OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG | MB | FAKULTÄT FÜR MASCHINENBAU

Material Plasticity
Evolution of the stiffness tetrad in fiber materials with large plastic strain

Dissertation
zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

von Dipl.-Ing. Martin Weber
geb. am 06.09.1986 in Staßfurt
genehmigt durch die Fakultät für Maschinenbau
der Otto-von-Guericke-Universität Magdeburg

Gutachter: Prof. Dr.-Ing. habil. Dr. h. c. mult. Holm Altenbach
Otto-von-Guericke-Universität Magdeburg

Prof. em. Dr.-Ing. Dr. E.h. Otto Bruhns
Ruhr-Universität Bochum

Prof. Dr.-Ing. habil. Rainer Glüge
Universität Bremen

Promotionskolloquium am 21.12.2020

Zusammenfassung

Das Ziel dieser Arbeit ist die Herleitung einer phänomenologischen Theorie für große plastische Deformationen, welche die Entwicklung der Anisotropie beschreibt. Dazu wird eine allgemeine Theorie ausgearbeitet und neben verschiedenen Sonderfällen wird der Fall einer sogenannten materiellen Plastizitätstheorie näher untersucht. Dabei wird angenommen, dass die Achsen der Anisotropie als materielle Linienelemente deformiert werden. Diese Theorie wird z.B. auf faserverstärkte Materialien anwendbar sein. Der Hauptunterschied zu einer gebräuchlichen Plastizitätstheorie besteht in der unterschiedlichen Entwicklung der Steifigkeitstetrade und der spannungsfreien Platzierung während einer plastischen Deformation. Zur Verifizierung der Ergebnisse dieser Theorie und zur Identifizierung und zum Vergleich der Steifigkeitstetraden vor und nach großen plastischen Deformationen wird ein repräsentatives Volumenelement (RVE) mit einer faserigen Mikrostruktur verwendet. Dazu benutzen wir das Finite-Elemente-Programm Abaqus und dessen Skriptsprache Python. Uni-, bi- und tri-direktional verstärkte Proben werden berücksichtigt. Auf der Mikroskala wird ein finites elastisch-plastisches Materialmodell verwendet, das auf den Konzepten der Isomorphie und der maximalen plastischen Dissipation basiert. Nach der Berechnung der effektiven Steifigkeitstetraden der verschiedenen Materialproben untersuchen wir deren Entwicklung bei verschiedenen Deformationen. Zusätzlich stellen wir einen Algorithmus vor, um den Abstand einer so gemessenen Steifigkeitstetrade zu den verschiedenen Symmetrieklassen zu bestimmen. Dazu entwickeln wir eine Projektionsmethode mit einem Tensor 8. Stufe, der alle Gruppenelemente der gewählten Symmetriegruppe abdeckt. Im Anschluss ist es notwendig herauszufinden, wie man die Entwicklung der Steifigkeitstetrade vorhersagen kann. Es stellt sich heraus, dass es möglich ist, diese Änderung mit ausreichender Genauigkeit mit einem zusätzlichen Tensor 2. Stufe zu beschreiben. Abschließend entwickeln wir eine analytische Evolutionsgleichung für diesen Tensor.

Abstract

The objective of this work is to develop a phenomenological finite plasticity theory which describes the evolution of the anisotropy. We develop a framework of a general finite plasticity theory, some special cases are examined and the case of a so called Material Plasticity is examined more closely, assuming that the axes of anisotropy deform as material line elements. It will be applicable to, e.g., fiber reinforced materials. The main difference to a common plasticity theory is the different evolution of the stiffness tetrad and of the stress-free placement during a plastic deformation. For a verification of the results of this theory and to identify and compare the stiffness tetrads before and after large plastic deformations, a representative volume element (RVE) with a fibrous microstructure is used. Therefore we use the finite element program Abaqus and its scripting language Python. Uni-, bi- and and tri-directionally reinforced samples are considered. On the micro-scale, a finite elastic-plastic material model based on the concepts of isomorphy and maximum plastic dissipation is used. After calculating the effective stiffnesses of the different material samples, we investigate their evolution during different deformations. In addition, we present a fast algorithm to determine the distance of such a measured stiffness tetrad to all possible symmetries of an elastic stiffness tetrad. Therefore, we develop a projection method using an 8^{th}-order tensor covering all group elements of the chosen symmetry group. Then it is necessary to find out how to predict the evolution of the stiffness tetrad. It turns out that it is possible to denote the change of the stiffness tetrad with sufficient accuracy using one additional 2^{nd}-order tensor. We finally propose an analytical evolution equation for this tensor.

Contents

List of Figures

List of Tables

1 Introduction

In modern engineering, one of the main challenges is to optimize processes regarding costs and efficiency. In simulation processes, the material model has a big impact on time efficiency and accuracy. All existing theories are a compromise between generality and applicability. On the one hand, there are detailed material models which are based on the micro-scale behavior of the material. In this case, the physical properties on the micro-scale are modeled. These calculations are very expensive especially in the case of structure models. On the other hand, there are phenomenological models which show the main features of the material on the macro-scale. These models provide fast calculations but the accuracy strongly depends on the mathematical model and the understanding of the material behavior.

The objective of this work is to develop a phenomenological finite plasticity theory which describes the evolution of the elastic anisotropy. We assume to have only small elastic strains. Therefore we use a linear law for the elastic part of our modeling approach. This is reasonable because the focus of this work lies in the evolution of anisotropic behavior during large plastic deformations. A related but fully fleshed-out theory for the evolution of the stiffness tetrad exists for Crystal Plasticity. Then, one can resort to a single crystal elasticity reference law and needs to consider only orientation changes, summarized as texture evolution in polycrystals. The crystal orientation distribution evolves due to the lattice spin and the effective elasticity is estimated by orientation averages of the single crystal stiffness. For this setting, textbook knowledge is available, e.g. Bunge (1982), Roters (2011), and Suwas and Ray (2014).

In other cases, the assumption of a constant elasticity reference law is not applicable, for example, when the elastic directors deform with the material. The setting in which the directors of elasticity move with the material is referred to as Material Plasticity after Forest and Parisot (2000) and Bertram (2012) due to the fact that the elastic directors behave like material line elements. The most obvious example is fiber-reinforced materials, which became very important in the last decades (H. Altenbach, J. Altenbach, and Kissing, 2018). For the orientation distribution evolution of short fibers in viscous flow, theories have been developed, starting from the seminal work Jeffery and Filon (1922) to industrial application Martinie and Roussel (2011).

The intention of a material theory is to describe the behaviour of a certain material or material class with common properties. One way to form general material equations (also called material laws, functionals, constitutive equations or material models) for them is to assume certain principles Bertram (2012)[p.153]. For example the principle of determinism which states that the stresses in a material point at a certain time are determined by the current (elastic material behaviour) and the past (inelastic material behaviour) motion of the body. In the large strain setting, the finite elasticity theory is well understood. There is a large amount of books and articles, for example, Truesdell and Noll (1965), Haupt (2000) and Lubarda (2002). In contrast, the finite plasticity, where the inelastic material behaviour is path-dependent, has conceptional problems. It is until today not clear how to define the plastic deformation. Already Casey and Naghdi (1992)[p.1257] stated: *"There is no general agreement on how it is to be identified, either conceptually or experimentally, in the presence of finite deformations."*

At the DFG-Meeting in December 2006 the anisotropic plasticity theory was announced as one of the most important future research topics in engineering mechanics. Today this statement increases in significance because the results in this field are still sparse, though the need of models is high. Some works within the last years produced important progress (Broese, Sfyris, and Tsakmakis, 2012; Freund, Shutov, and Ihlemann, 2012; Miehe, 1995, 1998a,b; Panhans and Kreißig, 2006; Pietryga, Vladimirov, and Reese, 2012; Reese, Pietryga, and Vladimirov, 2009; Shutov and Ihlemann, 2012; Shutov and Kreißig, 2008, 2010; Vladimirov, Pietryga, and Reese, 2011; Vladimirov and Reese, 2008; Xiao, Bruhns, and Meyers, 1999).

While the theory of finite isotropic plasticity is standing more or less on a secure ground this is not true for anisotropic plasticity. In this case, the multiplicative decomposition after E. Lee and Liu (1967) fails because the rotations stay undetermined. The concept of an isoclinic intermediate placement after Mandel (1973) also fails because it does not exist anymore in the case of anisotropy. This is irrelevant in the isotropic case but of fundamental importance in an anisotropic setting. If we think of the fact that nearly all materials are anisotropic after large plastic deformations the importance of this problem will become even clearer. Practical applications will be found e.g. in forming processes or crash simulations. Large deformations of fiber reinforced materials also result in an anisotropic structure which develops together with the material.

Mathematically, the difference of a Material Plasticity theory to the usual large strain plasticity theory with a multiplicative decomposition of the deformation gradient is that the evolution of the stiffness tetrad and of the stress-free placement during plastic deformation are decoupled. To work out the details, we motivate and extend the usual equations by a second-order tensor which accounts for the evolution of the stiffness tetrad. We also compare this extension to the isomorphy concept after Del Piero (1975), Noll (1958), Šilhavý and Kratochvíl (1977a,b), and Truesdell and

Noll (1965) and Bertram (1999, 2003, 2012). They have developed a finite plasticity theory which is based on the transformation of the elastic law during yielding. For many materials, it is known that the elastic behavior remains almost unchanged during large plastic deformations in the sense that in every elastic region after any plastic deformation the local elastic law is obtained from a push-forward of the elastic reference law. Examples are Crystal Plasticity and isotropic von Mises plasticity. In both ways, we end up with the same equation for a general plasticity theory. We identify four special cases for this equation, one being the equation of our Material Plasticity theory.

If we think of a Material Plasticity in the sense of Forest and Parisot (2000) the isomorphy concept will fail completely because the stiffness tetrad and thus the elastic law evolves completely different compared to the stress-free placement. Fiber reinforced materials are one example for this. The fibers deform together with the matrix material (Hufenbach, Langkamp, Adam, Krahl, Hornig, Zscheyge, and Modler (2011), Waffenschmidt, Polindara, Menzel, and Blanco (2014), Rastellini, Oller, Salomón, and Oñate (2008), Mitschang, Blinzler, and Wöginger (2003)). Another field of application could be twinning in crystals (Christian and Mahajan (1995), Glüge (2009)). One could also think of elastical damage mechanisms with an ansatz where the damage causes a weakening of the elastic properties leads to a modification of the stiffness tetrad of the material. Another example, constitute materials with lamella structure which, for example, occur during the solidification of grey cast iron (Berns and Theisen (2008), Bertolino and Perez-Ipiña (2006), Park and Verhoeven (1996)). The occurrence of lamellae is also observed in eutectic copper-silver-alloys which are used as a conductor material for magnets (Dodla, Bertram, and Krüger (2015), Grünberger, Heilmaier, and Schultz (2001), Sakai, Inoue, Asano, Wada, and Maeda (2001)). For such materials, an initially cubic elastic law can result in an elastic law with less symmetry or also a triclinic one. This means, we have completely no elastic isomorphy anymore.

Next, we like to compare other similar approaches from literature. Boehler (1987) proposed an invariant formulation of anisotropic constitutive equations. Within these equations, several scalar-valued functions appear which have to be identified in experimental investigations. In our work, we have to identify only three scalars. They can easily be determined using our RVE calculations which is rather simple and fast. Pastor, Turgeman, and Boehler (1990) uses the formulation of Boehler (1987) to define isotropic problems associated with originally orthotropic ones. Again, it is necessary to determine anisotropy coefficients and equations to define the association between the isotropic and the anisotropic problem. Menzel and Steinmann (2003) started with the multiplicative elasto-plasticity. Additionally, we also used the isomorphy concept as a second access to our equation. They introduce additional symmetric arguments into the scalar-valued isotropic tensor functions and develop a modular framework of the fundamental covariance relation. Again, a number of symmetric tensorial fields of second-order and scalar-valued internal variables are neces-

sary which, in contrast to our approach, leads to much greater identification effort. Lu and Papadopoulos (2004) also introduced a covariant formulation of anisotropic finite plasticity. Theoretical results are carried out and structural tensors for characterizing the anisotropy are introduced. In contrast to our work, the evolution of the anisotropy during the deformation is not mentioned. In Kaiser, Lu, Menzel, and Papadopoulos (2018, 2019) the covariant formulation of Lu and Papadopoulos for describing the evolution of the anisotropy is mentioned. They use a convected-type evolution equation for the structural tensors and an additional constitutive equation for the plastic spin. A close match up with the experimental results of Kim and Yin (1997) was reached. In Itskov and Aksel (2004) an elastic-plastic constitutive model for orthotropic materials at large strain is presented. Within the theory, an evolution of the anisotropy axes due to rigid rotations is modeled. They also use the multiplicative decomposition of the deformation gradient and a quadratic yield function formulated in terms of the Mandel stress tensor.

In this work, the theory of Material Plasticity has to be applicable to fiber reinforced materials and is compared with the results of RVE simulations done by the finite element program Abaqus. The benefit is the possibility to avoid the expensive RVE method to model large plastic deformation for fiber reinforced materials. We develop the framework of a general finite plasticity. Some special cases are examined and the case of a Material Plasticity theory is considered more closely. Its main feature is that the elasticity law changes during plastic deformations. To identify and compare the stiffness tetrads before and after large plastic deformations, a representative volume element (RVE) Glüge, Weber, and Bertram (2012) with a fibrous microstructure is generated. To capture the evolution of a stiffness tetrad, we introduce three example materials. These are a uni-directionally reinforced material having an hexagonal fiber arrangement and thus an effectively transversal isotropic symmetry, a bi-directionally reinforced material with tetragonal symmetry and a tri-directionally reinforced material with cubic symmetry. We use the program code Abaqus to model an RVE for each of them. The generation of the different RVEs is automated and parameterized using the scripting language Python and the corresponding interface of Abaqus. We developed an elastic-plastic material model for large deformations for the matrix and the fiber material using the isomorphy concept on the micro-scale both having different material parameters which are the Young's modulus E, the Poisson's ratio ν and the yield stress σ_{F}. Together with the RVE and the material model, we need a possibility to determine the effective stiffness of the initial material sample and in addition for the material sample after a large plastic deformation. This is done by performing small elastic test deformations and applying the difference quotient afterwards. We compare the stiffness tetrads before and after the deformation and find out that the stiffness tetrads evolve during large deformations. This means the isomorphy concept fails and it is necessary to introduce a separate evolution equation for the stiffness tetrad. To get a better understanding of this evolution, we investigate the change of the symmetry of the stiffness tetrad during these large deformations (Weber, Glüge, and Bertram, 2019). We introduce

a fast projection method to determine the distance of a measured stiffness tetrad to a specific symmetry class. We define a symmetry class as a set of all tetrads having a certain symmetry. Afterwards we examine the evolution of the symmetry for all three sample materials under different loads. With the better understanding of the stiffness tetrads, we finally find the sought evolution equation of the evolving stiffness tetrad introducing an additional second-order tensor with three free variables depending on the velocity gradient (Weber, Glüge, and H. Altenbach, 2020; Weber, Glüge, and Bertram, 2017).

1.1 Notation

A direct notation is preferred. Vectors are denoted as bold minuscules like $\boldsymbol{v} = v_i \boldsymbol{e}_i$. Summation over repeated indices is implied. The base vectors $\boldsymbol{e}_i$ are normalized and orthogonal for which $\boldsymbol{e}_i \cdot \boldsymbol{e}_j = \delta_{ij}$ holds with the Kronecker symbol δ_{ij}. The dyadic product and scalar contractions are denoted by $(\boldsymbol{a} \otimes \boldsymbol{b} \otimes \boldsymbol{c}) : (\boldsymbol{d} \otimes \boldsymbol{e}) = (\boldsymbol{b} \cdot \boldsymbol{d})(\boldsymbol{c} \cdot \boldsymbol{e})\boldsymbol{a}$, with a dot being the usual scalar product between vectors. Second-order tensors are written as bold majuscules, like $\boldsymbol{T} = T_{ij} \boldsymbol{e}_i \otimes \boldsymbol{e}_j$. The scalar contractions are carried out such that the positive definiteness is inherited to tensors, i.e. $\boldsymbol{A} : \boldsymbol{A} = A_{ij} A_{ij} \geq 0$. When the product is clear from the context, as in $\boldsymbol{a} = \boldsymbol{A}\boldsymbol{b}$, the single scalar dot is occasionally omitted. In the case of multiple scalar contractions, as in $\boldsymbol{T} = \mathbb{K} : \boldsymbol{E}$, we will always write the dots for better comprehensibility.

Normalized Voigt notation Fourth-order tensors (see e.g. Bertram (2012) and Bruhns (2003)) play a central role in this work. They are denoted by blackboard bold letters like the stiffness tetrad $\mathbb{K}$. It has the subsymmetries and the major symmetry which implies

$$K_{ijkl} = K_{ijlk} = K_{jikl} = K_{klij} \ , \tag{1.1}$$

where the components are given w.r.t. the base tetrads $\boldsymbol{e}_i \otimes \boldsymbol{e}_j \otimes \boldsymbol{e}_k \otimes \boldsymbol{e}_l$. A more convenient representation is obtained by introducing the symmetric base dyads

$$\boldsymbol{B}_1 = \boldsymbol{e}_x \otimes \boldsymbol{e}_x \qquad \boldsymbol{B}_2 = \boldsymbol{e}_y \otimes \boldsymbol{e}_y \tag{1.2}$$

$$\boldsymbol{B}_3 = \boldsymbol{e}_z \otimes \boldsymbol{e}_z \qquad \boldsymbol{B}_4 = \frac{1}{\sqrt{2}} (\boldsymbol{e}_x \otimes \boldsymbol{e}_y + \boldsymbol{e}_y \otimes \boldsymbol{e}_x) \tag{1.3}$$

$$\boldsymbol{B}_5 = \frac{1}{\sqrt{2}} (\boldsymbol{e}_x \otimes \boldsymbol{e}_z + \boldsymbol{e}_z \otimes \boldsymbol{e}_x) \qquad \boldsymbol{B}_6 = \frac{1}{\sqrt{2}} (\boldsymbol{e}_y \otimes \boldsymbol{e}_z + \boldsymbol{e}_z \otimes \boldsymbol{e}_y), \tag{1.4}$$

for which $\boldsymbol{B}_i : \boldsymbol{B}_j = \delta_{ij}$, $i, j = 1 \ldots 6$. W.r.t. the base tetrads $\boldsymbol{B}_i \otimes \boldsymbol{B}_j$ the components K_{ij} of $\mathbb{K} = K_{ij} \boldsymbol{B}_i \otimes \boldsymbol{B}_j$ can be arranged in a symmetric 6×6 matrix, for which the usual rules of matrix calculus hold since the basis is orthonormal. The nota-

tion was used implicitly firstly by Thomson (1856), rediscovered by Mandel (1965) and popularized by Voigt (1910) in its non-normalized form, see also Cowin and Mehrabadi (1992) and Brannon (2018). Other bases are not used in this work.

Rayleigh product We further make use of the Rayleigh product which is defined between a second-order tensor and a tensor of arbitrary order,

$$\boldsymbol{A} * \overset{\langle n \rangle}{\mathbb{K}} = K_{\underbrace{i \dots p}_{n \text{ indices}}} \underbrace{(\boldsymbol{A}\boldsymbol{e}_i) \otimes \dots \otimes (\boldsymbol{A}\boldsymbol{e}_p)}_{n \text{ times}}, \qquad K_{\underbrace{i \dots p}_{n \text{ indices}}} = \overset{\langle n \rangle}{\mathbb{K}} \underbrace{\cdot \dots \cdot}_{n \text{ dots}} \underbrace{\boldsymbol{e}_i \otimes \dots \otimes \boldsymbol{e}_p}_{n \text{ times}}. \quad (1.5)$$

It changes the basis while keeping the components, which can be used, e.g., to represent rotations. For second-order tensors it can be written as $\boldsymbol{A} * \boldsymbol{T} = \boldsymbol{A}\boldsymbol{T}\boldsymbol{A}^{\mathrm{T}}$. It is associative in the first factor, i.e. $(\boldsymbol{A}\boldsymbol{B}) * \mathbb{K} = \boldsymbol{A} * (\boldsymbol{B} * \mathbb{K})$ and linear in the second factor, i.e. $\boldsymbol{A} * (\mathbb{K}_1 + \alpha\mathbb{K}_2) = \boldsymbol{A} * \mathbb{K}_1 + \alpha \boldsymbol{A} * \mathbb{K}_2$.

Differences between tetrads Differences between stiffness tetrads are normalized w.r.t. $\mathbb{K}_1$, where $\mathbb{K}_0$ is the initial stiffness and $\mathbb{K}_1$ the stiffness after plastic deformation. Depending on the context, $\mathbb{K}_1$ is the result of a numerical RVE experiment or a phenomenological model prediction, for which $\mathbb{K}_1$ can be replaced by an expression with $\mathbb{K}_0$. To quantify a deviation we take the Euclidean norm. The Euclidean norm of a fourth-order tensor is calculated by the square root of the 4-fold contraction of the tensor with itself. For example, the normalized difference between a phenomenological model prediction and an RVE result is given by

$$\|\mathbb{K}_1^{\mathrm{RVE}} - \mathbb{K}_1^{\mathrm{Model}}\| / \|\mathbb{K}_1^{\mathrm{RVE}}\| \,. \quad (1.6)$$

Another interesting result is the change of stiffness due to plastic deformations, i.e. the difference between initial and current stiffnesses.

1.2 List of frequently used symbols and operations

a,b,c,...	scalars, indices, constants
$\boldsymbol{a}$,$\boldsymbol{b}$,$\boldsymbol{c}$,...	vectors
a_i, A_{ij}, A_{ijkl}	components of $\boldsymbol{a}$, $\boldsymbol{A}$ and $\mathbb{A}$ w.r.t. $\boldsymbol{e}_i$
$\boldsymbol{A}$,$\boldsymbol{B}$,$\boldsymbol{C}$,...	tensors of second-order
$\mathbb{A}$,$\mathbb{B}$,$\mathbb{C}$,...	tensors of fourth-order

$\overset{<8>}{\mathbb{A}}$, $\overset{<8>}{\mathbb{B}}$, $\overset{<8>}{\mathbb{C}}$,...	tensors of eighth order
$\|\mathbb{A}\| = \sqrt{\mathbb{A} :: \mathbb{A}}/\mathbb{A}$	normalized Euclidean norm of a fourth-order tensor $\mathbb{A}$
$\{\boldsymbol{B}_i\}$	symmetric base dyads
$\boldsymbol{C} = \boldsymbol{F}^{\mathrm{T}}\boldsymbol{F}$	right Cauchy-Green tensor
$\{\boldsymbol{e}_i\}$	orthonormal vector basis
$\overset{\mathrm{m}}{\boldsymbol{E}} = \frac{1}{\mathrm{m}}(\boldsymbol{C}_{\mathrm{e}}^{\mathrm{m}/2} - \boldsymbol{I})$	Seth strain tensor
$\overset{\mathrm{G}}{\boldsymbol{E}} = \frac{1}{2}(\boldsymbol{C} - \boldsymbol{I})$	Green strain tensor
$\boldsymbol{F} = \boldsymbol{F}_{\mathrm{e}}\boldsymbol{F}_{\mathrm{p}}$	deformation Gradient with multiplicative decomposition
$\boldsymbol{F}_{\mathrm{e}}$	elastic part of the deformation gradient
$\boldsymbol{F}_{\mathrm{p}}$	plastic part of the deformation gradient
$\boldsymbol{H} = \mathrm{G}rad\,(\boldsymbol{u})$	displacement gradient
$\boldsymbol{I}$	unit tensor of 2^{nd}-order
$\mathbb{I}$	unit tensor of 4^{th}-order
$J = \det\boldsymbol{F}$	determinant of the deformation gradient
$\mathbb{K}$	material stiffness tetrad
$\boldsymbol{L} = grad\,(\boldsymbol{v}) = \dot{\boldsymbol{F}}\boldsymbol{F}^{-1}$	velocitiy gradient
$\boldsymbol{n}$	normal vector
$\boldsymbol{P}_{\mathrm{C}} = \boldsymbol{F}_{\mathrm{p}}^{-1}$	transformation tensor of the stress-free placement
$\boldsymbol{P}_{\mathrm{K}}$	transformation tensor of tangent vectors
$\boldsymbol{Q}$	orthogonal rotation tensor
$\boldsymbol{T}$	Cauchy stress tensor

$\overset{\text{1PK}}{\mathbf{T}} = J\boldsymbol{T}\boldsymbol{F}^{-\text{T}}$	1st Piola-Kirchhoff stress tensor
$\overset{\text{2PK}}{\boldsymbol{T}} = J\boldsymbol{F}^{-1}\boldsymbol{T}\boldsymbol{F}^{-\text{T}}$	2nd Piola-Kirchhoff stress tensor
$\boldsymbol{t} = \boldsymbol{T}\boldsymbol{n}$	stress vector on surface
$\boldsymbol{u}$	displacement vector
$\boldsymbol{v}$	velocitiy vector
w	elastic strain energy

2 Material Plasticity

In this chapter, we want to derive a general equation for a plasticity theory and subsequently the Material Plasticity theory as mentioned in the introduction as a special case.

2.1 Plasticity theory with the evolution of the stiffness tetrad

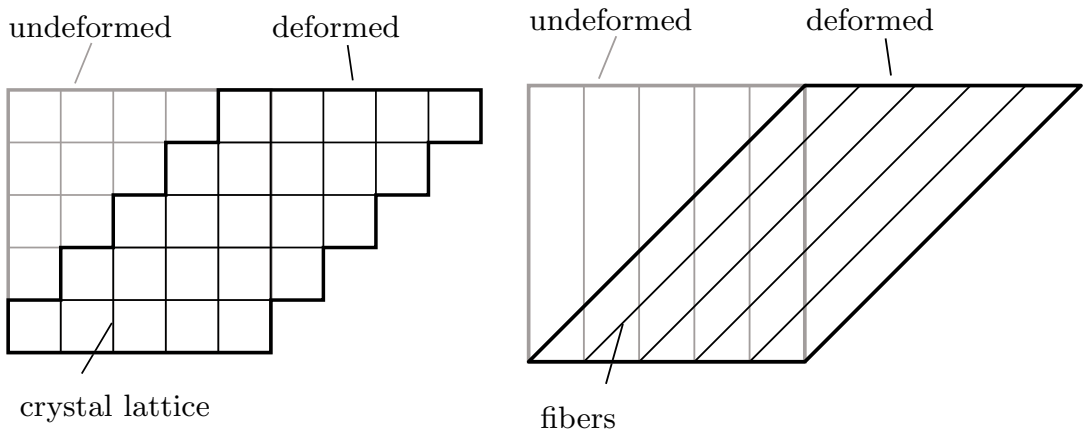
As explained in the introduction, the aim of this work is to develop a phenomenological finite plasticity theory that describes the evolution of anisotropy. To explain this idea, a comparison with the known Crystal Plasticity theory is helpful. In the latter case, under plastic deformations, the crystal structure develops differently than the material (see Fig. 2.1(a)). In the theory of Material Plasticity, the opposite occurs. If there is a plastic deformation in the fiber and the matrix material, the fibers will deform together with the material and not differently. This means that the anisotropy induced by the fibers, in contrast to the crystal lattice, is firmly bonded with the material. The best example here is a fiber-reinforced material (see Fig. 2.1(b)).

2.2 Derivation of the theory with the multiplicative decomposition

For creating a phenomenological model for a Material Plasticity theory, we start with the kinematics. We define the reference placement, the stress-free placement, and the current placement shown in Fig. 2.2. The multiplicative decomposition of the deformation gradient $\boldsymbol{F}$ (see e.g. Bruhns (2015), H. Altenbach (2018)) is

$$\boldsymbol{F} = \boldsymbol{F}_{\mathrm{e}}\boldsymbol{F}_{\mathrm{p}} \ . \tag{2.1}$$

Next, we choose a general elastic law in the stress-free placement with a material stress tensor $\overset{\text{mat}}{\boldsymbol{T}}$, the stiffness tetrad of the material as a tensor of fourth-order $\mathbb{K}$



(a) Crystal Plastictiy (b) Material Plasticity

Figure 2.1: Introduction of a Material Plasticity theory with fibers deforming together with the material on the right-hand side. The classical Crystal Plasticity theory is depicted on the left-hand side. Here, the orientation of the crystal lattice remains the same with respect to the shear direction and slip plane normal while the material deforms.

and a strain measure. We use the Seth strain tensors

$$\overset{\mathrm{m}}{\boldsymbol{E}} = \frac{1}{\mathrm{m}}(\boldsymbol{C}_{\mathrm{e}}^{\mathrm{m}/2} - \boldsymbol{I}) \tag{2.2}$$

with "m" being a real parameter and "e" marking the elastic part of a variable Seth (1964). A general framework can simply be set up by postulating quadratic elastic strain energy in terms of the Seth strain tensors,

$$w = \frac{1}{2}\mathbb{K} :: \overset{\mathrm{m}}{\boldsymbol{E}} \otimes \overset{\mathrm{m}}{\boldsymbol{E}} \,, \tag{2.3}$$

We assumed small elastic deformations, such that it is sufficient to account only for the quadratic term in the strain energy. It would be possible to present our proposal with a general potential. A generalization could be obtained by a series expansion of the strain energy:

$$\begin{aligned} w = & \frac{1}{2}\overset{4}{\mathbb{K}} :: \overset{\mathrm{m}}{\boldsymbol{E}} \otimes \overset{\mathrm{m}}{\boldsymbol{E}} \\ & + \frac{1}{3}\overset{6}{\mathbb{K}} ::: \overset{\mathrm{m}}{\boldsymbol{E}} \otimes \overset{\mathrm{m}}{\boldsymbol{E}} \otimes \overset{\mathrm{m}}{\boldsymbol{E}} \\ & + \frac{1}{4}\overset{8}{\mathbb{K}} :::: \overset{\mathrm{m}}{\boldsymbol{E}} \otimes \overset{\mathrm{m}}{\boldsymbol{E}} \otimes \overset{\mathrm{m}}{\boldsymbol{E}} \otimes \overset{\mathrm{m}}{\boldsymbol{E}} \\ & + \dots \,. \end{aligned} \tag{2.4}$$

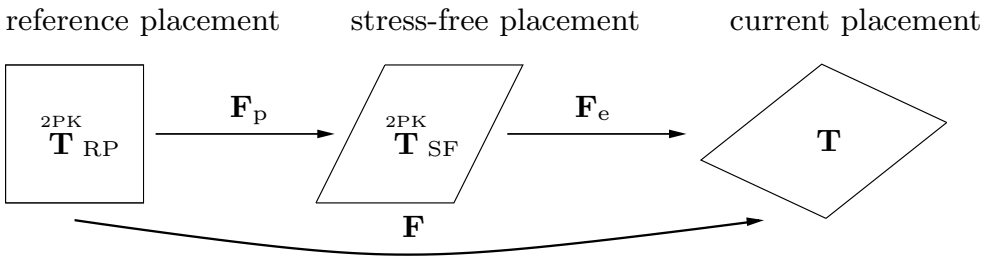


Figure 2.2: Scheme for introducing the used placements, stress tensors and the multiplicative decomposition

One could then approach the change of all higher-order $\mathbb{K}$ tensors similar to our strategy for $\mathbb{K}$ (4^{th} order) given below. It would be interesting to assess the quality of such an approach, but it this beyond the scope of our work. The elastic law is

$$\overset{\text{mat}}{\boldsymbol{T}} = \mathbb{K} : \overset{\text{m}}{\boldsymbol{E}} \ . \tag{2.5}$$

We write the elastic part $\boldsymbol{C}_e$ as

$$\boldsymbol{C}_e = \boldsymbol{F}_e^{\text{T}} \boldsymbol{F}_e = \boldsymbol{F}_\text{p}^{-\text{T}} \boldsymbol{F}^{\text{T}} \boldsymbol{F} \boldsymbol{F}_\text{p}^{-1} = \boldsymbol{F}_\text{p}^{-\text{T}} \boldsymbol{C} \boldsymbol{F}_\text{p}^{-1} \ . \tag{2.6}$$

Now we insert this equation into Eq. (2.2) and expand the $\boldsymbol{I}$ with $\boldsymbol{F}_\text{p}$ to

$$\overset{\text{m}}{\boldsymbol{E}} = \frac{1}{\text{m}}((\boldsymbol{F}_\text{p}^{-\text{T}} \boldsymbol{C} \boldsymbol{F}_\text{p}^{-1})^{\text{m}/2} - \boldsymbol{F}_\text{p}^{-\text{T}} \boldsymbol{F}_\text{p}^{\text{T}} \boldsymbol{F}_\text{p} \boldsymbol{F}_\text{p}^{-1}) \ . \tag{2.7}$$

If we confine to the special case $\text{m} = 2$, it will be possible to factor out $\boldsymbol{F}_\text{p}$, and thus we can further simplify to

$$\overset{\text{m}}{\boldsymbol{E}} = \frac{1}{2} \boldsymbol{F}_\text{p}^{-\text{T}} (\boldsymbol{C} - \boldsymbol{F}_\text{p}^{\text{T}} \boldsymbol{F}_\text{p}) \boldsymbol{F}_\text{p}^{-1} \ . \tag{2.8}$$

The choice m=2 results in further simplifications. The stress tensor $\overset{\text{mat}}{\boldsymbol{T}}$ becomes $\overset{\text{2PK}}{\mathbf{T}}_{\text{SF}}$ in the stress free placement. This allows for a simple conversion to the Cauchy stresses $\boldsymbol{T}$. The next step is to introduce our Ansatz for capturing the evolution of the anisotropy axes during the plastic deformation. We know that the anisotropy axes transform by vectors. We will call the initial vector $\boldsymbol{a}_0$ and the vector of the transformed axis $\boldsymbol{a}$. Second-order tensors transform tangent vectors that represent line elements. We will name this tensor $\boldsymbol{P}_\text{K}$ with

$$\boldsymbol{a} = \boldsymbol{P}_\text{K} \boldsymbol{a}_0 \ . \tag{2.9}$$

Let us consider a transversal isotropic material. From representation theory, we can write the material stiffness tetrad $\mathbb{K}^{\mathrm{ti}}$ of this material as

$$\begin{aligned}\mathbb{K}^{\mathrm{ti}} =& c_1\mathbb{I} + c_2\boldsymbol{I}\otimes\boldsymbol{I} + c_3\boldsymbol{a}\otimes\boldsymbol{a}\otimes\boldsymbol{a}\otimes\boldsymbol{a} \\ &+ c_4(\boldsymbol{I}\otimes\boldsymbol{a}\otimes\boldsymbol{a})_{\mathrm{sym}} + c_5(\boldsymbol{a}\otimes\boldsymbol{I}\otimes\boldsymbol{a})_{\mathrm{sym}} \ ,\end{aligned} \tag{2.10}$$

where "sym" is the symmetrization according to Eq. (1.1). Because we know this representation and there is only one vector $\boldsymbol{a}$, we can transform the whole stiffness tetrad by transforming the vector $\boldsymbol{a}$. In other cases, we either do not know the representation theorem or it changes during plastic deformation due to a change of the symmetry class (e.g. cubic symmetry). As our theory has to be valid for all possible material symmetries, we have to make a simplification. In Eq. (2.10), we identify the summand with c_3 as the part capturing the main anisotropy features. The structure is similar to the Rayleigh product so we decide to directly apply the transformation $\boldsymbol{P}_{\mathrm{K}}$ to the whole stiffness tetrad $\mathbb{K}$

$$\mathbb{K} = \boldsymbol{P}_{\mathrm{K}} * \mathbb{K}_0 \ . \tag{2.11}$$

$\boldsymbol{P}_{\mathrm{K}}$ is defined in the stress-free placement and transforms the stiffness tetrad. It is not a change of placement. Furthermore, it is now possible to summarize $\boldsymbol{F}_{\mathrm{p}}$ and $\boldsymbol{P}_{\mathrm{K}}$ in the following equation and compare it with other theories. In the stress-free placement, the 2nd Piola-Kirchhoff stresses are

$$\overset{\mathrm{2PK}}{\boldsymbol{T}}_{\mathrm{SF}} = (\boldsymbol{P}_{\mathrm{K}} * \mathbb{K}_0) : \frac{1}{2}\boldsymbol{F}_{\mathrm{p}}^{-\mathrm{T}}(\boldsymbol{C} - \boldsymbol{F}_{\mathrm{p}}^{\mathrm{T}}\boldsymbol{F}_{\mathrm{p}})\boldsymbol{F}_{\mathrm{p}}^{-1} \ . \tag{2.12}$$

With the transformation between the stress-free placement and the reference placement using $\boldsymbol{F}_{\mathrm{p}}$ and its determinant J_{p}

$$\overset{\mathrm{2PK}}{\boldsymbol{T}}_{\mathrm{RP}} = \boldsymbol{F}_{\mathrm{p}}^{-1}\overset{\mathrm{2PK}}{\boldsymbol{T}}_{\mathrm{SF}}\boldsymbol{F}_{\mathrm{p}}^{-\mathrm{T}}J_{\mathrm{p}} \ , \tag{2.13}$$

we can factor out $\boldsymbol{F}_{\mathrm{p}}$ and summarize with $\boldsymbol{P}_{\mathrm{K}}$. We consider only isochoric plastic deformation and therefore $J_{\mathrm{p}} = 1$. Finally, we end up with the following equation for the 2nd Piola-Kirchhoff stresses in the reference placement

$$\overset{\mathrm{2PK}}{\boldsymbol{T}}_{\mathrm{RP}} = \frac{1}{2}\left(\boldsymbol{F}_{\mathrm{p}}^{-1}\boldsymbol{P}_{\mathrm{K}}\right) * \mathbb{K}_0 : \left(\boldsymbol{C} - \boldsymbol{F}_{\mathrm{p}}^{T}\boldsymbol{F}_{\mathrm{p}}^{-1}\right) \tag{2.14}$$

2.3 Derivation of the theory with the isomorphy concept

We want to find another access to Eq. (2.14) and start with a physically linear elastic law according to St. Venant-Kirchhoff

$$\overset{\text{2PK}}{\boldsymbol{T}}_0(\boldsymbol{C}) = \frac{1}{2}\mathbb{K}_\text{p} : (\boldsymbol{C} - \boldsymbol{C}_\text{up}) \ . \tag{2.15}$$

Denoting $\overset{\text{2PK}}{\boldsymbol{T}}_0$ as a constant elastic reference law and $\overset{\text{2PK}}{\boldsymbol{T}}_\text{p}$ as the current elastic law describing the material at any given time, the isomorphism $\boldsymbol{P}$ holds for the elastic law of an elastic-plastic material

$$\overset{\text{2PK}}{\boldsymbol{T}}_\text{p}(\boldsymbol{C}) = \boldsymbol{P}\overset{\text{2PK}}{\boldsymbol{T}}_0\left(\boldsymbol{P}^\text{T}\boldsymbol{C}\boldsymbol{P}\right)\boldsymbol{P}^\text{T} \ . \tag{2.16}$$

Substituting Eq. (2.15) into Eq. (2.16), one obtains the generalized form of a plasticity theory with isomorphic elastic ranges according to Bertram (2012)[p.285ff]. $\boldsymbol{P}$ is also called a plastic transformation and describes the change of the stress-free placement ($\boldsymbol{C}_\text{u0}$) and the stiffness tetrad ($\mathbb{K}_0$)

$$\overset{\text{2PK}}{\boldsymbol{T}}_\text{p}(\boldsymbol{C}) = \frac{1}{2}\left(\boldsymbol{P} * \mathbb{K}_0\right) : \left(\boldsymbol{C} - \boldsymbol{P}^{-T} * \boldsymbol{C}_\text{u0}\right) \ . \tag{2.17}$$

For our theory, the constant elastic reference law has to be replaced. A generalization of the isomorphy concept is to choose two different plastic transformations $\boldsymbol{P}_\text{K}$ and $\boldsymbol{P}_\text{C}$, where $\boldsymbol{P}_\text{C}$ transforms the stress-free configuration and $\boldsymbol{P}_\text{C}\boldsymbol{P}_\text{K}$ transforms the elastic stiffness into the stress-free configuration:

$$\overset{\text{2PK}}{\boldsymbol{T}}_\text{p}(\boldsymbol{C}) = \frac{1}{2}\left(\boldsymbol{P}_\text{C}\boldsymbol{P}_\text{K} * \mathbb{K}_0\right) : \left(\boldsymbol{C} - \boldsymbol{P}_\text{C}{}^{-T} * \boldsymbol{C}_\text{u0}\right) \ . \tag{2.18}$$

With the relation

$$\boldsymbol{P}_\text{C} = \boldsymbol{F}_\text{p}^{-1} \tag{2.19}$$

and the free choice of the stress-free placement as

$$\boldsymbol{C}_\text{u0} = \boldsymbol{I} \ , \tag{2.20}$$

we obtain Eq. (2.14) for our general plasticity theory. In this equation, the isomorphy condition no longer holds because there are two different plastic transformations. With $Orth^+$ defining the special orthogonal group (rotational group with determinant $+1$) and $\boldsymbol{P}_\text{K} \in Orth^+$, we are able to combine both transformations to one. This is always possible for crystals where $\boldsymbol{P}_\text{K} \in Orth^+$ always holds. Now we want to examine this equation for possible special cases.

2.4 Special cases for the general equation

In this section, we use the special unimodular group "$Unim^{+}$", defining all tensors with determinant equal to $+1$. The special linear group "Inv^{+}" defines all invertible tensors with positive determinant. By choosing $\boldsymbol{P}_{\mathrm{K}} = \boldsymbol{P}_{\mathrm{C}} = \boldsymbol{I}$, it is clear that this will lead to an elasticity theory which can be used for elastic materials or the elastic range of a material (see Table 2.1, column 1)

$$\overset{\text{2PK}}{\boldsymbol{T}} = \frac{1}{2}\mathbb{K} : \left(\boldsymbol{C} - \boldsymbol{I}\right) \ . \tag{2.21}$$

In the case that $\boldsymbol{P}_{\mathrm{K}} = \boldsymbol{I}$ and $\boldsymbol{P}_{\mathrm{C}} \in Unim^{+}$, we will obtain the isomorphy concept which is e.g. applicable as Crystal Plastictiy theory as described in Bertram (2012) where only one plastic transformation exists (see Table 2.1, column 2)

$$\overset{\text{2PK}}{\boldsymbol{T}} = \frac{1}{2}\boldsymbol{P}_{\mathrm{C}} * \mathbb{K} : \left(\boldsymbol{C} - \boldsymbol{P}_{\mathrm{C}}^{-\mathrm{T}}\boldsymbol{P}_{\mathrm{C}}^{-1}\right) \ . \tag{2.22}$$

By choosing $\boldsymbol{P}_{\mathrm{K}} \in Inv^{+}$ and $\boldsymbol{P}_{\mathrm{C}} = \boldsymbol{I}$, one would describe evolving material properties without a deformation of the material. This could be applicable to the description of some recrystallization processes where $\boldsymbol{P}_{\mathrm{K}} = \boldsymbol{Q}$ (see Table 2.1, column 3)

$$\overset{\text{2PK}}{\boldsymbol{T}} = \frac{1}{2}\boldsymbol{P}_{\mathrm{K}} * \mathbb{K} : \left(\boldsymbol{C} - \boldsymbol{I}\right) \ . \tag{2.23}$$

$\boldsymbol{Q}$ is an orthogonal tensor of second-order and refers to a rotation. We refer to a recrystallization process on the micro-scale, where each material point has just one lattice orientation. This lattice orientation can change by recrystallization, namely when grain boundaries move upon coarse-grain annealing. The local orientation change can be expressed by a single orthogonal tensor. On the macro-scale, the orientation distribution evolves, which is, of course, not captured by a single second-order tensor. The last possible combination is to allow a different evolution of $\boldsymbol{P}_{\mathrm{K}}$ and $\boldsymbol{P}_{\mathrm{C}}$ and thus a different evolution of material properties and the stress-free placement. In this case, one can describe e.g. fiber-reinforced materials with the idea of a Material Plasticity theory explained in the following (see Table 2.1, column 4). The final equation for our theory is

$$\overset{\text{2PK}}{\boldsymbol{T}} = \frac{1}{2}\left(\boldsymbol{P}_{\mathrm{C}}\boldsymbol{P}_{\mathrm{K}}\right) * \mathbb{K} : \left(\boldsymbol{C} - \boldsymbol{P}_{\mathrm{C}}^{-\mathrm{T}}\boldsymbol{P}_{\mathrm{C}}^{-1}\right) \ . \tag{2.24}$$

Elasticity	Crystal plasticity	Evolving material	Material plasticity
$\boldsymbol{P}_{\mathrm{K}} = \boldsymbol{I}$ $\boldsymbol{P}_{\mathrm{C}} = \boldsymbol{I}$	$\boldsymbol{P}_{\mathrm{K}} = \boldsymbol{I}$ $\boldsymbol{P}_{\mathrm{C}} \in \mathit{Unim}^{+}$	$\boldsymbol{P}_{\mathrm{K}} \in \mathit{Inv}^{+}$ $\boldsymbol{P}_{\mathrm{C}} = \boldsymbol{I}$	$\boldsymbol{P}_{\mathrm{K}} \in \mathit{Inv}^{+}$ $\boldsymbol{P}_{\mathrm{C}} \in \mathit{Unim}^{+}$
St. Venant-Kirchhoff	crystal structure is preserved under plastic deformations	evolving material properties	different evolution of material properties and stress-free placement
elastic materials	materials with crystal structure	recrystallisation ($\boldsymbol{P}_{\mathrm{K}} = \boldsymbol{Q}$)	fiber reinforced materials

Table 2.1: Overview of possible special cases for the general equation

2.5 Summary

In this chapter, we introduced the idea of a so called Material Plasticity theory to track the evolution of the anisotropy during a plastic deformation. We explained two different ways to access the general equation Eq. (2.14). By investigating this equation, we formulated four special cases and summarized them in Table 2.1. One of the special case is the Material Plasticity theory. The final equation reads:

$$\overset{\text{2PK}}{\boldsymbol{T}} = \frac{1}{2}\left(\boldsymbol{P}_{\mathrm{C}}\boldsymbol{P}_{\mathrm{K}}\right) * \mathbb{K} : \left(\boldsymbol{C} - {\boldsymbol{P}_{\mathrm{C}}}^{-T}{\boldsymbol{P}_{\mathrm{C}}}^{-1}\right) \tag{2.25}$$

with $\boldsymbol{P}_{\mathrm{K}}$ being a second-order tensor. This tensor should be able to transform the stiffness tetrad during a plastic deformation. The further investigation of the evolution of the stiffness tetrad will be investigated in Chapter 5.

3 Representative volume elements

With the increase in the requirements for computer simulation, the requirements for the accuracy of the material modeling are also growing. Increasingly, it makes sense to capture microstructural details of certain materials and to take them into account in the material law through so-called homogenization at the macroscopic level. Examples of such materials are polycrystals showing a morphological or crystallographic texture, fiber- or particle-reinforced composites, and foams and laminates in which the microstructural details have a decisive influence on material behavior. So-called representative volume elements (RVEs) are used for the efficient integration of these details with the finite element method. This chapter summarizes the basics of the RVE method and explains the automated generation into the finite element system Abaqus using a Python script.

3.1 Introduction of representative volume elements

Representative volume elements are used for the effective calculation of material properties of complex structures with e.g. inclusions or cracks. Mathematically speaking, an RVE is the length scale of a single heterogeneity, with which the material appears evenly and thus the continuum concepts can be applied Shan and Gokhale (2002). Until now, the method of RVEs has been defined differently by different researchers. The first formal definition was given by Hill (1963). It requires that an RVE must be typical for the entire microstructure on average and has a sufficiently large number of microstructural heterogeneities. This is to ensure that the material characteristics are independent of the boundary conditions. In Drugan and Willis (1996), an RVE is defined as the smallest volume element of a material for which the material characteristics are sufficiently accurate to describe the behavior of the continuum. In I. Gitman, Askes, and Sluys (2007), further definitions are summarized in the following overview:

- the RVE should be sufficiently large compared to the microstructure Drugan and Willis (1996).
- the RVE should be small enough compared to the macro structure Zaoui (2001).

- the RVE should contain a sufficiently large amount of information on the microstructure Hashin (1983).
- the RVE must contain a large number of microstructural inhomogeneities Hill (1963).
- the RVE should react regardlessly of the type of boundary conditions Ostoja-Startzewski (1998).
- the statistically homogeneous properties ensure that the RVE is statistically representative of the macrostructure Ostoja-Startzewski (1998), Sab (1992).
- the size of the RVE is given, among other things, by the volume content of heterogeneities I. Gitman, M. Gitman, and Askes (2006), Segurado and Llorca (2002).

Since the calculation of the entire structure is too complex, the repetitive properties, e.g. the heterogeneities mentioned are mapped with an RVE. The RVE must have a sufficiently large number of heterogeneities to realistically reflect the structure. Because of this, the calculation on an RVE can be very complex. In addition, many RVEs have to be analyzed in order to achieve statistically representative results Fedelinski (2011)[p.333f]. With an RVE defined in this way, finite element (FE) calculations are often carried out for simulation. The results can be considered reasonably if the microstructure periodically consists of the structure depicted with the RVE.

From a numerical point of view for the FE calculation, however, it is cheaper to use a smaller section for the RVE to reduce the computing time. It is therefore always necessary to choose a sufficiently small RVE which is nevertheless representative of the overall structure. Since the influence of the microstructure of the material can be very large the RVE method is consistently used these days to significantly reduce the rigidity of the material at the macro level. The calculations of the RVE are part of the numerical homogenization of the micro-structural heterogeneities. In this context, homogenization is the process of calculating material behavior at the macro level based on the arrangement and properties at the micro-level.

For this connection, efficient methods are already available for linear elastic materials and geometrically linear theory, e.g. summarized and compared in Klusemann and Svendsen (2010). However, there is interest in finding efficient homogenization methods for nonlinear materials and large deformations. Simple possibilities are the use of the mean deformations (Taylor model, Taylor (1938)) or the mean stresses (Sachs model, Sachs (1928)) for the micro level. The advantage of this method is the low numerical effort, but fluctuations in the strains and stresses are not taken into account. There is a high level of interest in better approximations.

In addition to various combinations of the Taylor and Sachs models, a more practical method is to determine a material law at the macro level using numerical methods. RVEs are used to map the properties of the microstructure and to define appropriate boundary conditions according to the mean elongation or mean stress. The initial and boundary value problem is solved then using the finite element method (FEM). The advantage of this method is that there are no restrictions (e.g. on small deformations), as it is typically necessary with analytical and semi-analytical homogenization schemes. In this way, however, no material law is found in a closed form for the macro level. Instead, there is a relationship between stress and strain for a specific deformation path. The numerical complex FE ^{2} method is used to establish a connection to the macro level.

The most important tasks are therefore the optimization of the RVE with regard to networking, size, boundary conditions and shape in order to achieve a good compromise between numerical effort and accuracy.

3.2 Classification of the scales

The division of the scales requires that the inhomogeneities on the micro level (l_{micro}) are much smaller than the characteristic length on the macro level (l_{macro}, dimension of the body)

$$l_{\text{micro}} \ll l_{\text{macro}} \ . \tag{3.1}$$

If this is the case and thus area and volume integrals cover a large number of inhomogeneities, the exact position of these will be unimportant and only the statistical properties will be relevant. According to Hill (Hill (1956)), the smallest dimension of the RVE should be 10^3 times larger than the greatest inhomogeneity. Since the RVE describes a material point on the macro level, you can define the length of the RVE as l_{mini} and get Hashins micro-mini-macro principle (MMM, Hashin (1983)):

$$l_{\text{micro}} \ll l_{\text{mini}} \ll l_{\text{macro}} \tag{3.2}$$

Fig. 3.1 illustrates the three lengths mentioned on the macroscopic structure of a double T-beam and a cubic RVE with the edge length l_{mini}.

