Dept. of Math. University of Oslo
Mechanics and Applied Mathematics
ISBN 82-553-1309-5 No. 5
ISSN 0809-4403 October 2001

Implementation of a Geometrically Nonlinear Module for Composite Materials in Diffpack

Harald Osnes

Mechanics Division, Dept. of Mathematics,
University of Oslo, P.O. Box 1053 Blindern,
N-0316 Oslo, Norway

October 17, 2001

Abstract

In the present report we describe the geometrically nonlinear method for orthotropic (composite) materials that is implemented using the commercial C++ library *Diffpack*. The implementation is based on a module dealing with linear elasticity for isotropic materials, and the modifications performed to extend the functionality to handle orthotropic materials are explained. Analysis of composite joints is one of the main aims of the module. Such joints typically fail for laminate strains in the order of 1-2%, which indicates that sufficient accuracy may be obtained by a geometrically linear procedure. However, for non-symmetric geometries, e.g. single lap joints, geometrically nonlinear effects become crucial for loads far below ultimate strength limits. Thus, in order to be able to analyse such joints accurately, the code is extended with functionality capturing geometrically nonlinear effects.

In addition to the description of the extended functionality, the report covers some verification of the code. It is tested on a couple of examples, showing reasonable results in all cases. It should also be mentioned that in a previous study solutions obtained by the present module were in excellent agreement with *ANSYS* results.

1 Introduction

The use of fiber composites has shown a tremendous growth in many fields during the last decades. The application of composite materials range from trivial, industrial products such as boxes and covers produced in enormous numbers each day to pipelines and large, crucial, load bearing parts of constructions. Composites are also extensively used in the aerospace and marine industries

[6, 13]. Important reasons for this popularity are the high strength (and stiffness) to weight ratio, the possibility of controlling the anisotropy and the fact that fiber composites are resistant to corrosion. The possibility of making products of almost any geometry is also of great advantage. As a result of this rare combination of properties and in spite of design difficulties, composites have been used more and more frequently in various combinations and situations over the last years [1].

Before starting the production of a new component or structure one has to be sure that it will meet certain functional requirements, e.g. that it will work properly and not fail when exposed to the environmental conditions or loads it will experience during service life. Due to the relatively complicated nature of composite materials in general and the production process (including the number of layers, lay-up, adhesives, joints etc.) in particular various experimental testing plays an important role when such components are verified/qualified. However, this flexibility, and the enormous ways in which composite parts may be constructed and produced ensure that it is impossible to make a complete test program covering all combinations of design, production and loads. Therefore, theoretical predictions are crucial in the design and optimization process of composite components. Such theoretical models may be divided into two main categories; analytical and numerical methods. While numerical methods may be applied in a large number of applications, most analytical predictions are restricted to simple geometries and certain choices of lay-up. Most commercial software packages for applications in solid mechanics provide functionality for composite (or other kinds of orthotropic) materials.

In the present report we will discuss the implementation of a geometrically nonlinear method for composite (or orthotropic) materials in the commercial C++ library *Diffpack* [5]. One may question the value of implementing a module that is available in *ANSYS* [3] and almost all other commercial packages for solid mechanics. However, in a research context it is always valuable to have a code in which all programming details are known. Self-implemented codes often provide important insight into crucial modelling aspects. Finally, the *Diffpack* code will be extremely useful when new functionality, e.g. various failure analysis methods, is required. In such cases it is much simpler to extend the *Diffpack* code than trying to add similar functionality to existing software packages.

The report will be organized as follows. In section 2 the *Diffpack* implementation of the module dealing with isotropic linear elasticity will be briefly presented. The extensions required in order to solve orthotropic problems will be provided by section 3, while section 4 presents the geometrically nonlinear procedure implemented. In section 5 a couple of numerical examples are solved, and section 6 is devoted to some discussion and concluding remarks. Finally, some tedious derivations of material stiffness matrices for orthotropic materials and the entire code implemented are included in the appendices A and B,

respectively.

2 Linear Elasticity in *Diffpack*

In the following we present some of the main features of the isotropic, linear elasticity module in *Diffpack*. For a comprehensive treatment of aspects related to elastic deformations we refer to texts on continuum mechanics, e.g. [7, 8, 11, 12, 14].

The equilibrium equation for elastic media may be expressed by

$$\sigma_{rs,s} = -\varrho b_r, \quad r = 1, \ldots, d, \tag{1}$$

where σ_{rs} is the rs component of the (Cauchy) stress tensor, ρ is the density of the medium, b_r represents the r component of the body forces (per unit mass), e.g. gravity, and d is the number of space dimensions. The index after the comma (in $\sigma_{rs,s}$) means that σ_{rs} is differentiated with respect to component s of the spatial coordinates, i.e. x_s. The summation convention is applied (for all terms with repeated indices).

The governing equation (1) yields three scalar equations (in 3-D) that introduce six unknown stress components (the stress tensor is symmetric, and ρ and b_r are prescribed quantities). Therefore, in order to close the system, more relations have to be introduced. For elastic materials certain expressions regarding the stresses and strains exist. The components of the strain tensor are given by

$$\varepsilon_{rs} = \frac{1}{2}\left(u_{r,s} + u_{s,r}\right), \quad r, s = 1, \ldots, d, \tag{2}$$

where u_r represents component r of the displacement field. The relation between the stress and strain tensors for isotropic materials (the generalized Hooke's law) may be expressed as

$$\sigma_{ij} = C_{ijrs}\varepsilon_{rs}, \quad i, j = 1, \ldots, d, \tag{3}$$

where C_{ijrs} is defined by

$$C_{ijrs} = \lambda\delta_{ij}\delta_{rs} + \mu\left(\delta_{ir}\delta_{js} + \delta_{is}\delta_{jr}\right). \quad i, j, r, s = 1, \ldots, d, \tag{4}$$

Here, λ and μ are the elasticity coefficients of Lamé, and δ_{ij} is the Kronecker delta. These quantities are given by

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)},$$

$$\mu = \frac{E}{2(1+\nu)},$$

$$\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}.$$

E and ν are the Young's modulus and Poisson's ratio, respectively. By inserting (2) and (3) into the equilibrium equation (1) three partial differential equations may be obtained for the displacement components.

In the context of the finite element method (FEM) one assumes that the displacement field u_r is approximated by

$$\hat{u}_r = \sum_{j=1}^{n} u_j^r N_j(x_1, \ldots, x_d), \quad r = 1, \ldots, d, \tag{5}$$

where N_j is the prescribed test function corresponding to node j and u_j^r is the approximation of the r component of the displacement in node j. Thus, the total number of unknown displacement parameters is $n \cdot d$. In the standard Galerkin FEM, which is adopted here, the weighting functions are identical to the test functions. After multiplying (1) by N_i and integrating by parts (or using Gauss' theorem) we obtain the weak formulation

$$\int_{\Omega} \sigma_{rs} N_{i,s} \mathrm{d}\Omega = \int_{\partial\Omega} N_i \sigma_{rs} n_s \mathrm{d}\Gamma + \int_{\Omega} \rho b_r N_i \mathrm{d}\Omega, \quad i = 1, \ldots, n, \tag{6}$$

where Ω and $\partial\Omega$ are the volume and boundary of the elastic material, respectively, and n_s is the s component of the unit outward normal vector. By substituting (3) into (6) and introducing the approximation (5) one obtains a particular finite element formulation of isotropic, linear elastic problems. However, in the following we will adopt a formulation (the so called engineering finite element formulation) that is more suitable for extensions dealing with, e.g, orthotropic materials. Also geometrically nonlinear effects can be taken into account without the need for a complete restructuring of the code. In this formulation, the stress and strain tensors are exchanged by vectors containing similar components. The new stress and strain vectors read

$$\boldsymbol{\sigma} = (\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{yz}, \sigma_{zx})^T \tag{7}$$

and

$$\boldsymbol{\varepsilon} = (\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}, \gamma_{xy}, \gamma_{yz}, \gamma_{zx})^T, \tag{8}$$

where one applies engineering shear strains, e.g. $\gamma_{xy} = 2\varepsilon_{xy}$. Hooke's law can now be expressed as

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\varepsilon}, \tag{9}$$

with

$$\boldsymbol{D} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1 & \frac{\nu}{1-\nu} & \frac{\nu}{1-\nu} & 0 & 0 & 0 \\ & 1 & \frac{\nu}{1-\nu} & 0 & 0 & 0 \\ & & 1 & 0 & 0 & 0 \\ & & & \frac{1-2\nu}{2(1-\nu)} & 0 & 0 \\ & \text{symmetric} & & & \frac{1-2\nu}{2(1-\nu)} & 0 \\ & & & & & \frac{1-2\nu}{2(1-\nu)} \end{pmatrix} \tag{10}$$

The expressions above, (7), (8) and (10), are valid for 3-D problems.

In many applications 3-D problems may be well approximated by 2-D analysis methods. For example, for thin plates where the through thickness stresses can be neglected, a plane stress model may be adopted, and in problems where all strain components in one direction are small compared to the others, a plane strain analysis method can be applied. These 2-D models may also be handled by the constitutive law (9). However, the stress and strain vectors now read

$$\boldsymbol{\sigma} = (\sigma_{xx}, \sigma_{yy}, \sigma_{xy})^T, \qquad \boldsymbol{\varepsilon} = (\varepsilon_{xx}, \varepsilon_{yy}, \gamma_{xy})^T, \tag{11}$$

while the $\boldsymbol{D}$ matrix is given by

$$\boldsymbol{D} = \frac{E}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix} \tag{12}$$

for plane stress and

$$\boldsymbol{D} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{pmatrix} \tag{13}$$

for plane strain.

Before substituting Hooke's law (9) into the weak formulation (6) let us introduce the relation between the finite element displacement field and the strain vector. With vector notation, the approximate displacement field (5) is given by

$$\hat{\boldsymbol{u}} = \sum_{j=1}^{n} \boldsymbol{u}_j N_j(x_1, \ldots, x_d), \tag{14}$$

where $\boldsymbol{u}_j$ is the displacement vector at node j. The corresponding approximation for the strain vector may now be expressed as

$$\boldsymbol{\varepsilon} = \sum_{j=1}^{n} \boldsymbol{B}_j \boldsymbol{u}_j, \tag{15}$$

with

$$\boldsymbol{B}_j = \begin{pmatrix} N_{j,x} & 0 & 0 \\ 0 & N_{j,y} & 0 \\ 0 & 0 & N_{j,z} \\ N_{j,y} & N_{j,x} & 0 \\ 0 & N_{j,z} & N_{j,y} \\ N_{j,z} & 0 & N_{j,x} \end{pmatrix}. \tag{16}$$

In plane stress and plane strain problems, where only the x and y components of the velocity field $\boldsymbol{u}_j$ enter the equations, the $\boldsymbol{B}_j$ matrix reads

$$\boldsymbol{B}_j = \begin{pmatrix} N_{j,x} & 0 \\ 0 & N_{j,y} \\ N_{j,y} & N_{j,x} \end{pmatrix}. \tag{17}$$

Substituting (9) and (15) into (6) results in the weak Galerkin formulation

$$\sum_{j=1}^{n} \int_{\Omega} \boldsymbol{B}_i^T \boldsymbol{D} \boldsymbol{B}_j \mathrm{d}\Omega \, \boldsymbol{u}_j = \int_{\partial\Omega} N_i \boldsymbol{t} \mathrm{d}\Gamma + \int_{\Omega} N_i \rho \boldsymbol{b} \mathrm{d}\Omega, \quad i = 1, \ldots, n, \tag{18}$$

where $\boldsymbol{t}$ is the traction vector ($t_r = \sigma_{rs} n_s$) and $\boldsymbol{b}$ is the body force vector. In most codes dealing with FEM the contributions from (18) are calculated at the element level before assembled into the system stiffness matrix and load vector. In the context of engineering mechanics the system of algebraic equations (18) is often written as

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}, \tag{19}$$

where $\boldsymbol{K}$ is the global stiffness matrix, $\boldsymbol{u}$ is the vector of unknown nodal displacements and $\boldsymbol{f}$ is the vector of nodal loads.

Details of the implementation of the elasticity module using *Diffpack* is given by Langtangen [10]. However, some crucial aspects will be briefly discussed in the following. The elasticity module is defined by the class `Elasticity1`, which is inherited from the library class `FEM` in which all general aspects of the finite element method are defined. The `Elasticity1` class contains the required data structure, e.g. the vector of unknown displacement values and derived quantities such as stresses and strains, as well as a large number of useful member functions. These functions handles, e.g., the input/output of data, the boundary conditions, the solution procedure and the calculations of derived quantities. One of the most important functions in the module is the `integrands` function. Here, the volume integrals in the weak formulation, which defines the (linear) algebraic equation system to be solved, are evaluated at the integration (Gauss) points of each finite element. The contribution from the boundary integral is handled by the function `integrands4side`. In the original version [10] the `integrands` function applies Hooke's law on a form similar to (4). However, the code has been modified, by the present author, to rely on the weak formulation given by (18). In this case the `integrands` procedure calls functions that evaluate the $\boldsymbol{B}_i$ and $\boldsymbol{D}$ matrices at the present integration point. These matrices are subsequently multiplied and integrated according to the left hand side of (18), and the result is put into the coefficient matrix of the final algebraic equation system. There are several advantages connected to this strategy. All parameters describing the elasticity properties of the material are included in the $\boldsymbol{D}$ matrix, while the $\boldsymbol{B}_i$ matrices depend on the test functions used and the geometry of the elements. Thus, as long as we restrict ourselves to (geometrically and materially) linear problems, the only modification required when extending the analysis from isotropic to orthotropic or anisotropic materials is to reimplement the procedure that calculates the $\boldsymbol{D}$ matrix. In the next section, which is devoted to the analysis of composite (orthotropic) materials, we will derive expressions for an extended $\boldsymbol{D}$ matrix. It should also be mentioned that the present formulation is a suitable basis for

development of various nonlinear methods, which is the topic of a later section. Furthermore, the possibility of performing 2-D plane stress, 2-D plane strain and 3-D analysis is of no concern in the `integrands` function. This is dealt with in the functions that evaluate the $\boldsymbol{B}_i$ and $\boldsymbol{D}$ matrices. Finally, it should be mentioned that the terms on the right hand side of (18) are trivially evaluated in the `integrands` and `integrands4side` procedures.

3 Composite Materials

In the present section we will discuss the extensions required in order to extend the module presented in the previous section for isotropic, linear elastic media to orthotropic materials. A general 3-D, orthotropic material has three mutually perpendicular axes of symmetry. It can be shown, see e.g. Agarwal et.al. [1], that the number of independent elastic coefficients needed to describe an orthotropic media is nine. Thus, it lies somewhere in between isotropic and anisotropic materials, which require two and twenty-one independent coefficients, respectively. One important type of orthotropic materials is uni-directional composites, which contain continuous, uni-directional fibres (e.g. glass or carbon) embedded in a matrix material (which is often isotropic, e.g. polyester or epoxy). Such composites provide high stiffness (and strength) in the longitudinal directions (along the fibres) and low stiffness (and strength) in the transverse direction. This is due to the fact that the longitudinal material properties are controlled by the fibres, whereas the transverse properties are matrix dominated. In most engineering applications the transverse properties of uni-directional composites are found to be unsatisfactory. This apparent limitation on the use of purely uni-directional composites is overcome by forming laminates from uni-directional layers. A laminate is formed from two or more uni-directional laminae bonded together to act as an integral structural element, and the laminae are oriented to produce a structural element with the desired properties in all directions. This implies that the laminate may contain fibres oriented in a number of different directions. Thus, although all the laminae are identical, their elastic properties related to a fixed, global coordinate system may vary due to the different layer orientation. Therefore, to ensure well-defined material properties in all the solid elements applied in the finite element module to be presented, it is required that one element can only cover a single layer (unless several elements have identical material properties and fibre orientation). However, it is possible to apply several elements through the thickness of each layer.

