Analysis of Problems in Linear Elasticity with the Finite Element Method (FEM)

Bauhaus Universität Weimar, Informatik im Bauwesen

2009

Contents

Analysis of Problems in Linear Elasticity with the Finite Element Method (FEM)

1 Installation

Application software on the basis of the FEM-Framework may be downloaded as a zip-file that is provided under the name ***FEM.zip***. Root directory for the installation is *FEM*. The zip-files contains:

- the source files (subdirectory *source*) for various FEM-applications which can either be compiled separately in an individual environment or imported into an IDE (e.g. Eclipse),
- the class files (subdirectory *binary*) for applications to be run directly without recompilation,
- some input files (subdirectory *input*) for example calculations to be performed,
- the Java-documentation (subdirectory *doc*) and
- some class notes (subdirectory *notes*) for explanation of the fundamental equations and solution strategies.

Currently, three different applications are provided

- linear elasticity analysis (subdirectory *source/elasticity*),
- linear stationary and transient heat transfer analysis (subdirectory *source/heat*) and
- linear static and dynamic analysis of structural beams and frames (subdirectory *source/structuralAnalysis*).

Finally, a 2D-CAD application is provided (subdirectory *cad*) for integration of analysis into an interactive CAD-environment. The CAD-software used for these purposes is openly available under the name *cademia* (http://www.cademia.org/).

If not only access to the source files is required but if major development efforts are to be undertaken, an “Integrated Development Environment – IDE” is a preferable choice. A powerful and free IDE is available, for example, under “Eclipse IDE for Java Developers” (http://www.eclipse.org/downloads/). In this case, the zip-file with the source files may be unpacked and the sources may be imported into the IDE.

The additional overhead of installing and learning an IDE may seem prohibitive to many users but over time with many enhancements and modifications to be performed, the additional investment quickly pays off. New efforts often start by “only needing to do little development” but quickly turning into something much more complex as initially intended.

It is our experience that even students in class quickly appreciate the additional help and support from an IDE over the tedious development using console input with a context sensitive editor and a standard compiler invocation. They are ready to accept the additional efforts for learning an IDE and this even motivates them better.

2 Formulation of the FEM Solution

The analysis of problems in linear elasticity consists of the Finite Element approximation of the system equations with a corresponding formulation of the element equations, the influences on the model (loads), the constitutive relations (material properties) and the external support conditions.

Furthermore, a specific element must be chosen that is appropriate for the problem to be solved defining the geometry of the problem (element geometry) and the element characteristics for the model.

2.1 Finite Element Approximation for Linear Elasticity

The **system equations** governing the behaviour of a linear elastic **two-dimensional** continuum are given as follows (see for example: [3]):

$$\int_A \delta\boldsymbol{\varepsilon}^T \mathbf{s}\, dA = \int_A \delta\mathbf{w}^T \mathbf{p}\, dA + \sum_{K_q} \delta u_i\, q_{ri} + \sum_{K_u} \delta u_{ri}\, a_i$$

with: **A** center plane of the body, $\delta\boldsymbol{\varepsilon}$ variation of strain state, **s** state of internal forces, $\delta\mathbf{w}$ variation of displacement state within an element, **p** area load per unit of center plane, $\mathbf{K}_q$ set of external influences (loads), **u** is the vector of system nodal degrees of freedom (displacements), $\mathbf{q}_{ri}$ prescribed external load on boundary nodes, $\mathbf{K}_u$ set of prescribed displacements, $\mathbf{a}_i$ support force associated with prescribed displacements at nodal degrees of freedom.

The **constitutive relations** (**stress-strain-equations**) for plane strain are defined by:

$$\boldsymbol{\sigma}_e = \begin{Bmatrix} \sigma_{00} \\ \sigma_{11} \\ \sigma_{01} \end{Bmatrix} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{Bmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{Bmatrix} \begin{Bmatrix} \varepsilon_{00} \\ \varepsilon_{11} \\ \varepsilon_{01} \end{Bmatrix}$$

$$\boldsymbol{\sigma}_e = \mathbf{E}_e * \varepsilon_e$$

where: E is Young's modulus of elasticity and
ν is Poisson's ratio

Discretization over the center plane results in two sums of element integrals $\mathbf{A}_e$ over all elements:

$$\sum_e \int_{A_e} \delta\boldsymbol{\varepsilon}_e^T\, \mathbf{s}_e\, dA = \sum_e \int_{A_e} \delta\mathbf{w}_e^T\, \mathbf{p}_e\, dA + \sum_{K_q} \delta u_i\, q_{ri} + \sum_{K_u} \delta u_{ri}\, a_i$$

Element displacements and strains are conventionally approximated via **element shape function expressions** S_e which may assume different forms depending on the choice of element:

$$\mathbf{w} \cong \sum_e S_e \; \mathbf{u}_e \qquad \text{with: } \mathbf{u}_e \text{ vector of nodal displacements}$$

$$\mathbf{p} \cong \sum_e S_e \; \mathbf{b}_e \qquad \text{with: } \mathbf{b}_e \text{ vector of nodal forces}$$

The **linear strain state** is defined by the vector of element strains ε_e:

$$\boldsymbol{\varepsilon}_e = \begin{Bmatrix} \varepsilon_{00} \\ \varepsilon_{11} \\ \varepsilon_{01} \end{Bmatrix} = \begin{Bmatrix} \dfrac{\partial u_0}{\partial x_0} \\ \dfrac{\partial u_1}{\partial x_1} \\ \dfrac{\partial u_0}{\partial x_1} + \dfrac{\partial u_1}{\partial x_0} \end{Bmatrix} = \begin{Bmatrix} \dfrac{\partial S_e}{\partial x_0} & 0 \\ 0 & \dfrac{\partial S_e}{\partial x_1} \\ \dfrac{\partial S_e}{\partial x_1} & \dfrac{\partial S_e}{\partial x_0} \end{Bmatrix} * \mathbf{u}_e$$

$$\varepsilon_e = \mathbf{B}_e * \mathbf{u}_e$$

Assuming that we have **constant stresses over the width of the body**, we have

$$\mathbf{s}_e = t_e * \boldsymbol{\sigma}_e = t_e * \mathbf{E}_e * \varepsilon_e = t_e * \mathbf{E}_e * \mathbf{B}_e * \mathbf{u}_e$$

where: t_e is the constant thickness (width) of an element

Substituting this information into the system equation above results in the following set of equations for a general formulation in linear elasticity:

$$\sum_e \delta \mathbf{u}_e^T \int_{A_e} t_e \mathbf{B}_e^T \mathbf{E}_e \mathbf{B}_e u_e dA = \sum_e \delta \mathbf{u}_e^T \int_{A_e} \mathbf{S}_e^T \mathbf{p}_e \, dA + \sum_{K_u} \delta u_i \, q_{ri} + \sum_{K_u} \delta u_{ri} \, a_i$$

Depending on the type of elements and interpolation functions chosen this will result in a different set of linear equations of the form:

$$\mathbf{K}_S \mathbf{u}_S = \mathbf{p}_S - \mathbf{q}_S$$

The system matrix $\mathbf{K}_S$ is multiplied by the vector of nodal displacements $\mathbf{u}_S$ referred to as primal system vector. The Right-Hand-Side (RHS) consists of the external influences on the system referred to as load vector $\mathbf{p}_S$ (forces applied at the boundaries and inside the body) and of the support conditions on the system $\mathbf{q}_S$ referred to as dual system vector (forces resulting from predefined displacements at boundary nodes).

The **system equations** governing the behaviour of a linear elastic **three-dimensional** continuum are given as follows (see for example: [3]):

$$\int_V \delta \boldsymbol{\varepsilon}^T \mathbf{s} \, dV = \int_V \delta \mathbf{w}^T \mathbf{p} \, dV + \sum_{K_q} \delta u_i \, q_{ri} + \sum_{K_u} \delta u_{ri} \, a_i$$

with: **V** volume of the body, $\delta\boldsymbol{\varepsilon}$ variation of strain state, **s** state of internal forces, $\delta\mathbf{w}$ variation of displacement state within an element, **p** load per unit of volume, $\mathbf{K}_q$ set of external influences (loads), **u** is the vector of system nodal degrees of freedom (displacements), $\mathbf{q}_{ri}$ prescribed external load on boundary nodes, $\mathbf{K}_u$ set of prescribed displacements, $\mathbf{a}_i$ support force associated with prescribed displacements at nodal degrees of freedom.

