Practical Training I

Preparation

During the Introduction to the Finite Element Method course, three practical exercises must be executed before students are admitted to the exams. The exercises are elaborated with a computer and should be performed by groups of two students. Every exercise will take half a day. This will only be enough if the exercise is properly prepared. This means that you should be up to date with the lectures, have studied the exercise description and have thought about how to approach the problem. The instructors will determine whether an exercise was fulfilled sufficiently. During the exercise some data must be sorted out and registered. The instructor can ask questions like: 'if this changes in the exercise, how would that affect your elaboration'. With this, you will have to show that you understand what you are doing. It is not required to write a report. The exercises give access to the exams, it does not contribute to the final mark.

The first practicum concerns a truss frame. In this exercise, the displacements of nodes and forces in trusses must be calculated with the help of the MATLAB program. MATLAB is not a finite element program, but a general analysis program in which it is relatively easy to manipulate with matrices and vectors.

In the second practicum a beam frame must be modeled. Here, besides normal strains like in trusses, also bending plays a role. In this exercise, displacements, rotations, forces and torques must be calculated. This is done with the widely used commercial finite element program ANSYS.

During the last practicum a structure must be modeled with two-dimensional planar elements. By holes or notches, stress concentrations are introduced in this structure. The goal of this exercise is to calculate the equivalent stiffness and the maximum stress. The determination of the accuracy of the solution is also a part of this exercise using ANSYS.

1 Truss framework

1.1 The exercise

In the finite element method it is common to make use of matrices and vectors. Using such notation enables to write the needed formulas in a general way. Hence, the formulas will be suitable for both a simple 2-dimensional truss-element as for a complex 3-dimensional volume element. The content and the size of the matrices and vectors will be different, but the used formulas can be the same.

During the first practicum you will be training with this matrix-vector formulation. As an example the simplest element type is chosen, a 2-dimensional truss element which is being used in a truss framework (see chapter 2 of the reader). During the practicum you will receive the exact exercise. You can prepare for the practicum using the (representative) example in figure 1.1.

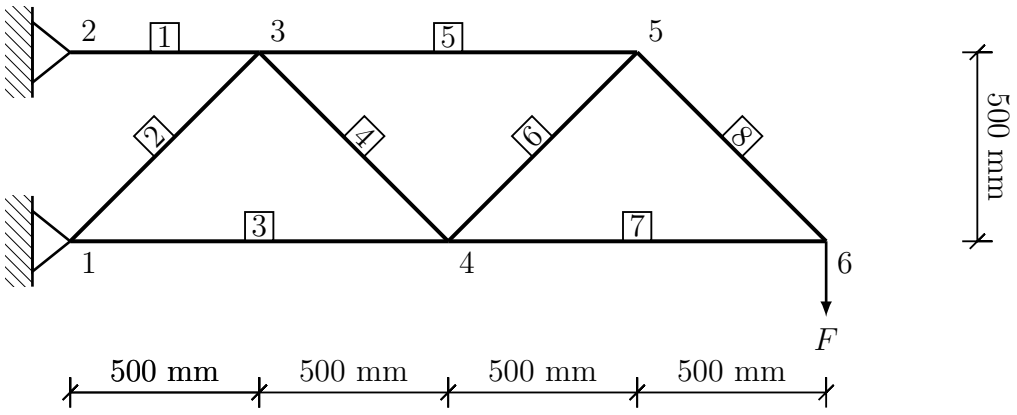

Figure 1.1: Truss framework

The elasticity modulus of the material is 210000 Nmm^{-2}. The cross section of the trusses 1, 3, 5 and 7 is 50 mm^2 and for the trusses 2, 4, 6 and 8 it is 30 mm^2. The force F is 10.000 N, which is equivalent to the weight of a car. The assignment is to use the finite element method to calculate the nodal displacements and the forces in the trusses. The starting point has to be the $[B]$-matrix, which containes the derivatives of the local interpolation functions (equation 2.6 of the reader):