As was explained in the previous section, the major modification that has to be performed to extend the isotropic module to be able to handle orthotropic materials is to reimplement the procedure that calculates the $\boldsymbol{D}$ matrix, which defines the relation between stresses and strains.

For a 3-D orthotropic material the $\boldsymbol{D}$ matrix may be written

$$\boldsymbol{D} = \begin{pmatrix} D_{11} & D_{12} & D_{13} & 0 & 0 & 0 \\ D_{12} & D_{22} & D_{23} & 0 & 0 & 0 \\ D_{13} & D_{23} & D_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & D_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & D_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & D_{66} \end{pmatrix} \tag{20}$$

when the coordinate axes are coincident with the axes of material symmetry. However, as we have seen, the laminae in a laminate may be oriented in different directions. It is therefore generally impossible to express the $\boldsymbol{D}$ matrix in accordance with (20) for all layers when referred to a single global coordinate system. To overcome this problem the stiffness matrix for each layer is first expressed as above with respect to a local coordinate system with axes in coincidence with the material axes of symmetry. Then the $\boldsymbol{D}$ matrices are transferred to represent elasticity properties related to a fixed, global coordinate system by pre- and post-multiplication with certain transformation matrices. This will be explained in the following. The relation between stresses and strains for a layer may be given by

$$\begin{pmatrix} \sigma_{LL} \\ \sigma_{TT} \\ \sigma_{\hat{T}\hat{T}} \\ \sigma_{LT} \\ \sigma_{T\hat{T}} \\ \sigma_{\hat{T}L} \end{pmatrix} = \boldsymbol{D} \begin{pmatrix} \varepsilon_{LL} \\ \varepsilon_{TT} \\ \varepsilon_{\hat{T}\hat{T}} \\ \gamma_{LT} \\ \gamma_{T\hat{T}} \\ \gamma_{\hat{T}L} \end{pmatrix}, \tag{21}$$

where the layer stiffness matrix $\boldsymbol{D}$ is given by (20) and the subscripts L, T and $\hat{T}$ denote the fibre direction, transverse in-plane direction and the transverse through thickness direction, respectively, of the lamina. These local stress and strain vectors may be related to vectors expressed with respect to a global (xyz) coordinate system. From [1] it is seen that

$$\begin{pmatrix} \sigma_{LL} \\ \sigma_{TT} \\ \sigma_{\hat{T}\hat{T}} \\ \sigma_{LT} \\ \sigma_{T\hat{T}} \\ \sigma_{\hat{T}L} \end{pmatrix} = \boldsymbol{T} \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix}, \tag{22}$$

where $\boldsymbol{T}$ is a transformation matrix that will be defined below. The correspond-

ing relation between local and global strains reads

$$\begin{pmatrix} \varepsilon_{LL} \\ \varepsilon_{TT} \\ \varepsilon_{\hat{T}\hat{T}} \\ \frac{1}{2}\gamma_{LT} \\ \frac{1}{2}\gamma_{T\hat{T}} \\ \frac{1}{2}\gamma_{\hat{T}L} \end{pmatrix} = \boldsymbol{T} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \frac{1}{2}\gamma_{xy} \\ \frac{1}{2}\gamma_{yz} \\ \frac{1}{2}\gamma_{zx} \end{pmatrix}. \tag{23}$$

Remark that a factor 1/2 is required for the transformation of the engineering shear strains. A slightly modified relation between local stresses and strains may now be written

$$\begin{pmatrix} \sigma_{LL} \\ \sigma_{TT} \\ \sigma_{\hat{T}\hat{T}} \\ \sigma_{LT} \\ \sigma_{T\hat{T}} \\ \sigma_{\hat{T}L} \end{pmatrix} = \begin{pmatrix} D_{11} & D_{12} & D_{13} & 0 & 0 & 0 \\ D_{12} & D_{22} & D_{23} & 0 & 0 & 0 \\ D_{13} & D_{23} & D_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & 2D_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & 2D_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 2D_{66} \end{pmatrix} \begin{pmatrix} \varepsilon_{LL} \\ \varepsilon_{TT} \\ \varepsilon_{\hat{T}\hat{T}} \\ \frac{1}{2}\gamma_{LT} \\ \frac{1}{2}\gamma_{T\hat{T}} \\ \frac{1}{2}\gamma_{\hat{T}L} \end{pmatrix}. \tag{24}$$

Now, we introduce global measures for stresses and strains by substituting (22) and (23) into (24). This yields

$$\boldsymbol{T} \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix} = \begin{pmatrix} D_{11} & D_{12} & D_{13} & 0 & 0 & 0 \\ D_{12} & D_{22} & D_{23} & 0 & 0 & 0 \\ D_{13} & D_{23} & D_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & 2D_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & 2D_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 2D_{66} \end{pmatrix} \boldsymbol{T} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \frac{1}{2}\gamma_{xy} \\ \frac{1}{2}\gamma_{yz} \\ \frac{1}{2}\gamma_{zx} \end{pmatrix}. \tag{25}$$

By multiplying equation (25) by $\boldsymbol{T}^{-1}$ we obtain

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix} = \boldsymbol{T}^{-1} \begin{pmatrix} D_{11} & D_{12} & D_{13} & 0 & 0 & 0 \\ D_{12} & D_{22} & D_{23} & 0 & 0 & 0 \\ D_{13} & D_{23} & D_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & 2D_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & 2D_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 2D_{66} \end{pmatrix} \boldsymbol{T} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \frac{1}{2}\gamma_{xy} \\ \frac{1}{2}\gamma_{yz} \\ \frac{1}{2}\gamma_{zx} \end{pmatrix}. \tag{26}$$

This relation may alternatively be expressed as

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{pmatrix} = \begin{pmatrix} \bar{D}_{11} & \bar{D}_{12} & \bar{D}_{13} & \bar{D}_{14} & \bar{D}_{15} & \bar{D}_{16} \\ \bar{D}_{12} & \bar{D}_{22} & \bar{D}_{23} & \bar{D}_{24} & \bar{D}_{25} & \bar{D}_{26} \\ \bar{D}_{13} & \bar{D}_{23} & \bar{D}_{33} & \bar{D}_{34} & \bar{D}_{35} & \bar{D}_{36} \\ \bar{D}_{14} & \bar{D}_{24} & \bar{D}_{34} & \bar{D}_{44} & \bar{D}_{45} & \bar{D}_{46} \\ \bar{D}_{15} & \bar{D}_{25} & \bar{D}_{35} & \bar{D}_{45} & \bar{D}_{55} & \bar{D}_{56} \\ \bar{D}_{16} & \bar{D}_{26} & \bar{D}_{36} & \bar{D}_{46} & \bar{D}_{56} & \bar{D}_{66} \end{pmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{pmatrix}, \tag{27}$$

where the layer stiffness matrix (denoted $\bar{\boldsymbol{D}}$ in the following) is obtained by pre- and post-multiplication of the (slightly modified) $\boldsymbol{D}$ matrix in (26) by $\boldsymbol{T}^{-1}$ and $\boldsymbol{T}$, respectively, and finally dividing all entries in the last three columns by 2 to account for the introduction of engineering shear strains in the strain vector. While it is obvious that the $\boldsymbol{D}$ matrix in (21) contains nine independent material coefficients, it looks like being 21 independent parameters in the $\bar{\boldsymbol{D}}$ matrix above. However, it can be shown that there exist several relations between the coefficients, and that there are still only nine independent parameters, which is in agreement with the assumption that the layer is orthotropic.

Let us now be a bit more specific and develop the expressions that are actually implemented in the module to be described in the present report. We will first present the transformation matrix $\boldsymbol{T}$. The numerical module of consideration is mainly applied in the analysis of plane laminates, which might be connected to other plane laminates by, e.g., adhesive bonding. It is therefore assumed that all the layers are located in horizontal planes, and that the fibres of a layer might have been rotated an arbitrary angle θ with respect to the fixed, global x-axis. The horizontal plane is defined by the x and y axes, while the z axis points in the vertical direction. After introducing these assumptions, the transformation matrix can be given by

$$\boldsymbol{T} = \begin{pmatrix} c^2 & s^2 & 0 & 2sc & 0 & 0 \\ s^2 & c^2 & 0 & -2sc & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -sc & sc & 0 & c^2 - s^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & c & -s \\ 0 & 0 & 0 & 0 & s & c \end{pmatrix}, \tag{28}$$

where $c = \cos\theta$ and $s = \sin\theta$. As we have already stated, in order to describe an elastic, orthotropic layer nine independent material properties are required, for example E_L, E_T, $E_{\hat{T}}$, ν_{LT}, $\nu_{T\hat{T}}$, $\nu_{L\hat{T}}$, G_{LT}, $G_{T\hat{T}}$ and $G_{L\hat{T}}$. However, in the present applications it is assumed that all the layers possess transversely isotropic properties. This means that all cross sections (of the layers) perpendicular to the fibres possess isotropically. Thus, the following relations are fulfilled

$$\begin{aligned} E_T &= E_{\hat{T}} \\ G_{LT} &= G_{L\hat{T}} \\ \nu_{LT} &= \nu_{L\hat{T}} \\ G_{T\hat{T}} &= \frac{E_T}{2(1+\nu_{T\hat{T}})}. \end{aligned} \tag{29}$$

Due to these relations the number of independent material parameters required reduces from nine to five, e.g. E_L, E_T, ν_{LT}, $\nu_{T\hat{T}}$ and G_{LT}. This leads to considerable simplifications in the $\boldsymbol{D}$ and $\bar{\boldsymbol{D}}$ matrices. Nevertheless, the level of

complexity for the contributions in the stiffness matrices are still relatively high. Therefore, the definitions of the matrices are deferred to appendix A, which also includes the expressions for 2-D plane strain and plane stress approximations.

To conclude the present section, the main modification required to extend the elasticity module for isotropic materials described in the previous section, is to reimplement the procedure that calculates the material stiffness matrix at each integration (Gauss) point. For isotropic materials (in 3-D) the matrix is defined by the simple expressions in (10), while it is given by the rather involved definition of the $\bar{\boldsymbol{D}}$ matrix, see appendix A, in the context of orthotropic materials. Furthermore, some additional data, regarding the orientation of fibres and the extended number of material properties, as well as some minor modifications related to the handling of input and output data are required.

4 Geometrically Nonlinear Procedure

Until now we have concentrated on linear problems. The generalized Hooke's law defined in (3) describes the behaviour of isotropic, linear elastic media. Also the extensions introduced in section 3 and appendix A dealing with orthotropic materials are based on a linear elastic behaviour. Furthermore, the original elasticity module is restricted to geometric linear problems, assuming the deformations to be small.

Most laminated composites possess a linear elastic behaviour for small strains. However, although the analysis is limited to very small strains, geometrically nonlinear effects might be crucial [2]. For example, when analysing single lap joints such effects are significant for loads far below the ultimate strength of the joint. This is mainly due to the non-symmetric geometry of such joints, resulting in considerable deformations (deflections) even though the strains are small. Since the code described in this report is aimed at analysing a variety of composite joints it is extended to take geometrically nonlinear effects into account. Such effects may be included in numbers of ways, see e.g. [4, 11, 15]. In the present code the method of Zienkiewicz and Taylor [15], which will be described in the following, is implemented. After a general derivation of the solution procedure, we will specify expressions for the vectors and matrices included in the iterative process.

Using the principle of virtual work one may obtain the equilibrium equation

$$\boldsymbol{\Psi}(\boldsymbol{u}) = \int_V \bar{\boldsymbol{B}}^T \boldsymbol{\sigma} \mathrm{d}V - \boldsymbol{f} = 0, \tag{30}$$

where $\boldsymbol{\Psi}$ represents the sum of external and internal generalized forces, $\boldsymbol{u}$ is the nodal displacement vector from (19), $\boldsymbol{\sigma}$ is the vector of stress components given in (7) and $\boldsymbol{f}$ is the nodal load vector from (19). Finally, $\bar{\boldsymbol{B}}$ is the global matrix defining the nonlinear relation between strains and displacements;

$$\mathrm{d}\boldsymbol{\varepsilon} = \bar{\boldsymbol{B}} \mathrm{d}\boldsymbol{u}. \tag{31}$$

The $\bar{\boldsymbol{B}}$ matrix may be expressed by

$$\bar{\boldsymbol{B}} = \boldsymbol{B}_0 + \boldsymbol{B}_L(\boldsymbol{u}), \tag{32}$$

where $\boldsymbol{B}_0$ is the matrix that is applied in linear strain analysis, while $\boldsymbol{B}_L$, which depends on $\boldsymbol{u}$, is introduced to give an accurate representation of strains for problems with large deformations. Remark that the matrices in the relation (32) are global matrices. This means that, e.g., the $\boldsymbol{B}_0$ matrix is given by

$$\boldsymbol{B}_0 = \begin{pmatrix} \boldsymbol{B}_1 & \boldsymbol{B}_2 & \dots & \boldsymbol{B}_n \end{pmatrix}, \tag{33}$$

where $\boldsymbol{B}_j,\ j = 1, \dots, n$ is defined in (16). Hence, the dimension of the $\bar{\boldsymbol{B}}$ matrix in 3-D problems is $6 \times 3n$, where n is the total number of nodes in the grid. Although this representation is valid for large deformations we assume that the strains are limited to the linear elastic regime. Thus, the relation between (ply) stresses and (ply) strains is given by

$$\boldsymbol{\sigma} = \bar{\boldsymbol{D}}\boldsymbol{\varepsilon}, \tag{34}$$

where the material stiffness matrix $\bar{\boldsymbol{D}}$ is defined in appendix A.

Having (31), (32) and (34) in mind, it is obvious that the equation system (30) is nonlinear in $\boldsymbol{u}$. Therefore, the solution has to be approached iteratively. When using the Newton Raphson iteration method we need a relation between $\mathrm{d}\Psi$ and $\mathrm{d}\boldsymbol{u}$. From (30) we obtain

$$\mathrm{d}\Psi = \int_V \mathrm{d}\bar{\boldsymbol{B}}^T \boldsymbol{\sigma} \mathrm{d}V + \int_V \bar{\boldsymbol{B}}^T \mathrm{d}\boldsymbol{\sigma} \mathrm{d}V = \boldsymbol{K}_T \mathrm{d}\boldsymbol{u}, \tag{35}$$

where $\boldsymbol{K}_T$ is the total, tangential stiffness matrix to be defined later. From (31), (32) and (34) it is clear that the differentials in (35) may be expressed as

$$\mathrm{d}\boldsymbol{\sigma} = \bar{\boldsymbol{D}}\mathrm{d}\boldsymbol{\varepsilon} = \bar{\boldsymbol{D}}\bar{\boldsymbol{B}}\mathrm{d}\boldsymbol{u} \tag{36}$$

and

$$\mathrm{d}\bar{\boldsymbol{B}} = \mathrm{d}\boldsymbol{B}_L. \tag{37}$$

Inserting these expressions into (35) yields

$$\mathrm{d}\Psi = \int_V \mathrm{d}\boldsymbol{B}_L^T \boldsymbol{\sigma} \mathrm{d}V + \bar{\boldsymbol{K}} \mathrm{d}\boldsymbol{u}, \tag{38}$$

where $\bar{\boldsymbol{K}}$ is given by

$$\bar{\boldsymbol{K}} = \int_V \bar{\boldsymbol{B}}^T \bar{\boldsymbol{D}} \bar{\boldsymbol{B}} \mathrm{d}V = \boldsymbol{K} + \boldsymbol{K}_L. \tag{39}$$

Here, $\boldsymbol{K}$ is the usual, global stiffness matrix from (19), while $\boldsymbol{K}_L$, which is due to the large displacements, is given by

$$\boldsymbol{K}_L = \int_V \left(\boldsymbol{B}_0^T \bar{\boldsymbol{D}} \boldsymbol{B}_L + \boldsymbol{B}_L^T \bar{\boldsymbol{D}} \boldsymbol{B}_L + \boldsymbol{B}_L^T \bar{\boldsymbol{D}} \boldsymbol{B}_0 \right) \mathrm{d}V. \tag{40}$$

It is observed that $\boldsymbol{K}_L$ only contains terms that are linear and quadratic in $\boldsymbol{u}$. The first term on the right hand side of (38) can generally be written as

$$\int_V \mathrm{d}\boldsymbol{B}_L^T \boldsymbol{\sigma} \mathrm{d}V = \boldsymbol{K}_\sigma \mathrm{d}\boldsymbol{u}, \tag{41}$$

where $\boldsymbol{K}_\sigma$ is a symmetric matrix that depends on the stress level. From (38) and (41) it is now obvious that (35) may be expressed as

$$\mathrm{d}\boldsymbol{\Psi} = (\boldsymbol{K} + \boldsymbol{K}_\sigma + \boldsymbol{K}_L)\,\mathrm{d}\boldsymbol{u} = \boldsymbol{K}_T \mathrm{d}\boldsymbol{u}. \tag{42}$$

The Newton Raphson solution procedure for the nonlinear problem (30) contains the following steps:

1. The solution of the linear problem $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$ is obtained as a first approximation $\boldsymbol{u}^0$.
2. $\boldsymbol{\Psi}^0$ is calculated using (30) and the approximation $\boldsymbol{u}^0$.
3. The matrix $\boldsymbol{K}_T^0$ is established.
4. The correction to the solution vector is calculated as $\Delta\boldsymbol{u}^0 = -\left(\boldsymbol{K}_T^0\right)^{-1} \boldsymbol{\Psi}^0$.