The **constitutive relations** (**stress-strain-equations**) for plane strain are defined by:

$$\boldsymbol{\sigma}_e = \begin{Bmatrix} \sigma_{00} \\ \sigma_{11} \\ \sigma_{22} \\ \sigma_{01} \\ \sigma_{12} \\ \sigma_{20} \end{Bmatrix} = \frac{E}{(1+\nu)} \begin{Bmatrix} \frac{1-\nu}{1-2\nu} & \frac{\nu}{1-2\nu} & \frac{\nu}{1-2\nu} & 0 & 0 & 0 \\ \frac{\nu}{1-2\nu} & \frac{1-\nu}{1-2\nu} & \frac{\nu}{1-2\nu} & 0 & 0 & 0 \\ \frac{\nu}{1-2\nu} & \frac{\nu}{1-2\nu} & \frac{1-\nu}{1-2\nu} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \end{Bmatrix} \begin{Bmatrix} \varepsilon_{00} \\ \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{01} \\ \varepsilon_{12} \\ \varepsilon_{20} \end{Bmatrix}$$

$$\boldsymbol{\sigma}_e \quad = \quad \mathbf{E}_e \quad * \ \boldsymbol{\varepsilon}_e$$

where: E is Young's modulus of elasticity and
ν is Poisson's ratio

Discretization over the volume results in two sums of element integrals $\mathbf{V}_e$ over all elements:

$$\sum_e \int_{V_e} \delta\boldsymbol{\varepsilon}_e^T \, \mathbf{s}_e \, dV = \sum_e \int_{V_e} \delta\mathbf{w}_e^T \, \mathbf{p}_e \, dV + \sum_{K_q} \delta u_i \, q_{ri} + \sum_{K_u} \delta u_{ri} \, a_i$$

Element displacements and strains are conventionally approximated via **element shape function expressions** S_e which may assume different forms depending on the choice of element:

$\mathbf{w} \cong \sum_e S_e \ \mathbf{u}_e$ with: $\mathbf{u}_e$ vector of nodal displacements

$\mathbf{p} \cong \sum_e S_e \ \mathbf{b}_e$ with: $\mathbf{b}_e$ vector of nodal forces

The **linear strain state** is defined by the vector of element strains ε_e:

$$\boldsymbol{\varepsilon}_e = \begin{Bmatrix} \varepsilon_{00} \\ \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{01} \\ \varepsilon_{12} \\ \varepsilon_{20} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u_0}{\partial x_0} \\ \frac{\partial u_1}{\partial x_1} \\ \frac{\partial u_2}{\partial x_2} \\ \frac{\partial u_0}{\partial x_1} + \frac{\partial u_1}{\partial x_0} \\ \frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1} \\ \frac{\partial u_0}{\partial x_2} + \frac{\partial u_2}{\partial x_0} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial S_e}{\partial x_0} & 0 & 0 \\ 0 & \frac{\partial S_e}{\partial x_1} & 0 \\ 0 & 0 & \frac{\partial S_e}{\partial x_2} \\ \frac{\partial S_e}{\partial x_1} & \frac{\partial S_e}{\partial x_1} & 0 \\ 0 & \frac{\partial S_e}{\partial x_2} & \frac{\partial S_e}{\partial x_1} \\ \frac{\partial S_e}{\partial x_2} & 0 & \frac{\partial S_e}{\partial x_0} \end{Bmatrix} * \ \mathbf{u}_e$$

$$\varepsilon_e \quad = \quad \mathbf{B}_e \quad * \ \mathbf{u}_e$$

Substituting this information into the system equation above results in the following set of equations for a general formulation in linear elasticity:

$$\sum_e \delta \mathbf{u}_e^T \int_{V_e} \mathbf{B}_e^T \, \mathbf{E}_e \, \mathbf{B}_e \, u_e dV = \sum_e \delta \mathbf{u}_e^T \int_{V_e} \mathbf{S}_e^T \mathbf{p}_e \, dV \ + \sum_{K_u} \delta u_i \, q_{ri} \ + \sum_{K_u} \delta u_{ri} \, a_i$$

Depending on the type of elements and interpolation functions chosen this will result in a different set of linear equations of the form:

$\mathbf{K}_S \mathbf{u}_S = \mathbf{p}_S - \mathbf{q}_S$

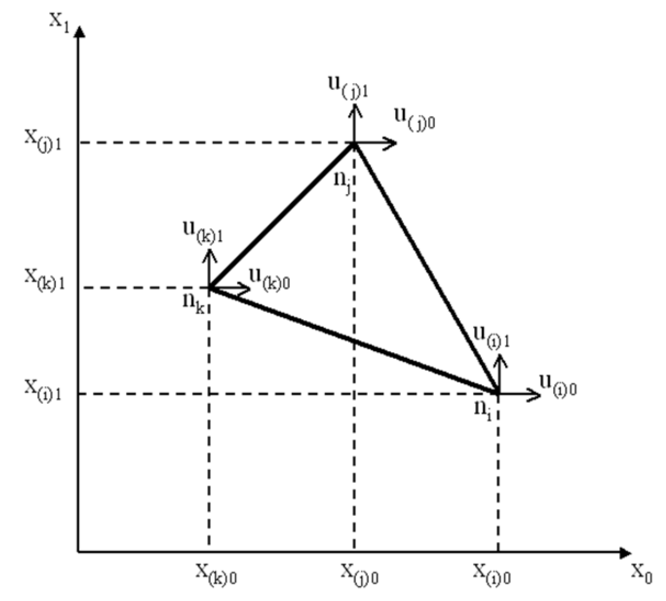
The system matrix $\mathbf{K}_S$ is multiplied by the vector of nodal displacements $\mathbf{u}_S$ referred to as primal system vector. The Right-Hand-Side (RHS) consists of the external influences on the system referred to as load vector $\mathbf{p}_S$ (forces applied at the boundaries and inside the body) and of the support conditions on the system $\mathbf{q}_S$ referred to as dual system vector (forces resulting from predefined displacements at boundary nodes).

2.2 Linear 3-Node Elements for Linear Elasticity

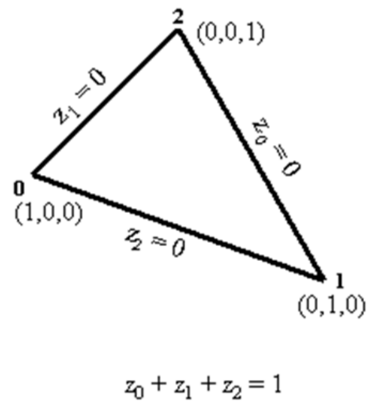
Generally, a variety of elements and approximation functions may be chosen for approximating the geometry of the model and the functional behaviour of the model.

For the purposes of this text, triangular elements were chosen to approximate the element geometry and linear functions for the element characteristics in linear elasticity. The element geometry is defined by deriving the implementation from the abstract implementation *AbstractLinear3Node2D*.

Physical problems of linear elasticity are defined by two degrees of freedom per node, i.e. the displacements in x_0- and x_1-direction ($u_{(i)0}, u_{(i)1}$; $u_{(j)0}, u_{(j)1}$; $u_{(k)0}, u_{(k)1}$) as primal unknowns.



global coordinates



normalized triangular coordinates

The approximation of displacements and forces within an element is accomplished by the following linear interpolation using the **shape functions** in triangular coordinates (z_0, z_1, z_2) for all three nodes (i,j,k) in both directions (0,1):

$$\mathbf{u}(z)_e = \begin{Bmatrix} z_0 & 0 & z_1 & 0 & z_2 & 0 \\ 0 & z_0 & 0 & z_1 & 0 & z_2 \end{Bmatrix} \begin{Bmatrix} u_{(i)0} \\ u_{(i)1} \\ u_{(j)0} \\ u_{(j)1} \\ u_{(k)0} \\ u_{(k)1} \end{Bmatrix}$$

$$\mathbf{u}(z)_e = \qquad \mathbf{S}^T_e \qquad \mathbf{u}_e$$

The derivative of the shape function vector $\mathbf{S}^T_e$ with respect to the coordinate z_i is called the i-th **normalized derivative of the shape function vector**. They are arranged rowwise in a matrix $\mathbf{U_z}$.

$$\frac{\partial}{\partial z_0}(\mathbf{S}^T{}_e) = \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{Bmatrix}$$

$$\frac{\partial}{\partial z_1}(\mathbf{S}^T{}_e) = \begin{Bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{Bmatrix} \qquad \frac{\partial}{\partial z_i}(\mathbf{S}^T{}_e) = \mathbf{U_z}$$

$$\frac{\partial}{\partial z_2}(\mathbf{S}^T{}_e) = \begin{Bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{Bmatrix}$$

where: $\mathbf{U_z}$ is the identity matrix **I**

The **topological mapping** of normalized local onto corresponding global coordinates is defined by matrix $\mathbf{R}_e$ which maps local degrees of freedom (dof) $\mathbf{u}_e$ to global degrees of freedom $\mathbf{u}_S$. Several local dof may be mapped to the same global dof. This mapping is implicitly handled by the FEM Framework on the basis of the system indices generated for all nodal degrees of freedom. It does not need to be performed by the user. System indices refer to the global degrees of freedom associated with each node (e.g. u_i) and elements with normalized coordinates are generated with reference to these nodes.

$$\begin{Bmatrix} u_{(i)0} \\ u_{(i)1} \\ u_{(j)0} \\ u_{(j)1} \\ u_{(k)0} \\ u_{(k)1} \end{Bmatrix} = \begin{Bmatrix} 0 & 0 & .. & 1 & 0 & .. & 0 & 0 & .. & 0 & 0 & .. & 0 & 0 \\ 0 & 0 & .. & 0 & 1 & .. & 0 & 0 & .. & 0 & 0 & .. & 0 & 0 \\ 0 & 0 & .. & 0 & 0 & .. & 1 & 0 & .. & 0 & 0 & .. & 0 & 0 \\ 0 & 0 & .. & 0 & 0 & .. & 0 & 1 & .. & 0 & 0 & .. & 0 & 0 \\ 0 & 0 & .. & 0 & 0 & .. & 0 & 0 & .. & 1 & 0 & .. & 0 & 0 \\ 0 & 0 & .. & 0 & 0 & .. & 0 & 0 & .. & 0 & 1 & .. & 0 & 0 \end{Bmatrix} \begin{Bmatrix} u_0 \\ u_1 \\ .. \\ u_{2i} \\ u_{2i+1} \\ .. \\ u_{2j} \\ u_{2j+1} \\ .. \\ u_{2k} \\ u_{2k+1} \\ .. \\ u_{n-1} \\ u_n \end{Bmatrix}$$

$\mathbf{u}_e$ = $\mathbf{R}_e$ $\mathbf{u}_S$

Element strains in triangular elements define the total strain at any point within the element and can be defined by the element strain vector ε_e contributing to internal work.