$$[B] = \frac{1}{L}[-1, 1] \tag{1.1}$$

Given this equation, the element stiffness matrix $[K_{el}]$ can be easily derived for the displacements and forces in the local coordinate system (equation 2.8 of the reader). The local element stiffness matrix has to be rewritten to the global coordinate system (equation 2.21 of the reader). Next, the element stiffness matrices need to be assembled in the global stiffness matrix. To solve the equilibrium equation $[\mathrm{K}]\{\mathrm{U}\} = \{\mathrm{F}\}$ the reduced stiffness matrix $[\mathrm{K}^{\mathrm{red}}]$ has to be selected corresponding to the set of unknown displacements. The equation can

be solved using the loading on the unknown degrees of freedom $\{F^{red}\}$. Solving the set of equations using MATLAB will give the nodal displacements. In §1.3 the steps needed to solve the truss problem are further elaborated. First an introduction to MATLAB is given in the following section.

1.2 Introduction to Matlab

MATLAB is an extensive program which is very suitable for structuring and automating any type of calculation. The MATLAB library includes a large amount of linear algebra operations, which makes it very suitable for applying the finite element method. MATLAB also offers a lot of options to graphically evaluate your results. Only a limited part of the MATLAB functionality will be used in this practicum. If you are an experienced MATLAB user it should be sufficient to scan this MATLAB introduction. However, if you are new to using MATLAB you should consider putting extra effort into learning MATLAB since it may make life a lot easier when executing any mathematical or engineering task during the rest of your study and your professional career.

After starting MATLAB from WINDOWS a screen similar to the following image will appear:

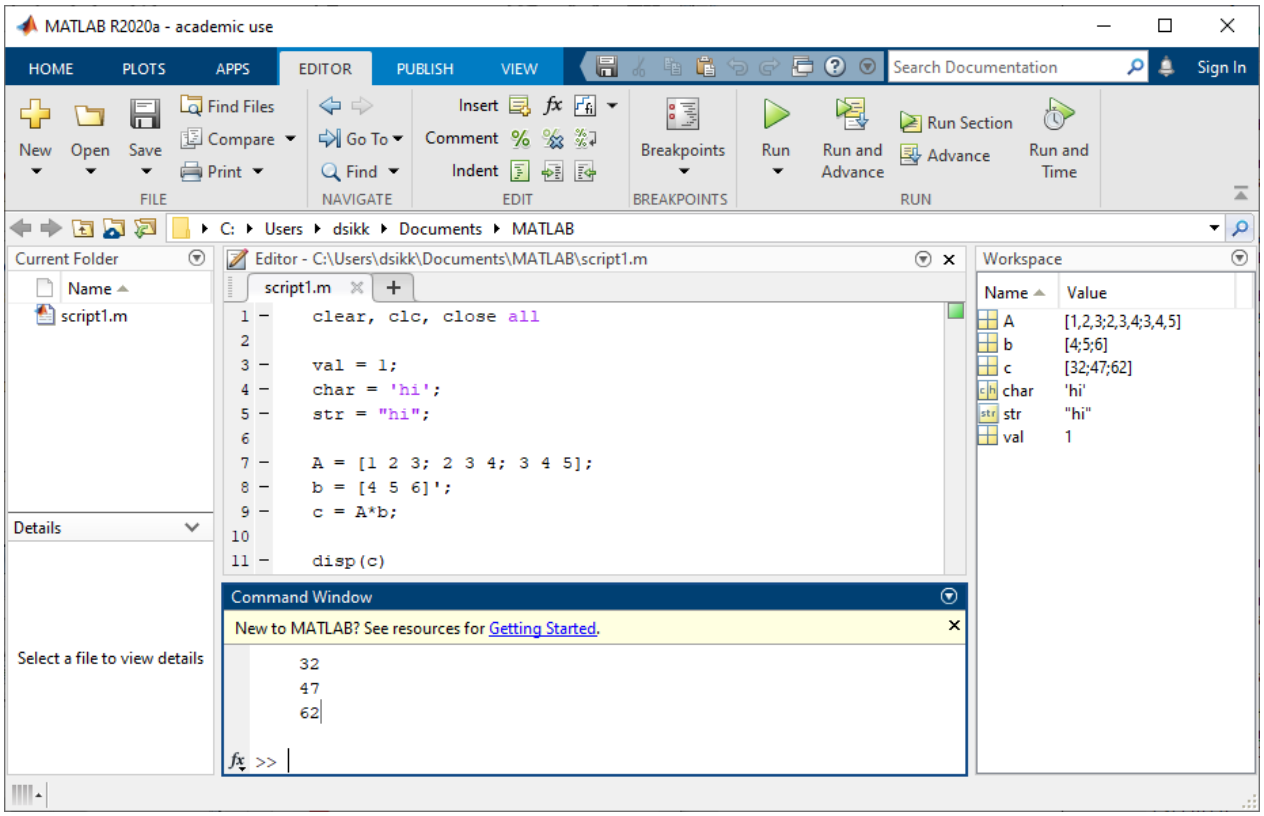

Figure 1.2: Matlab program

The exact layout may be different, depending on your personal settings. Different *Desktop tools* may be opened and closed in the *Desktop* menu. Make sure to have at least the *Com-*

mand Window, the *Workspace* and the *Editor* activated.

Using the *Command Window*, MATLAB can be used in an interactive way. This means that any command entered in the *Command Window* will be instantly executed. While doing this, you should notice that the state of the *Workspace* is directly updated after executing a command. For example, a matrix $[A]$ and a vector $\{b\}$ can be entered, following by the command `c = A*b`, which calculates and stores the vector $\{c\}$. However, if MATLAB is closed, the *Workspace* will be cleared and $[A]$, $\{b\}$ will $\{c\}$ will be lost. Because your result of the practicum has to be assessed by the practicum assistant *and* because it is unlikely that you don't make errors at your first attempt, it is wise to make a script with all calculations. This will help to easily find and correct errors in the calculation. In fact, writing a MATLAB script can be seen as writing a program in the MATLAB language.

In the following text it will be first explained how to enter MATLAB commands in the *Command Window*. Afterwards, it will be explained how to make scripts and functions using the *Editor*. A command can be entered in the *Command Window* behind the arrows `>>`(fig. 1.2). A command has to be closed with the 'Enter ↩' key. Spend a maximum of fifteen minutes watching MATLAB 's own introduction videos by typing `demo` in the command line. Remember that all MATLAB variables are *case sensitive*. This means that the uppercase `A` and the lowercase `a` are two different variables. MATLAB comes with good documentation of all functions. The *help*-function can be accessed through the menu.

Entering matrices and vectors

A matrix or vector can be entered manually. A matrix can be assigned to the variable `A` by entering a set of numbers between square brackets. Numbers within a row must be separated by comma's or spaces. Different rows must be separated with a semicolon. The matrix must be entered row-by-row. After pressing 'Enter ↩' the command is executed and the result is displayed:

```
>> A = [ 1,2,3;2,3,4;5,0,0 ]

A =

     1     2     3
     2     3     4
     5     0     0
```

Now MATLAB waits for the following command. A column vector is simply a $n \times 1$-matrix:

```
>> b = [ 5.2 ; 9.2 ; 2 ]

b =

  5.2000
  9.2000
  2.0000
```

A single element of a matrix can be changed with:

```
>> A(3,3) = 1

A =

     1     2     3
     2     3     4
     5     0     1
```

To initialize a matrix containing only zeros the command `zeros(m,n)` can be used for a matrix of size $m \times n$. For a matrix of size $n \times n$ the command `zeros(n)` can be used.

```
>> B = zeros(3)

B =

     0     0     0
     0     0     0
     0     0     0
```