The steps 2, 3 and 4 are repeated until $\boldsymbol{\Psi}^i$ becomes sufficiently small.

We will now give the expressions for the various quantities above (in particular $\boldsymbol{B}_L$ and $\boldsymbol{K}_T$) that are applicable for general 3-D problems. Let us start by the strains, which are based on the accurate Green-Lagrange strain tensor. The strain vector is separated into the usual infinitesimal displacement component and an additional part that is required in order to describe large displacements accurately; i.e.

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}_0 + \boldsymbol{\varepsilon}_L, \tag{43}$$

where $\boldsymbol{\varepsilon}_0$ is identical to the strain vector $\boldsymbol{\varepsilon}$ defined in (8), and $\boldsymbol{\varepsilon}_L$ is given by

$$\boldsymbol{\varepsilon}_L = \frac{1}{2} \begin{pmatrix} \boldsymbol{\Theta}_x^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Theta}_y^T & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\Theta}_z^T \\ \boldsymbol{\Theta}_y^T & \boldsymbol{\Theta}_x^T & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Theta}_z^T & \boldsymbol{\Theta}_y^T \\ \boldsymbol{\Theta}_z^T & \mathbf{0} & \boldsymbol{\Theta}_x^T \end{pmatrix} \begin{pmatrix} \boldsymbol{\Theta}_x \\ \boldsymbol{\Theta}_y \\ \boldsymbol{\Theta}_z \end{pmatrix} = \frac{1}{2}\boldsymbol{A}\boldsymbol{\Theta}. \tag{44}$$

Here

$$\boldsymbol{\Theta}_x^T = \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial v}{\partial x} & \frac{\partial w}{\partial x} \end{pmatrix}, \quad \boldsymbol{\Theta}_y^T = \begin{pmatrix} \frac{\partial u}{\partial y} & \frac{\partial v}{\partial y} & \frac{\partial w}{\partial y} \end{pmatrix}, \quad \boldsymbol{\Theta}_z^T = \begin{pmatrix} \frac{\partial u}{\partial z} & \frac{\partial v}{\partial z} & \frac{\partial w}{\partial z} \end{pmatrix}, \tag{45}$$

and $\mathbf{0}$ is a zero matrix with dimension 1×3. This means that $\boldsymbol{A}$ is a 6×9 matrix. From (44) it is seen that $\mathrm{d}\boldsymbol{\varepsilon}_L$ may be expressed as

$$\mathrm{d}\boldsymbol{\varepsilon}_L = \frac{1}{2}\mathrm{d}\boldsymbol{A}\boldsymbol{\Theta} + \frac{1}{2}\boldsymbol{A}\mathrm{d}\boldsymbol{\Theta} = \boldsymbol{A}\mathrm{d}\boldsymbol{\Theta}. \tag{46}$$

The $\boldsymbol{\Theta}$ matrix may be separated into a $\boldsymbol{G}$ matrix containing derivatives of the test functions N_i and the vector $\boldsymbol{u}$ consisting of nodal displacement components in the following way

$$\boldsymbol{\Theta} = \boldsymbol{G}\boldsymbol{u}. \tag{47}$$

For 3-D problems the $\boldsymbol{G}$ matrix then reads

$$\boldsymbol{G} = \begin{pmatrix} \frac{\partial N_1}{\partial x} & 0 & 0 & \dots & \frac{\partial N_n}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial x} & 0 & \dots & 0 & \frac{\partial N_n}{\partial x} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial x} & \dots & 0 & 0 & \frac{\partial N_n}{\partial x} \\ \frac{\partial N_1}{\partial y} & 0 & 0 & \dots & \frac{\partial N_n}{\partial y} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \dots & 0 & \frac{\partial N_n}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial y} & \dots & 0 & 0 & \frac{\partial N_n}{\partial y} \\ \frac{\partial N_1}{\partial z} & 0 & 0 & \dots & \frac{\partial N_n}{\partial z} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial z} & 0 & \dots & 0 & \frac{\partial N_n}{\partial z} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial z} & \dots & 0 & 0 & \frac{\partial N_n}{\partial z} \end{pmatrix}. \tag{48}$$

It is now obvious that $\mathrm{d}\boldsymbol{\varepsilon}_L$ and $\boldsymbol{B}_L$ are given by

$$\mathrm{d}\boldsymbol{\varepsilon}_L = \boldsymbol{A}\boldsymbol{G}\mathrm{d}\boldsymbol{u} \tag{49}$$

and

$$\boldsymbol{B}_L = \boldsymbol{A}\boldsymbol{G}, \tag{50}$$

respectively. Thus the $\bar{\boldsymbol{B}}$ matrix (32) is now completely specified.

With the definitions above most of the quantities required in the geometrically nonlinear procedure are established. This includes, e.g., the $\bar{\boldsymbol{K}}$ matrix introduced in (39) that depends on $\bar{\boldsymbol{B}}$. However, the contribution $\boldsymbol{K}_\sigma$ in the total, tangential stiffness matrix $\boldsymbol{K}_T$ is still not specified. Let us now take a careful look at $\boldsymbol{K}_\sigma$. From (41) and (50) we obtain

$$\boldsymbol{K}_\sigma \mathrm{d}\boldsymbol{u} = \int_V \mathrm{d}\boldsymbol{B}_L^T \boldsymbol{\sigma} \mathrm{d}V = \int_V \boldsymbol{G}^T \mathrm{d}\boldsymbol{A}^T \boldsymbol{\sigma} \mathrm{d}V. \tag{51}$$

Furthermore, it may be shown that we can write

$$\mathrm{d}\boldsymbol{A}^T\boldsymbol{\sigma} = \begin{pmatrix} \sigma_{xx}\boldsymbol{I}_3 & \sigma_{xy}\boldsymbol{I}_3 & \sigma_{xz}\boldsymbol{I}_3 \\ \sigma_{xy}\boldsymbol{I}_3 & \sigma_{yy}\boldsymbol{I}_3 & \sigma_{yz}\boldsymbol{I}_3 \\ \sigma_{xz}\boldsymbol{I}_3 & \sigma_{yz}\boldsymbol{I}_3 & \sigma_{zz}\boldsymbol{I}_3 \end{pmatrix} \mathrm{d}\boldsymbol{\Theta} = \boldsymbol{M}\boldsymbol{G}\mathrm{d}\boldsymbol{u}, \tag{52}$$

in which $\boldsymbol{I}_3$ is a 3×3 identity matrix. Substituting (52) into (51) yields

$$\boldsymbol{K}_\sigma = \int_V \boldsymbol{G}^T \boldsymbol{M} \boldsymbol{G} \mathrm{d}V, \tag{53}$$

where $\boldsymbol{M}$ is a 9×9 matrix of the six stress components arranged as in the parenthesis in (52). The symmetric form of $\boldsymbol{K}_\sigma$ is once again demonstrated.

All the expressions derived so forth in the present section are valid for 3-D problems. However, in order to solve plane problems (in the xz plane) all y components of the stresses, strains and displacements as well as all quantities differentiated with respect to y should be neglected, and the size of the system matrices and vectors should be reduced correspondingly.

5 Results

The geometrically nonlinear module for orthotropic materials that is presented in the present report has been thoroughly tested. For example, in Andersen and Osnes [2] results obtained by this module is compared with *ANSYS* results for a variety of test problems. In all cases, the *Diffpack* results are in close agreement with the solutions obtained by *ANSYS*. The purpose of the present report is, however, not to perform a comprehensive verification of the module. On the contrary, the major aim is to present the entire solution method implemented, including the expressions defining the behaviour of orthotropic materials. Nevertheless, in order to show that the module offers reasonable results we will present a couple of test examples.

In the first problem we consider uni-axial tension of a rectangular composite plate consisting of eight uni-directional plies with a thickness of 0.125 *mm* each. Thus, the total thickness of the laminate is 1.0 *mm*, while the length (in the loading direction) and the width of the plate are assumed to be 10 *mm* and 2 *mm*, respectively. The plate is modeled as a 2-D plane stress problem in a vertical coordinate system, in which the horizontal axis is parallel to the loading direction. The 2-D domain is discretized with 800 standard four noded bilinear elements. In order to obtain an accurate representation of the discontinuous stress fields several elements (five to be exact) are applied through the thickness of each layer. The boundary nodes on the left hand side are fixed (zero displacement), while a prescribed horizontal displacement of 0.15 *mm* is applied on the right hand side boundary. This is a more realistic way of simulating a test machine than applying nodal forces for each layer, because the stresses in each layer will depend on the stiffness of the layers. The same procedure is followed by Kairouz and Matthews in [9]. In the present example it is assumed that the laminate is made of $XAS/914C$ carbon fibre/epoxy resin. The material properties for the plies are listed in table 1. This simple test problem is run with two different choices of the lay-up; $[0/90]_{2s}$ and $[90/0]_{2s}$. The definitions of the 0-layer and 90-layer are based on the direction of the fibres in the layers. In the former case the fibres are parallel with a prescribed horizontal axis (or the loading direction in the present problem), while the fibres are orthogonal to this direction (but still oriented in the horizontal plane) in the latter case. The von Mises equivalent stress for these two choices of lay-up is shown in figures 1 and 2, respectively. A highly discontinuous stress distribution is apparent in both figures. A careful look at the results show that the von Mises equivalent stress in regions not to close to the left and right boundaries varies from approximately 0.141 *GPa* to 2.07 *GPa* in both cases. Thus the stress level differs by a factor near 14.7, which is due to (and comparable to) the difference in longitudinal and transverse stiffness properties for the plies. However, the location of the zones with high and low stress levels are completely opposite in the figures. This is simply due to the fact that the lay-up is opposite for the

two problems. In both cases, the high stress levels are observed in the (stiff) 0-layers, while the stresses in the (soft) 90-layers are low.

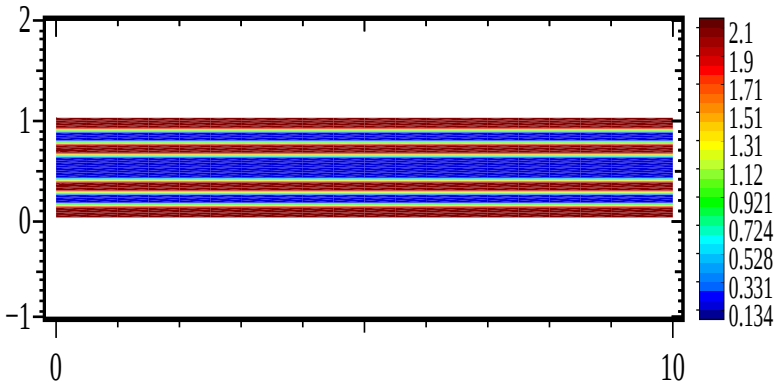

Figure 1: Distribution of von Mises equivalent stress (GPa) for the plate with lay-up $[0/90]_{2s}$. (Not to scale)

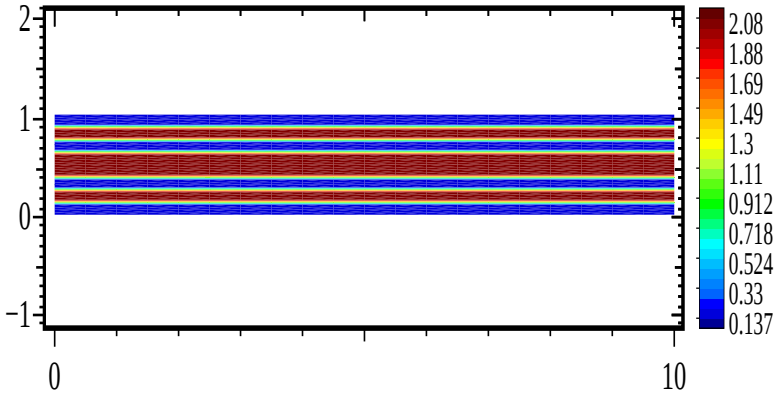

Figure 2: Distribution of von Mises equivalent stress (GPa) for the plate with lay-up $[90/0]_{2s}$. (Not to scale)

Now we will turn to a more complicated example, namely an adhesively bonded composite single lap joint. The joint is assumed to be wide compared to the thickness of the adherends and the length of the overlap region. Furthermore, each laminated plate consists of plies with fibre directions 0 and 90 degrees. The adhesive is uniformly distributed in the overlap and has a uniform thickness. Thus, out-of-plane bending is avoided, and the joint will be investigated as a 2-D plane strain problem in a vertical coordinate system.

The numerical, 2-D domain is shown in figure 3, which also defines the boundary conditions applied. As the figure shows, the $x = 0$ line is prevented from moving in the x-direction while the upper boundary from $x = 0$ to $x = 5$ and the lower boundary from $x = 45$ to $x = 50$ is prevented from moving in the z-direction. A prescribed displacement is then applied at the right hand side of the model.

Longitudinal elastic modulus, E_1 (GPa)	138
Transverse elastic modulus, E_2 (GPa)	9.4
In-plane shear modulus, G_{12} (GPa)	6.7
Major Poisson's ratio, ν_{12}	0.32

Table 1: Mechanical properties of $XAS/914C$ composite.

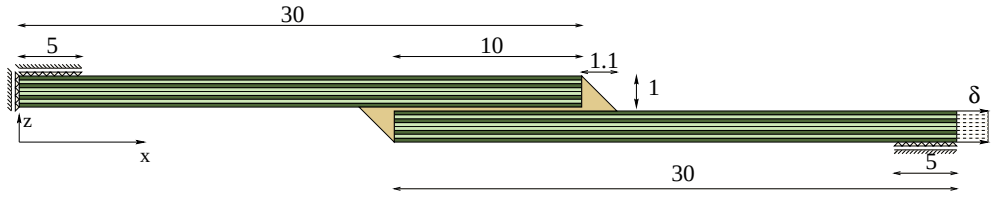

Figure 3: The 2-D domain for the numerical simulations. (Not to scale)

The laminated plates consist of eight plies with thickness 0.125 mm each. Furthermore, the thickness of the adhesive layer is 0.1 mm as shown in figure 3. Notice the different length scales along the x- and z-directions in the illustration. The stacking sequence of the adherends is $[0/90]_{2s}$. The mesh consists mainly of four-noded bilinear elements, but there are some three-noded triangular elements in the spew fillet. There are three elements through the thickness of each ply, and through the thickness of the adhesive layer there are eight elements. This is more than required to capture the discontinuities of the stress distribution in the joint. The total number of elements is 8778. Figure 4 shows the element mesh around the fillet tip. The adherends are made of $XAS/914C$ carbon fibre/epoxy resin and the adhesive used is Ciba-Geigy Redux $308A$. Mechanical properties for these materials are listed in tables 1 and 2, respectively.