The transformation between element strains and displacements as defined above can be applied to a general three node element as follows:

$$\varepsilon_{00} = u_{0,0} = \frac{\partial u_0}{\partial z_0}\frac{\partial z_0}{\partial x_0} + \frac{\partial u_0}{\partial z_1}\frac{\partial z_1}{\partial x_0} + \frac{\partial u_0}{\partial z_2}\frac{\partial z_2}{\partial x_0}$$

$$\varepsilon_{11} = u_{1,1} = \frac{\partial u_1}{\partial z_0}\frac{\partial z_0}{\partial x_1} + \frac{\partial u_1}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial u_1}{\partial z_2}\frac{\partial z_2}{\partial x_1}$$

$$\varepsilon_{01} = u_{0,1} + u_{1,0} = \frac{\partial u_0}{\partial z_0}\frac{\partial z_0}{\partial x_1} + \frac{\partial u_0}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial u_0}{\partial z_2}\frac{\partial z_2}{\partial x_1} + \frac{\partial u_1}{\partial z_0}\frac{\partial z_0}{\partial x_0} + \frac{\partial u_1}{\partial z_1}\frac{\partial z_1}{\partial x_0} + \frac{\partial u_1}{\partial z_2}\frac{\partial z_2}{\partial x_0}$$

The derivatives of the element displacements with respect to the normalized triangular coordinates can be derived from the equation $u(z)_e = \mathbf{S}^T_e * \mathbf{u}_e$ defined above, where

with: $\frac{\partial u(z)_i}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\mathbf{S}^T_e * \mathbf{u}_e\right) = \mathbf{U}_z\, \mathbf{u}_e = \mathbf{u}_e$

$$\boldsymbol{\varepsilon}_e = \begin{Bmatrix} \varepsilon_{00} \\ \varepsilon_{11} \\ \varepsilon_{01} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u_0}{\partial x_0} \\ \frac{\partial u_1}{\partial x_1} \\ \frac{\partial u_0}{\partial x_1} + \frac{\partial u_1}{\partial x_0} \end{Bmatrix} = \begin{bmatrix} \frac{\partial z_0}{\partial x_0} & 0 & \frac{\partial z_1}{\partial x_0} & 0 & \frac{\partial z_2}{\partial x_0} & 0 \\ 0 & \frac{\partial z_0}{\partial x_1} & 0 & \frac{\partial z_1}{\partial x_1} & 0 & \frac{\partial z_2}{\partial x_1} \\ \frac{\partial z_0}{\partial x_1} & \frac{\partial z_0}{\partial x_0} & \frac{\partial z_1}{\partial x_1} & \frac{\partial z_1}{\partial x_0} & \frac{\partial z_2}{\partial x_1} & \frac{\partial z_2}{\partial x_0} \end{bmatrix} \begin{Bmatrix} u_{(i)0} \\ u_{(i)1} \\ u_{(j)0} \\ u_{(j)1} \\ u_{(k)0} \\ u_{(k)1} \end{Bmatrix}$$

substituting the derivatives of the shape function vector from above:

$$\mathbf{S}_{xe} = \begin{Bmatrix} Sx_{(0)0} & Sx_{(0)1} \\ Sx_{(1)0} & Sx_{(1)1} \\ Sx_{(2)0} & Sx_{(2)1} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial z_0}{\partial x_0} & \frac{\partial z_0}{\partial x_1} \\ \frac{\partial z_1}{\partial x_0} & \frac{\partial z_1}{\partial x_1} \\ \frac{\partial z_2}{\partial x_0} & \frac{\partial z_2}{\partial x_1} \end{Bmatrix} = \frac{1}{\det \mathbf{X}_{zu}} * \begin{Bmatrix} c_{11} & -c_{01} \\ -c_{10} & c_{00} \\ c_{10} - c_{11} & c_{01} - c_{00} \end{Bmatrix}$$

$$\varepsilon_e = \frac{1}{\det \mathbf{X}_{zu}} \begin{Bmatrix} c_{11} & 0 & -c_{10} & 0 & c_{10} - c_{11} & 0 \\ 0 & -c_{01} & 0 & c_{00} & 0 & c_{01} - c_{00} \\ -c_{01} & c_{11} & c_{00} & -c_{10} & c_{01} - c_{00} & c_{10} - c_{11} \end{Bmatrix} \begin{Bmatrix} u_{(i)0} \\ u_{(i)1} \\ u_{(j)0} \\ u_{(j)1} \\ u_{(k)0} \\ u_{(k)1} \end{Bmatrix}$$

$$\varepsilon_e = \qquad \mathbf{B}_e \qquad * \quad \mathbf{u}_e$$

Element matrices are determined by evaluating for each element the integrals:

$$\int_{A_e} t_e\, \mathbf{B}_e^T\, E_e\, \mathbf{B}_e\, \mathrm{d}A$$

In case of a linear triangle, the element strain displacement matrix $\mathbf{B_e}$ and the value of the determinant are independent of the normalized coordinates z_i and thus constant at each point within the triangle. The element thickness t_e was also assumed to be constant. The integrals can therefore be evaluated analytically for each element.

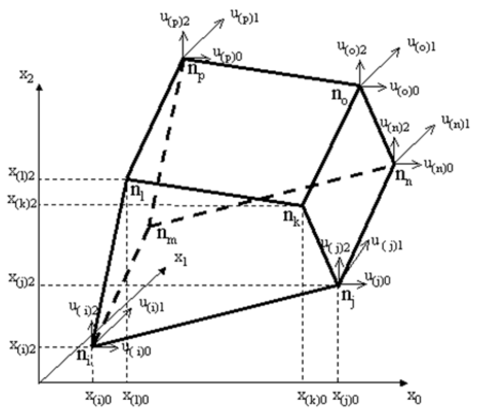
$$\mathbf{k}_e = t_e\, \mathbf{B}^T_e\, E_e\, \mathbf{B}_e\, \text{area}_e$$

In general, integration of element matrices is accomplished numerically, i.e. by evaluating the individual contributions at each Gauss point, by summing up contributions for all Gauss points and by multiplying the results with the corresponding Gauss weights. Thus, evaluation at three Gauss point and application of Gauss weight 1/3 would result in the same constant value of the shape function derivatives and the determinant.

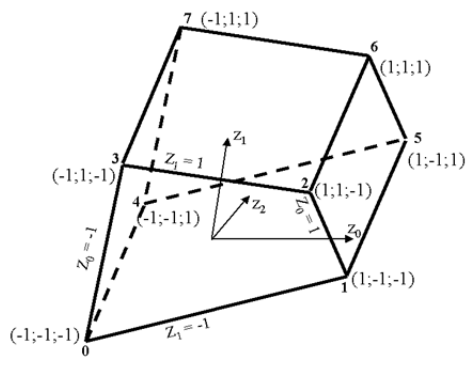
2.3 Linear 8-Node Elements in 3D for Linear Elasticity

Elements with eight nodes and a linear variation of coordinate functions in each direction may be chosen to approximate the element geometry and element characteristics in elasticity analysis. The element geometry with the shape function definitions, the shape function derivatives and the Jacobian is defined by deriving the implementation from the abstract implementation *AbstractLinear8Node3D*.

Physical problems of linear elasticity are defined by three degrees of freedom per node, i.e. the displacements at the nodes i,j,k,l,m,n,o,p (e.g. $u_{0(i)}$, $u_{1(i)}$, $u_{2(i)}$) as primal unknowns).



global coordinates



normalized coordinates

The approximation of temperatures and heat streams within an element is accomplished by linear interpolation using the **shape functions** in global coordinates (z_0, z_1, $z_{"}$) for all eight nodes (0,1, 2, 3, 4, 5, 6, 7) in all directions (0,1,2):

$$\mathbf{u}(z)_e \quad = \quad \mathbf{S}^T_e \quad \mathbf{u}_e$$

where: $\mathbf{u}_e$ is the vector of nodal displacements at 8 nodes in 3 directions $(u_{(i)0}, u_{(i)1}, u_{(i)2})$

and $\mathbf{S}^T_e$ is the (3*24) matrix of shape function vectors **s**(z) in all 3 directions

The **topological mapping** of normalized onto corresponding global coordinates is defined by matrix $\mathbf{R}_e$ which maps local field variables $\mathbf{u}_e$ at all nodes to global field variables $\mathbf{u}_s$. This mapping is implicitly handled by the FEM Framework on the basis of the system indices generated for all nodal degrees of freedom. It does not need to be performed by the user. System indices refer to the global degrees of freedom associated with each node (e.g. u_i) and elements with normalized coordinates are generated with reference to these nodes.

$$\mathbf{u}_e \quad = \quad \mathbf{R}_e \quad \mathbf{u}_s$$

Element strains in 3D elements define the total strain at any point within the element and can be defined by the element strain vector ε_e contributing to internal work.

