University of California, Los Angeles

MAE 168A – Introduction to Finite Element Methods – Spring 2015

MALAB Truss Solver

Huy Dinh

Instructor: Dr. Vinay Goyal
Teaching Assistant: Linus Lee

Overview

This MATLAB code takes the input from a standard Abaqus input file of a 2D truss structure, solves for the nodes' displacements and elements' axial stresses. Post-processing includes calculating the maximum tensile and compressive stress in the structure, identifying the elements under those stresses, and producing plots that give a clear visual understanding of the structure. The project's code is broken into many smaller methods that allow extending the project to include other elements, or to be reused by other projects.

Code overview

1. High level description

The code contains three main functions:

- main.m
- truss_read
- truss_solve
- truss_plot

The only input to function main is the name of the .input file (.inp extension) in Abaqus standard format. Function main will call function truss_read to scan the input file and get all necessary information about the structure. Function main will then call truss_solve to solve for all displacements and reaction forces at all nodes.

2. Input file requirements

There are a few requirements for the input files asides from those imposed by Abaqus standard:

- The only cards that will be read are: node, nset, element, elset, solid section, elastic, boundary, cload, transform. Anything other than those will be ignored.
- The required cards are: node, element, solid section, elastic, boundary, cload.
- Optional cards are: transform, elset, nset.
- Once an elset name is defined with certain elements, no other element can be added to that elset.
- Each node can only be transformed once. Redefining a node transformation will produce wrong results.

3. Outputs

At the post-processing step, the program will output the highest compressive and tensile stress in the structure, and the element number that they occur. The function truss_plot is then called by main to plot the results into 3 figures:

- Figure 1 shows the structures with numbered nodes and elements so that the user can easily spot out a geometry mistake in the input file.
- Figure 2 shows the structure in the undeformed state and the deformed state (with a displacement scaling factor) so that the user has a general sense about where things move.
- Figure 3 shows the fringe plot of the stresses of the structure in the deformed state.

Examples

1. Example 1:

This is a crane structure grounded at the two legs. There is a downward force acting on node 15, an upward for acting on node 11, and a leftward force acting on node 25.

