X_S = [matrix_S(:,1) matrix_S(:,3)];
    Y_S = [matrix_S(:,2) matrix_S(:,4)];
    plot(X_S,Y_S,'Color',[1 0.5 0],'LineWidth',2) %S-line
    X_M = [matrix_M(:,1) matrix_M(:,3)];
    Y_M = [matrix_M(:,2) matrix_M(:,4)];
    plot(X_M,Y_M,'Color',[0.5 0 0.5],'LineWidth',2) %M-line
end

label = zeros(numelem,1);
label(1:2:end) = transpose(1:numelem/2);
label(2:2:end) = transpose(1:numelem/2);

for n = 1:numelem
    line_undef = line([matrix_1(n,1),matrix_1(n,3)],...
                      [matrix_1(n,2),matrix_1(n,4)]);
    line_undef.LineStyle = '--';
    line_undef.LineWidth = 0.75;
    line_undef.Color = 'k';
    text((matrix_1(n,1)+matrix_1(n,3))/2,...
         (matrix_1(n,2)+matrix_1(n,4))/2,...
         ['E #' num2str(label(n))],'Color','r');
end
```

```
353
354 for n = 1:numnode
355     scatter(nodes(n,1),nodes(n,2),'k');
356     text(nodes(n,1),nodes(n,2),['N #' num2str(n)],'Color','r');
357 end
358
359 set(gca,'xtick',[])
360 set(gca,'ytick',[])
361 title('S-line and M-line')
362
363 hold off
364
365 %% SECTION 10
366 % Non-graphical output: displacements, stresses and strains
367
368 node = reshape(1:numnode,numnode,1);
369 element = label;
370 integration_point = zeros(numelem,1);
371 integration_point(1:2:end) = 1;
372 integration_point(2:2:end) = 2;
373
374 u = U(1:3:end);
375 v = U(2:3:end);
376 theta = U(3:3:end);
377
378 F_point = boundaries(1:end,2)-q_glob; F_point(isnan(F_point)) = 0;
379 F_x = F_point(1:3:end)*10^0;
380 F_y = F_point(2:3:end)*10^0;
381 T_z = F_point(3:3:end)*10^0;
382
383 q_x = q_glob(1:3:end)*10^0;
384 q_y = q_glob(2:3:end)*10^0;
385
386 R = zeros(3*numnode,1); R(vector_b) = F_ext(vector_b);
387 R_x = R(1:3:end)*10^0;
388 R_y = R(2:3:end)*10^0;
389 M_z_ext = R(3:3:end)*10^0;
390
391 table(node,u,v,theta,F_x,F_y,T_z,q_x,q_y,R_x,R_y,M_z_ext)
392 table(element,integration_point,epsilon,gamma,kappa)
```