The transformation between element strains and displacements as defined above can be applied to a general eight node element as follows:

$$\varepsilon_{00} = u_{0,0} = \frac{\partial u_0}{\partial z_0}\frac{\partial z_0}{\partial x_0} + \frac{\partial u_0}{\partial z_1}\frac{\partial z_1}{\partial x_0} + \frac{\partial u_0}{\partial z_2}\frac{\partial z_2}{\partial x_0}$$

$$\varepsilon_{11} = u_{1,1} = \frac{\partial u_1}{\partial z_0}\frac{\partial z_0}{\partial x_1} + \frac{\partial u_1}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial u_1}{\partial z_2}\frac{\partial z_2}{\partial x_1}$$

$$\varepsilon_{22} = u_{2,2} = \frac{\partial u_2}{\partial z_0}\frac{\partial z_0}{\partial x_2} + \frac{\partial u_2}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial u_2}{\partial z_2}\frac{\partial z_2}{\partial x_2}$$

$$\varepsilon_{01} = u_{0,1} + u_{1,0} = \frac{\partial u_0}{\partial z_0}\frac{\partial z_0}{\partial x_1} + \frac{\partial u_0}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial u_0}{\partial z_2}\frac{\partial z_2}{\partial x_1} + \frac{\partial u_1}{\partial z_0}\frac{\partial z_0}{\partial x_0} + \frac{\partial u_1}{\partial z_1}\frac{\partial z_1}{\partial x_0} + \frac{\partial u_1}{\partial z_2}\frac{\partial z_2}{\partial x_0}$$

$$\varepsilon_{12} = u_{1,2} + u_{2,1} = \frac{\partial u_1}{\partial z_0}\frac{\partial z_0}{\partial x_2} + \frac{\partial u_1}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial u_1}{\partial z_2}\frac{\partial z_2}{\partial x_2} + \frac{\partial u_2}{\partial z_0}\frac{\partial z_0}{\partial x_1} + \frac{\partial u_2}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial u_2}{\partial z_2}\frac{\partial z_2}{\partial x_1}$$

$$\varepsilon_{20} = u_{0,2} + u_{2,0} = \frac{\partial u_0}{\partial z_0}\frac{\partial z_0}{\partial x_2} + \frac{\partial u_0}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial u_0}{\partial z_2}\frac{\partial z_2}{\partial x_2} + \frac{\partial u_2}{\partial z_0}\frac{\partial z_0}{\partial x_0} + \frac{\partial u_2}{\partial z_1}\frac{\partial z_1}{\partial x_0} + \frac{\partial u_2}{\partial z_2}\frac{\partial z_2}{\partial x_0}$$

with: $\frac{\partial u(z)_i}{\partial x_i} = \frac{\partial \mathbf{s}}{\partial x_i}\,\mathbf{u}_e = \mathbf{S}_{xe}\,\mathbf{u}_e$

and substituting the derivatives of the shape function vector $\mathbf{S}_{xe}$ from above for each node (indicated by brackets) we get for the first 2 of a total of 8 nodes:

$$\varepsilon_e = \frac{1}{\det \mathrm{X}_{zu}} \begin{Bmatrix} Sx_{(0)0} & 0 & 0 & Sx_{(1)0} & 0 & 0 & \dots \\ 0 & Sx_{(0)1} & 0 & 0 & Sx_{(1)1} & 0 & \dots \\ 0 & 0 & Sx_{(0)2} & 0 & 0 & Sx_{(1)2} & \dots \\ Sx_{(0)1} & Sx_{(0)0} & 0 & Sx_{(1)1} & Sx_{(1)0} & 0 & \dots \\ 0 & Sx_{(0)2} & Sx_{(0)1} & 0 & Sx_{(1)2} & Sx_{(1)1} & \dots \\ Sx_{(0)2} & 0 & Sx_{(0)0} & Sx_{(1)2} & 0 & Sx_{(1)0} & \dots \end{Bmatrix} * \begin{Bmatrix} u_{(i)0} \\ u_{(i)1} \\ u_{(i)2} \\ u_{(j)0} \\ u_{(j)1} \\ u_{(j)2} \\ .. \\ .. \end{Bmatrix}$$

$$\varepsilon_e = \qquad \mathbf{B}_e \qquad * \quad \mathbf{u}_e$$

Element matrices are determined by evaluating for each element the integrals:

$$\int_{V_e} \mathbf{B}_e^T \mathbf{E}_\mathbf{e}\, \mathbf{B}_\mathbf{e}\, \mathrm{d}V$$

$$\mathrm{d}V = \mathrm{d}x_0\, \mathrm{d}x_1\, \mathrm{d}x_2 = \det \mathbf{X}_z\, \mathrm{d}z_0\, \mathrm{d}z_1\, \mathrm{d}z_2$$

In case of a linear trilateral, the element strain displacement transformation matrix $\mathbf{B_e}$ and the value of the determinant are evaluated numerically by Gaussian integration. For this purpose, the values of matrix $\mathbf{B}_e$ are evaluated at corresponding Gauss points and multiplied by associated Gauss weights. Integration over the volume of the elements is carried out in normalized coordinates.

Thus, element matrices may be evaluated by the sum over the Gauss points for the product:

$$\mathbf{k}_e = \sum_{Gauss\ points} \mathbf{B}^T_e * \mathbf{E}_e * \mathbf{B}_e * \det \mathbf{X}_z$$

where : $\mathbf{E}_e$ is constant over the element and

the values of $\mathbf{B}_e$ and det $\mathbf{X}_z$ are evaluated at each Gauss point

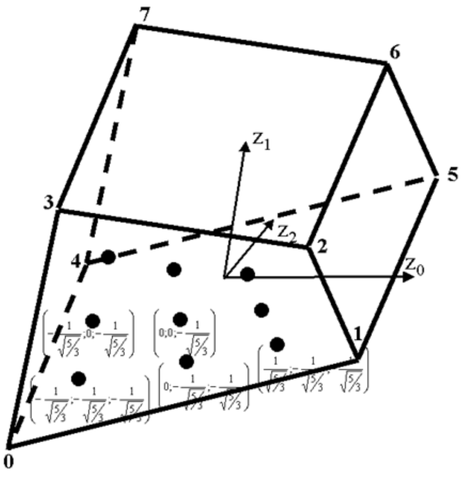
For the case of the linear element with 8 nodes, an order of integration of 3×3x3 is sufficient. The corresponding Gauss points p(z_0, z_1, z_2) are defined in all 3 directions by the 3 normalized coordinates:

$$-\frac{1}{\sqrt{5/3}} \qquad 0 \qquad \frac{1}{\sqrt{5/3}}$$

and the corresponding Gauss weights are:

$$\frac{5}{9} \qquad \frac{8}{9} \qquad \frac{5}{9}$$

This can be shown for the 3x3 Gauss points in a plane in direction $z_2 = -\frac{1}{\sqrt{5/3}}$:



7
6
5
3
4
2
1
0
z_1
z_2
z_0
$\left(-\frac{1}{\sqrt{5/3}};0;-\frac{1}{\sqrt{5/3}}\right)$
$\left(0;0;-\frac{1}{\sqrt{5/3}}\right)$
$\left(-\frac{1}{\sqrt{5/3}};-\frac{1}{\sqrt{5/3}};-\frac{1}{\sqrt{5/3}}\right)$
$\left(0;-\frac{1}{\sqrt{5/3}};-\frac{1}{\sqrt{5/3}}\right)$
$\left(\frac{1}{\sqrt{5/3}};-\frac{1}{\sqrt{5/3}};-\frac{1}{\sqrt{5/3}}\right)$

normalized coordinates

with the associated Gauss weights equal to

$$\frac{5}{9} \qquad \frac{8}{9} \qquad \frac{5}{9}$$

in both directions

3 Implementation on the basis of the FEM Framework

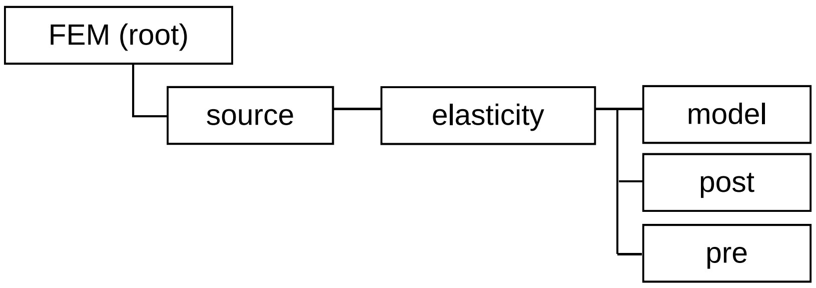
Applications of the FEM for the solution of physical problems are embedded within the context of the general FEM Framework defined before.

In this context, all applications are stored in the hierarchical structure of the FEM Framework under the hierarchy *elasticity*.