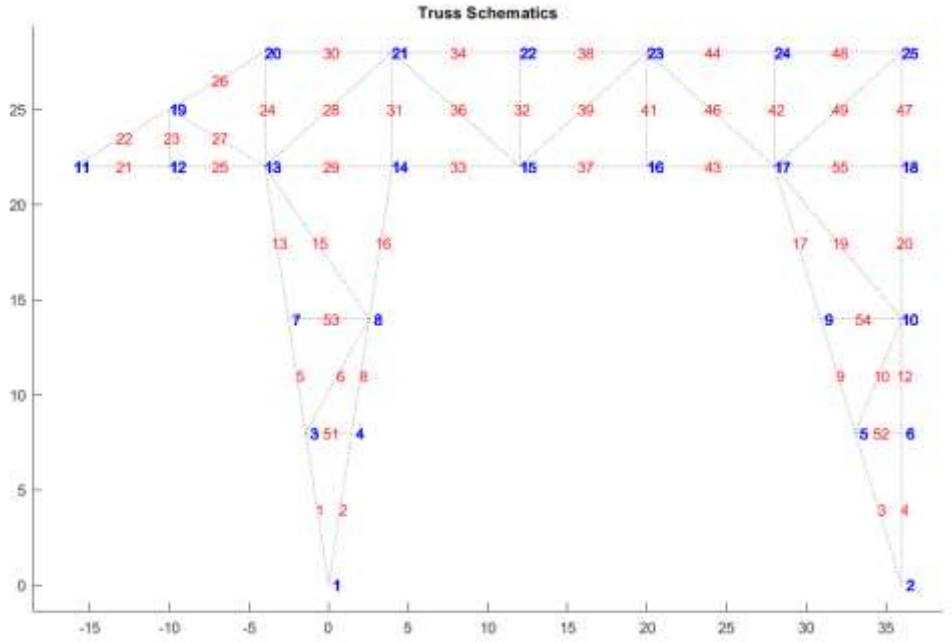


Figure 1: Schematics of nodes and elements

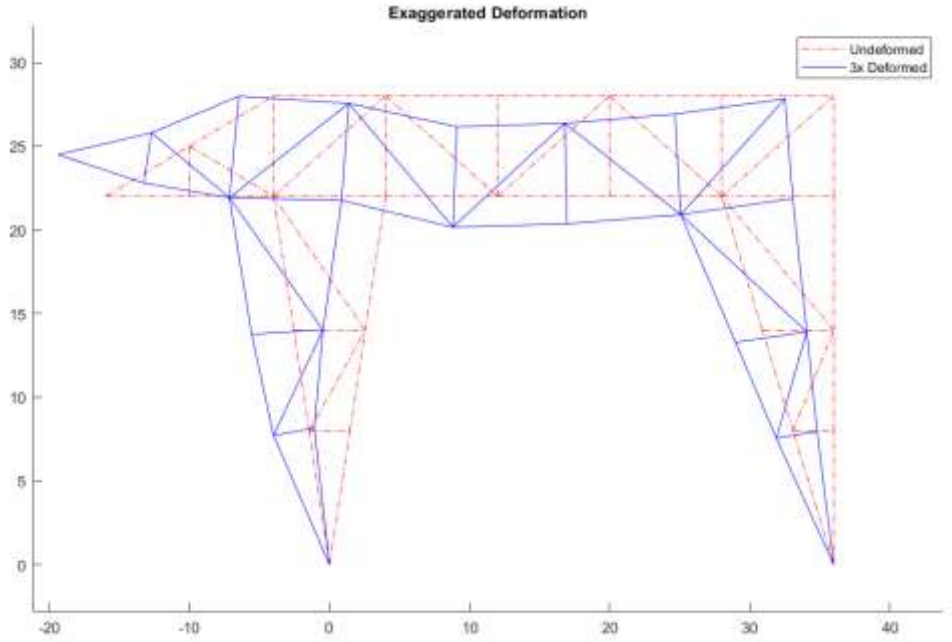


Figure 2: Undeformed vs. Exaggerated Deformed States

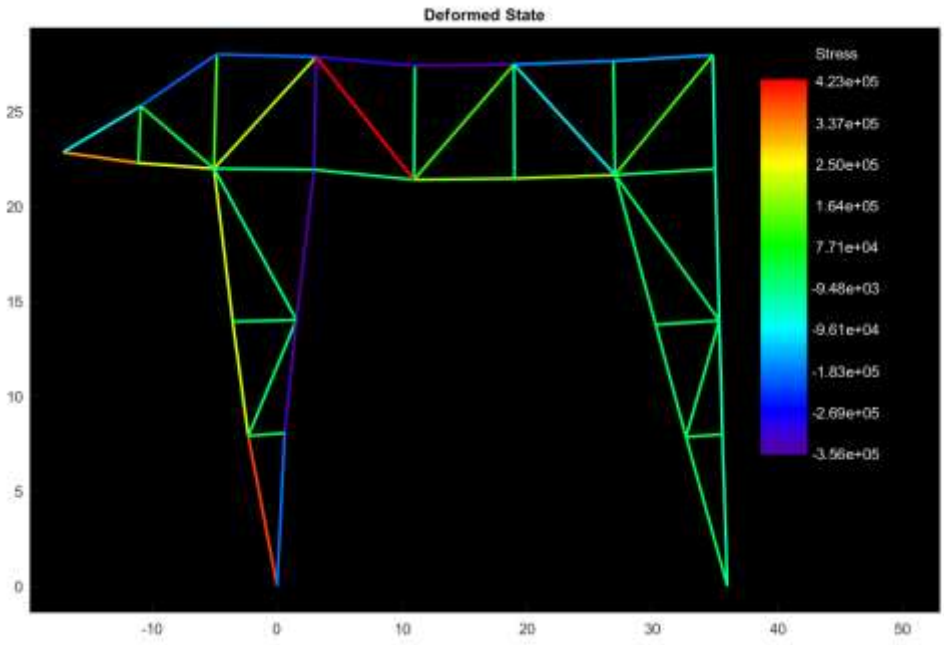


Figure 3: Stress distribution in the structure

The output to the command line for this case

```
Max compression stress : element number 28, magnitude = -6.46076e+03
Max tension stress     : element number 10, magnitude = 3.58623e+03
```

2. Example 2

This is a symmetric model of a truss bridge. There is a downward force at every node at the bottom. Node 8 and 9, due to the symmetry requirement, cannot move in the x-direction, but are free to move vertically.