3.3 Micro-macro coupling

The following notation is used to differentiate between sizes in the RVE and on the macro level. First, the index “0” denotes sizes in the reference placement. The RVE area is marked with Ω. A dash marks sizes at the macro level which do not necessarily

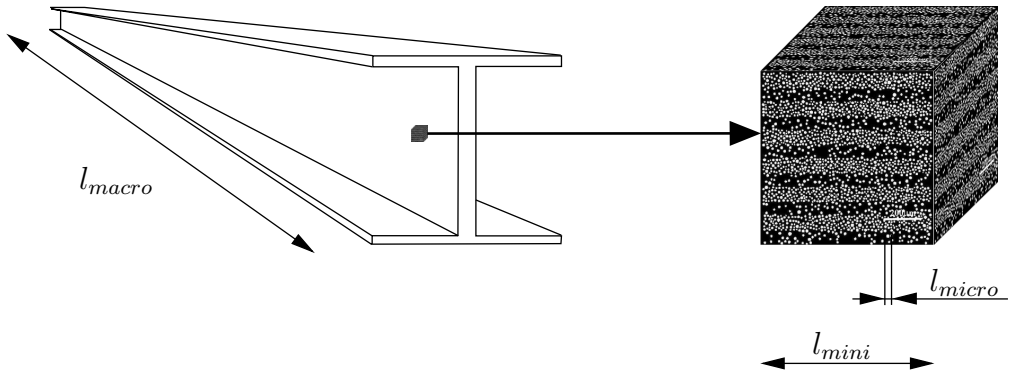


Figure 3.1: Illustration of the scale classification

match those at the micro level ($\overline{\boldsymbol{A}} \neq \langle \boldsymbol{A} \rangle$). The evaluation of all unweighted volume averages over the area Ω in the reference placement is e.g. noted as follows:

$$\langle \cdot \rangle := \frac{1}{V_0} \int_{\partial\Omega_0} \cdot dV_0 \ . \tag{3.3}$$

In principle, the choice of kinematic and dynamic quantities for coupling is arbitrary (see Nemat-Nasser (1999)), however, in order to avoid ambiguities, it must be chosen which measurement is used. This is usually the deformation gradient

$$\overline{\boldsymbol{F}} := \langle \boldsymbol{F} \rangle \, . \tag{3.4}$$

It has proven to be advantageous to use the aforementioned deformation gradient and the power-conjugated 1st Piola - Kirchhoff stress tensor $\overset{\text{1PK}}{\mathbf{T}}$ for the coupling. In the next sections, we will omit the 1PK for reasons of better readability.

$$\overline{\boldsymbol{T}} := \langle \boldsymbol{T} \rangle \tag{3.5}$$

Both $\overline{\boldsymbol{F}}$ and $\overline{\boldsymbol{T}}$ can be formulated as surface and volume integral. For the formulation of the stress tensor as a volume integral, the Gauss-Ostrogradski integral theorem and the static equilibrium condition $\boldsymbol{T} \cdot \nabla_0 = \boldsymbol{O}$ (siehe Section 3.4) are necessary.

$$\langle \boldsymbol{F} \rangle = \frac{1}{V_0} \int_{\partial\Omega_0} \boldsymbol{x} \otimes \boldsymbol{n}_0 dA_0 = \frac{1}{V_0} \int_{\Omega_0} \boldsymbol{x} \otimes \nabla_0 dV_0 \tag{3.6}$$

$$\langle \boldsymbol{T} \rangle = \frac{1}{V_0} \int_{\partial\Omega_0} \boldsymbol{t} \otimes \boldsymbol{x}_0 dA_0 = \frac{1}{V_0} \int_{\Omega_0} \boldsymbol{T} dV_0 \tag{3.7}$$

Due to the time independence of the reference placement, the material time deriva-

tives and the unweighted volume averages can be exchanged:

$$\overline{\dot{\boldsymbol{F}}} = \left\langle \dot{\boldsymbol{F}} \right\rangle = \langle \dot{\boldsymbol{F}} \rangle = \dot{\overline{\boldsymbol{F}}} \tag{3.8}$$

$$\overline{\dot{\boldsymbol{T}}} = \left\langle \dot{\boldsymbol{T}} \right\rangle = \langle \dot{\boldsymbol{T}} \rangle = \dot{\overline{\boldsymbol{T}}} \tag{3.9}$$

3.4 Global equilibrium

For the RVE, the momentum and angular momentum conservation on Ω must apply:

$$\int_{\Omega} \boldsymbol{\sigma} \cdot \nabla dV = \int_{\Omega} \rho \left(\ddot{\boldsymbol{x}} - \boldsymbol{b} \right) dV \tag{3.10}$$

$$\int_{\Omega} \left(\boldsymbol{\sigma} - \boldsymbol{\sigma}^{\mathrm{T}} \right) dV = \boldsymbol{O}. \tag{3.11}$$

Usually the neglect of the mass and inertial forces in the momentum balance law Eq. (3.10) is justified at this point. For this purpose the Gauss-Ostrogradski integral theorem and the theorem by Cauchy $\boldsymbol{t} = \boldsymbol{n} \cdot \boldsymbol{\sigma}$ are used to rewrite the balance law, so that a surface and a volume integral are created.

$$\int_{\partial\Omega} \boldsymbol{t} dA = \int_{\Omega} \rho \left(\ddot{\boldsymbol{x}} - \boldsymbol{b} \right) dV \tag{3.12}$$

Since l_{mini} of the RVE is $l_{\mathrm{mini}} \ll l_{\mathrm{macro}}$ and l_{mini} in the volume integral is order $O\left(l_{\mathrm{mini}}^3\right)$, it can be neglected in comparison to the surface integral with order $O\left(l_{\mathrm{mini}}^2\right)$ (see Miehe (2003)[p.562]). The goal of RVE calculations is to find a material law for the macro level based on the micro level. It is also necessary that this is independent of mass and inertial forces at the RVE. The equations for the global balance of momentum and angular momentum conservation are:

$$\int_{\partial\Omega} \boldsymbol{t} dA = \boldsymbol{O} \tag{3.13}$$

and

$$\int_{\partial\Omega} \left(\boldsymbol{t} \otimes \boldsymbol{x} - \boldsymbol{x} \otimes \boldsymbol{t} \right) dA = \boldsymbol{O} \ . \tag{3.14}$$

3.5 Hill-Mandel condition

The Hill-Mandel condition (HMC) demands the equality of work which is done on the micro and macro level. Hill's interpretation of the condition is: "... if a sufficiently large number of grains is contained and macroscopically homogeneously integrated over a large volume" (Hill, 1951). In order to contain a sufficiently large number of inhomogeneities and thus to appear as homogeneous as possible, the RVE must be chosen to be correspondingly large. For linear elastic materials with small deformations the condition reads

$$\langle \boldsymbol{\sigma} \cdot\cdot \boldsymbol{\varepsilon} \rangle = \langle \boldsymbol{\sigma} \rangle \cdot\cdot \langle \boldsymbol{\varepsilon} \rangle \tag{3.15}$$

and is fulfilled for homogeneous $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$. According to Suquet (1985), the HMC is fulfilled a priori for certain boundary conditions and with any RVE size. The boundary conditions should therefore always be selected so that they meet the HMC. To extend the HMC from Eq. (3.18) to large deformations, one begins with the stress power:

$$\frac{1}{\rho_0} \langle \boldsymbol{T} \cdot\cdot \dot{\boldsymbol{F}} \rangle = \frac{1}{\rho_0} \langle \boldsymbol{T} \rangle \cdot\cdot \langle \dot{\boldsymbol{F}} \rangle \ . \tag{3.16}$$

If, as mentioned in Section 3.3, the power conjugate variables $\boldsymbol{T}$ and $\dot{\boldsymbol{F}}$ are used, the HMC can be formulated in an easy way. The following decomposition is defined for the velocity field:

$$\dot{\boldsymbol{u}}\left(\boldsymbol{x}_0, t\right) := \dot{\bar{\boldsymbol{H}}}\left(t\right) \cdot \boldsymbol{x}_0 + \dot{\tilde{\boldsymbol{u}}}\left(\boldsymbol{x}_0, t\right) \tag{3.17}$$

With the relationship

$$\boldsymbol{F} = \boldsymbol{H} + \boldsymbol{I} = \boldsymbol{u} \otimes \nabla_0 + \boldsymbol{I} \tag{3.18}$$

and with Eq. (3.18) follows

$$\dot{\boldsymbol{F}}\left(\boldsymbol{x}_0, t\right) = \underbrace{\dot{\bar{\boldsymbol{H}}}\left(t\right) + \boldsymbol{I}}_{\dot{\bar{\boldsymbol{F}}}(t)} + \underbrace{\dot{\tilde{\boldsymbol{u}}}\left(\boldsymbol{x}_0, t\right) \otimes \nabla_0}_{\dot{\tilde{\boldsymbol{F}}}(\boldsymbol{x}_0, t)} \ . \tag{3.19}$$

For the still required time derivative of the deformation gradient $\dot{\boldsymbol{F}}$, the following applies to the micro and macro level

$$\dot{\bar{\boldsymbol{F}}}\left(t\right) := \langle \dot{\boldsymbol{F}}\left(\boldsymbol{x}_0, t\right) \rangle \ . \tag{3.20}$$

Next, we insert the decomposition of $\dot{\boldsymbol{F}}$ according to Eq. (3.19) and (3.20) into Eq. (3.16) returns after shortening from $1/\rho_0$

$$\langle \boldsymbol{T}\left(\boldsymbol{x}_0, t\right) \cdot\cdot \dot{\tilde{\boldsymbol{F}}}\left(t\right) \rangle + \langle \boldsymbol{T}\left(\boldsymbol{x}_0, t\right) \cdot\cdot \dot{\tilde{\boldsymbol{F}}}\left(\boldsymbol{x}_0, t\right) \rangle = \langle \boldsymbol{T}\left(\boldsymbol{x}_0, t\right) \rangle \cdot\cdot \dot{\bar{\boldsymbol{F}}}\left(t\right) \ . \tag{3.21}$$

In the first summand, $\dot{\boldsymbol{F}}(t)$ can be extracted from the volume averaging, since $\dot{\boldsymbol{F}}$ is independent of $\boldsymbol{x}_0$. We get the same term as on the right side of the equation which ultimately leads to the following expression:

$$\langle \boldsymbol{T}(\boldsymbol{x}_0, t) \cdot\cdot \dot{\tilde{\boldsymbol{F}}}(\boldsymbol{x}_0, t) \rangle = 0 \ . \tag{3.22}$$

In order to check certain boundary conditions for the fulfillment of the HMC, this can be expressed as a surface integral by force and displacement. First, the relationship $\dot{\tilde{\boldsymbol{F}}} = \dot{\tilde{\boldsymbol{u}}} \otimes \nabla_0$ is used and inserted in Eq. (3.22). The term $\boldsymbol{T} \cdot\cdot \left(\dot{\tilde{\boldsymbol{u}}} \otimes \nabla_0\right)$ can be seen as one summand of a product rule $\boldsymbol{T} \cdot\cdot \left(\dot{\tilde{\boldsymbol{u}}} \otimes \nabla_0\right) + (\boldsymbol{T} \cdot \nabla_0) \cdot \dot{\tilde{\boldsymbol{u}}} = \left(\dot{\tilde{\boldsymbol{u}}} \cdot \boldsymbol{T}\right) \cdot \nabla_0$ and the following equation results:

$$\langle (\boldsymbol{T} \cdot \nabla_0) \cdot \dot{\tilde{\boldsymbol{u}}} - \left(\dot{\tilde{\boldsymbol{u}}} \cdot \boldsymbol{T}\right) \cdot \nabla_0 \rangle = 0 \ . \tag{3.23}$$

Due to the neglect of the mass and inertial forces, as explained in Section 3.4, $\boldsymbol{T} \cdot \nabla_0 = \boldsymbol{O}$ and the remaining summand with the Gauss integral theorem becomes the sought expression of the HMC:

$$0 = \frac{1}{V_0} \int_{\delta\Omega} \dot{\tilde{\boldsymbol{u}}} \cdot \boldsymbol{T} \cdot \boldsymbol{n}_0 dA = \frac{1}{V_0} \int_{\delta\Omega} \dot{\tilde{\boldsymbol{u}}} \cdot \boldsymbol{t} dA \ . \tag{3.24}$$

3.6 Boundary condition for the RVE

This chapter is intended to give an overview of the boundary conditions used in the literature for an RVE and to make a selection for the boundary conditions used in this work.

The boundary value problem with static equilibrium on simple materials is limited to the specification of the displacement $\boldsymbol{u}_i$ and the stress $\boldsymbol{t}_i$ at each surface point i. The boundary conditions can thus be divided into the two groups of displacement and stress boundary conditions. The displacement boundary conditions are much more popular than the stress boundary conditions for several reasons. On the one hand, most material laws are formulated in such a way that the displacement is the independent variable and the stress is the dependent variable ($\boldsymbol{T}(\boldsymbol{F}(\boldsymbol{u}))$). On the other hand, in the majority of commercial numerical programs, the displacement is introduced as an independent variable.

There are various ways of defining the boundary conditions on an RVE. Since there are no natural boundary conditions for a particular RVE, the choice is largely arbitrary. An exception is given in Miehe (2003) to periodic microstructures, where periodic boundary conditions are appropriate. Table 3.2 gives an overview of common boundary conditions. The linear displacement boundary conditions (LDBC) and dynamic minimal boundary conditions (DMBC) as well as the homogeneous

Boundary conditions	$\bar{\mathbf{H}}$	$\bar{\mathbf{T}}$
Linear displacement LDBC	$\mathbf{u} = \bar{\mathbf{H}} \cdot \mathbf{x}_0$ $\bar{\mathbf{H}}$ given	not possible
Dynamic minimal DMBC	$\mathbf{u} = \bar{\mathbf{H}} \cdot \mathbf{x}_0$ $\bar{\boldsymbol{H}}$ degree of freedom	$\bar{\mathbf{T}} = \frac{1}{V_0} \int_{\partial\Omega_0} \mathbf{t} \otimes \mathbf{x}_0 dA$ $\bar{\mathbf{T}}$ given
Periodic displacement Antiperiodic stresses PBC	$\mathbf{u}^+ - \mathbf{u}^- =$ $\bar{\mathbf{H}} \cdot \left(\mathbf{x}_0^+ - \mathbf{x}_0^-\right)$ $\bar{\mathbf{H}}$ given	$\mathbf{t}^+ + \mathbf{t}^- = \mathbf{O}$
Homogeneous stresses HSBC	not possible	$\mathbf{t} = \bar{\mathbf{T}} \cdot \mathbf{n}_0$ $\bar{\mathbf{T}}$ given
Kinematic minimal KMBC	$\bar{\mathbf{H}} = \frac{1}{V_0} \int_{\partial\Omega_0} \mathbf{u} \otimes \mathbf{n}_0 dA$ $\bar{\mathbf{H}}$ given	$\mathbf{t} = \bar{\mathbf{T}} \cdot \mathbf{n}_0$ $\bar{\mathbf{T}}$ degree of freedom

Figure 3.2: Usual boundary conditions for an RVE

stress boundary conditions (HSBC) and kinematic minimal boundary conditions (KMBC) are each equivalent and differ only in the choice of degrees of freedom. In this work, the LDBC and the PBC are implemented.

3.7 Python script for generating RVEs in Abaqus

In this section, the scripting language Python in general and in connection with Abaqus is discussed and with the help of this a script for the automated generation of three different RVEs is created. Familiarization with Python takes place with the very helpful book "Python For Software Design - How To Think Like A Computer Scientist" by Downey (2009) as well as the detailed online documentation on *python.org*. The execution of the script was tested and carried out both under the

Linux distribution Ubuntu 16.04 LTS and under Windows 7. Abaqus version 6.10-2 was used on Ubuntu and version 6.8-1 on Windows 7. The script recognizes the operating system wherever there are differences in the operating system.

To work with Abaqus, the Abaqus documentation in version 6.10 (see Simulia (2010)) was used as operating instructions and reference work. This documentation contains 20 individual documents, of which e.g. the Abaqus Keywords Reference Manual, the Abaqus User Subroutines Reference Manual and the Abaqus / CAE User's Manual were used. The Abaqus Scripting User's Manual and the Abaqus Scripting Reference Manual were used to work with Python in connection with Abaqus.

Abaqus has a scripting interface (see Simulia (2010)), which is a so-called application programming interface (API) to access the models and data used by Abaqus. This interface is an extension of the object-oriented programming language Python and can be used for the following tasks:

- Create and edit parts of an Abaqus model
- Create, edit and start Abaqus jobs
- Read from and write to the Abaqus output database (ODB)

A programmed script is saved with the file extension for Python "* .py". This is a series of instructions that are saved in ASCII format. These can be started from Abaqus via "File → Run Script" or directly in a shell with "abaqus cae script = myscript.py" or without a graphical user interface with "abaqus cae noGUI = myscript.py".

The main part of all calculations is the generation of the RVE itself in the Python script "0_RVE.py". In this script, different choices regarding the material type

- U = uni-directional reinforcement with transversal isotropic symmetry
- C = bi-directional reinforcement with cubic symmetry
- T = tri-directional reinforcement with tetragonal symmetry,

the boundary conditions on the RVE

- PBC = periodic boundary conditions
- LDBC = linear displacement boundary conditions,

the element size and the element shape are given (see Appendix A.1).

```
print '=============================================================='
print 'Start RVE routine'
print '=============================================================='
t0abs = time.time();
#CPU
cpunumber = '1'
#Material type
typ    = 'T'
#Boundary conditions
rb     = 'PBC'
#Elementkantenlaenge
ele = 1.25
#Element shape  HEX, WEDGE, TET
eleshape = 'HEX'
#Element type Hex:    lin: C3D8 quad: C3D20
#             Wedge:  lin: C3D6 quad: C3D15
#             Tet:    lin: C3D4 quad: C3D10
eletyp = 'C3D20'
if typ == 'T':
...
if typ == 'C':
...
if typ == 'U':
...
#Read material parameters
mat = open('0_material.py','r')
for line in mat:
    #read = str(mat.readline())
    f.write('%s' %line)
mat.close()
print '=============================================================='
print 'End RVE routine'
print '=============================================================='
```

This procedure is the same for all types of materials. The structure of this module can first be divided into two fundamentally different sections. In the first section, Python directly accesses Abaqus functions. After creating the part and the assembly, the information available up to that point are written into the Abaqus input file. This first section differs in dependency of the chosen material. It will be described in the Sects. 3.8.1, 3.8.2 and 3.8.3. The second section is completely independent of Abaqus and includes the processing and completion of the input file using Python.

3.7.1 Read node information

The input file written by Abaqus in the first section will be opened by Python again. It contains all nodes and elements that have arisen from the meshing. It is necessary to prepare this information for further processing. After opening the input file in

the mode *r+* (read and write), a list of the nodes is generated first. The Python list object is used for this. A list *data* with the method *append* is attached to each entry in the list *nodes* which contains the node number and the coordinates x, y and z. While reading the file, the script searches for different keywords. The first one is "*Nod". After this keyword, the nodes list starts. The next keyword is "*Elements" which marks the end of the nodes list. All nodes of the generated RVE mesh are saved in the Python list "nodes".

```
##############################################################################
# Open input file
##############################################################################
    f = open(inpname+".inp",'r+')
    read = str(f.readline())
    while read[0:5] != '*Node':
        read = str(f.readline())
##############################################################################
# Generate list of nodes
##############################################################################
    print 'Read nodes'
    nodes = []
    data = []
    read = str(f.readline())
    while read[:8] != '*Element':
        data = read.split(",")
        data = map(float, data)
        data[0] = int(data[0])
        nodes.append(data)
        read = str(f.readline())
    print len(nodes), 'Nodes'
```

3.7.2 Read element information

The same procedure is performed to read the list of all elements. They are written in the input file between the keywords "*Elements" and "*Nset". The list of all elements is generated containing the element number and the associated nodes for linear elements (C3D8) with 8 nodes and quadratic hexahedron elements (C3D20) with 20 nodes. This means we need a case distinction while reading the element definitions by using an IF statement.

```
##############################################################################
# Generate list of elements
##############################################################################
    # read elements until element definition is reached
    # for C3D20 element definition need two rows: if necessary
    print 'Elemente einlesen!'
    elements = []
    data = []
```

```
    # HEX
    if str(eletyp)[0:4]=='C3D8':
      print 'HEX C3D8'
        read = str(f.readline())
    elif str(eletyp)[0:5]=='C3D20':
      read = str(f.readline()) + str(f.readline())
      print 'HEX C3D20'
    # WEDGE
    elif str(eletyp)[0:4]=='C3D6':
      read = str(f.readline())
      print 'WEDGE C3D6'
    elif str(eletyp)[0:5]=='C3D15':
      read = str(f.readline())
      print 'WEDGE C3D15'
    else:
      print 'Elementtyp nicht vorgesehen! Code ueberpruefen'
    while read[:5] != '*Nset':
      data = read.split(",")
      elements.append(map(int, data))
      if eval(eletyp)==C3D8:
        read = str(f.readline())
      if eval(eletyp)==C3D20:
        read = str(f.readline()) + str(f.readline())
      if eval(eletyp)==C3D6:
        read = str(f.readline())
      if eval(eletyp)==C3D15:
        read = str(f.readline())
    print len(elements), 'Elements'
```

In the following sections, we will discuss the generation of the different RVEs for the different materials.

3.7.3 Sort the nodes for the boundary conditions

The next step is to sort the list of nodes for the later implementation of the boundary conditions. We have to find all surface nodes and will sort them for corner, edge and surface nodes. For this purpose, we additionally distinguish between the alignment of edges and surfaces in x, y and z direction and their positions relative to the origin (which is in the center of mass for all types of RVE). All remaining internal nodes are saved in an additional list for control purposes. This sorting results in a total number of 20 lists.

```
    for i in range(len(nodes)):
      #Ecke1
      if round(nodes[i][1],stellen) == round(XNUC*drve/2.,stellen) and
          \
        round(nodes[i][2],stellen) == round(YNUC*drve/2.,stellen) and \
```

```
        round(nodes[i][3],stellen) == round(ZNUC*drve/2.,stellen):
          ecken.append(nodes[i][0])
      #Ecke2
[...]
      #Kanten X+Y-Z
      elif round(nodes[i][1],stellen) != round(+drve/2.,stellen) and \
        round(nodes[i][1],stellen) != round(-drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(+drve/2.,stellen) and \
        round(nodes[i][3],stellen) == round(-drve/2.,stellen):
        kantenXpYmZ.append(nodes[i][0])
      #Kanten X-Y-Z
      elif round(nodes[i][1],stellen) != round(+drve/2.,stellen) and \
        round(nodes[i][1],stellen) != round(-drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(-drve/2.,stellen) and \
        round(nodes[i][3],stellen) == round(-drve/2.,stellen):
        kantenXmYmZ.append(nodes[i][0])
[...]
      #Flaechen Xp
      elif round(nodes[i][1],stellen) == round(+drve/2.,stellen):
        flaechenXp.append(nodes[i][0])
      #Flaechen Xm
[...]
      #Innen
      else:
        innen.append(nodes[i][0])
```

This is enough for the LDBC but for the PBC we need an additional sorting to find the opposing nodes couples. The pairs of opposite nodes must be in the lists one after the other. For the corner nodes, all three components of the location vector must have the same components, but be directed in opposite directions. For nodes on the edges, this condition must apply to two components. The third component must have exactly the same value. Similarly, for nodes on the surfaces, one component is directed in the opposite direction and the other two are exactly the same.

```
    ##########################################################################
    # Additional sorting for PBC
    ##########################################################################
    if rb == 'PBC':
      print 'Additional sorting for PBC ...'
      # Sorting opposite nodes within the list
      # Sorting corners
      eckensort = []

      for i in range(len(ecken)):
        #Ecke1
        if round(nodes[ecken[i]-1][1],stellen) == round(drve/
                2.,stellen) and \
          round(nodes[ecken[i]-1][2],stellen) == round(drve/
          2.,stellen) and \
          round(nodes[ecken[i]-1][3],stellen) == round(drve/
```

```
            2.,stellen):
            eckensort.append(ecken[i])
            break
[...]
```

In the third and last step, the lists sorted in this way are combined in a new list and the node list *nodes* is sorted accordingly.

3.7.4 Implement boundary conditions using constraint equations

To implement the boundary conditions, it is necessary to insert three artificial nodes. The displacement boundary conditions, which are predetermined by the mean displacement gradient $\bar{\boldsymbol{H}}$, are later applied to these three nodes. This step is necessary because the boundary conditions are implemented, as described later, with the help of *linear constraint equations* (see Simulia (2010)). The artificial nodes are all created in the coordinate origin and are named $K1$, $K2$ and $K3$. An element set is created that contains all elements and this set is assigned a *Solid Section* and the corresponding material. Next, the *linear constraint equations* are introduced and the type of boundary conditions is chosen. According to the introduced abbreviations LDBC and PBC, it is decided how these boundary conditions are defined.

```
##############################################################################
# Write input file
##############################################################################
shutil.copyfile(inpname+".inp", inpname+"_backup.inp")
f.seek(0,0)
while read[:5] != '*Node':
    read = str(f.readline())
f.truncate()
f.write('%d,0.0,0.0,0.0 \n' %(len(nodes)+1))
f.write('%d,0.0,0.0,0.0 \n' %(len(nodes)+2))
f.write('%d,0.0,0.0,0.0 \n' %(len(nodes)+3))
b = open(inpname+"_backup.inp",'r+')
lines1 = b.readlines()
b.close()
os.remove(inpname+"_backup.inp")
for i in range(17,len(lines1)):
    f.write(lines1[i])
f.seek(0,0)
# Append additional nodes
print 'Inputdatei verfollstaendigen ...'
while read[:11] != '** Section:':
    read = str(f.readline())
f.truncate()
if "posix" in os.name:
```

```
    f.seek(-17,2)
  .
  .
  .
# Define equation
f.write('**\n')
f.write('** Constraint: Constraint-1 \n')
f.write('**\n')
f.write('*Equation \n')
```

3.7.5 Implementation of the LDBC

The linear displacement boundary conditions (LDBC) are given by the average displacement gradient with

$$\boldsymbol{u} = \bar{\boldsymbol{H}}\boldsymbol{x}_0 \ . \tag{3.25}$$

To check the Hill-Mandel condition, the decomposition of the velocity field according to Eq. (3.17) is used. With Eq. (3.25) follows $\dot{\boldsymbol{u}}(\boldsymbol{x}, t) = 0$ and the Hill-Mandel condition is fulfilled.

The discretized version of Eq. (3.25) is used to implement the LDBC:

$$\begin{aligned} u_j^i &= \sum_{k=1}^{3} \bar{H}_{jk} x_k^i \\ &= \bar{H}_{j1} x_1^i + \bar{H}_{j2} x_2^i + \bar{H}_{j3} x_3^i \\ &= u_1^{Kj} x_1^i + u_2^{Kj} x_2^i + u_3^{Kj} x_3^i \ . \end{aligned} \tag{3.26}$$

The indexing was chosen as follows. The node number is identified by the index "i" and "j" defines the degree of freedom of the node. The three additional nodes are labeled $K1$, $K2$ and $K3$. Their displacements are given in the arrangement of the average displacement gradient $\bar{\boldsymbol{H}}$:

$$\boldsymbol{H} = \begin{bmatrix} u_1^{K1} & u_2^{K1} & u_3^{K1} \\ u_1^{K2} & u_2^{K2} & u_3^{K2} \\ u_1^{K3} & u_2^{K3} & u_3^{K3} \end{bmatrix} \ . \tag{3.27}$$

By doing this, the LDBC can be implemented by applying displacement constraints to the three artificial nodes. For this, it is necessary to formulate a *linear constraint equation* for every degree of freedom of each surface node (see Simulia (2010)). An *equation* basically consists of the first line which defines the number of terms. In the case of the LDBC, these are always four terms according to Eq. (3.26).

$$0 = u_j^i - \bar{H}_{j1} x_1^i - \bar{H}_{j2} x_2^i - \bar{H}_{j3} x_3^i \tag{3.28}$$

Each subsequent line represents a term of the *equation* and has the following structure: “Node number, degree of freedom, coefficient”. If you wanted to implement the equation

$$0 = u_2^{100} - \bar{H}_{21} x_1^{100} - \bar{H}_{22} x_2^{100} - \bar{H}_{23} x_3^{100} \tag{3.29}$$

for the node with number 100 and degree of freedom 2 (y-direction, second artificial node $K2$), the *equation* would look like this:

```
4
100,2,1
K2,1,-x^100_1
K2,2,-x^100_2
K2,3,-x^100_3
```

An “if query” checks the selected boundary condition and realizes it in the case of the LDBC as follows:

```
if RB == 'LV':
##############################################################################
#Lineare Verschiebungs-RB (LVRB)
##############################################################################
    print 'Lineare Verschiebungs-RB (LVRB) gewaehlt!'
    #DOF1
    for i in range(0,letzteoberflaeche+1):
        f.write("4\n")
        f.write("%d,1,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
            %(nodes[i][0],len(nodes)+1,-nodes[i][1],len(nodes)+1, \
            -nodes[i][2],len(nodes)+1,-nodes[i][3]))
    #DOF2
    for i in range(0,letzteoberflaeche+1):
        f.write("4\n")
        f.write("%d,2,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
            %(nodes[i][0],len(nodes)+2,-nodes[i][1],len(nodes)+2, \
            -nodes[i][2],len(nodes)+2,-nodes[i][3]))
                                .
                                .
                                .
```

3.7.6 Implementation of the PBC

The periodic displacement boundary conditions (PBC) combine the LDBC and the HSBC in such a way that they evenly split the BCs into $\boldsymbol{t}$ and $\boldsymbol{u}$. It is necessary that each point has an opposite point to which the following equation holds

$$\boldsymbol{n}_0^+ = -\boldsymbol{n}_0^- \ . \tag{3.30}$$

The formulation of the PBC in dependency of $\boldsymbol{H}$ reads

$$\boldsymbol{u}^+ - \boldsymbol{u}^- = \bar{\boldsymbol{H}} \cdot \left(\boldsymbol{x}_0^+ - \boldsymbol{x}_0^-\right) \tag{3.31}$$

and in dependency of $\boldsymbol{T}$

$$\boldsymbol{t}^+ + \boldsymbol{t}^- = \boldsymbol{O} \ . \tag{3.32}$$

The Hill-Mandel condition is automatically fulfilled because opposite points in the surface integral in the Eq. (3.24) cancel out each other. The same applies to the surface forces and thus the conservation of momentum is also fulfilled. To check the conservation of angular momentum, the surface integral in Eq. (3.11) is split into $\partial\Omega^+$ and $\partial\Omega^-$. Additionally, we insert $\boldsymbol{t}^- = -\boldsymbol{t}^+$ and $\boldsymbol{x}^- = \boldsymbol{x}^+ - \bar{\boldsymbol{F}}\boldsymbol{x}_0^+ + \bar{\boldsymbol{F}}\boldsymbol{x}_0^-$:

$$\int_{\partial\Omega_0^+} \boldsymbol{t}^+ \otimes \left(\boldsymbol{x}^+ - \bar{\boldsymbol{F}}\boldsymbol{x}_0\right) dA_0 - \int_{\partial\Omega_0^-} \boldsymbol{t}^+ \otimes \left(\boldsymbol{x}^+ - \bar{\boldsymbol{F}}\boldsymbol{x}_0\right) dA_0 = \boldsymbol{O} \tag{3.33}$$

It can be seen that, due to the periodicity, the two surface integrals cancel out each other and the conservation of angular momentum is fulfilled. The discretized form of Eq. (3.31) is initially:

$$\begin{aligned} u_j^{i+} - u_j^{i-} &= \sum_{k=1}^{3} \bar{H}_{jk} \left(x_k^{i+} - x_k^{i-}\right) \\ &= \bar{H}_{j1} \left(x_1^{i+} - x_1^{i-}\right) + \bar{H}_{j2} \left(x_2^{i+} - x_2^{i-}\right) + \bar{H}_{j3} \left(x_3^{i+} - x_3^{i-}\right) \ . \end{aligned} \tag{3.34}$$

As with the LDBC, the equation is implemented in a homogeneous form:

$$0 = u_j^{i+} - u_j^{i-} - \bar{H}_{j1} \left(x_1^{i+} - x_1^{i-}\right) - \bar{H}_{j2} \left(x_2^{i+} - x_2^{i-}\right) - \bar{H}_{j3} \left(x_3^{i+} - x_3^{i-}\right) \ . \tag{3.35}$$

The node pairs are already sorted as described in Subsection 3.7.3. The loop for generating the *equation* thus again runs over all surface nodes, but with a step size of 2.

```
elif RB == 'PBC':
##############################################################################
# Periodic boundary conditions (PBC)
##############################################################################
    print 'Periodic boundary conditions (PBC)!'
    #DOF1
    for i in range(0,letzteoberflaeche+1,2):
      f.write("5\n")
      f.write("%d,1,1.0\n%d,1,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
        %(nodes[i][0], \
        nodes[i+1][0], \
        lennodes+1,(nodes[i+1][1]-nodes[i][1]), \
        lennodes+1,(nodes[i+1][2]-nodes[i][2]), \
        lennodes+1,(nodes[i+1][3]-nodes[i][3])))
[...]
```

3.8 Examples used in this work

In the following subsections, we present the representative volume elements (RVEs) used in this work. These are purely academic examples with the aim of analyzing the development of the stiffness tetrads. Of course, real materials look different. All of the following simulations are displacement controlled by prescribing the components of the effective displacement gradient $\mathbf{H}$ and using periodic boundary conditions. In all simulations, we used the 20 nodes quadratic elements C3D20. We also ensured that no large effective volume dilatations occur during the FE simulations by

- prescribing only isochoric effective final deformations,
- allowing for free lateral straining such that no volumetric straining is enforced,
- prescribe a strain path without intermediate large volumetric strains that may occur when ramping up the effective displacement gradient linearly.

3.8.1 Uni-directional reinforcement

The first material is supposed to have an effectively transversal isotropic symmetry. Therefore we choose a uni-directional reinforced material with circular fibers in an hexagonal arrangement. In Fig. 3.3, the unit cell is shown. In the "0_RVE.py" script, this material is chosen with the parameter "type=U". To create the meshed RVE with Abaqus, various individual steps are necessary. First a new ModelDataBase and two homogeneous solid sections for the matrix material and the fiber material are defined. Afterwards the parts depicted in Fig. 3.3 are defined in dependence of the fiber radius and thus in dependence of the fiber volume fraction. Afterwards the section is assigned. The next step is to build the assembly out of the 12 parts and to pattern them when more than one unit cell is defined (Abaqus function InstanceFromBooleanMerge is needed). The last step before the Abaqus input file is written is to mesh the assembly. Here, the parameters "element size", "element shape" and "element type" are used to define the mesh. The following Python commands are performed:

```
# Mesh
    a1.makeIndependent(instances=(a1.instances['FIBBI-1'], ))
    partInstances =(a.instances['FIBBI-1'], )
    a1.seedPartInstance(regions=partInstances, size=ele,
        deviationFactor=0.1)
    a = mdb.models['Model-1'].rootAssembly
    c1 = a.instances['FIBBI-1'].cells
    cel = c1.pointsOn
    pickedRegions = c1.findAt(cel[0],)
    for i in range(1,len(cel)):
        pickedRegions += c1.findAt(cel[i],)
```

```
a.setMeshControls(regions=pickedRegions, elemShape=eval(eleshape),
    technique=SWEEP)
elemType1 = mesh.ElemType(eval(eletyp))
pickedRegions = (pickedRegions,)
a.setElementType(regions=(pickedRegions), elemTypes=(elemType1, ))
partInstances =(a.instances['FIBBI-1'], )
a.generateMesh(regions=partInstances)
```

In Fig. 3.4 the meshed RVE is depicted in an undeformed and deformed situation.

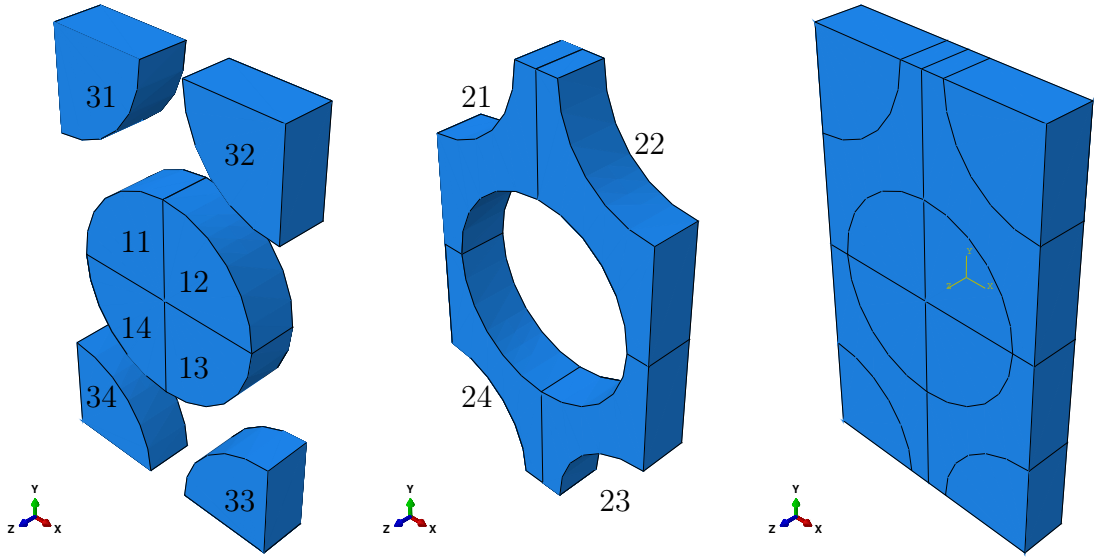


Figure 3.3: RVE with uni-directional reinforcement. Left: Fiber parts, Middle: Matrix parts, Right: Assembly

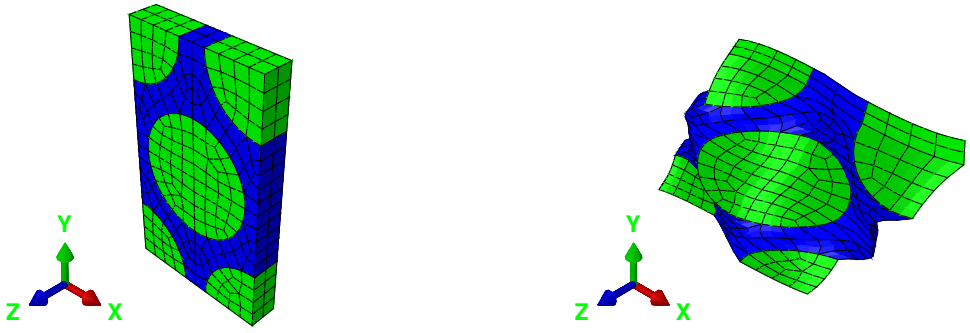


Figure 3.4: One unit-cell of the uni-directional RVE with effectively transversal isotropic symmetry. Left: undeformed, Right: deformed unit-cell

3.8.2 Bi-directional reinforcement

The second material will have a tetragonal symmetry. For the sake of simplicity, the RVE for this material will have the shape of a cube. The cube will be bi-directional reinforced with cubic fibers in the direction of x and y. In Fig. 3.5, the unit cell is shown. In the "0_RVE.py" script, this material is chosen with the parameter "type=T". The generation of the meshed RVE is similar to the procedure of the

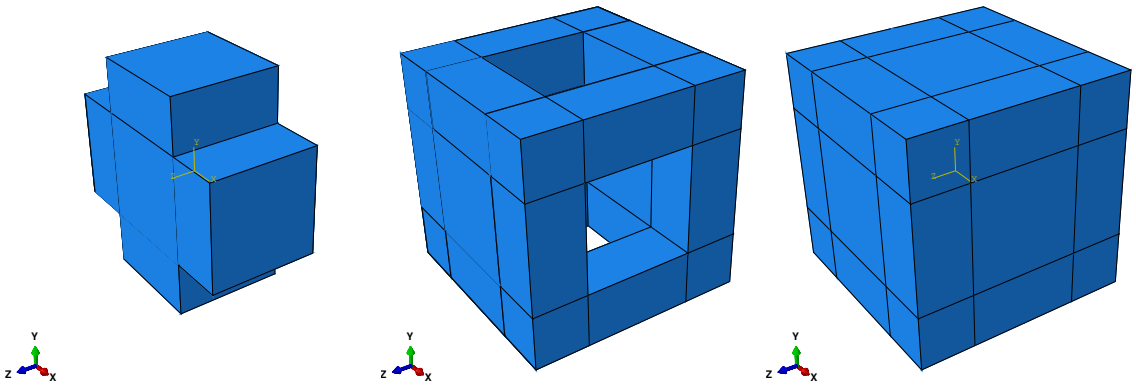


Figure 3.5: RVE with bi-directional reinforcement. Left: Fiber parts, Middle: Matrix parts, Right: Assembly

uni-directionally reinforced RVE. For details, the source code can be found in the Appendix A.1. In Fig. 3.6 the meshed RVE is depicted in an undeformed and deformed situation.

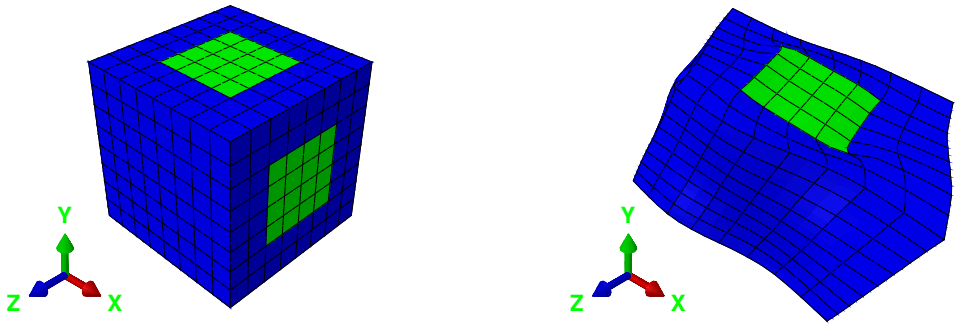


Figure 3.6: One unit-cell of the bi-directional RVE with effectively tetragonal symmetry. Left: undeformed, Right: deformed unit-cell

3.8.3 Tri-directional reinforcement

The third material will be tri-directionally reinforced which results in a cubic symmetry. The RVE is nearly the same like for the bi-directional reinforcement but now there are fibers in all three direction x, y and z. In Fig. 3.7, the unit cell is shown. In the "0_RVE.py" script, this material is chosen with the parameter "type=C". To

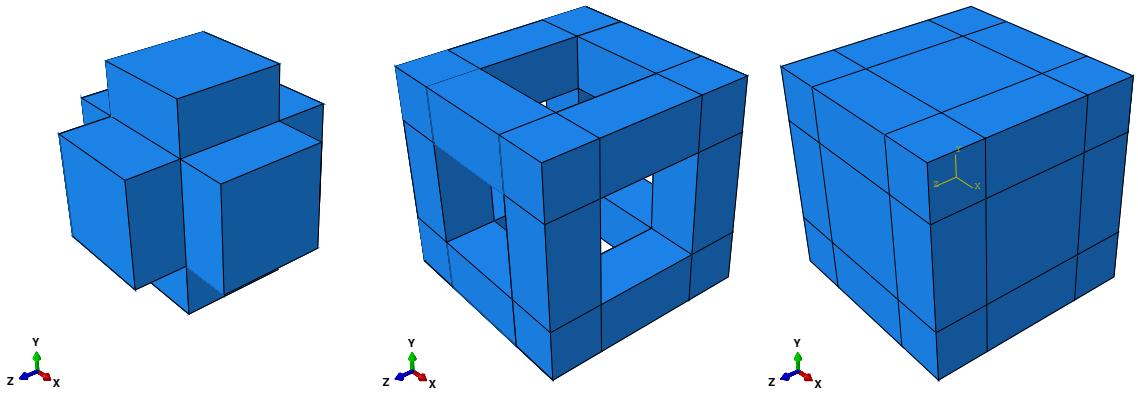


Figure 3.7: RVE with tri-directional reinforcement. Left: Fiber parts, Middle: Matrix parts, Right: Assembly

find details of the generation and meshing of the RVE again, the source code can be found in the Appendix A.1. In Fig. 3.8, the meshed RVE is depicted in an undeformed and deformed situation.

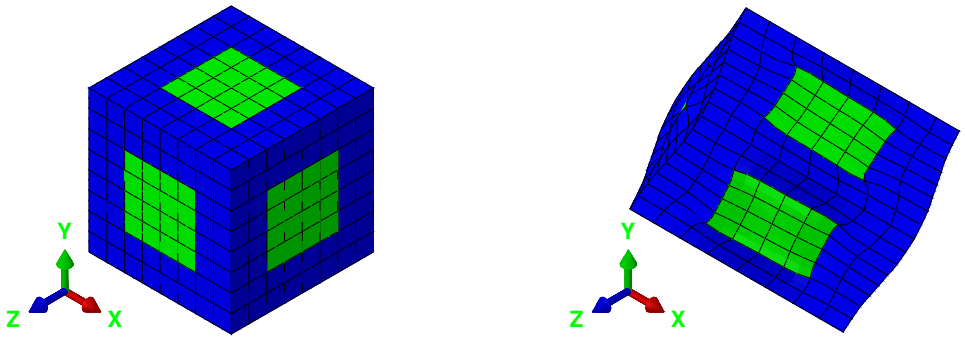


Figure 3.8: One unit-cell of the tri-directional RVE with effectively cubic symmetry. Left: undeformed, Right: deformed unit-cell

3.8.4 Obsolet materials

During this work, two other materials were designed but not used for further studies. The first material is a one-directional reinforced material with the shape of a cube and one cubic fiber. In principle, it is the same RVE like the bi- and tri-directionally reinforced RVE but with only one fiber in x-direction. In the "0_RVE.py" script, this material is chosen with the parameter"type=T1F"

The second one is a bi-directional reinforced materiel with circular fibers which has a parameterized fiber angel. In the "0_RVE.py" script, this material is chosen with the parameter "type=B". In Fig. 3.9, the meshed RVE is depicted in an undeformed and deformed situation.

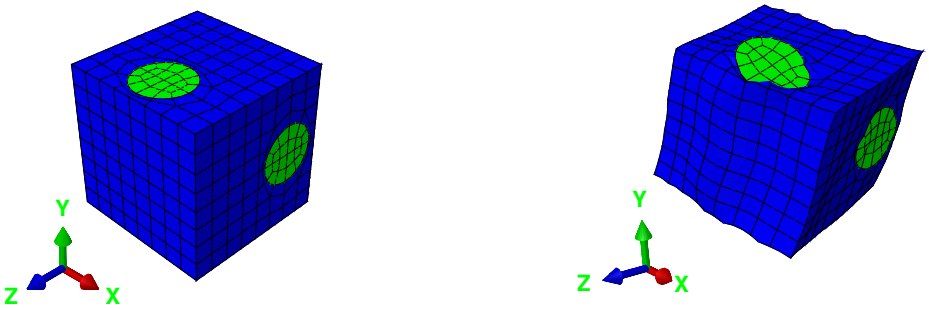


Figure 3.9: One unit-cell of the old bi-directional RVE. Left: undeformed, Right: deformed unit-cell

3.9 Summary

In this chapter, we introduced RVEs for modeling a uni-directional, bi-directional and tri-directional material. They all have different material symmetries namely transversal isotropic, tetragonal and cubic. We used the finite element code Abaqus in combination with the scripting language Python. The Python scripts allow the automated generation of the RVEs with different parameters.

4 Material model on the micro-scale

In this chapter, we develop a finite elasto-plasticity theory for large plastic deformations on the micro-scale. We use this material model for both material phases (matrix and fiber) of the RVE. For the elastic part of the model, we use the St. Venant-Kirchhoff elasticity. The plastic part is described by the isomorphy concept.

4.1 St. Venant-Kirchhoff elasticity

For the elastic deformation of the material, we use the physically linear elastic St. Venant-Kirchhoff reference law. It is constant and denoted with the 2nd Piola-Kirchhoff stress tensor:

$$\overset{\text{2PK}}{\boldsymbol{T}} = J\boldsymbol{F}^{-1}\boldsymbol{T}\boldsymbol{F}^{-T} \tag{4.1}$$

with J being the determinant of the deformation gradient $\boldsymbol{F}$, and $\boldsymbol{T}$ the Cauchy stress tensor.

$$\overset{\text{2PK}}{\boldsymbol{T}} = k_0\left(\boldsymbol{F}\right) = \mathbb{K}_0 \cdot\cdot \overset{\text{G}}{\boldsymbol{E}} \,. \tag{4.2}$$

With the Green strain tensor $\overset{\text{G}}{\boldsymbol{E}} = \frac{1}{2}\left(\boldsymbol{C} - \boldsymbol{I}\right)$ and the right Cauchy-Green tensor $\boldsymbol{C} = \boldsymbol{F}^T\boldsymbol{F}$ we can write this elastic law as

$$\overset{\text{2PK}}{\boldsymbol{T}} = k_0\left(\boldsymbol{F}\right) = \frac{1}{2}\mathbb{K}_0 \cdot\cdot \boldsymbol{F}^{\mathrm{T}}\boldsymbol{F} - \boldsymbol{I} \,. \tag{4.3}$$

4.2 Isomorphy concept for large deformations

It is a well known fact that the elastic behavior of a material hardly changes during large plastic deformations. Therefore we want to use a plasticity theory with isomorphic elastic ranges. To transform between these elastic ranges, we use a second-order

tensor $\boldsymbol{P}$. With this plastic transformation $\boldsymbol{P}$, we will be able to calculate the current elastic law k_p for the reference placement (see Bertram (2012) p.271) by the following equation:

$$\overset{2PK}{\boldsymbol{T}} = k_\mathrm{p}\left(\boldsymbol{F}\right) = \boldsymbol{P} k_0 \left(\boldsymbol{P}^\mathrm{T}\boldsymbol{F}^\mathrm{T}\boldsymbol{F}\boldsymbol{P}\right)\boldsymbol{P}^\mathrm{T} \ . \tag{4.4}$$

If we now insert our chosen elastic reference law k_0 Eq. (4.3) in Eq. (4.4), we will be able to calculate the 2$^{\mathrm{nd}}$ Piola-Kirchhoff stress tensor in the reference placement

$$\overset{2PK}{\boldsymbol{T}} = k_\mathrm{p}\left(\boldsymbol{F}\right) = \boldsymbol{P}\left(\frac{1}{2}\mathbb{K}_0 : \left(\boldsymbol{P}^\mathrm{T}\boldsymbol{F}^\mathrm{T}\boldsymbol{F}\boldsymbol{P} - \boldsymbol{I}\right)\right)\boldsymbol{P}^\mathrm{T} \ . \tag{4.5}$$

which can also be written with the Rayleigh product

$$\overset{2PK}{\boldsymbol{T}} = k_\mathrm{p}\left(\boldsymbol{F}\right) = \frac{1}{2}\left(\boldsymbol{P} * \mathbb{K}_0\right) : \left(\boldsymbol{F}^\mathrm{T}\boldsymbol{F} - \boldsymbol{P}^{-T} * \boldsymbol{I}\right) \ . \tag{4.6}$$

Now we continue with Eq. (4.5) and use the relation

$$\widetilde{\boldsymbol{F}} = \boldsymbol{F}\boldsymbol{P} \tag{4.7}$$

resulting from the isomorphy condition and notate the stresses in all three placements.

- reference placement:

$$\overset{2PK}{\boldsymbol{T}} = \boldsymbol{P}\left(\frac{1}{2}\mathbb{K}_0[\widetilde{\boldsymbol{F}}^\mathrm{T}\widetilde{\boldsymbol{F}} - \boldsymbol{I}]\right)\boldsymbol{P}^\mathrm{T} \tag{4.8}$$

- intermediate placement:

$$\overset{\widetilde{2PK}}{\boldsymbol{T}} = \frac{1}{2}\mathbb{K}_0[\widetilde{\boldsymbol{F}}^\mathrm{T}\widetilde{\boldsymbol{F}} - \boldsymbol{I}] \tag{4.9}$$

- current placement:

$$\boldsymbol{T} = \frac{1}{\widetilde{J}}\widetilde{\boldsymbol{F}}\left(\frac{1}{2}\mathbb{K}_0[\widetilde{\boldsymbol{F}}^\mathrm{T}\widetilde{\boldsymbol{F}} - \boldsymbol{I}]\right)\widetilde{\boldsymbol{F}}^\mathrm{T} \tag{4.10}$$

4.3 Yield condition: Isotropic J2 by Huber and von Mises

We use an isotropic yield condition following the J_2 theory by Huber and von Mises. This means that the equivalent stress only depends on the second principle invariant of the deviatoric part of the Cauchy stresses. Together with the yield stress σ_{F}, we get the yield condition as

$$\sqrt{\frac{3}{2}}\|\boldsymbol{T}'\| - \sigma_{\mathrm{F}} = 0 \ . \tag{4.11}$$

4.4 Yield criterion: Principle of maximum plastic dissipation

For our yield criterion, we start with the stress power defined as

$$l = \frac{1}{\rho}\boldsymbol{T} \cdot\cdot \boldsymbol{L} \ . \tag{4.12}$$

Together with Eq. (4.7), we write the velocity gradient $\boldsymbol{L}$ as

$$\begin{aligned}
\boldsymbol{L} &= \dot{\boldsymbol{F}}\boldsymbol{F}^{-1} \\
&= \left(\widetilde{\boldsymbol{F}}\boldsymbol{P}^{-1}\right)^{\cdot} \left(\widetilde{\boldsymbol{F}}\boldsymbol{P}^{-1}\right)^{-1} \\
&= \left[\dot{\widetilde{\boldsymbol{F}}}\boldsymbol{P}^{-1} + \widetilde{\boldsymbol{F}}\left(\boldsymbol{P}^{-1}\right)^{\cdot}\right]\boldsymbol{P}\widetilde{\boldsymbol{F}}^{-1} \\
&= \dot{\widetilde{\boldsymbol{F}}}\widetilde{\boldsymbol{F}}^{-1} + \widetilde{\boldsymbol{F}}\left(\boldsymbol{P}^{-1}\right)^{\cdot}\boldsymbol{P}\widetilde{\boldsymbol{F}}^{-1}
\end{aligned} \tag{4.13}$$

and insert it into Eq. (4.12)

$$l = \underbrace{\frac{1}{\rho}\boldsymbol{T} \cdot\cdot \dot{\widetilde{\boldsymbol{F}}}\widetilde{\boldsymbol{F}}^{-1}}_{\dot{w},\ \text{elastic}} + \underbrace{\frac{1}{\rho}\boldsymbol{T} \cdot\cdot \widetilde{\boldsymbol{F}}\left(\boldsymbol{P}^{-1}\right)^{\cdot}\boldsymbol{P}\widetilde{\boldsymbol{F}}^{-1}}_{D,\ \text{dissipation}} \ . \tag{4.14}$$

with $\widetilde{\boldsymbol{F}} = \boldsymbol{F}\boldsymbol{P}$, and assume the principle of maximum plastic dissipation. The dissipation D will be maximal if the scalar product is maximal and this is the case for $\boldsymbol{T} \parallel \widetilde{\boldsymbol{F}}\left(\boldsymbol{P}^{-1}\right)^{\cdot}\boldsymbol{P}\widetilde{\boldsymbol{F}}^{-1}$ with an undetermined skew part of $\widetilde{\boldsymbol{F}}\left(\boldsymbol{P}^{-1}\right)^{\cdot}\boldsymbol{P}\widetilde{\boldsymbol{F}}^{-1}$ which is arbitrary. Together with the consistency parameter λ the yield criterion has the

following form:

$$\widetilde{\boldsymbol{F}}\left(\boldsymbol{P}^{-1}\right)^{\cdot}\boldsymbol{P}\widetilde{\boldsymbol{F}}^{-1}=\frac{\lambda}{\rho}\boldsymbol{T}\ . \tag{4.15}$$

Now we replace the Cauchy stress tensor $\boldsymbol{T}$ by the 2nd Piola-Kirchhoff stress tensor $\overset{\widetilde{2PK}}{\boldsymbol{T}}$

$$\widetilde{\boldsymbol{F}}\left(\boldsymbol{P}^{-1}\right)^{\cdot}\boldsymbol{P}\widetilde{\boldsymbol{F}}^{-1}=\frac{\lambda}{\rho\widetilde{J}}\widetilde{\boldsymbol{F}}\overset{\widetilde{2PK}}{\boldsymbol{T}}\widetilde{\boldsymbol{F}}^{\mathrm{T}}\ . \tag{4.16}$$

We use the relation $\rho_0=\widetilde{J}\rho$ and rewrite as follows:

$$\left(\boldsymbol{P}^{-1}\right)^{\cdot}\boldsymbol{P}=\frac{\lambda}{\rho_0}\overset{\widetilde{2PK}}{\boldsymbol{T}}\widetilde{\boldsymbol{F}}^{\mathrm{T}}\widetilde{\boldsymbol{F}}\ . \tag{4.17}$$

We take the deviatoric part of the right Cauchy-Green tensor $\widetilde{\boldsymbol{C}}$ because $\det(\boldsymbol{P})=1$ and therefore $\mathrm{tr}\left(\left(\boldsymbol{P}^{-1}\right)^{\cdot}\boldsymbol{P}\right)=0$

$$\left(\boldsymbol{P}^{-1}\right)^{\cdot}\boldsymbol{P}=\widetilde{\lambda}\left(\overset{\widetilde{2PK}}{\boldsymbol{T}}\widetilde{\boldsymbol{C}}\right)^{\prime} \tag{4.18}$$

and end up with the final form of our evolution equation

$$\dot{\boldsymbol{P}}=-\widetilde{\lambda}\boldsymbol{P}\left(\overset{\widetilde{2PK}}{\boldsymbol{T}}\widetilde{\boldsymbol{C}}\right)^{\prime}\ . \tag{4.19}$$

4.5 Numerical implementation

We implement the material model in the Abaqus user subroutine UMAT with the programming language Fortran. To solve the problem, we use the implicit Euler scheme (see Hairer, Nørsett, and Wanner (1993))

$$\boldsymbol{P}_{n+1}=\boldsymbol{P}_n+\Delta t\dot{\boldsymbol{P}}_{n+1} \tag{4.20}$$

together with the flow rule we get

$$\boldsymbol{P}_n=\boldsymbol{P}_{n+1}\left(\boldsymbol{I}+\lambda\left(\overset{\widetilde{2PK}}{\boldsymbol{T}}_{n+1}\widetilde{\boldsymbol{C}}_{n+1}\right)^{\prime}\right)\ . \tag{4.21}$$

As the next step, we use the exponential map (see Miehe (1996)) to ensure $\boldsymbol{P} \in Unim^{+}$. With the power series for the $\exp(x)$ function

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... \approx 1 + x \ , \tag{4.22}$$

we obtain

$$\boldsymbol{P}_n \approx \boldsymbol{P}_{n+1} \exp\left(\lambda \left(\widetilde{\overset{2PK}{\boldsymbol{T}}}_{n+1} \widetilde{\boldsymbol{C}}_{n+1}\right)'\right) \ . \tag{4.23}$$

The final residuum to solve the problem with Newton's method (see Ortega and Rheinboldt (1970)) reads

$$\boldsymbol{R} = \ln\left(\boldsymbol{P}_{n+1}^{-1}\boldsymbol{P}_n\right) - \lambda \left(\widetilde{\overset{2PK}{\boldsymbol{T}}}_{n+1} \widetilde{\boldsymbol{C}}_{n+1}\right)' \ . \tag{4.24}$$

The algorithm for this method is found in the Appendix C.5.

