

1 Linear truss element theories

1.1 Element-Topology

The elements used here have 2 nodes and only inherit uniaxial stress. They can thus only elongate in their axis-direction.

The nodal displacements are defined as

$$\mathbf{u}^I = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}. \quad (1)$$

The response of the element is defined by the second order differential equation of balance of linear momentum

$$\operatorname{div} \sigma = 0. \quad (2)$$

1.2 Constitutive Relation

The scalar quantity σ denotes the uniaxial stress described by a linear elastic material, where

$$\begin{aligned} E &= \text{Young's Modulus} \\ A &= \text{Cross Section.} \end{aligned} \quad (3)$$

Furthermore, σ is defined as

$$\sigma = EA \varepsilon = EA \tilde{u}'. \quad (4)$$

1.3 Kinematics

In the 3 methods that are used here, the nodal displacements are projected along the truss-axis with

$$\tilde{u}^I = \mathbf{t} \cdot \mathbf{u}^I. \quad (5)$$

Then, the following specification for the derivative of displacements is used:

$$\tilde{u}' = \frac{\tilde{u}^2 - \tilde{u}^1}{l^e}. \quad (6)$$

The elemental vector of unknowns is defined as

$$\mathbf{d}^e = (u_1^1 \quad u_2^1 \quad u_3^1 \quad u_1^2 \quad u_2^2 \quad u_3^2)^T. \quad (7)$$

1.4 Potential

The potential of a linear truss element is given by

$$\Pi = \frac{1}{2} \int_B EA \tilde{u}'^2 dx. \quad (8)$$

In order to reach equilibrium, Π has to become stationary, hence we state

$$\frac{\partial \Pi}{\partial \mathbf{d}^e} = 0. \quad (9)$$

Thus, the \mathbf{R} and \mathbf{K} matrices are defined as

$$\mathbf{R} = \frac{\partial \Pi}{\partial \mathbf{d}^e} \quad \text{and} \quad \mathbf{K} = \frac{\partial \mathbf{R}}{\partial \mathbf{d}^e}. \quad (10)$$

1.5 Standard Galerkin Method

Here, the second order differential equation from (8) is transformed to a first order differential equation via multiplying with a trial function and partial integration:

$$G = \int_B EA \tilde{u}' \delta \tilde{u}' dx = 0. \quad (11)$$

With the standard Galerkin Method, the element matrices \mathbf{R} and \mathbf{K} are obtained via variations and within this linear framework yields

$$\mathbf{R} = \frac{\partial G}{\partial \delta \mathbf{u}^I} \quad \text{and} \quad \mathbf{K} = \frac{\partial \mathbf{R}}{\partial \mathbf{u}^I}. \quad (12)$$

1.6 Pseudo Potential

The pseudo potential is an *AceGen* specific method, and in this case looks like this:

$$\begin{aligned} G^p &= \int_B EA \tilde{u}'^2 dx = 0 \\ &= \int_B \sigma \tilde{u}' dx = 0, \end{aligned} \quad (13)$$

where σ is defined as in (4).

In *AceGen* we use the automatic differentiation capabilities. In the given framework the element right hand side \mathbf{R} is procured using a differentiation exception. Thus we obtain

$$\mathbf{R} = \left. \frac{\partial G^p}{\partial \mathbf{d}^e} \right|_{\sigma=\text{const.}} \quad \text{and} \quad \mathbf{K} = \frac{\partial \mathbf{R}}{\partial \mathbf{d}^e}. \quad (14)$$

2 AceGen implementation

```
<< AceGen` ;
SMSInitialize["truss_galerkin", "Environment" → "AceFEM", "Mode" → "Prototype"];
SMSTemplate[
  "SMSTopology" → "C1",
  "SMSGroupDataNames" → {"E -elastic modulus", "A -cross section"},
  "SMSDefaultData" → {21000, 0.02}
];
```

Figure 1: First input

Loading the *AceGen* package and initializing constants.

```
SMSStandardModule["Tangent and residual"];
{Em, A} = SMSReal[Table[es$$["Data", i], {i, 2}]];
XI = Table[SMSReal[nd$$[i, "X", j], {i, 2}, {j, 3}]];
UI = Table[SMSReal[nd$$[i, "at", j], {i, 2}, {j, 3}]];
δUI = SMSFreeze[Table[0, {i, 2}, {j, 3}]];
DOFVector = Flatten[UI];
δDOFVector = Flatten[δUI];
```

Figure 2: Picking a subroutine and defining variables

nd\$\$ denotes the nodal data, and *es\$\$* denotes the element structure. The initialized variational quantities only find use in the standard Galerkin Method.

```
Dir = (XI[[2]] - XI[[1]]);
le = SMSSqrt[Dir.Dir];
tang = Dir/le;
UDerivative =  $\frac{\text{UI}[[2]].\text{tang} - \text{UI}[[1]].\text{tang}}{\text{le}}$ ;
Potential = 0.5 (Em * A * UDerivative * UDerivative) * le;
R = SMSD[Potential, DOFVector];
K = SMSD[R, DOFVector];
```