The variable B can be cleared using the command `clear B`. The command `clear` will clear all the variables in the *Workspace*. A last way of defining a matrix is by selecting a part of an existing matrix. This must be used for the assembly of the system matrix from the element stiffness matrices. Just as `K(2,3)` gives element K_{23} from matrix $[K]$, `K1([1:3],[2:4,6])` gives the submatrix of $[K_1]$ from rows 1 to 3 and columns 2,3,4 and 6. For example:

```
K1 =

     1     2     3     4     5     6
     7     8     9     0     1     2
     3     4     5     6     7     8
     9     0     1     2     3     4


>> K1([1:3],[2:4,6])

ans =

     2     3     4     6
     8     9     0     2
     4     5     6     8
```

We can even select a specific set of rows and columns by defining the rows and columns in two separate variables. To select rows 1 and 3 and columns 1, 3 and 6 from matrix $[K_1]$, we can assign a variable (for example `row`) with the values 1 and 3 and another variable (for example `column`) with the numbers 1, 3 and 6. Then we can select the submatrix with `K1(row,column)`.

```
>> row = [1 3];
>> column = [1 3 6];
>> K1(row,column)

ans =

     1     3     6
     3     5     8
```

The same method can be used to assign values to a part of a matrix. The command `K1(row,column) = [1:3;4:6]` will assign the values 1 to 6 to the the submatrix defined with the vectors `row` and `column`. The remaining entries of `K1` remain unchanged. Note that MATLAB will give an error if the size of the matrices on both sides of the equal sign is different.

Matrix-manipulation

After defining matrices and vectors we can start with matrix-manipulation commands. We can calculate the product $[A]\{b\}$ using the command `A*b`. MATLAB will give the following result:

```
>> A*b

ans =

   29.6000
   46.0000
   28.0000
```

We can also assign the answer to a new variable, for example to a newly created vector $\{c\}$:

```
>> c = A*b;
```

Displaying the output of the command can be suppressed by adding a semicolon to the end of the command. This is useful within scripts and functions since you can select which (intermediate) results you want to see in the *Command Window*. The value of $\{c\}$ can be seen by double-clicking on the variable `c` in the *Workspace* or by entering:

```
>> c

c =

   29.6000
   46.0000
   28.0000
```

The transposed of a variable can be found by placing an accent behind the name of the variable. A premultiplication of matrix $[A]$ with vector $\{b\}$ can be calculated with:

```
>> b'*A

ans =

   33.6000   38.0000   54.4000
```

To calculate the inverse of a matrix the function `inv` can be used. This could be used to find the solution of $[A]\{x\} = \{c\}$. However, this function is computationally inefficient. It is better to use the *Left division*: '\', especially for large matrices.

```
>> x = A\c

x =

   5.2000
   9.2000
   2.0000
```

This command solves the equation with Gaussian elimination. You can see that the vector $\{b\}$ is retrieved.

Scripts and functions

A new script file can be opened with the menu-option `File>New>Script`. A blank document will be opened in the *Editor*. The stiffness matrix of element (1) from the example of figure 1.1 can be calculated with the equation $[K] = \int_V [B]^{\mathrm{T}}[D][B]\,\mathrm{d}V = AL[B]^{\mathrm{T}}[D][B]$. This can be entered in the script file with the following commands:

```
A = 50;
L = 500;
D = 210000;
B = 1/L * [-1,1];
K = A*L*B'*D*B;
```

Notice that the commands are not automatically executed. First save the file with extension `.m` (for example `kmat.m`) on your computer. Then change the MATLAB working directory from the default MATLAB working directory to the location of your script, for example:

```
>> cd p:\workingdirectory
```

Any script in the working directory can be called by entering the filename without extension `.m` in the *Command Window*. Note that you should save the script before running it. Another way of running the script is clicking on the `Save and run` button in the toolbar of the *Editor* or pressing `F5`.
Running the script does not display any results because all commands in `kmat.m` end with a semicolon. Entering `K` in the *Command Window* displays the resulting stiffness matrix:

```
>> kmat
>> K

K =

  21000  -21000
 -21000   21000
```