4.6 Validation of the material model

4.6.1 Elastic

First we validate the elastic behavior of our material model by calculating the stiffness tetrad of a homogeneous material and compare the results with literature and the Abaqus material. To determine the stiffness tetrad, we use the difference quotient (see Chapter 5) with a small elastic test deformation δ. In Fig. 4.1, we compare the deviation of the resulting stiffness tetrad to the analytical one in dependence of δ. Our own material model shows a better convergence even for larger δ-values. Within the shown figure, the largest relative error (0.0001) for the St. Venant-Kirchhoff model corresponds to the smallest error of the Abaqus model within the range 1e-9 $< \delta <$ 0.001. As a result, we find that our model firstly calculates the correct stiffness tetrads and secondly this is done with a better convergence.

4.6.2 Plastic

To validate the plastic behavior of our material model, we perform a tensile (see Fig. 4.2) and a shear test (see Fig. 4.3) with the material parameters Young's modulus E = 100GPa, Poissont's ratio ν=0.3 and yield stress of σ_{F}=100MPa. The results show,

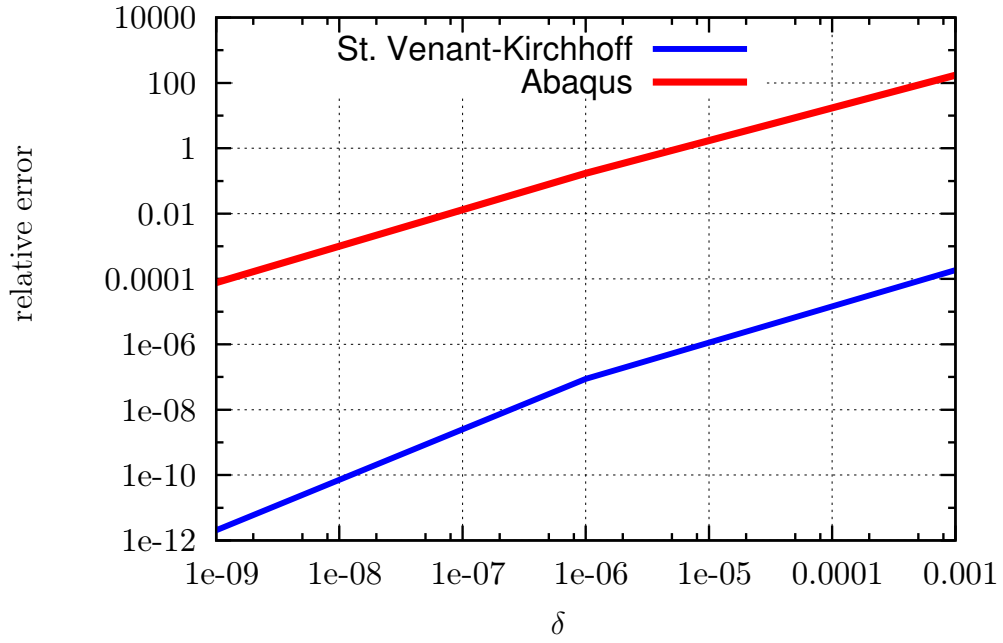


Figure 4.1: Error regarding the original stiffness tetrad of the St. Venant-Kirchhoff and Abaqus material over δ value of difference quotient (log-log)

as expected, a constant stress level while yielding for the tensile and the shear test. This means, we get proper results. For a final test, we compare the results of a 10% tensile test of our material model and the Abaqus elastic-plastic material applied on a notched cylindrical bar (Fig. 4.4) with the same material parameters as above. As a result, the von Mises stresses in Fig. 4.5 show a very good agreement comparing both material models. Additionally, we find a very good agreement of the von Mises stress distribution depicted in Fig. 4.5. Even small details look exactly the same. We can conclude that the material model on the micro-scale provides reliable results using the St.-Venant-Kichhoff law for the elastic behavior, the isomorphy concept for the change of the elastic range during large plastic deformations, the isotropic J2 yield condition by von Mises and the principle of maximum plastic dissipation as yield criterion. We use this material model for all following calculations.

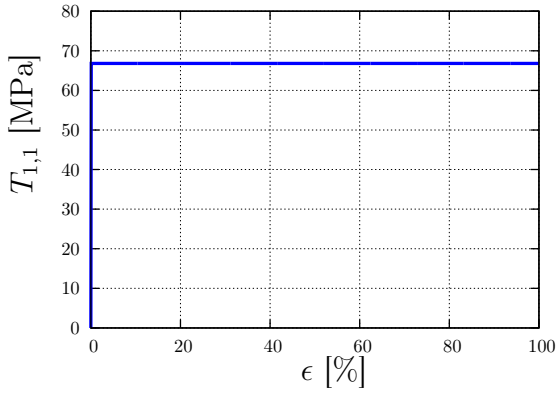


Figure 4.2: Tensile test: 100% strain

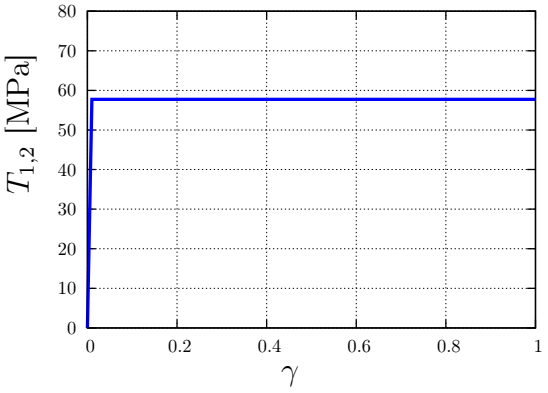


Figure 4.3: Shear test: shear number γ=1

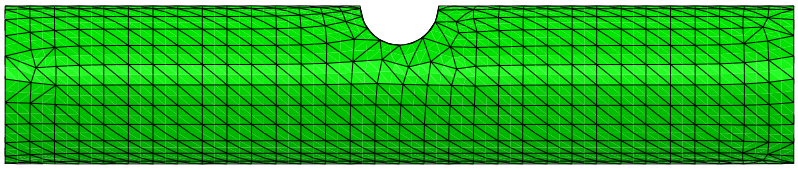


Figure 4.4: Mesh of a notched cylindrical bar

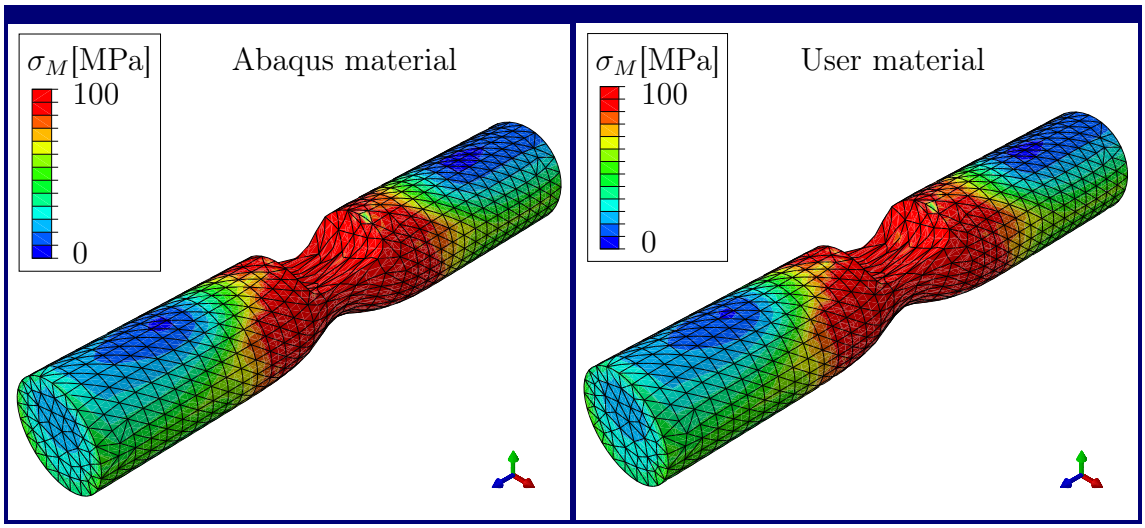


Figure 4.5: Von Mises stresses of the Abaqus material (left) and the user material (right).

4.7 Summary

In this chapter, an elastic-plastic material model for large deformations and the isomorphy concept on the micro-scale was used for the RVE calculations. A flowchart with further details can also be found in Fig. 5.2. The elastic behavior was captured by the St. Venant-Kirchhoff elasticity and the plastic behavior by the mentioned isomorphy concept. The yield condition follows the isotropic J2 theory by Huber and von Mises. The yield criterion was written in terms of the principle of maximum plastic dissipation. We applied different material parameters within this model for the matrix and the fiber phase. The model was validated regarding the elastic and plastic material behavior.

5 Determine the effective stiffness

In this chapter, we present a method to determine the effective stiffness of arbitrary inhomogeneous materials before and after large deformations. We will use the defined RVEs (see Chapter 3) to perform test deformations $\overset{\text{G}}{\boldsymbol{E}}$. Together with the resulting $\overset{\text{2PK}}{\boldsymbol{T}}$ stresses, we calculate the stiffness tetrad $\mathbb{K}$ during the post-processing with the computer algebra system Mathematica. A flowchart with further details can be found in Fig. 5.2.

5.1 RVE setup and extraction of the stiffness tetrads

To track the evolution of the stiffness tetrads $\mathbb{K}$, we have to determine the stiffness of the undeformed material ($\mathbb{K}_0$) and the stiffness tetrad of a deformed material after a large plastic deformation ($\mathbb{K}_1$). The stiffness tetrads will be calculated and compared in the reference placement and therefore we use the 2$^{\text{nd}}$ Piola-Kirchhoff stresses $\overset{\text{2PK}}{\boldsymbol{T}}$ and Green's strain tensor $\overset{\text{G}}{\boldsymbol{E}}$. If the theory of Material Plasticity holds, there will be a second-order tensor $\boldsymbol{P}_{\text{K}}$ which transforms the stiffness $\mathbb{K}_0$ of the undeformed material to $\mathbb{K}_1$ after a large plastic deformation.

For calculating the stiffness tetrads, we use the difference quotient out of six test calculations. Within the test calculations, six different small elastic test strains δ_i with i=1...6 are applied to the presented materials (RVEs). With the definitions

- $\boldsymbol{H}_0$: displacement gradient of the unloaded placement
- $\Delta\boldsymbol{H}_i$: displacement gradients of the 6 different elastic deformations

and the scheme in Figure 5.1, we are able to calculate a stiffness tetrad by the equation:

$$\Delta\overset{\text{2PK}}{\boldsymbol{T}}_i = \mathbb{K} : \Delta\overset{\text{G}}{\boldsymbol{E}}_i \ . \tag{5.1}$$

The scheme is carried out at the (average) stress-free placement. To get to this

$$\Delta \overset{\text{2PK}}{\boldsymbol{T}}_i \left\{ \begin{array}{lll} \overset{\text{2PK}}{\boldsymbol{T}}_0 \leftarrow & \boldsymbol{H}_0 & \rightarrow \overset{\text{G}}{\boldsymbol{E}}_0 \\ & \underline{+\ \Delta \boldsymbol{H}_i} & \\ \overset{\text{2PK}}{\boldsymbol{T}}_i \leftarrow & \underline{\underline{=\quad \boldsymbol{H}_i}} & \rightarrow \overset{\text{G}}{\boldsymbol{E}}_i \end{array} \right\} \Delta \overset{\text{G}}{\boldsymbol{E}}_i$$

Figure 5.1: Scheme to determine the test stresses $\Delta \overset{\text{2PK}}{\boldsymbol{T}}_i$ and the test strains $\Delta \overset{\text{G}}{\boldsymbol{E}}_i$ out of the test displacement gradients $\Delta \boldsymbol{H}_i$ for calculating the stiffness tetrad $\mathbb{K}$

placement, we unload component-wise and obtain the macroscopic $\boldsymbol{P}_{\text{C}}$. It turns out that due to the small elastic strains the difference between $\boldsymbol{P}_{\text{C}}$ and the deformation gradient $\boldsymbol{F}^{-1}$ is very small. This simplifies greatly the empirical approach, since we can in essence avoid an independent evolution of $\boldsymbol{P}_{\text{C}}$ but take $\boldsymbol{P}_{\text{C}} = \boldsymbol{F}^{-1}$ instead. The equation is implemented in a Mathematica script (see Appendix F).

5.2 Python script

For a better overview, the Python script generating the RVEs and performing the starting and test calculations is divided into different parts. In Fig. 5.2 a flowchart provides an overview over the different Python scripts, Fortran subroutines and functions and the Mathematica post-processing.

5.2.1 Starting calculation 1: "0_A.py"

The main part which has to be started is called "0_A.py" (see Appendix B.1). This script first calls "0_subroutines.py" (see Appendix B.9) to initialize subrountines like the calculation of a matrix determinant or a matrix multiplication. Afterwards the displacement gradient $\boldsymbol{H}_0$ is prescribed. Next the script "0_RVE.py" is called (see Chapter 3).

After that, the material parameters will be imported from the file 0_material.py (see Appendix B.11). At this point, the subrountine 0_rve.py is finished. A more detailed description of the generation of the different types of RVEs is given in the sections 3.8.1, 3.8.2 and 3.8.3.

The next step is to write the Abaqus input file. We are able to distinguish between different operating systems like Windows or Linux on which the Python script is running. The syntax slightly differs.

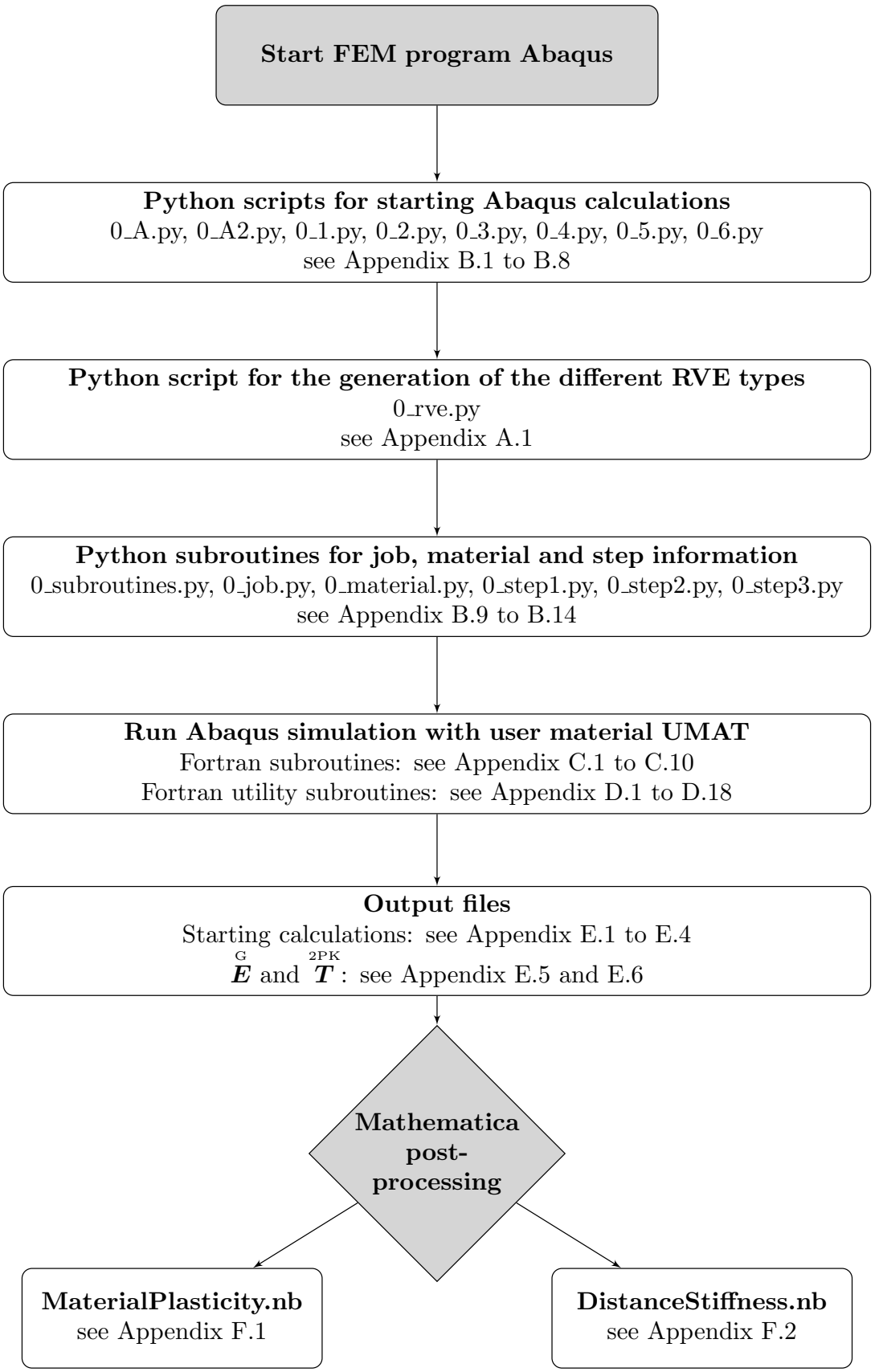


Figure 5.2: Flowchart giving an overview of the used Python, Fortran, and Mathematica codes

Hereinafter, the Abaqus Step definitions are given. Step 1 is the loading step prescribing the deformation as displacement following $\boldsymbol{H}_0$. In Step 2, the unloading takes place. Two different methods are distinguished. In case of uni-directional tension, the load is gradually removed (tension components, two shear components, finally the whole $\boldsymbol{H}_0$) to avoid rotations. $\boldsymbol{H}_0$ is removed at once in all other cases.

After the step definitions, the subroutine "0_job.py" is called (see Appendix B.10) using the before defined input file name "inpname" and the "cpunumber". The script will detect when the job is finished by checking the lock file. Also the successful execution is verified, else an error file is generated.

```
print '=============================================================='
print 'Run job'
print '=============================================================='
if "posix" in os.name:
    execfile('0_job.py')
# Wait for lck file, then wait for lck file is deleted -> job finished
t0 = time.time();
while not os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
while os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
# Check if calculation was successful, else STOP
time.sleep(5)
f = open(inpname+'.sta','r')
sta = []
for line in f:
    sta.append(str(line.rstrip()))
f.close()
if sta[len(sta)-1] == ' THE ANALYSIS HAS COMPLETED SUCCESSFULLY':
    print sta[len(sta)-1]
else:
    print sta[len(sta)-1]
    f = open('0_Auswertung_ERROR_in_'+inpname+'.txt','w')
    f.close()
    time.sleep(5)
    sys.exit("Calculation cancelled")
```

In the last section of the Python script "0_A.py", the post-processing is performed. Therefore the node numbers of the three artificial nodes are searched. Then the output database ODB is opened and the reaction forces ($RF1$, $RF2$ and $RF3$) such as the displacements ($U1$, $U2$ and $U3$) are read out. The predeformation is saved in the text file "K0_ANLAUFRECHNUNG_H0.txt" and the unloaded placement as $\boldsymbol{H}_1$ as "K0_ANLAUFRECHNUNG_H1.txt".

5.2.2 Starting calculation 2: "0_A2.py"

Next we need a second starting calculation "0_A2.py" (see Appendix B.2) to calculate the 2nd Piola-Kirchhoff stresses $\overset{2PK}{\boldsymbol{T}}_1$. For this purpose, we prescribe $\boldsymbol{H}_1$ from "0_A1.py" directly after the pre-deformation $\boldsymbol{H}_0$. Following the post-processing is performed like described for "0_A1.py". Additionally, the reaction forces divided by the volume result in the 1st Piola-Kirchhoff stresses $\overset{1PK}{\boldsymbol{T}}_1$. Using the equations

$$\boldsymbol{F}_1 = \boldsymbol{H}_1 + \boldsymbol{I} \tag{5.2}$$

and

$$\overset{2PK}{\boldsymbol{T}}_1 = \boldsymbol{F}_1^{-1} \overset{1PK}{\boldsymbol{T}}_1 \,, \tag{5.3}$$

we are able to calculate the 2nd Piola-Kirchhoff stresses $\overset{2PK}{\boldsymbol{T}}_1$. The stresses are saved in the text file "K0_ANLAUFRECHNUNG_T1.txt".

```
# Generate matrix T1PK
T1PK = nullmatrix(3,3)
T1PK[0][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].
    data[increm][1]/V0
T1PK[0][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR].
    data[increm][1]/V0
.
.
.
T1PK[2][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR].
    data[increm][1]/V0

del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR]
.
.
.
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR]

odb.close()

# Calculate T2PK
# F1 = H1 + 1
F1 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F1[i][j] = H1[i][j]

F1[0][0] += 1.
F1[1][1] += 1.
F1[2][2] += 1.
F1i = invert3(F1)
```

```
T2PK = matmul(F1i,T1PK)

print 'H1 end'
for line in H1: print line
print 'T1PK end'
for line in T1PK: print line
print 'T2PK end'
for line in T2PK: print line

#ausT.write('T2PK\n')
ausT = open('K0_ANLAUFRECHNUNG_T1.txt','w')
for i in range(3):
    for j in range(3):
        ausT.write('%e' %(T2PK[i][j]))
        if j != 2: ausT.write(',')
    ausT.write('\n')
ausT.close()
```

5.2.3 Elastic test strain calculations : "0_1.py" to "0_6.py"

Now the unloaded placement $\boldsymbol{H}_1$ and the residual stresses in the unloaded placement $\overset{1\text{PK}}{\boldsymbol{T}}_1$ are known. In the six elastic test calculations (i=1...6) to determine the stiffness tetrad $\mathbb{K}_1$ the following three steps are applied:

- Step 1: load step by prescribing $\boldsymbol{H}_0$
- Step 2: unload step by prescribing $\boldsymbol{H}_1$
- Step 3: elastic test strain by prescribing $\boldsymbol{H}_2^i$.

In the post-processing part the differences

$$\Delta \overset{\text{G}}{\boldsymbol{E}} = \overset{\text{G}}{\boldsymbol{E}}_2 - \overset{\text{G}}{\boldsymbol{E}}_1 \tag{5.4}$$

and

$$\Delta \overset{2\text{PK}}{\boldsymbol{T}} = \overset{2\text{PK}}{\boldsymbol{T}}_2 - \overset{2\text{PK}}{\boldsymbol{T}}_1 \tag{5.5}$$

are calculated. Therefore Green's strain tensor of the placements $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ are needed:

$$\overset{\text{G}}{\boldsymbol{E}}_1 = \frac{1}{2}\left(\boldsymbol{F}_1^{\text{T}}\boldsymbol{F}_1 - \boldsymbol{I}\right) \tag{5.6}$$

$$\overset{\mathrm{G}}{\boldsymbol{E}}_2 = \frac{1}{2}\left(\boldsymbol{F}_2^{\mathrm{T}}\boldsymbol{F}_2 - \boldsymbol{I}\right) \ . \tag{5.7}$$

The 2nd Piola-Kirchhoff stresses $\overset{\mathrm{2PK}}{\boldsymbol{T}}_1$ are already known. The $\overset{\mathrm{2PK}}{\boldsymbol{T}}_2$ stresses are calculated according to the Eqs. (5.2) and (5.3). For the further post-processing with the computer algebra system Mathematica, two files are generated. Both are saved as text file. The first one is "K0_matheEGREEN _einzeln.txt" and consists of the difference ΔE_G in the Abaqus notation of Cowin:

$$\Delta \overset{\mathrm{G}}{E} = \begin{pmatrix} \Delta\overset{\mathrm{G}}{E}_{11} \\ \Delta\overset{\mathrm{G}}{E}_{22} \\ \Delta\overset{\mathrm{G}}{E}_{33} \\ \Delta\overset{\mathrm{G}}{E}_{12} \\ \Delta\overset{\mathrm{G}}{E}_{13} \\ \Delta\overset{\mathrm{G}}{E}_{23} \end{pmatrix} \boldsymbol{B}_i \tag{5.8}$$

of all six calculation so that the file has 36 entries (see Appendix E.5). Analogously the second file consists of the difference of the 2nd Piola-Kirchhoff stresses $\overset{\mathrm{2PK}}{\boldsymbol{T}}$ and is saved in the text file "K0_matheT2PK_einzeln.txt" as

$$\Delta \overset{\mathrm{2PK}}{T} = \begin{pmatrix} \Delta\overset{\mathrm{2PK}}{T}_{11} \\ \Delta\overset{\mathrm{2PK}}{T}_{22} \\ \Delta\overset{\mathrm{2PK}}{T}_{33} \\ \Delta\overset{\mathrm{2PK}}{T}_{12} \\ \Delta\overset{\mathrm{2PK}}{T}_{13} \\ \Delta\overset{\mathrm{2PK}}{T}_{23} \end{pmatrix} \boldsymbol{B}_i \tag{5.9}$$

5.3 Validation of the stiffness calculation

We will compare the calculated stiffnesses to the literature to validate our implementation. If we prescribe $\boldsymbol{F}_0 = \boldsymbol{0}$ we will get the stiffness of the undeformed material

$\mathbb{K}_0$. For $\boldsymbol{F}_1$ being the unloaded placement after a large plastic deformation we will get the stiffness $\mathbb{K}_1$. To compare both, it is necessary to push-forward $\mathbb{K}_1$ to the new stress-free placement using the deformation gradient $\boldsymbol{F}$ which is equal to ${\boldsymbol{P}_\mathrm{C}}^{-1}$.

$$\begin{aligned} \mathbb{K}_0 &= \boldsymbol{F} * \mathbb{K}_1 \\ &= {\boldsymbol{P}_\mathrm{C}}^{-1} * \mathbb{K}_1 \end{aligned} \tag{5.10}$$

5.3.1 Isotropic material

We study an isotropic material in this section. The material parameters of the matrix and the fiber are identically (see Table 5.1). The components of the stiffness tetrad of

Material	Young's modulus E	Poisson's ratio ν	Yield stress σ_F
Matrix	100 GPa	0.3	200 MPa
Fiber	100 GPa	0.3	200 MPa

Table 5.1: Material parameters for the isotropic material on the micro-scale

an isotropic material can be found in the literature and are represented in Table 5.2. Using our implementation, we get the following stiffness tetrad $\mathbb{K}_0$ (values given in [GPa]) for the undeformed material (Table 5.3). It correctly represents an isotropic material. The next step is to validate the implementation for a plastically deformed material and to calculate the stiffness tetrad $\mathbb{K}_1$. First, we choose a 50% tensile test, unload the material sample and measure the stiffness tetrad. The resulting stiffness tetrad $\boldsymbol{F} * \mathbb{K}_1$ in Table 5.4 is identical to $\mathbb{K}_0$. The second test is a shear test with $\gamma = 0.5$ perpendicular to the fiber direction. After unloading and measuring the stiffness tetrad $\boldsymbol{F} * \mathbb{K}_1$, we find it is again identical to $\mathbb{K}_0$ (see Table 5.5).

$$\mathbb{K} = \begin{bmatrix} K_{1111} & K_{1122} & K_{1122} & 0 & 0 & 0 \\ & K_{1111} & K_{1122} & 0 & 0 & 0 \\ & & K_{1111} & 0 & 0 & 0 \\ & & & K_{1111}-K_{1122} & 0 & 0 \\ \text{sym} & & & & K_{1111}-K_{1122} & 0 \\ & & & & & K_{1111}-K_{1122} \end{bmatrix} \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

Table 5.2: Stiffness tetrad of an isotropic material

$$\mathbb{K}_0 = \begin{bmatrix} 135 & 58 & 58 & 0 & 0 & 0 \\ & 135 & 58 & 0 & 0 & 0 \\ & & 135 & 0 & 0 & 0 \\ & & & 77 & 0 & 0 \\ \text{sym} & & & & 77 & 0 \\ & & & & & 77 \end{bmatrix} \text{ GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

Table 5.3: Stiffness tetrad $\mathbb{K}_0$ of the calculated isotropic material

$$\boldsymbol{P}_{\mathrm{C}}{}^{-1} * \mathbb{K}_1 = \begin{bmatrix} 135 & 58 & 58 & 0 & 0 & 0 \\ & 135 & 58 & 0 & 0 & 0 \\ & & 135 & 0 & 0 & 0 \\ & & & 77 & 0 & 0 \\ \text{sym} & & & & 77 & 0 \\ & & & & & 77 \end{bmatrix} \text{ GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

$$= \mathbb{K}_0 = \begin{bmatrix} 135 & 58 & 58 & 0 & 0 & 0 \\ & 135 & 58 & 0 & 0 & 0 \\ & & 135 & 0 & 0 & 0 \\ & & & 77 & 0 & 0 \\ \text{sym} & & & & 77 & 0 \\ & & & & & 77 \end{bmatrix} \text{ GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

Table 5.4: Initially isotropic material after the tension test

$$\boldsymbol{P}_{\mathrm{C}}{}^{-1} * \mathbb{K}_1 = \begin{bmatrix} 135 & 58 & 58 & 0 & 0 & 0 \\ & 135 & 58 & 0 & 0 & 0 \\ & & 135 & 0 & 0 & 0 \\ & & & 77 & 0 & 0 \\ \text{sym} & & & & 77 & 0 \\ & & & & & 77 \end{bmatrix} \text{ GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

$$= \mathbb{K}_0 = \begin{bmatrix} 135 & 58 & 58 & 0 & 0 & 0 \\ & 135 & 58 & 0 & 0 & 0 \\ & & 135 & 0 & 0 & 0 \\ & & & 77 & 0 & 0 \\ \text{sym} & & & & 77 & 0 \\ & & & & & 77 \end{bmatrix} \text{ GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

Table 5.5: Initially isotropic material after the shear test

5.3.2 Transversal isotropic material

In the foregoing section, the material was isotropic. We want to study an anisotropic material and again use the RVE with the uni-directional reinforcement. But now the material parameters of the matrix and fiber material are different (see Table 5.6). The stiffness tetrad of a transversal isotropic material is given in literature

Material	Young's modulus E	Poisson's ratio ν	Yield stress σ_{F}
Matrix	10 GPa	0.3	100 MPa
Fiber	100 GPa	0.3	200 MPa

Table 5.6: Material parameters for the anisotropic material on the micro-scale

with the following five different material parameters (see Table 5.7). Using again our implementation, we get the following stiffness tetrad $\mathbb{K}_0$ for the undeformed transversal isotropic material (see Table 5.8). It correctly represents the transversal isotropic material from literature. Next, we validate the implementation for the plastically deformed material and calculate the stiffness tetrad $\mathbb{K}_1$. First, we choose a 50% tensile test, unload the material sample and measure the stiffness tetrad. The resulting stiffness tetrad $\boldsymbol{P}_{\mathrm{C}}{}^{-1} * \mathbb{K}_1$ in Table 5.9 is identical to $\mathbb{K}_0$ of the transversal isotropic material. Again, the second test is a shear test with $\gamma = 0.5$ perpendicular to the fiber direction. After unloading and measuring the stiffness tetrad $\boldsymbol{P}_{\mathrm{C}}{}^{-1} * \mathbb{K}_1$, we find it is not identical to $\mathbb{K}_0$ (see Table 5.10). The shear test of this anisotropic material shows the first time that the calculation of $\mathbb{K}_0 = \boldsymbol{P}_{\mathrm{C}}{}^{-1} * \mathbb{K}_1$ fails when we change the symmetry of the material during the deformation. There is a difference between both stiffnesses which means that for certain deformation the stiffness tetrad evolves. This is exactly the case what happens during the shear test of the transversal isotropic material. We will investigate and present some results for all three materials

$$\mathbb{K} = \begin{bmatrix} K_{1111} & K_{1122} & K_{1133} & 0 & 0 & 0 \\ & K_{1111} & K_{1133} & 0 & 0 & 0 \\ & & K_{3333} & 0 & 0 & 0 \\ & & & 2K_{2323} & 0 & 0 \\ \text{sym} & & & & 2K_{2323} & 0 \\ & & & & & K_{1111} - K_{1122} \end{bmatrix} \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

Table 5.7: Stiffness tetrad of a transversal isotropic material

$$\mathbb{K}_0 = \begin{bmatrix} 35 & 14 & 15 & 0 & 0 & 0 \\ & 35 & 15 & 0 & 0 & 0 \\ & & 73 & 0 & 0 & 0 \\ & & & 23 & 0 & 0 \\ sym & & & & 23 & 0 \\ & & & & & 21 \end{bmatrix} \text{ GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

Table 5.8: Stiffness tetrad $\mathbb{K}_0$ of the calculated transversal isotropic material

$$\boldsymbol{P}_{\mathrm{C}}{}^{-1} * \mathbb{K}_1 = \begin{bmatrix} 35 & 14 & 15 & 0 & 0 & 0 \\ & 35 & 15 & 0 & 0 & 0 \\ & & 73 & 0 & 0 & 0 \\ & & & 23 & 0 & 0 \\ sym & & & & 23 & 0 \\ & & & & & 21 \end{bmatrix} \text{ GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

$$= \mathbb{K}_0 = \begin{bmatrix} 35 & 14 & 15 & 0 & 0 & 0 \\ & 35 & 15 & 0 & 0 & 0 \\ & & 73 & 0 & 0 & 0 \\ & & & 23 & 0 & 0 \\ sym & & & & 23 & 0 \\ & & & & & 21 \end{bmatrix} \text{ GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

Table 5.9: Initially transversal isotropic material after the tension test

$$\boldsymbol{P}_{\mathrm{C}}{}^{-1} * \mathbb{K}_1 = \begin{bmatrix} 35 & 12 & 14 & 0 & 0 & 0 \\ & 40 & 16 & 0 & 0 & 0 \\ & & 73 & 0 & 0 & 0 \\ & & & 18 & 0 & 0 \\ sym & & & & 22 & 0 \\ & & & & & 25 \end{bmatrix} \text{ GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

$$\neq \mathbb{K}_0 = \begin{bmatrix} 35 & 14 & 15 & 0 & 0 & 0 \\ & 35 & 15 & 0 & 0 & 0 \\ & & 73 & 0 & 0 & 0 \\ & & & 23 & 0 & 0 \\ sym & & & & 23 & 0 \\ & & & & & 21 \end{bmatrix} \text{ GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

Table 5.10: Initially transversal isotropic material after the shear test

in the following sections. An overview of all performed calculations is given in Table 7.1 in Chapter 7.

5.4 Test cases

We specified different material parameters for the matrix and the fiber material as shown in Table 5.6. These parameters were chosen as an academic example. Ten different test deformations were examined as depicted in Table 7.1. Whenever a strain component is not specified it is implied to adjust freely such that the corresponding effective reaction stress is zero. In Eq. (5.11), we chose three elongation tests

$$\boldsymbol{H}_1 = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & - & 0 \\ 0 & 0 & - \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j, \quad \boldsymbol{H}_2 = \begin{bmatrix} - & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & - \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j, \quad \boldsymbol{H}_3 = \begin{bmatrix} - & 0 & 0 \\ 0 & - & 0 \\ 0 & 0 & 0.5 \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j. \tag{5.11}$$

In this case, we always deform the material parallel or perpendicular to the fiber direction. With the free lateral straining, this corresponds to a uniaxial tensile state. Next, we choose six shear tests which lead to different deformation modes depending on the shear mode and the chosen material (5.12)

$$\begin{aligned} \boldsymbol{H}_4 &= \begin{bmatrix} 0 & 0.5 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j, \quad \boldsymbol{H}_5 = \begin{bmatrix} 0 & 0 & 0.5 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j, \quad \boldsymbol{H}_6 = \begin{bmatrix} 0 & 0 & 0 \\ 0.5 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j, \\ \boldsymbol{H}_7 &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0.5 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j, \quad \boldsymbol{H}_8 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.5 & 0 & 0 \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j, \quad \boldsymbol{H}_9 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j. \end{aligned} \tag{5.12}$$

They can lead to a transport of the fibers with and without an effect on the symmetry and to a change of the fiber direction. We will discuss this in the next subsection. The last test (5.13) combines one tensile and two shear tests and acts as the worst case

$$\boldsymbol{H}_{10} = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0 & - & 0 \\ 0 & 0 & - \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j \ . \tag{5.13}$$

5.5 Results for the uni-directional reinforcement

The following three examples will show the deviation between the initial stiffness tetrad $\mathbb{K}_0$ and the stiffness tetrad $\mathbb{K}_1$ measured after a large deformation. We will investigate the three characteristic shear tests for the uni-directional reinforced material H_{xy} (equivalent to H_{yx}), H_{xz} (equivalent to H_{yz}) and H_{zx} (equivalent to H_{zy}). The first case is the shear test with H_{xy}. Fig. 5.3(a) shows that we have a transport of the fibers which leads to a change of the symmetry. The fiber direction remains the same. This type of deformation leads to a change of about 8% between the stiffnesses.

$$||\mathbb{K}_1 - \boldsymbol{P}_C * \mathbb{K}_0|| \;/\; ||\mathbb{K}_1|| = 7.7\% \tag{5.14}$$

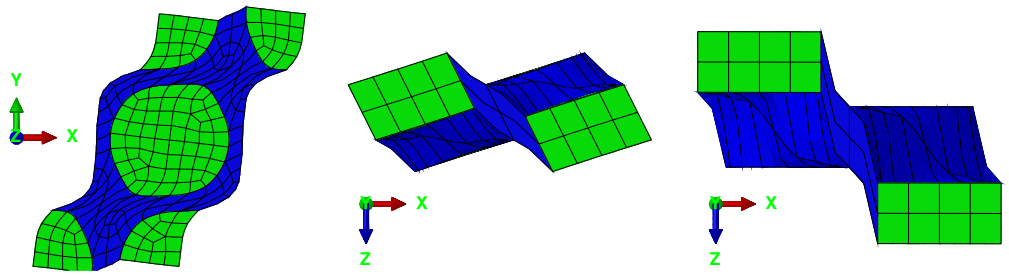
In Fig. 5.3(b), we see the shear test $H_{xz} = 0.5$ and no transport of the fibers. Instead, the fiber direction changes which leads to a change of the symmetry. We find the largest change in the stiffness tetrad as shown in Eq. (5.15).

$$||\mathbb{K}_1 - \boldsymbol{P}_C * \mathbb{K}_0|| \;/\; ||\mathbb{K}_1|| = 35.2\% \tag{5.15}$$

The shear test $H_{zx} = 0.5$ is depicted in Fig. 5.3(c) and shows that we have a transport of fibers in the fiber direction. The fiber direction remains the same so this motion has nearly no effect on the symmetry.

$$||\mathbb{K}_1 - \boldsymbol{P}_C * \mathbb{K}_0|| \;/\; ||\mathbb{K}_1|| = 0.5\% \tag{5.16}$$

We see that the deviation of the stiffness tetrad $\mathbb{K}_1$ from $\mathbb{K}_0$ can be considerable. A shear parallel to the fiber normal plane with $\gamma = 0.5$ results in a deviation by approximately 35%.



(a) $H_{xy} = 0.5$ (b) $H_{xz} = 0.5$ (c) $H_{zx} = 0.5$

Figure 5.3: Three different shear tests of the uni-directional reinforced material

5.6 Results for the bi-directional reinforcement

For the bi-directional reinforced material having its fibers in the directions x and y, the following three tests are performed: H_{xy} (equivalent to H_{yx}), H_{xz} (equivalent to H_{yz}) and H_{zx} (equivalent to H_{zy}). In Fig. 5.4(a), we find no transport of the fibers. Instead, the fiber direction changes which leads to a change of the symmetry

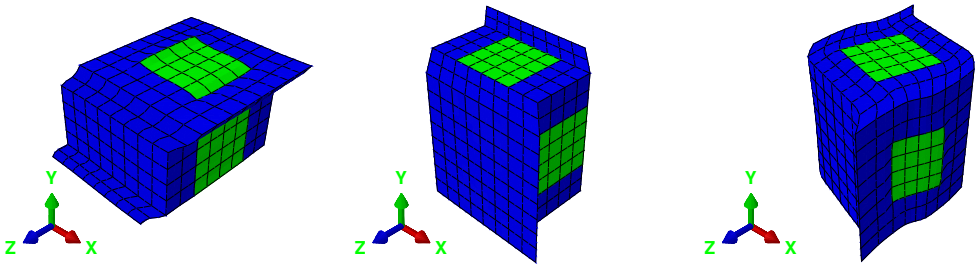
$$||\mathbb{K}_1 - \boldsymbol{P}_C * \mathbb{K}_0|| \;/\; ||\mathbb{K}_1|| = 23.6\% \tag{5.17}$$

The shear test H_{xz} results in nearly no change of the symmetry because neither the fiber direction nor the angle between the fibers is changed (see Fig. 5.4(b)).

$$||\mathbb{K}_1 - \boldsymbol{P}_C * \mathbb{K}_0|| \;/\; ||\mathbb{K}_1|| = 1.5\% \tag{5.18}$$

The shear test $H_{zx} = 0.5$ in Fig. 5.4(c) shows that we have a transport of the fibers which leads to large symmetry change of about 30%. The fiber direction remains the same.

$$||\mathbb{K}_1 - \boldsymbol{P}_C * \mathbb{K}_0|| \;/\; ||\mathbb{K}_1|| = 29.4\% \tag{5.19}$$



(a) $H_{xy} = 0.5$ (b) $H_{xz} = 0.5$ (c) $H_{zx} = 0.5$

Figure 5.4: Three different shear tests of the bi-directional reinforced material

5.7 Results for the tri-directional reinforcement

In the case of the tri-directional reinforced material, all shear tests lead to the same result. This is because the three fibers are arranged in the axis directions x, y and z. In all cases, the fiber direction between two of the fibers changes and the third fiber is transported. The deviation results in about 23%.

$$||\mathbb{K}_1 - \boldsymbol{P}_C * \mathbb{K}_0|| \;/\; ||\mathbb{K}_1|| = 22.6\% \tag{5.20}$$

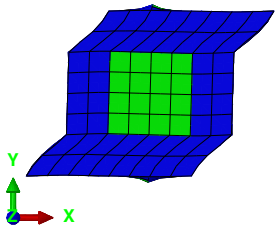


Figure 5.5: Shear test $H_{xy} = 0.5$ of the tri-directional reinforced material which is representative for all shear modes

5.8 Summary

The method to measure the stiffness tetrads was successfully implemented and validated. The results lead us to the Material Plasticity theory as discussed in Chapter 2. We need a second-order tensor $\boldsymbol{P}_{\mathrm{K}}$ which transforms between $\mathbb{K}_1$ and $\mathbb{K}_0$:

$$\mathbb{K}_0 = \boldsymbol{P}_{\mathrm{C}}\boldsymbol{P}_{\mathrm{K}} * \mathbb{K}_1 \ . \tag{5.21}$$

It will predict the evolution of the stiffness tetrad during a plastic deformation of an anisotropic material which changes the material symmetry. In Chapter 7 we will investigate the tensor $\boldsymbol{P}_{\mathrm{K}}$ and its evolution.

6 Distance of a stiffness tetrad to the symmetry classes of linear elasticity

The problems of identifying the symmetry class to which a specific stiffness tensor belongs, and further its distance to a higher symmetry class to which it does not belong are not trivial. Most materials, like crystals, exhibit spatial symmetries in their structure. These symmetries must be reflected in their material properties (Curie, 1894; Neumann, 1885). In particular, these symmetries must be obeyed by all material tensors which appear in the constitutive equations. Depending on the tensorial order, the symmetry requirement affects the material property tensor differently. The odd order tensors vanish in case that the material structure has centro-symmetry, i.e. that $-\boldsymbol{I}$ is in the materials symmetry group. The subgroups of the orthogonal group are for example detailed in Olive and Auffray (2014) Sec. 2.3 or Butler (1981) Sec. 5. In general, the higher the tensorial order, the more symmetries can be distinguished in the material properties. For instance, the second order heat conduction tensor is isotropic even for cubic crystals, while the fourth order stiffness tensor (tetrad) with cubic symmetry has three independent parameters, but an isotropic stiffness tetrad has only 2 independent parameters. Forte and Vianello (1996) show that in linear elasticity 8 material symmetries can be distinguished, namely triclinic, monoclinic, orthotropic, tetragonal, trigonal, hexagonal, cubic and isotropic. For a given stiffness tetrad of unknown symmetry resulting from measurement or numerical simulations, the question arises to which symmetry class it belongs. Since the coefficients of the stiffness tetrad are in general not exact, the affiliation to one of the eight material symmetry classes may neither be exact. It is therefore reasonable to define a distance as a deviation measure between a stiffness tetrad and the set of all stiffness tetrads with a certain symmetry, which we will call the symmetry class. This may serve to filter stochastic scatterings from measurements, see also Guilleminot and Soize (2010) and Norris (2006). Further, one may be able to reduce the complexity of the elastic model by replacing a complex (possibly triclinic) stiffness tetrad by a more symmetric candidate that is close enough to the original stiffness.

In literature one can find several approaches to this topic. In Moakher and Norris (2006), three different distance functions are defined to determine the closest elastic

tensor of arbitrary symmetry to an elasticity tensor of lower symmetry. They consider the Euclidean/Frobenius norm, the log-Euclidean norm and the Riemannian norm. In Guilleminot and Soize (2010), these distance measures are used as a starting point to determine the distances of stochastically generated stiffness tetrads to the set of transversal isotropic stiffnesses. Böhlke (2001) and Fedorov (1968) use the Frobenius norm to measure the distance between two stiffness tensors as we will do later on. It has been shown by Gazis, Tadjbakhsh, and Toupin (1963) that the average group action gives a projection of a stiffness tetrad onto the subset of all stiffness tetrads with a given symmetry. In Glüge, Weber, and Bertram (2012), in the course of determining the anisotropy induced by a representative volume element (RVE) of cubic shape, a similar projection method is used to quantify the anisotropy of a stiffness tetrad and also the distance to the cubic symmetry class. In the same article, one projection is onto the isotropic stiffnesses and the other one to the cubic stiffnesses of the anisotropy axes that are aligned with the RVE. Thus we do not need to find the orientation in this special situation. In the same article, a logarithmic normalization is presented, which ensures consistency in the sense that the same distance is measured if the inverse of the stiffness tetrad, namely the compliance tetrad, is considered.

When the anisotropy axes are unknown, the sought distance is w.r.t. a symmetry class, but not w.r.t. a specific symmetry group. One of the first to take the unknown orientation into account were Francois, Geymonat, and Berthaud (1998), who inspect pole figures to determine the symmetry visually. Afterwards, similar to our projection method, a so called orbit is defined as a collection of all transformations within the chosen group. In contrast to our approach, Francois, Geymonat, and Berthaud (ibid.) perform the minimization of the distance over the Euler angles after the projection. Further, this problem is addressed in Diner, Kochetov, and M. Slawinski (2011) and Kochetov and M. Slawinski (2009), who also use the Frobenius norm and distance functions from transversal and monoclinic symmetry, respectively. They also take into account the unknown orientation of the symmetry axes or planes, minimizing over two angles. This is possible due to a simplification that is only applicable for these two symmetry classes. We will use this simplification to remove one of the Euler angles from the minimization procedure. A different approach is chosen by Zou, Tang, and W. Lee (2013). They determine the exact symmetry class to which a tetrad belongs by a harmonic decomposition and a multipole representation of its deviators. Then the angular deviation of the vectors from the multipole representation is used as a distance measure. They state that *it is impossible to find a simple function to define the distance between the elastic tensor measured experimentally and its nearest possible symmetry groups*.

In this chapter, we present a method to determine this distance. As Zou, Tang, and W. Lee (2013) state, we can not give a simple, closed form solution for the problem, but a fast and reliable numerical algorithm instead. To find out the closest symmetry class of the measured stiffness tetrad we determine the Euclidean distance

to all possible symmetry classes by employing a projection method. Further, we need to find the closest orientation (see Cowin and Mehrabadi (1987) and Francois, Geymonat, and Berthaud (1998)). We do this by minimizing the Euclidean distance between the rotated stiffness to the rotated and projected stiffness over the three Euler angles. Although this minimization problem exhibits some peculiarities which need to be considered, it leads to a fast and robust algorithm for finding the distance to the seven non-trivial symmetry classes of linear elasticity. In addition to these distances, the algorithm also gives the elements in the respective symmetry classes that are closest to the stiffness tetrad under consideration. Finally, we apply the method to stiffnesses of unknown symmetries which emerge from RVE simulations of three different sample materials.

6.1 Definitions

6.1.1 Symmetry transformation

A tetrad $\mathbb{K}$ is symmetric w.r.t. the rotation $\boldsymbol{Q}$ if

$$\mathbb{K} = \boldsymbol{Q} * \mathbb{K} \tag{6.1}$$

holds. Thus, the transformation $\boldsymbol{Q} * \mathbb{K}$ does not alter $\mathbb{K}$ and is called a symmetry transformation. The set of all such symmetry transformations form the symmetry group of the tetrad. Due to the even number of entries in $\mathbb{K}$, we have $\mathbb{K} = -\boldsymbol{I} * \mathbb{K}$. Thus, all elements from $\mathcal{O}rth^-$ can be generated by multiplying the elements of $\mathcal{O}rth^+$ with $-\boldsymbol{I}$. Therefore, it is sufficient to focus on $\mathcal{O}rth^+$ and its subgroups, the expansion to the whole $\mathcal{O}rth$ does not give any new insight for tensors of even order.

6.1.2 Symmetry group

All symmetry groups $\mathcal{G} = \{\boldsymbol{I}, \boldsymbol{Q}_2, \ldots, \boldsymbol{Q}_n\}$ of solids are subgroups of the orthogonal group $\mathcal{O}rth^+$ which is closed under an associative composition $\boldsymbol{Q}_i = \boldsymbol{Q}_j \cdot \boldsymbol{Q}_k$, $\boldsymbol{Q}_{i,j,k} \in \mathcal{G}$. Further, it contains the identity $\boldsymbol{I} \cdot \boldsymbol{Q}_i = \boldsymbol{Q}_i$, by which an inverse is assigned to any group element $\boldsymbol{Q}_i \cdot \boldsymbol{Q}_i^T = \boldsymbol{I}$, which is also part of the group. All N elements $\boldsymbol{Q}_i$ can be generated from a non-unique minimum set of generators by these group operations.

One can see that an expansion of the form $\mathcal{G}^* = \{\boldsymbol{I}, \boldsymbol{Q}^T\boldsymbol{Q}_2\boldsymbol{Q}, \ldots, \boldsymbol{Q}^T\boldsymbol{Q}_N\boldsymbol{Q}\}$ with $\boldsymbol{Q} \in \mathcal{O}rth^+$ produces a symmetry group $\mathcal{G}^*$ which is isomorphic to $\mathcal{G}$. Hence, for all isomorphic groups one may choose a reference group. The eight reference groups that are used here are given in Table 6.1 (see e.g. Hermann (1934) and Voigt (1910)). One

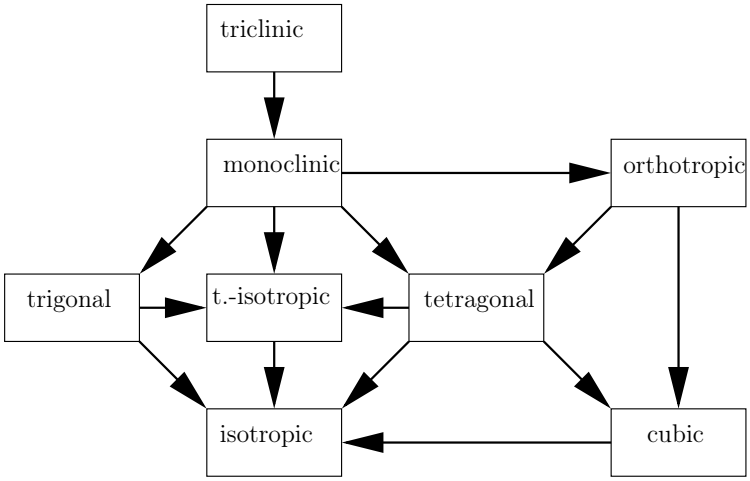


Figure 6.1: Symmetry inclusion scheme. The arrows indicate a subset relation, pointing from the larger set to the subset.

can sort the symmetry classes according to the set inclusion scheme given in Fig. 6.1, see Bóna, Bucataru, and A. Slawinski (2004). We refer to symmetries as higher and lower if a subset relation holds. Lower symmetries include higher symmetries as special cases, but not all symmetry classes are nested. One can see that in Fig. 6.1 trigonal and tetragonal symmetries are not connected by a unidirectional path of arrows, so there exists no subset relation for these two symmetries. It should be noted that one can also sort the symmetries into higher and lower by the symmetry group size N (right column in Table 6.1).