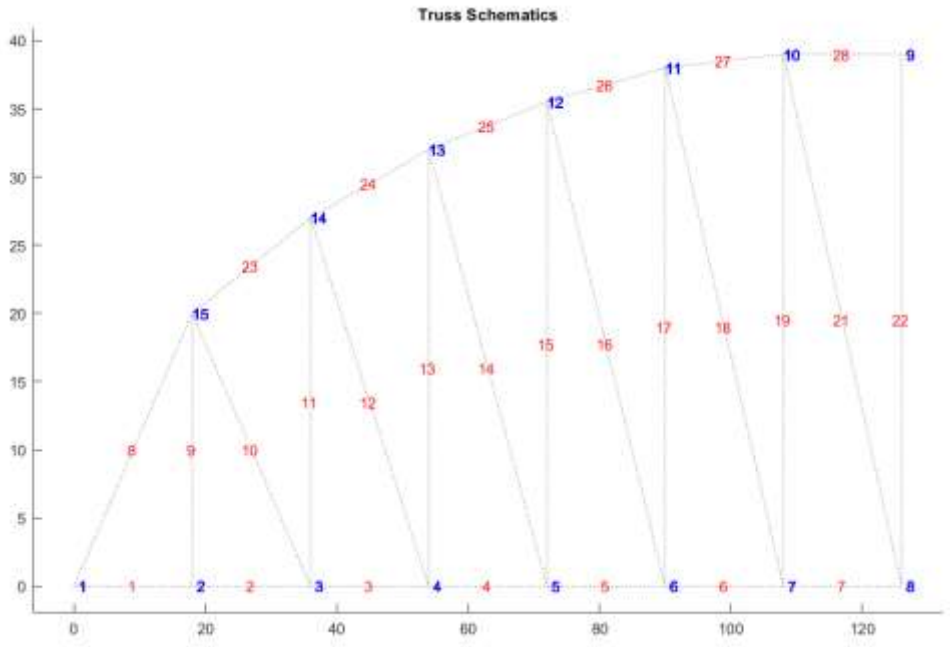


Figure 4: Schematics of nodes and elements

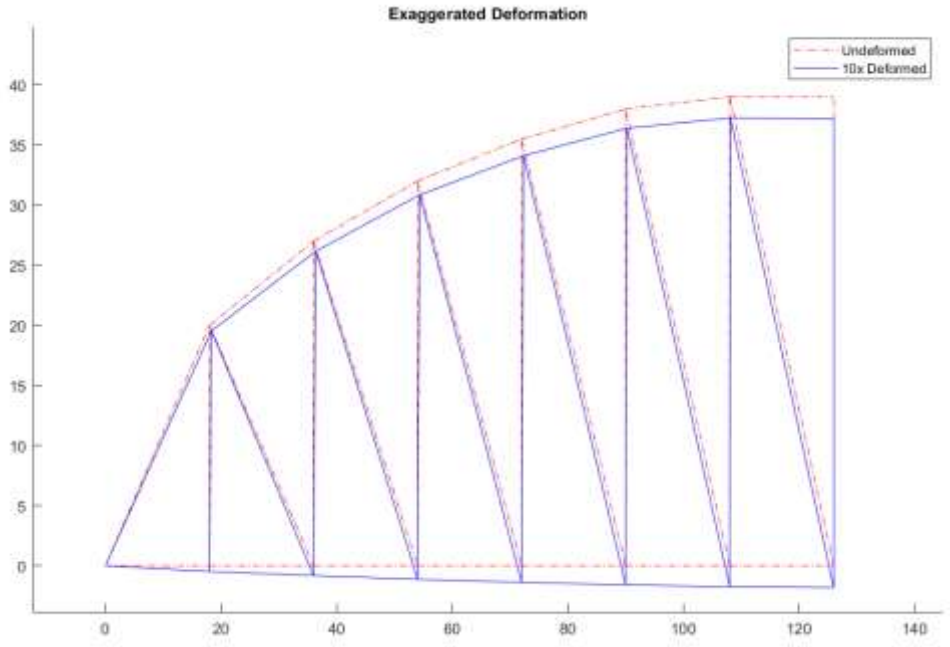
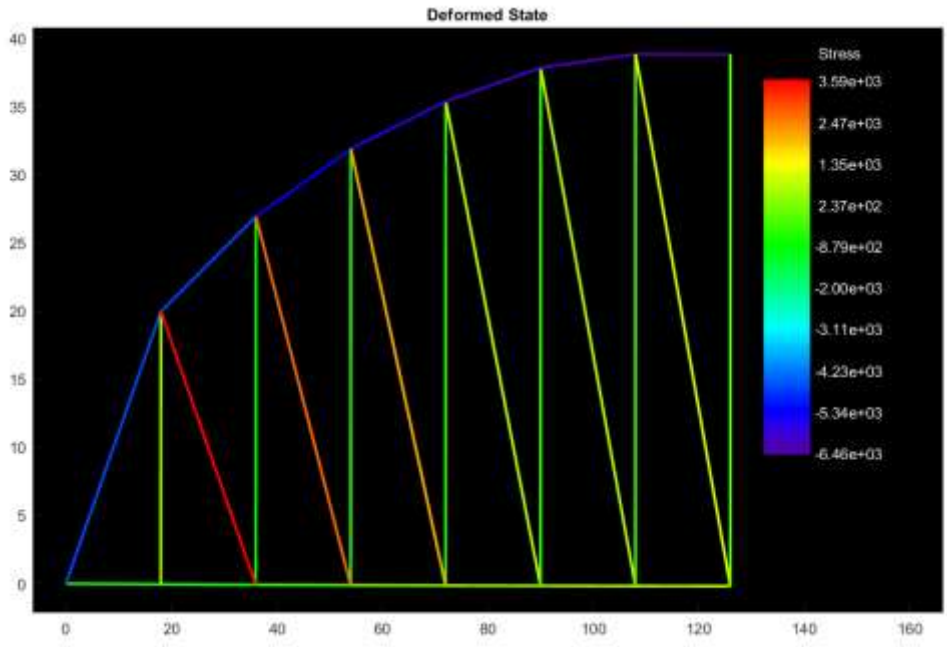