MATLAB files with extension `.m` are called m-files. A m-file can also be used to construct a function. The filename must be equal to the function name. The first line of the function m-file must start with `function`, followed by the output of the function between square brackets, followed by an equal sign and thereafter the function name and the input of the function between brackets. A function can have multiple inputs and outputs, separated by commas. The following example gives the rotation matrix as a function of the orientation of a truss (see equation 2.12 of the reader). Write the following code in the file `rotationmat.m`:

```
function [R] = rotationmat(x,y)
% Determine the rotationmatrix for a truss element under
% an angle, using coordinates (x,y) seen from (0,0).

length = sqrt(x^2 + y^2);
sinphi = y / length;
cosphi = x / length;

R = [ cosphi, sinphi,      0,      0;
           0,      0, cosphi, sinphi ];
```

Evaluate the function in the *Command Window*:

```
>> rotationmat(1,1)

ans =
    0.7071    0.7071         0         0
         0         0    0.7071    0.7071
```

In this example `^2` is used to calculate x to the power of 2 and `sqrt()` gives the square root. Entering a percentage sign at the beginning of a line tells MATLAB to skip the line. Hence, this can be used to add comments to the code. The command `rotationmat(1,1)` executes the function and gives the rotation matrix for a truss under 45°.

1.3 Truss framework calculation

Now we have enough MATLAB knowledge to be able to solve the truss framework problem. First a function has to be written to calculate the local stiffness matrix for each element as a function of the coordinates of the element.

```
function [K] = k_local( E, A, Ni, Nj );
% E = Elasticity modulus
% A = Cross-section area
% Ni, Nj = coordinates node i and node j [x,y]

L = sqrt( (Nj(1)-Ni(1))^2 + (Nj(2)-Ni(2))^2 );  % length element
B = 1/L * [-1,1];
K = A*L* B'*E*B;
```

To simplify the use of the function `rotationmat` from the previous paragraph, the function could be changed to be a function of `Ni,Nj` instead of `x,y`. Consider changing the function for the practicum to generalize the use of the function.
Make a new script file `trussframework.m` with the definition of the model. The node coordinates are vectors of two entries, the elasticity modulus is a scalar and both cross sections have a scalar value:

```
N1 = [0,0];
N2 = [0,500];
N3 = [500,500];
N4 = [1000,0];
N5 = [1500,500];
N6 = [2000,0];
E = 210000;
Ahigh = 50;
Alow = 30;
```

Then the local stiffness matrix of all elements can be calculated using the function `k_local`:

```
K1_loc = k_local(E,Ahigh,N2,N3);
K2_loc = k_local(E,Alow,N1,N3);
K3_loc = k_local(E,Ahigh,N1,N4);
K4_loc = ...
```