Any application must define

- a **main program** for controlling a specific application analysis
  contained in the hierarchy structure of the specific application, i.e. *elasticity*
- a **file parser** for parsing the input file containing the persistently stored model data contained in the hierarchy under *elasticity.pre*
- several classes defining the complete **model** information, which in the case of linear elasticity problems includes element, material, load and support information
  contained in the hierarchy under elasticity.*model*
- a class for **output** of information on the console and optionally on a graphical screen contained in the hierarchy under *elasticity.post*

Any new application is embedded into the hierarchical structure of the FEM Framework in the following way:

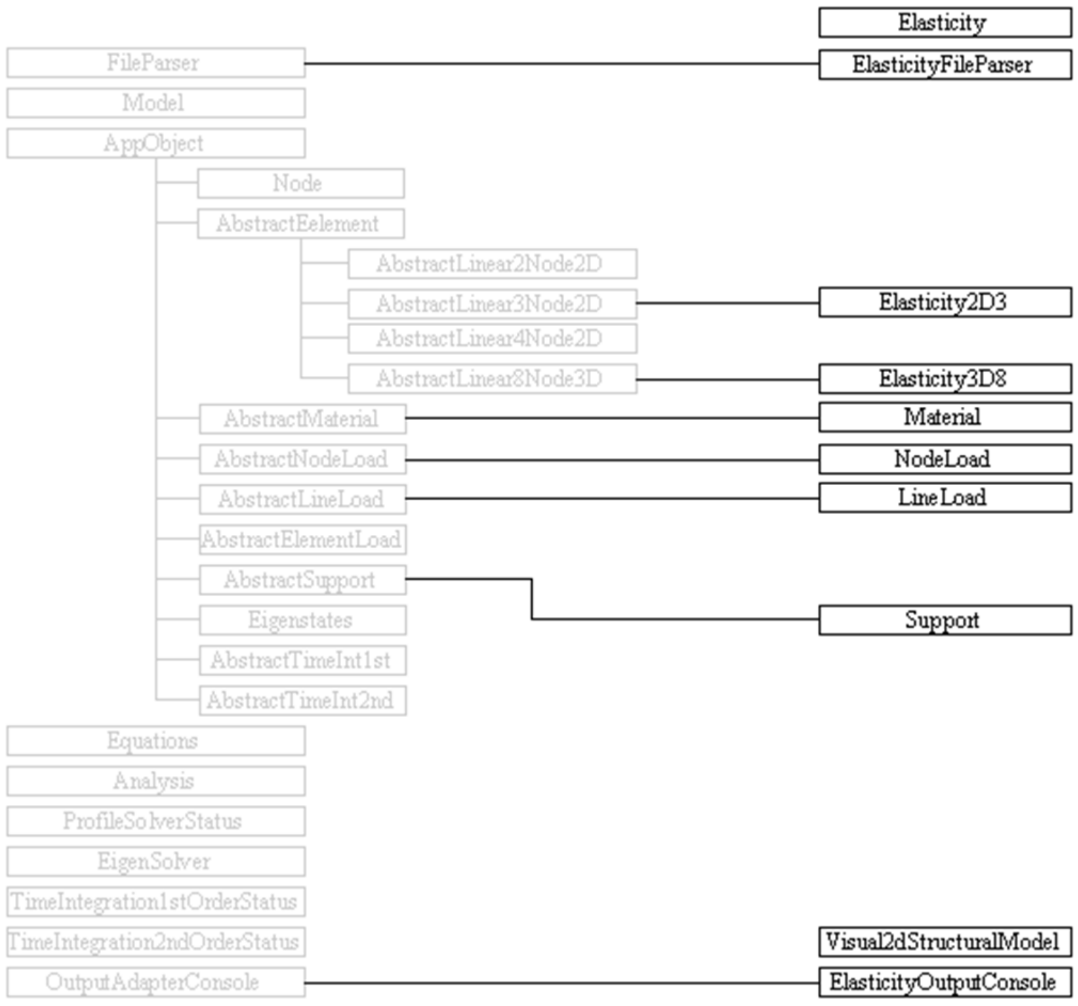


The implementation of classes defining a new application in case of elasticity problems using FEM consists of a file parser for reading a new model, of the application specific element, material, load and support definitions, of the definitions for time integration of the model and of the respective functionality for output of system results.

It is important to note that the mathematical description of the physical problem is almost exclusively restricted to the classes in the model.

The new classes are embedded into the class hierarchy of the FEM Framework in the following way:

Class Definitions (Elasticity) on the basis of the Framework:



3.1 Main program for linear elasticity analysis

The main program for the analysis of problems in linear elasticity is implemented in class Elasticity.

Both applications need to import general Java functionality of File input and output (File IO). They further need to import functionality of the FEM Framework for the model and the analysis and functionality for file handling contained in the FEM utilities. Finally, functionality for parsing input information and displaying model definition and behavior in heat applications need to be imported.

Class Elasticity is contained in package elasticity. It needs to import general Java functionality of File input and File output (File IO), utility functionality for file handling, framework functionality for the model and pre- and post-processing functionality for application elasticity.

Method main starts by invoking method *getInputFile* from utility class *FileHandling*. The selection of applicable input files is accomplished via a file-chooser dialog box. Method *getInputFile* will require two arguments, one for specifying storage location relative to the root-directory, i.e. FEM, and a header definition for the corresponding dialog box.

The essential part of method main is defined by the part that controls the flow of the analysis enclosed in a try and catch-block for possible exceptions generated during analysis.

First, a new model object is generated on the basis of input data as a result from parsing the input file using the *ElasticityFileParser*. A message will report the successful execution of file parsing.

Next, a new object *output* of class *AbstractOutput* is generated for the specific model to be analyzed. Method *output* of this object will allow for checking model input.

Next, a new analysis object is generated for the specific model and method compute*SystemMatrix* of that object is initiated. Subsequently, the system vector is evaluated, the system equations are solved and the results are saved. Successful execution is acknowledged by a corresponding message.

Next, object *output* of class *AbstractOutput* is used for output of results for the model analyzed.

```
public class Elasticity  {
    private static Model model;
    private static Analysis analysis;
    private static AbstractOutput output;

    public static void main(String args[ ]) {
    File file = FileHandling.getInputFile("./input/elasticity", "FEM input files", "inp");
    try {
        // setup and visualize model, analyze model and output results
        model = new ElasticityFileParser(file).getModel();
        output = new ElasticityOutput(model);
        output.outModel();

        analysis = new Analysis(model);
        analysis.computeSystemMatrix();
        analysis.computeSystemVector();
        analysis.solveEquations();

        output.outStationary();
```

3.2 Parsing persistent model information (input file)

Class ElasticityFileParser is provided for parsing of model information that is persistently stored in an input file in Unicode format.

The functionality is contained in *package elasticity.pre*. It needs to import general Java functionality about Java file input and output (IO). It further needs to import functionality of the FEM Framework with respect to *Model* objects, the *FileParser* and corresponding exceptional conditions. It needs to import application specific functionality for FEM components as there are elements, influences, materials and supports. Separate sections of the input file are defined for each individual set of components. Each section is defined by a preceding keyword.

Overview over supported **keywords** and associated **number** of input items in class ElasticityFileParser.

Class *ElasticityFileParser* is derived from class *FileParser* of the Framework and can thus utilize the keywords defined in that class:

```
// public void parseIdentifier() throws IOException, ParseException
//      identifier                           1 modelName

// public void parseDimensions() throws IOException, ParseException
//      dimensions                           2 spatialDimensions, nodalDegreesOfFreedom

// public void parseNodes() throws IOException, ParseException
//      nodes                                1 number of nodal degrees of freedom
//                                           2 name, x
//                                           3 name, x, y
//                                           4 name, x, y, z
//      uniformNodeArray    dimension=1      4 nodeInitial, x, xInterval, nIntervals
//                          dimension=2      7 nodeInitial, x, xInterval, nIntervalsX,
//                                             y, yInterval, nIntervalsY
//                          dimension=3     10 nodeInitial, x, xInterval, nIntervalsX,
//                                             y, yInterval, nIntervalsY,
//                                             z, zInterval, nIntervalsZ
//      variableNodeArray   dimension=1      ? meshSpacings
//                                           2 nodeInitial, x
//                          dimension=2      ? meshSpacings
//                                           3 nodeInitial, x, y
//                          dimension=3      ? meshSpacings
//                                           4: nodeInitial, x, y, z

// public void parseEigenstates() throws IOException, ParseException
//      eigenstates                          2 name, numberOfStates
```

The **additional keywords** defined in **class *ElasticityFileParser*** are defined as follows:

```
//      elasticity2D3          triangular  element in 2D with 3 nodes
//                              6: name, node1, node2, node3, thickness, material
//      elasticity3D8          quadrilateral element in 3D with 8 nodes
//                             10: name, node1, node2, node3, node4, node5, node6, node7,
//                                  node8, material
//      3D8ElementArray        array of quadrilateral elements in 3D with 8 nodes
//                              4: initial, name, nIntervals, material

//      materials               3: name, Young's modulus, Poisson ratio
//                              4: name, Young's modulus, Poisson ratio, shear modulus

//      nodeLoads               4: name, node, valueX, valueY
//                              5: name, node, valueX, valueY, valueZ

//      lineLoads               7: name, startNode, p1x, p1y, endNode, p2x, p2y

//      supports          3,4,5,6: name, node, type(xyz), [preX, preY, preZ]
//      supportFace       5,6,7,8: supportInitial, face, nodeInitial, nNodes, type(xyz),
//                                 [preX, preY, preZ]
//      supportBoussinesq  5: supportInitial, face, nodeInitial, nNodes, type(xyz)
```

Class ***ElasticityFileParser*** provides a constructor which will require the name of the file to be parsed as a parameter and which will throw exceptions if errors occur during input and output or during parsing of information. It will then check if the file to be parsed exists and it will subsequently set the size of the file. It will create an instance of class *BufferedReader* and *FileReader* for parsing the input file.

Finally, it will provide a method for each predefined keyword in order to analyze the contents associated with it.

Class *FileParser* will also provide a method *getModel* for retrieving a model object.