6.1.3 Symmetry class

The symmetry class $\mathcal{K}_{\mathrm{sym}_i}$ denotes all stiffness tetrads for which a rotation $\boldsymbol{Q}$ can be found such that the rotated stiffness is invariant under the action of the reference symmetry group sym_i as labeled in the first column in Table 6.1,

$$\boldsymbol{Q} * \mathbb{K} = \boldsymbol{Q}_j * (\boldsymbol{Q} * \mathbb{K}) \quad \forall \boldsymbol{Q}_j \in \mathcal{G}_{\mathrm{sym}_i} \quad \forall \mathbb{K} \in \mathcal{K}_{\mathrm{sym}_i} \tag{6.2}$$

which can be written as

$$\mathbb{O} = \boldsymbol{Q} * \mathbb{K} - (\boldsymbol{Q}_j \boldsymbol{Q}) * \mathbb{K} \quad \forall \boldsymbol{Q}_j \in \mathcal{G}_{\mathrm{sym}_i} \quad \forall \mathbb{K} \in \mathcal{K}_{\mathrm{sym}_i}. \tag{6.3}$$

By the properties of the Rayleigh product, this may be rewritten as

$$\mathbb{O} = \mathbb{K} - (\boldsymbol{Q}^T \boldsymbol{Q}_j \boldsymbol{Q}) * \mathbb{K} \quad \forall \boldsymbol{Q}_j \in \mathcal{G}_{\mathrm{sym}_i} \quad \forall \mathbb{K} \in \mathcal{K}_{\mathrm{sym}_i}\ . \tag{6.4}$$

It is obvious that one can either rotate $\mathbb{K}$ and check w.r.t. the reference symmetry group or keep $\mathbb{K}$ fixed and check the rotated reference symmetry group. From a

symmetry	reference generators	indep. const. $\mathbb{K}$	no. of ele. N in $\mathcal{G}$
triclinic	$\boldsymbol{I}$ $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	21	1
monoclinic	$\boldsymbol{Q}^{\pi}_{\boldsymbol{e}_1}$ $\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$	13	2
orthotropic	$\boldsymbol{Q}^{\pi}_{\boldsymbol{e}_1}$, $\boldsymbol{Q}^{\pi}_{\boldsymbol{e}_2}$ $\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$, $\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$	9	4
trigonal	$\boldsymbol{Q}^{\pi}_{\boldsymbol{e}_2}$, $\boldsymbol{Q}^{2\pi/3}_{\boldsymbol{e}_3}$ $\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$, $\begin{pmatrix} -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$	6	6
tetragonal	$\boldsymbol{Q}^{\pi}_{\boldsymbol{e}_1}$, $\boldsymbol{Q}^{3\pi/2}_{\boldsymbol{e}_3}$ $\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$, $\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	6	8
hexagonal	$\boldsymbol{Q}^{\pi}_{\boldsymbol{e}_1}$, $\boldsymbol{Q}^{\pi/3}_{\boldsymbol{e}_3}$ $\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$, $\begin{pmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$	5	12
cubic	$\boldsymbol{Q}^{3\pi/2}_{\boldsymbol{e}_1}$, $\boldsymbol{Q}^{2\pi/3}_{\frac{1}{\sqrt{3}}(\boldsymbol{e}_1+\boldsymbol{e}_2+\boldsymbol{e}_3)}$ $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}$, $\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$	3	24
isotropic	$\boldsymbol{Q}^{\phi}_{\boldsymbol{v}}$, ϕ and $\boldsymbol{v}$ arbitrary -	2	∞

Table 6.1: Possible symmetries of the stiffness tetrad $\mathbb{K}$

practical point of view, Eq. (6.3) offers the advantage that the Rayleigh product with $\boldsymbol{Q}$ appears only once in each summand, and that only one rotation (only $\boldsymbol{Q}$, not $\boldsymbol{Q}^T$) is needed.

6.1.4 Distance measure

Let us parametrize a rotation $\boldsymbol{Q}$ by three Euler angles α, β and γ. For stiffness tetrads which do not belong to a certain symmetry class $\mathcal{G}_{\mathrm{sym}}$, a residuum remains on the left side of Eq. (6.3), i.e.

$$\mathbb{R}_i(\alpha,\beta,\gamma) = \boldsymbol{Q}_{\alpha,\beta,\gamma} * \mathbb{K} - (\boldsymbol{Q}_i \boldsymbol{Q}_{\alpha,\beta,\gamma}) * \mathbb{K} \quad \boldsymbol{Q}_i \in \mathcal{G}_{\mathrm{sym}} \quad \mathbb{K} \in \mathcal{K}. \tag{6.5}$$

$\mathbb{R}_i(\alpha,\beta,\gamma)$ is a difference between stiffnesses. We obtain a relative residuum by normalizing with $\|\mathbb{K}\|$. Since $\|\boldsymbol{Q}*\mathbb{K}\| = \|\mathbb{K}\|$ we can define the relative residuum by

$$\mathbb{R}_i^*(\alpha,\beta,\gamma) = \boldsymbol{Q}_{\alpha,\beta,\gamma} * \mathbb{K}^* - (\boldsymbol{Q}_i \boldsymbol{Q}_{\alpha,\beta,\gamma}) * \mathbb{K}^* \quad \boldsymbol{Q}_i \in \mathcal{G}_{\mathrm{sym}} \tag{6.6}$$

with $\mathbb{K}^* = \mathbb{K}/\|\mathbb{K}\|$. We can consider the norm $\|\mathbb{R}_i^*(\alpha,\beta,\gamma)\|$ as the relative difference between some stiffness tetrad that belongs to the symmetry class $\mathcal{K}_{\mathrm{sym}}$ and the stiffness tetrad $\mathbb{K}$. The global minimum

$$d = \min_{i,\alpha,\beta,\gamma} \|\mathbb{R}_i^*(\alpha,\beta,\gamma)\| \tag{6.7}$$

gives the closest relative distance between $\mathbb{K}$ and all tetrads that belong to the symmetry class $\mathcal{K}_{\mathrm{sym}}$.

6.2 Minimization over the elements of the symmetry group by a projection method

The minimization over the index i that labels the group elements is carried out by a projection. This projection is basically the average under the action of all group members. It can be formulated for the seven anisotropic classes with finite group sizes N as

$$\overline{\mathbb{K}}_{\mathcal{G}} = \frac{1}{N} \sum_{z=1}^{N} (\boldsymbol{Q}_z * \mathbb{K}) \,. \tag{6.8}$$

Because of the linearity of the Rayleigh product in the second argument

$$\boldsymbol{Q} * (\alpha\mathbb{K}_1 + \mathbb{K}_2) = \boldsymbol{Q} * (\alpha\mathbb{K}_1) + \boldsymbol{Q} * \mathbb{K}_2, \tag{6.9}$$

it is possible to denote this with an 8^{th} order linear mapping $\overset{\langle 8\rangle}{\mathbb{P}}$ as

$$\overline{\mathbb{K}}_{\mathcal{G}} = \overset{\langle 8\rangle}{\mathbb{P}} :: \mathbb{K} \qquad \text{with} \tag{6.10}$$

$$\overset{\langle 8\rangle}{\mathbb{P}} = \frac{1}{N} \sum_{z=1}^{N} Q^z_{im} Q^z_{jn} Q^z_{ko} Q^z_{lp} \boldsymbol{e}_i \otimes \boldsymbol{e}_j \otimes \boldsymbol{e}_k \otimes \boldsymbol{e}_l \otimes \boldsymbol{e}_m \otimes \boldsymbol{e}_n \otimes \boldsymbol{e}_o \otimes \boldsymbol{e}_p \tag{6.11}$$

which we label $\overset{\langle 8\rangle}{\mathbb{P}}$ in anticipation of its projector properties. To demonstrate this property, we rotate the mean value with a second group element $\boldsymbol{Q}_y$.

$$\boldsymbol{Q}_y * \overline{\mathbb{K}}_{\mathcal{G}} = \frac{1}{N}\sum_{z=1}^{N} \boldsymbol{Q}_y * (\boldsymbol{Q}_z * \mathbb{K}) \tag{6.12}$$

$$= \frac{1}{N}\sum_{z=1}^{N} (\boldsymbol{Q}_y \boldsymbol{Q}_z) * \mathbb{K} \tag{6.13}$$

$$= \frac{1}{N}\sum_{z=1}^{N} \boldsymbol{Q}_z^* * \mathbb{K} \tag{6.14}$$

$$= \overline{\mathbb{K}}_{\mathcal{G}}. \tag{6.15}$$

The composition of $\boldsymbol{Q}_y$ and $\boldsymbol{Q}_z$ gives, following the group axioms, a mere re-indexing of the group members to $\boldsymbol{Q}_z^*$, which does not affect the result. $\overset{\langle 8\rangle}{\mathbb{P}}$ acts as the identity on the set of all $\mathbb{K}$ which are $\mathcal{G}$-symmetric, and projects otherwise all non-$\mathcal{G}$-symmetric $\mathbb{K}$ into their $\mathcal{G}$-symmetric part.

The section is concluded by noting that the linearity of the Rayleigh product Eq. (6.9) in the second argument together with the definition of symmetry $\boldsymbol{Q}_i * \mathbb{K} = \mathbb{K}$ under the action of $\boldsymbol{Q}_i$ imply the convexity of the set of all tensors $\mathbb{K}$ that are invariant under the action of a certain symmetry group $\mathcal{G}$, i.e.

$$\text{If } \mathbb{K}_1, \mathbb{K}_2 \in \mathcal{K}_{\text{sym}} \text{ then } \alpha\mathbb{K}_1 + (1-\alpha)\mathbb{K}_2 \ \in \mathcal{K}_{\text{sym}} \ \forall\ \alpha \in [0,1]. \tag{6.16}$$

Because of the convexity of the subspace $\mathcal{K}_{\text{sym}} \subset \mathcal{K}$, the projection operation is unique. It is not hard to verify that the part of $\mathbb{K} - \overline{\mathbb{K}}_{\mathcal{G}}$ that is removed by the projection is perpendicular to the remainder $\overline{\mathbb{K}}_{\mathcal{G}}$, i.e.

$$(\mathbb{K} - \overline{\mathbb{K}}_{\mathcal{G}}) :: \overline{\mathbb{K}}_{\mathcal{G}} = 0. \tag{6.17}$$

This can be easily seen by expanding the right factor with the projector $\overline{\mathbb{K}}_{\mathcal{G}} = \overset{\langle 8\rangle}{\mathbb{P}} :: \overline{\mathbb{K}}_{\mathcal{G}}$ and associating it with the left factor. In conjunction, the orthogonality and the uniqueness due to the convexity of the subset guarantee that the associated distance between $\mathbb{K}$ and $\overset{\langle 8\rangle}{\mathbb{P}}\mathbb{K}$ is minimal. This has also been found by Gazis, Tadjbakhsh, and Toupin (1963), Sec. 2.

6.3 Minimization over the Euler angles

We have to minimize the distance between the arbitrarily oriented stiffness tetrad and a chosen fixed reference symmetry class. The orientation is parameterized by three Euler angles. We use the yaw-pitch-roll convention that rotates w.r.t. the fixed axes firstly by α around $\boldsymbol{e}_z$, then β around $\boldsymbol{e}_y$ and finally by γ around $\boldsymbol{e}_x$ (see Morawiec (2004) Sec. 2.4.1). This leads to a component form of the rotation tensor

$$\boldsymbol{Q}_{\alpha,\beta,\gamma} = \begin{pmatrix} \cos\alpha\cos\beta & -\cos\beta\sin\alpha & \sin\beta \\ \cos\gamma\sin\alpha & \cos\alpha\cos\gamma & -\cos\beta\sin\gamma \\ +\cos\alpha\sin\beta\sin\gamma & -\sin\alpha\sin\beta\sin\gamma & \\ -\cos\alpha\cos\gamma\sin\beta & \cos\gamma\sin\alpha\sin\beta & \cos\beta\cos\gamma \\ +\sin\alpha\sin\gamma & +\cos\alpha\sin\gamma & \end{pmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j. \quad (6.18)$$

Again, the application of the Rayleigh product can be written as a linear mapping $\boldsymbol{Q}_{\alpha,\beta,\gamma} * \mathbb{K} = \overset{\langle 8\rangle}{\mathbb{Q}}_{\alpha,\beta,\gamma} :: \mathbb{K}$ with an 8^{th} order tensor which covers the rotation

$$\overset{\langle 8\rangle}{\mathbb{Q}}_{\alpha,\beta,\gamma} = Q_{im|\alpha,\beta,\gamma} Q_{jn|\alpha,\beta,\gamma} Q_{ko|\alpha,\beta,\gamma} Q_{lp|\alpha,\beta,\gamma} \boldsymbol{e}_i \otimes \boldsymbol{e}_j \otimes \boldsymbol{e}_k \otimes \boldsymbol{e}_l \otimes \boldsymbol{e}_m \otimes \boldsymbol{e}_n \otimes \boldsymbol{e}_o \otimes \boldsymbol{e}_p \quad (6.19)$$

6.4 Combination of projection and rotation

Finally, we combine the 8^{th} order projection tensor $\overset{\langle 8\rangle}{\mathbb{P}}$ and the 8^{th} order rotation tensor $\overset{\langle 8\rangle}{\mathbb{Q}}_{\alpha,\beta,\gamma}$ to the final transformation

$$\overset{\langle 8\rangle}{\mathbb{P}\mathbb{Q}}_{\alpha,\beta,\gamma} = \overset{\langle 8\rangle}{\mathbb{P}} :: \overset{\langle 8\rangle}{\mathbb{Q}}_{\alpha,\beta,\gamma} \ . \quad (6.20)$$

In the projection $\overset{\langle 8\rangle}{\mathbb{P}\mathbb{Q}}_{\alpha,\beta,\gamma}$ there are three unknown Euler angles α, β and γ. W.r.t. these angles, the Frobenius norm on fourth order tensors of the difference between the rotated $\mathbb{K}$ and the rotated and projected $\mathbb{K}$ has to be minimized. A sketch of the orbit and the projection to the symmetry groups is given in Figure 6.2. This procedure differs from Francois, Geymonat, and Berthaud (1998) since we first apply the rotation and then determine the relative distance to the symmetry classes.

$$d = \min_{i,\alpha,\beta,\gamma} \| \overset{\langle 8\rangle}{\mathbb{Q}}_{\alpha,\beta,\gamma} :: \mathbb{K}^* - \overset{\langle 8\rangle}{\mathbb{P}\mathbb{Q}}_{\alpha,\beta,\gamma} :: \mathbb{K}^* \|, \qquad \mathbb{K}^* = \mathbb{K}/\|\mathbb{K}\|. \quad (6.21)$$

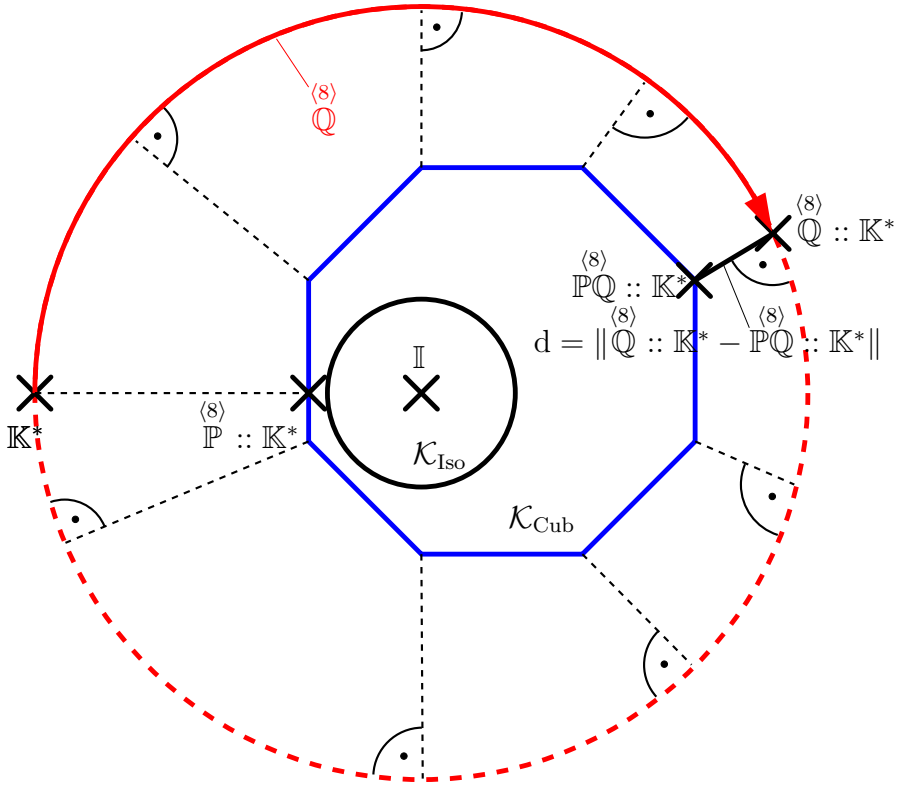


Figure 6.2: Schematic representation of the projection method. The paper plane represents all stiffness tetrads, where we pick some specific $\mathbb{K}^* = \mathbb{K}/\|\mathbb{K}\|$. The octagonal region is the subset of all cubic stiffness tetrads $\mathcal{K}_{\mathrm{Cub}}$, w.r.t. fixed axes. Within this region, the solid circle region represents the set of all isotropic stiffness tetrads $\mathcal{K}_{\mathrm{Iso}}$ as a subset of the octagonal region. The outer circle section is a part of the orbit of $\mathbb{K}$ under the action of $\mathcal{O}rth^+$ depicted as the outer large dashed circle.

6.5 Distance to isotropy

In the case of isotropy, we have a continuous group with infinitely many members and generators. Therefore we have to apply a different method for defining distances

with the isotropy class. For this purpose, we use the isotropic projectors

$$\mathbb{K} = \lambda_1 \mathbb{P}_1 + \lambda_2 \mathbb{P}_2, \tag{6.22}$$

$$\mathbb{P}_1 = \frac{1}{3} \boldsymbol{I} \otimes \boldsymbol{I}, \tag{6.23}$$

$$\mathbb{P}_2 = \mathbb{I} - \mathbb{P}_1, \tag{6.24}$$

with $\mathbb{I}$ the identity tensor on symmetric second order tensors and the eigenvalues $\lambda_{1,2}$. The isotropic projectors map into isotropic 1- and 5-dimensional subspaces of the space of symmetric second order tensors, namely the spherical and deviatoric subspaces, respectively. Most important for our work is the fact that both of these subspaces are closed under all proper rotations $\boldsymbol{Q} \in \mathcal{O}rth^+$. Thus, from the discussion in the previous section we do not have to distinguish between a reference symmetry group and all isomorphic symmetry groups in the isotropic case. Consequently, the minimization over the three Euler angels is not needed and we can directly apply directly the derived 8^{th} order projector. Generalizing the sums from the discrete sets to integrals (Morawiec, 2004), we get

$$\overset{\langle 8 \rangle}{\mathbb{P}}_{\text{iso}} = \frac{1}{|\mathcal{O}rth^+|} \int_{\mathcal{O}rth^+} Q_{im} Q_{jn} Q_{ko} Q_{lp} \boldsymbol{e}_i \otimes \boldsymbol{e}_j \otimes \boldsymbol{e}_k \otimes \boldsymbol{e}_l \otimes \boldsymbol{e}_m \otimes \boldsymbol{e}_n \otimes \boldsymbol{e}_o \otimes \boldsymbol{e}_p \mathrm{d}\boldsymbol{Q} \tag{6.25}$$

$$= \mathbb{P}_1 \otimes \mathbb{P}_1 + \frac{1}{5} \mathbb{P}_2 \otimes \mathbb{P}_2. \tag{6.26}$$

6.6 Minimization procedure

The function value to be minimized is periodic due to the Euler angle parametrization. It is sufficient to consider the intervals $\alpha, \gamma \in [0, 2\pi)$ and $\beta \in [-\pi/2, \pi/2]$ for the three Euler angles. Especially when minimizing the distance of a highly symmetric stiffness tetrad to a high symmetric symmetry class (e.g. both cubic), the actual periodicity is much smaller than these intervals, and the global minimum has a large number of equivalent solutions in terms of the Euler angles. The most anisotropic case is when we seek the distance of a triclinic stiffness tetrad to the monoclinic symmetry class. In this situation, the global minimum which we seek is two-fold due to the one nontrivial symmetry operation in the monoclinic case (see Table 6.1 and Fig. 6.3). In all other cases, the multiplicity will be larger than two. It is therefore possible to halve the search interval without loss of information in general. However, the reduction of the search domain in the Euler angle space is not trivial, see Jöchen and Böhlke (2012). Nevertheless, in the case of the monoclinic and the transversely isotropic distance functions, one can choose the symmetry group representation and the parametrization of $\mathcal{O}rth^+$ by the Euler angles α, β and γ such that the distance becomes independent of one of the Euler angles, see Diner, Kochetov, and M. Slaw-

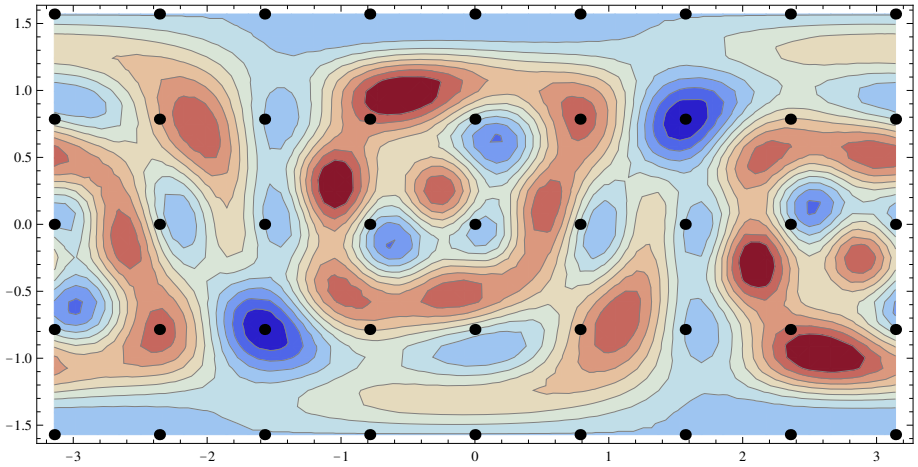


Figure 6.3: Minimization landscape when minimizing the distance d of a triclinic $\mathbb{K}$ to the monoclinic symmetry class, with γ fixed and d depending only on α (horizontal axes) and β (vertical axes). The local minima are colored in blue, local maxima are colored in red. One can see the poles ($\beta = \pm\pi/2$) and a grid (black points) of the initial values for the local minimization. The overall frequency of the extrema is $\pi/8$, but since they alternate, a spacing of $\pi/4$ is sufficient to detect all the local minima. One can see the presence of a two-fold global minimum.

inski (2011). To reduce the numerical effort, we apply these simplifications, detailed in the supplementary material.

Further, it is a well known result from representation theory that at most a four-fold rotational symmetry can be distinguished in tetrads (Böhlke and Brüggemann, 2001). For example, hexagonal materials and quasi-crystals with a five-fold symmetry are not distinguishable from transversal isotropy in linear elasticity, but in case of higher order theories, like strain gradient elasticity (Olive and Auffray, 2013, 2014). Therefore, the frequencies at which the distance measure can oscillate over the Euler angles is at most $2\pi/4/4$, taking into account the potential fourfold symmetry of $\mathbb{K}$ and of the symmetry group. For our practical purpose, this means that a fixed discretization of a starting point grid is sufficient to guarantee the finding of the global minimum. This allows for an efficient and robust implementation. We employed a gradient based downhill search method from a conservative starting point grid of $8\times4\times8$ points in the space of Euler angles, with a spacing of $\pi/8$. It is clear that the discretization of rotations in terms of Euler angles is not optimal. The equidistant Euler angle grid results in cluster points in $\mathcal{O}rth^+$. To optimize the numerical effort, one may use starting points that correspond to a fair discretization of $\mathcal{O}rth^+$, see, e.g. Nawratil and Pottmann (2008). However, in our case, the additional effort is not justified, since we have a small grid of starting points and apply the distance minimization only to a few stiffness tetrads.

6.7 Example applications

In the following subsections, we present the application of this method to determine the distance of the stiffness tetrads. The following figures show the distance d of the stiffness tetrad of the undeformed material $\mathbb{K}_0$ (dark grey) and the stiffness tetrad of the deformed material after the different tests pushed forward to the current stress-free placement $\boldsymbol{F} * \mathbb{K}_1$ (light grey) from the symmetry groups. The shape of the undeformed and deformed material and its associated coordinate system is illustrated on the right hand side of each figure.

6.7.1 Results for the uni-directionally reinforced material

We know that the uni-directional reinforcement results in an hexagonal symmetry of the initial stiffness tetrad $\mathbb{K}_0$. We find this result for the dark grey bars of $\mathbb{K}_0$ with $d = 0\%$ in all the resulting figures of this subsection. There is no distance to the hexagonal symmetry group and to all other symmetry groups with less symmetry namely trigonal, tetragonal, orthotropic and monoclinic.

Transverse fiber transport

We see the situation after a shear test with $H_{\mathrm{xy}} = 0.5$ in Fig. 6.4. This deformation causes a transport of the fibres relatively to each other. The fiber axis orientation remains unchanged. For the cubic symmetry group, we find a distance of $d = 31\%$ and the distance to the isotropic symmetry group is $d = 33\%$. After the plastic deformation, the stiffness tetrad has changed to $\mathbb{K}_1$. Following the results, it turns out that the symmetry of the deformed material is almost orthotropic. The reason is the shear number of $H_{\mathrm{xy}} = 0.5$ which leads to a fiber arrangement close to one above the other. The distance to the tetragonal symmetry is already $d = 5\%$ and to the initial hexagonal symmetry of the undeformed material $d = 8\%$. The same is true for a shear deformation with $H_{\mathrm{yx}} = 0.5$.

Fiber inclination

Fig. 6.5 shows the results after the shear test with $H_{\mathrm{xz}} = 0.5$ (also true for $H_{\mathrm{yz}} = 0.5$) which leads to the change of the fiber axes orientation. During the deformation, the symmetry remains nearly the same, namely hexagonal. The negligible distances to the lower symmetry groups result from the fact that the inter-fiber distance is slightly decreased due to the fiber inclination.

Parallel fiber transport

In Fig. 6.6, we find exactly no change regarding the symmetry because the shear $H_{zx} = 0.5$ test itself caused only a parallel shift of the fibres. There is no transport of the fibers and no change of the axes. This is again true for the shear deformation $H_{zy} = 0.5$.

Elongation tests

In case of a tension test with $H_{xx} = 0.5$ as depicted in Fig. 6.7, the material symmetry becomes tetragonal with a distance of less than 5%. For the trigonal and the initial hexagonal group, the distance results in about 5%. The same is true for a tension test with $H_{yy} = 0.5$ because the hexagonal fiber arrangement is destroyed during these deformations.

For a tension test in fiber direction, $H_{zz} = 0.5$, there is exactly no change of the material symmetry because the fibers are only elongated along their axes.

Mixed mode test

The results for the last test calculation (Fig. 6.9) with a shear deformation of $H_{xy} = H_{xz} = 0.5$ and a uniaxial tension of $H_{xx} = 0.5$ shows a similar behavior of the symmetry change compared to the first test case with $H_{xy} = 0.5$ because the shear in this direction leads to the biggest symmetry change. After the deformation, there is a distance of around 5% to the trigonal and the hexagonal symmetry. But we find that the material can be considered as tetragonal with a distance of only 2%.

To summarize, it turns out that the plastic deformation of the uniaxial deformed material leads to a change of the symmetry of the stiffness tetrad. The initial stiffness tetrad has the hexagonal symmetry. In all cases, the symmetry of the stiffness tetrad $\mathbb{K}_1$ after the deformation is at least orthotropic. The distance to the tetragonal, trigonal and hexagonal symmetry groups remains in the range between 0% and 8%.

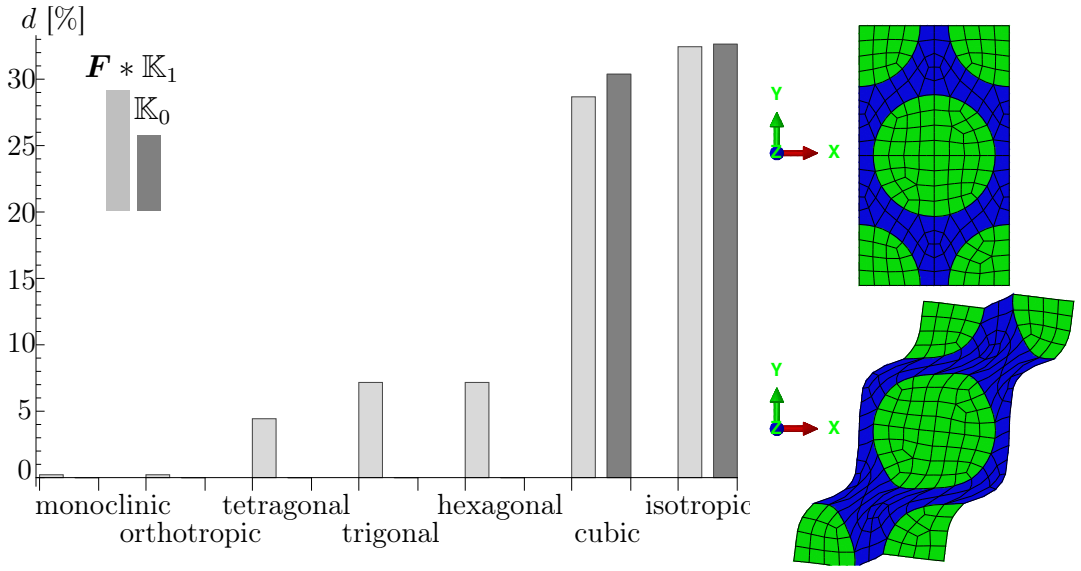


Figure 6.4: Distance d to the symmetry groups after the deformation $H_{\mathrm{xy}} = 0.5$

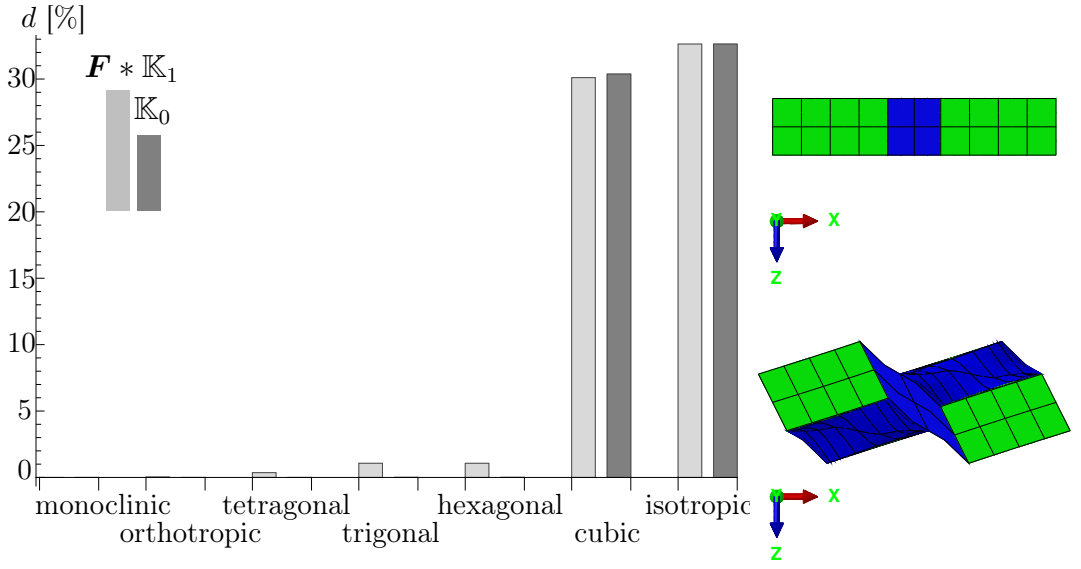


Figure 6.5: Distance d to the symmetry groups after the deformation $H_{\mathrm{xz}} = 0.5$

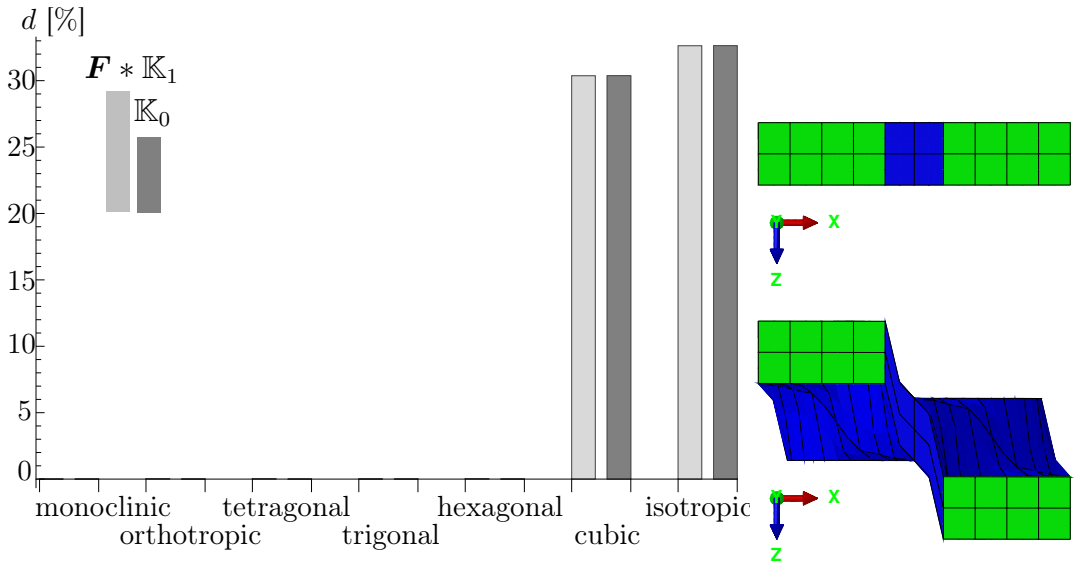


Figure 6.6: Distance d to the symmetry groups after the deformation $H_{zx} = 0.5$

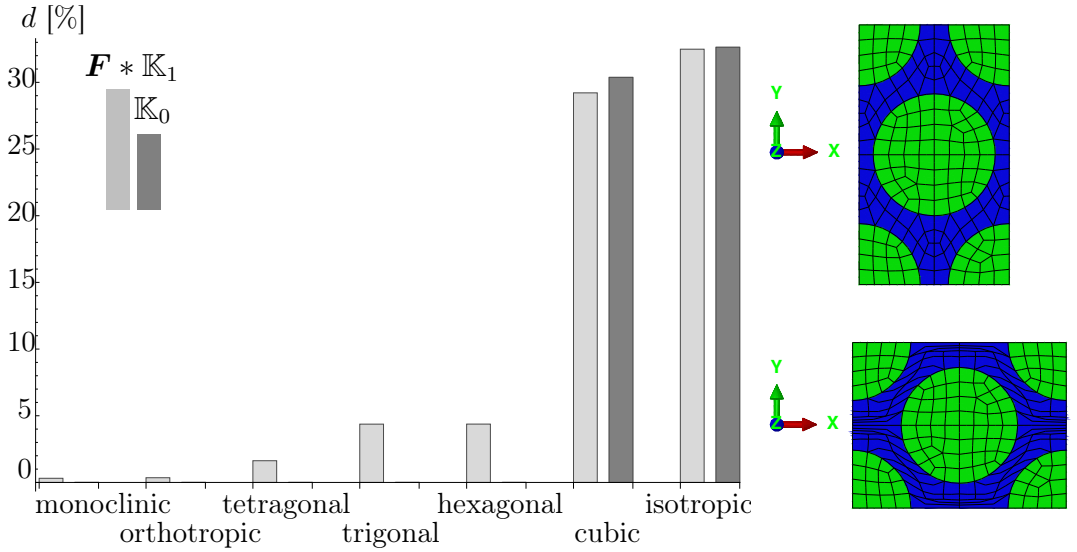


Figure 6.7: Distance d to the symmetry groups after the deformation $H_{xx} = 0.5$

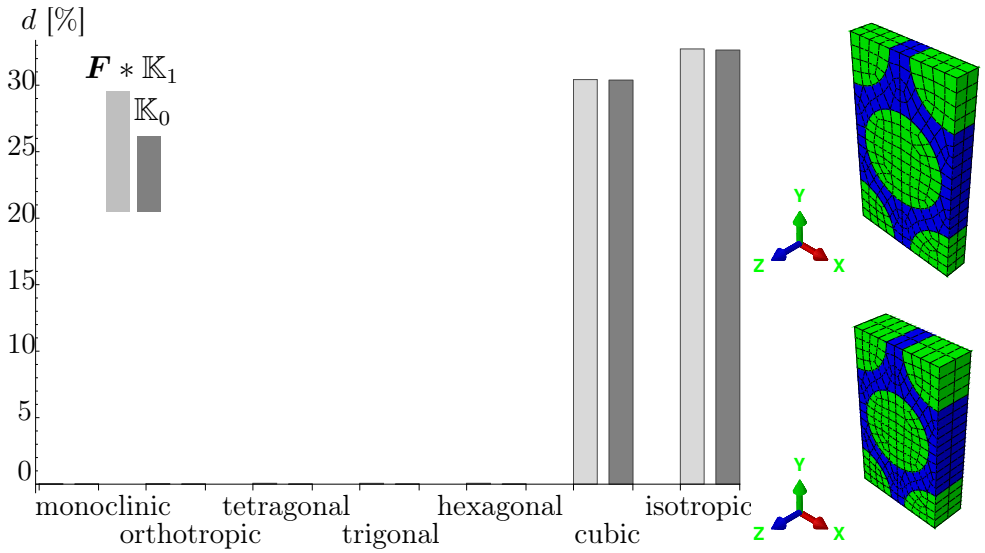


Figure 6.8: Distance d to the symmetry groups after the deformation $H_{zz} = 0.5$

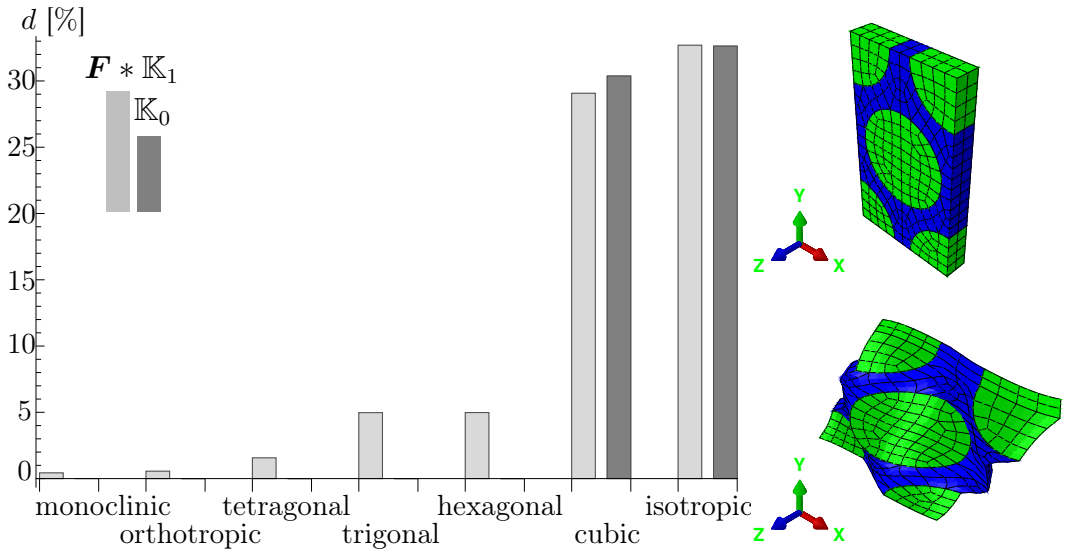


Figure 6.9: Distance d to the symmetry groups after the deformation $H_{xy} = H_{xz} = 0.5$ and $H_{xx} = 0.5$

6.7.2 Results for the bi-directionally reinforced material

The undeformed bi-directionally reinforced material $\mathbb{K}_0$ has the tetragonal symmetry (see Fig. 6.10).

Shear in the fiber plane

After the shear test $H_{\mathrm{xy}} = 0.5$, the material can be considered as orthotropic because the distance d for $\mathbb{K}_1$ is 15,5% to the tetragonal symmetry but less than 1% to the orthotropic symmetry group. Similar results are obtained by the shear deformation $H_{\mathrm{yx}} = 0.5$. The reason for the large distance to the initial tetragonal symmtry is the change of the angle between the fibers during the deformation.

Shear parallel to the fibers

In Fig. 6.11, we find that the material nearly remains tetragonal with a distance of less than 3% after the deformation $H_{\mathrm{xz}} = 0.5$ (also true for the deformation $H_{\mathrm{yz}} = 0.5$). This is because the angle between the fibers does not change during this shear deformation in contrast to the foregoing deformation.

Fiber plane inclination

For the deformations $H_{\mathrm{zx}} = 0.5$ (see Fig. 6.12) and $H_{\mathrm{zy}} = 0.5$, the distance to the initial tetragonal symmetry group also remains in the range of less than 3% which again results from the unchanged angle between the fibers. The results differ slightly from the foregoing case since the planes spanned by the fibers come closer together in the present case. Additionally, one fiber gets stretched, while the other is only displaced transversally.

Elongation tests

In Fig. 6.13, the material symmetry has to be considered as trigonal after the elongation $H_{\mathrm{xx}} = 0.5$ because the distance to the tetragonal group is nearly 5%. The reason is that after the deformation the spacing between the fibers in y direction is larger than the spacing between the fibers in x direction. The same is true for an elongation test in y direction.

Only in the case of an elongation test in z direction as depicted in Fig. 6.14, the material symmetry remains unchanged after the deformation because the fiber spacing changes simultaneously and remains the same in both x and y directions.

Mixed mode test

The deformation with shear $H_{xy} = H_{xz} = 0.5$ and uniaxial tension $H_{xx} = 0.5$ in Fig. 6.15 results in an orthotropic material. The distance to the tetragonal symmetry group is about 15%.

Summarizing, the initial elastic behavior of the bi-directionally reinforced material is tetragonal. The material remains at least orthotropic after different deformations.

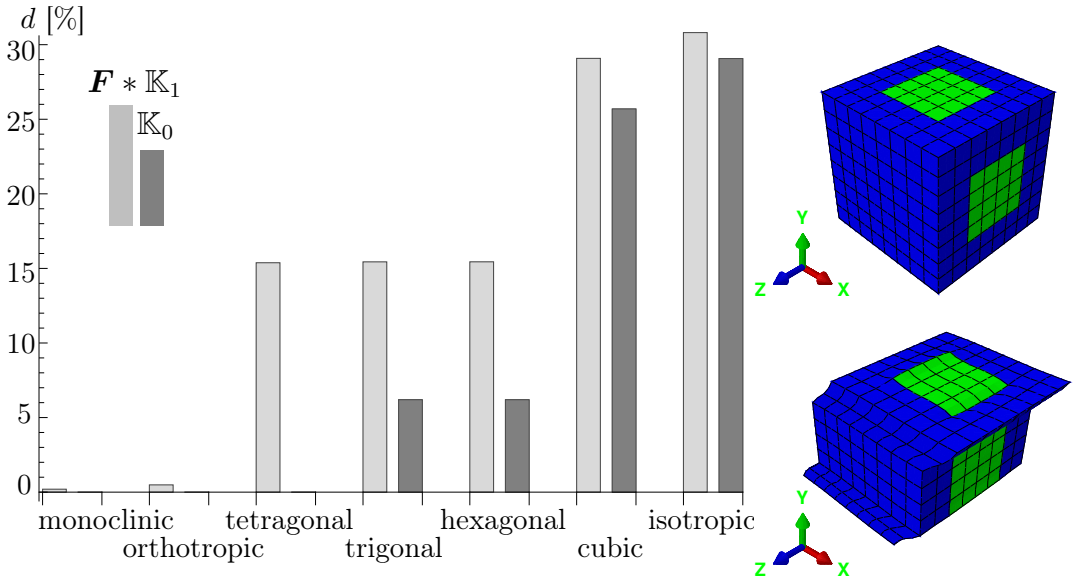


Figure 6.10: Distance d to the symmetry groups after the deformation $H_{xy} = 0.5$

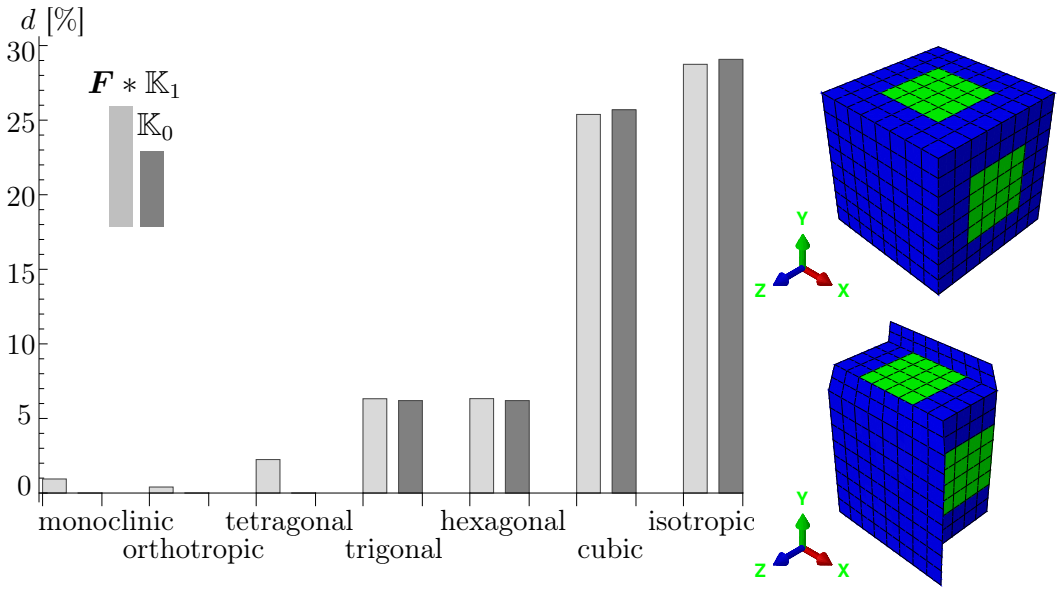


Figure 6.11: Distance d to the symmetry groups after the deformation $H_{\mathrm{xz}} = 0.5$

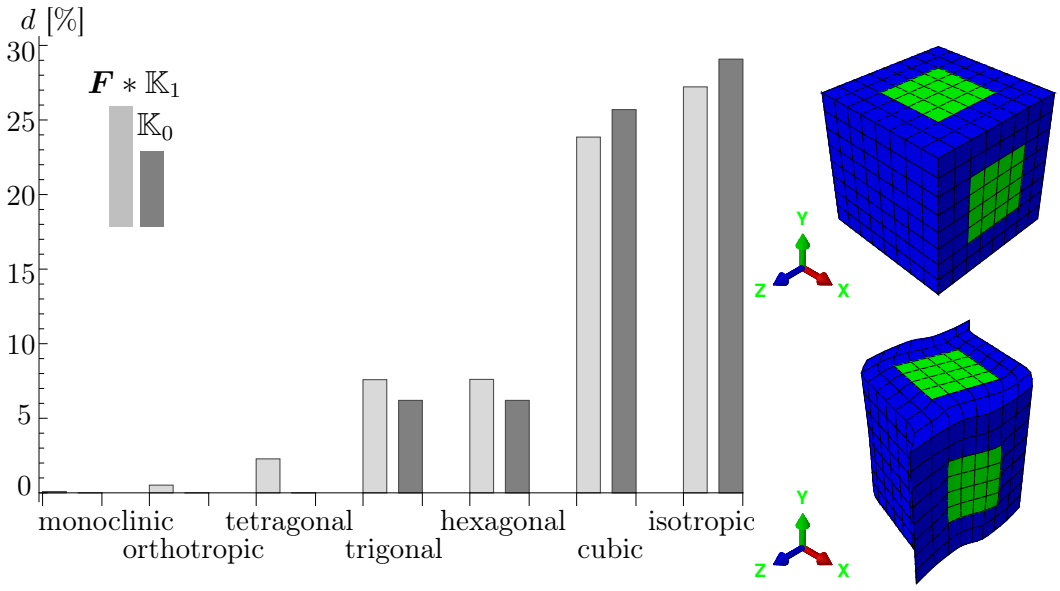


Figure 6.12: Distance d to the symmetry groups after the deformation $H_{\mathrm{zx}} = 0.5$

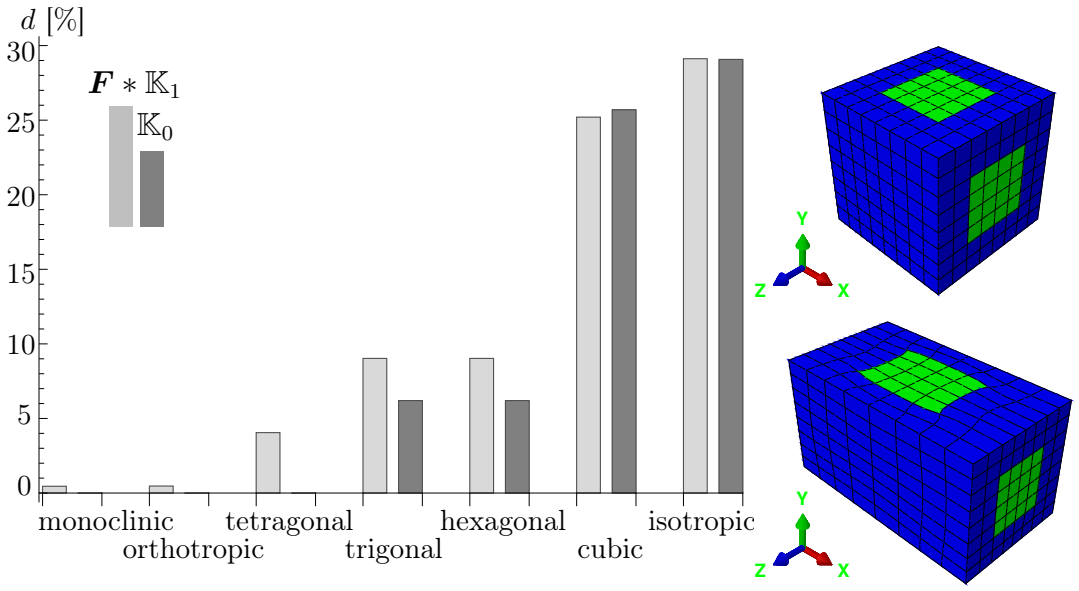


Figure 6.13: Distance d to the symmetry groups after the deformation $\gamma_{xx} = 0.5$

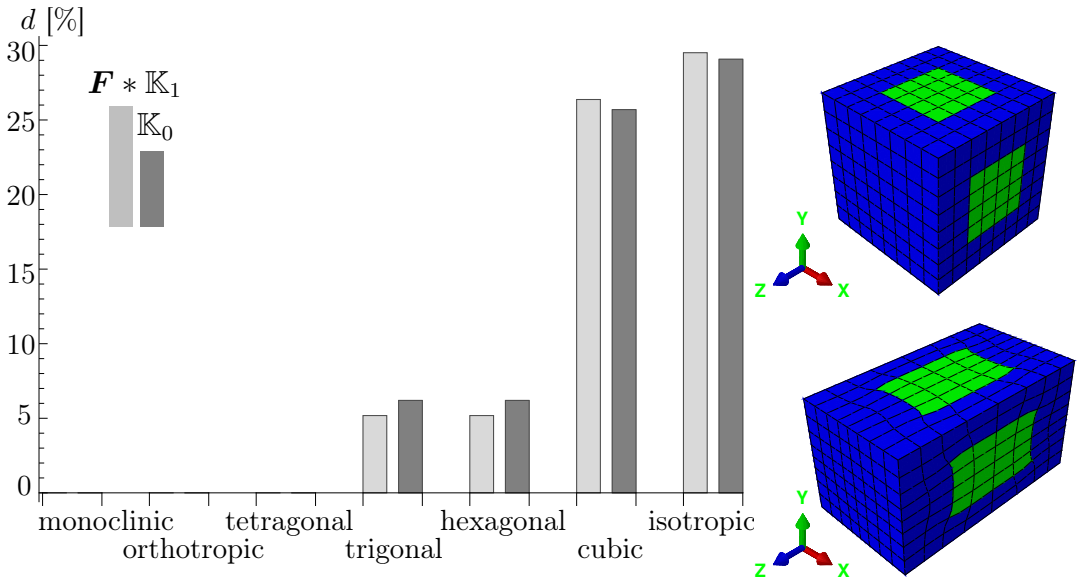


Figure 6.14: Distance d to the symmetry groups after the deformation $\gamma_{zz} = 0.5$

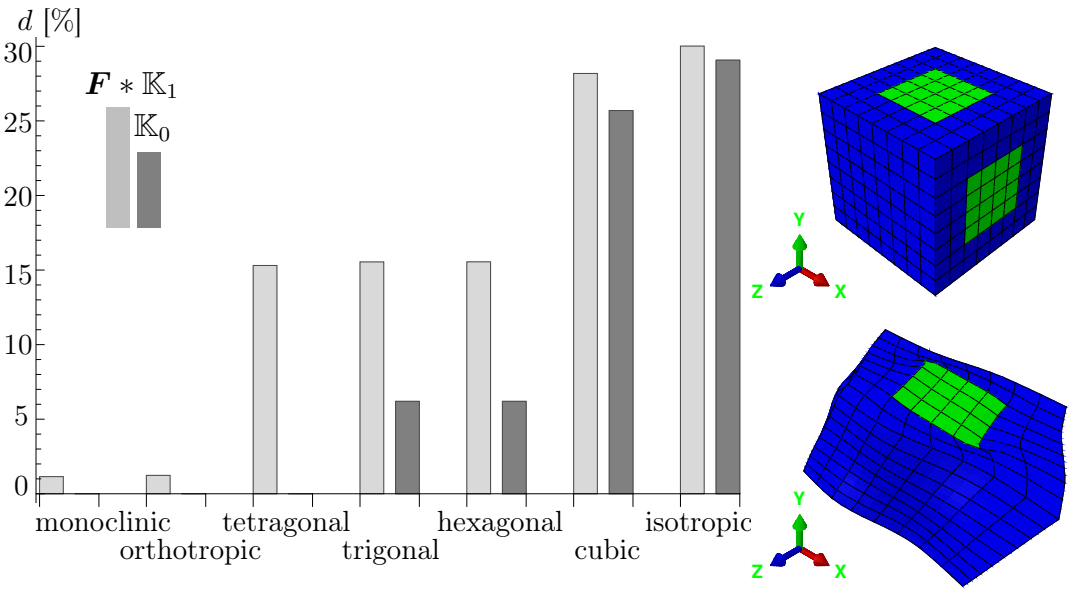


Figure 6.15: Distance d to the symmetry groups after the deformation $H_{\mathrm{xy}} = H_{\mathrm{xz}} = 0.5$ and $\epsilon_{xx} = 0.5$

6.7.3 Results for the tri-directionally reinforced material

The initial stiffness tetrad $\mathbb{K}_0$ of the tri-directionally reinforced material has a cubic symmetry. We find a distance of about 7% to the hexagonal and about 12% to the isotropic symmetry group for the dark grey bars of $\mathbb{K}_0$ in the following three Figures 6.16 to 6.18. The trigonal symmetry is included, as the three-fold rotation around the cube's space diagonal is a symmetry operation.