$$
[K^{syst}] = \begin{bmatrix}
K_{11}^2 + K_{11}^3 & K_{12}^2 & 0 & 0 & K_{13}^2 & K_{14}^2 & K_{12}^3 & 0 & 0 & 0 & 0 \\
K_{21}^2 & K_{22}^2 & 0 & 0 & K_{23}^2 & K_{24}^2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & K_{11}^1 & 0 & K_{12}^1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
K_{31}^2 & K_{32}^2 & K_{21}^1 & 0 & K_{22}^1 + K_{33}^2 + K_{11}^4 + K_{11}^5 & K_{34}^2 + K_{12}^4 & K_{13}^4 & K_{14}^4 & K_{12}^5 & 0 & 0 \\
K_{41}^2 & K_{42}^2 & 0 & 0 & K_{43}^2 + K_{21}^4 & K_{44}^2 + K_{22}^4 & K_{23}^4 & K_{24}^4 & 0 & 0 & 0 \\
K_{21}^3 & 0 & 0 & 0 & K_{31}^4 & K_{32}^4 & K_{22}^3 + K_{33}^4 + K_{11}^6 + K_{11}^7 & K_{34}^4 + K_{12}^6 & K_{13}^6 & K_{14}^6 & K_{12}^7 \\
0 & 0 & 0 & 0 & K_{41}^4 & K_{42}^4 & K_{43}^4 + K_{21}^6 & K_{44}^4 + K_{22}^6 & K_{23}^6 & K_{24}^6 & 0 \\
0 & 0 & 0 & 0 & K_{21}^5 & 0 & K_{31}^6 & K_{32}^6 & K_{22}^5 + K_{33}^6 + K_{11}^8 & K_{34}^6 + K_{12}^8 & K_{13}^8 & K_{14}^8 \\
0 & 0 & 0 & 0 & 0 & 0 & K_{41}^6 & K_{42}^6 & K_{43}^6 + K_{21}^8 & K_{44}^6 + K_{22}^8 & K_{23}^8 & K_{24}^8 \\
0 & 0 & 0 & 0 & 0 & 0 & K_{22}^7 & 0 & K_{31}^8 & K_{32}^8 & K_{22}^7 + K_{33}^8 & K_{34}^8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & K_{41}^8 & K_{42}^8 & K_{43}^8 & K_{44}^8
\end{bmatrix}
$$

Figure 1.3: Global stiffness matrix

The local stiffness matrix is based on the local element coordinate system. However, the forces and displacements need to be calculated in the global x- and y-coordinates. To determine the element stiffness in the global coordinate system a rotation matrix for the inclined trusses has to be defined. Hence, we define two rotation matrices `T_up` and `T_down` for the elements inclinded respectively upwards and downwards (from left to right):

```
T_up = rotationmat(500,500);
T_down = rotationmat(500,-500);
K2_glob = T_up'*K2_loc*T_up;
K4_glob = T_down'*K4_loc*T_down;
K6_glob = T_up'*K6_loc*T_up;
K8_glob = T_down'*K8_loc*T_down;
```

The elements which already are oriented in the x-direction are not rotated. For a more complex framework with more element orientations is should be considered to construct the rotation matrix for every element.

Now the element stiffness matrices have to be assembled in the global system stiffness matrix. Therefore it must be defined on which degrees of freedom of the global system the local stiffness matrices work. Of the total of 12 degrees of freedom (6 nodes times 2 directions), 4 are suppressed. Hence, a system of 8 equations with 8 unknowns will remain. To be able to calculate the reaction forces on the suppressed degrees of freedom, it is useful to construct the full system matrix.

In figure 1.3 the full system matrix is shown. Starting point are the local stiffness matrices K^1 to K^8, being a 2×2 matrix for the odd elements and a 4×4 matrix for the even elements. The degrees of freedom are numbered 1 to 12, starting with the x-displacement of node 1, followed by the y-displacement of node 1, followed by the x- and then the y-displacement of node 2, etc.

The system which has to be solved is the global stiffness matrix reduced with the rows and columns which correspond to the suppressed degrees of freedom. Hence, the stiffness matrix must be reduced with the first 4 columns and with the first 4 rows (the part which is not highlighted in figure 1.3 remains). In row 4 all values are 0, so the reaction force R_y for this degree of freedom will automatically be zero. This can be easily seen in figure 1.1.