Figure 5: Undeformed vs. Exaggerated Deformed States



The output to the command line for this case:

```
Max compression stress : element number 28, magnitude = -6.46076e+03
Max tension stress     : element number 10, magnitude = 3.58623e+03
```

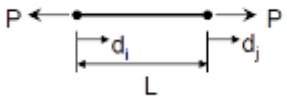
Truss theory

This section briefly walks through the derivation of necessary calculation steps taken behind the scene to produce the results shown above. For a full derivation, see reference [1].

1. Element formulation

A truss consists of typically straight two-force members (referred to as *elements* from now on) connected at *nodes*, which are modeled as frictionless pin joints. For this reason, no moments are present in the structure, so all elements are under either axial compression or tension.

Each element is modeled as a beam with Young's modulus E, length L, cross-sectional area A. The displacements at each node is d_i and d_j. At equilibrium:



Using Hooke's Law ε = Eσ and some simplification:

$$P = \frac{EA}{L}(d_j - d_i)$$

Let k = EA/L and denote the force at node i and j by f_i and f_j respectively, we can obtain:

$$\begin{Bmatrix} f_i \\ f_j \end{Bmatrix} = k \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} d_i \\ d_j \end{Bmatrix}$$

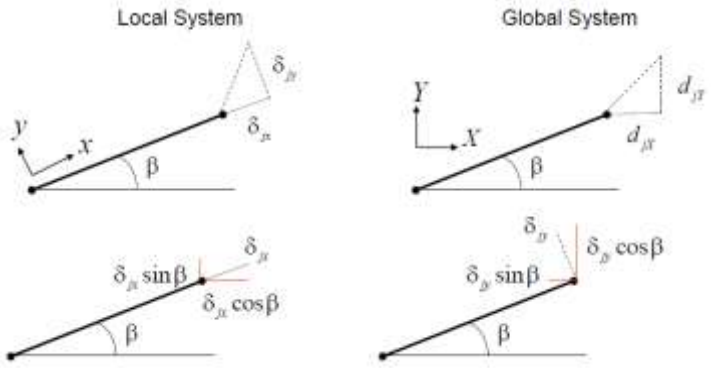
2. Element transformation

For elements with different orientations in the same structure, expressing the force and displacement along axial directions prevents assembling multiple elements into a general structure. Therefore, we need to transform the force and displacement along the element's axis into the forces and displacements in a *local* coordinate, and then transform those into a unified global coordinate. Note that this is different from *nodal coordinates*, which will be used to transform boundary conditions the the nodes.