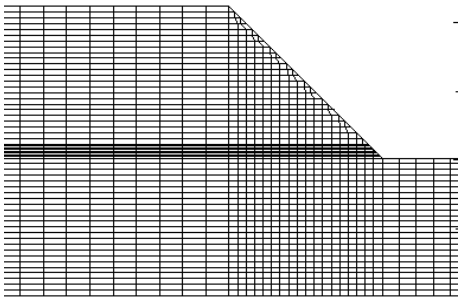

Figure 4: The *Diffpack* grid around the fillet tip.

Elastic modulus, E (MPa)	3000
Shear modulus, G (MPa)	1145
Poisson's ratio, ν	0.31

Table 2: Mechanical properties of Redux $308A$ adhesive.

A typical distribution of the von Mises equivalent stress resulting from the present geometrically nonlinear method is depicted in figure 5 for a crucial region of the (deformed) geometry. Here, the prescribed displacement, δ, of the right hand side boundary is 0.1 mm, which leads to stresses and strains far below critical ply crack limits (approximately one third). The maximum value of the equivalent stress (as well as σ_x and σ_z) is observed in the upper layer of the lower right laminate near the fillet tip. It is also seen that the equivalent stress in the 90-layer next to this 0-layer is much smaller. A careful inspection of the shear stresses in this region shows that the largest values occur in the adhesive layer near $x = 30$. The behaviour of all the stress components near the left spew fillet is similar. Thus, it may be concluded that the load path for the single lap joint of consideration is as follows: Near the left boundary most of the external load is carried by all the 0-layers. As we approach the fillet tip at $x = 20$ the curvature (due to the deformation of the non-symmetric joint) of the adherend ensures that major parts of the load is transmitted through the lower 0-layer. Then, the load is transferred from the upper left laminate to the lower right adherend by shear stresses in the adhesive layer. Finally, the distribution of stresses through the right laminate is similar to the observed behaviour of the left adherend.

Let us finally make some comments regarding the importance of the geometrically nonlinear module implemented. Conventional composite materials typically fail at axial strains in the region one to two percent. Thus, the necessity of using a geometrically nonlinear module may seem questionable. In the first example, uni-axial tension of a rectangular plate, the difference between geometrically linear and nonlinear results is negligible. However, for the single lap joint studied above Andersen and Osnes [2] have shown that this difference is considerable although the average longitudinal strain is as small as approximately 0.2%. This is mainly due to the non-symmetric geometry (and the corresponding eccentricity in the load path) of the joint, which leads to considerable bending of the adherends. This bending behaviour is much more accurately predicted by the nonlinear method than a linear model.

6 Conclusion

In the present report we have discussed the implementation of a geometrically nonlinear module dealing with orthotropic materials. The module, which is based on a code solving isotropic, linear elastic problems, is implemented in the

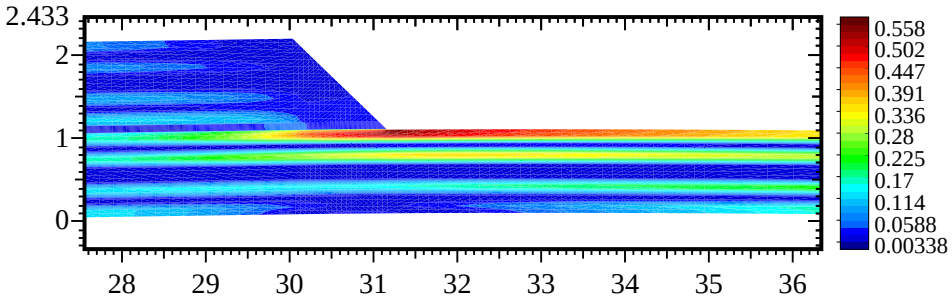

Figure 5: Distribution of the von Mises equivalent stress (GPa) near the fillet tip for $\delta = 0.1$ (mm).

commercial C++ library *Diffpack*. In the previous sections and the appendices a complete description, including the implemented code, of the new module is offered. Particular focus has been paid on the extensions related to orthotropic media (formulation of the material stiffness matrix $\bar{\boldsymbol{D}}$) and the geometrically nonlinear procedure. The code has been thoroughly verified. In section 5 the module proves to offer reasonable results for a couple of test cases. Additionally, in a former study [2], with emphasis on the significance of geometrically nonlinear effects in various problems, the code presented in this report has been extensively compared with the well-known *ANSYS* code, showing excellent agreement in most cases. Finally, it should be mentioned that in a future project we want to extend the module presented herein, with functionality for failure analysis purposes. The code is well suited for such extensions.

7 Acknowledgements

The project is supported by the University of Oslo, which is greatly acknowledged.

References

[1] B. D. Agarwal and L. J. Broutman. *Analysis and Performance of Fiber Composites*. Wiley, 2nd edition, 1990.

[2] A. Andersen and H. Osnes. Computational analysis of geometric nonlinear effects in adhesively bonded single lap composite joints. *Submitted to Composites B*, 2000.

[3] *ANSYS. World Wide Web home page*. http://www.ansys.com.

[4] K.-J. Bathe. *Finite Element Procedures*. Prentice-Hall, 1996.

[5] *Diffpack. World Wide Web home page.* http://www.nobjects.com/prodserv/diffpack.

[6] P. Goubalt and S. Mayes. Comparative analysis of metal and composite materials for the primary structures of a patrol craft. *Naval engineers journal*, pages 387–397, 1996.

[7] J. P. Den Hartog. *Advanced Strength of Materials.* Dover, 1987.

[8] S. C. Hunter. *Mechanics of Continuous Media.* Wiley, 2nd edition, 1983.

[9] K. C. Kairouz and F. L. Matthews. Strength and failure modes of bonded single lap joints between cross-ply adherends. *Composites*, 24(6):475–484, 1993.

[10] H. P. Langtangen. *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming.* Springer, 1999.

[11] L. E. Malvern. *Introduction to the Mechanics of a Continuous Medium.* Prentice-Hall, 1969.

[12] G. E. Mase. *Theory and Problems of Continuum Mechanics.* McGraw-Hill, 1970.

[13] M. Munro and S. Lee. Modeling in-plane shear modulus of composite materials for aerospace applications. *Journal of Reinforced plastics and Composites*, 14:471–495, 1995.

[14] L. A. Segel. *Mathematics Applied to Continuum Mechanics.* Dover, 1987.

[15] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method.* McGraw-Hill Book Company, 4th edition, 1989.

A Derivation of Stiffness Matrices for orthotropic materials

In the present appendix we will derive the expressions for the stiffness matrix $\bar{\boldsymbol{D}}$ introduced in section 3 for an orthotropic material which is assumed to be transversely isotropic. The derivation is most properly performed by starting with the compliance matrix S, which is, in fact, the inverse of the stiffness matrix ($\boldsymbol{\varepsilon} = \boldsymbol{S}\ \boldsymbol{\sigma}$). Under the present restrictions the compliance matrix is easily derived through simple analytical considerations of deformations due to a sequence of applied stresses, see e.g. [1]. When expressed in local (layer)

coordinates, the compliance matrix for a lamina reads

$$\boldsymbol{S} = \begin{pmatrix} \frac{1}{E_L} & -\frac{\nu_{LT}}{E_L} & -\frac{\nu_{LT}}{E_L} & 0 & 0 & 0 \\ -\frac{\nu_{LT}}{E_L} & \frac{1}{E_T} & -\frac{\nu_{T\hat{T}}}{E_T} & 0 & 0 & 0 \\ -\frac{\nu_{LT}}{E_L} & -\frac{\nu_{T\hat{T}}}{E_T} & \frac{1}{E_T} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{G_{LT}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{G_{T\hat{T}}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{G_{LT}} \end{pmatrix}, \tag{54}$$

where $G_{T\hat{T}}$ are related to E_T and $\nu_{T\hat{T}}$ through

$$G_{T\hat{T}} = \frac{E_T}{2(1+\nu_{T\hat{T}})}.$$

By inverting the compliance matrix we obtain the stiffness matrix for the layer

$$\boldsymbol{S}^{-1} = \boldsymbol{D} = \begin{pmatrix} \frac{E_L^2(\nu_{T\hat{T}}-1)}{m_1} & -\frac{E_L E_T \nu_{LT}}{m_1} & -\frac{E_L E_T \nu_{LT}}{m_1} & 0 & 0 & 0 \\ -\frac{E_L E_T \nu_{LT}}{m_1} & \frac{E_T(-E_L+E_T\nu_{LT}^2)}{m_2} & -\frac{E_T(E_L\nu_{T\hat{T}}+E_T\nu_{LT}^2)}{m_2} & 0 & 0 & 0 \\ -\frac{E_L E_T \nu_{LT}}{m_1} & -\frac{E_T(E_L\nu_{T\hat{T}}+E_T\nu_{LT}^2)}{m_2} & \frac{E_T(-E_L+E_T\nu_{LT}^2)}{m_2} & 0 & 0 & 0 \\ 0 & 0 & 0 & G_{LT} & 0 & 0 \\ 0 & 0 & 0 & 0 & G_{T\hat{T}} & 0 \\ 0 & 0 & 0 & 0 & 0 & G_{LT} \end{pmatrix}, \tag{55}$$

where the denominators are given by

$$m_1 = E_L\nu_{T\hat{T}} - E_L + 2E_T\nu_{LT}^2 \tag{56}$$

and

$$m_2 = E_L(-1+\nu_{T\hat{T}}^2) + 2E_T\nu_{LT}^2(1+\nu_{T\hat{T}}), \tag{57}$$

respectively. In order to simplify the notation for the $\boldsymbol{D}$ matrix let us introduce the following parameters:

$$\begin{aligned} D_1 &= \frac{E_L^2(\nu_{T\hat{T}}-1)}{m_1} \\ D_2 &= -\frac{E_L E_T \nu_{LT}}{m_1} \\ D_3 &= \frac{E_T(-E_L+E_T\nu_{LT}^2)}{m_2} \\ D_4 &= -\frac{E_T(E_L\nu_{T\hat{T}}+E_T\nu_{LT}^2)}{m_2}. \end{aligned} \tag{58}$$

Then, the stiffness matrix $\boldsymbol{D}$ may be written

$$\boldsymbol{D} = \begin{pmatrix} D_1 & D_2 & D_2 & 0 & 0 & 0 \\ D_2 & D_3 & D_4 & 0 & 0 & 0 \\ D_2 & D_4 & D_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & G_{LT} & 0 & 0 \\ 0 & 0 & 0 & 0 & G_{T\hat{T}} & 0 \\ 0 & 0 & 0 & 0 & 0 & G_{LT} \end{pmatrix}. \tag{59}$$

Remark that the matrix contains six different coefficients, of which only five are independent.

We will now transform the $\boldsymbol{D}$ matrix to represent the actual stiffness for a layer with respect to a fixed global coordinate system. Following the procedure introduced in the section 3 (which includes multiplication of the elements in the last three columns of $\boldsymbol{D}$ by 2, pre- and post-multiplication of $\boldsymbol{D}$ by $\boldsymbol{T}^{-1}$ and $\boldsymbol{T}$, respectively, and, finally, divide the elements in the last three columns by a factor of 2) we obtain

$$\bar{\boldsymbol{D}} = \begin{pmatrix} \bar{D}_{11} & \bar{D}_{12} & \bar{D}_{13} & \bar{D}_{14} & 0 & 0 \\ \bar{D}_{12} & \bar{D}_{22} & \bar{D}_{23} & \bar{D}_{24} & 0 & 0 \\ \bar{D}_{13} & \bar{D}_{23} & \bar{D}_{33} & \bar{D}_{34} & 0 & 0 \\ \bar{D}_{14} & \bar{D}_{24} & \bar{D}_{34} & \bar{D}_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & \bar{D}_{55} & \bar{D}_{56} \\ 0 & 0 & 0 & 0 & \bar{D}_{56} & \bar{D}_{66} \end{pmatrix}, \tag{60}$$

where

$$\begin{aligned}
\bar{D}_{11} &= c^4 D_1 + 2c^2 s^2 D_2 + s^4 D_3 + 4c^2 s^2 G_{LT} \\
\bar{D}_{12} &= c^2 s^2 D_1 + \left(c^4 + s^4\right) D_2 + c^2 s^2 D_3 - 4c^2 s^2 G_{LT} \\
\bar{D}_{13} &= c^2 D_2 + s^2 D_4 \\
\bar{D}_{14} &= c^3 s \left(D_1 - D_2 - 2G_{LT}\right) + cs^3 \left(D_2 - D_3 + 2G_{LT}\right) \\
\bar{D}_{22} &= s^4 D_1 + 2c^2 s^2 D_2 + c^4 D_3 + 4c^2 s^2 G_{LT} \\
\bar{D}_{23} &= s^2 D_2 + c^2 D_4 \\
\bar{D}_{24} &= cs^3 \left(D_1 - D_2 - 2G_{LT}\right) + c^3 s \left(D_2 - D_3 + 2G_{LT}\right) \\
\bar{D}_{33} &= D_3 \\
\bar{D}_{34} &= cs \left(D_2 - D_4\right) \\
\bar{D}_{44} &= c^2 s^2 \left(D_1 - 2D_2 + D_3\right) + \left(c^2 - s^2\right)^2 G_{LT} \\
\bar{D}_{55} &= s^2 G_{LT} + c^2 G_{T\hat{T}} \\
\bar{D}_{56} &= cs \left(G_{LT} - G_{T\hat{T}}\right) \\
\bar{D}_{66} &= c^2 G_{LT} + s^2 G_{T\hat{T}}.
\end{aligned} \tag{61}$$

To reduce the amount of CPU-time required when analysing isotropic elasticity problems one often try to introduce various simplifying assumptions. For example, in several applications the stress or strain distributions may be well approximated by assuming plane conditions to exist. This typically reduces the $\boldsymbol{D}$ matrix (applied in the constitutive relation) from a 6×6 to a 3×3 matrix, see section 2. The size of the global element stiffness matrix is decreased correspondingly, and the time spent on the FEM analyses may thus be reduced dramatically. Similar approximations may be introduced in the context of composite materials. When studying thin laminates the through thickness stresses and strains are often neglected, and the well-known theory of Kirchhoff valid for

thin plates may then be adopted. This leads to the classical laminate theory [1]. Several commercial software packages, e.g. *LAM 101*, *MIC-MAC* and *LAMINATE*, are based on that theory. However, in the present report we will choose a different approach. A main application area of the present code is to analyse adhesively bonded composite joints. Although the laminates of consideration are located in the horizontal (xy) plane, considerable through thickness stresses and strains will presented. On the other hand, the width of bonded joints is often large compared to the overlap length and laminate thickness. Furthermore, the material and geometrical properties are often constant in the width direction. Therefore, a 2-D plane strain model in the vertical (xz) plane may often be adopted. This approach is implemented in the *Diffpack* module that is documented in the present report. Furthermore, a similar 2-D plane stress model (in the xz plane) is developed for comparison. As long as the fibres are oriented in the 0 and/or 90 degrees directions, the plane strain model might be extremely accurate. However, for fibre directions between 0 and 90 degrees, significant y components of the stresses and strains may be locally presented. Then, complete 3-D analysis should be performed.