Shear tests

Fig. 6.16 shows that after a shear test $H_{\mathrm{xy}} = 0.5$ (true for all other shear tests with shear direction and shear plane normal parallel to the fiber axes) the material becomes orthotropic with a distance of only 1% to this group. The distances are around 10% to all other higher symmetry groups including the initial cubic one.

Elongation tests

By elongating the material parallel to a fiber (example depicted in Fig. 6.17) with $H_{\mathrm{xx}} = 0.5$, the distance to the cubic symmetry group increases but remains below 5%. The same is true for the trigonal and tetragonal symmetry groups. The reason is that the fiber spacing is not the same in all three directions after the deformation.

Mixed mode test

The deformation with shear $H_{xy} = H_{xz} = 0.5$ and uniaxial tension $H_{xx} = 0.5$ in Fig. 6.18 results in a triclinic material because the distance to all material symmetry groups is clearly larger than 5%.

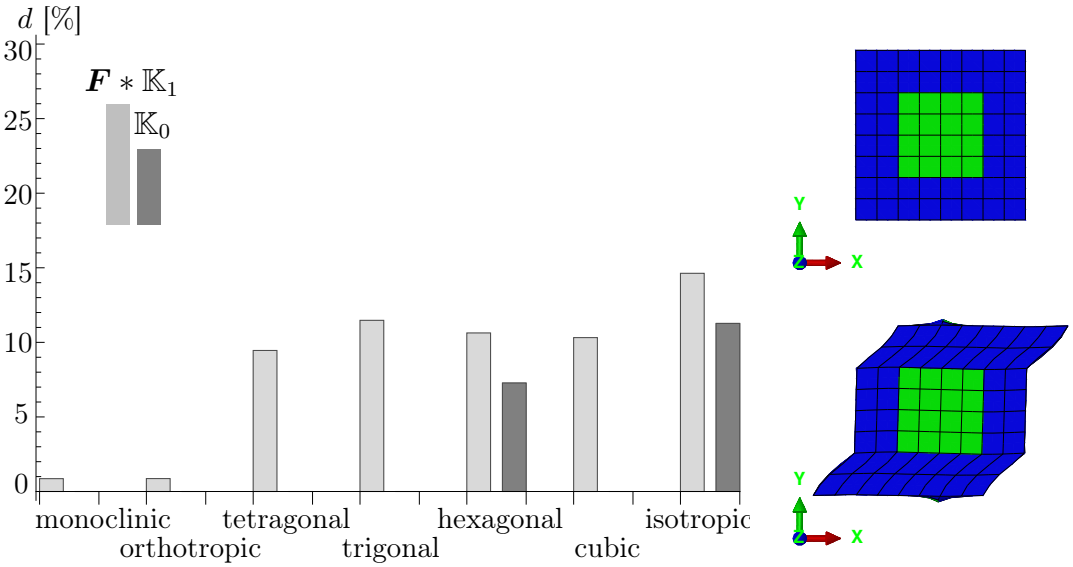


Figure 6.16: Distance d to the symmetry groups after the deformation $H_{xy} = 0.5$

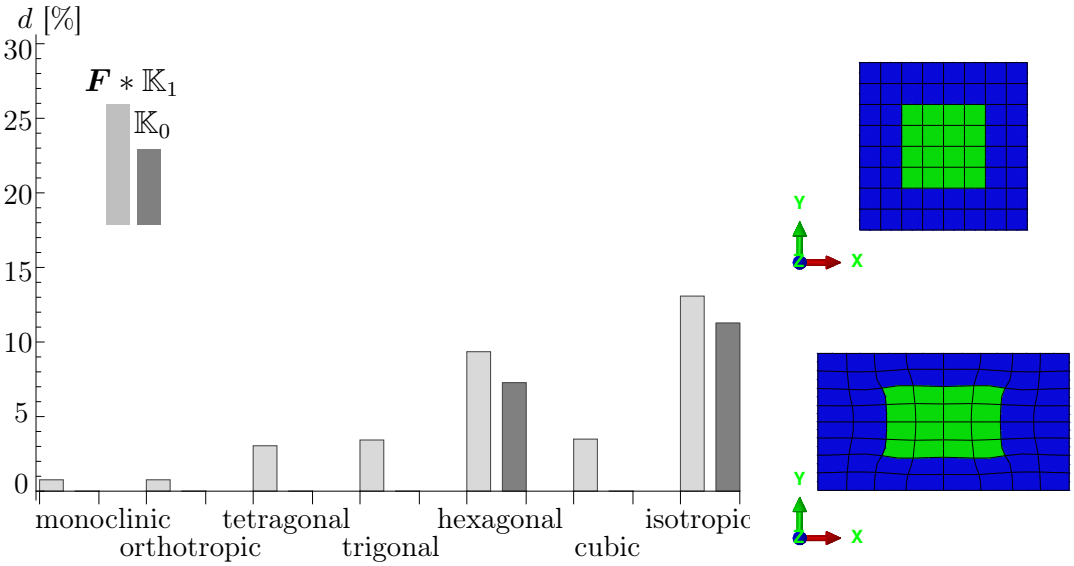


Figure 6.17: Distance d to the symmetry groups after the deformation $H_{xx} = 0.5$

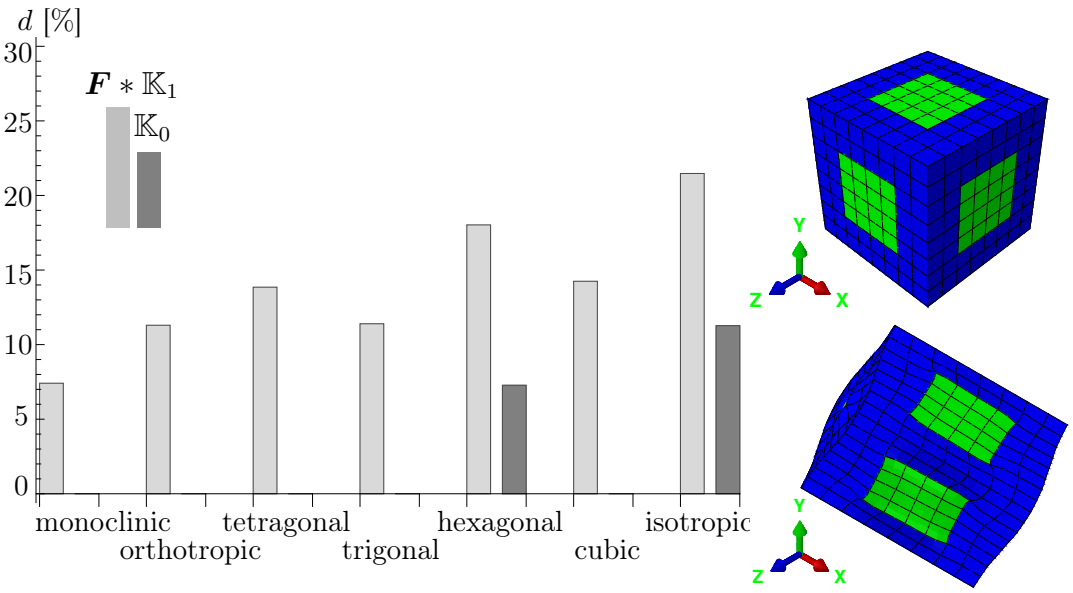


Figure 6.18: Distance d to the symmetry groups after the deformation $H_{xy} = H_{xz} = 0.5$ and $H_{xx} = 0.5$

6.8 Summary

In this chapter, we present a fast algorithm to determine the distance of a measured stiffness tetrad to all possible symmetries of an elastic stiffness tetrad. Using a projection method, we build up an 8^{th} order tensor covering all group elements of the chosen group. Applying this projector to a measured stiffness tetrad, we only have to minimize over the orientation of the symmetry axis and therefore only over the three Euler angles. Using this method, we investigated the uni-, bi- and tri-directionally reinforced material.

In all cases, the distance measures corresponded well to the apparent symmetry classes. For example, all bi-directionally reinforced materials (Subsection 6.7.2) are very close to the orthotropic symmetry group. The deviations are due to inhomogeneous deformations inside the RVE, while the general closeness to the orthotropic symmetry group is due to the fact that in any bi-directional material with equal fiber properties, three orthogonal directions can be found, namely the fiber plane normal and the two angle bisectors. Other tests of plausibility, like the presence of trigonal symmetry around the 111-direction in an orthogonal tri-directional fiber arrangement in the (100) directions are captured as well (Subsection 6.7.3).

The proposed methodology relies on the robust determination of the global minimum of the distance measure over the orientation in terms of Euler angles. Since there are local minima, a simple gradient approach starting from one point is not sufficient, and precautions must be taken to ensure the finding of the global minimum. However, this is not hard. As the Euler angle space is periodic and may be further

reduced by invoking symmetries from the groups to which the stiffness is projected, it is sufficient to consider a small minimization domain. This is confirmed by the fact that we never observed results that are contradictory to the group inclusion schemes, i.e. the distance to the tetragonal group was always smaller or equal to the distance to the cubic group, which makes us confident that the presented scheme is well applicable in an algorithmic manner.

7 Modeling on the macro-scale

In Chapter 5, we presented a method to determine the stiffness tetrads before and after large plastic strains. The results show that the stiffness tetrad changes significantly, and that this change cannot be predicted by the plastic transformation $\boldsymbol{P}_\mathrm{C}$ of the isomorphy concept. Therefore, we decide to use the theory of Material Plasticity (see Chapter 2, Eq. (2.25))

$$\overset{\text{2PK}}{\boldsymbol{T}} = \frac{1}{2}\left(\boldsymbol{P}_\mathrm{C}\boldsymbol{P}_\mathrm{K}\right) * \mathbb{K} : \left(\boldsymbol{C} - \boldsymbol{P}_\mathrm{C}^{-\mathrm{T}}\boldsymbol{P}_\mathrm{C}^{-1}\right) \tag{7.1}$$

using two different plastic transformations $\boldsymbol{P}_\mathrm{C}$ and $\boldsymbol{P}_\mathrm{K}$. When we think of the form of the evolution laws for $\boldsymbol{P}_\mathrm{K}$ and $\boldsymbol{P}_\mathrm{C}$, we need to distinguish the scales. On the micro-scale in the RVE, we locally use the von Mises flow rule, which is thermodynamically consistent. On the macro-scale, the empirical evolution of $\boldsymbol{P}_\mathrm{K}$ alone is not enough to assess the thermodynamic consistency. Due to $\boldsymbol{P}_\mathrm{C} = \boldsymbol{F}^{-1}$, i.e. $\boldsymbol{F}_\mathrm{e} = \boldsymbol{I}$, we have a rigid plastic process in which, somewhat paradoxically, we track the evolution of the stiffness indirectly by tracking the evolution of $\boldsymbol{P}_\mathrm{K}$. In the present form, the approach is only applicable when the strain process is given. This may be enough for estimating the stiffness evolution in a forming process. To use the approach in a boundary value problem, we have to drop the simplification $\boldsymbol{P}_\mathrm{C} = \boldsymbol{F}^{-1}$ to be able to calculate stresses and search the equilibrium. In that case, we need to complete the set of equations by adding a flow rule for the evolution of $\boldsymbol{P}_\mathrm{C}$, for example by flow in direction of maximum plastic dissipation. Then thermodynamic consistency may be checked. On a side note, we believe that the approach is thermodynamically safe for reasonable flow rules. In any case, errors due to the potential violation of thermodynamic consistency are probably much smaller than errors due to the overall simplicity of the approach.

7.1 Evolution of the transformation $\boldsymbol{P}_\mathrm{C}$

Following our assumptions, we have only small elastic strains. This directly leads us to the major simplification that

$$\boldsymbol{P}_\mathrm{C} \approx \boldsymbol{F}^{-1} \ . \tag{7.2}$$

This simplification has already been underpinned by the results of Chapter 5. The difference between $\boldsymbol{P}_{\mathrm{C}}$ and $\boldsymbol{F}^{-1}$ is very small and a result of the difference of order of magnitude between the yield stress σ_{F} and the Young's modulus E in the two phases.

7.2 Finding a suitable transformation for $\boldsymbol{P}_{\mathrm{K}}$

The aim is to finally develop an evolution equation for the transformation $\boldsymbol{P}_{\mathrm{K}}$ of the stiffness tetrad $\mathbb{K}$. To find this evolution equation, we want to study different possibilities. The first and one of the easiest approaches will be to choose $\boldsymbol{P}_{\mathrm{K}} = \boldsymbol{P}_{\mathrm{C}}{}^{-1}$. This would mean that the stiffness tetrad does not transform during a plastic transformation in contrast to the unloaded placement which will transform accordingly to $\boldsymbol{P}_{\mathrm{C}}$. The second choice will be $\boldsymbol{P}_{\mathrm{K}} = \boldsymbol{I}$. This is the known isomorphy case and we already know that this approach will fail. For small plastic transformations, we can expect only a slight deviation while for large plastic deformations it will increase. The third possibility, we want to study, is to numerically fit a second-order tensor $\boldsymbol{P}$ modifying the known transformation $\boldsymbol{P}_{\mathrm{C}}$. We will use two different sets of non-zero components. The numerically best fit is calculated with a minimization procedure in Mathematica. In the first case, we choose to take all possible nine components of a second-order tensor and call it $\boldsymbol{P}_{\mathrm{K}} = \boldsymbol{P}_{9}^{\mathrm{num}}$ with

$$\boldsymbol{P}_{9}^{\mathrm{num}} = \begin{bmatrix} P_{xx} & P_{xy} & P_{xz} \\ P_{yx} & P_{yy} & P_{yz} \\ P_{zx} & P_{zy} & P_{zz} \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j \ . \tag{7.3}$$

If the scalar contraction of $\boldsymbol{P}_{\mathrm{C}}$ with a second-order tensor is able to track the evolution of the stiffness tetrad, we will get the exact results for the numerical fit of this choice. The second choice is to consider only the main diagonal components and call this transformation $\boldsymbol{P}_{\mathrm{K}} = \boldsymbol{P}_{3}^{\mathrm{num}}$

$$\boldsymbol{P}_{3}^{\mathrm{num}} = \begin{bmatrix} P_{xx} & 0 & 0 \\ 0 & P_{yy} & 0 \\ 0 & 0 & P_{zz} \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j \ . \tag{7.4}$$

This means, we limit the field of application to materials having their anisotropic axes perpendicular to each other, where the directions of anisotropy are along $\boldsymbol{e}_i$. The main point of this idea is that $\boldsymbol{P}_{\mathrm{K}}$ transforms the original stiffness tetrad $\mathbb{K}_0$ in the reference placement. Only afterwards, the transformation $\boldsymbol{P}_{\mathrm{C}}$ is applied. To compare the different choices, we choose the tri-directionally reinforced material as example material. The material properties are the same as in the foregoing sections. We perform a shear test, a tensile test and the combined test of two shears and one tension and compare the deviation of the initial stiffness $\mathbb{K}_0$ and the stiffness $\mathbb{K}_1$ after

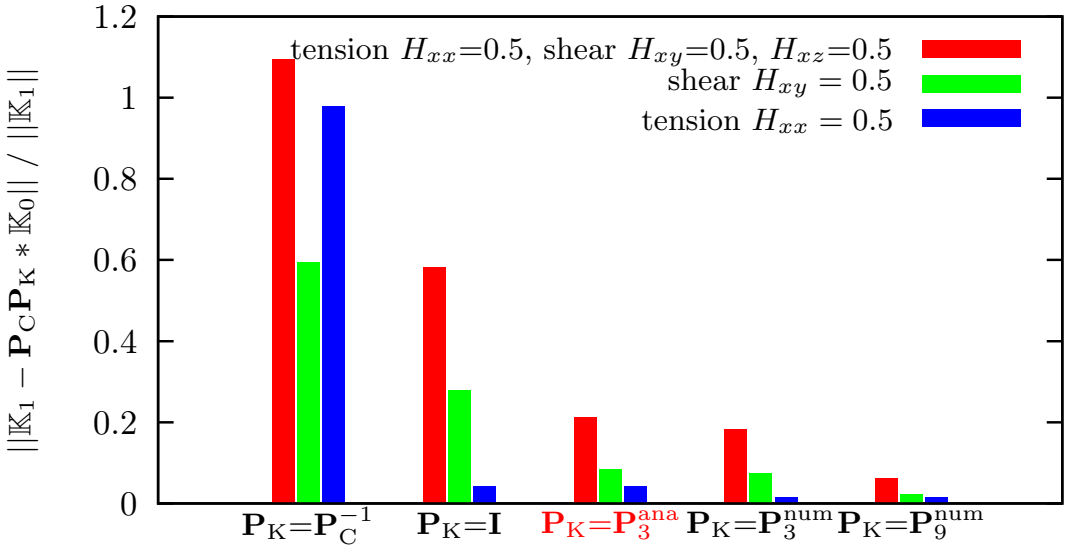


Figure 7.1: Deviation of the measured stiffness tetrad from the respective modeling approach for the tri-directional reinforcement. The choice $\boldsymbol{P}_K = \boldsymbol{P}_C{}^{-1}$ is clearly the worst, followed by elastic isomorphy $\boldsymbol{P}_K = \boldsymbol{I}$. With $\boldsymbol{P}_3^{num}$ having three degrees of freedom a significant improvement is reached. The improvement using $\boldsymbol{P}_9^{num}$ with 9 free values is not much better. The analytical approach $\boldsymbol{P}_3^{ana}$ is almost as good as the corresponding numerical fit $\boldsymbol{P}_3^{num}$ with 3 parameters.

the deformation (see Chapter 5). The deviation of the measured stiffness tetrad from the respective modeling approach for the tri-directional reinforcement is depicted in Fig. 7.1. The choice $\boldsymbol{P}_K = \boldsymbol{P}_C{}^{-1}$ is clearly the worst. There is no agreement if we do not transform the initial stiffness tetrad after such large deformations. With the approach of elastic isomorphy $\boldsymbol{P}_K = \boldsymbol{I}$, we find still a large deviation for the shear test and the combined test. In this case, the anisotropy of the stiffness tetrad is changed during the deformation. During the tensile test, the choice of $\boldsymbol{P}_K = \boldsymbol{I}$ works because during this deformation the symmetry of the material is not changed. Using the transformation $\boldsymbol{P}_K = \boldsymbol{P}_{K3}$ with the three degrees of freedom, a significant improvement is reached. The improvement using $\boldsymbol{P}_9^{num}$ is not much better. The main results of this section are: Firstly, it turns out that it is possible to approximate the evolution of the stiffness tetrad with sufficient accuracy using one second-order tensor. Secondly, we find the best compromise between accuracy and effort with $\boldsymbol{P}_K$=$\boldsymbol{P}_3^{num}$.

7.3 Evolution of the transformation $\boldsymbol{P}_{\mathsf{K}}$

To derive an analytical evolution equation for $\dot{\boldsymbol{P}}_3^{\text{ana}}$, we use the pull-back $\boldsymbol{L}^{\text{ref}} = \boldsymbol{F}^{-1}\boldsymbol{L}\boldsymbol{F}$ of the velocity gradient $\boldsymbol{L} = \dot{\boldsymbol{F}}\boldsymbol{F}^{-1}$, motivated by the fact that the fiber directions are constant in the reference placement

$$\boldsymbol{L}^{\text{ref}} = \boldsymbol{F}^{-1}\dot{\boldsymbol{F}} \ . \tag{7.5}$$

There is no direct physical interpretation for the pull-back of the velocity gradient, just like there is no direct physical interpretation for the reference placement or any other quantity that is pulled back. Nevertheless, they are mathematical auxiliary quantities that may be used. The main purpose of such pulled back material quantities is to formulate constitutive laws in accordance with the principles of material modeling, specifically the principle of invariance under superimposed rigid body motions. For this reason, the St. Venant-Kirchhoff law is used, with $\mathbb{K}_0$ defined in the reference placement. Consequently, when modifying the elastic law, we need to do it in the reference placement. But, of course, the actual deformation determines its evolution. Hence, we need to realize some kind of pull-back of a kinematic variable to model the evolution of $\mathbb{K}_0$. We could also have used a push-forward of the elasticity law where the evolution of $\boldsymbol{F} * \mathbb{K}_0$ is tracked, but then we would mix the change of $\mathbb{K}$ due to rigid rotations with the change due to the deformation. It is precise for this separation of effects that we describe the evolution of $\mathbb{K}_0$ in the reference placement.

Since we want to make the approach as simple as possible, it should be linear. In Chapter 5, we have seen that shear has the greatest influence on the anisotropy change. Looking at a single fiber, there is always transverse isotropy. Hence the shear directions in the cross-sectional plane are equal, we determine the shear rate by the theorem of Pythagoras to get the amount of the shear rate. To derive the evolution of the three main diagonal entries of

$$\dot{\boldsymbol{P}}_3^{\text{ana}} = \begin{bmatrix} \dot{P}_{xx} & 0 & 0 \\ 0 & \dot{P}_{yy} & 0 \\ 0 & 0 & \dot{P}_{zz} \end{bmatrix} \boldsymbol{e}_i \otimes \boldsymbol{e}_j \ , \tag{7.6}$$

we take the Euclidean norm of the corresponding components of $\boldsymbol{L}^{\text{ref}}$ following Eq. (7.7)

$$\dot{P}_{ii} = k_i \sqrt{{L^{\text{ref}}_{i+1\,i}}^2 + {L^{\text{ref}}_{i+2\,i}}^2} \ . \tag{7.7}$$

This gives basically the magnitude of the shear rate in the plane with normal vector $\boldsymbol{e}_i$. The simulations have shown that this is the most dominant contribution to the evolution of the stiffness tetrad, see Section 5.5, Eq. (5.15). It implies that

the inclination of the fiber due to shear is the same in any shear direction. The coefficients k_i depend on the material parameters and on the volume fractions of the fibers in direction i ($k_i = 0$ if there is no fiber in direction i). The indices $i+1$ and $i+2$ are taken modulo 3 which results in

$$\dot{P}_{xx} = k_1 \sqrt{L_{yx}^{\text{ref}\ 2} + L_{zx}^{\text{ref}\ 2}} \tag{7.8}$$

$$\dot{P}_{yy} = k_2 \sqrt{L_{zy}^{\text{ref}\ 2} + L_{xy}^{\text{ref}\ 2}} \tag{7.9}$$

$$\dot{P}_{zz} = k_3 \sqrt{L_{xz}^{\text{ref}\ 2} + L_{yz}^{\text{ref}\ 2}}\ . \tag{7.10}$$

We have to determine the coefficients for the chosen material once. Therefore, we use RVE calculations for the three characteristic shear tests H_{yx} (k_1), H_{xy} (k_2) and H_{xz} (k_3) and take the i^{th} entry in the numerically best $\boldsymbol{P}_3^{\text{num}}$ for k_i. We verified the analytical evolution equation of $\boldsymbol{P}_{\text{K}}$ by using the new strain path of the mixed test H_{xy}, H_{xz} and H_{xx}. Afterwards we used these constants to calculate all other tests. In Fig. 7.1, we finally show the results of the analytically calculated $\mathbf{P}_{\text{K}}$=$\mathbf{P}_3^{\text{ana}}$. It turns out that the analytical solution is very close to the numerical best $\boldsymbol{P}_3^{\text{num}}$. This is a highly satisfying result because the assumptions and simplifications we made were extensive. First, we assumed that a second-order tensor connected with the Rayleigh product to the fourth-order stiffness tetrad is enough to capture the evolution of the stiffness tetrad during a symmetry changing deformation. Secondly, we reduced the number of free components within the second-order tensor from nine to three in order to be able to find an analytical evolution equation which is easy to access. Our approach clearly gives better results than the known isomorphy concept and the multiplicative decomposition where the material stiffness transforms accordingly to the stress-free placement. In Table 7.1, we provide the results of all three different materials and ten different tests.

7.4 Evolution of the values and eigenvalues of the stiffness tetrad

Finally, we like to represent and discuss the evolution of the stiffness tetrad during the transformation with the chosen second-order tensor $\boldsymbol{P}_3^{\text{ana}}$. For this purpose, we use the shear test H_{xz} on the uni-directional reinforced material as depicted in Fig. 7.2. We will analyze the 21 different components of the stiffness tetrad itself and additionally the six Eigenvalues of the stiffness tetrad. In Fig. 7.3, we see the change of the components during the shear test. Additionally, we show the index arrangement of the Voigt notation for a stiffness tetrad $\mathbb{K}$, the absolute values of $\mathbb{K}_0$ and $\mathbb{K}_1$ in Fig. 7.4.

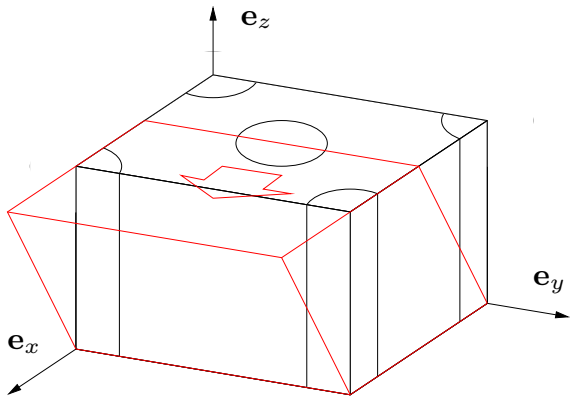


Figure 7.2: Shear test $H_{xz} = 0.5$ on the uni-directional reinforced material

Because the shear test is in the 1-3 plane we see the largest change in the components "xxxx" and "zzzz" marked in dark red, in the components containing two times "x" and "z" namely "xxzz" and "xzxz" marked in red and in the two components "xxxz" and "zzxz" marked in light red. All other components containing at least one time the second index are marked in blue. In dark blue, we marked the components with a large change of nearly 100%. These components were zero in the initial stiffness $\mathbb{K}_0$ and changed to some big value in $\mathbb{K}_1$. The components are "yyxz" and "xyyz". The components with a medium change are "xxyy", "yyzz", "xyxy" and "yzyz" and are marked in blue. The component "yyyy" has an initial value but zero change and is marked in light blue. All values with a zero initial value and zero change are marked in black and are "xxxy", "yyxy", "zzxy", "xyxz", "xxyz", "yyyz", "zzyz" and "xzyz".

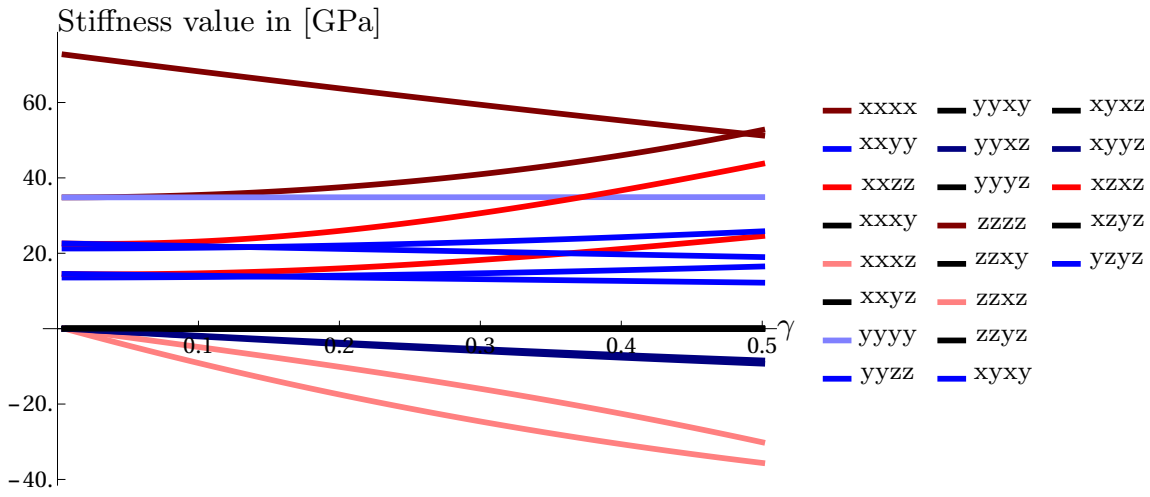


Figure 7.3: Evolution of the components of the stiffness tetrad of the uni-directional reinforced material during a shear test H_{xz}.

$$K_{ij} = \begin{pmatrix} \text{xxxx} & \text{xxyy} & \text{xxzz} & \text{xxxy} & \text{xxxz} & \text{xxyz} \\ & \text{yyyy} & \text{yyzz} & \text{yyxy} & \text{yyxz} & \text{yyyz} \\ & & \text{zzzz} & \text{zzxy} & \text{zzxz} & \text{zzyz} \\ & & & \text{xyxy} & \text{xyxz} & \text{xyyz} \\ & & & & \text{xzxz} & \text{xzyz} \\ \text{sym} & & & & & \text{yzyz} \end{pmatrix}$$

$$\mathbb{K}_0 = \begin{pmatrix} 35 & 14 & 15 & 0 & 0 & 0 \\ & 35 & 15 & 0 & 0 & 0 \\ & & 73 & 0 & 0 & 0 \\ & & & 21 & 0 & 0 \\ & & & & 23 & 0 \\ \text{sym} & & & & & 23 \end{pmatrix} \text{GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

$$\mathbb{K}_1 = \begin{pmatrix} 53 & 16 & 25 & 0 & -30 & 0 \\ & 35 & 12 & 0 & -8 & 0 \\ & & 51 & 0 & -36 & 0 \\ & & & 26 & 0 & -9 \\ & & & & 44 & 0 \\ \text{sym} & & & & & 19 \end{pmatrix} \text{GPa } \boldsymbol{B}_i \otimes \boldsymbol{B}_j$$

Figure 7.4: Matrices: Evolution of the components of the stiffness tetrad of the uni-directional reinforced material during a shear test H_{xz}.

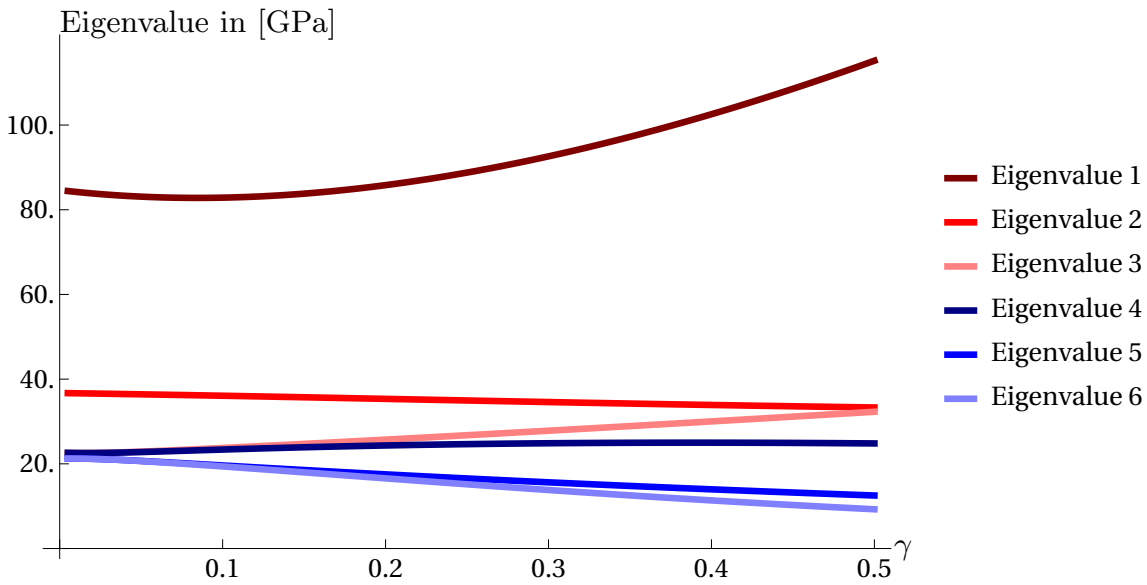


Figure 7.5: Evolution of the Eigenvalues of the stiffness tetrad of the uni-directional reinforced material during a shear test H_{xz}.

In Fig. 7.5, we additionally investigated the evolution of the Eigenvalues during the deformation. We sorted them in descending order from the largest one (dark red) to the smallest one (dark blue). Because the Eigenvalues are free of any rotation we clearly see that the stiffness tetrad evolves and the symmetry changes.

7.5 Summary

In this Chapter, we found that it is possible to predict the evolution of the stiffness tetrad during large deformations using one second-order tensor $\boldsymbol{P}_{\mathrm{K}}$ within the equation of our Material Plasticity theory:

$$\overset{\text{2PK}}{\boldsymbol{T}} = \frac{1}{2}\left(\boldsymbol{P}_{\mathrm{C}}\boldsymbol{P}_{\mathrm{K}}\right) * \mathbb{K} : \left(\boldsymbol{C} - \boldsymbol{P}_{\mathrm{C}}^{-\mathrm{T}}\boldsymbol{P}_{\mathrm{C}}^{-1}\right) \ . \tag{7.11}$$

We found an analytical evolution equation to predict this tensor $\boldsymbol{P}_{\mathrm{K}}$ with only three free variables. This is valid for a class of fiber reinforced materials having fiber angles of 90° with a sufficient accuracy.

Deformation	$\boldsymbol{P}_{\mathrm{K}}$	Uni-directional	Bi-directional	Tri-directional
tension H_{xx}	$\boldsymbol{P}_{\mathrm{C}}{}^{-1}$	133.0	91.8	97.9
	$\boldsymbol{I}$	5.3	4.5	2.5
	$\boldsymbol{P}_3^{\mathrm{ana}}$	5.3	4.5	2.5
	$\boldsymbol{P}_3^{\mathrm{num}}$	1.4	1.1	1.0
	$\boldsymbol{P}_9^{\mathrm{num}}$	1.4	1.1	0.9
tension H_{yy}	$\boldsymbol{P}_{\mathrm{C}}{}^{-1}$	99.5	91.8	97.9
	$\boldsymbol{I}$	3.21	4.5	2.5
	$\boldsymbol{P}_3^{\mathrm{ana}}$	3.2	4.5	2.5
	$\boldsymbol{P}_3^{\mathrm{num}}$	1.7	1.1	1.0
	$\boldsymbol{P}_9^{\mathrm{num}}$	1.7	1.1	0.9
tension H_{zz}	$\boldsymbol{P}_{\mathrm{C}}{}^{-1}$	90.6	115.4	97.9
	$\boldsymbol{I}$	0.1	2.1	2.5
	$\boldsymbol{P}_3^{\mathrm{ana}}$	0.1	2.1	2.5
	$\boldsymbol{P}_3^{\mathrm{num}}$	0.1	1.2	1.0
	$\boldsymbol{P}_9^{\mathrm{num}}$	0.1	1.2	0.9
shear H_{xy}	$\boldsymbol{P}_{\mathrm{C}}{}^{-1}$	67.3	68.0	59.6
	$\boldsymbol{I}$	7.7	23.6	22.6
	$\boldsymbol{P}_3^{\mathrm{ana}}$	6.2	7.5	6.8
	$\boldsymbol{P}_3^{\mathrm{num}}$	4.3	6.4	6.1
	$\boldsymbol{P}_9^{\mathrm{num}}$	3.8	3.6	1.9
shear H_{yx}	$\boldsymbol{P}_{\mathrm{C}}{}^{-1}$	68.8	68.2	59.6
	$\boldsymbol{I}$	5.7	23.5	22.6
	$\boldsymbol{P}_3^{\mathrm{ana}}$	5.6	6.9	6.8
	$\boldsymbol{P}_3^{\mathrm{num}}$	3.7	6.3	6.1
	$\boldsymbol{P}_9^{\mathrm{num}}$	2.6	3.5	1.9
shear H_{xz}	$\boldsymbol{P}_{\mathrm{C}}{}^{-1}$	67.2	56.1	59.6
	$\boldsymbol{I}$	35.2	1.5	22.6
	$\boldsymbol{P}_3^{\mathrm{ana}}$	14.2	1.5	6.8
	$\boldsymbol{P}_3^{\mathrm{num}}$	14.2	0.8	6.1
	$\boldsymbol{P}_9^{\mathrm{num}}$	1.1	0.6	1.9
shear H_{zx}	$\boldsymbol{P}_{\mathrm{C}}{}^{-1}$	66.2	57.9	59.6
	$\boldsymbol{I}$	0.5	29.4	22.6
	$\boldsymbol{P}_3^{\mathrm{ana}}$	0.4	11.4	6.8
	$\boldsymbol{P}_3^{\mathrm{num}}$	0.1	11.2	6.1
	$\boldsymbol{P}_9^{\mathrm{num}}$	0.0	1.4	1.9
shear H_{yz}	$\boldsymbol{P}_{\mathrm{C}}{}^{-1}$	65.8	56.8	59.6
	$\boldsymbol{I}$	36.6	1.7	22.6
	$\boldsymbol{P}_3^{\mathrm{ana}}$	15.1	1.7	6.8
	$\boldsymbol{P}_3^{\mathrm{num}}$	14.7	0.6	6.1
	$\boldsymbol{P}_9^{\mathrm{num}}$	0.9	0.4	1.9
shear H_{zy}	$\boldsymbol{P}_{\mathrm{C}}{}^{-1}$	66.3	57.9	59.6
	$\boldsymbol{I}$	0.5	29.4	22.6
	$\boldsymbol{P}_3^{\mathrm{ana}}$	0.4	13.0	6.8
	$\boldsymbol{P}_3^{\mathrm{num}}$	0.1	11.2	6.1
	$\boldsymbol{P}_9^{\mathrm{num}}$	0.0	1.4	1.9
mix H_{xy}, H_{xz}, H_{xx}	$\boldsymbol{P}_{\mathrm{C}}{}^{-1}$	128.8	117.1	109.5
	$\boldsymbol{I}$	33.5	22.6	33.7
	$\boldsymbol{P}_3^{\mathrm{ana}}$	14.3	7.7	12.3
	$\boldsymbol{P}_3^{\mathrm{num}}$	13.8	7.4	10.6
	$\boldsymbol{P}_9^{\mathrm{num}}$	3.2	2.9	3.6

Table 7.1: Overview of the deviation $||\mathbb{K}_1 - \boldsymbol{P}_{\mathrm{C}}\boldsymbol{P}_{\mathrm{K}} * \mathbb{K}_0||$ / $||\mathbb{K}_1||$ in % of the stiffness tetrads for the four suggested choices of $\boldsymbol{P}_{\mathrm{K}}$ ($\boldsymbol{P}_{\mathrm{C}}{}^{-1}$, $\boldsymbol{I}$, $\boldsymbol{P}_3^{\mathrm{num}}$ and $\boldsymbol{P}_9^{\mathrm{num}}$) and the analytical $\boldsymbol{P}_3^{\mathrm{ana}}$ for the three RVEs. The value of γ and ϵ is 0.5.

8 Summary and outlook

Summary In Chapter 1, we introduced the topic of the work and explained the structure of the following thesis. An overview of similar works was presented to help better understand the work. Additionally, a notation section and a list of frequently used symbols were provided.

In Chapter 2, we postulated a general equation for different plasticity theories. We explained two different ways to access this general equation. First, we started with the multiplicative decomposition of the deformation gradient. The second access to our theory was given by the isomorphy concept. We formulated four special cases for the obtained general equation and summarized them. One of the special cases is the so called Material Plasticity theory. Here, we allowed a different evolution of the material symmetry within the stiffness tetrad and of the stress-free placement. We were also able to account for an evolution of the stiffness tetrad. The special case of fibers moving as material line elements was examined more closely.

We set up a numerical laboratory in Chapter 3. We introduced RVEs for modeling a uni-directional, bi-directional and tri-directional material. They all have different material symmetries namely transversal isotropic, tetragonal and cubic. We used the finite element code Abaqus in combination with the scripting language Python. It is possible to automate the generation of the RVEs with different parameters using the Python scripts.

In Chapter 4, an elastic-plastic material model for large deformations using the isomorphy concept on the micro-scale was used for the RVE calculations. The elastic behavior was captured by the St. Venant-Kirchhoff elasticity and the plastic behavior by the mentioned isomorphy concept. The yield condition follows the isotropic J2 theory by Huber and von Mises. The yield criterion was written in terms of the principle of maximum plastic dissipation. We applied different material parameters within this model for the matrix and the fiber phase. The model was validated regarding the elastic and plastic material behavior.

We implemented and validated a method to measure material stiffness tetrads in Chapter 5. It is based on the difference quotient. The results lead us to our Material Plasticity theory as discussed in Chapter 2. We saw the need of a second-order tensor $\boldsymbol{P}_{\mathrm{K}}$ which transforms between $\mathbb{K}_1$ and $\mathbb{K}_0$. It will predict the evolution of the stiffness tetrad during a plastic deformation of an anisotropic material which

changes the material symmetry. For these calculations, we used the finite element program Abaqus with its scripting language Python. The computer algebra system Mathematica was used for the post-processing.

In Chapter 6, we presented a fast algorithm to determine the distance of a measured stiffness tetrad to all possible symmetries of an elastic stiffness tetrad (Weber, Glüge, and Bertram, 2019). Using a projection method, we build up an 8^{th} order tensor covering all group elements of the chosen group. Applying this projector to a measured stiffness tetrad, we only had to minimize over the orientation of the symmetry axis and therefore only over the three Euler angles.

In the last Chapter 7, we found that it is possible to predict the evolution of the stiffness tetrad during large deformations using one second-order tensor $\boldsymbol{P}_{\mathrm{K}}$ within the equation of our Material Plasticity theory:

$$\overset{\text{2PK}}{\boldsymbol{T}} = \frac{1}{2}\left(\boldsymbol{P}_{\mathrm{C}}\boldsymbol{P}_{\mathrm{K}}\right) * \mathbb{K} : \left(\boldsymbol{C} - \boldsymbol{P}_{\mathrm{C}}^{-\mathrm{T}}\boldsymbol{P}_{\mathrm{C}}^{-1}\right) \ . \tag{8.1}$$

We ascertained an analytical evolution equation to predict this tensor $\boldsymbol{P}_{\mathrm{K}}$ with only three free variables. This is valid for a class of fiber reinforced materials having fiber angles of 90° with a sufficient accuracy (see Weber, Glüge, and H. Altenbach (2020) and Weber, Glüge, and Bertram (2017)).

Outlook Finally, we like to give an outlook. With our theory, we are able to predict the evolution of the stiffness tetrad during large plastic deformations. For this purpose, we finally used a second-order tensor with three free variables which is, for us, the best compromise between accuracy and applicability. For more accurate results or materials with a microstructure different to our 90° scheme, one could use more free variables. It is possible to enhance the theory in this direction.

We assumed small elastic deformations, such that it is sufficient to account only for the quadratic term in the strain energy. It would be possible to present our proposal with a general potential as mentioned in Section 2.2. A generalization could be obtained by a series expansion of the strain energy. One could then approach the change of all higher-order $\mathbb{K}$ tensors similar to our strategy for $\mathbb{K}$ (4^{th} order). It would be interesting to assess the quality of such an approach, but this is beyond the scope of our work.

We neglected viscosity, relaxation and the like. This is a strength of the method and a necessary simplification in our theory. If we mix too many effects it will not be possible to identify individual contributions to the overall behavior. Here, we are concerned with the evolution of the elastic anisotropy due to a fiber reorientation due to plasticity. After having identified a reasonable approach and finding the parameters, it is possible to construct a more sophisticated model, for example, by

adding a viscous contribution or hardening. But for identification purposes, we need to consider only the effect that we seek to model.

It would be interesting to apply the material model to other materials or material mechanism as mentioned in the introduction. This could be twinning in crystals (Christian and Mahajan (1995), Glüge (2009)) or elastic damage mechanisms with an Ansatz where the damage causes a weakening of the elastic properties leads to a modification of the stiffness tetrad of the material. Also materials with lamella structure which for example occur during the solidification of grey cast iron (Berns and Theisen (2008), Bertolino and Perez-Ipiña (2006), Park and Verhoeven (1996)) could be one field of application. The occurrence of lamellae is also observed in eutectic copper-silver-alloys which are used as a conductor material for magnets (Dodla, Bertram, and Krüger (2015), Grünberger, Heilmaier, and Schultz (2001), Sakai, Inoue, Asano, Wada, and Maeda (2001)). For such materials, an initially cubic elastic law can result in an elastic law with less symmetry or also a triclinic one. One can also investigate the third special case within our general equation for plasticity theories. Here, it is possible to describe evolving material properties.

9 Bibliography

Altenbach, H. (2018). *Kontinuumsmechanik Einführung in die materialunabhängigen und materialabhängigen Gleichungen*. Springer Vieweg.

Altenbach, H., Altenbach, J., and Kissing, W. (2018). *Mechanics of Composite Structural Elements*. Springer Nature Singapore Pte Ltd.

Berns, H. and Theisen, W. (2008). *Ferrous Materials - Steel and Cast Iron*. Springer-Verlag Berlin Heidelberg.

Bertolino, G. and Perez-Ipiña, J.E. (2006). "Geometrical effects on lamellar grey cast iron fracture toughness". In: *Journal of Materials Processing Technology* 179, pp. 202–206.

Bertram, A. (1999). "An alternative approach to finite plasticity based on material isomorphisms". In: *International Journal of Plasticity* 15 (3), pp. 353–374.

Bertram, A. (2003). "Finite thermoplasticity based on isomorphisms". In: *International Journal of Plasticity* 19 (11), pp. 2027–2050.

Bertram, A. (2012). *Elasticity and Plasticity of Large Deformations - an Introduction*. 3rd ed. Springer-Verlag, Berlin Heidelberg.

Boehler, J.P. (1987). "Applications of Tensor Functions in Solid Mechanics. International Centre for Mechanical Sciences (Courses and Lectures)". In: vol. 292. Springer, Vienna. Chap. Introduction to the Invariant Formulation of Anisotropic Constitutive Equations.

Böhlke, T. (2001). "Crystallographic Texture Evolution and Elastic Anisotropy: Simulation, Modeling and Applications". PhD thesis. Otto-von-Guericke-University Magdeburg.

Böhlke, T. and Brüggemann, C. (2001). "Graphical Representation of the Generalized Hooke's Law". In: *Technische Mechanik* 21, pp. 145–158.

Bóna, A., Bucataru, I., and Slawinski, A. (2004). "Material Symmetries of Elasticity Tensors". In: *The Quarterly Journal of Mechanics and Applied Mathematics* 54 (4), pp. 584–598.

Brannon, R.M. (2018). *Rotation, Reflection, and Frame Changes*. IOP Publishing.

Broese, C., Sfyris, D., and Tsakmakis, C. (2012). "Isoclinic versus arbitrary rotated intermediate configuration for gradient plasticity applications". In: *Composites Part B: Engineering* 43, pp. 2633–2645.

Bruhns, O.T. (2003). *Advanced Mechanics of Solids*. Springer-Verlag Berlin Heidelberg.

Bruhns, O.T. (2015). "From Creep Damage Mechanics to Homogenization Methods". In: ed. by H. Altenbach, T. Matsuda, and D. Okumura. Springer. Chap. The Multiplicative Decomposition of the Deformation Gradient in Plasticity—Origin and Limitations, pp. 37–66.

Bunge, H.-J. (1982). "Texture Transformation". In: *Texture Analysis in Materials Science*. Ed. by H.-J. Bunge. Butterworth-Heinemann, pp. 188–193.

Butler, P.H. (1981). *Point Group Symmetry Applications: Methods and Tables*. Plenum Press.

Casey, J. and Naghdi, P.M. (1992). "A prescription for the identification of finite plastic strain". In: *International Journal of Engineering Science* 30,10, pp. 1257–1278.

Christian, J.W. and Mahajan, S. (1995). "Deformation twinning". In: *Progress in Materials Science* 39, pp. 1–157.

Cowin, S.C. and Mehrabadi, M.M. (1987). "On the identification of material symmetry for anisotropic elastic materials". In: *The Quarterly Journal of Mechanics and Applied Mathematics* 40 (4), pp. 451–476.

Cowin, S.C. and Mehrabadi, M.M. (1992). "The structure of the linear anisotropic elastic symmetries". In: *Journal of the Mechanics and Physics of Solids* 40 (7), pp. 1459–1471.

Curie, P. (1894). "Sur la symétrie dans les phénomènes physiques, symétrie d'un champ électrique et d'un champ magnétique". In: *Journal de Physique* 3, pp. 393–415.

Del Piero, G. (1975). "On the elastic-plastic material element". In: *Archive for Rational Mechanics and Analysis* 59 (2), pp. 111–129.

Diner, Ç., Kochetov, M., and Slawinski, M.A. (2011). "Identifying Symmetry Classes of Elasticity Tensors Using Monoclinic Distance Function". In: *Journal of Elasticity* 102 (2), pp. 175–190.

Dodla, S., Bertram, A., and Krüger, M. (2015). "Simulation of Flow Behavior and Texture Evolution of Cu-Ag compositesincorporating XRD data". In: *Technische Mechanik* 35 (2), pp. 71–79.

Downey, A.B. (2009). *Python For Software Design - How To Think Like A Computer Scientist*. Cambridge University Press.

Drugan, W.J. and Willis, J.R. (1996). "A Micromechanics-Based Nonlocal Constitutive Equation and Estimates of Representative Volume Element Size for Elastic Composites". In: *Journal of the Mechanics and Physics of Solids* 44, pp. 497–524.

Fedelinski, P. (2011). "Analysis of Representative Volume Elements with Random Microcracks". In: *Computational Modelling and Advanced Simulations*. Ed. by J. Murín, V. Kompiš, and V. Kutiš. Vol. 24. Springer Dordrecht, Heidelberg, London, New York, pp. 333–341.

Fedorov, F.I. (1968). *Theory of Elastic Waves in Crystals*. Plenum Press.

Forest, S. and Parisot, R. (2000). "Material crystal plasticity and deformation twinning". In: *Geom. Cont. and Micros.* 58, pp. 99–112.

Forte, S. and Vianello, M. (1996). "Symmetry Classes for Elasticity Tensors". In: *Journal of Elasticity* 43 (2), pp. 81–108.

Francois, M., Geymonat, G., and Berthaud, Y. (1998). "Determination of the symmetries of an experimentally determined stiffness tensor; application to acoustic measurements". In: *International Journal of Solids and Structures* 35 (31-32), pp. 4091–4106.

Freund, M., Shutov, A.V., and Ihlemann, J. (2012). "Simulation of distortional hardening by generalizing a uniaxial model of finite strain viscoplasticity". In: *International Journal of Plasticity* 36, pp. 113–129.

Gazis, D.C., Tadjbakhsh, I., and Toupin, R.A. (1963). "The elastic tensor of given symmetry nearest to an anisotropic elastic tensor". In: *Acta Crystallographica* 16 (9), pp. 917–922.

Gitman, I.M., Askes, H., and Sluys, L.J. (2007). "Representative volume: Existence and size determination". In: *Engineering Fracture Mechanics* 74, pp. 2518–2534.

Gitman, I.M., Gitman, M.B., and Askes, H. (2006). "Quantification of Stochastically Stable Representative Volumes for Random Heterogeneous Materials". In: *Archive of Applied Mechanics* 75, pp. 79–92.

Glüge, R. (2009). "Elastic modelling of deformation twinning on the microscale". PhD thesis. Otto-von-Guericke Universität Magdeburg.

Glüge, R., Weber, M., and Bertram, A. (2012). "Comparison of spherical and cubical statistical volume elements with respect to convergence, anisotropy, and localization behavior". In: *Computational Material Science* 63, pp. 91–104.

Grünberger, W., Heilmaier, M., and Schultz, L. (2001). "Development of high-strength and high-conductivity conductor materials for pulsed high-field magnets at Dresden". In: *Physica B: Condensed Matter* 294-295 (0), pp. 643–647.

Guilleminot, J. and Soize, C. (2010). "A stochastic model for elasticity tensors with uncertain material symmetries". In: *International Journal of Solids and Structures* 47 (22-23), pp. 3121–3130.

Hairer, E., Nørsett, S.P., and Wanner, G. (1993). *Solving Ordinary Differential Equations I.* Springer-Verlag Berlin Heidelberg.

Hashin, Z. (1983). "Analysis of Composite Materials - A Survey". In: *Journal of Applied Mechanics* 50, pp. 481–505.

Haupt, P. (2000). *Continuum Mechanics and Theory of Materials.* Springer, Berlin.

Hermann, C. (1934). "Tensoren und Kristallsymmetrie". In: *Zeitschrift für Kristallographie - Crystalline Materials* 89 (1), pp. 32–48.

Hill, R. (1951). "The Elastic Behaviour of a Crystalline Aggregate". In: *Proceedings of the Royal Society of London A* 65, pp. 349–354.

Hill, R. (1956). "The mechanics of quasi-static plastic deformation in metals". In: *Surveys in Mechanics - The G. I. Taylor 70th Anniversary Volume.* Cambridge University Press, pp. 7–31.

Hill, R. (1963). "Elastic Properties of Reinforced Solids: Some Theoretical Principles". In: *Journal of the Mechanics and Physics of Solids* 11, pp. 357–372.

Hufenbach, W., Langkamp, A., Adam, F., Krahl, M., Hornig, A., Zscheyge, M., and Modler, K.-H. (2011). "An integral design and manufacturing concept for crash resistant textile and long-fibre reinforced polypropylene structural components". In: *Procedia Engineering* 10 (0), pp. 2086–2091.

Itskov, M. and Aksel, N. (2004). "A constitutive model for orthotropic elasto-plasticity at large strains". In: *Archive of Applied Mechanics* 74 (1-2), pp. 75–91.

Jeffery, G. B. and Filon, L.N.G. (1922). "The motion of ellipsoidal particles immersed in a viscous fluid". In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 102 (715), pp. 161–179.

Jöchen, K. and Böhlke, T. (2012). "Prediction of Texture Evolution in Rolled Sheet Metals by Using Homogenization Schemes". In: *Material Forming ESAFORM 2012.* Vol. 504. Key Engineering Materials. Trans Tech Publications, pp. 649–654.

Kaiser, T., Lu, J., Menzel, A., and Papadopoulos, P. (2018). "On anisotropy evolution in finite strain plasticity". In: *Proceedings in Applied Mathematics and Mechanics* 18 (1).

Kaiser, T., Lu, J., Menzel, A., and Papadopoulos, P. (2019). "A Covariant Formulation of Finite Plasticity with Plasticity-induced Evolution of Anisotropy: Modeling, Algorithmics, Simulation, and Comparison to Experiments". In: *International Journal of Solids and Structures*.

Kim, K.H. and Yin, J.J. (1997). "Evolution of anisotropy under plane stress". In: *Journal of the Mechanics and Physics of Solids* 45 (5), pp. 841–851.