First step is to transform the force and displacement along the main axis into the local coordinate. For a general element oriented at an angle β with respect to the global coordinate XY, their displacements at each end i and j in the global system are as follows:

$$at\ i\text{:}\ \begin{cases} d_{iX} = \delta_{ix} cos\beta - \delta_{iy} sin\beta \\ d_{iY} = \delta_{ix} sin\beta + \delta_{iy} cos\beta \end{cases}$$

$$at\ j\text{:}\ \begin{cases} d_{jX} = \delta_{jx} cos\beta - \delta_{jy} sin\beta \\ d_{jY} = \delta_{jx} sin\beta + \delta_{jy} cos\beta \end{cases}$$



We can assemble the four expressions above into a single matrix equation:

$$\boldsymbol{d} = \boldsymbol{T}\,\boldsymbol{\delta}$$

$$\begin{Bmatrix} d_{iX} \\ d_{iY} \\ d_{jX} \\ d_{jY} \end{Bmatrix} = \begin{bmatrix} cos\beta & -sin\beta & 0 & 0 \\ sin\beta & cos\beta & 0 & 0 \\ 0 & 0 & cos\beta & -sin\beta \\ 0 & 0 & sin\beta & cos\beta \end{bmatrix} \begin{Bmatrix} \delta_{ix} \\ \delta_{iy} \\ \delta_{jx} \\ \delta_{jy} \end{Bmatrix}$$

Use the same geometry transformation for the forces:

$$\boldsymbol{F} = \boldsymbol{T}\,\boldsymbol{f}$$

$$\begin{Bmatrix} F_{iX} \\ F_{iY} \\ F_{jX} \\ F_{jY} \end{Bmatrix} = \begin{bmatrix} cos\beta & -sin\beta & 0 & 0 \\ sin\beta & cos\beta & 0 & 0 \\ 0 & 0 & cos\beta & -sin\beta \\ 0 & 0 & sin\beta & cos\beta \end{bmatrix} \begin{Bmatrix} f_{ix} \\ f_{iy} \\ f_{jx} \\ f_{jy} \end{Bmatrix}$$

Note the orthogonality property of the transformation matrix **T**:

$$\boldsymbol{T}^{-1} = \boldsymbol{T}^{T}$$

In the local coordinate system, Hooke's law can be expressed simply as:

$$\begin{matrix} \boldsymbol{f} \\ \begin{Bmatrix} f_{ix} \\ f_{iy} \\ f_{jx} \\ f_{jy} \end{Bmatrix} \end{matrix} = \begin{matrix} \boldsymbol{k'^{(e)}} \\ \begin{bmatrix} k & 0 & -k & 0 \\ 0 & 0 & 0 & 0 \\ -k & 0 & k & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \begin{matrix} \boldsymbol{\delta} \\ \begin{Bmatrix} \delta_{ix} \\ \delta_{iy} \\ \delta_{jx} \\ \delta_{jy} \end{Bmatrix} \end{matrix}$$

Substituting the expression of **δ** and **f**:

$$\boldsymbol{T}^{-1}\boldsymbol{F} = \boldsymbol{k'^{(e)}}\boldsymbol{T}^{-1}\,\boldsymbol{d}$$

Left-multiply both sides with **T** and use the orthogonality identity $\mathbf{T}^{-1} = \mathbf{T}^{T}$:

$$\boldsymbol{F} = \left[\boldsymbol{T}\,\boldsymbol{k'^{(e)}}\,\boldsymbol{T}^{T}\right]\boldsymbol{d}$$

Let

$$\boldsymbol{k} = \boldsymbol{T}\,\boldsymbol{k'^{(e)}}\boldsymbol{T}^{T}$$

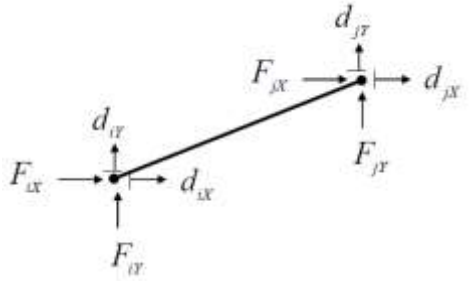
The overall stiffness matrix of the element is therefore:

$$\boldsymbol{k}^{(e)} = k\begin{bmatrix} c^2 & sc & -c^2 & -sc \\ & s^2 & -sc & -s^2 \\ & & c^2 & sc \\ symmetry & & & s^2 \end{bmatrix}$$

where c = cosβ and s = sinβ.