The rest of this appendix will be devoted to the development of the material stiffness matrices for the 2-D plane stress and plane strain approximations that are introduced above. Let us begin with the $\bar{\boldsymbol{D}}$ matrix valid for plane strain problems. In this case the matrix is derived from the corresponding 3-D matrix by simply neglecting all contributions related to the y components of the stresses and strains. From (27) and (60) it is easily seen that this matrix reads

$$\bar{\boldsymbol{D}} = \begin{pmatrix} \bar{D}_{11} & \bar{D}_{13} & 0 \\ \bar{D}_{13} & \bar{D}_{33} & 0 \\ 0 & 0 & \bar{D}_{66} \end{pmatrix}. \tag{62}$$

The derivation of the stiffness matrix for the 2-D plane stress situation is slightly more involved. In this case, certain stress components are assumed to be zero. Thus, from (27) it is seen that the $\bar{\boldsymbol{D}}$ matrix cannot be obtained simply by neglecting several elements from the 3-D stiffness matrix in (60). It is now advantageous to start with the 3-D compliance matrix

$$\boldsymbol{S} = \begin{pmatrix} S_1 & S_2 & S_2 & 0 & 0 & 0 \\ S_2 & S_3 & S_4 & 0 & 0 & 0 \\ S_2 & S_4 & S_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & S_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & S_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & S_5 \end{pmatrix}, \tag{63}$$

where

$$S_1 = \frac{1}{E_L}$$

$$
\begin{aligned}
S_2 &= -\frac{\nu_{LT}}{E_L} \\
S_3 &= \frac{1}{E_T} \\
S_4 &= -\frac{\nu_{T\hat{T}}}{E_T} \\
S_5 &= \frac{1}{G_{LT}} \\
S_6 &= \frac{1}{G_{T\hat{T}}}.
\end{aligned}
\tag{64}
$$

(65)

Remark, that although the above compliance matrix contains six different quantities, there are still only five independent coefficients. We will now transform the matrix above to represent the compliance matrix for a lamina related to a fixed, global (xyz) coordinate system. By dividing the last three rows of $\boldsymbol{S}$ by a factor of 2, performing pre- and post-multiplication by $\boldsymbol{T}^{-1}$ and $\boldsymbol{T}$, respectively, and, finally, multiplying the last three rows by 2, we obtain

$$
\bar{\boldsymbol{S}} = \begin{pmatrix}
\bar{S}_{11} & \bar{S}_{12} & \bar{S}_{13} & \bar{S}_{14} & 0 & 0 \\
\bar{S}_{12} & \bar{S}_{22} & \bar{S}_{23} & \bar{S}_{24} & 0 & 0 \\
\bar{S}_{13} & \bar{S}_{23} & \bar{S}_{33} & \bar{S}_{34} & 0 & 0 \\
\bar{S}_{14} & \bar{S}_{24} & \bar{S}_{34} & \bar{S}_{44} & 0 & 0 \\
0 & 0 & 0 & 0 & \bar{S}_{55} & \bar{S}_{56} \\
0 & 0 & 0 & 0 & \bar{S}_{56} & \bar{S}_{66}
\end{pmatrix},
\tag{66}
$$

where

$$
\begin{aligned}
\bar{S}_{11} &= c^4 S_1 + 2c^2 s^2 S_2 + s^4 S_3 + c^2 s^2 S_5 \\
\bar{S}_{12} &= c^2 s^2 S_1 + \left(c^4 + s^4\right) S_2 + c^2 s^2 S_3 - c^2 s^2 S_5 \\
\bar{S}_{13} &= c^2 S_2 + s^2 S_4 \\
\bar{S}_{14} &= c^3 s \left(2S_1 - 2S_2 - S_5\right) + c s^3 \left(2S_2 - 2S_3 + S_5\right) \\
\bar{S}_{22} &= s^4 S_1 + 2c^2 s^2 S_2 + c^4 S_3 + c^2 s^2 S_5 \\
\bar{S}_{23} &= s^2 S_2 + c^2 S_4 \\
\bar{S}_{24} &= c s^3 \left(2S_1 - 2S_2 - S_5\right) + c^3 s \left(2S_2 - 2S_3 + S_5\right) \\
\bar{S}_{33} &= S_3 \\
\bar{S}_{34} &= 2cs \left(S_2 - S_4\right) \\
\bar{S}_{44} &= 4c^2 s^2 \left(S_1 - 2S_2 + S_3\right) + \left(c^2 - s^2\right)^2 S_5 \\
\bar{S}_{55} &= s^2 S_5 + c^2 S_6 \\
\bar{S}_{56} &= cs \left(S_5 - S_6\right) \\
\bar{S}_{66} &= c^2 S_5 + s^2 S_6.
\end{aligned}
\tag{67}
$$

Now, we can implement the assumption of plane stress by neglecting all elements in the $\bar{\boldsymbol{S}}$ matrix that are connected to the stress components in the y-direction.

This reads

$$\bar{\boldsymbol{S}} = \begin{pmatrix} \bar{S}_{11} & \bar{S}_{13} & 0 \\ \bar{S}_{13} & \bar{S}_{33} & 0 \\ 0 & 0 & \bar{S}_{66} \end{pmatrix}. \tag{68}$$

Finally, in order to obtain the stiffness matrix for the plane stress problem the compliance matrix (68) has to be inverted;

$$\bar{\boldsymbol{D}} = \bar{\boldsymbol{S}}^{-1} = \begin{pmatrix} \frac{\bar{S}_{33}}{m_3} & -\frac{\bar{S}_{13}}{m_3} & 0 \\ -\frac{\bar{S}_{13}}{m_3} & \frac{\bar{S}_{11}}{m_3} & 0 \\ 0 & 0 & \frac{1}{\bar{S}_{66}} \end{pmatrix}, \tag{69}$$

where the denominator is given by

$$m_3 = \bar{S}_{11}\bar{S}_{33} - \bar{S}_{13}^2. \tag{70}$$

B Code

The implementation of the geometrically nonlinear module for orthotropic materials is organized as three C++ files. The `main.cpp` file contains a very simple main program, while the class structure and the functionality required in the module are introduced in the `Elasticity2.h` file. Finally, the procedures are completely defined in the `Elasticity2.cpp` file. All these files will be listed in the following.

The first file to be introduced is `main.cpp`, which reads:

```
                                                        // -*- C++ -*-
#include <Elasticity2.h>

int main (int nargs, const char** args)
{
  initDiffpack (nargs, args);
  global_menu.init ("Geometrically Nonlinear Simulator for Orthotropic "
                    "Materials","Elasticity2");
  Elasticity2 problem;
  global_menu.multipleLoop (problem);
  return 0;
}
```

The class structure and all the procedures are introduced in the `Elasticity2.h` file as follows:

```
#ifndef Elasticity2_h_IS_INCLUDED
#define Elasticity2_h_IS_INCLUDED

#include <FEM.h>
#include <DegFreeFE.h>
#include <NonLinEqSolver.h>
#include <NonLinEqSolver_prm.h>
#include <NonLinEqSolverUDC.h>
#include <LinEqAdmFE.h>
#include <SaveSimRes.h>
#include <FieldFormat.h>
#include <FieldsFEatItgPt.h>
```

```
#define Type Mat(real)
#include <VecSimplest_Type.h>
#undef Type

class Elasticity2 : public FEM, public NonLinEqSolverUDC
{
protected:
  Handle(GridFE)          grid;
  Handle(DegFreeFE)       dof;
  Handle(FieldsFE)        u;                // displacement field
  Handle(FieldsFEatItgPt) stress_measures;  // stress components including
                                            // von Mises equivalent stress
  Handle(FieldsFE)        smooth_stress_measures;
  Handle(SaveSimRes)      database;

  enum Elasticity_type
       { PLANE_STRESS, PLANE_STRAIN, THREE_DIM, AXISYMMETRY };
  Elasticity_type  elasticity_tp;

  int no_stress_strain_components;  // Typically 6 in 3-D and 3 in 2-D.

  // internal structures for avoiding time consuming reallocation:
  VecSimplest(Mat(real)) B0mats;    // containing derivatives of test functions

  Vec(real)                 linsol;
  Handle(LinEqAdmFE)        lineq;
  Vec(real)                 nonlinsol;
  Handle(NonLinEqSolver_prm) nlsolver_prm;
  Handle(NonLinEqSolver)    nlsolver;

  // Elasticity data for the composite layers. One VecSimple for each mat. type
  // Each VecSimple object contain: fiber angle, E_L, E_T, nu_LT, G_LT, nu_TT
  // It is presumed that the fibers are directed in the (global) xy-plane
  VecSimplest(VecSimple(real)) elasticity_data;

  // Strength data for the composite layers (not applied in the presently).
  // strength_L_ten, strength_L_compr, strength_T_ten, strength_T_compr,
  // strength_LT, strength_TT.
  VecSimplest(VecSimple(real)) strength_data;


  VecSimplest(Mat(real))       Dmats;   // sigma = D epsilon

  // In case of 2-D plane strain problems we need additional data in
  // the calculation of stresses
  VecSimplest(Mat(real))       Dmats_help;

  // Data introduced in connection with the geometric nonlinear functionality
  Mat(real)              Amat, Mmat;
  VecSimplest(Mat(real)) BLmats, Bbarmats, Gmats;
  Vec(real)              theta, sigma_pt;

                                    // referred to a general xyz-system
  Mat(real)                  matBtDB; // used in integrands
  Mat(real)                  matBtD;  // used in integrands
  Mat(real)                  matGtMG;     // used in integrands
  Mat(real)                  matGtM;      // used in integrands

  FieldFormat      rho_format;      // density: format
  Handle(Field)    rho;             // density
  Ptv(real)        g_dir;           // direction of gravity: x(nsd)-dir

  real             pressure1;       // for boundary indicator 1
  real             pressure2;       // for boundary indicator 2

  // Prescribed displacement
```

```
  Ptv(real)        u0;              // Prescribed constant displacement
                                    // for boundary indicators ranging from
                                    // 2+nsd+1 (u0(1)) to 2+2*nsd (u0(nsd)

  Ptv(real)        a,b;             // Prescribed linearly varying (ax+b)
                                    // displacement for boundary indicators
                                    // ranging from
                                    // 2+2*nsd+1 (u0(1)) to 2+3*nsd


  real             magnification;   // factor for exaggerated displacement
  Handle(GridFE)   deformed_grid;   // grid + magnification*u
  Handle(FieldFE)  equiv_stress1;   // equiv. stress over undeformed grid
                                    // introduced to obtain suitable
                                    // fieldname when saved to file
  Handle(FieldFE)  equiv_stress2;   // equiv. stress over deformed grid
  Handle(FieldFE)  u_magnitude;     // magnitude of displacement vector

  // internal structures for avoiding time consuming reallocation:
  Mat(real)            matdxd;      // used in integrands
  Ptv(real)            normal_vec;  // used in integrands4side
  VecSimple(Ptv(real)) Du_pt;       // used in derivedQuantitiesAtItgPt

  Vec(real)            eps_pt;      // used in derivedQuantitiesAtItgPt
  Vec(real)            help_pt;     // used in derivedQuantitiesAtItgPt
  Vec(real)            help2_pt;    // used in integrands

  Ptv(real) grid_centroid;
public:
  Elasticity2 ();
 ~Elasticity2 () {}
  virtual void adm    (MenuSystem& menu);
  virtual void define (MenuSystem& menu, int level = MAIN);
  virtual void scan   ();
  virtual void solveProblem ();

  // calculate the D matrices (sigma = D epsilon) for the different
  // composite layers (referred to a global xyz-system)
  virtual void calcDmats  ();

  // calculate B matrices (containing various dNi/dxj) at an integration point
  virtual void calcB0mats (const FiniteElement& fe);
  virtual void calcAmatThetaEps (const FiniteElement& fe);
  // calculate system matrices at an integration point
  virtual void calcSystemMats (const FiniteElement& fe);

protected:
  void saveResults ();
  void calcDerivedQuantities ();

  virtual void fillEssBC ();
  virtual void integrands (ElmMatVec& elmat, const FiniteElement& fe);

  // must be redefined: requires side integration
  virtual void calcElmMatVec (int e, ElmMatVec& elmat, FiniteElement& fe);
  virtual void integrands4side
    (int side, int boind, ElmMatVec& elmat, const FiniteElement& fe);

  virtual void makeAndSolveLinearSystem ();

  virtual void derivedQuantitiesAtItgPt
    (VecSimple(NUMT)& quantities, const FiniteElement& fe);

  // report facilities:
  MultipleReporter  report;       // for ascii, latex and html reports
  String            purpose;      // one-line purpose in report header
```

```
  virtual void resultReport ();   // for multiple runs, single compact output
  virtual void openReport ();     // for multiple runs, open output file
  virtual void closeReport ();

  virtual void writeHeadings (StringList& headings);
  virtual void writeResults  (StringList& results);
  virtual void writeExtendedResults (MultipleReporter& rep);
};

// calculation and smoothing of von Mises stress:
class SmoothEquivStress : public IntegrandCalc
{
  Elasticity2& data;
public:
  SmoothEquivStress (Elasticity2& data_) : data(data_) {}
 ~SmoothEquivStress () {}
  virtual void integrands (ElmMatVec& elmat, const FiniteElement& fe);
};
#endif
```