The process of parsing input for a FEM analysis from a file is rather schematic:

- A **keyword** identifies a set of corresponding data (lines) to be read,
- each subsequent line contains a specified **number of arguments** defining the information needed for generating a new FEM component. Each line is analyzed item by item (token by token),
  arguments are separated by predefined separators, i.e. blank, tab, comma
  StringTokenizer tokenizer = new StringTokenizer(s, " \t,");
- the constructor of the new component is invoked with the items as parameters and
- this is carried on until an empty line is encountered completing the particular keyword section.

Once a keyword section is completed, method *reset* will cause reading the input file from the beginning again for the next keyword to be processed.

Method *parseIdentifier* is defined in the framework. Keyword **identifier** requires input of a string for a new model identifier, otherwise an exception will be thrown. The string will be passed to the constructor of a new object of class model.

Method *parseDimensions* is defined in the framework. Keyword **dimensions** requires input of two integer numbers indicating the spatial dimensionality of the problem to be solved and the number of nodal degrees of freedom (e.g. 2 1 defines a 2-dimensional problem with 1 degree of freedom per node).

Method *parseNodes* is also defined in the framework. Keyword **nodes** is used for parsing node components of a FEM model. It throws an exception if a given node identifier already exists (not unique) and if the number of parameters found in the input line is not 1 (spatial dimension), 2 (name and x-value), 3 (name and x,y-values) or 4 (name and x,y,z-values).

Nodes may also be generated. For this purpose two different keywords are available.

Keyword ***uniformNodeArray*** will generate a mesh of equally spaced node components. This is accomplished by entering an initial identifier for all nodes to be generated, followed by starting coordinate(s), the spacing(s) of the mesh and the number of intervals in each direction. This process is supported for 1, 2 and 3 dimensions in which case the starting coordinate, the mesh spacing and the number of intervals need to be specified for each dimension. Unique identifiers for each node will be generated based upon the initial identifier which will then be concatenated with integer numbers for each successive interval, e.g.

uniformNodeArray			
N	0.	2.	10

Generate a 1-dimensional array of 11 nodes at a spacing of 2. units starting from the origin.

New nodes will be generated with respect to the origin specified (0.) Each node will have a unique identifier based upon the initial identifier (N0 through N10), e.g.

uniformNodeArray						
n	0.	1.	3	1.	1.	3

Generate a 2-dimensional array of 3x3 nodes at a spacing of 1. unit starting from the origin (e.g. 0.;1.).

New nodes will be generated with respect to the origin specified (0.;1.) Each node will have a unique identifier based upon the initial identifier (n00, n01, n02, n10, n11, n12, n20, n21, n22).

Mesh generation will start with the second direction followed by the first direction, i.e. n00(0.;0.), n01(0.;1.), n02(0.;3.), n10(1.;0.), ..., n22(3.;3.)

Keyword ***variableNodeArray*** will generate a non-uniformly spaced mesh of nodes based upon a sequence of mesh distances to be entered in the first line after the keyword. The mesh distances to be specified will define a sequence of distances (offset) with respect to the origin (initial coordinates). The number of distances entered will determine the number of nodes to be generated. For this purpose, simply an initial nodal identifier and nodal coordinates for the mesh origin need to be entered, e.g.

variableNodeArray			
0.	1.	3.	6.
N	0.	0.	0.

Will generate an array of nodes at a mesh spacing of 0, 1, 3 and 6 units from the origin for each spatial dimension.

New nodes will be generated with respect to the origin specified (0., 0., 0.) Each node will have a unique identifier based upon an initial identifier (N000, N001, N002, N003, N010 through N333).

Mesh generation will start with the third direction, followed by the second concluding with the first direction, i.e. N000(0.;0.;0.), N001(0.;0.;1.), N002(0.;0.;3.), N003(0.;0.;6.), N010(0.;1.;0.), ..., N333(6.;6.;6.).

Method *parseEigenstates* is used for parsing eigenvalue and eigenvector information of an FEM model. Keyword **eigenstates** requires input of an identifier and the number of eigenstates to be considered.

Application specific file parser:

Information that is specific for a particular application must be parsed via an application specific file parser.

Separate methods are defined for processing application specific input information. Each method starts by looping over the input file and keeps reading new lines until a respective keyword is encountered. Once a particular keyword is found, all subsequent lines are read in sequential order and each line is checked if the number of arguments is according to definitions, otherwise a corresponding exception is thrown. Parsing a keyword section is completed when an empty line is encountered.

This process is repeated for each method invoked and each keyword defined in the application specific file parser. While generally the names of the methods invoked remain unchanged, the contents – applicable keywords – must be adjusted for each application and the looping over the input file is carried out for each of the keywords defined.

Class *ElementParser* is defined for parsing element components of the FEM model. It throws an exception if a given element identifier already exists or if the number of arguments is not in accordance with the keyword.

Keywords **Element2D3, Element3D8** and **3D8ElementArray** are used for parsing element components of a FEM model. For this purpose method *parseElements* is defined. It starts by returning to the top of the file and keeps reading new lines until the corresponding keyword is encountered. It throws an exception if a given element identifier already exists.

Element2D3 keeps reading new lines until an empty line is encountered. For each line read method *parseElements* will read the individual items given in an input line and will check if the number of parameters given is equal to six, otherwise a corresponding exception is thrown. Next, attributes are initialized for the element name and the node and material identifiers. Next, the individual items found in a line will be analyzed.

The first item is interpreted as the string identifier for the element. An exception will be thrown if the identifier already exists as element identifiers must be unique.

The next three items are interpreted as identifiers for the nodes of the triangular element. The last two item are interpreted as thickness of the element and identifier for the element material. After successful processing of the 6 items, a new *Element2D3* object will be generated and added to the model.

Element3D8 keeps reading new lines until an empty line is encountered. For each line read method *parseElements* will read the individual items given in an input line and will check if the number of parameters given is equal to ten, otherwise a corresponding exception is thrown. Next, attributes are initialized for the element name and the node and material identifiers. Next, the individual items found in a line will be analyzed.

The first item is interpreted as the string identifier for the element. An exception will be thrown if the identifier already exists as element identifiers must be unique.

The next eight items are interpreted as identifiers for the nodes of the triangular element. The last item is interpreted as identifier for the element material. After successful processing of the 10 items, a new *Element3D8* object will be generated and added to the model.

3D8ElementArray is defined to generate 8 node elements in 3-dimensional analysis in form of an element array. It reads an initial identifier for the elements to be generated, an initial identifier for the corresponding nodes, the number of intervals and an identifier for the element material. The elements to be generated closely correspond to the keywords *uniformNodeArray* and *variableNodeArray* which will generate corresponding node definitions. The initial element identifier will be concatenated with interval counters for each direction. The initial identifier for the nodes will be concatenated with the corresponding identifiers for the node array.

```
e.g.    3D8ElementArray
        E  N  4  iso          4 intervals of elements starting with E and nodes with N
```

Class *MaterialParser* is defined for parsing material definitions for the FEM model. It throws an exception if a given material identifier already exists or if the number of arguments is not in accordance with the keyword.
Keyword **materials** requires input of either equal to 3 or 4 arguments otherwise a corresponding exception is thrown. The first item is interpreted as the string identifier for the material. An exception will be thrown if the identifier already exists as material identifiers must be unique. The next two items are interpreted as double values defining the modulus of elasticity and Poisson’s ratio. An optional fourth item will be interpreted as the shear modulus needed for the evaluation of the Boussinesq solution of linear elastic half space problems.

Class *InfluenceParser* is defined for parsing nodal load definitions for elasticity elements of the FEM model. It throws an exception if a given node load identifier already exists or if the number of arguments is not equal to 4 or 5.
Keyword **nodeloads** expects input of the node load id, the id of the node the load is applied to and the values of the nodal loads in x-, y- and optionally z-direction.

It also is defined for parsing line load definitions for elasticity elements of the FEM model. It throws an exception if a given line load identifier already exists or if the number of arguments is not equal to 7.
Keyword **lineLoads** expects input of the line load identifier, the identifier of the start node with two load values for x- and y-direction, the identifier of the end again with two load values for x- and y-direction. Load values are assumed to vary linearly between start and end nodes.

Class *BoundaryConditionParser* is defined for parsing boundary (support) conditions of the FEM model. It throws an exception if a given support identifier already exists or if the number of arguments is not in accordance with the keyword.
Keyword **supports** requires input of 3,4,5 or 6 arguments for the identifier of the support condition, the identifier of a supported node, type of support condition and optional predefined boundary values in x-, y- and z-direction.

For 3-dimensional analysis, support definitions may also be generated using the keywords **supportFace** for constraining a complete face in a 3D node array and **supportBoussinesq** for setting 3D displacements at a boundary to the Boussinesq solution.

e.g.	supportFace	
	S X0 N 5 x	support initial, face initial, node initial, number of nodes and optional input for directional constraints
	supportBoussinesq	
	0. 1. 3. 7. 15.	offset of node intervals from an origin
	B X4 N 5 x	support initial, face initial, node initial, number of nodes and type of boundary condition

3.3 Model information for the analysis

Model information for the analysis of problems in linear elasticity consists of the Finite Element approximation defining the element equations, the influences on the model (loads), the constitutive relations (material properties) and the external support conditions.

Furthermore, a specific element must be chosen that is appropriate for the problem to be solved defining the geometry of the problem (element geometry) and the element characteristics for the model.