Klusemann, B. and Svendsen, B. (2010). "Homogenization methods for multi-phase elastic composites: Comparisons and benchmarks". In: *Technische Mechanik* 30 (4), pp. 374–386.

Kochetov, M. and Slawinski, M.A. (2009). "On Obtaining Effective Transversely Isotropic Elasticity Tensors". In: *Journal of Elasticity* 94 (1), pp. 1–13.

Lee, E. and Liu, D.T. (1967). "Finite-strain elastic-plastic theory with application to plane-wave analysis". In: *Journal of Applied Physics* 38, pp. 19–27.

Lu, J. and Papadopoulos, P. (2004). "A covariant formulation of anisotropic finite plasticity: theoretical developments". In: *Computer Methods in Applied Mechanics and Engineering* 193 (48-51), pp. 5339–5358.

Lubarda, V.A. (2002). *Elastoplasticity Theory*. CRC Press, Boca Raton.

Mandel, J. (1965). "Generalisation de la theorie de plasticite de W. T. Koiter". In: *International Journal of Solids and Structures* 1 (3), pp. 273–295.

Mandel, J. (1973). "Equations constitutives et directeurs dans les milieux plastique et viscoplastiques". In: *International Journal of Solids and Structures* 9, pp. 725–740.

Martinie, L. and Roussel, N. (2011). "Simple tools for fiber orientation prediction in industrial practice". In: *Cement and Concrete Research* 41 (10), pp. 993–1000.

Menzel, A. and Steinmann, P. (2003). "On the spatial formulation of anisotropic multiplicative elasto-plasticity". In: *Computer Methods in Applied Mechanics and Engineering* 192 (31-32), pp. 3431–3470.

Miehe, C. (1995). "A theory of large-strain isotropic thermoplasticity based on metric transformation tensors". In: *Archive of Applied Mechanics* 66, pp. 45–64.

Miehe, C. (1996). "Exponential map algorithm for stress updates in anisotropic multiplicative elastoplasticity for single crystals". In: *International Journal for Numerical Methods in Engineering* 39 (19), pp. 3367–3390.

Miehe, C. (1998a). "A constitutive frame of elastoplasticity at large strains based on the notion of a plastic metric". In: *International Journal of Solids and Structures* 35, pp. 3859–3897.

Miehe, C. (1998b). "A formulation of finite elastoplasticity based on dual co- and contra-variant eigenvector triads normalized with respect to a plastic metric". In: *Computer Methods in Applied Mechanics and Engineering* 159, pp. 223–260.

Miehe, C. (2003). "Computational micro-to-macro transitions for discretized microstructures of heterogeneous materials at finite strains based on the minimization of averaged incremental energy". In: *Computer Methods in Applied Mechanics and Engineering* 192, pp. 559–591.

Mitschang, P., Blinzler, M., and Wöginger, A. (2003). "Processing technologies for continuou fibre reinforced thermoplastics with novel polymer blend". In: *Composites Science and Technology* 63, pp. 2099–2110.

Moakher, M. and Norris, A.N. (2006). "The Closest Elastic Tensor of Arbitrary Symmetry to an Elasticity Tensor of Lower Symmetry". In: *Journal of Elasticity* 85 (3), pp. 215–263.

Morawiec, A. (2004). *Orientations and Rotations - Computations in Crystallographic Textures*. Springer.

Nawratil, G. and Pottmann, H. (2008). "Subdivision schemes for the fair discretization of the spherical motion group". In: *Journal of Computational and Applied Mathematics* 222 (2), pp. 574–591.

Nemat-Nasser, S. (1999). "Averaging theorems in finite deformation plasticity". In: *Mech. Mater.* 31, pp. 493–523.

Neumann, F.E. (1885). *Vorlesungen über die Theorie der Elastizität der festen Körper und des Lichtäthers*. Ed. by O.E. Meyer. Teubner-Verlag Leipzig.

Noll, W. (1958). "A mathematical theory of the mechanical behavior of continuous media". In: *Archive for Rational Mechanics and Analysis* 2, pp. 197–226.

Norris, A.N. (2006). "Elastic moduli approximation of higher symmetry for the acoustical properties of an anisotropic material". In: *Journal of the Acoustical Society of America* 119 (4), pp. 2114–2121.

Olive, M. and Auffray, N. (2013). "Symmetry classes for even-order tensors". In: *Mathematics and Mechancis of Complex Systems* 1 (2), pp. 177–210.

Olive, M. and Auffray, N. (2014). "Symmetry classes for odd-order tensors". In: *Zeitschrift für Angewandte Mathematik und Mechanik* 94 (5), pp. 421–447.

Ortega, J.M. and Rheinboldt, W.C. (1970). "Chapter 7 - General iterative methods". In: *Iterative Solution of Nonlinear Equations in Several Variables*. Ed. by J.M. Ortega and W.C. Rheinboldt. Academic Press, pp. 181–239. ISBN: 978-0-12-528550-6.

Ostoja-Startzewski, M. (1998). "Random Field Models of Heterogeneous Materials". In: *International Journal of Solids and Structures* 35, pp. 2429–2455.

Panhans, S. and Kreißig, R. (2006). "A viscoplastic material model of overstress type with a non-quadratic yield function". In: *European Journal of Mechanics - A/Solids* 25, pp. 283–298.

Park, J.S. and Verhoeven, J.D. (1996). "Directional solidification of white cast iron". In: *Metallurgical and Materials Transactions A* 27 (8), pp. 2328–2337.

Pastor, J., Turgeman, S., and Boehler, J.P. (1990). "Solution of anisotropic plasticity problems by using associated isotropic problems". In: *International Journal of Plasticity* 6 (2), pp. 143–168.

Pietryga, M.P., Vladimirov, I.N., and Reese, S. (2012). "A finite deformation model for evolving flow anisotropy with distortional hardening including experimental validation". In: *Mechanics of Materials* 44 (0), pp. 163–173.

Rastellini, F., Oller, S., Salomón, O., and Oñate, E. (2008). "Composite materials non-linear modelling for long fibre-reinforced laminates: Continuum basis, computational aspects and validations". In: *Computers & Structures* 86 (9), pp. 879–896.

Reese, S., Pietryga, M.P., and Vladimirov, I.N. (2009). "Anisotropic finite plasticity based on structural tensors - different strategies and application to forming simulations". In: *International Journal of Material Forming* 2, pp. 459–462.

Roters, F. (2011). *Advanced Material Models for the Crystal Plasticity Finite Element Method*. Habilitation Thesis.

Sab, K. (1992). "On the Homogenization and the Simulation of Random Materials". In: *European Journal of Mechanics - A/Solids* 11, pp. 585–607.

Sachs, E. (1928). "Zur Ableitung einer Fließbedingung". In: *Zeitschrift des Vereins deutscher Ingenieure* 72, pp. 734–736.

Sakai, Y., Inoue, K., Asano, T., Wada, H., and Maeda, H. (2001). "Development of high-strength, high-conductivity Cu-Ag alloys for high-field pulsed magnet use". In: *Applied Physics Letters* 59, 23, pp. 2965–2967.

Segurado, J. and Llorca, J. (2002). "A numerical approximation to the elastic properties of sphere-reinforced composites". In: *Journal of the Mechanics and Physics of Solids* 50, pp. 2107–2121.

Seth, B.R. (1964). "Generalized strain measure with applications to physical problems". In: *Second-Order Effects in Elasticity, Plasticity and Fluid Dynamics*. Ed. by M. Reiner and D. Abir. Pergamon Press, Oxford, pp. 162–172.

Shan, Z. and Gokhale, A.M. (2002). "Representative volume element for non-uniform micro-structure". In: *Computational Materials Science* 24, pp. 361–379.

Shutov, A.V. and Ihlemann, J. (2012). "A viscoplasticity model with an enhanced control of the yield surface distortion". In: *International Journal of Plasticity* 39, pp. 152–167.

Shutov, A.V. and Kreißig, R. (2008). "Finite strain viscoplasticity with nonlinear kinematic hardening: Phenomenological modeling and time integration". In: *Computer Methods in Applied Mechanics and Engineering* 197, pp. 2015–2029.

Shutov, A.V. and Kreißig, R. (2010). "Geometric integrators for multiplicative viscoplasticity: Analysis of error accumulation". In: *Computer Methods in Applied Mechanics and Engineering* 199, pp. 700–711.

Šilhavý, M. and Kratochvíl, J. (June 1977a). "A theory of inelastic behavior of materials Part I. Ideal inelastic materials". In: *Archive for Rational Mechanics and Analysis* 65, pp. 97–129.

Šilhavý, M. and Kratochvíl, J. (June 1977b). "A theory of inelastic behavior of materials Part II. Inelastic materials". In: *Archive for Rational Mechanics and Analysis* 65, pp. 131–152.

Simulia (2010). *Abaqus Version 6.10 Documentation*. Dassault Systèmes.

Suquet, P.M. (1985). "Averages Boundary conditions". In: *Homogenization Techniques for Composite Media*. Ed. by E. Sanchez-Palencia and A. Zaoui. Vol. 272. Lecture Notes in Physics, pp. 199–208.

Suwas, S. and Ray, R.K. (2014). *Crystallographic Texture of Materials*. Engineering Materials and Processes. Springer-Verlag London.

Taylor, G. (1938). "Analysis of plastic strain in a cubic crystal". In: *Journal of the Institute of Metals (Reprinted in The scientific papers of Sir Geoffrey Ingram Taylor, Cambridge University Press 1958)* 62, pp. 307–324.

Thomson, W. (1856). "XXI. Elements of a mathematical theory of elasticity". In: *Philosophical Transactions of the Royal Society of London* 146, pp. 481–498.

Truesdell, C.A. and Noll, W. (1965). *The Non-Linear Field Theories of Mechanics*. Springer-Verlag Berlin Heidelberg.

Vladimirov, I.N., Pietryga, M.P., and Reese, S. (2011). "On the influence of kinematic hardening on plastic anisotropy in the context of finite strain plasticity". In: *International Journal of Material Forming* 4 (2), pp. 255–267.

Vladimirov, I.N. and Reese, S. (2008). "Anisotropic finite plasticity with combined hardening and application to sheet metal forming". In: *International Journal of Material Forming* 1, pp. 293–296.

Voigt, W. (1910). *Lehrbuch der Kristallphysik (mit Ausschluß der Kristalloptik)*. Vol. 34. B. G. Teubner's Sammlung von Lehrbüchern auf dem Gebiete der mathematischen Wissenschaften. Teubner, Leipzig.

Waffenschmidt, T., Polindara, C., Menzel, A., and Blanco, S. (2014). "A gradient-enhanced large-deformation continuum damage model for fibre-reinforced materials". In: *Computer Methods in Applied Mechanics and Engineering* 268, pp. 801–842.

Weber, M., Glüge, R., and Altenbach, H. (2020). "Evolution of the stiffness tetrad in fiber-reinforced materials under large plastic strain - Material Plasticity". In: *Archive of Applied Mechanics*.

Weber, M., Glüge, R., and Bertram, A. (2017). "Material plasticity to track the elastic anisotropy at finite strains". In: *Proceedings in Applied Mathematics and Mechanics* 17 (1), pp. 481–482.

Weber, M., Glüge, R., and Bertram, A. (2019). "Distance of a stiffness tetrad to the symmetry classes of linear elasticity". In: *International Journal of Solids and Structures* 156-157, pp. 281–293.

Xiao, H., Bruhns, O.T., and Meyers, A. (1999). "A Natural Generalization of Hypoelasticity and Eulerian Rate Type Formulation of Hyperelasticity". In: *Journal of Elasticity* 56, pp. 59–93.

Zaoui, A. (2001). "Changement d'échelle: motivation et méthodologie". In: *Homogénéisation en mécanique des matériaux*. Ed. by M. Bornert, T. Bretheau, and P. Gilormini. Hérmes Science, 19–39 (Kapitel 1).

Zou, W.N., Tang, C.X., and Lee, W.H. (2013). "Identification of symmetry type of linear elastic stiffness tensor in an arbitrarily orientated coordinate system". In: *International Journal of Solids and Structures* 50 (14-15), pp. 2457–2467.