Figure 3: Main Code for the Potential

Figure (3) shows the constitutive law, vector **R**, and matrix **K**.

```

Dir = XI[[2]] - XI[[1]];
le = SMSSqrt[Dir.Dir];
tang = Dir/le;
UDerivative =  $\frac{UI[[2]].tang - UI[[1]].tang}{le}$ ;
 $\delta UDerivative = \frac{\delta UI[[2]].tang - \delta UI[[1]].tang}{le}$ ;
G = (Em * A * UDerivative *  $\delta UDerivative$ ) * le;
R = SMSD[G,  $\delta DOFVector$ ];
K = SMSD[R,  $DOFVector$ ];

```

Figure 4: Main Code for the Standard Galerkin Method

The Standard Galerkin Method uses variational quantities or *trial functions*.

```

Dir = (XI[[2]] - XI[[1]]);
le = SMSSqrt[Dir.Dir];
tang = Dir/le;
UDerivative =  $\frac{UI[[2]].tang - UI[[1]].tang}{le}$ ;
 $\delta UDerivative = \frac{\delta UI[[2]].tang - \delta UI[[1]].tang}{le}$ ;
 $\sigma = Em * A * UDerivative$ ;
GPseudo =  $\sigma * UDerivative * le$ ;
R = SMSD[GPseudo,  $DOFVector$ , "Constant" → { $\sigma$ }];
K = SMSD[R,  $DOFVector$ ];

```

Figure 5: Main Code for the Pseudo Potential

In the Pseudo Potential, a differentiation exception as mentioned in (14) is used.

```

SMSExport[R, p$$];
SMSExport[K, s$$];
SMSWrite[];

```

Figure 6: Export

Lastly, the element is finalized and the element matrices \mathbf{R} and \mathbf{K} are exported to the fields $p\$\$,$ the element load vector, and $s\$\$,$ the element stiffness matrix, in *AceFEM*.

3 AceFEM examples

```
<< AceFEM` ;
SMTInputData[];
SMTAddDomain["Ω", "truss_galerkin", {"E *" → 21 000, "A *" → 0.02}];
```

Figure 7: AceFEM system

AceFEM is initialized and a domain is created.

```
SMTAddMesh[Line[{{0, 0, 0}, {2, 0, 0}}, "Ω", "C1"];
SMTAddMesh[Line[{{2, 0, 0}, {2, 2, 0}}, "Ω", "C1"];
SMTAddMesh[Line[{{2, 2, 0}, {0, 2, 0}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 0, 0}, {0, 2, 0}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 0, 0}, {0, 0, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{2, 0, 0}, {2, 0, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{2, 2, 0}, {2, 2, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 2, 0}, {0, 2, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 0, 0}, {2, 0, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 2, 0}, {2, 2, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 0, 0}, {0, 2, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{2, 0, 0}, {2, 2, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 0, 2}, {2, 0, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{2, 0, 2}, {2, 2, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{2, 2, 2}, {0, 2, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 0, 2}, {0, 2, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 0, 2}, {2, 2, 2}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 0, 2}, {0, 0, 4}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 2, 2}, {0, 2, 4}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 0, 2}, {0, 2, 4}}, "Ω", "C1"];
SMTAddMesh[Line[{{0, 0, 4}, {0, 2, 4}}, "Ω", "C1"];
SMTAddMesh[Line[{{2, 0, 2}, {0, 0, 4}}, "Ω", "C1"];
SMTAddMesh[Line[{{2, 2, 2}, {0, 2, 4}}, "Ω", "C1"];
```

Figure 8: AceFEM system

This mesh is done manually by generating lines with the given element type *C1* on domain Ω . A visual example is provided in Figures (11a) and (11b).

```

SMTAddEssentialBoundary["Z" == 0 &, 1 → 0, 2 → 0, 3 → 0];
SMTAddNaturalBoundary["Z" == 4 && "Y" == 0 &, 1 → -12];
SMTAddNaturalBoundary["Z" == 4 && "Y" == 2 &, 1 → 6];
SMTAnalysis[];

```

Figure 9: Boundary Conditions And Calculus Phase

Boundary conditions are applied where necessary, and *AceFEM* is tasked to proceed to calculations.

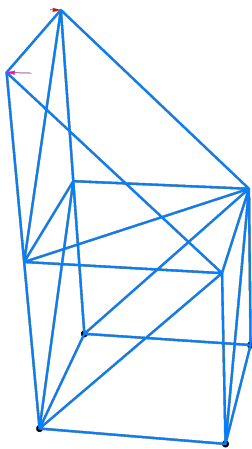
```

SMTNextStep[1, 1];
SMTNewtonIteration[]
SMTNewtonIteration[]

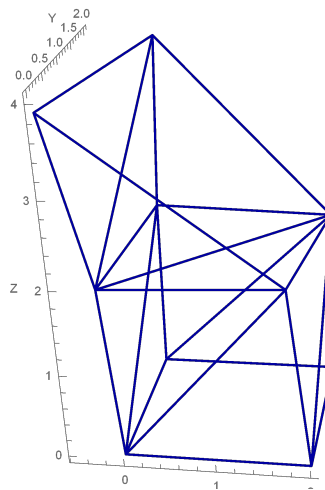
```

Figure 10: The Last Input: Newton Iterations

The problem is analyzed by setting the load factor to one, and performing two Newton steps, which is sufficient for a linear problem.



(a) Undeformed mesh



(b) Deformed mesh

The 2 Figures above show the results of the aforementioned code.