Each application that is defined in the context of the FEM Framework simply needs to implement the additional application specific functionality for

- the element matrices,
- the material properties,
- the external influences on the model and
- the support conditions for the model.

These classes will be used by the parser in order to generate application specific objects.

3.3.1 establishing element matrices

Functionality that is required for 2-dimensional linear triangular elements in elasticity analysis is collected in **class Elasticity2D3**.

Class Elasticity2D3 is a simple triangular element for linear elasticity analysis in two dimensions. It is named *Element2D3* as generally there could be other element types (e.g. four node or higher order) for the analysis of the same class of problems. It will be derived from *abstract class AbstractLinear3Node2D*.

It is contained in *package.elasticity.model*. It requires to import the functionality of the model and the node implementation from the general FEM Framework. It throws an algebraic exception for some selected algebraic problems (e.g. area of triangle is zero).

```
package elasticity.model;

import java.util.Locale;

import util.MatrixAlgebra;
import framework.model.AlgebraicException;
import framework.model.abstractclasses.AbstractLinear3Node2D;
```

The programming interface of class *Element2D3* provides methods for computing the element matrix, the strain-displacement transformation, the material matrix, the element stresses and for output of element properties and stresses..

```
//public Elasticity2D3(String id, String[ ] eNodes, double thickness, String materialId)
// public double[ ][ ] computeMatrix( )  throws AlgebraicException
// public double[ ] computeDiagonalMatrix( )  throws AlgebraicException
// public void computeStrainDisplacementTransformation( )
// public void computeMaterial()
// public double[ ] computeElementStresses()
// public void printElementProperties()
// public void printElementStresses()
```

Class Elasticity2D3 defines a number of static attributes for

- the number of degrees of freedom per node,
- the element matrix,
- the element deformation vector,
- the element thickness,
- the strain displacement transformation matrix B and
- the material matrix E.

It is not necessary to store all the element information for each individual object of class Element2D3 as this information is only utilized to set up the corresponding system equations. Each new element will thus use the identical memory space for evaluation of the information required for the analysis, e.g. calculating the element stiffness matrices. Element information will be regenerated when needed again.

The constructor will generate new elements on the basis of the string identifier for that element, the identifiers of the 3 element nodes, the element thickness and the material identifier.

```
public class Elasticity2D3 extends AbstractLinear3Node2D  {
    static final int ELEMENT_DOF_PER_NODE = 2;
    static double matrix[ ][ ] = new double[6][6];
    static double elementDeformations[ ] = new double[6];  // at element nodes
    static double thickness;
    static double B[ ][ ] = new double[3][6];        // strain-displacement transformation
    static double E[ ][ ] = new double[3][3];            // material matrix

    // ....Constructor.............................................
    public Elasticity2D3(String id, String[ ] eNodes, double thick, String materialId) {
        super(id, eNodes, materialId, ELEMENT_DOF_PER_NODE);
        thickness = thick;
    }
```

At the he heart of the element implementation is a method for computing the **element matrix** (*computeMatrix*). It starts by evaluation the element geometry (*computeGeometry*) and by evaluating the strain displacement transformation. It next determines the material properties for the current element.The element matrix is then computed by evaluating the product $\boldsymbol{K}_e$ = *area*thickness**$\boldsymbol{B}_e^T$***$\boldsymbol{E}$***$\boldsymbol{B}_e$.

```
    public double[ ][ ] computeMatrix() throws AlgebraicException {
        computeGeometry();
        computeStrainDisplacementTransformation();
        computeMaterial();
        double[ ][ ] temp = new double[6][3];
        // Ke = 0.5*t*determinant*BT*E*B
        temp=MatrixAlgebra.multTransposedMatrix
                                        (temp,0.5*thickness*determinant, B, E);
        matrix = MatrixAlgebra.mult(matrix, temp, B);
        return matrix;
    }
```

Next, the **strain displacement** Matrix $\mathbf{B}_e$ must be evaluated and the **material** matrix $\mathbf{E}_e$ for plane stress analysis.

```
// compute strain-displacement transformation matrix eps = B * u
public void computeStrainDisplacementTransformation()  {
     B[0][0]= Xzu[1][1];   B[0][1]= 0.;                   B[0][2]=-Xzu[1][0];
     B[0][3]= 0.           B[0][4]=Xzu[1][0]-Xzu[1][1];   B[0][5]= 0.;
     B[1][0]= 0.           B[1][1]=-Xzu[0][1];            B[1][2]= 0.
     B[1][3]= Xzu[0][0];   B[1][4]= 0.                    B[1][5]=Xzu[0][1]-Xzu[0][0];
     B[2][0]=-Xzu[0][1];   B[2][1]= Xzu[1][1];            B[2][2]= Xzu[0][0];
     B[2][3]=-Xzu[1][0];   B[2][4]=Xzu[0][1]-Xzu[0][0];   B[2][5]=Xzu[1][0]-Xzu[1][1];
}
// compute material matrix for plane strain
public void computeMaterial()  {
     double emod = ((Material)material).getModulusOfElasticity();
     double ratio = ((Material)material).getPoissonRatio();
     double factor = emod*(1.-ratio)/((1.+ratio)*(1.-2.*ratio));
     double coeff =  ratio/(1.-ratio);
     E[0][0] = factor;         E[0][1] = coeff*factor; E[0][2] = 0.;
     E[1][0] = coeff*factor;   E[1][1] = factor;       E[1][2] = 0.;
     E[2][0] = 0.;             E[2][1] = 0.;           E[2][2] = (1.-2.*ratio)/2./
                                                                 (1.-ratio)*factor;
}
```

The **element behavior** (element stresses) within an element is computed from the equation:

$$\mathbf{s}_e = t_e * \boldsymbol{\sigma}_e = t_e * \mathbf{E}_e * \varepsilon_e = t_e * \mathbf{E}_e * \mathbf{B}_e * \mathbf{u}_e$$

Method *computeElementStresses* uses attributes *node*, a vector for the element stresses, the number of nodal degrees of freedom and two temporary attributes (temp, sum) for performing the operations. The element deformations are retrieved for each node and the matrix multiplications are performed.

```
// ....Compute element stresses: sigma = E * B * Ue (element deformations)
public double[ ] computeElementStresses()  {
     double[ ] elementStresses = new double[3];
     double[ ][ ] temp = new double[3][6];
     int nodalDOF;
     for (int i = 0; i < 3; i++) {
          nodalDOF = i*2;
          elementDeformations[nodalDOF]     = node[i].getNodalDOF()[0];
          elementDeformations[nodalDOF+1] = node[i].getNodalDOF()[1];
     }
     temp = MatrixAlgebra.mult(temp, E, B);
     elementStresses=MatrixAlgebra.mult(temp, elementDeformations);
     return elementStresses;
}
```

Functionality that is required for 3-dimensional linear 8-node brick elements in elasticity analysis is collected in **class Elasticity3d8**.

Class Elasticity3d8 is named *Elasticity3D8* as generally there could be other element types (e.g. four node or higher order) for the analysis of the same class of problems. It will be derived from *abstract class AbstractLinear8Node3D*.

It is contained in *package elasticity3d.model*. It requires to import the functionality of the model and the node implementation from the general FEM Framework. It throws an algebraic exception for some selected algebraic problems (e.g. area of triangle is zero).

The programming interface of class *Elasticity3d8* provides methods for defining the node identifiers of the element nodes, for computing the element matrix, for computing the heat state at the midpoint of the elements and for printing element properties and behaviour.

3.3.2 defining material properties

The material description in elasticity problems problems is given in general by the **elasticity matrix E**. The elasticity matrix is needed for the evaluation of the element matrices as described before and generated for these purposes in the element evaluation.

It relies on two separate material properties to be defined and stored:

E	Young's modulus of elasticity and
ν	Poisson's ratio

Class Material is contained in package *elasticity.model* and imports class *model.implementation.AbstractMaterial*. It is derived from class *AbstractMaterial*.

It simply consists of a constructor for passing the material identifier to the model and for storing the material property under names "emodulus" and "poisson" and it finally provides a method for retrieving the material properties (*getModulusOfElasticity* and *getPoissonRatio*).

```
package elasticity.model;

import java.util.Locale;

import framework.model.abstractclasses.AbstractMaterial;

public class Material extends AbstractMaterial {
    public Material(String id, double emodulus, double poisson ) {
        super(id);
        set("emodulus",emodulus);
        set("poisson",poisson);
    }
    // ....get()........................................................
    public double getModulusOfElasticity() { return get("emodulus"); }
    public double getPoissonRatio() { return get("poisson"); }
}
```

3.3.3 defining external influences on the system

External influences on the model can either be defined as

- nodal influences
  are concentrated loads at the nodes of the element mesh. The intensity of the load is given. The intensity is positive in positive coordinate directions.
- line influences
  are line loads between two neighbouring nodes of the element mesh. The line load intensities are defined. If the line load intensity varies linearly along the line, the intensities at the end nodes are given.
- area influences
  are surface forces with a given load intensity. Surface forces are allowed to vary linearly across the element, its value are given at each of the three nodes of the element.

Class ***NodeLoad*** contains the implementation of **nodal influences**. These are simply defined by specifying the load intensity at a specific node in both directions.