Appendix A

Python script to generate the different RVEs

A.1 Build RVEs "0_rve.py"

```
# ======================================================================
# RVE subroutine, able to generate 5 different RVE types with partially
# different boundary conditions, element sizes and shapes
# ======================================================================
print '======================================================================'
print 'Start RVE subroutine'
print '======================================================================'
t0abs = time.time();
# CPU number
cpunumber = '1'
# Set RVE type
typ    = 'U'
# Set type of boundary condition
rb     = 'PBC'
# Set element size
ele = 0.10
# Set element shape HEX, WEDGE, TET
eleshape = 'HEX'
# Element type Hex:    lin: C3D8 quad: C3D20
#              Wedge: lin: C3D6 quad: C3D15
#              Tet:    lin: C3D4 quad: C3D10
# Set element type
eletyp = 'C3D20'
# ======================================================================
# Element type T1F: 1 square fiber in x direction
#                   tetragonal symmetry
# ======================================================================
if typ == 'T1F':
    print '======================================================================'
    print 'Element type T1F: 1 square fiber in x direction'
    print '======================================================================'
    ##########################################################################
    #Faserradius
    rf1=3.
    rf2=3.
    rf3=3.
    #RVE thickness d
    drve= 10.
    #Number of unit cells
    XNUC=1
```

```
YNUC=1
ZNUC=1
#Calculate COOS
movex = -0.5*(XNUC-1.)
movey = -sqrt(3.)/2.*(YNUC-1.)
#Area A0
#A0 = fa1*fa2*sin(anglerad)
#A0 = fa1*YNUC*fa2*XNUC
#Volume V0
V0 = drve*drve*drve
#Alt, nur noch fuer namen
seed = []
seed.append(XNUC)
##############################################################################
#      faserinput.txt anlegen
##############################################################################
inpname = 'RVE_%s_%s_%s_ele%s_%s' %(typ,'X'+str(XNUC)+'Y'+str(YNUC)+'Z'+str(
    ZNUC),rb,str(ele).replace('.',''),extname)
print inpname
f = open(inpname+'_fiberinput.txt','w')
f.write('CPU number:\n')
f.write('%s\n' %(cpunumber))
f.write('Type of RVE:\n')
f.write('%s\n' %(typ))
f.write('Type of boundary condition:\n')
f.write('%s\n' %(rb))
f.write('Element size [ele]:\n')
f.write('%.3f\n' %(ele))
f.write('Element shape:\n')
f.write('%s\n' %(eleshape))
f.write('Element type:\n')
f.write('%s\n' %(eletyp))
f.write('Radius fiber 1:\n')
f.write('%.3f\n' %(rf1))
f.write('Distance fiber 1:\n')
#f.write('%.3f\n' %(fa1))
#f.write('Radius fiber 2:\n')
#f.write('%.3f\n' %(rf2))
#f.write('Distance fiber 2:\n')
#f.write('%.3f\n' %(fa2))
#f.write('Layer Distance:\n')
#f.write('%.3f\n' %(la))
f.write('RVE thickness d:\n')
f.write('%.3f\n' %(drve))
#f.write('Angle between fibers:\n')
#f.write('%.3f\n' %(angle))

f.write('Displacement Gradient\n')
f.write('H0:\n')
f.write('%s,%s,%s\n' %(str(H0[0][0]),str(H0[0][1]),str(H0[0][2])))
f.write('%s,%s,%s\n' %(str(H0[1][0]),str(H0[1][1]),str(H0[1][2])))
f.write('%s,%s,%s\n' %(str(H0[2][0]),str(H0[2][1]),str(H0[2][2])))

f.write('Name of input file\n')
f.write(inpname)
f.write('\n ')
f.close()
##############################################################################
# Generate model
##############################################################################
# Close open model
Mdb()
session.viewports['Viewport: 1'].setValues(displayedObject=None)
session.journalOptions.setValues(replayGeometry=COORDINATE, recoverGeometry=
    COORDINATE)
# Section
```

```
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber1', material='Fiber1',
    thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber2', material='Fiber2',
    thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber3', material='Fiber3',
    thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Matrix', material='Matrix',
    thickness=None)
# Build rhombus and fibers
session.viewports['Viewport: 1'].partDisplay.setValues(sectionAssignments=OFF,
    engineeringFeatures=OFF)
session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
    referenceRepresentation=ON)
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=STANDALONE)
s.rectangle(point1=(-5.0, 5.0), point2=(5.0, -5.0))
p = mdb.models['Model-1'].Part(name='Part-1', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-1']
p.BaseSolidExtrude(sketch=s, depth=5.0)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-1']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']
session.viewports['Viewport: 1'].view.setValues(nearPlane=22.3108,
    farPlane=40.9522, width=31.4415, height=10.6695, cameraPosition=(
    28.8266, 4.06394, -9.87444), cameraUpVector=(-0.374061, 0.881389,
    0.288499))
p = mdb.models['Model-1'].parts['Part-1']
f, e = p.faces, p.edges
t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(-1.666667,
    1.666667, 0.0)), sketchUpEdge=e.findAt(coordinates=(-5.0, 2.5, 0.0)),
    sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, origin=(0.0, 0.0, 0.0))
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=28.28, gridSpacing=0.7, transform=t)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['Model-1'].parts['Part-1']
p.projectReferencesOntoSketch(sketch=s1, filter=COPLANAR_EDGES)
s1.rectangle(point1=(-5.0, 5.0), point2=(5.0, -5.0))
p = mdb.models['Model-1'].parts['Part-1']
f1, e1 = p.faces, p.edges
p.SolidExtrude(sketchPlane=f1.findAt(coordinates=(-1.666667, 1.666667, 0.0)),
    sketchUpEdge=e1.findAt(coordinates=(-5.0, 2.5, 0.0)),
    sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, sketch=s1, depth=5.0,
    flipExtrudeDirection=OFF)
s1.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])
p = mdb.models['Model-1'].parts['Part-1']
f, e, d1 = p.faces, p.edges, p.datums
t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(-1.666667, 5.0,
    3.333333)), sketchUpEdge=e.findAt(coordinates=(5.0, 5.0, -1.25)),
    sketchPlaneSide=SIDE1, origin=(0.0, 5.0, 0.0))
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=34.64, gridSpacing=0.86, transform=t)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['Model-1'].parts['Part-1']
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
s.rectangle(point1=(-2.5, 2.5), point2=(2.5, -2.5))
p = mdb.models['Model-1'].parts['Part-1']
f = p.faces
pickedFaces = f.findAt(((-1.666667, 5.0, 3.333333), ))
```

```
e1, d2 = p.edges, p.datums
p.PartitionFaceBySketch(sketchUpEdge=e1.findAt(coordinates=(5.0, 5.0, -1.25)),
    faces=pickedFaces, sketch=s)
s.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
pickedCells = c.findAt(((-0.833333, 5.0, 0.833333), ))
e, d1 = p.edges, p.datums
pickedEdges =(e.findAt(coordinates=(-2.5, 5.0, 1.25)), e.findAt(coordinates=(
    1.25, 5.0, 2.5)), e.findAt(coordinates=(2.5, 5.0, -1.25)), e.findAt(
    coordinates=(-1.25, 5.0, -2.5)))
p.PartitionCellByExtrudeEdge(line=e.findAt(coordinates=(5.0, 2.5, 5.0)),
    cells=pickedCells, edges=pickedEdges, sense=FORWARD)
p = mdb.models['Model-1'].parts['Part-1']
f1, e1, d2 = p.faces, p.edges, p.datums
t = p.MakeSketchTransform(sketchPlane=f1.findAt(coordinates=(5.0, 1.666667,
    3.333333)), sketchUpEdge=e1.findAt(coordinates=(5.0, -2.5, -5.0)),
    sketchPlaneSide=SIDE1, origin=(5.0, 0.0, 0.0))
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=34.64, gridSpacing=0.86, transform=t)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['Model-1'].parts['Part-1']
p.projectReferencesOntoSketch(sketch=s1, filter=COPLANAR_EDGES)
s1.rectangle(point1=(-2.5, 2.5), point2=(2.5, -2.5))
p = mdb.models['Model-1'].parts['Part-1']
f = p.faces
pickedFaces = f.findAt(((5.0, 1.666667, 3.333333), ))
e, d1 = p.edges, p.datums
p.PartitionFaceBySketch(sketchUpEdge=e.findAt(coordinates=(5.0, -2.5, -5.0)),
    faces=pickedFaces, sketch=s1)
s1.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
pickedCells = c.findAt(((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0,
    0.833333), ))
e1, d2 = p.edges, p.datums
pickedEdges =(e1.findAt(coordinates=(5.0, -1.25, 2.5)), e1.findAt(coordinates=(
    5.0, -2.5, -1.25)), e1.findAt(coordinates=(5.0, 1.25, -2.5)),
    e1.findAt(coordinates=(5.0, 2.5, 1.25)))
p.PartitionCellByExtrudeEdge(line=e1.findAt(coordinates=(2.5, -5.0, 5.0)),
    cells=pickedCells, edges=pickedEdges, sense=FORWARD)
p = mdb.models['Model-1'].parts['Part-1']
f, e, d1 = p.faces, p.edges, p.datums
t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(1.666667, 1.666667,
    5.0)), sketchUpEdge=e.findAt(coordinates=(5.0, 2.5, 5.0)),
    sketchPlaneSide=SIDE1, origin=(0.0, 0.0, 5.0))
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=34.64, gridSpacing=0.86, transform=t)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['Model-1'].parts['Part-1']
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
s.rectangle(point1=(-2.5, 2.5), point2=(2.5, -2.5))
p = mdb.models['Model-1'].parts['Part-1']
f = p.faces
pickedFaces = f.findAt(((1.666667, 1.666667, 5.0), ))
e1, d2 = p.edges, p.datums
p.PartitionFaceBySketch(sketchUpEdge=e1.findAt(coordinates=(5.0, 2.5, 5.0)),
    faces=pickedFaces, sketch=s)
s.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
```

```
    pickedCells = c.findAt(((-0.833333, -5.0, -0.833333), ), ((-0.833333, 0.833333,
        2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((-0.833333, -0.833333,
        5.0), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0, 0.833333), ))
    e, d1 = p.edges, p.datums
    pickedEdges =(e.findAt(coordinates=(-2.5, -1.25, 5.0)), e.findAt(coordinates=(
        1.25, -2.5, 5.0)), e.findAt(coordinates=(2.5, 1.25, 5.0)), e.findAt(
        coordinates=(-1.25, 2.5, 5.0)))
    p.PartitionCellByExtrudeEdge(line=e.findAt(coordinates=(5.0, -5.0, 1.25)),
        cells=pickedCells, edges=pickedEdges, sense=REVERSE)
    # Assign Section
    session.viewports['Viewport: 1'].partDisplay.setValues(sectionAssignments=ON,
        engineeringFeatures=ON)
    session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
        referenceRepresentation=OFF)
    p = mdb.models['Model-1'].parts['Part-1']
    session.viewports['Viewport: 1'].setValues(displayedObject=p)
    session.viewports['Viewport: 1'].view.setValues(session.views['Front'])
    session.viewports['Viewport: 1'].view.setProjection(projection=PARALLEL)
    p = mdb.models['Model-1'].parts['Part-1']
    c = p.cells
    # Tetragonal only x fiber
    cells = c.findAt(((-0.833333, 0.833333, 2.4995), ),
        ((-5.0, -0.833333, 0.833333), ), ((5.0, 0.833333, 0.833333), ))
    # Tetragonal x and y fiber
    #cells = c.findAt(((-0.833333, -5.0, -0.833333), ), ((-0.833333, 0.833333,
        2.4995), ),
        #((-5.0, -0.833333, 0.833333), ), ((5.0, 0.833333, 0.833333), ), ((
        #-0.833333, 5.0, 0.833333), ))
    # Original x, y and z fiber
#     cells = c.findAt(((-0.833333, 0.833333, -5.0), ), ((-0.833333, -0.833333, 5.0)
    ,
#         ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333, 0.833333, 2.4995), ),
#         ((-5.0, -0.833333, 0.833333), ), ((5.0, 0.833333, 0.833333), ), ((
#         -0.833333, 5.0, 0.833333), ))
    region = regionToolset.Region(cells=cells)
    p = mdb.models['Model-1'].parts['Part-1']
    p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
        offsetType=MIDDLE_SURFACE, offsetField='',
        thicknessAssignment=FROM_SECTION)
    p = mdb.models['Model-1'].parts['Part-1']
    c = p.cells
    cells = c.findAt(((-0.833333, -5.0, -0.833333), ), ((-0.833333, 5.0, 0.833333),
        ),
                     ((-0.833333, 0.833333, -5.0), ), ((-0.833333, -0.833333, 5.0),
                         ),
                     ((4.166667, -5.0, 2.5), ))
    region = regionToolset.Region(cells=cells)
    p = mdb.models['Model-1'].parts['Part-1']
    p.SectionAssignment(region=region, sectionName='Matrix', offset=0.0,
        offsetType=MIDDLE_SURFACE, offsetField='',
        thicknessAssignment=FROM_SECTION)
    session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])
    # Assembly
    a = mdb.models['Model-1'].rootAssembly
    session.viewports['Viewport: 1'].setValues(displayedObject=a)
    session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=OFF)
    session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
        meshTechnique=OFF)
    a1 = mdb.models['Model-1'].rootAssembly
    a1.DatumCsysByDefault(CARTESIAN)
    p = mdb.models['Model-1'].parts['Part-1']
    a1.Instance(name='FIBBI-1', part=p, dependent=OFF)
    session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=ON)
    session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
        meshTechnique=ON)
    # Partition
```

```
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((-0.833333, 0.833333, -5.0), ), ((-0.833333,
    -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333,
    0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((4.166667,
    -5.0, 2.5), ), ((-0.833333, 5.0, 0.833333), ))
v21 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v21.findAt(coordinates=(-2.5, -2.5,
    5.0)), point2=v21.findAt(coordinates=(-2.5, 2.5, 5.0)),
    point3=v21.findAt(coordinates=(-2.5, 5.0, 2.5)), cells=pickedCells)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((-0.833333, 0.833333, -5.0), ), ((-0.833333,
    -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333,
    0.833333, 2.4995), ), ((-0.833333, -5.0, 3.333333), ), ((5.0, 0.833333,
    0.833333), ), ((-0.833333, 5.0, 0.833333), ))
v1 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v1.findAt(coordinates=(2.5, -2.5,
    5.0)), point2=v1.findAt(coordinates=(2.5, 2.5, 5.0)), point3=v1.findAt(
    coordinates=(2.5, 5.0, 2.5)), cells=pickedCells)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((0.833333, 4.166667, 5.0), ), ((0.833333, -5.0,
    4.166667), ), ((3.333333, -5.0, -4.166667), ), ((-3.333333, 5.0,
    -4.166667), ), ((-0.833333, -0.833333, 5.0), ), ((-0.833333, -5.0,
    -0.833333), ), ((-0.833333, 0.833333, 2.4995), ), ((-5.0, -0.833333,
    0.833333), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0,
    0.833333), ))
v21 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v21.findAt(coordinates=(5.0, -2.5,
    2.5)), point2=v21.findAt(coordinates=(5.0, 2.5, 2.5)),
    point3=v21.findAt(coordinates=(-2.5, 5.0, 2.5)), cells=pickedCells)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((0.833333, 5.0, -4.166667), ), ((4.166667, -5.0,
    1.666667), ), ((-4.166667, 5.0, 1.666667), ), ((-0.833333, 0.833333,
    -5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333, 0.833333,
    2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((0.833333, -4.166667,
    -5.0), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0, 0.833333),
    ))
v1 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v1.findAt(coordinates=(5.0, -2.5,
    -2.5)), point2=v1.findAt(coordinates=(5.0, 2.5, -2.5)),
    point3=v1.findAt(coordinates=(-2.5, 5.0, -2.5)), cells=pickedCells)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((-5.0, -4.166667, -1.666667), ), ((-4.166667, 5.0,
    -3.333333), ), ((4.166667, -5.0, -3.333333), ), ((-4.166667, -5.0,
    3.333333), ), ((4.166667, 5.0, 3.333333), ), ((0.833333, -3.333333,
    5.0), ), ((4.166667, -5.0, -1.666667), ), ((-0.833333, 0.833333, -5.0),
    ), ((-0.833333, -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), (
    (-0.833333, 0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((
    0.833333, -4.166667, -5.0), ), ((5.0, 0.833333, 0.833333), ))
v21 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v21.findAt(coordinates=(-2.5, -2.5,
    5.0)), point2=v21.findAt(coordinates=(2.5, -2.5, 5.0)),
    point3=v21.findAt(coordinates=(5.0, -2.5, 2.5)), cells=pickedCells)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((-5.0, -0.833333, 3.333333), ), ((3.333333, -0.833333,
    -5.0), ), ((-4.166667, 5.0, -3.333333), ), ((4.166667, 5.0, -1.666667),
    ), ((4.166667, 5.0, 3.333333), ), ((0.833333, 4.166667, 5.0), ), ((
    0.833333, 3.333333, -5.0), ), ((-5.0, 4.166667, -1.666667), ), ((
    -0.833333, 0.833333, -5.0), ), ((-0.833333, -0.833333, 5.0), ), ((
    -0.833333, 0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((
    5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0, 0.833333), ))
```

```
v1 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v1.findAt(coordinates=(-2.5, 2.5,
    5.0)), point2=v1.findAt(coordinates=(2.5, 2.5, 5.0)), point3=v1.findAt(
    coordinates=(5.0, 2.5, -2.5)), cells=pickedCells)
#Mesh
elemType1 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
elemType3 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
cells1 = c1.findAt(((5.0, 0.833333, 4.166667), ), ((-5.0, 0.833333, -4.166667),
    ), ((4.166667, 3.333333, -5.0), ), ((-5.0, 3.333333, 4.166667), ), ((
    3.333333, -5.0, 3.333333), ), ((-5.0, 0.833333, 3.333333), ), ((
    3.333333, 0.833333, -5.0), ), ((-5.0, -3.333333, -4.166667), ), ((
    -4.166667, -5.0, -1.666667), ), ((-4.166667, 5.0, -3.333333), ), ((
    4.166667, 5.0, -1.666667), ), ((5.0, -3.333333, -3.333333), ), ((
    -3.333333, -3.333333, 5.0), ), ((4.166667, 5.0, 3.333333), ), ((
    0.833333, 4.166667, 5.0), ), ((0.833333, 3.333333, -5.0), ), ((
    0.833333, -5.0, 4.166667), ), ((5.0, -4.166667, -1.666667), ), ((-5.0,
    4.166667, -1.666667), ), ((-0.833333, 0.833333, -5.0), ), ((-0.833333,
    -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333,
    0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((0.833333,
    -5.0, -3.333333), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0,
    0.833333), ))
pickedRegions =(cells1, )
a.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
    elemType3))
a = mdb.models['Model-1'].rootAssembly
partInstances =(a.instances['FIBBI-1'], )
a.seedPartInstance(regions=partInstances, size=1.25, deviationFactor=0.1)
a = mdb.models['Model-1'].rootAssembly
partInstances =(a.instances['FIBBI-1'], )
a.generateMesh(regions=partInstances)
##############################################################################
# Write input file
##############################################################################
#The name of the operating system dependent module imported.
#The following names have currently been registered:
#'posix', 'nt', 'os2', 'ce', 'java', 'riscos'.
if "posix" in os.name:
#     print 'LINUX'
##Cluster
#     mdb.Job(name=inpname, model='Model-1', type=ANALYSIS,
#          explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
#          description='', parallelizationMethodExplicit=DOMAIN,
#          multiprocessingMode=DEFAULT, numDomains=1, userSubroutine='',
#          numCpus=1, preMemory=16056.0, standardMemory=16056.0,
#          standardMemoryPolicy=MODERATE, scratch='', echoPrint=OFF,
#          modelPrint=OFF, contactPrint=OFF, historyPrint=OFF)
#Local
  mdb.Job(name=inpname, model='Model-1', description='', type= ANALYSIS,
    atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
    memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
    modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
    scratch='', parallelizationMethodExplicit=DOMAIN, numDomains=1,
    activateLoadBalancing=False, multiprocessingMode=DEFAULT, numCpus=1)
  mdb.jobs[inpname].writeInput(consistencyChecking=OFF)
elif "nt" in os.name:
#     print 'WINDOWS'
  mdb.Job(name=inpname, model='Model-1', type=ANALYSIS,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
    description='', parallelizationMethodExplicit=DOMAIN,
    multiprocessingMode=DEFAULT, numDomains=1, userSubroutine='',
    numCpus=1, memory=50, memoryUnits=PERCENTAGE, scratch='',
    echoPrint=OFF, modelPrint=OFF, contactPrint=OFF,
```

```
      historyPrint=OFF)
  mdb.jobs[inpname].setValues( \
      userSubroutine='C:\\SIMULIA\\Abaqus_Temp\\umat3\\rate_ana.for')
  mdb.jobs[inpname].writeInput(consistencyChecking=OFF)
##############################################################################
# Open input file
##############################################################################
f = open(inpname+".inp",'r+')
read = str(f.readline())
while read[0:5] != '*Node':
  read = str(f.readline())
##############################################################################
# Generate list of nodes
##############################################################################
print 'Read nodes!'
nodes = []
data = []
read = str(f.readline())
while read[:8] != '*Element':
  data = read.split(",")
  data = map(float, data)
  data[0] = int(data[0])
  nodes.append(data)
  read = str(f.readline())
lennodes = len(nodes)
#print("Nodes")
#for line in nodes: print line
##############################################################################
# Generate list of elements
##############################################################################
print 'Read elements!'
elements = []
data = []
if str(eletyp)[0:4]=='C3D8':
    read = str(f.readline())
elif str(eletyp)[0:5]=='C3D10':
    read = str(f.readline())
elif str(eletyp)[0:5]=='C3D20':
    read = str(f.readline()) + str(f.readline())
else:
  print 'No such element type, check code'
while read[:5] != '*Nset':
  data = read.split(",")
  elements.append(map(int, data))
  if eval(eletyp)==C3D8:
    read = str(f.readline())
    if eval(eletyp)==C3D10:
    read = str(f.readline())
  if eval(eletyp)==C3D20:
    read = str(f.readline()) + str(f.readline())
  if eval(eletyp)==C3D20R:
    read = str(f.readline()) + str(f.readline())
print len(elements), 'Elemente'
print 'Sorting for LDBC and PBC ...'
##############################################################################
# Sorting surface node couples
##############################################################################
tol = 0.1 #*ele
stellen = 3
ecken = []

kantenXpYpZ = []
kantenXmYpZ = []
kantenXpYmZ = []
kantenXmYmZ = []
```

```
kantenYpXpZ = []
kantenYmXpZ = []
kantenYpXmZ = []
kantenYmXmZ = []

kantenZpXpY = []
kantenZmXpY = []
kantenZpXmY = []
kantenZmXmY = []

flaechenXm = []
flaechenXp = []
flaechenYm = []
flaechenYp = []
flaechenZm = []
flaechenZp = []

innen = []

for i in range(len(nodes)):
  #Ecke1
  if round(nodes[i][1],stellen) == round(XNUC*drve/2.,stellen) and \
    round(nodes[i][2],stellen) == round(YNUC*drve/2.,stellen) and \
    round(nodes[i][3],stellen) == round(ZNUC*drve/2.,stellen):
    ecken.append(nodes[i][0])
  #Ecke2
  elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
    round(nodes[i][2],stellen) == round(YNUC*drve/2. ,stellen) and \
    round(nodes[i][3],stellen) == round(ZNUC*drve/2. ,stellen):
    ecken.append(nodes[i][0])
  #Ecke3
  elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
    round(nodes[i][2],stellen) == round(YNUC*drve/2.      ,stellen) and \
    round(nodes[i][3],stellen) == round(ZNUC*-drve/2.       ,stellen):
    ecken.append(nodes[i][0])
  #Ecke4
  elif round(nodes[i][1],stellen) == round(drve/2.,stellen) and \
    round(nodes[i][2],stellen) == round(drve/2.      ,stellen) and \
    round(nodes[i][3],stellen) == round(-drve/2.       ,stellen):
    ecken.append(nodes[i][0])
  #Ecke5
  elif round(nodes[i][1],stellen) == round(drve/2.,stellen) and \
    round(nodes[i][2],stellen) == round(YNUC*-drve/2.     ,stellen) and \
    round(nodes[i][3],stellen) == round(ZNUC*-drve/2.       ,stellen):
    ecken.append(nodes[i][0])
  #Ecke6
  elif round(nodes[i][1],stellen) == round(XNUC*drve/2.,stellen) and \
    round(nodes[i][2],stellen) == round(YNUC*-drve/2.      ,stellen) and \
    round(nodes[i][3],stellen) == round(ZNUC*drve/2.       ,stellen):
    ecken.append(nodes[i][0])
  #Ecke7
  elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
    round(nodes[i][2],stellen) == round(YNUC*-drve/2.      ,stellen) and \
    round(nodes[i][3],stellen) == round(ZNUC*drve/2.       ,stellen):
    ecken.append(nodes[i][0])
  #Ecke8
  elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
    round(nodes[i][2],stellen) == round(YNUC*-drve/2.      ,stellen) and \
    round(nodes[i][3],stellen) == round(ZNUC*-drve/2.       ,stellen):
    ecken.append(nodes[i][0])

  #Kanten X+Y-Z
  elif round(nodes[i][1],stellen) != round(+drve/2.,stellen)  and \
    round(nodes[i][1],stellen) !=  round(-drve/2.,stellen)  and \
    round(nodes[i][2],stellen) ==  round(+drve/2.,stellen)  and \
    round(nodes[i][3],stellen) ==  round(-drve/2.,stellen):
```

```
        kantenXpYmZ.append(nodes[i][0])
    #Kanten X–Y–Z
    elif round(nodes[i][1],stellen) != round(+drve/2.,stellen) and \
      round(nodes[i][1],stellen) != round(-drve/2.,stellen) and \
      round(nodes[i][2],stellen) == round(-drve/2.,stellen) and \
      round(nodes[i][3],stellen) == round(-drve/2.,stellen):
      kantenXmYmZ.append(nodes[i][0])
    #Kanten X+Y+Z
    elif round(nodes[i][1],stellen) != round(+drve/2.,stellen) and \
      round(nodes[i][1],stellen) != round(-drve/2.,stellen) and \
      round(nodes[i][2],stellen) == round(+drve/2.,stellen) and \
      round(nodes[i][3],stellen) == round(+drve/2.,stellen):
      kantenXpYpZ.append(nodes[i][0])
    #Kanten X–Y+Z
    elif round(nodes[i][1],stellen) != round(+drve/2.,stellen) and \
      round(nodes[i][1],stellen) != round(-drve/2.,stellen) and \
      round(nodes[i][2],stellen) == round(-drve/2.,stellen) and \
      round(nodes[i][3],stellen) == round(+drve/2.,stellen):
      kantenXmYpZ.append(nodes[i][0])

    #Kanten Y+X–Z
      elif round(nodes[i][2],stellen) != round(+drve/2.,stellen) and \
      round(nodes[i][2],stellen) != round(-drve/2.,stellen) and \
      round(nodes[i][1],stellen) == round(+drve/2.,stellen) and \
      round(nodes[i][3],stellen) == round(-drve/2.,stellen):
      kantenYpXmZ.append(nodes[i][0])
    #Kanten Y–X–Z
    elif round(nodes[i][2],stellen) != round(+drve/2.,stellen) and \
      round(nodes[i][2],stellen) != round(-drve/2.,stellen) and \
      round(nodes[i][1],stellen) == round(-drve/2.,stellen) and \
      round(nodes[i][3],stellen) == round(-drve/2.,stellen):
      kantenYmXmZ.append(nodes[i][0])
    #Kanten Y+X+Z
    elif round(nodes[i][2],stellen) != round(+drve/2.,stellen) and \
      round(nodes[i][2],stellen) != round(-drve/2.,stellen) and \
      round(nodes[i][1],stellen) == round(+drve/2.,stellen) and \
      round(nodes[i][3],stellen) == round(+drve/2.,stellen):
      kantenYpXpZ.append(nodes[i][0])
    #Kanten Y–X+Z
    elif round(nodes[i][2],stellen) != round(+drve/2.,stellen) and \
      round(nodes[i][2],stellen) != round(-drve/2.,stellen) and \
      round(nodes[i][1],stellen) == round(-drve/2.,stellen) and \
      round(nodes[i][3],stellen) == round(+drve/2.,stellen):
      kantenYmXpZ.append(nodes[i][0])

    #Kanten  Z+X–Y
    elif round(nodes[i][3],stellen) != round(+drve/2.,stellen) and \
      round(nodes[i][3],stellen) != round(-drve/2.,stellen) and \
      round(nodes[i][1],stellen) == round(+drve/2.,stellen) and \
      round(nodes[i][2],stellen) == round(-drve/2.,stellen):
      kantenZpXmY.append(nodes[i][0])
    #Kanten  Z–X–Y
    elif round(nodes[i][3],stellen) != round(+drve/2.,stellen) and \
      round(nodes[i][3],stellen) != round(-drve/2.,stellen) and \
      round(nodes[i][1],stellen) == round(-drve/2.,stellen) and \
      round(nodes[i][2],stellen) == round(-drve/2.,stellen):
      kantenZmXmY.append(nodes[i][0])
    #Kanten  Z+X+Y
    elif round(nodes[i][3],stellen) != round(+drve/2.,stellen) and \
      round(nodes[i][3],stellen) != round(-drve/2.,stellen) and \
      round(nodes[i][1],stellen) == round(+drve/2.,stellen) and \
      round(nodes[i][2],stellen) == round(+drve/2.,stellen):
      kantenZpXpY.append(nodes[i][0])
    #Kanten  Z–X+Y
    elif round(nodes[i][3],stellen) != round(+drve/2.,stellen) and \
      round(nodes[i][3],stellen) != round(-drve/2.,stellen) and \
```

```
        round(nodes[i][1],stellen) ==  round(-drve/2.,stellen)  and \
        round(nodes[i][2],stellen) ==  round(+drve/2.,stellen):
        kantenZmXpY.append(nodes[i][0])
      # Minus (m) and Plus (p) side
      #Flaechen Xp
      elif round(nodes[i][1],stellen) == round(+drve/2.,stellen):
        flaechenXp.append(nodes[i][0])
      #Flaechen Xm
      elif round(nodes[i][1],stellen) == round(-drve/2.,stellen):
        flaechenXm.append(nodes[i][0])
      #Flaechen Yp
      elif round((nodes[i][2]),stellen) == round(+drve/2.,stellen):
        flaechenYp.append(nodes[i][0])
      #Flaechen Ym
      elif round((nodes[i][2]),stellen) == round(-drve/2.,stellen):
        flaechenYm.append(nodes[i][0])
      #Flaechen Zp
      elif round(nodes[i][3],stellen) == round(+drve/2.,stellen):
        flaechenZp.append(nodes[i][0])
      #Flaechen Zm
      elif round(nodes[i][3],stellen) == round(-drve/2.,stellen):
        flaechenZm.append(nodes[i][0])
      #Innen
      else:
        innen.append(nodes[i][0])
  ##################################################################
  # Additional sorting for PBC
  ##################################################################
  if rb == 'PBC':
    print 'Additional sorting for PBC ...'
    # Sorting opposite nodes within the list
    # Sorting corners
    eckensort = []

    for i in range(len(ecken)):
      #Ecke1
      if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(drve/2.,stellen) and \
        round(nodes[ecken[i]-1][3],stellen) == round(drve/2.,stellen):
        eckensort.append(ecken[i])
        break
    for i in range(len(ecken)):
      #Ecke2
      if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(drve/2. ,stellen) and \
        round(nodes[ecken[i]-1][3],stellen) == round(drve/2. ,stellen):
        eckensort.append(ecken[i])
        break
    eckensort.append(ecken[i])
    for i in range(len(ecken)):
      #Ecke3
      if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(drve/2.       ,stellen) and \
        round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.       ,stellen):
        eckensort.append(ecken[i])
        break
    eckensort.append(ecken[i])
    for i in range(len(ecken)):
      #Ecke4
      if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(drve/2.       ,stellen) and \
        round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.       ,stellen):
        eckensort.append(ecken[i])
        break
    eckensort.append(ecken[i])
    for i in range(len(ecken)):
```

```
    #Ecke5
    if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
      round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and
          \
      round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.      ,stellen):
      eckensort.append(ecken[i])
      break
  eckensort.append(ecken[i])
  for i in range(len(ecken)):
    #Ecke6
    if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
      round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and
          \
      round(nodes[ecken[i]-1][3],stellen) == round(drve/2.       ,stellen):
      eckensort.append(ecken[i])
      break
  eckensort.append(ecken[i])
  for i in range(len(ecken)):
    #Ecke7
    if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
      round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and
          \
      round(nodes[ecken[i]-1][3],stellen) == round(drve/2.       ,stellen):
      eckensort.append(ecken[i])
      break
  eckensort.append(ecken[i])
  for i in range(len(ecken)):
    #Ecke8
    if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
      round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and \
      round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.      ,stellen):
      eckensort.append(ecken[i])
      break
  # Sorting edges

  kantenXsort = []
  kantenYsort = []
  kantenZsort = []

  Xkanten = len(kantenXpYmZ)
  for i in range(Xkanten):
    Xvalue = round(nodes[kantenXpYmZ[i]-1][1],stellen)
    kantenXsort.append(kantenXpYmZ[i])
    for j in range(Xkanten):
      if Xvalue == round(nodes[kantenXmYmZ[j]-1][1],stellen):
        break
    kantenXsort.append(kantenXmYmZ[j])
    kantenXsort.append(kantenXmYmZ[j])

    for j in range(Xkanten):
      if Xvalue == round(nodes[kantenXmYpZ[j]-1][1],stellen):
        break
    kantenXsort.append(kantenXmYpZ[j])
    kantenXsort.append(kantenXmYpZ[j])

    for j in range(Xkanten):
      if Xvalue == round(nodes[kantenXpYpZ[j]-1][1],stellen):
        break
    kantenXsort.append(kantenXpYpZ[j])
  #print 'Kanten X sortiert'
  #print kantenXsort

  Ykanten = len(kantenYpXmZ)
  for i in range(Ykanten):
    Yvalue = round(nodes[kantenYpXmZ[i]-1][2],stellen)
    kantenYsort.append(kantenYpXmZ[i])
```

```
    for j in range(Ykanten):
      if Yvalue == round(nodes[kantenYmXmZ[j]-1][2],stellen):
        break
    kantenYsort.append(kantenYmXmZ[j])
    kantenYsort.append(kantenYmXmZ[j])

    for j in range(Ykanten):
      if Yvalue == round(nodes[kantenYmXpZ[j]-1][2],stellen):
        break
    kantenYsort.append(kantenYmXpZ[j])
    kantenYsort.append(kantenYmXpZ[j])

    for j in range(Ykanten):
      if Yvalue == round(nodes[kantenYpXpZ[j]-1][2],stellen):
        break
    kantenYsort.append(kantenYpXpZ[j])
  #print 'Kanten Y sortiert'
  #print kantenYsort

  Zkanten = len(kantenZpXmY)
  for i in range(Zkanten):
    Yvalue = round(nodes[kantenZpXmY[i]-1][3],stellen)
    kantenZsort.append(kantenZpXmY[i])

    for j in range(Zkanten):
      if Yvalue == round(nodes[kantenZmXmY[j]-1][3],stellen):
        break
    kantenZsort.append(kantenZmXmY[j])
    kantenZsort.append(kantenZmXmY[j])

    for j in range(Zkanten):
      if Yvalue == round(nodes[kantenZmXpY[j]-1][3],stellen):
        break
    kantenZsort.append(kantenZmXpY[j])
    kantenZsort.append(kantenZmXpY[j])

    for j in range(Zkanten):
      if Yvalue == round(nodes[kantenZpXpY[j]-1][3],stellen):
        break
    kantenZsort.append(kantenZpXpY[j])
  #print 'Kanten Z sortiert'
  #print kantenZsort

  flaechenXsort = []
  flaechenYsort = []
  flaechenZsort = []
  #stellen = 1
  Xflaechen = len(flaechenXp)
  for i in range(Xflaechen):
    Yvalue = nodes[flaechenXp[i]-1][2]
    Zvalue = nodes[flaechenXp[i]-1][3]
    flaechenXsort.append(flaechenXp[i])
    for j in range(Xflaechen):
      #print Yvalue - nodes[flaechenXm[j]-1][2]
      #print Zvalue - nodes[flaechenXm[j]-1][3]
      if abs(Yvalue - nodes[flaechenXm[j]-1][2]) <= tol*ele  and \
        abs(Zvalue - nodes[flaechenXm[j]-1][3]) <= tol*ele:
        break
    flaechenXsort.append(flaechenXm[j])
  #print 'Flaechen X sortiert'
  #print flaechenXsort

  Yflaechen = len(flaechenYp)
  for i in range(Yflaechen):
    Xvalue = nodes[flaechenYp[i]-1][1]
```

```
        Zvalue = nodes[flaechenYp[i]-1][3]
        flaechenYsort.append(flaechenYp[i])
        for j in range(Yflaechen):
          if abs(Xvalue - nodes[flaechenYm[j]-1][1]) <= tol*ele and \
            abs(Zvalue - nodes[flaechenYm[j]-1][3]) <= tol*ele:
            break
        flaechenYsort.append(flaechenYm[j])
      #print 'Flaechen Y sortiert'
      #print flaechenYsort

      Zflaechen = len(flaechenZp)
      for i in range(Zflaechen):
        Xvalue = nodes[flaechenZp[i]-1][1]
        Yvalue = nodes[flaechenZp[i]-1][2]
        flaechenZsort.append(flaechenZp[i])
        for j in range(Zflaechen):
          if abs(Xvalue - nodes[flaechenZm[j]-1][1]) <= tol*ele and \
            abs(Yvalue - nodes[flaechenZm[j]-1][2]) <= tol*ele:
            break
        flaechenZsort.append(flaechenZm[j])
      #print 'Flaechen Z sortiert'
      #print flaechenZsort
    if rb=='LDBC':
      print 'LDBC: Sorting nodes list'
      # Merge all lists
      nrf = []
      nrf.extend(ecken)
      nrf.extend(kantenXpYpZ)
      nrf.extend(kantenXmYpZ)
      nrf.extend(kantenXpYmZ)
      nrf.extend(kantenXmYmZ)
      nrf.extend(kantenYpXpZ)
      nrf.extend(kantenYmXpZ)
      nrf.extend(kantenYpXmZ)
      nrf.extend(kantenYmXmZ)
      nrf.extend(kantenZpXpY)
      nrf.extend(kantenZmXpY)
      nrf.extend(kantenZpXmY)
      nrf.extend(kantenZmXmY)
      nrf.extend(flaechenXm)
      nrf.extend(flaechenXp)
      nrf.extend(flaechenYm)
      nrf.extend(flaechenYp)
      nrf.extend(flaechenZm)
      nrf.extend(flaechenZp)
      #nrf.extend(innen)
      # Sorting nodes list following nrf
      ifortschritt = 0
      i = 0
      for i in range(len(nrf)):
        for k in range(len(nodes)):
          if nodes[k][0]==nrf[i]:
            break
        nodes[k],nodes[i] = nodes[i],nodes[k]
      letzteoberflaeche = len(nrf)-1
    elif rb=='PBC':
      print 'PBC: Sorting nodes list'
      # Merge all lists
      nrf = []
      nrf.extend(eckensort)
      nrf.extend(kantenXsort)
      nrf.extend(kantenYsort)
      nrf.extend(kantenZsort)
      nrf.extend(flaechenXsort)
      nrf.extend(flaechenYsort)
      nrf.extend(flaechenZsort)
```

```
    nrf.extend(innen)
    nodesneu = []
    #print 'len nrf',len(nrf)
    #print 'len nodes',len(nodes)
    i=0
    while i != len(nrf):
    #     if i % 1000 == 0:
    #         print 'i',i
      if i < len(nrf)-1:
        if nrf[i] == nrf[i+1]:
          for k in range(len(nodes)):
            if nodes[k][0] == nrf[i]:
              break
          nodesneu.append(nodes[k])
          nodesneu.append(nodes[k])
          nodes.pop(k)
          i+=2
        else:
          for k in range(len(nodes)):
            if nodes[k][0] == nrf[i]:
              break
          nodesneu.append(nodes[k])
          nodes.pop(k)
          i+=1
      else:
        for k in range(len(nodes)):
          if nodes[k][0] == nrf[i]:
            break
        nodesneu.append(nodes[k])
        nodes.pop(k)
        i+=1
    letzteoberflaeche = len(nrf)-len(innen)-1
    nodes = nodesneu
      #print("Nodesneu")
      #for line in nodes: print line
##############################################################################
# Write input file
##############################################################################
shutil.copyfile(inpname+".inp", inpname+"_backup.inp")
f.seek(0,0)
while read[:5] != '*Node':
  read = str(f.readline())
f.truncate()
f.write('%d,0.0,0.0,0.0 \n' %(lennodes+1))
f.write('%d,0.0,0.0,0.0 \n' %(lennodes+2))
f.write('%d,0.0,0.0,0.0 \n' %(lennodes+3))


b = open(inpname+"_backup.inp",'r+')
lines1 = b.readlines()
b.close()
os.remove(inpname+"_backup.inp")
for i in range(17,len(lines1)):
  f.write(lines1[i])

f.seek(0,0)

# Add additional nodes
print 'Complete input file ...'
while read[:11] != '** Section:':
  read = str(f.readline())
f.truncate()
if "posix" in os.name:
  f.seek(-17,2)
    #f.seek(-22,2)
```

```
    if XNUC==1 and YNUC==1 and ZNUC==1:
      f.write('** Section: Fiber1\n')
      f.write('*Solid Section, elset=_PickedSet9, material=Fiber1\n')
      f.write(',\n')
      f.write('** Section: Matrix\n')
      f.write('*Solid Section, elset=_PickedSet10, material=Matrix\n')
      f.write(',\n')
    elif XNUC==2 and YNUC==2 and ZNUC==2:
      f.write('** Section: Fiber1\n')
      f.write('*Solid Section, elset=_PickedSet4_#1, material=Fiber1\n')
      f.write(',\n')
      f.write('** Section: Fiber2\n')
      f.write('*Solid Section, elset=_PickedSet7_#2, material=Fiber2\n')
      f.write(',\n')
      f.write('** Section: Matrix\n')
      f.write('*Solid Section, elset=_PickedSet8_#3, material=Matrix\n')
      f.write(',\n')
    # Define equation
    f.write('**\n')
    f.write('** Constraint: Constraint-1 \n')
    f.write('**\n')
    f.write('*Equation \n')
    if rb == 'LDBC':
    ##########################################################################
    # Linear displacement boundary conditions (LDBC)
    ##########################################################################
      print 'Linear displacement boundary conditions (LDBC)!'
      #DOF1
      for i in range(0,letzteoberflaeche+1):
        f.write("4\n")
        f.write("%d,1,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
          %(nodes[i][0],len(nodes)+1,-nodes[i][1],len(nodes)+1, \
          -nodes[i][2],len(nodes)+1,-nodes[i][3]))
      #DOF2
      for i in range(0,letzteoberflaeche+1):
        f.write("4\n")
        f.write("%d,2,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
          %(nodes[i][0],len(nodes)+2,-nodes[i][1],len(nodes)+2, \
          -nodes[i][2],len(nodes)+2,-nodes[i][3]))
      #DOF3
      for i in range(0,letzteoberflaeche+1):
        f.write("4\n")
        f.write("%d,3,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
          %(nodes[i][0],len(nodes)+3,-nodes[i][1],len(nodes)+3, \
          -nodes[i][2],len(nodes)+3,-nodes[i][3]))

    elif rb == 'PBC':
    ##########################################################################
    # Periodic boundary conditions (PBC)
    ##########################################################################
      print 'Periodic boundary conditions (PBC)!'
      #DOF1
      for i in range(0,letzteoberflaeche+1,2):
        f.write("5\n")
        f.write("%d,1,1.0\n%d,1,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
          %(nodes[i][0], \
          nodes[i+1][0], \
          lennodes+1,(nodes[i+1][1]-nodes[i][1]), \
          lennodes+1,(nodes[i+1][2]-nodes[i][2]), \
          lennodes+1,(nodes[i+1][3]-nodes[i][3])))
      #DOF2
      for i in range(0,letzteoberflaeche+1,2):
        f.write("5\n")
        f.write("%d,2,1.0\n%d,2,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
          %(nodes[i][0], \
          nodes[i+1][0], \
```

```
            lennodes+2,(nodes[i+1][1]-nodes[i][1]), \
            lennodes+2,(nodes[i+1][2]-nodes[i][2]), \
            lennodes+2,(nodes[i+1][3]-nodes[i][3])))
        #DOF3
        for i in range(0,letzteoberflaeche+1,2):
            f.write("5\n")
            f.write("%d,3,1.0\n%d,3,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
            %(nodes[i][0], \
            nodes[i+1][0], \
            lennodes+3,(nodes[i+1][1]-nodes[i][1]), \
            lennodes+3,(nodes[i+1][2]-nodes[i][2]), \
            lennodes+3,(nodes[i+1][3]-nodes[i][3])))

    ##prescribe H symmetric
    #f.write("2\n")
    #f.write("%d,2,1.0\n%d,1,-1.0\n" %(lennodes+1, lennodes+2))
    #f.write("2\n")
    #f.write("%d,3,1.0\n%d,1,-1.0\n" %(lennodes+1, lennodes+3))
    #f.write("2\n")
    #f.write("%d,3,1.0\n%d,2,-1.0\n" %(lennodes+2, lennodes+3))

    f.write('*End Instance \n')
    #Ursprung bestimmen
    ursprung = 'kein'
    for i in range(0,len(nodes)):
        if abs(nodes[i][1]) == 0. and \
            abs(nodes[i][2]) == 0. and \
            abs(nodes[i][3]) == 0. <= tol*ele:
            ursprung=nodes[i][0]
    if ursprung == 'kein':
        print 'No origin, check mesh'

    #zusatz1 = 'kein'
    #for i in range(0,len(nodes)):
        #if abs(nodes[i][1] - (YNUC*fa1/tan(anglerad)/2.+XNUC*fa2)) <= tol*ele and \
            #abs(nodes[i][2] - (YNUC*fa1/2.)) <= tol*ele and \
            #abs(nodes[i][3] - (ZNUC*drve/2.)) <= tol*ele:
            #zusatz1=nodes[i][0]
    #if zusatz1 == 'kein':
        #print 'Kein Zusatz1, Vernetzung pruefen'

    #zusatz2 = 'kein'
    #for i in range(0,len(nodes)):
        #if abs(nodes[i][1] - (YNUC*fa1/tan(anglerad)/2.+XNUC*fa2/2.)) <= tol*ele and
            \
            #abs(nodes[i][2] - (YNUC*fa1/2.)) <= tol*ele and \
            #abs(nodes[i][3] - (ZNUC*drve)) <= tol*ele:
            #zusatz2=nodes[i][0]
    #if zusatz2 == 'kein':
        #print 'Kein Zusatz2, Vernetzung pruefen'

    # Node sets for boundary of the artificial nodes
    f.write('*Nset, nset=K1, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+1))
    f.write('*Nset, nset=K2, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+2))
    f.write('*Nset, nset=K3, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+3))

    if rb != 'LDBC':
        f.write('*Nset, nset=Ursprung, internal, instance=FIBBI-1 \n')
        f.write('%d, \n' %ursprung)

    #f.write('*Nset, nset=zusatz1, internal, instance=FIBBI-1 \n')
    #f.write('%d, \n' %zusatz1)
    #f.write('*Nset, nset=zusatz2, internal, instance=FIBBI-1 \n')
```

```
    #f.write('%d, \n' %zusatz2)

    f.write('*End Assembly \n')
    f.write('** \n')
    f.write('** MATERIALS\n')
    f.write('** \n')
# ==============================================================
# Element type T: 2 square fibers in x and y direction
#                 tetragonal symmetry
# ==============================================================
if typ == 'T':
    ##########################################################################
    print '============================================================'
    print 'Element type T: 2 square fibers in x and y direction'
    print '============================================================'
    ##########################################################################
    # Fiber radius
    rf1=3.
    rf2=3.
    rf3=3.
    # RVE thickness d
    drve= 10.
    # Number of unit cells
    XNUC=1
    YNUC=1
    ZNUC=1
    # Calculate COOS
    movex = -0.5*(XNUC-1.)
    movey = -sqrt(3.)/2.*(YNUC-1.)
    #Area A0
    #A0 = fa1*fa2*sin(anglerad)
    #A0 = fa1*YNUC*fa2*XNUC
    #Volume V0
    V0 = drve*drve*drve
    # Obsolet
    seed = []
    seed.append(XNUC)
    ##########################################################################
    # Generate faserinput.txt
    ##########################################################################
    inpname = 'RVE_%s_%s_%s_ele%s_%s' %(typ,'X'+str(XNUC)+'Y'+str(YNUC)+'Z'+str(
        ZNUC),rb,str(ele).replace('.',''),extname)
    print inpname
    f = open(inpname+'_fiberinput.txt','w')
    f.write('CPU number:\n')
    f.write('%s\n' %(cpunumber))
    f.write('Type of RVE:\n')
    f.write('%s\n' %(typ))
    f.write('Type of boundary condition:\n')
    f.write('%s\n' %(rb))
    f.write('Element size [ele]:\n')
    f.write('%.3f\n' %(ele))
    f.write('Element shape:\n')
    f.write('%s\n' %(eleshape))
    f.write('Element type:\n')
    f.write('%s\n' %(eletyp))
    f.write('Radius fiber 1:\n')
    f.write('%.3f\n' %(rf1))
    f.write('Distance fiber 1:\n')
    #f.write('%.3f\n' %(fa1))
    #f.write('Radius fiber 2:\n')
    #f.write('%.3f\n' %(rf2))
    #f.write('Distance fiber 2:\n')
    #f.write('%.3f\n' %(fa2))
    #f.write('Layer Distance:\n')
    #f.write('%.3f\n' %(la))
```

```
f.write('RVE thickness d:\n')
f.write('%.3f\n' %(drve))
#f.write('Angle between fibers:\n')
#f.write('%.3f\n' %(angle))
f.write('Displacement Gradient\n')
f.write('H0:\n')
f.write('%s,%s,%s\n' %(str(H0[0][0]),str(H0[0][1]),str(H0[0][2])))
f.write('%s,%s,%s\n' %(str(H0[1][0]),str(H0[1][1]),str(H0[1][2])))
f.write('%s,%s,%s\n' %(str(H0[2][0]),str(H0[2][1]),str(H0[2][2])))
f.write('Name Input-Datei\n')
f.write(inpname)
f.write('\n ')
f.close()
##########################################################################
# Generate model
##########################################################################
# Delete old model
Mdb()
session.viewports['Viewport: 1'].setValues(displayedObject=None)
session.journalOptions.setValues(replayGeometry=COORDINATE, recoverGeometry=
    COORDINATE)
# Section
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber1', material='Fiber1',
     thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber2', material='Fiber2',
     thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber3', material='Fiber3',
     thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Matrix', material='Matrix',
     thickness=None)
# Build rhombus and fibers
session.viewports['Viewport: 1'].partDisplay.setValues(sectionAssignments=OFF,
    engineeringFeatures=OFF)
session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
    referenceRepresentation=ON)
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=STANDALONE)
s.rectangle(point1=(-5.0, 5.0), point2=(5.0, -5.0))
p = mdb.models['Model-1'].Part(name='Part-1', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-1']
p.BaseSolidExtrude(sketch=s, depth=5.0)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-1']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']
session.viewports['Viewport: 1'].view.setValues(nearPlane=22.3108,
    farPlane=40.9522, width=31.4415, height=10.6695, cameraPosition=(
    28.8266, 4.06394, -9.87444), cameraUpVector=(-0.374061, 0.881389,
    0.288499))
p = mdb.models['Model-1'].parts['Part-1']
f, e = p.faces, p.edges
t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(-1.666667,
    1.666667, 0.0)), sketchUpEdge=e.findAt(coordinates=(-5.0, 2.5, 0.0)),
    sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, origin=(0.0, 0.0, 0.0))
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=28.28, gridSpacing=0.7, transform=t)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['Model-1'].parts['Part-1']
p.projectReferencesOntoSketch(sketch=s1, filter=COPLANAR_EDGES)
s1.rectangle(point1=(-5.0, 5.0), point2=(5.0, -5.0))
p = mdb.models['Model-1'].parts['Part-1']
f1, e1 = p.faces, p.edges
```

```
p.SolidExtrude(sketchPlane=f1.findAt(coordinates=(-1.666667, 1.666667, 0.0)),
    sketchUpEdge=e1.findAt(coordinates=(-5.0, 2.5, 0.0)),
    sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, sketch=s1, depth=5.0,
    flipExtrudeDirection=OFF)
s1.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])
p = mdb.models['Model-1'].parts['Part-1']
f, e, d1 = p.faces, p.edges, p.datums
t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(-1.666667, 5.0,
    3.333333)), sketchUpEdge=e.findAt(coordinates=(5.0, 5.0, -1.25)),
    sketchPlaneSide=SIDE1, origin=(0.0, 5.0, 0.0))
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=34.64, gridSpacing=0.86, transform=t)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['Model-1'].parts['Part-1']
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
s.rectangle(point1=(-2.5, 2.5), point2=(2.5, -2.5))
p = mdb.models['Model-1'].parts['Part-1']
f = p.faces
pickedFaces = f.findAt(((-1.666667, 5.0, 3.333333), ))
e1, d2 = p.edges, p.datums
p.PartitionFaceBySketch(sketchUpEdge=e1.findAt(coordinates=(5.0, 5.0, -1.25)),
    faces=pickedFaces, sketch=s)
s.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
pickedCells = c.findAt(((-0.833333, 5.0, 0.833333), ))
e, d1 = p.edges, p.datums
pickedEdges =(e.findAt(coordinates=(-2.5, 5.0, 1.25)), e.findAt(coordinates=(
    1.25, 5.0, 2.5)), e.findAt(coordinates=(2.5, 5.0, -1.25)), e.findAt(
    coordinates=(-1.25, 5.0, -2.5)))
p.PartitionCellByExtrudeEdge(line=e.findAt(coordinates=(5.0, 2.5, 5.0)),
    cells=pickedCells, edges=pickedEdges, sense=FORWARD)
p = mdb.models['Model-1'].parts['Part-1']
f1, e1, d2 = p.faces, p.edges, p.datums
t = p.MakeSketchTransform(sketchPlane=f1.findAt(coordinates=(5.0, 1.666667,
    3.333333)), sketchUpEdge=e1.findAt(coordinates=(5.0, -2.5, -5.0)),
    sketchPlaneSide=SIDE1, origin=(5.0, 0.0, 0.0))
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=34.64, gridSpacing=0.86, transform=t)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['Model-1'].parts['Part-1']
p.projectReferencesOntoSketch(sketch=s1, filter=COPLANAR_EDGES)
s1.rectangle(point1=(-2.5, 2.5), point2=(2.5, -2.5))
p = mdb.models['Model-1'].parts['Part-1']
f = p.faces
pickedFaces = f.findAt(((5.0, 1.666667, 3.333333), ))
e, d1 = p.edges, p.datums
p.PartitionFaceBySketch(sketchUpEdge=e.findAt(coordinates=(5.0, -2.5, -5.0)),
    faces=pickedFaces, sketch=s1)
s1.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
pickedCells = c.findAt(((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0,
    0.833333), ))
e1, d2 = p.edges, p.datums
pickedEdges =(e1.findAt(coordinates=(5.0, -1.25, 2.5)), e1.findAt(coordinates=(
    5.0, -2.5, -1.25)), e1.findAt(coordinates=(5.0, 1.25, -2.5)),
    e1.findAt(coordinates=(5.0, 2.5, 1.25)))
p.PartitionCellByExtrudeEdge(line=e1.findAt(coordinates=(2.5, -5.0, 5.0)),
    cells=pickedCells, edges=pickedEdges, sense=FORWARD)
```

```
    p = mdb.models['Model-1'].parts['Part-1']
    f, e, d1 = p.faces, p.edges, p.datums
    t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(1.666667, 1.666667,
        5.0)), sketchUpEdge=e.findAt(coordinates=(5.0, 2.5, 5.0)),
        sketchPlaneSide=SIDE1, origin=(0.0, 0.0, 5.0))
    s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
        sheetSize=34.64, gridSpacing=0.86, transform=t)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=SUPERIMPOSE)
    p = mdb.models['Model-1'].parts['Part-1']
    p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
    s.rectangle(point1=(-2.5, 2.5), point2=(2.5, -2.5))
    p = mdb.models['Model-1'].parts['Part-1']
    f = p.faces
    pickedFaces = f.findAt(((1.666667, 1.666667, 5.0), ))
    e1, d2 = p.edges, p.datums
    p.PartitionFaceBySketch(sketchUpEdge=e1.findAt(coordinates=(5.0, 2.5, 5.0)),
        faces=pickedFaces, sketch=s)
    s.unsetPrimaryObject()
    del mdb.models['Model-1'].sketches['__profile__']
    p = mdb.models['Model-1'].parts['Part-1']
    c = p.cells
    pickedCells = c.findAt(((-0.833333, -5.0, -0.833333), ), ((-0.833333, 0.833333,
        2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((-0.833333, -0.833333,
        5.0), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0, 0.833333), ))
    e, d1 = p.edges, p.datums
    pickedEdges =(e.findAt(coordinates=(-2.5, -1.25, 5.0)), e.findAt(coordinates=(
        1.25, -2.5, 5.0)), e.findAt(coordinates=(2.5, 1.25, 5.0)), e.findAt(
        coordinates=(-1.25, 2.5, 5.0)))
    p.PartitionCellByExtrudeEdge(line=e.findAt(coordinates=(5.0, -5.0, 1.25)),
        cells=pickedCells, edges=pickedEdges, sense=REVERSE)
    # Assign Section
    session.viewports['Viewport: 1'].partDisplay.setValues(sectionAssignments=ON,
        engineeringFeatures=ON)
    session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
        referenceRepresentation=OFF)
    p = mdb.models['Model-1'].parts['Part-1']
    session.viewports['Viewport: 1'].setValues(displayedObject=p)
    session.viewports['Viewport: 1'].view.setValues(session.views['Front'])
    session.viewports['Viewport: 1'].view.setProjection(projection=PARALLEL)
    p = mdb.models['Model-1'].parts['Part-1']
    c = p.cells
    # Tetragonal only x and y fiber
    cells = c.findAt(((-0.833333, -5.0, -0.833333), ), ((-0.833333, 0.833333,
        2.4995), ),
        ((-5.0, -0.833333, 0.833333), ), ((5.0, 0.833333, 0.833333), ), ((
        -0.833333, 5.0, 0.833333), ))
    # Original x, y and z fiber
#     cells = c.findAt(((-0.833333, 0.833333, -5.0), ), ((-0.833333, -0.833333, 5.0)
    ,
#         ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333, 0.833333, 2.4995), ),
#         ((-5.0, -0.833333, 0.833333), ), ((5.0, 0.833333, 0.833333), ), ((
#         -0.833333, 5.0, 0.833333), ))
    region = regionToolset.Region(cells=cells)
    p = mdb.models['Model-1'].parts['Part-1']
    p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
        offsetType=MIDDLE_SURFACE, offsetField='',
        thicknessAssignment=FROM_SECTION)
    p = mdb.models['Model-1'].parts['Part-1']
    c = p.cells
    cells = c.findAt(((-0.833333, 0.833333, -5.0), ), ((-0.833333, -0.833333, 5.0),
        ),
                     ((4.166667, -5.0, 2.5), ))
    region = regionToolset.Region(cells=cells)
    p = mdb.models['Model-1'].parts['Part-1']
    p.SectionAssignment(region=region, sectionName='Matrix', offset=0.0,
```

```
        offsetType=MIDDLE_SURFACE, offsetField='',
        thicknessAssignment=FROM_SECTION)
    session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])
    # Assembly
    a = mdb.models['Model-1'].rootAssembly
    session.viewports['Viewport: 1'].setValues(displayedObject=a)
    session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=OFF)
    session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
        meshTechnique=OFF)
    a1 = mdb.models['Model-1'].rootAssembly
    a1.DatumCsysByDefault(CARTESIAN)
    p = mdb.models['Model-1'].parts['Part-1']
    a1.Instance(name='FIBBI-1', part=p, dependent=OFF)
    session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=ON)
    session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
        meshTechnique=ON)
    # Partition
    a = mdb.models['Model-1'].rootAssembly
    c1 = a.instances['FIBBI-1'].cells
    pickedCells = c1.findAt(((-0.833333, 0.833333, -5.0), ), ((-0.833333,
        -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333,
        0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((4.166667,
        -5.0, 2.5), ), ((-0.833333, 5.0, 0.833333), ))
    v21 = a.instances['FIBBI-1'].vertices
    a.PartitionCellByPlaneThreePoints(point1=v21.findAt(coordinates=(-2.5, -2.5,
        5.0)), point2=v21.findAt(coordinates=(-2.5, 2.5, 5.0)),
        point3=v21.findAt(coordinates=(-2.5, 5.0, 2.5)), cells=pickedCells)
    a = mdb.models['Model-1'].rootAssembly
    c1 = a.instances['FIBBI-1'].cells
    pickedCells = c1.findAt(((-0.833333, 0.833333, -5.0), ), ((-0.833333,
        -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333,
        0.833333, 2.4995), ), ((-0.833333, -5.0, 3.333333), ), ((5.0, 0.833333,
        0.833333), ), ((-0.833333, 5.0, 0.833333), ))
    v1 = a.instances['FIBBI-1'].vertices
    a.PartitionCellByPlaneThreePoints(point1=v1.findAt(coordinates=(2.5, -2.5,
        5.0)), point2=v1.findAt(coordinates=(2.5, 2.5, 5.0)), point3=v1.findAt(
        coordinates=(2.5, 5.0, 2.5)), cells=pickedCells)
    a = mdb.models['Model-1'].rootAssembly
    c1 = a.instances['FIBBI-1'].cells
    pickedCells = c1.findAt(((0.833333, 4.166667, 5.0), ), ((0.833333, -5.0,
        4.166667), ), ((3.333333, -5.0, -4.166667), ), ((-3.333333, 5.0,
        -4.166667), ), ((-0.833333, -0.833333, 5.0), ), ((-0.833333, -5.0,
        -0.833333), ), ((-0.833333, 0.833333, 2.4995), ), ((-5.0, -0.833333,
        0.833333), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0,
        0.833333), ))
    v21 = a.instances['FIBBI-1'].vertices
    a.PartitionCellByPlaneThreePoints(point1=v21.findAt(coordinates=(5.0, -2.5,
        2.5)), point2=v21.findAt(coordinates=(5.0, 2.5, 2.5)),
        point3=v21.findAt(coordinates=(-2.5, 5.0, 2.5)), cells=pickedCells)
    a = mdb.models['Model-1'].rootAssembly
    c1 = a.instances['FIBBI-1'].cells
    pickedCells = c1.findAt(((0.833333, 5.0, -4.166667), ), ((4.166667, -5.0,
        1.666667), ), ((-4.166667, 5.0, 1.666667), ), ((-0.833333, 0.833333,
        -5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333, 0.833333,
        2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((0.833333, -4.166667,
        -5.0), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0, 0.833333),
        ))
    v1 = a.instances['FIBBI-1'].vertices
    a.PartitionCellByPlaneThreePoints(point1=v1.findAt(coordinates=(5.0, -2.5,
        -2.5)), point2=v1.findAt(coordinates=(5.0, 2.5, -2.5)),
        point3=v1.findAt(coordinates=(-2.5, 5.0, -2.5)), cells=pickedCells)
    a = mdb.models['Model-1'].rootAssembly
    c1 = a.instances['FIBBI-1'].cells
    pickedCells = c1.findAt(((-5.0, -4.166667, -1.666667), ), ((-4.166667, 5.0,
        -3.333333), ), ((4.166667, -5.0, -3.333333), ), ((-4.166667, -5.0,
        3.333333), ), ((4.166667, 5.0, 3.333333), ), ((0.833333, -3.333333,
```

```
    5.0), ), ((4.166667, -5.0, -1.666667), ), ((-0.833333, 0.833333, -5.0),
    ), ((-0.833333, -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), (
    (-0.833333, 0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((
    0.833333, -4.166667, -5.0), ), ((5.0, 0.833333, 0.833333), ))
v21 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v21.findAt(coordinates=(-2.5, -2.5,
    5.0)), point2=v21.findAt(coordinates=(2.5, -2.5, 5.0)),
    point3=v21.findAt(coordinates=(5.0, -2.5, 2.5)), cells=pickedCells)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((-5.0, -0.833333, 3.333333), ), ((3.333333, -0.833333,
    -5.0), ), ((-4.166667, 5.0, -3.333333), ), ((4.166667, 5.0, -1.666667),
    ), ((4.166667, 5.0, 3.333333), ), ((0.833333, 4.166667, 5.0), ), ((
    0.833333, 3.333333, -5.0), ), ((-5.0, 4.166667, -1.666667), ), ((
    -0.833333, 0.833333, -5.0), ), ((-0.833333, -0.833333, 5.0), ), ((
    -0.833333, 0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((
    5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0, 0.833333), ))
v1 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v1.findAt(coordinates=(-2.5, 2.5,
    5.0)), point2=v1.findAt(coordinates=(2.5, 2.5, 5.0)), point3=v1.findAt(
    coordinates=(5.0, 2.5, -2.5)), cells=pickedCells)
# Mesh
elemType1 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
elemType3 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
cells1 = c1.findAt(((5.0, 0.833333, 4.166667), ), ((-5.0, 0.833333, -4.166667),
    ), ((4.166667, 3.333333, -5.0), ), ((-5.0, 3.333333, 4.166667), ), ((
    3.333333, -5.0, 3.333333), ), ((-5.0, 0.833333, 3.333333), ), ((
    3.333333, 0.833333, -5.0), ), ((-5.0, -3.333333, -4.166667), ), ((
    -4.166667, -5.0, -1.666667), ), ((-4.166667, 5.0, -3.333333), ), ((
    4.166667, 5.0, -1.666667), ), ((5.0, -3.333333, -3.333333), ), ((
    -3.333333, -3.333333, 5.0), ), ((4.166667, 5.0, 3.333333), ), ((
    0.833333, 4.166667, 5.0), ), ((0.833333, 3.333333, -5.0), ), ((
    0.833333, -5.0, 4.166667), ), ((5.0, -4.166667, -1.666667), ), ((-5.0,
    4.166667, -1.666667), ), ((-0.833333, 0.833333, -5.0), ), ((-0.833333,
    -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333,
    0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((0.833333,
    -5.0, -3.333333), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0,
    0.833333), ))
pickedRegions =(cells1, )
a.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
    elemType3))
a = mdb.models['Model-1'].rootAssembly
partInstances =(a.instances['FIBBI-1'], )
a.seedPartInstance(regions=partInstances, size=1.25, deviationFactor=0.1)
a = mdb.models['Model-1'].rootAssembly
partInstances =(a.instances['FIBBI-1'], )
a.generateMesh(regions=partInstances)
##################################################################################
# Write input file
##################################################################################
#The name of the operating system dependent module imported.
#The following names have currently been registered:
#'posix', 'nt', 'os2', 'ce', 'java', 'riscos'.
if "posix" in os.name:
#     print 'LINUX'
##Cluster
#     mdb.Job(name=inpname, model='Model-1', type=ANALYSIS,
#          explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
#          description='', parallelizationMethodExplicit=DOMAIN,
#          multiprocessingMode=DEFAULT, numDomains=1, userSubroutine='',
#          numCpus=1, preMemory=16056.0, standardMemory=16056.0,
#          standardMemoryPolicy=MODERATE, scratch='', echoPrint=OFF,
#          modelPrint=OFF, contactPrint=OFF, historyPrint=OFF)
```

```
#Local
  mdb.Job(name=inpname, model='Model-1', description='', type= ANALYSIS,
    atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
    memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
    modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
    scratch='', parallelizationMethodExplicit=DOMAIN, numDomains=1,
    activateLoadBalancing=False, multiprocessingMode=DEFAULT, numCpus=1)
  mdb.jobs[inpname].writeInput(consistencyChecking=OFF)
elif "nt" in os.name:
#     print 'WINDOWS'
  mdb.Job(name=inpname, model='Model-1', type=ANALYSIS,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
    description='', parallelizationMethodExplicit=DOMAIN,
    multiprocessingMode=DEFAULT, numDomains=1, userSubroutine='',
    numCpus=1, memory=50, memoryUnits=PERCENTAGE, scratch='',
    echoPrint=OFF, modelPrint=OFF, contactPrint=OFF,
    historyPrint=OFF)
  mdb.jobs[inpname].setValues( \
    userSubroutine='C:\\SIMULIA\\Abaqus_Temp\\umat3\\rate_ana.for')
  mdb.jobs[inpname].writeInput(consistencyChecking=OFF)
##############################################################################
# Open input file
##############################################################################
f = open(inpname+".inp",'r+')
read = str(f.readline())
while read[0:5] != '*Node':
  read = str(f.readline())
##############################################################################
# Generate list of nodes
##############################################################################
print 'Read nodes!'
nodes = []
data = []
read = str(f.readline())
while read[:8] != '*Element':
  data = read.split(",")
  data = map(float, data)
  data[0] = int(data[0])
  nodes.append(data)
  read = str(f.readline())
lennodes = len(nodes)
#print("Nodes")
#for line in nodes: print line
##############################################################################
# Generate list of elements
##############################################################################
print 'Read elements!'
elements = []
data = []
if str(eletyp)[0:4]=='C3D8':
    read = str(f.readline())
elif str(eletyp)[0:5]=='C3D10':
    read = str(f.readline())
elif str(eletyp)[0:5]=='C3D20':
    read = str(f.readline()) + str(f.readline())
else:
  print 'No such element type! Check code'
while read[:5] != '*Nset':
  data = read.split(",")
  elements.append(map(int, data))
  if eval(eletyp)==C3D8:
    read = str(f.readline())
    if eval(eletyp)==C3D10:
    read = str(f.readline())
  if eval(eletyp)==C3D20:
```

```
        read = str(f.readline()) + str(f.readline())
    if eval(eletyp)==C3D20R:
        read = str(f.readline()) + str(f.readline())
print len(elements), 'Elemente'
print 'Sorting for LDBC and PBC ...'
##############################################################################
# Sorting surface node couples
##############################################################################
tol = 0.1 #*ele
stellen = 3
ecken = []

kantenXpYpZ = []
kantenXmYpZ = []
kantenXpYmZ = []
kantenXmYmZ = []

kantenYpXpZ = []
kantenYmXpZ = []
kantenYpXmZ = []
kantenYmXmZ = []

kantenZpXpY = []
kantenZmXpY = []
kantenZpXmY = []
kantenZmXmY = []

flaechenXm = []
flaechenXp = []
flaechenYm = []
flaechenYp = []
flaechenZm = []
flaechenZp = []

innen = []

for i in range(len(nodes)):
    #Ecke1
    if round(nodes[i][1],stellen) == round(XNUC*drve/2.,stellen) and \
       round(nodes[i][2],stellen) == round(YNUC*drve/2.,stellen) and \
       round(nodes[i][3],stellen) == round(ZNUC*drve/2.,stellen):
        ecken.append(nodes[i][0])
    #Ecke2
    elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
       round(nodes[i][2],stellen) == round(YNUC*drve/2. ,stellen) and \
       round(nodes[i][3],stellen) == round(ZNUC*drve/2. ,stellen):
        ecken.append(nodes[i][0])
    #Ecke3
    elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
       round(nodes[i][2],stellen) == round(YNUC*drve/2.      ,stellen) and \
       round(nodes[i][3],stellen) == round(ZNUC*-drve/2.       ,stellen):
        ecken.append(nodes[i][0])
    #Ecke4
    elif round(nodes[i][1],stellen) == round(drve/2.,stellen) and \
       round(nodes[i][2],stellen) == round(drve/2.      ,stellen) and \
       round(nodes[i][3],stellen) == round(-drve/2.       ,stellen):
        ecken.append(nodes[i][0])
    #Ecke5
    elif round(nodes[i][1],stellen) == round(drve/2.,stellen) and \
       round(nodes[i][2],stellen) == round(YNUC*-drve/2.    ,stellen) and \
       round(nodes[i][3],stellen) == round(ZNUC*-drve/2.      ,stellen):
        ecken.append(nodes[i][0])
    #Ecke6
    elif round(nodes[i][1],stellen) == round(XNUC*drve/2.,stellen) and \
       round(nodes[i][2],stellen) == round(YNUC*-drve/2.      ,stellen) and \
       round(nodes[i][3],stellen) == round(ZNUC*drve/2.       ,stellen):
```

```
        ecken.append(nodes[i][0])
    #Ecke7
    elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
      round(nodes[i][2],stellen) == round(YNUC*-drve/2.       ,stellen) and \
      round(nodes[i][3],stellen) == round(ZNUC*drve/2.        ,stellen):
      ecken.append(nodes[i][0])
    #Ecke8
    elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
      round(nodes[i][2],stellen) == round(YNUC*-drve/2.       ,stellen) and \
      round(nodes[i][3],stellen) == round(ZNUC*-drve/2.        ,stellen):
      ecken.append(nodes[i][0])

    #Kanten X+Y-Z
    elif round(nodes[i][1],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][1],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][2],stellen) ==  round(+drve/2.,stellen)  and \
      round(nodes[i][3],stellen) ==  round(-drve/2.,stellen):
      kantenXpYmZ.append(nodes[i][0])
    #Kanten X-Y-Z
    elif round(nodes[i][1],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][1],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][2],stellen) ==  round(-drve/2.,stellen)  and \
      round(nodes[i][3],stellen) ==  round(-drve/2.,stellen):
      kantenXmYmZ.append(nodes[i][0])
    #Kanten X+Y+Z
    elif round(nodes[i][1],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][1],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][2],stellen) ==  round(+drve/2.,stellen)  and \
      round(nodes[i][3],stellen) ==  round(+drve/2.,stellen):
      kantenXpYpZ.append(nodes[i][0])
    #Kanten X-Y+Z
    elif round(nodes[i][1],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][1],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][2],stellen) ==  round(-drve/2.,stellen)  and \
      round(nodes[i][3],stellen) ==  round(+drve/2.,stellen):
      kantenXmYpZ.append(nodes[i][0])

    #Kanten Y+X-Z
      elif round(nodes[i][2],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][2],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][1],stellen) ==  round(+drve/2.,stellen)  and \
      round(nodes[i][3],stellen) ==  round(-drve/2.,stellen):
      kantenYpXmZ.append(nodes[i][0])
    #Kanten Y-X-Z
    elif round(nodes[i][2],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][2],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][1],stellen) ==  round(-drve/2.,stellen)  and \
      round(nodes[i][3],stellen) ==  round(-drve/2.,stellen):
      kantenYmXmZ.append(nodes[i][0])
    #Kanten Y+X+Z
    elif round(nodes[i][2],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][2],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][1],stellen) ==  round(+drve/2.,stellen)  and \