Finally, the `Elasticity2.cpp` file is included for completeness;

```
#include <Elasticity2.h>
#include <ElmMatVec.h>
#include <FiniteElement.h>
#include <readOrMakeGrid.h>
#include <DegFreeFE.h>
#include <VecSimple_Ptv_real.h>
#include <MatDiag_real.h>
#include <SimReport.h>

#define Type Mat(real)
#include <VecSimplest_Type.cpp>
#undef Type

Elasticity2:: Elasticity2 () : FEM () {}

void Elasticity2:: adm (MenuSystem& menu)
{
  SimCase::attach (menu);
  define (menu);
  menu.prompt();
  scan ();
}

void Elasticity2:: define (MenuSystem& menu, int level)
{
  menu.addItem (level,          // menu level (1 is main, 2 is first submenu)
                "gridfile",     // menu command/name
                "file or preprocessor command",
                "P=PreproBox | d=2 [0,1]x[0,1] | d=2 e=ElmB4n2D [4,4] [1,1]");
  menu.addItem (level, "redefine boundary indicators",
                "GridFE::redefineBoinds(Is) syntax (\"NONE\"=no change)",
                "NONE");
  menu.addItem (level, "add boundary nodes",
                "GridFE::addBoIndNodes(Is) syntax (\"NONE\"=no change)",
                "NONE");
  menu.addItem (level, "add material",
                "GridFE::addMatrial syntax", "NONE");

  menu.addItem (level, "data for the composite layers",
                "For each layer give the following:"
                "angle (in deg.) E_L E_T nu_LT G_LT nu_TT (no ',' in between)",
                "");
  menu.addItem (level, "strength values for the composite layers",
                "For each layer give the following:"
                "strength_L_ten strength_L_compr strength_T_ten strength_T_compr strength_LT strength_TT",
```

```
                "");

  rho_format.  setFieldname("rho").  define (menu, "CONSTANT=0.0",  level);

  menu.addItem (level, "pressure 1",
                "pressure boundary condition (bo-ind 1)", "1.0");
  menu.addItem (level, "pressure 2",
                "pressure boundary condition (bo-ind 2)", "0.0");
  menu.addItem (level, "prescribed displacement",
                "prescribed displacement, e.g. 1.0 1.5 2.0 in 3D "
                "(bo-ind ranging from 2+nsd+1 to 2+2*nsd)","1.0 1.0 1.0");
  menu.addItem (level, "linear displacement coeff a",
                "linear displacement coeff a, e.g. 1.0 1.5 2.0 in 3D "
                "(bo-ind ranging from 2+2*nsd+1 to 2+3*nsd)","1.0 1.0 1.0");
  menu.addItem (level, "linear displacement coeff b",
                "linear displacement coeff b, e.g. 1.0 1.5 2.0 in 3D "
                "(bo-ind ranging from 2+2*nsd+1 to 2+3*nsd)","1.0 1.0 1.0");
  menu.addItem (level, "gravity direction",
                "e.g. 0 0 -1 if gravity acts in -x3 direction", "0 0 0");
  menu.addItem (level, "deformed grid magnification",
                "for deformed grid visualization of displacements: <0 "
                "implies 10% deformation relative to domain size",
                "-1.0");

  menu.addItem (level, "purpose",
                "one-line description of the purpose of this simulation",
                "No purpose stated");

  menu.addItem (level, "elasticity type", "elasticity_tp",
                "plane strain, plane stress, or axisymmetry in 2D (3D is "
                "automatically detected)", "PLANE_STRAIN",
                "S/PLANE_STRAIN/PLANE_STRESS/AXISYMMETRY/THREE_DIM/");

  LinEqAdmFE    ::defineStatic (menu, level+1);
  FEM           ::defineStatic (menu, level+1);
  SaveSimRes    ::defineStatic (menu, level+1);

  NonLinEqSolver_prm:: defineStatic(menu,level+1);
}


void Elasticity2:: scan ()
{
  MenuSystem& menu = SimCase::getMenuSystem();
  String gridfile = menu.get ("gridfile");
  grid.rebind (new GridFE());
  readOrMakeGrid (*grid, gridfile);  // fill grid

  String redef   = menu.get ("redefine boundary indicators");
  if (!redef.contains("NONE"))  grid->redefineBoInds (redef);

  String addbn   = menu.get ("add boundary nodes");
  if (!addbn.contains("NONE"))  grid->addBoIndNodes (addbn);

  String addmat  = menu.get ("add material");
  if (!addmat.contains("NONE")) grid->addMaterial (addmat);

  String elast_answer = menu.get ("data for the composite layers") + ";";
  SetOfNo(real) mat_data;
  mat_data.scan (elast_answer);
  // The number of input data is checked in the end of the present procedure

  String strength_answer =
    menu.get ("strength values for the composite layers") + ";";
  SetOfNo(real) str_data;
  str_data.scan (strength_answer);
  // The number of input data is checked in the end of the present procedure
```

```
rho_format.  scan (menu).allocateAndInit (rho, grid.getPtr());

const int nsd = grid->getNoSpaceDim();
g_dir.redim (nsd); g_dir = 0.0; g_dir.scan (menu.get ("gravity direction"));
magnification = menu.get( "deformed grid magnification").getReal();

pressure1 = menu.get ("pressure 1").getReal();
pressure2 = menu.get ("pressure 2").getReal();

u0.redim (nsd); u0.scan (menu.get ("prescribed displacement"));
a.redim (nsd); a.scan (menu.get ("linear displacement coeff a"));
b.redim (nsd); b.scan (menu.get ("linear displacement coeff b"));

FEM:: scan (menu);
lineq.rebind (new LinEqAdmFE());
lineq->scan (menu);
database.rebind (new SaveSimRes());
database->scan (menu, grid->getNoSpaceDim());

u.rebind (new FieldsFE (*grid, "u"));
dof.rebind (new DegFreeFE (*grid, nsd));

linsol.redim (u->getNoValues());
lineq->attach (linsol);
linsol.fill (0.0);   // init for iterative solvers

nonlinsol.redim (u->getNoValues ());  // use in iterations
nlsolver_prm.rebind (NonLinEqSolver_prm:: construct ());
nlsolver_prm->scan (menu);
nlsolver.rebind (nlsolver_prm->create ());
if (nlsolver->getCurrentState ().method != NEWTON_RAPHSON)
  errorFP ("Elasticity2:: scan",
           "The nonlinear solution method has to be Newton Raphson");

nlsolver->attachUserCode (*this);
nlsolver->attachNonLinSol (nonlinsol);
nlsolver->attachLinSol (linsol);

String el_tp = menu.get ("elasticity type");
if (grid->getNoSpaceDim() == 3) {
  elasticity_tp = THREE_DIM;
  no_stress_strain_components = 6;
}
else if (grid->getNoSpaceDim() == 2) {
  if (el_tp == "PLANE_STRAIN") {
    elasticity_tp = PLANE_STRAIN;
    no_stress_strain_components = 3;
  }
  else if (el_tp == "PLANE_STRESS") {
    elasticity_tp = PLANE_STRESS;
    no_stress_strain_components = 3;
  }
  else if (el_tp == "AXISYMMETRY") {
    elasticity_tp = AXISYMMETRY;
    no_stress_strain_components = 4;
    errorFP("Elasticity2::scan", "elasticity type AXISYMMETRY not impl.");
  }
  else
    errorFP("Elasticity2::scan","wrong elasticity type %s",el_tp.c_str());
}

stress_measures.rebind (new FieldsFEatItgPt ("stress_measures"));
smooth_stress_measures.rebind (new FieldsFE (*grid,
                                             no_stress_strain_components+1,
                                             "stress_component"));
u_magnitude.rebind (new FieldFE (*grid, "u_magnitude"));
```

```
equiv_stress1.rebind (new FieldFE (*grid, smooth_stress_measures()
                                   (no_stress_strain_components+1).values(),
                                   "equiv_stress"));

// extra data structures for visualization on the deformed grid:
deformed_grid.rebind (new GridFE());
*deformed_grid = *grid;  // will be deformed when the displ. is calculated
// equiv_stress2 is just defined over the deformed grid for plotting
// purposes, but can borrow the nodal values vector from equiv_stress:
equiv_stress2.rebind
  (new FieldFE (*deformed_grid, smooth_stress_measures()(no_stress_strain_components+1).values(),
                "equiv_stress_over_deformed_grid"));

purpose = menu.get ("purpose");      // for report
if (purpose.contains("No purpose"))  // default answer
  purpose = "";

B0mats.redim (grid->getMaxNoNodesInElm ());
BLmats.redim (grid->getMaxNoNodesInElm ());
Bbarmats.redim (grid->getMaxNoNodesInElm ());
Gmats.redim (grid->getMaxNoNodesInElm ());
int i;
for (i=1; i<=grid->getMaxNoNodesInElm (); i++) {
  B0mats(i).redim(no_stress_strain_components,nsd);
  B0mats(i).fill (0.0);
  BLmats(i).redim(no_stress_strain_components,nsd);
  BLmats(i).fill (0.0);
  Bbarmats(i).redim(no_stress_strain_components,nsd);
  Bbarmats(i).fill (0.0);
  Gmats(i).redim(nsd*nsd,nsd);
  Gmats(i).fill (0.0);
}


Amat.redim (no_stress_strain_components,nsd*nsd);
Amat.fill (0.0);
Mmat.redim (nsd*nsd,nsd*nsd);
Mmat.fill (0.0);

theta.redim (nsd*nsd);
theta.fill (0.0);
sigma_pt.redim (no_stress_strain_components);
sigma_pt.fill (0.0);


matdxd.redim (nsd, nsd);  // used just for avoiding frequent reallocation
normal_vec.redim (nsd);
Du_pt.redim (nsd);
for (i=1; i<=nsd; i++)
  Du_pt(i).redim(nsd);

eps_pt.redim (no_stress_strain_components);
help_pt.redim (no_stress_strain_components);
help2_pt.redim (nsd);                        help2_pt.fill (0.0);

grid_centroid.redim(nsd);
Ptv(real) grid_min(nsd), grid_max(nsd);
grid->getMinMaxCoord (grid_min, grid_max);
grid_centroid = 0.5*(grid_min + grid_max);
grid_centroid.print ("FILE=grid_centroid");

// Here we check the number of input material data.
const int no_mat = grid->getNoMaterials ();
const int no_data = mat_data.getNoMembers ();
const int no_str_data = str_data.getNoMembers ();
```

```
  if ( no_data != (6*no_mat) )
    errorFP ("Elasticity2:: scan",
             oform("The number of input material data (%d) is not consistent "
                   "with the number of material types (%d), six properties "
                   "are required for each material type",no_data,no_mat));

  elasticity_data.redim (no_mat);
  //  strength_data.redim (no_mat);
  Dmats.redim (no_mat);
  if ( elasticity_tp == PLANE_STRAIN )
    Dmats_help.redim (no_mat);

  matBtDB.redim (nsd, nsd); // used just for avoiding frequent reallocation
  matBtD.redim (nsd, no_stress_strain_components);
  matGtMG.redim (nsd, nsd); // used just for avoiding frequent reallocation
  matGtM.redim (nsd, nsd*nsd);

  int j;
  int k = 0;
  for ( i=1; i<=no_mat; i++) {
    elasticity_data(i).redim (6);
    //    strength_data(i).redim (6);
    for ( j=1; j<=6; j++) {
      k++;
      elasticity_data(i)(j) = mat_data.getMember (k);
      //      strength_data(i)(j) = str_data.getMember (k);
    }
    Dmats(i).redim (no_stress_strain_components);
    if ( elasticity_tp == PLANE_STRAIN )
      Dmats_help(i).redim (no_stress_strain_components);
  }
}


void Elasticity2:: solveProblem ()
{

  fillEssBC();
  calcDmats ();
  dof->field2vec (*u,nonlinsol);
  // Due to the particular solution
  // method of the present application. 1. iteration linear,
  // the rest of the iterations nonlinear.
  nlsolver->solve ();
  dof->vec2field(nonlinsol,*u);
  calcDerivedQuantities ();       // calculate stresses
  saveResults();
}


void Elasticity2:: fillEssBC ()
{
  // convention:
  // bo-ind 1 and 2: two constant normal stress conditions
  // bo-ind 2+k,..,2+nsd: zero displacement component no. k
  // bo-ind 2+nsd+k,..,2+2*nsd: prescribed displacement component no. k
  // bo-ind 2+2*nsd+k,..,2+3*nsd: prescribed linearly varying displacement
                               // component no. k

  dof->initEssBC ();
  int nno = grid->getNoNodes();
  int d   = grid->getNoSpaceDim();
  Ptv(real) node_coor(d);
  int i,k;
  for (i = 1; i <= nno; i++)
    for (k = 1; k <= d; k++) {
```

```
      if (grid->boNode (i, 2+k))
        dof->fillEssBC (i,k, 0.0);
      if (grid->boNode (i, 2+d+k))
        dof->fillEssBC (i,k, u0(k));
      if (grid->boNode (i, 2+(2*d)+k)) {
        node_coor = grid->getCoor (i);
        dof->fillEssBC (i,k, a(k)*node_coor(k) + b(k));
      }
    }
}


void Elasticity2:: calcElmMatVec
  (int elm_no, ElmMatVec& elmat, FiniteElement& fe)
{
  // volume integral:
  fe.refill (elm_no, itg_rules);
  numItgOverElm (elmat, fe);

  // element surface integral (prescribed stress vector):
  int s, nsides = fe.getNoSides();
  for (s = 1; s <= nsides; s++) {
    if (fe.boSide (s, 1)) // pressure condition 1
      numItgOverSide (s, 1, elmat, fe);
    if (fe.boSide (s, 2)) // pressure condition 2
      numItgOverSide (s, 2, elmat, fe);
  }
}


void Elasticity2:: integrands(ElmMatVec& elmat,const FiniteElement& fe)
{
  const int  d      = fe.getNoSpaceDim();
  const int  nbf    = fe.getNoBasisFunc();
  const real detJxW = fe.detJxW();

  const int material_number = fe.grid().getMaterialType(fe.getElmNo());

  // Handle(Field) rho must be interpolated at current point:
  const real rho_pt   = rho->valueFEM (fe);   // density

  int i,j;    // basis function counters
  int r,s;  // 1,..,nsd (space dimension) counters
  int ig,jg;  // element dof, based on i,j,r,s
  real body_force_term;

  // matBtDB and matBtD are class members to avoid repeated local allocation

  if (nlsolver->getCurrentState ().iteration_no == 1) {
    calcB0mats (fe);
    for (i = 1; i <= nbf; i++) {
      matBtD.prod (B0mats(i),Dmats(material_number),TRANSPOSED);
      for (j = 1; j <= nbf; j++) {
        matBtDB.prod (matBtD,B0mats(j));

        // add block matrix (i,j) to elmat.A:
        for (r = 1; r <= d; r++)
          for (s = 1; s <= d; s++) {
            ig = d*(i-1)+r;
            jg = d*(j-1)+s;
            elmat.A(ig,jg) += matBtDB(r,s)*detJxW;
          }
      }

      // add block matrix i to elmat.b:
      for (r = 1; r <= d; r++) {
        body_force_term = rho_pt*9.81*g_dir(r)*fe.N(i);
```

```
        ig = d*(i-1)+r;