Line influences are specified in general by the relation between elasticity input at boundary $q_{r(i)}$ and the concentrated elasticity problems at node i: Q_{Ri}

$$\begin{pmatrix} Q_{R1x} \\ Q_{R1y} \\ Q_{R2x} \\ Q_{R2y} \end{pmatrix} = \frac{L}{6} \begin{pmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} p_{1x} \\ p_{1y} \\ p_{2x} \\ p_{2y} \end{pmatrix} \tag{3.55}$$

The implementation of the line load is contained in *package elasticity.model*. It needs to import class *AbstractLineLoad* from *package model.abstractclasses* of the general FEM Framework. It will be derived from *abstract class AbstractLineLoad*.

Class LineLoad defines a constructor for a linearly varying line load. The method for specifying the intensity of the line load (*setLoads*) will expect the nodal values of the load intensity as input parameters.

The method for computing the load vector (*computeLoadVector*) will evaluate the element load vector based upon equations given above. For this purpose, the two nodes of the line load need to be retrieved, the corresponding coordinates need to be evaluated, the length of the edge between the two nodes will be computed and the element load vector will be returned.

Area influences are not supported in this implementation.

3.3.4 setting up support conditions

Support conditions for the model can be defined for each node. Since nodes in elastic analysis have two or three degrees of freedom, the prescribed values at a node need to be defined and the system status vector needs to be adjusted.

Class Support is contained in *package .elasticity.model*. It is derived from class *AbstractSupport* which needs to be imported from *package model. implementation*. It also needs to implement interface *model.interfaces.ISupport*.

Each support object provides attributes for the types of support conditions (x-, y- and z-restraints) with all possible combinations of support restraints, for the reaction and prescribed values and if a directional deformation is constrained.

Methods *getReaction*, *getPrescribed* and *getRestraint* will return the values for the reactions for the prescribed nodal values and if a directional deformation is constrained.

Method *setPrescribed* defines if a nodal degree of freedom is restraint and if it has a prescribed value of displacement. Method (*setReactions*) stores the external support forces.

Methods *printSupportConditions* and *printSupportReactions* provide the required output functionality.

3.4 Output of system results

Output functionality as a result of the analysis of linear elasticity problems is implemented in two different classes, one for alphanumeric console output (ElasticityOutputConsole) and another one (VisualElasticity2DModel) for output in a graphical frame (window Elasticity2D Visualizer) created with Java Swing technology

3.4.1 Alphanumeric output of results

Class *ElasticityOutputConsole* is contained in package elasticity.output. It requires to import general Java functionality (*interface Iterator* of the Java Collections Framework) for iterating Java collections.

It further needs to import functionality of the FEM Framework for the model, i.e. FEMException, Node and interface OutputAdapterConsole. It also needs to import general utility functionalities for input and output of information.

It finally, needs to import the specific FEM components of the application model, i.e. element, load, material and support.

```
package elasticity.post;

import java.util.Iterator;
import java.util.Scanner;

import elasticity.model.Elasticity2D3;
import elasticity.model.Elasticity3D8;
import elasticity.model.LineLoad;
import elasticity.model.Material;
import elasticity.model.NodeLoad;
import elasticity.model.Support;
import framework.model.FemException;
import framework.model.Model;
import framework.model.Node;
import framework.post.OutputAdapterConsole;
```

Class *ElasticityOutputConsole* has attributes for object iteration – object *iter* of interface *Iterator*, for the status of the analysis process (equations solved, eigensolution performed, time integration done) and for scanning console input.

Method *output* will control the flow of output of information interactively requested by the user. It will present five different options for the output of the model, output of the influences (loads), for the visualization of model, influences and results, for starting the analysis of the model and for terminating the program. Each option is selected by entering the first character of the option on the console input. If the analysis has been performed already (solved, eigen or timeint are true), output of the system results is initiated directly.

Method *printModel* provides functionality for output of node, element, support and material definitions of the model to be analyzed. Method *printInfluences* provides functionality for output of influences (loads) on the model. Method *printBehaviour* provides functionality for output of nodal deformations, element stresses, support reactions and for visualizing results.

It also will output results of an eigensolution and visualize time history results for each nodal degree of freedom.

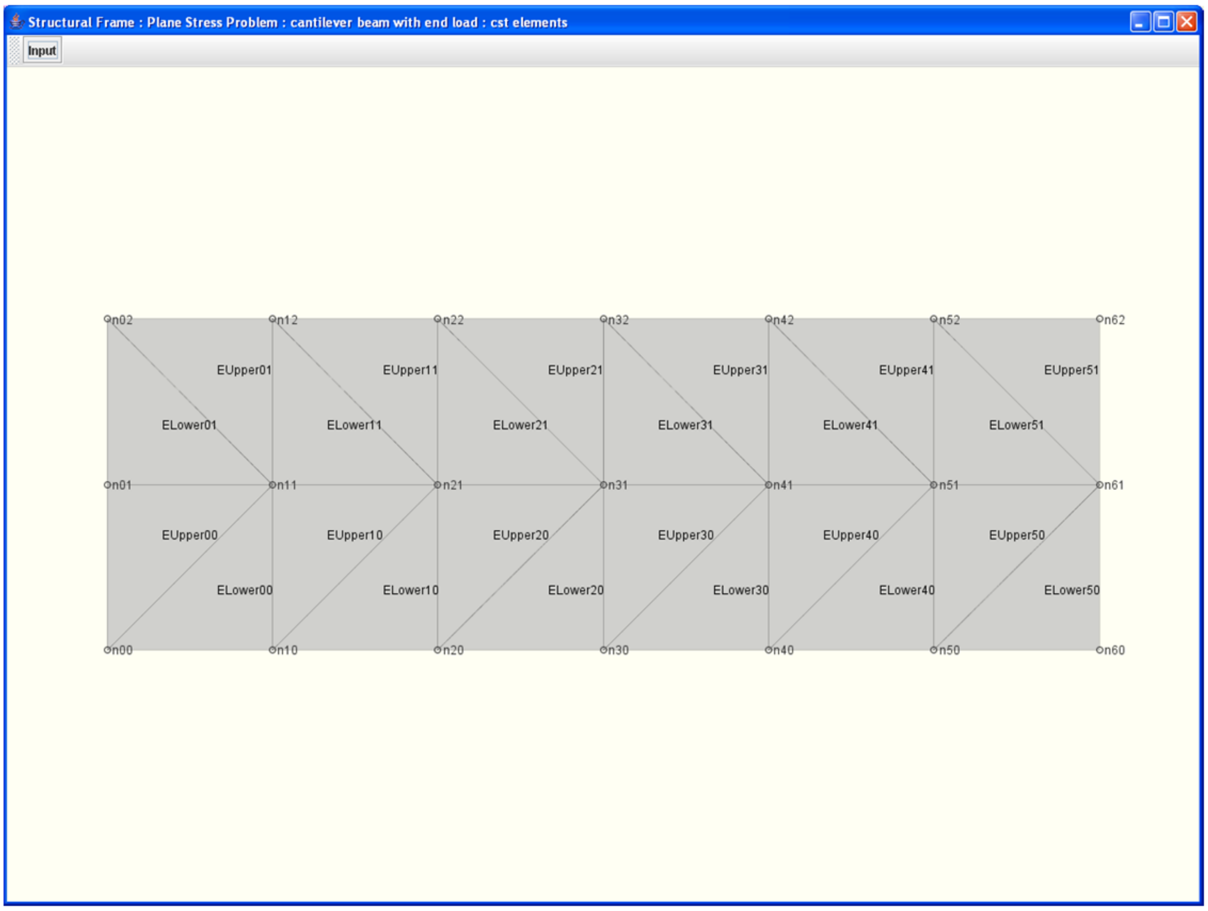
The general process for output of information is schematic. An iterator is set up for the type of class of the corresponding information, a header is printed, all objects of the specific class are traversed and the corresponding output functionality is invoked. This is demonstrated for output of element properties of 3-node elements in 2D and 8-node elements in 3D:

```
else if (cc == 'e') {
    System.out.println("\nELEMENT PROPERTIES\n")

    Elasticity2D3 element3;
    iter = m_model.iterator(Elasticity2D3.class);
    if (iter.hasNext())
         System.out.println("Element   Node1   Node2   Node3   Material");
    while (iter.hasNext()) {
         element3 = (Elasticity2D3) iter.next();
         element3.printElementProperties();
    }
    Elasticity3D8 element8;
    iter = m_model.iterator(Elasticity3D8.class);
    if (iter.hasNext())
         System.out.println("Element  Node0  Node1  Node2  Node3  Node4  Node5
                                                   Node6  Node7  Material");
    while (iter.hasNext()) {
         element8 = (Elasticity3D8) iter.next();
         element8.printElementProperties();
    }
}
```

3.4.2 Graphical output of results

Class *VisualElasticity2DModel* is also contained in package elasticity.output. It is implemented solely on the basis of Java Swing technology. For the purposes of this text only the visualization of the input data will be demonstrated.



Literaturverzeichnis

[1] The Finite Element Method, Its Basis & Fundamentals; O.C. Zienkiewicz, R.L. Taylor, J.Z. Zhu, 6th Edition, Elsevier 2005.

[2] Pahl, P. J.: Finite Element Method, Stationary Heat Flow, TU Berlin, November 2000

[3] Pahl, P. J.: Finite Element Methoden für Physikalische Aufgaben, TU Berlin, Oktober 1991

[4] Sun Microsystems, Inc., Copyright 2006: Download The Java Tutorial, http://java.sun.com/docs/books/tutorial/.

[5] http://www.cademia.org, © 2006, Bauhaus-Universität Weimar | Lombego Systems

[6] http://www.opensource.org