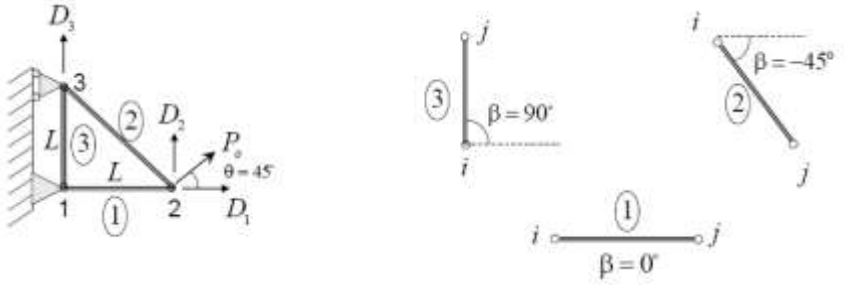
3. Element Assembly

Since each node has two degrees of freedom (DOFs), every element will correspond to four DOFS, or *code numbers*.



$$\mathbf{d}^{(e)} = \begin{Bmatrix} d_{iX} \\ d_{iY} \\ d_{jX} \\ d_{jY} \end{Bmatrix} \begin{matrix} iX \\ iY \\ jX \\ jY \end{matrix} \qquad \mathbf{k}^{(e)} = \begin{matrix} iX & iY & jX & jY \\ \end{matrix} \begin{bmatrix} kc^2 & ksc & -kc^2 & -ksc \\ & ks^2 & -ksc & -ks^2 \\ & & kc^2 & ksc \\ symm & & & ks^2 \end{bmatrix} \begin{matrix} iX \\ iY \\ jX \\ jY \end{matrix}$$

We can use this, and the fact that DOFs are shared and applied forces are additive, to "add" or assemble the complete system of equations with all of our variables of interests. For example, given the structure of three elements below, we can assign D_1, D_2, D_3 to be the three unknowns of interests, and D_0 to be grounded nodes:



$$\mathbf{k}^{(1)} = \begin{array}{c} \begin{matrix} 0 & 0 & 1 & 2 \end{matrix} \\ \begin{bmatrix} k_{11}^{(1)} & k_{12}^{(1)} & k_{13}^{(1)} & k_{14}^{(1)} \\ k_{21}^{(1)} & k_{22}^{(1)} & k_{23}^{(1)} & k_{24}^{(1)} \\ k_{31}^{(1)} & k_{32}^{(1)} & k_{33}^{(1)} & k_{34}^{(1)} \\ k_{41}^{(1)} & k_{42}^{(1)} & k_{43}^{(1)} & k_{44}^{(1)} \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 1 \\ 2 \end{matrix} \end{array} \quad \mathbf{k}^{(2)} = \begin{array}{c} \begin{matrix} 0 & 3 & 1 & 2 \end{matrix} \\ \begin{bmatrix} k_{11}^{(2)} & k_{12}^{(2)} & k_{13}^{(2)} & k_{14}^{(2)} \\ k_{21}^{(2)} & k_{22}^{(2)} & k_{23}^{(2)} & k_{24}^{(2)} \\ k_{31}^{(2)} & k_{32}^{(2)} & k_{33}^{(2)} & k_{34}^{(2)} \\ k_{41}^{(2)} & k_{42}^{(2)} & k_{43}^{(2)} & k_{44}^{(2)} \end{bmatrix} \begin{matrix} 0 \\ 3 \\ 1 \\ 2 \end{matrix} \end{array} \quad \mathbf{k}^{(3)} = \begin{array}{c} \begin{matrix} 0 & 0 & 0 & 3 \end{matrix} \\ \begin{bmatrix} k_{11}^{(3)} & k_{12}^{(3)} & k_{13}^{(3)} & k_{14}^{(3)} \\ k_{21}^{(3)} & k_{22}^{(3)} & k_{23}^{(3)} & k_{24}^{(3)} \\ k_{31}^{(3)} & k_{32}^{(3)} & k_{33}^{(3)} & k_{34}^{(3)} \\ k_{41}^{(3)} & k_{42}^{(3)} & k_{43}^{(3)} & k_{44}^{(3)} \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 3 \end{matrix} \end{array}$$