        elmat.b(ig) += body_force_term*detJxW;
      }
    }
  }
  else {
    calcSystemMats (fe);

    for (i = 1; i <= nbf; i++) {
      matBtD.prod (Bbarmats(i),Dmats(material_number),TRANSPOSED);
      matGtM.prod (Gmats(i),Mmat,TRANSPOSED);
      for (j = 1; j <= nbf; j++) {
        matBtDB.prod (matBtD,Bbarmats(j));
        matGtMG.prod (matGtM,Gmats(j));

        // add block matrix (i,j) to elmat.A:
        for (r = 1; r <= d; r++)
          for (s = 1; s <= d; s++) {
            ig = d*(i-1)+r;
            jg = d*(j-1)+s;
            elmat.A(ig,jg) += (matBtDB(r,s)+matGtMG(r,s))*detJxW;
          }
      }

      // add block matrix i to elmat.b:
      Bbarmats(i).prod (sigma_pt, help2_pt, TRANSPOSED);
      for (r = 1; r <= d; r++) {
        body_force_term = rho_pt*9.81*g_dir(r)*fe.N(i);

        ig = d*(i-1)+r;
        elmat.b(ig) += (body_force_term-help2_pt(r))*detJxW;
      }
    }
  }
}


void Elasticity2:: integrands4side
  (int /*side*/, int boind, ElmMatVec& elmat, const FiniteElement& fe)
{
  const int d = fe.getNoSpaceDim();
  const int nbf = fe.getNoBasisFunc();
  const real JxW = fe.detSideJxW();
  fe.getNormalVectorOnSide (normal_vec);

  real pressure;
  if (boind == 1)       pressure = pressure1;
  else if (boind == 2)  pressure = pressure2;
  else
    fatalerrorFP("Elasticity2::integrands4side","wrong boind=%d",boind);

  int i,r,ig; real h;
  for (i = 1; i <= nbf; i++)
    for (r = 1; r <= d; r++)
      {
        h = fe.N(i)*pressure*normal_vec(r);
        ig = d*(i-1)+r;
        elmat.b(ig) += h*JxW;
      }
}


void Elasticity2:: makeAndSolveLinearSystem ()
{
  dof->vec2field (nonlinsol,*u);
  if ( (nlsolver->getCurrentState().method == NEWTON_RAPHSON) &&
```

```
      (nlsolver->getCurrentState ().iteration_no > 1) )
    dof->fillEssBC2zero();    // ensure no correction of known values!
  else
    dof->unfillEssBC2zero();  // normal treatment of ess. b.c. (default)

  makeSystem (*dof, *lineq);

  // init start vector (linsol) for iterative solver:
  if ( (nlsolver->getCurrentState().method == NEWTON_RAPHSON) &&
       (nlsolver->getCurrentState ().iteration_no > 1) )
    linsol.fill (0.0);  // start for a correction vector (should -> 0)
  else {
    linsol = nonlinsol;   //  Use the most recent nonlinear solution
                          //  However, in this particular application
                          //  nonlinsol equals 0 in initially
    dof->insertEssBC (linsol);  // linsol is filled with essential B.C.
  }

  lineq->solve();
}


void Elasticity2:: calcDerivedQuantities ()
{
  stress_measures->derivedQuantitiesAtItgPt
    (*this, *grid, no_stress_strain_components + 1   /* no strain measures */,
     GAUSS_POINTS, -1  /* reduced Gauss-Legendre sampling points */);

  int e;
  const int nel = grid->getNoElms ();
  bool elmt6n2d = false;
  for (e=1; e<=nel; e++)
    if ( grid->getElmType (e) == "ElmT6n2D" ) {
      elmt6n2d = true;
      break;
    }

  if ( elmt6n2d )
    FEM::smoothFields (*smooth_stress_measures, *stress_measures, MOVING_LS);
  else
    FEM::smoothFields (*smooth_stress_measures, *stress_measures);

  u->magnitude (*u_magnitude);
}


void Elasticity2:: derivedQuantitiesAtItgPt
(VecSimple(NUMT)& quantities, const FiniteElement& fe)
{
  int i;
  const int nsd = fe.getNoSpaceDim();

  // Calculations of the stresses.
  // The Dmats matrices are already calculated
  // The following matrix will be applied this time
  const int material_number = grid->getMaterialType (fe.getElmNo ());

  for (i = 1; i <= nsd; i++) {
    u()(i).derivativeFEM (Du_pt(i), fe);
    eps_pt(i) = Du_pt(i)(i);
  }

  if ( elasticity_tp == THREE_DIM ) {
    eps_pt(4) = Du_pt(1)(2) + Du_pt(2)(1);
    eps_pt(5) = Du_pt(2)(3) + Du_pt(3)(2);
    eps_pt(6) = Du_pt(3)(1) + Du_pt(1)(3);
  }
```

```
else if ( (elasticity_tp == PLANE_STRESS) ||
          (elasticity_tp == PLANE_STRAIN) )
  eps_pt(3) = Du_pt(1)(2) + Du_pt(2)(1);
else
  errorFP ("Elasticity2:: derivedQuantitiesAtItgPt",
           "Only implementet for 3-D and plane stress/strain problems");


Dmats(material_number).prod (eps_pt, help_pt);

for (i=1; i<=no_stress_strain_components; i++)
  quantities(i) = help_pt(i);


// Calculations of equivalent stress.
// Here, temperature effects are neglected

// use the scratch matrix matdxd (class member) as stress tensor:
Mat(real)& s = matdxd;
s.redim (3,3);   // always 3x3, even in 2D
s.fill (0.0);

if ( elasticity_tp == THREE_DIM ) {
  for (i=1; i<=nsd; i++)
    s(i,i) = quantities(i);

  s(1,2) = quantities(4);
  s(2,3) = quantities(5);
  s(1,3) = quantities(6);
  s(2,1) = s(1,2);
  s(3,2) = s(2,3);
  s(3,1) = s(1,3);
}
else if ( elasticity_tp == PLANE_STRESS  ) {
  // The 2-D plane stress/strain problems are in fact solved in the xz-plane
  s(1,1) = quantities(1);
  s(3,3) = quantities(2);

  s(1,3) = quantities(3);
  s(3,1) = s(1,3);
}
else if ( elasticity_tp == PLANE_STRAIN ) {
  // The 2-D plane stress/strain problems are in fact solved in the xz-plane
  s(1,1) = quantities(1);
  s(3,3) = quantities(2);

  s(1,3)  = quantities(3);
  s(3,1) = s(1,3);

  // Additional stress components due to the assumption of plane strain
  Dmats_help(material_number).prod (eps_pt, help_pt);

  s(2,2) = help_pt(1);
  s(1,2) = help_pt(2);
  s(2,3) = help_pt(3);
  s(3,2) = s(2,3);
  s(2,1) = s(1,2);
}
else if ( elasticity_tp == AXISYMMETRY )
  errorFP ("Elasticity2::derivedQuantAtItgPt",
           "Not implemented for Axi-symmetric problems");


// equivalent stress (see Mase p. 182):
real e2=0.5*(6.0*(sqr(s(1,2)) + sqr(s(2,3)) + sqr(s(1,3))) +
             sqr(s(1,1)-s(2,2)) + sqr(s(2,2)-s(3,3)) + sqr(s(3,3)-s(1,1)) );
```

```
  quantities(no_stress_strain_components+1) = sqrt(e2);
}


void Elasticity2:: saveResults ()
{
  database->dump (*u);
  database->dump (smooth_stress_measures()(1));
  database->dump (smooth_stress_measures()(2));
  database->dump (smooth_stress_measures()(3));
  database->dump (*equiv_stress1);
  database->dump (*u_magnitude);
  database->lineCurves (smooth_stress_measures()(1));
  database->lineCurves (smooth_stress_measures()(2));
  database->lineCurves (smooth_stress_measures()(3));
  database->lineCurves (smooth_stress_measures()(no_stress_strain_components+1));

  // calculate deformed_grid, according to u and the magnification factor
  if (magnification < 0.0) {
    // automatic calculation of a suitable magnification value:
    // let magnification be such that the largest displacement is about
    // 10 percent of the size of the domain

    // calculate characteristic size L of the domain:
    Ptv(real) mincoord, maxcoord, grid_length;
    grid->getMinMaxCoord (mincoord, maxcoord);
    grid_length = maxcoord - mincoord;
    const real L = grid_length.norm();  // characteristic length of grid

    // find characteristic size of displacement:
    real Umin, Umax; u_magnitude->minmax (Umin, Umax);

    // visual displacement in deformed_grid = Umax*magnification = L/10:
    magnification = L/10.0/Umax;
  }
  else
    magnification = 1.0; // real deformation of grid

  *deformed_grid = *grid;          // original configuration
  deformed_grid->move (*u, magnification);
  database->dump (*equiv_stress2); // equiv_stress2 uses deformed_grid
  // could bind other quantities to deformed_grid as well and dump them

  // Some extensions in case of 3D problems
  const int nsd = grid->getNoSpaceDim ();
  if ( nsd == 3 ) {
    Ptv(real) mincoord(nsd), maxcoord(nsd), displ(nsd);
    grid->getMinMaxCoord (mincoord, maxcoord);
    Handle(GridFE) grid_2D, deformed_grid_2D;
    grid_2D.rebind (new GridFE ());
    deformed_grid_2D.rebind (new GridFE ());
    String gridfile_2D = oform("P=PreproBox | d=2 [%g,%g]x[%g,%g] | d=2 e=ElmB4n2D [20,10] [1,1]", mincoord(
    readOrMakeGrid (*grid_2D, gridfile_2D);  // fill grid_2D
    *deformed_grid_2D = *grid_2D;  // will be deformed

    Handle(FieldFE) equiv_stress_2D, sigmax_2D, deformed_equiv_stress_2D;
    Handle(FieldsFE) u_2D;

    sigmax_2D.rebind (new FieldFE (*grid_2D, "sigmax_2D"));
    equiv_stress_2D.rebind (new FieldFE (*grid_2D, "equiv_stress_2D"));
    deformed_equiv_stress_2D.rebind
      (new FieldFE (*deformed_grid_2D, equiv_stress_2D->values (),
                    "deformed_equiv_stress_2D"));
    u_2D.rebind (new FieldsFE (*grid_2D, "u_2D"));

    Ptv(real) coord_3D(3), u_3D(3);
    coord_3D.fill (0.5*(mincoord(3)+maxcoord(3)));
```

```
    const int nno_2D = grid_2D->getNoNodes ();
    int i;
    for (i=1; i<=nno_2D; i++) {
      coord_3D(1) = grid_2D->getCoor (i,1);
      coord_3D(2) = grid_2D->getCoor (i,2);

      sigmax_2D->valueNode (i) =
        smooth_stress_measures()(1).valuePt (coord_3D);
      equiv_stress_2D->valueNode (i) =
        smooth_stress_measures()(no_stress_strain_components+1).
        valuePt (coord_3D);

      u->valuePt (u_3D, coord_3D);
      u_2D()(1).valueNode (i) = u_3D(1);
      u_2D()(2).valueNode (i) = u_3D(2);
    }

    deformed_grid_2D->move (*u_2D, 1.0);

    database->dump (*sigmax_2D);
    database->dump (*equiv_stress_2D);
    database->dump (*u_2D);
    database->dump (*deformed_equiv_stress_2D);
  }
}


// --------- automatic report generation ----------

void Elasticity2:: writeHeadings (StringList& headings)
{
  // define the names of the parameters to appear in the summary report:
  SimReport:: writeHeadings (headings, *grid);
  // example: headings.append ("my parameter");
  LinEqStatBlk::writeHeadings (headings);
  // etc
}

void Elasticity2:: writeResults (StringList& results)
{
  // write values of parameters (the parameters are defined in writeHeadings)
  // all "values" appended to the results list must be strings!
  // moreover, the order in the list must correspond to the order of the
  // names in writeHeadings

  SimReport:: writeResults (results, *grid);
  // example: results.append (aform("%f",my_parameter));
  lineq->getPerformance().writeResults (results);
  // etc
}

void Elasticity2:: writeExtendedResults (MultipleReporter& rep)
{
  // --- more verbose presentation of results in the detailed report: ---
  rep.multipleLoopSection(getMultipleLoopIndex(),SimCase::getMenuSystem());
  rep.subsection("Problem dependent parameters");
  rep.beginItemize();
  SimReport:: writeExtendedResults (rep, *grid);
  rep.put (aform("pressure at boind 1   : %f",pressure1));
  rep.put (aform("pressure at boind 2   : %f",pressure2));
  rho_format.  writeExtendedResults (rep);
  rep.put (aform("gravity direction : %s",g_dir.printAsIndex().chars()));
  rep.endItemize();

  rep.subsection("Summary of the equation solver");
  lineq->getPerformance().writeExtendedResults (rep, COMPACT);
```

```
  rep.subsection("Visualization of results");

  // Here one can add cross section plots etc...

  if (database->line1Defined()) {
    // user has defined line curve 1 in SaveSimRes
    SimReport:: makeCurvePlot
      (rep,
       database->cplotfile,
       "Gnuplotdpc", // plotting package, "Plotmtvdpc" is another choice
       "Cross",      // title of plot (use regex here)
       "s1",         // curve name, regex for curve along line 1
       ".",          // comment regex
       aform("Plot of the solution along line 1, run no. %d",
             getMultipleLoopIndex()),getMultipleLoopIndex(),
       NULL          // plotting program options
       );
  }
  // similar statements for line 2 (s1 -> s2)

  const int nsd = grid->getNoSpaceDim ();
  // make a plot of equivalent stress, using plotmtv:
  SimReport:: makeScalarFieldPlot
    (rep,     // the report object where the plot will be included
     equiv_stress1(),
     // plot equivalent stress
     database->resfile,
     DUMMY,   // point of time for plot (no meaning here in a stationary case)
     getMultipleLoopIndex(), // run no in a multiple loop
     2,       // 2: color plot, 1: contour plot, 3: elevated mesh
     nsd,       // 2D plot, 3 will give a perspective 3D plot
     // make a comment in the plot with no of nodes, element type etc:
     SimReport::compactInfo(*grid,&itg_rules,NULL).chars(),
     aform("The %s field in run no. %d",u->getFieldname().chars(),
           getMultipleLoopIndex()).chars(),   // figure caption
     NULL,  // plotting program (plotmtv) options
     "equiv_stress",
     // proper simres name of field to be plotted
     15,    // total width of figures in LaTeX (in cm)
     true,  // add finite element mesh to the field plot
     NULL); // vector of boundary indicator numbers for 3D grid plot

  // make a plot of equivalent stress over deformed grid, using plotmtv:
  SimReport:: makeScalarFieldPlot
    (rep,     // the report object where the plot will be included
     equiv_stress2(),
     // plot equivalent stress over deformed grid
     database->resfile,
     DUMMY,   // point of time for plot (no meaning here in a stationary case)
     getMultipleLoopIndex(), // run no in a multiple loop
     2,       // 2: color plot, 1: contour plot, 3: elevated mesh
     nsd,       // 2D plot, 3 will give a perspective 3D plot
     // make a comment in the plot with no of nodes, element type etc:
     SimReport::compactInfo(*grid,&itg_rules,NULL).chars(),
     aform("The %s field in run no. %d",u->getFieldname().chars(),
           getMultipleLoopIndex()).chars(),   // figure caption