A convinient and systematic way of building the full system matrix in MATLAB is by initializing the matrix with zeros and filling the matrix with the local stiffness matrices. To do so, 8 location-vectors (L1 to L8) are defined, which define for each element on which degrees of freedom it acts. For example, the first element acts on the x-direction of node 2 and on the x-direction of node 3. This corresponds to the degrees of freedom 3 and 5. The second element acts on the x- and y- displacements of the nodes 1 and 3, corresponding to the degrees of freedom 1, 2, 5 and 6. Hence, we get the following code for all elements:

```
L1 = [3,5];
L2 = [1,2,5,6];
L3 = [1,7];
L4 = [5,6,7,8];
L5 = [5,9];
L6 = [7,8,9,10];
L7 = [7,11];
L8 = [9,10,11,12];
%
K = zeros(12);
K(L1,L1) = K(L1,L1) + K1_loc;
K(L2,L2) = K(L2,L2) + K2_glob;
K(L3,L3) = K(L3,L3) + K3_loc;
K(L4,L4) = K(L4,L4) + K4_glob;
K(L5,L5) = K(L5,L5) + K5_loc;
K(L6,L6) = K(L6,L6) + K6_glob;
K(L7,L7) = K(L7,L7) + K7_loc;
K(L8,L8) = K(L8,L8) + K8_glob;
```

In essence every command adds stiffness between the degrees of freedom. Try to execute the commands above in the *Command Window* without the semicolons to see how the system matrix is gradually built. The system of equations that has to be solved is defined by removing the first 4 rows and 4 columns of the system matrix. In MATLAB the reduced stiffness matrix can be easily made by selecting the *range* of the indexes in `K`:

```
K_red = K(5:12,5:12);
```

After defining the reduced stiffness matrix, the load vector $\{F\}$ has to be defined. In this case, the force acts on the y-direction of node 6. This corresponds to the degree of freedom number 12. Notice that degree of freedom 12 moved to position 8 in the reduced stiffness matrix. The displacements can be determined by solving the equations using the reduced system matrix and the load vector. The total displacement vector can be made with a vector with zeros on positions 1 to 4 (the constrained degrees of freedom) and the reduced displacement vector on the positions 5 to 12:

```
F = zeros(8,1);
F(8,1) = -10000;
u_red = K_red\F

u_total(1:4,1) = [0;0;0;0];
u_total(5:12,1) = u_red;
```

The results are shown in §1.5. To determine the force in truss number 6 the displacements in the direction of the truss are needed. This can be easily found with the rotation matrix, the location vector and the local stiffness matrix, since the following holds: $\{u_{loc}\} = [T]\{u_{glob}\}$ and $\{F_{el}\} = [K_{loc}]\{u_{loc}\}$. The global displacements for any element can be easily selected

from the total displacement vector using the location vector. A more direct way of calculating the truss forcing is by using the following equations: $\varepsilon = [B]\{u\}$ en $F = \sigma A = EA[B]\{u\}$.

The reaction forces can be determined using the first 4 rows of the system matrix (fig. 1.3). Multypling the total system matrix with the total displacement vector gives the resulting nodal forces. Hence, the load on node 6 has to be recognized, and the reaction forces on nodes 1 and 2 are found.

```
F6 = K6_loc * T_up * u_total(L6);

R = K*u_total;
```

Now all results have been found, you should check the results to ensure that they are correct. For example, check whether the elongation of a truss and the truss force correspond. Another check is to look whether the total load on the system (external load on node 6 plus reaction forces on node 1 and 2) correspond with the laws of equilibrium of forces and moments.

1.4 Visualization

MATLAB has a large number of plotting functions. Consider plotting the results in MATLAB to visualize them. Visualization of results is usually very helpful to detect errors in the calculations. The command `figure` opens a figure window. The command `hold on` prevents MATLAB of overwriting the plot every time the `plot` function is called. The command `axis equal` sets an equal scaling to the x- and y-direction of the plot. The command `plot(x,y)` connects the points given by the set of `x` and `y` coordinates. Run the following script to make a plot of the initial framework:

```
figure
hold on
plot([N2(1),N3(1)],[N2(2),N3(2)])
plot([N1(1),N3(1)],[N1(2),N3(2)])
plot([N1(1),N4(1)],[N1(2),N4(2)])
plot([N3(1),N4(1)],[N3(2),N4(2)])
plot([N3(1),N5(1)],[N3(2),N5(2)])
plot([N4(1),N5(1)],[N4(2),N5(2)])
plot([N4(1),N6(1)],[N4(2),N6(2)])
plot([N5(1),N6(1)],[N5(2),N6(2)])
axis equal
```