And the global stiffness matrix will be:

$$\mathbf{K} = \begin{array}{c} \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} k_{33}^{(1)} + k_{33}^{(2)} & k_{34}^{(1)} + k_{34}^{(2)} & k_{32}^{(2)} \\ & k_{44}^{(1)} + k_{44}^{(2)} & k_{24}^{(2)} \\ symm. & & k_{22}^{(2)} + k_{44}^{(3)} \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \end{array}$$

For each DOF, either the force or displacement is prescribed. Once we have the assembly form of

$$\boldsymbol{K}\,\boldsymbol{D} \;=\; \boldsymbol{F}$$

Rearranging the rows and columns of the system so that we get this form

$$\begin{bmatrix} \boldsymbol{K}_{\overline{u}\overline{u}} & \boldsymbol{K}_{\overline{u}u} \\ \boldsymbol{K}_{u\overline{u}} & \boldsymbol{K}_{uu} \end{bmatrix} \begin{Bmatrix} \overline{\boldsymbol{u}} \\ \boldsymbol{u} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{Q} \\ \overline{\boldsymbol{Q}} \end{Bmatrix}$$

where $\overline{\boldsymbol{u}}$ and $\overline{\boldsymbol{Q}}$ denote the set of unknowns within **u** and **Q**. The complete displacement vector can be calculated explicitly using

$$\boldsymbol{u} = \boldsymbol{K}_{uu}^{-1}\,(\overline{\boldsymbol{Q}} - \boldsymbol{K}_{u\overline{u}}\,\overline{\boldsymbol{u}})$$

Reaction force vector **Q**, although of less importance, can also be found via:

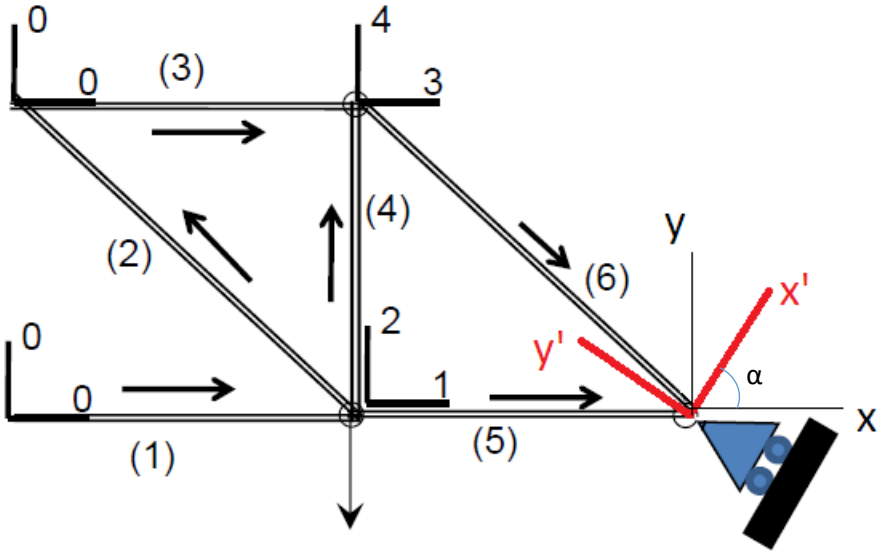
$$\boldsymbol{Q} = \boldsymbol{K}_{\overline{u}\overline{u}}\,\overline{\boldsymbol{u}} + \boldsymbol{K}_{\overline{u}u}\,\boldsymbol{u}$$

With all displacements found, we can find the new locations of the nodes, and thus the lengths of each element. Use Hooke's law once again to find the stresses

$$\sigma = E\epsilon = E\frac{L - L_o}{L_o}$$

4. Nodal transformation

There are cases where it is convenient to transform the local coordinate system of a node at some angle. For example, if one of the nodes is on an inclined roller support at an angle α i.e. the node can only translate at α degrees with respect to the x-axis, rotating the Oxy coordinate α degrees into Ox'y' will allow specifying the y'-displacement to be 0. Although being a convenience with hand calculations or manually programming the code, nodal transformation is a necessity to correctly prescribe boundary conditions in Abaqus.