     NULL,  // plotting program (plotmtv) options
     "equiv_stress_over_deformed_grid",
     // proper simres name of field to be plotted
     15,    // total width of figures in LaTeX (in cm)
     false, // finite element mesh is not added to the field plot
     NULL); // vector of boundary indicator numbers for 3D grid plot
}


void Elasticity2:: openReport ()
{
```

```
  report.open();
  report.header("Report from Simulation, class Elasticity2",
                "Harald Osnes",  // author
                true, true, purpose.chars());
  report.dumpMultipleFigures ();  // compact figure representation in LaTeX

  // initialization of summary part of MultipleEr:
  StringList headings;  // name of results to be Ed
  // always include input that data are varied in a multiple loop:
  SimCase::getMenuSystem().writeHeadings4multipleAnswers (headings);
  // write headings in user defined summary report:
  writeHeadings (headings);
  Strings headings_str;  headings.convert2vector (headings_str);
  report.putSummaryHeading ("Summary of Simulation, class Elasticity2",
                            "Core Dump", headings_str, true);
}


void Elasticity2:: closeReport ()
{
  report.endSummary();
  report.trailer();
}


void Elasticity2:: resultReport ()
{
  // write extended results in user defined detailed report:
  writeExtendedResults (report);

  // write summary:
  StringList results;  // name of results to be Ed
  // always include input that data are varied in a multiple loop:
  SimCase::getMenuSystem().writeResults4multipleAnswers (results);
  // write results in user defined summary report:
  writeResults (results);
  Strings results_str;  results.convert2vector (results_str);
  report.putSummary4oneSim (results_str, getMultipleLoopIndex());
}


void Elasticity2:: calcDmats ()
{
  real theta, E_L, E_T, nu_LT, G_LT, nu_TT;
  real c, s;

  // needed in 3D and plane strain 2D problems
  real D11, D12, D13, D14, D15, D16;
  real D22, D23, D24, D25, D26;
  real D33, D34, D35, D36;
  real D44, D45, D46;
  real D55, D56;
  real D66;

  // needed in plane stress 2D problems
  real S11, S12, S13, S22, S23, S33, S44, S55, S66;
  real SM11, SM13, SM33, SM66;

  int i;
  const int no_mat = elasticity_data.size ();

  for ( i=1; i<=no_mat; i++) {
    theta = elasticity_data(i)(1);
    E_L = elasticity_data(i)(2);
    E_T = elasticity_data(i)(3);
    nu_LT = elasticity_data(i)(4);
    G_LT = elasticity_data(i)(5);
```

```
nu_TT = elasticity_data(i)(6);

c = cos(theta*M_PI/180.0);
s = sin(theta*M_PI/180.0);

if ( (elasticity_tp == THREE_DIM) ||
     (elasticity_tp == PLANE_STRAIN) ) {
  const real denom1 = E_L*(-1.0 + sqr(nu_TT))
    + 2.0*sqr(nu_LT)*E_T*(1.0 + nu_TT);
  const real denom2 = E_L*(nu_TT - 1.0) + 2.0*sqr(nu_LT)*E_T;

  D11 = sqr(E_L)*(nu_TT - 1.0)/denom2;
  D12 = -E_T*E_L*nu_LT/denom2;
  D13 = D12;
  D14 = D15 = D16 = 0.0;

  D22 = -(E_L - sqr(nu_LT)*E_T)*E_T/denom1;
  D23 = -(nu_TT*E_L + sqr(nu_LT)*E_T)*E_T/denom1;
  D24 = D25 = D26 = 0.0;

  D33 = D22;
  D34 = D35 = D36 = 0.0;

  D44 = G_LT;
  D45 = D46 = 0.0;

  D55 = E_T/(2.0*(1.0 + nu_TT));
  D56 = 0.0;

  D66 = D44;

  if ( elasticity_tp == THREE_DIM ) {
    const real factor1 = s*c*D44*(c*c - s*s);

    Dmats(i)(1,1) = c*c*(D11*c*c + D12*s*s) + s*s*(D12*c*c + D22*s*s)
      + 4.0*s*s*c*c*D44;
    Dmats(i)(1,2) = c*c*(D11*s*s + D12*c*c) + s*s*(D12*s*s + D22*c*c)
      - 4.0*s*s*c*c*D44;
    Dmats(i)(1,3) = D13*c*c + D23*s*s;
    Dmats(i)(1,4) = s*c*(D11*c*c + D12*s*s) - s*c*(D12*c*c + D22*s*s)
      - 2.0*factor1;
    Dmats(i)(1,5) = 0.0;
    Dmats(i)(1,6) = 0.0;

    Dmats(i)(2,1) = Dmats(i)(1,2);
    Dmats(i)(2,2) = s*s*(D11*s*s + D12*c*c) + c*c*(D12*s*s + D22*c*c)
      + 4.0*s*s*c*c*D44;
    Dmats(i)(2,3) = D13*s*s + D23*c*c;
    Dmats(i)(2,4) = s*c*(D11*s*s + D12*c*c) - s*c*(D12*s*s + D22*c*c)
      + 2.0*factor1;
    Dmats(i)(2,5) = 0.0;
    Dmats(i)(2,6) = 0.0;

    Dmats(i)(3,1) = Dmats(i)(1,3);
    Dmats(i)(3,2) = Dmats(i)(2,3);
    Dmats(i)(3,3) = D33;
    Dmats(i)(3,4) = D13*s*c - D23*s*c;
    Dmats(i)(3,5) = 0.0;
    Dmats(i)(3,6) = 0.0;

    Dmats(i)(4,1) = Dmats(i)(1,4);
    Dmats(i)(4,2) = Dmats(i)(2,4);
    Dmats(i)(4,3) = Dmats(i)(3,4);
    Dmats(i)(4,4) = (D11 + D22 - 2.0*D12 - 2.0*D44)*s*s*c*c
      + D44*(s*s*s*s + c*c*c*c);
    Dmats(i)(4,5) = 0.0;
    Dmats(i)(4,6) = 0.0;
```

```
    Dmats(i)(5,1) = Dmats(i)(1,5);
    Dmats(i)(5,2) = Dmats(i)(2,5);
    Dmats(i)(5,3) = Dmats(i)(3,5);
    Dmats(i)(5,4) = Dmats(i)(4,5);
    Dmats(i)(5,5) = c*c*D55 + s*s*D66;
    Dmats(i)(5,6) = s*c*(-D55 + D66);

    Dmats(i)(6,1) = Dmats(i)(1,6);
    Dmats(i)(6,2) = Dmats(i)(2,6);
    Dmats(i)(6,3) = Dmats(i)(3,6);
    Dmats(i)(6,4) = Dmats(i)(4,6);
    Dmats(i)(6,5) = Dmats(i)(5,6);
    Dmats(i)(6,6) = s*s*D55 + c*c*D66;
  }
  else if ( elasticity_tp == PLANE_STRAIN ) {
    // Only some of the components (of Dmats above) are required in
    // 2-D plane strain problems
    //        | 11 13 16 |
    // Dmat = | 13 33 36 |, where, e.g., 11 refers to Dmats(i)(1,1) above
    //        | 16 36 66 |

    // To calculate the total stresses we also need a help mat. defined by
    //             | 12 23 26 |
    // Dmat_help = | 14 34 46 |
    //             | 15 35 56 |

    const real factor1 = s*c*D44*(c*c - s*s);

    // Dmats
    Dmats(i)(1,1) = c*c*(D11*c*c + D12*s*s) + s*s*(D12*c*c + D22*s*s)
      + 4.0*s*s*c*c*D44;
    Dmats(i)(1,2) = D13*c*c + D23*s*s;
    Dmats(i)(1,3) = 0.0;

    Dmats(i)(2,1) = Dmats(i)(1,2);
    Dmats(i)(2,2) = D33;
    Dmats(i)(2,3) = 0.0;

    Dmats(i)(3,1) = 0.0;
    Dmats(i)(3,2) = 0.0;
    Dmats(i)(3,3) = s*s*D55 + c*c*D66;


    // Dmats_help
    Dmats_help(i)(1,1) = c*c*(D11*s*s + D12*c*c) + s*s*(D12*s*s + D22*c*c)
      - 4.0*s*s*c*c*D44;
    Dmats_help(i)(1,2) = D13*s*s + D23*c*c;
    Dmats_help(i)(1,3) = 0.0;

    Dmats_help(i)(2,1) = s*c*(D11*c*c + D12*s*s) - s*c*(D12*c*c + D22*s*s)
      - 2.0*factor1;
    Dmats_help(i)(2,2) = D13*s*c - D23*s*c;
    Dmats_help(i)(2,3) = 0.0;

    Dmats_help(i)(3,1) = 0.0;
    Dmats_help(i)(3,2) = 0.0;
    Dmats_help(i)(3,3) = s*c*(-D55 + D66);
  }
}
else if ( elasticity_tp == PLANE_STRESS ) {
  S11 = 1.0/E_L;
  S12 = S13 = -nu_LT/E_L;
  S22 = S33 = 1.0/E_T;
  S23 = -nu_TT/E_T;
  S44 = S66 = 1.0/G_LT;
```

```
      S55 = 2.0*(1.0 + nu_TT)/E_T;

      SM11 = c*c*(S11*c*c + S12*s*s) + s*s*(S12*c*c + S22*s*s)
          + s*s*c*c*S66;
      SM13 = S13*c*c + S23*s*s;
      SM33 = S33;
      SM66 = s*s*S55 + c*c*S66;

      const real factor = SM11*SM33 - sqr(SM13);

      Dmats(i)(1,1) = SM33/factor;
      Dmats(i)(1,2) = Dmats(i)(2,1) = -SM13/factor;
      Dmats(i)(1,3) = Dmats(i)(3,1) = 0.0;
      Dmats(i)(2,2) = SM11/factor;
      Dmats(i)(2,3) = Dmats(i)(3,2) = 0.0;
      Dmats(i)(3,3) = 1.0/SM66;
    }
    else errorFP ("Elasticity2:: calcDmats",
                  "Only 2D plane strain, 2D plane stress and 3D problems "
                  "are implemented yet");
  }
}


void Elasticity2:: calcB0mats (const FiniteElement& fe_)
{
    // It is assumed that the matrices in B0mats are filled by zeroes initially
  int i;
  const int nbf = fe_.getNoBasisFunc ();

  if ( elasticity_tp == THREE_DIM ){
    for ( i=1; i<=nbf; i++) {
      B0mats(i)(1,1) = fe_.dN (i,1);
      B0mats(i)(2,2) = fe_.dN (i,2);
      B0mats(i)(3,3) = fe_.dN (i,3);
      B0mats(i)(4,1) = fe_.dN (i,2);
      B0mats(i)(4,2) = fe_.dN (i,1);
      B0mats(i)(5,2) = fe_.dN (i,3);
      B0mats(i)(5,3) = fe_.dN (i,2);
      B0mats(i)(6,1) = fe_.dN (i,3);
      B0mats(i)(6,3) = fe_.dN (i,1);
    }
  }
  else if ( (elasticity_tp == PLANE_STRESS) ||
            (elasticity_tp == PLANE_STRAIN) ) {
    for ( i=1; i<=nbf; i++) {
      B0mats(i)(1,1) = fe_.dN (i,1);
      B0mats(i)(2,2) = fe_.dN (i,2);
      B0mats(i)(3,1) = fe_.dN (i,2);
      B0mats(i)(3,2) = fe_.dN (i,1);
    }
  }
  else if ( elasticity_tp == AXISYMMETRY ) {
    for ( i=1; i<=nbf; i++) {
      B0mats(i)(1,2) = fe_.dN (i,1);
      B0mats(i)(2,1) = fe_.dN (i,2);
      B0mats(i)(3,1) = fe_.N (i)/fe_.getGlobalEvalPt ()(2);
      B0mats(i)(4,1) = fe_.dN (i,1);
      B0mats(i)(4,2) = fe_.dN (i,2);
    }
  }
  else
    errorFP ("Elasticity2:: calcB0mats",
             "elasticity_tp should be THREE_DIM, PLANE_STRAIN, PLANE_STRESS "
             "or AXISYMMETRY.");
}
```

```
void Elasticity2:: calcAmatThetaEps (const FiniteElement& fe_)
{
  // It is assumed that all the matrices and vectors are filled by zeroes
  // initially
  int i;
  const int nsd = fe_.getNoSpaceDim ();

  for (i = 1; i <= nsd; i++)
    u()(i).derivativeFEM (Du_pt(i), fe_);

  if ( elasticity_tp == THREE_DIM ){
    Amat(1,1) = Du_pt(1)(1);
    Amat(1,2) = Du_pt(2)(1);
    Amat(1,3) = Du_pt(3)(1);

    Amat(2,4) = Du_pt(1)(2);
    Amat(2,5) = Du_pt(2)(2);
    Amat(2,6) = Du_pt(3)(2);

    Amat(3,7) = Du_pt(1)(3);
    Amat(3,8) = Du_pt(2)(3);
    Amat(3,9) = Du_pt(3)(3);

    Amat(4,1) = Du_pt(1)(2);
    Amat(4,2) = Du_pt(2)(2);
    Amat(4,3) = Du_pt(3)(2);
    Amat(4,4) = Du_pt(1)(1);
    Amat(4,5) = Du_pt(2)(1);
    Amat(4,6) = Du_pt(3)(1);

    Amat(5,4) = Du_pt(1)(3);
    Amat(5,5) = Du_pt(2)(3);
    Amat(5,6) = Du_pt(3)(3);
    Amat(5,7) = Du_pt(1)(2);
    Amat(5,8) = Du_pt(2)(2);
    Amat(5,9) = Du_pt(3)(2);

    Amat(6,1) = Du_pt(1)(3);
    Amat(6,2) = Du_pt(2)(3);
    Amat(6,3) = Du_pt(3)(3);
    Amat(6,7) = Du_pt(1)(1);
    Amat(6,8) = Du_pt(2)(1);
    Amat(6,9) = Du_pt(3)(1);

    theta(1) = Du_pt(1)(1);
    theta(2) = Du_pt(2)(1);
    theta(3) = Du_pt(3)(1);
    theta(4) = Du_pt(1)(2);
    theta(5) = Du_pt(2)(2);
    theta(6) = Du_pt(3)(2);
    theta(7) = Du_pt(1)(3);
    theta(8) = Du_pt(2)(3);
    theta(9) = Du_pt(3)(3);

    // eps_pt = eps0_pt + epsL_pt, where epsL_pt = 0.5*Amat*theta;
    Amat.prod (theta, eps_pt);
    eps_pt.mult (0.5);
    eps_pt(1) += Du_pt(1)(1);
    eps_pt(2) += Du_pt(2)(2);
    eps_pt(3) += Du_pt(3)(3);
    eps_pt(4) += (Du_pt(1)(2) + Du_pt(2)(1));
    eps_pt(5) += (Du_pt(2)(3) + Du_pt(3)(2));
    eps_pt(6) += (Du_pt(3)(1) + Du_pt(1)(3));
  }
  else if ( (elasticity_tp == PLANE_STRESS) ||
            (elasticity_tp == PLANE_STRAIN)) {
```

```
    Amat(1,1) = Du_pt(1)(1);
    Amat(1,2) = Du_pt(2)(1);

    Amat(2,3) = Du_pt(1)(2);
    Amat(2,4) = Du_pt(2)(2);

    Amat(3,1) = Du_pt(1)(2);
    Amat(3,2) = Du_pt(2)(2);
    Amat(3,3) = Du_pt(1)(1);
    Amat(3,4) = Du_pt(2)(1);

    theta(1) = Du_pt(1)(1);
    theta(2) = Du_pt(2)(1);
    theta(3) = Du_pt(1)(2);
    theta(4) = Du_pt(2)(2);

    // eps_pt = eps0_pt + epsL_pt, where epsL_pt = 0.5*Amat*theta;
    Amat.prod (theta, eps_pt);
    eps_pt.mult (0.5);
    eps_pt(1) += Du_pt(1)(1);
    eps_pt(2) += Du_pt(2)(2);
    eps_pt(3) += (Du_pt(1)(2) + Du_pt(2)(1));
  }
  else
    errorFP ("Elasticity2:: calcAmatThetaEps",
             "Only implemented for 3D problems, more versions will come!");
}


void Elasticity2:: calcSystemMats (const FiniteElement& fe_)
{
  // It is assumed that matrices defined by Dmats are already calculated
  // and that the other matrices and vectors are filled by zeroes
  // initially
  int i;
  const int nbf = fe_.getNoBasisFunc ();

  const int material_number = fe_.grid().getMaterialType(fe_.getElmNo());

  calcB0mats (fe_);
  calcAmatThetaEps (fe_);    // eps_pt etc. are calculated here

  Dmats(material_number).prod (eps_pt, sigma_pt);

  if ( elasticity_tp == THREE_DIM ){
    Mmat(1,1) = sigma_pt(1);
    Mmat(1,4) = sigma_pt(4);
    Mmat(1,7) = sigma_pt(6);

    Mmat(2,2) = sigma_pt(1);
    Mmat(2,5) = sigma_pt(4);
    Mmat(2,8) = sigma_pt(6);

    Mmat(3,3) = sigma_pt(1);
    Mmat(3,6) = sigma_pt(4);
    Mmat(3,9) = sigma_pt(6);

    Mmat(4,1) = sigma_pt(4);
    Mmat(4,4) = sigma_pt(2);
    Mmat(4,7) = sigma_pt(5);

    Mmat(5,2) = sigma_pt(4);
    Mmat(5,5) = sigma_pt(2);
    Mmat(5,8) = sigma_pt(5);

    Mmat(6,3) = sigma_pt(4);
    Mmat(6,6) = sigma_pt(2);
```

```
    Mmat(6,9) = sigma_pt(5);

    Mmat(7,1) = sigma_pt(6);
    Mmat(7,4) = sigma_pt(5);
    Mmat(7,7) = sigma_pt(3);

    Mmat(8,2) = sigma_pt(6);
    Mmat(8,5) = sigma_pt(5);
    Mmat(8,8) = sigma_pt(3);

    Mmat(9,3) = sigma_pt(6);
    Mmat(9,6) = sigma_pt(5);
    Mmat(9,9) = sigma_pt(3);

    for ( i=1; i<=nbf; i++) {
      Gmats(i)(1,1) = fe_.dN (i,1);
      Gmats(i)(2,2) = fe_.dN (i,1);
      Gmats(i)(3,3) = fe_.dN (i,1);
      Gmats(i)(4,1) = fe_.dN (i,2);
      Gmats(i)(5,2) = fe_.dN (i,2);
      Gmats(i)(6,3) = fe_.dN (i,2);
      Gmats(i)(7,1) = fe_.dN (i,3);
      Gmats(i)(8,2) = fe_.dN (i,3);
      Gmats(i)(9,3) = fe_.dN (i,3);

      BLmats(i).prod (Amat, Gmats(i));
      Bbarmats(i).add (B0mats(i),BLmats(i));
    }
  }
  else if ( (elasticity_tp == PLANE_STRESS) ||
            (elasticity_tp == PLANE_STRAIN)) {
    Mmat(1,1) = sigma_pt(1);
    Mmat(1,3) = sigma_pt(3);

    Mmat(2,2) = sigma_pt(1);
    Mmat(2,4) = sigma_pt(3);

    Mmat(3,1) = sigma_pt(3);
    Mmat(3,3) = sigma_pt(2);

    Mmat(4,2) = sigma_pt(3);
    Mmat(4,4) = sigma_pt(2);

    for ( i=1; i<=nbf; i++) {
      Gmats(i)(1,1) = fe_.dN (i,1);
      Gmats(i)(2,2) = fe_.dN (i,1);
      Gmats(i)(3,1) = fe_.dN (i,2);
      Gmats(i)(4,2) = fe_.dN (i,2);

      BLmats(i).prod (Amat, Gmats(i));
      Bbarmats(i).add (B0mats(i),BLmats(i));
    }
  }
  else
    errorFP ("Elasticity2:: calcSystemMats",
             "Only implemented for 3D problems, more versions will come!");
}
```