Now plot the final shape of the framework. To visualize the small displacements we add an exaggeration factor `pl`, otherwise the displacements would be hardly visible to the eye:

```
pl = 20;
figure
hold on
plot([N2(1),N3(1) + pl*u_total(5)],[N2(2),N3(2) + pl*u_total(6)])
```

```
plot([N1(1),N3(1) + pl*u_total(5)],[N1(2),N3(2) + pl*u_total(6)])
plot([N1(1),N4(1) + pl*u_total(7)],[N1(2),N4(2) + pl*u_total(8)])
plot([N3(1) + pl*u_total(5),N4(1) + pl*u_total(7)],...
[N3(2) + pl*u_total(6),N4(2) + pl*u_total(8)])
plot([N3(1) + pl*u_total(5),N5(1) + pl*u_total(9)],...
[N3(2) + pl*u_total(6),N5(2) + pl*u_total(10)])
plot([N4(1) + pl*u_total(7),N5(1) + pl*u_total(9)],...
[N4(2) + pl*u_total(8),N5(2) + pl*u_total(10)])
plot([N4(1) + pl*u_total(7),N6(1) + pl*u_total(11)],...
[N4(2) + pl*u_total(8),N6(2) + pl*u_total(12)])
plot([N5(1) + pl*u_total(9),N6(1) + pl*u_total(11)],...
[N5(2) + pl*u_total(10),N6(2) + pl*u_total(12)])
axis equal
```

Notice that the code for plotting a simple framework is already quite large. Hence, it would be helpful to construct a `for` loop to automatically plot all elements one-by-one. Type `help for` and try to simplify the code. Enter all coordinate variables N1 to N6 in one matrix and construct a matrix that stores which nodes are connected by each element to do so.

The final shape of the framework is shown in figure 1.4. Try changing the loading of the framework or the cross-section of the trusses to see how it affects the results. Notice the benefit of working with script files. Different loadcases are easily evaluated by changing a few values and running the script again.

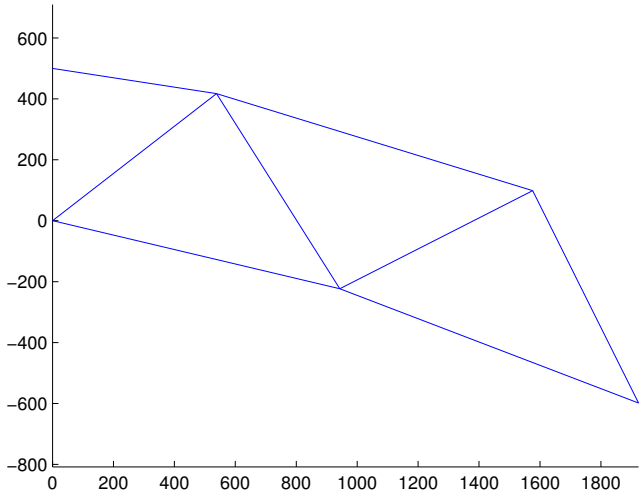

Figure 1.4: Figure of framework displacements, exaggerated 20 times

1.5 Results

The displacements, reaction forces ans some truss forces calculated with MATLAB are shown below:

```
>> vakwerk
>> u_red

u_red =
    1.9048
   -4.1495
   -2.8571
  -11.1562
    3.8095
  -20.0677
   -3.8095
  -29.9315

>> R

R =
  1.0e+004 *

    4.0000
    1.0000
   -4.0000
         0
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
   -1.0000

>> F7

F7 =
  1.0e+004 *

    1.0000
   -1.0000

>> F8

F8 =
  1.0e+004 *

   -1.4142
    1.4142
```