The nodal transformation matrix **Λ** can be used as follows (in this example):

$$\begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5' \\ D_6' \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos\alpha & \sin\alpha \\ 0 & 0 & 0 & 0 & -\sin\alpha & \cos\alpha \end{bmatrix} \begin{Bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ D_6 \end{Bmatrix} \qquad \begin{Bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5' \\ Q_6' \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos\alpha & \sin\alpha \\ 0 & 0 & 0 & 0 & -\sin\alpha & \cos\alpha \end{bmatrix} \begin{Bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \end{Bmatrix}$$

D' **Λ** **D** **Q'** **Λ** **Q**

Where **D'**, **Q'** are the displacements and reaction forces in the transformed coordinate at each node, and **D**, **Q** are those in the global xy-coordinate. Using the fact that $\boldsymbol{\Lambda^{-1}} = \boldsymbol{\Lambda^{T}}$, we can obtain the ultimate equations that are used for solving the unknowns in D and Q

$$\left(\boldsymbol{\Lambda K \Lambda^{T}}\right)\boldsymbol{D'} = \boldsymbol{Q'}$$

$$\boldsymbol{D} = \boldsymbol{\Lambda^{T}}\boldsymbol{D'}$$

$$\boldsymbol{Q} = \boldsymbol{\Lambda^{T}}\boldsymbol{Q'}$$

References

[1] V. K. Goyal, V. K. Goyal, and H. M. Rodríguez. “Plane Truss and Plane Frame Elements”, in *Finite Element Analysis*. Puerto Rico: Pearson Education, 2017, pp. 441-446.

Appendices

The input files used are courtesy of my friend, Kevin Kauffman, with some modifications by me.

1. Input file in example 1

```
*heading
example5
*node,nset=nall
1,0,0
2,36,0
3,-1.454545454545455,8
4,1.454545454545455,8
5,33.09090909090909,8
6,36,8
7,-2.545454545454545,14
8,2.545454545454545,14
9,30.90909090909091,14
10,36,14
11,-16,22
12,-10,22
13,-4,22
14,4,22
15,12,22
16,20,22
17,28,22
18,36,22
19,-10,25
20,-4,28
21,4,28
22,12,28
23,20,28
24,28,28
25,36,28
*element,type=t2d2,elset=elall
1,1,3
2,1,4
3,2,5
4,2,6
5,3,7
6,3,8
8,4,8
9,5,9
10,5,10
12,6,10
13,7,13
15,8,13
16,8,14
17,9,17
19,10,17
20,10,18
21,11,12
22,11,19
23,12,19
24,13,20
25,12,13
26,19,20
```

```
27,19,13
28,13,21
29,13,14
30,20,21
31,14,21
32,15,22
33,14,15
34,21,22
36,15,21
37,15,16
38,22,23
39,15,23
41,16,23
42,17,24
43,16,17
44,23,24
46,17,23
47,18,25
48,24,25
49,17,25
51,3,4
52,5,6
53,7,8
54,9,10
55,17,18
*solid section,elset=elall,material=steel
1
*material,name=steel
*elastic
30e6,.29
*step
*static
.2,1,1e-5,.2
*boundary
1,1,2,0
2,1,2,0
*cload
11,2,100e3
15,2,-300e3
25,1,-100e3
*end step
```

2. Input file in example 2

```
*heading
example4
*node,nset=nall
1,0,0
2,18,0
3,36,0
4,54,0
5,72,0
6,90,0
7,108,0
8,126,0
9,126,39
10,108,39
11,90,38
12,72,35.5
13,54,32
14,36,27
15,18,20
*element,type=t2d2,elset=el4
1,1,2
```

```
2,2,3
3,3,4
4,4,5
5,5,6
6,6,7
7,7,8
*element,type=t2d2,elset=el2
8,1,15
9,2,15
11,3,14
13,4,13
15,5,12
16,6,12
17,6,11
18,7,11
19,7,10
23,15,14
24,14,13
25,13,12
26,12,11
27,11,10
28,10,9
*element,type=t2d2,elset=el1
10,15,3
12,14,4
14,13,5
21,10,8
22,8,9
*solid section,elset=el4,material=steel
4
*solid section,elset=el2,material=al
2
*solid section,elset=el1,material=steel
1
*material,name=steel
*elastic
30e6,.29
*material,name=al
*elastic
10e6,.33
*step
*static
.2,1,1e-5,.2
*boundary
1,1,2,0
8,1,1,0
9,1,1,0
*cload
1,2,-1000
2,2,-1000
3,2,-1000
4,2,-1000
5,2,-1000
6,2,-1000
7,2,-1000
8,2,-1000
*end step
```

3. The complete working source code and input files will soon be made available at https://www.github.com/hdinh/