      round(nodes[i][3],stellen) ==  round(+drve/2.,stellen):
      kantenYpXpZ.append(nodes[i][0])
    #Kanten Y-X+Z
    elif round(nodes[i][2],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][2],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][1],stellen) ==  round(-drve/2.,stellen)  and \
      round(nodes[i][3],stellen) ==  round(+drve/2.,stellen):
      kantenYmXpZ.append(nodes[i][0])

    #Kanten  Z+X-Y
    elif round(nodes[i][3],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][3],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][1],stellen) ==  round(+drve/2.,stellen)  and \
```

```
        round(nodes[i][2],stellen) == round(-drve/2.,stellen):
          kantenZpXmY.append(nodes[i][0])
      #Kanten  Z-X-Y
      elif round(nodes[i][3],stellen) != round(+drve/2.,stellen) and \
        round(nodes[i][3],stellen) != round(-drve/2.,stellen) and \
        round(nodes[i][1],stellen) == round(-drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(-drve/2.,stellen):
        kantenZmXmY.append(nodes[i][0])
      #Kanten  Z+X+Y
      elif round(nodes[i][3],stellen) != round(+drve/2.,stellen) and \
        round(nodes[i][3],stellen) != round(-drve/2.,stellen) and \
        round(nodes[i][1],stellen) == round(+drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(+drve/2.,stellen):
        kantenZpXpY.append(nodes[i][0])
      #Kanten  Z-X+Y
      elif round(nodes[i][3],stellen) != round(+drve/2.,stellen) and \
        round(nodes[i][3],stellen) != round(-drve/2.,stellen) and \
        round(nodes[i][1],stellen) == round(-drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(+drve/2.,stellen):
        kantenZmXpY.append(nodes[i][0])
      #Jeweils Minus (m) und Plus (p) Seite
      #Flaechen Xp
      elif round(nodes[i][1],stellen) == round(+drve/2.,stellen):
        flaechenXp.append(nodes[i][0])
      #Flaechen Xm
      elif round(nodes[i][1],stellen) == round(-drve/2.,stellen):
        flaechenXm.append(nodes[i][0])
      #Flaechen Yp
      elif round((nodes[i][2]),stellen) == round(+drve/2.,stellen):
        flaechenYp.append(nodes[i][0])
      #Flaechen Ym
      elif round((nodes[i][2]),stellen) == round(-drve/2.,stellen):
        flaechenYm.append(nodes[i][0])
      #Flaechen Zp
      elif round(nodes[i][3],stellen) == round(+drve/2.,stellen):
        flaechenZp.append(nodes[i][0])
      #Flaechen Zm
      elif round(nodes[i][3],stellen) == round(-drve/2.,stellen):
        flaechenZm.append(nodes[i][0])
      #Innen
      else:
        innen.append(nodes[i][0])
  ##################################################################
  # Additional sorting for PBC
  ##################################################################
  if rb == 'PBC':
    print 'Additional sorting for PBC ...'
    # Sorting opposite nodes within the list
    # Sorting corners
    eckensort = []

    for i in range(len(ecken)):
      #Ecke1
      if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(drve/2.,stellen) and \
        round(nodes[ecken[i]-1][3],stellen) == round(drve/2.,stellen):
        eckensort.append(ecken[i])
        break
    for i in range(len(ecken)):
      #Ecke2
      if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(drve/2. ,stellen) and \
        round(nodes[ecken[i]-1][3],stellen) == round(drve/2. ,stellen):
        eckensort.append(ecken[i])
        break
    eckensort.append(ecken[i])
```

```
for i in range(len(ecken)):
  #Ecke3
  if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
    round(nodes[ecken[i]-1][2],stellen) == round(drve/2.      ,stellen) and \
    round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.      ,stellen):
    eckensort.append(ecken[i])
    break
eckensort.append(ecken[i])
for i in range(len(ecken)):
  #Ecke4
  if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
    round(nodes[ecken[i]-1][2],stellen) == round(drve/2.      ,stellen) and \
    round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.      ,stellen):
    eckensort.append(ecken[i])
    break
eckensort.append(ecken[i])
for i in range(len(ecken)):
  #Ecke5
  if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
    round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and
        \
    round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.      ,stellen):
    eckensort.append(ecken[i])
    break
eckensort.append(ecken[i])
for i in range(len(ecken)):
  #Ecke6
  if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
    round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and
        \
    round(nodes[ecken[i]-1][3],stellen) == round(drve/2.      ,stellen):
    eckensort.append(ecken[i])
    break
eckensort.append(ecken[i])
for i in range(len(ecken)):
  #Ecke7
  if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
    round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and
        \
    round(nodes[ecken[i]-1][3],stellen) == round(drve/2.      ,stellen):
    eckensort.append(ecken[i])
    break
eckensort.append(ecken[i])
for i in range(len(ecken)):
  #Ecke8
  if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
    round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and \
    round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.      ,stellen):
    eckensort.append(ecken[i])
    break
# Sorting edges

kantenXsort = []
kantenYsort = []
kantenZsort = []

Xkanten = len(kantenXpYmZ)
for i in range(Xkanten):
  Xvalue = round(nodes[kantenXpYmZ[i]-1][1],stellen)
  kantenXsort.append(kantenXpYmZ[i])
  for j in range(Xkanten):
    if Xvalue == round(nodes[kantenXmYmZ[j]-1][1],stellen):
      break
  kantenXsort.append(kantenXmYmZ[j])
  kantenXsort.append(kantenXmYmZ[j])
```

```
    for j in range(Xkanten):
      if Xvalue == round(nodes[kantenXmYpZ[j]-1][1],stellen):
        break
    kantenXsort.append(kantenXmYpZ[j])
    kantenXsort.append(kantenXmYpZ[j])

    for j in range(Xkanten):
      if Xvalue == round(nodes[kantenXpYpZ[j]-1][1],stellen):
        break
    kantenXsort.append(kantenXpYpZ[j])
  #print 'Kanten X sortiert'
  #print kantenXsort

  Ykanten = len(kantenYpXmZ)
  for i in range(Ykanten):
    Yvalue = round(nodes[kantenYpXmZ[i]-1][2],stellen)
    kantenYsort.append(kantenYpXmZ[i])

    for j in range(Ykanten):
      if Yvalue == round(nodes[kantenYmXmZ[j]-1][2],stellen):
        break
    kantenYsort.append(kantenYmXmZ[j])
    kantenYsort.append(kantenYmXmZ[j])

    for j in range(Ykanten):
      if Yvalue == round(nodes[kantenYmXpZ[j]-1][2],stellen):
        break
    kantenYsort.append(kantenYmXpZ[j])
    kantenYsort.append(kantenYmXpZ[j])

    for j in range(Ykanten):
      if Yvalue == round(nodes[kantenYpXpZ[j]-1][2],stellen):
        break
    kantenYsort.append(kantenYpXpZ[j])
  #print 'Kanten Y sortiert'
  #print kantenYsort

  Zkanten = len(kantenZpXmY)
  for i in range(Zkanten):
    Yvalue = round(nodes[kantenZpXmY[i]-1][3],stellen)
    kantenZsort.append(kantenZpXmY[i])

    for j in range(Zkanten):
      if Yvalue == round(nodes[kantenZmXmY[j]-1][3],stellen):
        break
    kantenZsort.append(kantenZmXmY[j])
    kantenZsort.append(kantenZmXmY[j])

    for j in range(Zkanten):
      if Yvalue == round(nodes[kantenZmXpY[j]-1][3],stellen):
        break
    kantenZsort.append(kantenZmXpY[j])
    kantenZsort.append(kantenZmXpY[j])

    for j in range(Zkanten):
      if Yvalue == round(nodes[kantenZpXpY[j]-1][3],stellen):
        break
    kantenZsort.append(kantenZpXpY[j])
  #print 'Kanten Z sortiert'
  #print kantenZsort

  flaechenXsort = []
  flaechenYsort = []
  flaechenZsort = []
  #stellen = 1
  Xflaechen = len(flaechenXp)
```

```
    for i in range(Xflaechen):
      Yvalue = nodes[flaechenXp[i]-1][2]
      Zvalue = nodes[flaechenXp[i]-1][3]
      flaechenXsort.append(flaechenXp[i])
      for j in range(Xflaechen):
        #print Yvalue - nodes[flaechenXm[j]-1][2]
        #print Zvalue - nodes[flaechenXm[j]-1][3]
        if abs(Yvalue - nodes[flaechenXm[j]-1][2]) <= tol*ele  and \
          abs(Zvalue - nodes[flaechenXm[j]-1][3]) <= tol*ele:
          break
      flaechenXsort.append(flaechenXm[j])
    #print 'Flaechen X sortiert'
    #print flaechenXsort

    Yflaechen = len(flaechenYp)
    for i in range(Yflaechen):
      Xvalue = nodes[flaechenYp[i]-1][1]
      Zvalue = nodes[flaechenYp[i]-1][3]
      flaechenYsort.append(flaechenYp[i])
      for j in range(Yflaechen):
        if abs(Xvalue - nodes[flaechenYm[j]-1][1]) <= tol*ele and \
          abs(Zvalue - nodes[flaechenYm[j]-1][3]) <= tol*ele:
          break
      flaechenYsort.append(flaechenYm[j])
    #print 'Flaechen Y sortiert'
    #print flaechenYsort

    Zflaechen = len(flaechenZp)
    for i in range(Zflaechen):
      Xvalue = nodes[flaechenZp[i]-1][1]
      Yvalue = nodes[flaechenZp[i]-1][2]
      flaechenZsort.append(flaechenZp[i])
      for j in range(Zflaechen):
        if abs(Xvalue - nodes[flaechenZm[j]-1][1]) <= tol*ele and \
          abs(Yvalue - nodes[flaechenZm[j]-1][2]) <= tol*ele:
          break
      flaechenZsort.append(flaechenZm[j])
    #print 'Flaechen Z sortiert'
    #print flaechenZsort
  if rb=='LDBC':
    print 'LDBC: Sorting nodes list again'
    # Merge all lists
    nrf = []
    nrf.extend(ecken)
    nrf.extend(kantenXpYpZ)
    nrf.extend(kantenXmYpZ)
    nrf.extend(kantenXpYmZ)
    nrf.extend(kantenXmYmZ)
    nrf.extend(kantenYpXpZ)
    nrf.extend(kantenYmXpZ)
    nrf.extend(kantenYpXmZ)
    nrf.extend(kantenYmXmZ)
    nrf.extend(kantenZpXpY)
    nrf.extend(kantenZmXpY)
    nrf.extend(kantenZpXmY)
    nrf.extend(kantenZmXmY)
    nrf.extend(flaechenXm)
    nrf.extend(flaechenXp)
    nrf.extend(flaechenYm)
    nrf.extend(flaechenYp)
    nrf.extend(flaechenZm)
    nrf.extend(flaechenZp)
    #nrf.extend(innen)
    #Sorting nodes list corresponding to nrf
    ifortschritt = 0
    i = 0
```

```
    for i in range(len(nrf)):
      for k in range(len(nodes)):
        if nodes[k][0]==nrf[i]:
          break
      nodes[k],nodes[i] = nodes[i],nodes[k]
    letzteoberflaeche = len(nrf)-1
  elif rb=='PBC':
    print 'PBC: Sorting nodes list'
    # Merge all lists
    nrf = []
    nrf.extend(eckensort)
    nrf.extend(kantenXsort)
    nrf.extend(kantenYsort)
    nrf.extend(kantenZsort)
    nrf.extend(flaechenXsort)
    nrf.extend(flaechenYsort)
    nrf.extend(flaechenZsort)
    nrf.extend(innen)
    nodesneu = []
    #print 'len nrf',len(nrf)
    #print 'len nodes',len(nodes)
    i=0
    while i != len(nrf):
    #     if i % 1000 == 0:
    #         print 'i',i
      if i < len(nrf)-1:
        if nrf[i] == nrf[i+1]:
          for k in range(len(nodes)):
            if nodes[k][0] == nrf[i]:
              break
          nodesneu.append(nodes[k])
          nodesneu.append(nodes[k])
          nodes.pop(k)
          i+=2
        else:
          for k in range(len(nodes)):
            if nodes[k][0] == nrf[i]:
              break
          nodesneu.append(nodes[k])
          nodes.pop(k)
          i+=1
      else:
        for k in range(len(nodes)):
          if nodes[k][0] == nrf[i]:
            break
        nodesneu.append(nodes[k])
        nodes.pop(k)
        i+=1
    letzteoberflaeche = len(nrf)-len(innen)-1
    nodes = nodesneu
      #print("Nodesneu")
      #for line in nodes: print line
##############################################################################
# Write input file
##############################################################################
shutil.copyfile(inpname+".inp", inpname+"_backup.inp")
f.seek(0,0)
while read[:5] != '*Node':
  read = str(f.readline())
f.truncate()
f.write('%d,0.0,0.0,0.0 \n' %(lennodes+1))
f.write('%d,0.0,0.0,0.0 \n' %(lennodes+2))
f.write('%d,0.0,0.0,0.0 \n' %(lennodes+3))

b = open(inpname+"_backup.inp",'r+')
lines1 = b.readlines()
```

```
    b.close()
    os.remove(inpname+"_backup.inp")
    for i in range(17,len(lines1)):
      f.write(lines1[i])
    f.seek(0,0)

    # Add additional nodes
    print 'Complete input file ...'
    while read[:11] != '** Section:':
      read = str(f.readline())
    f.truncate()
    if "posix" in os.name:
      f.seek(-17,2)
        #f.seek(-22,2)

    if XNUC==1 and YNUC==1 and ZNUC==1:
      f.write('** Section: Fiber1\n')
      f.write('*Solid Section, elset=_PickedSet9, material=Fiber1\n')
      f.write(',\n')
      f.write('** Section: Matrix\n')
      f.write('*Solid Section, elset=_PickedSet10, material=Matrix\n')
      f.write(',\n')
    elif XNUC==2 and YNUC==2 and ZNUC==2:
      f.write('** Section: Fiber1\n')
      f.write('*Solid Section, elset=_PickedSet4_#1, material=Fiber1\n')
      f.write(',\n')
      f.write('** Section: Fiber2\n')
      f.write('*Solid Section, elset=_PickedSet7_#2, material=Fiber2\n')
      f.write(',\n')
      f.write('** Section: Matrix\n')
      f.write('*Solid Section, elset=_PickedSet8_#3, material=Matrix\n')
      f.write(',\n')
    # Define equation
    f.write('**\n')
    f.write('** Constraint: Constraint-1 \n')
    f.write('**\n')
    f.write('*Equation \n')
    if rb == 'LDBC':
    ##########################################################################
    # Linear displacment boundary conditions (LDBC)
    ##########################################################################
      print 'Linear displacment boundary conditions (LDBC)!'
      #DOF1
      for i in range(0,letzteoberflaeche+1):
        f.write("4\n")
        f.write("%d,1,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
          %(nodes[i][0],len(nodes)+1,-nodes[i][1],len(nodes)+1, \
          -nodes[i][2],len(nodes)+1,-nodes[i][3]))
      #DOF2
      for i in range(0,letzteoberflaeche+1):
        f.write("4\n")
        f.write("%d,2,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
          %(nodes[i][0],len(nodes)+2,-nodes[i][1],len(nodes)+2, \
          -nodes[i][2],len(nodes)+2,-nodes[i][3]))
      #DOF3
      for i in range(0,letzteoberflaeche+1):
        f.write("4\n")
        f.write("%d,3,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
          %(nodes[i][0],len(nodes)+3,-nodes[i][1],len(nodes)+3, \
          -nodes[i][2],len(nodes)+3,-nodes[i][3]))

    elif rb == 'PBC':
    ##########################################################################
    # Periodic boundary conditions (PBC)
    ##########################################################################
      print 'Periodic boundary conditions (PBC)'
```

```
    #DOF1
    for i in range(0,letzteoberflaeche+1,2):
      f.write("5\n")
      f.write("%d,1,1.0\n%d,1,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
        %(nodes[i][0], \
        nodes[i+1][0], \
        lennodes+1,(nodes[i+1][1]-nodes[i][1]), \
        lennodes+1,(nodes[i+1][2]-nodes[i][2]), \
        lennodes+1,(nodes[i+1][3]-nodes[i][3])))
    #DOF2
    for i in range(0,letzteoberflaeche+1,2):
      f.write("5\n")
      f.write("%d,2,1.0\n%d,2,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
        %(nodes[i][0], \
        nodes[i+1][0], \
        lennodes+2,(nodes[i+1][1]-nodes[i][1]), \
        lennodes+2,(nodes[i+1][2]-nodes[i][2]), \
        lennodes+2,(nodes[i+1][3]-nodes[i][3])))
    #DOF3
    for i in range(0,letzteoberflaeche+1,2):
      f.write("5\n")
      f.write("%d,3,1.0\n%d,3,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
        %(nodes[i][0], \
        nodes[i+1][0], \
        lennodes+3,(nodes[i+1][1]-nodes[i][1]), \
        lennodes+3,(nodes[i+1][2]-nodes[i][2]), \
        lennodes+3,(nodes[i+1][3]-nodes[i][3])))

  ##prescribe H symmetric
  #f.write("2\n")
  #f.write("%d,2,1.0\n%d,1,-1.0\n" %(lennodes+1, lennodes+2))
  #f.write("2\n")
  #f.write("%d,3,1.0\n%d,1,-1.0\n" %(lennodes+1, lennodes+3))
  #f.write("2\n")
  #f.write("%d,3,1.0\n%d,2,-1.0\n" %(lennodes+2, lennodes+3))

  f.write('*End Instance \n')
  # Determine origin
  ursprung = 'kein'
  for i in range(0,len(nodes)):
    if abs(nodes[i][1]) == 0. and \
      abs(nodes[i][2]) == 0. and \
      abs(nodes[i][3]) == 0. <= tol*ele:
      ursprung=nodes[i][0]
  if ursprung == 'kein':
    print 'No origin, check mesh'

  #zusatz1 = 'kein'
  #for i in range(0,len(nodes)):
    #if abs(nodes[i][1] - (YNUC*fa1/tan(anglerad)/2.+XNUC*fa2)) <= tol*ele and \
      #abs(nodes[i][2] - (YNUC*fa1/2.)) <= tol*ele and \
      #abs(nodes[i][3] - (ZNUC*drve/2.)) <= tol*ele:
      #zusatz1=nodes[i][0]
  #if zusatz1 == 'kein':
    #print 'Kein Zusatz1, Vernetzung pruefen'

  #zusatz2 = 'kein'
  #for i in range(0,len(nodes)):
    #if abs(nodes[i][1] - (YNUC*fa1/tan(anglerad)/2.+XNUC*fa2/2.)) <= tol*ele and
        \
      #abs(nodes[i][2] - (YNUC*fa1/2.)) <= tol*ele and \
      #abs(nodes[i][3] - (ZNUC*drve)) <= tol*ele:
      #zusatz2=nodes[i][0]
  #if zusatz2 == 'kein':
    #print 'Kein Zusatz2, Vernetzung pruefen'
```

```
    # Node sets for the boundary of the artificial nodes
    f.write('*Nset, nset=K1, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+1))
    f.write('*Nset, nset=K2, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+2))
    f.write('*Nset, nset=K3, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+3))

    if rb != 'LDBC':
        f.write('*Nset, nset=Ursprung, internal, instance=FIBBI-1 \n')
        f.write('%d, \n' %ursprung)

    #f.write('*Nset, nset=zusatz1, internal, instance=FIBBI-1 \n')
    #f.write('%d, \n' %zusatz1)
    #f.write('*Nset, nset=zusatz2, internal, instance=FIBBI-1 \n')
    #f.write('%d, \n' %zusatz2)

    f.write('*End Assembly \n')
    f.write('** \n')
    f.write('** MATERIALS\n')
    f.write('** \n')
# ==============================================================
# Element type C: 3 square fibers in x, y and z direction
#                 cubic symmetry
# ==============================================================
if typ == 'C':
    ##############################################################################
    print '============================================================'
    print 'Element type C: 3 square fibers in x, y and z direction'
    print '============================================================'
    ##############################################################################
    #Fiber radius
    rf1=3.
    rf2=3.
    rf3=3.
    #RVE thickness d
    drve= 10.
    #Number of unit cells
    XNUC=1
    YNUC=1
    ZNUC=1
    #Calculate COOS
    movex = -0.5*(XNUC-1.)
    movey = -sqrt(3.)/2.*(YNUC-1.)
    #Area A0
    #A0 = fa1*fa2*sin(anglerad)
    #A0 = fa1*YNUC*fa2*XNUC
    #Volume V0
    V0 = drve*drve*drve
    #Obsolet
    seed = []
    seed.append(XNUC)
    ##############################################################################
    # Generate faserinput.txt
    ##############################################################################
    inpname = 'RVE_%s_%s_%s_ele%s_%s' %(typ,'X'+str(XNUC)+'Y'+str(YNUC)+'Z'+str(
        ZNUC),rb,str(ele).replace('.',''),extname)
    print inpname
    f = open(inpname+'_fiberinput.txt','w' )
    f.write('CPU number:\n')
    f.write('%s\n' %(cpunumber))
    f.write('Type of RVE:\n')
    f.write('%s\n' %(typ))
    f.write('Type of boundary condition:\n')
    f.write('%s\n' %(rb))
    f.write('Element size:\n')
```

```
f.write('%.3f\n' %(ele))
f.write('Element shape:\n')
f.write('%s\n' %(eleshape))
f.write('Element type:\n')
f.write('%s\n' %(eletyp))
f.write('Radius fiber 1:\n')
f.write('%.3f\n' %(rf1))
f.write('Distance fiber 1:\n')
#f.write('%.3f\n' %(fa1))
#f.write('Radius fiber 2:\n')
#f.write('%.3f\n' %(rf2))
#f.write('Distance fiber 2:\n')
#f.write('%.3f\n' %(fa2))
#f.write('Layer Distance:\n')
#f.write('%.3f\n' %(la))
f.write('RVE thickness d:\n')
f.write('%.3f\n' %(drve))
#f.write('Angle between fibers:\n')
#f.write('%.3f\n' %(angle))
f.write('Displacement Gradient\n')
f.write('H0:\n')
f.write('%s,%s,%s\n' %(str(H0[0][0]),str(H0[0][1]),str(H0[0][2])))
f.write('%s,%s,%s\n' %(str(H0[1][0]),str(H0[1][1]),str(H0[1][2])))
f.write('%s,%s,%s\n' %(str(H0[2][0]),str(H0[2][1]),str(H0[2][2])))
f.write('Name Input-Datei\n')
f.write(inpname)
f.write('\n ')
f.close()
##########################################################################
# Generate model
##########################################################################
# Close old model
Mdb()
session.viewports['Viewport: 1'].setValues(displayedObject=None)
session.journalOptions.setValues(replayGeometry=COORDINATE, recoverGeometry=
    COORDINATE)
#Section
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber1', material='Fiber1',
     thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber2', material='Fiber2',
     thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber3', material='Fiber3',
     thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Matrix', material='Matrix',
     thickness=None)
#Build rhombus and fibers
session.viewports['Viewport: 1'].partDisplay.setValues(sectionAssignments=OFF,
    engineeringFeatures=OFF)
session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
    referenceRepresentation=ON)
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=STANDALONE)
s.rectangle(point1=(-5.0, 5.0), point2=(5.0, -5.0))
p = mdb.models['Model-1'].Part(name='Part-1', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-1']
p.BaseSolidExtrude(sketch=s, depth=5.0)
s.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-1']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
del mdb.models['Model-1'].sketches['__profile__']
session.viewports['Viewport: 1'].view.setValues(nearPlane=22.3108,
    farPlane=40.9522, width=31.4415, height=10.6695, cameraPosition=(
    28.8266, 4.06394, -9.87444), cameraUpVector=(-0.374061, 0.881389,
```

```
        0.288499))
    p = mdb.models['Model-1'].parts['Part-1']
    f, e = p.faces, p.edges
    t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(-1.666667,
        1.666667, 0.0)), sketchUpEdge=e.findAt(coordinates=(-5.0, 2.5, 0.0)),
        sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, origin=(0.0, 0.0, 0.0))
    s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
        sheetSize=28.28, gridSpacing=0.7, transform=t)
    g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
    s1.setPrimaryObject(option=SUPERIMPOSE)
    p = mdb.models['Model-1'].parts['Part-1']
    p.projectReferencesOntoSketch(sketch=s1, filter=COPLANAR_EDGES)
    s1.rectangle(point1=(-5.0, 5.0), point2=(5.0, -5.0))
    p = mdb.models['Model-1'].parts['Part-1']
    f1, e1 = p.faces, p.edges
    p.SolidExtrude(sketchPlane=f1.findAt(coordinates=(-1.666667, 1.666667, 0.0)),
        sketchUpEdge=e1.findAt(coordinates=(-5.0, 2.5, 0.0)),
        sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, sketch=s1, depth=5.0,
        flipExtrudeDirection=OFF)
    s1.unsetPrimaryObject()
    del mdb.models['Model-1'].sketches['__profile__']
    session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])
    p = mdb.models['Model-1'].parts['Part-1']
    f, e, d1 = p.faces, p.edges, p.datums
    t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(-1.666667, 5.0,
        3.333333)), sketchUpEdge=e.findAt(coordinates=(5.0, 5.0, -1.25)),
        sketchPlaneSide=SIDE1, origin=(0.0, 5.0, 0.0))
    s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
        sheetSize=34.64, gridSpacing=0.86, transform=t)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.setPrimaryObject(option=SUPERIMPOSE)
    p = mdb.models['Model-1'].parts['Part-1']
    p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
    s.rectangle(point1=(-2.5, 2.5), point2=(2.5, -2.5))
    p = mdb.models['Model-1'].parts['Part-1']
    f = p.faces
    pickedFaces = f.findAt(((-1.666667, 5.0, 3.333333), ))
    e1, d2 = p.edges, p.datums
    p.PartitionFaceBySketch(sketchUpEdge=e1.findAt(coordinates=(5.0, 5.0, -1.25)),
        faces=pickedFaces, sketch=s)
    s.unsetPrimaryObject()
    del mdb.models['Model-1'].sketches['__profile__']
    p = mdb.models['Model-1'].parts['Part-1']
    c = p.cells
    pickedCells = c.findAt(((-0.833333, 5.0, 0.833333), ))
    e, d1 = p.edges, p.datums
    pickedEdges =(e.findAt(coordinates=(-2.5, 5.0, 1.25)), e.findAt(coordinates=(
        1.25, 5.0, 2.5)), e.findAt(coordinates=(2.5, 5.0, -1.25)), e.findAt(
        coordinates=(-1.25, 5.0, -2.5)))
    p.PartitionCellByExtrudeEdge(line=e.findAt(coordinates=(5.0, 2.5, 5.0)),
        cells=pickedCells, edges=pickedEdges, sense=FORWARD)
    p = mdb.models['Model-1'].parts['Part-1']
    f1, e1, d2 = p.faces, p.edges, p.datums
    t = p.MakeSketchTransform(sketchPlane=f1.findAt(coordinates=(5.0, 1.666667,
        3.333333)), sketchUpEdge=e1.findAt(coordinates=(5.0, -2.5, -5.0)),
        sketchPlaneSide=SIDE1, origin=(5.0, 0.0, 0.0))
    s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
        sheetSize=34.64, gridSpacing=0.86, transform=t)
    g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
    s1.setPrimaryObject(option=SUPERIMPOSE)
    p = mdb.models['Model-1'].parts['Part-1']
    p.projectReferencesOntoSketch(sketch=s1, filter=COPLANAR_EDGES)
    s1.rectangle(point1=(-2.5, 2.5), point2=(2.5, -2.5))
    p = mdb.models['Model-1'].parts['Part-1']
    f = p.faces
    pickedFaces = f.findAt(((5.0, 1.666667, 3.333333), ))
```

```
e, d1 = p.edges, p.datums
p.PartitionFaceBySketch(sketchUpEdge=e.findAt(coordinates=(5.0, -2.5, -5.0)),
    faces=pickedFaces, sketch=s1)
s1.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
pickedCells = c.findAt(((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0,
    0.833333), ))
e1, d2 = p.edges, p.datums
pickedEdges =(e1.findAt(coordinates=(5.0, -1.25, 2.5)), e1.findAt(coordinates=(
    5.0, -2.5, -1.25)), e1.findAt(coordinates=(5.0, 1.25, -2.5)),
    e1.findAt(coordinates=(5.0, 2.5, 1.25)))
p.PartitionCellByExtrudeEdge(line=e1.findAt(coordinates=(2.5, -5.0, 5.0)),
    cells=pickedCells, edges=pickedEdges, sense=FORWARD)
p = mdb.models['Model-1'].parts['Part-1']
f, e, d1 = p.faces, p.edges, p.datums
t = p.MakeSketchTransform(sketchPlane=f.findAt(coordinates=(1.666667, 1.666667,
    5.0)), sketchUpEdge=e.findAt(coordinates=(5.0, 2.5, 5.0)),
    sketchPlaneSide=SIDE1, origin=(0.0, 0.0, 5.0))
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=34.64, gridSpacing=0.86, transform=t)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['Model-1'].parts['Part-1']
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
s.rectangle(point1=(-2.5, 2.5), point2=(2.5, -2.5))
p = mdb.models['Model-1'].parts['Part-1']
f = p.faces
pickedFaces = f.findAt(((1.666667, 1.666667, 5.0), ))
e1, d2 = p.edges, p.datums
p.PartitionFaceBySketch(sketchUpEdge=e1.findAt(coordinates=(5.0, 2.5, 5.0)),
    faces=pickedFaces, sketch=s)
s.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
pickedCells = c.findAt(((-0.833333, -5.0, -0.833333), ), ((-0.833333, 0.833333,
    2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((-0.833333, -0.833333,
    5.0), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0, 0.833333), ))
e, d1 = p.edges, p.datums
pickedEdges =(e.findAt(coordinates=(-2.5, -1.25, 5.0)), e.findAt(coordinates=(
    1.25, -2.5, 5.0)), e.findAt(coordinates=(2.5, 1.25, 5.0)), e.findAt(
    coordinates=(-1.25, 2.5, 5.0)))
p.PartitionCellByExtrudeEdge(line=e.findAt(coordinates=(5.0, -5.0, 1.25)),
    cells=pickedCells, edges=pickedEdges, sense=REVERSE)
#Assign Section
session.viewports['Viewport: 1'].partDisplay.setValues(sectionAssignments=ON,
    engineeringFeatures=ON)
session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
    referenceRepresentation=OFF)
p = mdb.models['Model-1'].parts['Part-1']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
session.viewports['Viewport: 1'].view.setValues(session.views['Front'])
session.viewports['Viewport: 1'].view.setProjection(projection=PARALLEL)
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
cells = c.findAt(((-0.833333, 0.833333, -5.0), ), ((-0.833333, -0.833333, 5.0),
    ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333, 0.833333, 2.4995), ),
    ((-5.0, -0.833333, 0.833333), ), ((5.0, 0.833333, 0.833333), ), ((
    -0.833333, 5.0, 0.833333), ))
region = regionToolset.Region(cells=cells)
p = mdb.models['Model-1'].parts['Part-1']
p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
```

```
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
cells = c.findAt(((4.166667, -5.0, 2.5), ))
region = regionToolset.Region(cells=cells)
p = mdb.models['Model-1'].parts['Part-1']
p.SectionAssignment(region=region, sectionName='Matrix', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
session.viewports['Viewport: 1'].view.setValues(session.views['Iso'])
#Assembly
a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=OFF)
a1 = mdb.models['Model-1'].rootAssembly
a1.DatumCsysByDefault(CARTESIAN)
p = mdb.models['Model-1'].parts['Part-1']
a1.Instance(name='FIBBI-1', part=p, dependent=OFF)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(mesh=ON)
session.viewports['Viewport: 1'].assemblyDisplay.meshOptions.setValues(
    meshTechnique=ON)
#Partition
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((-0.833333, 0.833333, -5.0), ), ((-0.833333,
    -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333,
    0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((4.166667,
    -5.0, 2.5), ), ((-0.833333, 5.0, 0.833333), ))
v21 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v21.findAt(coordinates=(-2.5, -2.5,
    5.0)), point2=v21.findAt(coordinates=(-2.5, 2.5, 5.0)),
    point3=v21.findAt(coordinates=(-2.5, 5.0, 2.5)), cells=pickedCells)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((-0.833333, 0.833333, -5.0), ), ((-0.833333,
    -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333,
    0.833333, 2.4995), ), ((-0.833333, -5.0, 3.333333), ), ((5.0, 0.833333,
    0.833333), ), ((-0.833333, 5.0, 0.833333), ))
v1 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v1.findAt(coordinates=(2.5, -2.5,
    5.0)), point2=v1.findAt(coordinates=(2.5, 2.5, 5.0)), point3=v1.findAt(
    coordinates=(2.5, 5.0, 2.5)), cells=pickedCells)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((0.833333, 4.166667, 5.0), ), ((0.833333, -5.0,
    4.166667), ), ((3.333333, -5.0, -4.166667), ), ((-3.333333, 5.0,
    -4.166667), ), ((-0.833333, -0.833333, 5.0), ), ((-0.833333, -5.0,
    -0.833333), ), ((-0.833333, 0.833333, 2.4995), ), ((-5.0, -0.833333,
    0.833333), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0,
    0.833333), ))
v21 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v21.findAt(coordinates=(5.0, -2.5,
    2.5)), point2=v21.findAt(coordinates=(5.0, 2.5, 2.5)),
    point3=v21.findAt(coordinates=(-2.5, 5.0, 2.5)), cells=pickedCells)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((0.833333, 5.0, -4.166667), ), ((4.166667, -5.0,
    1.666667), ), ((-4.166667, 5.0, 1.666667), ), ((-0.833333, 0.833333,
    -5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333, 0.833333,
    2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((0.833333, -4.166667,
    -5.0), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0, 0.833333),
    ))
v1 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v1.findAt(coordinates=(5.0, -2.5,
    -2.5)), point2=v1.findAt(coordinates=(5.0, 2.5, -2.5)),
```

```
    point3=v1.findAt(coordinates=(-2.5, 5.0, -2.5)), cells=pickedCells)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((-5.0, -4.166667, -1.666667), ), ((-4.166667, 5.0,
    -3.333333), ), ((4.166667, -5.0, -3.333333), ), ((-4.166667, -5.0,
    3.333333), ), ((4.166667, 5.0, 3.333333), ), ((0.833333, -3.333333,
    5.0), ), ((4.166667, -5.0, -1.666667), ), ((-0.833333, 0.833333, -5.0),
    ), ((-0.833333, -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), (
    (-0.833333, 0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((
    0.833333, -4.166667, -5.0), ), ((5.0, 0.833333, 0.833333), ))
v21 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v21.findAt(coordinates=(-2.5, -2.5,
    5.0)), point2=v21.findAt(coordinates=(2.5, -2.5, 5.0)),
    point3=v21.findAt(coordinates=(5.0, -2.5, 2.5)), cells=pickedCells)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
pickedCells = c1.findAt(((-5.0, -0.833333, 3.333333), ), ((3.333333, -0.833333,
    -5.0), ), ((-4.166667, 5.0, -3.333333), ), ((4.166667, 5.0, -1.666667),
    ), ((4.166667, 5.0, 3.333333), ), ((0.833333, 4.166667, 5.0), ), ((
    0.833333, 3.333333, -5.0), ), ((-5.0, 4.166667, -1.666667), ), ((
    -0.833333, 0.833333, -5.0), ), ((-0.833333, -0.833333, 5.0), ), ((
    -0.833333, 0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((
    5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0, 0.833333), ))
v1 = a.instances['FIBBI-1'].vertices
a.PartitionCellByPlaneThreePoints(point1=v1.findAt(coordinates=(-2.5, 2.5,
    5.0)), point2=v1.findAt(coordinates=(2.5, 2.5, 5.0)), point3=v1.findAt(
    coordinates=(5.0, 2.5, -2.5)), cells=pickedCells)
#Mesh
elemType1 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
elemType2 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
elemType3 = mesh.ElemType(elemCode=C3D20R, elemLibrary=STANDARD)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
cells1 = c1.findAt(((5.0, 0.833333, 4.166667), ), ((-5.0, 0.833333, -4.166667),
    ), ((4.166667, 3.333333, -5.0), ), ((-5.0, 3.333333, 4.166667), ), ((
    3.333333, -5.0, 3.333333), ), ((-5.0, 0.833333, 3.333333), ), ((
    3.333333, 0.833333, -5.0), ), ((-5.0, -3.333333, -4.166667), ), ((
    -4.166667, -5.0, -1.666667), ), ((-4.166667, 5.0, -3.333333), ), ((
    4.166667, 5.0, -1.666667), ), ((5.0, -3.333333, -3.333333), ), ((
    -3.333333, -3.333333, 5.0), ), ((4.166667, 5.0, 3.333333), ), ((
    0.833333, 4.166667, 5.0), ), ((0.833333, 3.333333, -5.0), ), ((
    0.833333, -5.0, 4.166667), ), ((5.0, -4.166667, -1.666667), ), ((-5.0,
    4.166667, -1.666667), ), ((-0.833333, 0.833333, -5.0), ), ((-0.833333,
    -0.833333, 5.0), ), ((-0.833333, -5.0, -0.833333), ), ((-0.833333,
    0.833333, 2.4995), ), ((-5.0, -0.833333, 0.833333), ), ((0.833333,
    -5.0, -3.333333), ), ((5.0, 0.833333, 0.833333), ), ((-0.833333, 5.0,
    0.833333), ))
pickedRegions =(cells1, )
a.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2,
    elemType3))
a = mdb.models['Model-1'].rootAssembly
partInstances =(a.instances['FIBBI-1'], )
a.seedPartInstance(regions=partInstances, size=1.25, deviationFactor=0.1)
a = mdb.models['Model-1'].rootAssembly
partInstances =(a.instances['FIBBI-1'], )
a.generateMesh(regions=partInstances)
##############################################################################
# Generate input filde
##############################################################################
#The name of the operating system dependent module imported.
#The following names have currently been registered:
#'posix', 'nt', 'os2', 'ce', 'java', 'riscos'.
if "posix" in os.name:
#    print 'LINUX'
##Cluster
#    mdb.Job(name=inpname, model='Model-1', type=ANALYSIS,
```

```
#           explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
#           description='', parallelizationMethodExplicit=DOMAIN,
#           multiprocessingMode=DEFAULT, numDomains=1, userSubroutine='',
#           numCpus=1, preMemory=16056.0, standardMemory=16056.0,
#           standardMemoryPolicy=MODERATE, scratch='', echoPrint=OFF,
#           modelPrint=OFF, contactPrint=OFF, historyPrint=OFF)
#Lokal
  mdb.Job(name=inpname, model='Model-1', description='', type= ANALYSIS,
    atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
    memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
    modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
    scratch='', parallelizationMethodExplicit=DOMAIN, numDomains=1,
    activateLoadBalancing=False, multiprocessingMode=DEFAULT, numCpus=1)
  mdb.jobs[inpname].writeInput(consistencyChecking=OFF)
elif "nt" in os.name:
#    print 'WINDOWS'
  mdb.Job(name=inpname, model='Model-1', type=ANALYSIS,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
    description='', parallelizationMethodExplicit=DOMAIN,
    multiprocessingMode=DEFAULT, numDomains=1, userSubroutine='',
    numCpus=1, memory=50, memoryUnits=PERCENTAGE, scratch='',
    echoPrint=OFF, modelPrint=OFF, contactPrint=OFF,
    historyPrint=OFF)
  mdb.jobs[inpname].setValues( \
    userSubroutine='C:\\SIMULIA\\Abaqus_Temp\\umat3\\rate_ana.for')
  mdb.jobs[inpname].writeInput(consistencyChecking=OFF)
##############################################################################
# Open input file
##############################################################################
f = open(inpname+".inp",'r+')
read = str(f.readline())
while read[0:5] != '*Node':
  read = str(f.readline())
##############################################################################
# Generate list of nodes
##############################################################################
print 'Read nodes!'
nodes = []
data = []
read = str(f.readline())
while read[:8] != '*Element':
  data = read.split(",")
  data = map(float, data)
  data[0] = int(data[0])
  nodes.append(data)
  read = str(f.readline())
lennodes = len(nodes)
#print("Nodes")
#for line in nodes: print line
##############################################################################
# Generate list of elements
##############################################################################
print 'Read elements!'
elements = []
data = []
if str(eletyp)[0:4]=='C3D8':
    read = str(f.readline())
elif str(eletyp)[0:5]=='C3D10':
    read = str(f.readline())
elif str(eletyp)[0:5]=='C3D20':
    read = str(f.readline()) + str(f.readline())
else:
  print 'No such element type! Check code'
while read[:5] != '*Nset':
  data = read.split(",")
```

```
      elements.append(map(int, data))
      if eval(eletyp)==C3D8:
        read = str(f.readline())
        if eval(eletyp)==C3D10:
        read = str(f.readline())
      if eval(eletyp)==C3D20:
        read = str(f.readline()) + str(f.readline())
      if eval(eletyp)==C3D20R:
        read = str(f.readline()) + str(f.readline())
    print len(elements), 'Elemente'
    print ' Sorting for LDBC and PBC ...'
    ##############################################################################
    # Sorting surface nodes
    ##############################################################################
    tol = 0.1 #*ele
    stellen = 3
    ecken = []

    kantenXpYpZ = []
    kantenXmYpZ = []
    kantenXpYmZ = []
    kantenXmYmZ = []

    kantenYpXpZ = []
    kantenYmXpZ = []
    kantenYpXmZ = []
    kantenYmXmZ = []

    kantenZpXpY = []
    kantenZmXpY = []
    kantenZpXmY = []
    kantenZmXmY = []

    flaechenXm = []
    flaechenXp = []
    flaechenYm = []
    flaechenYp = []
    flaechenZm = []
    flaechenZp = []

    innen = []

    for i in range(len(nodes)):
      #Ecke1
      if round(nodes[i][1],stellen) == round(XNUC*drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*drve/2.,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*drve/2.,stellen):
        ecken.append(nodes[i][0])
      #Ecke2
      elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*drve/2. ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*drve/2. ,stellen):
        ecken.append(nodes[i][0])
      #Ecke3
      elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*drve/2.      ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*-drve/2.       ,stellen):
        ecken.append(nodes[i][0])
      #Ecke4
      elif round(nodes[i][1],stellen) == round(drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(drve/2.       ,stellen) and \
        round(nodes[i][3],stellen) == round(-drve/2.        ,stellen):
        ecken.append(nodes[i][0])
      #Ecke5
      elif round(nodes[i][1],stellen) == round(drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*-drve/2.     ,stellen) and \
```

```
        round(nodes[i][3],stellen) == round(ZNUC*-drve/2.       ,stellen):
        ecken.append(nodes[i][0])
      #Ecke6
      elif round(nodes[i][1],stellen) == round(XNUC*drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*-drve/2.      ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*drve/2.       ,stellen):
        ecken.append(nodes[i][0])
      #Ecke7
      elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*-drve/2.       ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*drve/2.       ,stellen):
        ecken.append(nodes[i][0])
      #Ecke8
      elif round(nodes[i][1],stellen) == round(XNUC*-drve/2.,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*-drve/2.       ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*-drve/2.       ,stellen):
        ecken.append(nodes[i][0])

      #Kanten X+Y-Z
      elif round(nodes[i][1],stellen) != round(+drve/2.,stellen)  and \
        round(nodes[i][1],stellen) !=  round(-drve/2.,stellen)  and \
        round(nodes[i][2],stellen) ==  round(+drve/2.,stellen)  and \
        round(nodes[i][3],stellen) ==  round(-drve/2.,stellen):
        kantenXpYmZ.append(nodes[i][0])
      #Kanten X-Y-Z
      elif round(nodes[i][1],stellen) != round(+drve/2.,stellen)  and \
        round(nodes[i][1],stellen) !=  round(-drve/2.,stellen)  and \
        round(nodes[i][2],stellen) ==  round(-drve/2.,stellen)  and \
        round(nodes[i][3],stellen) ==  round(-drve/2.,stellen):
        kantenXmYmZ.append(nodes[i][0])
      #Kanten X+Y+Z
      elif round(nodes[i][1],stellen) != round(+drve/2.,stellen)  and \
        round(nodes[i][1],stellen) !=  round(-drve/2.,stellen)  and \
        round(nodes[i][2],stellen) ==  round(+drve/2.,stellen)  and \
        round(nodes[i][3],stellen) ==  round(+drve/2.,stellen):
        kantenXpYpZ.append(nodes[i][0])
      #Kanten X-Y+Z
      elif round(nodes[i][1],stellen) != round(+drve/2.,stellen)  and \
        round(nodes[i][1],stellen) !=  round(-drve/2.,stellen)  and \
        round(nodes[i][2],stellen) ==  round(-drve/2.,stellen)  and \
        round(nodes[i][3],stellen) ==  round(+drve/2.,stellen):
        kantenXmYpZ.append(nodes[i][0])

      #Kanten Y+X-Z
        elif round(nodes[i][2],stellen) != round(+drve/2.,stellen)  and \
        round(nodes[i][2],stellen) !=  round(-drve/2.,stellen)  and \
        round(nodes[i][1],stellen) ==  round(+drve/2.,stellen)  and \
        round(nodes[i][3],stellen) ==  round(-drve/2.,stellen):
        kantenYpXmZ.append(nodes[i][0])
      #Kanten Y-X-Z
      elif round(nodes[i][2],stellen) != round(+drve/2.,stellen)  and \
        round(nodes[i][2],stellen) !=  round(-drve/2.,stellen)  and \
        round(nodes[i][1],stellen) ==  round(-drve/2.,stellen)  and \
        round(nodes[i][3],stellen) ==  round(-drve/2.,stellen):
        kantenYmXmZ.append(nodes[i][0])
      #Kanten Y+X+Z
      elif round(nodes[i][2],stellen) != round(+drve/2.,stellen)  and \
        round(nodes[i][2],stellen) !=  round(-drve/2.,stellen)  and \
        round(nodes[i][1],stellen) ==  round(+drve/2.,stellen)  and \
        round(nodes[i][3],stellen) ==  round(+drve/2.,stellen):
        kantenYpXpZ.append(nodes[i][0])
      #Kanten Y-X+Z
      elif round(nodes[i][2],stellen) != round(+drve/2.,stellen)  and \
        round(nodes[i][2],stellen) !=  round(-drve/2.,stellen)  and \
        round(nodes[i][1],stellen) ==  round(-drve/2.,stellen)  and \
        round(nodes[i][3],stellen) ==  round(+drve/2.,stellen):
```

```
        kantenYmXpZ.append(nodes[i][0])

    #Kanten  Z+X-Y
    elif round(nodes[i][3],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][3],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][1],stellen) ==  round(+drve/2.,stellen)  and \
      round(nodes[i][2],stellen) ==  round(-drve/2.,stellen):
      kantenZpXmY.append(nodes[i][0])
    #Kanten  Z-X-Y
    elif round(nodes[i][3],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][3],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][1],stellen) ==  round(-drve/2.,stellen)  and \
      round(nodes[i][2],stellen) ==  round(-drve/2.,stellen):
      kantenZmXmY.append(nodes[i][0])
    #Kanten  Z+X+Y
    elif round(nodes[i][3],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][3],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][1],stellen) ==  round(+drve/2.,stellen)  and \
      round(nodes[i][2],stellen) ==  round(+drve/2.,stellen):
      kantenZpXpY.append(nodes[i][0])
    #Kanten  Z-X+Y
    elif round(nodes[i][3],stellen) != round(+drve/2.,stellen)  and \
      round(nodes[i][3],stellen) !=  round(-drve/2.,stellen)  and \
      round(nodes[i][1],stellen) ==  round(-drve/2.,stellen)  and \
      round(nodes[i][2],stellen) ==  round(+drve/2.,stellen):
      kantenZmXpY.append(nodes[i][0])
    # Minus (m) and Plus (p) side
    #Flaechen Xp
    elif round(nodes[i][1],stellen) == round(+drve/2.,stellen):
      flaechenXp.append(nodes[i][0])
    #Flaechen Xm
    elif round(nodes[i][1],stellen) == round(-drve/2.,stellen):
      flaechenXm.append(nodes[i][0])
    #Flaechen Yp
    elif round((nodes[i][2]),stellen) == round(+drve/2.,stellen):
      flaechenYp.append(nodes[i][0])
    #Flaechen Ym
    elif round((nodes[i][2]),stellen) == round(-drve/2.,stellen):
      flaechenYm.append(nodes[i][0])
    #Flaechen Zp
    elif round(nodes[i][3],stellen) == round(+drve/2.,stellen):
      flaechenZp.append(nodes[i][0])
    #Flaechen Zm
    elif round(nodes[i][3],stellen) == round(-drve/2.,stellen):
      flaechenZm.append(nodes[i][0])
    #Innen
    else:
      innen.append(nodes[i][0])
##############################################################
# Additional sorting for PBC
##############################################################
if rb == 'PBC':
  print 'Zusatzsortierung PBC ...'
  #Sorting opposite nodes within the lists
  #Sorting corners
  eckensort = []

  for i in range(len(ecken)):
    #Ecke1
    if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
      round(nodes[ecken[i]-1][2],stellen) == round(drve/2.,stellen) and \
      round(nodes[ecken[i]-1][3],stellen) == round(drve/2.,stellen):
      eckensort.append(ecken[i])
      break
  for i in range(len(ecken)):
    #Ecke2
```

```
        if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
          round(nodes[ecken[i]-1][2],stellen) == round(drve/2. ,stellen) and \
          round(nodes[ecken[i]-1][3],stellen) == round(drve/2. ,stellen):
          eckensort.append(ecken[i])
          break
      eckensort.append(ecken[i])
      for i in range(len(ecken)):
        #Ecke3
        if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
          round(nodes[ecken[i]-1][2],stellen) == round(drve/2.      ,stellen) and \
          round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.      ,stellen):
          eckensort.append(ecken[i])
          break
      eckensort.append(ecken[i])
      for i in range(len(ecken)):
        #Ecke4
        if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
          round(nodes[ecken[i]-1][2],stellen) == round(drve/2.      ,stellen) and \
          round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.      ,stellen):
          eckensort.append(ecken[i])
          break
      eckensort.append(ecken[i])
      for i in range(len(ecken)):
        #Ecke5
        if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
          round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and
              \
          round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.      ,stellen):
          eckensort.append(ecken[i])
          break
      eckensort.append(ecken[i])
      for i in range(len(ecken)):
        #Ecke6
        if round(nodes[ecken[i]-1][1],stellen) == round(drve/2.,stellen) and \
          round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and
              \
          round(nodes[ecken[i]-1][3],stellen) == round(drve/2.      ,stellen):
          eckensort.append(ecken[i])
          break
      eckensort.append(ecken[i])
      for i in range(len(ecken)):
        #Ecke7
        if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
          round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and
              \
          round(nodes[ecken[i]-1][3],stellen) == round(drve/2.      ,stellen):
          eckensort.append(ecken[i])
          break
      eckensort.append(ecken[i])
      for i in range(len(ecken)):
        #Ecke8
        if round(nodes[ecken[i]-1][1],stellen) == round(-drve/2.,stellen) and \
          round(nodes[ecken[i]-1][2],stellen) == round(-drve/2.      ,stellen) and \
          round(nodes[ecken[i]-1][3],stellen) == round(-drve/2.      ,stellen):
          eckensort.append(ecken[i])
          break
      # Sorting edges

      kantenXsort = []
      kantenYsort = []
      kantenZsort = []

      Xkanten = len(kantenXpYmZ)
      for i in range(Xkanten):
        Xvalue = round(nodes[kantenXpYmZ[i]-1][1],stellen)
        kantenXsort.append(kantenXpYmZ[i])
```

```
    for j in range(Xkanten):
      if Xvalue == round(nodes[kantenXmYmZ[j]-1][1],stellen):
        break
    kantenXsort.append(kantenXmYmZ[j])
    kantenXsort.append(kantenXmYmZ[j])

    for j in range(Xkanten):
      if Xvalue == round(nodes[kantenXmYpZ[j]-1][1],stellen):
        break
    kantenXsort.append(kantenXmYpZ[j])
    kantenXsort.append(kantenXmYpZ[j])

    for j in range(Xkanten):
      if Xvalue == round(nodes[kantenXpYpZ[j]-1][1],stellen):
        break
    kantenXsort.append(kantenXpYpZ[j])
  #print 'Kanten X sortiert'
  #print kantenXsort


  Ykanten = len(kantenYpXmZ)
  for i in range(Ykanten):
    Yvalue = round(nodes[kantenYpXmZ[i]-1][2],stellen)
    kantenYsort.append(kantenYpXmZ[i])

    for j in range(Ykanten):
      if Yvalue == round(nodes[kantenYmXmZ[j]-1][2],stellen):
        break
    kantenYsort.append(kantenYmXmZ[j])
    kantenYsort.append(kantenYmXmZ[j])

    for j in range(Ykanten):
      if Yvalue == round(nodes[kantenYmXpZ[j]-1][2],stellen):
        break
    kantenYsort.append(kantenYmXpZ[j])
    kantenYsort.append(kantenYmXpZ[j])

    for j in range(Ykanten):
      if Yvalue == round(nodes[kantenYpXpZ[j]-1][2],stellen):
        break
    kantenYsort.append(kantenYpXpZ[j])
  #print 'Kanten Y sortiert'
  #print kantenYsort

  Zkanten = len(kantenZpXmY)
  for i in range(Zkanten):
    Yvalue = round(nodes[kantenZpXmY[i]-1][3],stellen)
    kantenZsort.append(kantenZpXmY[i])

    for j in range(Zkanten):
      if Yvalue == round(nodes[kantenZmXmY[j]-1][3],stellen):
        break
    kantenZsort.append(kantenZmXmY[j])
    kantenZsort.append(kantenZmXmY[j])

    for j in range(Zkanten):
      if Yvalue == round(nodes[kantenZmXpY[j]-1][3],stellen):
        break
    kantenZsort.append(kantenZmXpY[j])
    kantenZsort.append(kantenZmXpY[j])

    for j in range(Zkanten):
      if Yvalue == round(nodes[kantenZpXpY[j]-1][3],stellen):
        break
    kantenZsort.append(kantenZpXpY[j])
  #print 'Kanten Z sortiert'
```

```
    #print kantenZsort

    flaechenXsort = []
    flaechenYsort = []
    flaechenZsort = []
    #stellen = 1
    Xflaechen = len(flaechenXp)
    for i in range(Xflaechen):
      Yvalue = nodes[flaechenXp[i]-1][2]
      Zvalue = nodes[flaechenXp[i]-1][3]
      flaechenXsort.append(flaechenXp[i])
      for j in range(Xflaechen):
        #print Yvalue - nodes[flaechenXm[j]-1][2]
        #print Zvalue - nodes[flaechenXm[j]-1][3]
        if abs(Yvalue - nodes[flaechenXm[j]-1][2]) <= tol*ele  and \
          abs(Zvalue - nodes[flaechenXm[j]-1][3]) <= tol*ele:
          break
      flaechenXsort.append(flaechenXm[j])
    #print 'Flaechen X sortiert'
    #print flaechenXsort

    Yflaechen = len(flaechenYp)
    for i in range(Yflaechen):
      Xvalue = nodes[flaechenYp[i]-1][1]
      Zvalue = nodes[flaechenYp[i]-1][3]
      flaechenYsort.append(flaechenYp[i])
      for j in range(Yflaechen):
        if abs(Xvalue - nodes[flaechenYm[j]-1][1]) <= tol*ele and \
          abs(Zvalue - nodes[flaechenYm[j]-1][3]) <= tol*ele:
          break
      flaechenYsort.append(flaechenYm[j])
    #print 'Flaechen Y sortiert'
    #print flaechenYsort

    Zflaechen = len(flaechenZp)
    for i in range(Zflaechen):
      Xvalue = nodes[flaechenZp[i]-1][1]
      Yvalue = nodes[flaechenZp[i]-1][2]
      flaechenZsort.append(flaechenZp[i])
      for j in range(Zflaechen):
        if abs(Xvalue - nodes[flaechenZm[j]-1][1]) <= tol*ele and \
          abs(Yvalue - nodes[flaechenZm[j]-1][2]) <= tol*ele:
          break
      flaechenZsort.append(flaechenZm[j])
    #print 'Flaechen Z sortiert'
    #print flaechenZsort
  if rb=='LDBC':
    print 'LDBC: Sorting nodes list'
    #Merge all lists
    nrf = []
    nrf.extend(ecken)
    nrf.extend(kantenXpYpZ)
    nrf.extend(kantenXmYpZ)
    nrf.extend(kantenXpYmZ)
    nrf.extend(kantenXmYmZ)
    nrf.extend(kantenYpXpZ)
    nrf.extend(kantenYmXpZ)
    nrf.extend(kantenYpXmZ)
    nrf.extend(kantenYmXmZ)
    nrf.extend(kantenZpXpY)
    nrf.extend(kantenZmXpY)
    nrf.extend(kantenZpXmY)
    nrf.extend(kantenZmXmY)
    nrf.extend(flaechenXm)
    nrf.extend(flaechenXp)
    nrf.extend(flaechenYm)
```

```
    nrf.extend(flaechenYp)
    nrf.extend(flaechenZm)
    nrf.extend(flaechenZp)
    #nrf.extend(innen)
    #Sorting nodes list following nrf
    ifortschritt = 0
    i = 0
    for i in range(len(nrf)):
      for k in range(len(nodes)):
        if nodes[k][0]==nrf[i]:
          break
      nodes[k],nodes[i] = nodes[i],nodes[k]
    letzteoberflaeche = len(nrf)-1
  elif rb=='PBC':
    print 'PBC: Nodesliste neu sortieren'
    #Merge all lists
    nrf = []
    nrf.extend(eckensort)
    nrf.extend(kantenXsort)
    nrf.extend(kantenYsort)
    nrf.extend(kantenZsort)
    nrf.extend(flaechenXsort)
    nrf.extend(flaechenYsort)
    nrf.extend(flaechenZsort)
    nrf.extend(innen)
    nodesneu = []
    #print 'len nrf',len(nrf)
    #print 'len nodes',len(nodes)
    i=0
    while i != len(nrf):
    #     if i % 1000 == 0:
    #         print 'i',i
      if i < len(nrf)-1:
        if nrf[i] == nrf[i+1]:
          for k in range(len(nodes)):
            if nodes[k][0] == nrf[i]:
              break
          nodesneu.append(nodes[k])
          nodesneu.append(nodes[k])
          nodes.pop(k)
          i+=2
        else:
          for k in range(len(nodes)):
            if nodes[k][0] == nrf[i]:
              break
          nodesneu.append(nodes[k])
          nodes.pop(k)
          i+=1
      else:
        for k in range(len(nodes)):
          if nodes[k][0] == nrf[i]:
            break
        nodesneu.append(nodes[k])
        nodes.pop(k)
        i+=1
    letzteoberflaeche = len(nrf)-len(innen)-1
    nodes = nodesneu
      #print("Nodesneu")
      #for line in nodes: print line
  ##############################################################################
  #  Write input file
  ##############################################################################
  shutil.copyfile(inpname+".inp", inpname+"_backup.inp")
  f.seek(0,0)
  while read[:5] != '*Node':
    read = str(f.readline())
```

```
f.truncate()
f.write('%d,0.0,0.0,0.0 \n' %(lennodes+1))
f.write('%d,0.0,0.0,0.0 \n' %(lennodes+2))
f.write('%d,0.0,0.0,0.0 \n' %(lennodes+3))

b = open(inpname+"_backup.inp",'r+')
lines1 = b.readlines()
b.close()
os.remove(inpname+"_backup.inp")
for i in range(17,len(lines1)):
  f.write(lines1[i])
f.seek(0,0)
# Add additional nodes
print 'Complete input file ...'
while read[:11] != '** Section:':
  read = str(f.readline())
f.truncate()
if "posix" in os.name:
  f.seek(-17,2)
    #f.seek(-22,2)
if XNUC==1 and YNUC==1 and ZNUC==1:
  f.write('** Section: Fiber1\n')
  f.write('*Solid Section, elset=_PickedSet9, material=Fiber1\n')
  f.write(',\n')
  f.write('** Section: Matrix\n')
  f.write('*Solid Section, elset=_PickedSet10, material=Matrix\n')
  f.write(',\n')
elif XNUC==2 and YNUC==2 and ZNUC==2:
  f.write('** Section: Fiber1\n')
  f.write('*Solid Section, elset=_PickedSet4_#1, material=Fiber1\n')
  f.write(',\n')
  f.write('** Section: Fiber2\n')
  f.write('*Solid Section, elset=_PickedSet7_#2, material=Fiber2\n')
  f.write(',\n')
  f.write('** Section: Matrix\n')
  f.write('*Solid Section, elset=_PickedSet8_#3, material=Matrix\n')
  f.write(',\n')
# Define equation
f.write('**\n')
f.write('** Constraint: Constraint-1 \n')
f.write('**\n')
f.write('*Equation \n')
if rb == 'LDBC':
##############################################################################
# Linear displacement boundary conditions (LDBC)
##############################################################################
  print 'Linear displacement boundary conditions (LDBC)!'
  #DOF1
  for i in range(0,letzteoberflaeche+1):
    f.write("4\n")
    f.write("%d,1,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
      %(nodes[i][0],len(nodes)+1,-nodes[i][1],len(nodes)+1, \
      -nodes[i][2],len(nodes)+1,-nodes[i][3]))
  #DOF2
  for i in range(0,letzteoberflaeche+1):
    f.write("4\n")
    f.write("%d,2,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
      %(nodes[i][0],len(nodes)+2,-nodes[i][1],len(nodes)+2, \
      -nodes[i][2],len(nodes)+2,-nodes[i][3]))
  #DOF3
  for i in range(0,letzteoberflaeche+1):
    f.write("4\n")
    f.write("%d,3,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
      %(nodes[i][0],len(nodes)+3,-nodes[i][1],len(nodes)+3, \
      -nodes[i][2],len(nodes)+3,-nodes[i][3]))
```

```
elif rb == 'PBC':
##############################################################################
# Periodic boundary conditions (PBC)
##############################################################################
  print 'Periodische Verschiebungs-RB (PVRB) gewaehlt!'
  #DOF1
  for i in range(0,letzteoberflaeche+1,2):
    f.write("5\n")
    f.write("%d,1,1.0\n%d,1,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
      %(nodes[i][0], \
      nodes[i+1][0], \
      lennodes+1,(nodes[i+1][1]-nodes[i][1]), \
      lennodes+1,(nodes[i+1][2]-nodes[i][2]), \
      lennodes+1,(nodes[i+1][3]-nodes[i][3])))
  #DOF2
  for i in range(0,letzteoberflaeche+1,2):
    f.write("5\n")
    f.write("%d,2,1.0\n%d,2,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
      %(nodes[i][0], \
      nodes[i+1][0], \
      lennodes+2,(nodes[i+1][1]-nodes[i][1]), \
      lennodes+2,(nodes[i+1][2]-nodes[i][2]), \
      lennodes+2,(nodes[i+1][3]-nodes[i][3])))
  #DOF3
  for i in range(0,letzteoberflaeche+1,2):
    f.write("5\n")
    f.write("%d,3,1.0\n%d,3,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
      %(nodes[i][0], \
      nodes[i+1][0], \
      lennodes+3,(nodes[i+1][1]-nodes[i][1]), \
      lennodes+3,(nodes[i+1][2]-nodes[i][2]), \
      lennodes+3,(nodes[i+1][3]-nodes[i][3])))

##prescribe H symmetric
#f.write("2\n")
#f.write("%d,2,1.0\n%d,1,-1.0\n" %(lennodes+1, lennodes+2))
#f.write("2\n")
#f.write("%d,3,1.0\n%d,1,-1.0\n" %(lennodes+1, lennodes+3))
#f.write("2\n")
#f.write("%d,3,1.0\n%d,2,-1.0\n" %(lennodes+2, lennodes+3))


f.write('*End Instance \n')
# Determine origin
ursprung = 'kein'
for i in range(0,len(nodes)):
  if abs(nodes[i][1]) == 0. and \
    abs(nodes[i][2]) == 0. and \
    abs(nodes[i][3]) == 0. <= tol*ele:
    ursprung=nodes[i][0]
if ursprung == 'kein':
  print 'No origin, check mesh'

#zusatz1 = 'kein'
#for i in range(0,len(nodes)):
  #if abs(nodes[i][1] - (YNUC*fa1/tan(anglerad)/2.+XNUC*fa2)) <= tol*ele and \
    #abs(nodes[i][2] - (YNUC*fa1/2.)) <= tol*ele and \
    #abs(nodes[i][3] - (ZNUC*drve/2.)) <= tol*ele:
    #zusatz1=nodes[i][0]
#if zusatz1 == 'kein':
  #print 'Kein Zusatz1, Vernetzung pruefen'

#zusatz2 = 'kein'
#for i in range(0,len(nodes)):
  #if abs(nodes[i][1] - (YNUC*fa1/tan(anglerad)/2.+XNUC*fa2/2.)) <= tol*ele and
      \
```

```
        #abs(nodes[i][2] - (YNUC*fa1/2.)) <= tol*ele and \
        #abs(nodes[i][3] - (ZNUC*drve)) <= tol*ele:
        #zusatz2=nodes[i][0]
    #if zusatz2 == 'kein':
      #print 'Kein Zusatz2, Vernetzung pruefen'

    # Node sets for boundary of the artificial nodes
    f.write('*Nset, nset=K1, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+1))
    f.write('*Nset, nset=K2, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+2))
    f.write('*Nset, nset=K3, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+3))

    if rb != 'LDBC':
        f.write('*Nset, nset=Ursprung, internal, instance=FIBBI-1 \n')
        f.write('%d, \n' %ursprung)

    #f.write('*Nset, nset=zusatz1, internal, instance=FIBBI-1 \n')
    #f.write('%d, \n' %zusatz1)
    #f.write('*Nset, nset=zusatz2, internal, instance=FIBBI-1 \n')
    #f.write('%d, \n' %zusatz2)

    f.write('*End Assembly \n')
    f.write('** \n')
    f.write('** MATERIALS\n')
    f.write('** \n')
# ==============================================================
# Element type B: 2 round fibers in x and y direction
#                 different fiber angle, size, distance
# ==============================================================
if typ == 'B':
    ##############################################################################
    print '============================================================'
    print 'Element type B: 2 round fibers in x and y direction'
    print '============================================================'
    ##############################################################################
    # Fiber radius
    rf1=0.2
    rf2=0.2
    # Fiber distance
    fa1=1.
    fa2=1.
    # Layer Distance
    la = 0.1
    # Calculate RVE thickness d
    drve = 2.*la + 2.*rf1 + 2.*rf2
    # Fiber angle <= 90 degree
    angle = 90.
    anglerad=angle/180.0*pi
    # Number of unit cells
    XNUC=1
    YNUC=1
    ZNUC=1
    # Calculate COOS
    movex = -0.5*(XNUC-1.)
    movey = -sqrt(3.)/2.*(YNUC-1.)
    # Area A0
    #A0 = fa1*fa2*sin(anglerad)
    A0 = fa1*YNUC*fa2*XNUC
    # Volume V0
    V0 = A0*drve*ZNUC
    # Obsolet
    seed = []
    seed.append(XNUC)
    ##############################################################################
```

```
# Generate faserinput.txt
##############################################################################
inpname = 'RVE_%s_%s_%s_ele%s_%s' %(typ, 'X'+str(XNUC)+'Y'+str(YNUC)+'Z'+str(
    ZNUC),rb,str(ele).replace('.',''),extname)
print inpname
f = open(inpname+'_fiberinput.txt','w' )
f.write('CPU number:\n')
f.write('%s\n' %(cpunumber))
f.write('RVE type:\n')
f.write('%s\n' %(typ))
f.write('Type of boundary condition:\n')
f.write('%s\n' %(rb))
f.write('Element size:\n')
f.write('%.3f\n' %(ele))
f.write('Element shape:\n')
f.write('%s\n' %(eleshape))
f.write('Element type:\n')
f.write('%s\n' %(eletyp))

f.write('Radius fiber 1:\n')
f.write('%.3f\n' %(rf1))
f.write('Distance fiber 1:\n')
f.write('%.3f\n' %(fa1))
f.write('Radius fiber 2:\n')
f.write('%.3f\n' %(rf2))
f.write('Distance fiber 2:\n')
f.write('%.3f\n' %(fa2))
f.write('Layer Distance:\n')
f.write('%.3f\n' %(la))
f.write('RVE thickness d:\n')
f.write('%.3f\n' %(drve))
f.write('Angle between fibers:\n')
f.write('%.3f\n' %(angle))

f.write('Displacement Gradient\n')
f.write('H0:\n')
f.write('%s,%s,%s\n' %(str(H0[0][0]),str(H0[0][1]),str(H0[0][2])))
f.write('%s,%s,%s\n' %(str(H0[1][0]),str(H0[1][1]),str(H0[1][2])))
f.write('%s,%s,%s\n' %(str(H0[2][0]),str(H0[2][1]),str(H0[2][2])))

f.write('Name of input file\n')
f.write(inpname)
f.write('\n ')
f.close()
##############################################################################
# Generate model
##############################################################################
# Close old model
Mdb()
session.viewports['Viewport: 1'].setValues(displayedObject=None)
session.journalOptions.setValues(replayGeometry=COORDINATE, recoverGeometry=
    COORDINATE)
# Section
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber1', material='Fiber1',
      thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber2', material='Fiber2',
      thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Matrix', material='Matrix',
      thickness=None)
# Build Rhombus
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
  sheetSize=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.Line(point1=(0.0, 0.0), point2=(fa2, 0.0))
s1.Line(point1=(fa2, 0.0), point2=(fa1/tan(anglerad)+fa2, fa1))
```

```
s1.Line(point1=(fa1/tan(anglerad)+fa2, fa1), point2=(fa1/tan(anglerad), fa1))
s1.Line(point1=(fa1/tan(anglerad), fa1), point2=(0.0, 0.0))
p = mdb.models['Model-1'].Part(name='Part-1', dimensionality=THREE_D,
  type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-1']
p.BaseSolidExtrude(sketch=s1, depth=drve)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-1']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
session.viewports['Viewport: 1'].view.setValues(session.views['Front'])
#Fiber1
p = mdb.models['Model-1'].parts['Part-1']
f1, e1, d2 = p.faces, p.edges, p.datums
t = p.MakeSketchTransform(
  sketchPlane=f1.findAt(coordinates=(fa1/2./tan(anglerad), fa1/2., drve/2.)),
  sketchUpEdge=e1.findAt(coordinates=(fa1/2./tan(anglerad), fa1/2., drve)),
  sketchPlaneSide=SIDE1, origin=(0.0, 0.0, 0.0))
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
  sheetSize=8.32, gridSpacing=0.2, transform=t)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=SUPERIMPOSE)
p = mdb.models['Model-1'].parts['Part-1']
p.projectReferencesOntoSketch(sketch=s1, filter=COPLANAR_EDGES)
s1.CircleByCenterPerimeter(center=(la/2.+rf1, fa1/2./sin(anglerad)), point1=(la
    /2., fa1/2./sin(anglerad)))
p = mdb.models['Model-1'].parts['Part-1']
f = p.faces
pickedFaces = f.findAt(((fa1/2./tan(anglerad), fa1/2., drve/2.), ))
e, d1 = p.edges, p.datums
p.PartitionFaceBySketch(sketchUpEdge=e.findAt(coordinates=(fa1/2./tan(anglerad)
    , fa1/2., drve)), faces=pickedFaces, sketch=s1)
s1.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
pickedCells = c.findAt(((0., 0., 0.), ))
e1, d2 = p.edges, p.datums
pickedEdges =(e1.findAt(coordinates=(fa1/2./tan(anglerad), fa1/2., la/2.)), )
p.PartitionCellByExtrudeEdge(line=e1.findAt(coordinates=(fa2/4., 0.0, 0.0)),
  cells=pickedCells, edges=pickedEdges, sense=FORWARD)
#Assign Section
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
cells = c.findAt(((fa1/2./tan(anglerad), fa1/2., la/2.+rf1), ))
region = regionToolset.Region(cells=cells)
p = mdb.models['Model-1'].parts['Part-1']
p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
  offsetType=MIDDLE_SURFACE, offsetField='',
  thicknessAssignment=FROM_SECTION)
#Fiber2
p = mdb.models['Model-1'].parts['Part-1']
f1, e, d1 = p.faces, p.edges, p.datums
t = p.MakeSketchTransform(
  sketchPlane=f1.findAt(coordinates=(fa2/2., 0.0, drve/2.)),
  sketchUpEdge=e.findAt(coordinates=(  0.0, 0.0, drve/2.)),
  sketchPlaneSide=SIDE1, sketchOrientation=LEFT, origin=(0.0, 0.0, 0.0))
s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=6.56,
  gridSpacing=0.16, transform=t)
g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
s.setPrimaryObject(option=SUPERIMPOSE)
p.projectReferencesOntoSketch(sketch=s, filter=COPLANAR_EDGES)
s.CircleByCenterPerimeter(center=(fa2/2., la/2.+2.*rf1+la+rf2), point1=(fa2/2.+
    rf2, la/2.+2.*rf1+la+rf2))
p = mdb.models['Model-1'].parts['Part-1']
f = p.faces
pickedFaces = f.findAt(((fa2/2., 0.0, drve/2.), ))
```

```
e1, d2 = p.edges, p.datums
p.PartitionFaceBySketch(sketchUpEdge=e1.findAt(coordinates=(0.0, 0.0, drve/2.))
    ,
  faces=pickedFaces, sketchOrientation=LEFT, sketch=s)
s.unsetPrimaryObject()
del mdb.models['Model-1'].sketches['__profile__']
c = p.cells
pickedCells = c.findAt(((fa2/2.,0.0, la/2.+2.*rf1+la+rf2), ))
e, d = p.edges, p.datums
pickedEdges =(e.findAt(coordinates=(fa2/2.+rf2, 0.0, la/2.+2.*rf1+la+rf2)), )
p.PartitionCellByExtrudeEdge(line=e.findAt(coordinates=(fa1/2./tan(anglerad)
    /2., fa1/2./2., 0.0)),
  cells=pickedCells, edges=pickedEdges, sense=FORWARD)
#Assign Section
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
cells = c.findAt(((fa2/2., 0.0, la/2.+2.*rf1+la+rf2), ))
region = regionToolset.Region(cells=cells)
p = mdb.models['Model-1'].parts['Part-1']
p.SectionAssignment(region=region, sectionName='Fiber2', offset=0.0,
  offsetType=MIDDLE_SURFACE, offsetField='',
  thicknessAssignment=FROM_SECTION)

#Assign Section
p = mdb.models['Model-1'].parts['Part-1']
c = p.cells
cells = c.findAt(((0., 0., 0.), ))
region = regionToolset.Region(cells=cells)
p = mdb.models['Model-1'].parts['Part-1']
p.SectionAssignment(region=region, sectionName='Matrix', offset=0.0,
  offsetType=MIDDLE_SURFACE, offsetField='',
  thicknessAssignment=FROM_SECTION)

#Generate PartitionCells

p = mdb.models['Model-1'].parts['Part-1']
f2, e2 = p.faces, p.edges
p.DatumPlaneByOffset(plane=f1.findAt(
  coordinates=(fa1/2./tan(anglerad), fa1/2., drve/2.)),
  flip=SIDE2, offset=fa2/2.*sin(anglerad))
p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=fa1/2.)
p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=la/2.+rf1)
p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=drve/2.)
p.DatumPlaneByPrincipalPlane(principalPlane=XYPLANE, offset=drve-la/2.-rf2)

c1 = p.cells
cel = c1.pointsOn
pickedCells = c1.findAt(cel[0],)
for i in range(1,len(cel)):
  pickedCells += c1.findAt(cel[i],)
d1 = p.datums
p.PartitionCellByDatumPlane(datumPlane=d1[9], cells=pickedCells)

c1 = p.cells
cel = c1.pointsOn
pickedCells = c1.findAt(cel[0],)
for i in range(1,len(cel)):
  pickedCells += c1.findAt(cel[i],)
d1 = p.datums
p.PartitionCellByDatumPlane(datumPlane=d1[10], cells=pickedCells)

c1 = p.cells
cel = c1.pointsOn
pickedCells = c1.findAt(cel[0],)
for i in range(1,len(cel)):
  pickedCells += c1.findAt(cel[i],)
```

```
d1 = p.datums
p.PartitionCellByDatumPlane(datumPlane=d1[11], cells=pickedCells)

c1 = p.cells
cel = c1.pointsOn
pickedCells = c1.findAt(cel[0],)
for i in range(1,len(cel)):
  pickedCells += c1.findAt(cel[i],)
d1 = p.datums
p.PartitionCellByDatumPlane(datumPlane=d1[12], cells=pickedCells)

c1 = p.cells
cel = c1.pointsOn
pickedCells = c1.findAt(cel[0],)
for i in range(1,len(cel)):
  pickedCells += c1.findAt(cel[i],)
d1 = p.datums
p.PartitionCellByDatumPlane(datumPlane=d1[13], cells=pickedCells)

session.viewports['Viewport: 1'].view.setValues(nearPlane=8.79367,
  farPlane=17.6814, cameraPosition=(-6.43266, 4.4182, 11.8694),
  cameraUpVector=(0.292388, 0.82094, -0.490476))
session.viewports['Viewport: 1'].view.fitView()

#InstanceFromBooleanMerge
if XNUC==1 and YNUC==1 and ZNUC==1:
  p1 = mdb.models['Model-1'].parts['Part-1']
  mdb.models['Model-1'].parts.changeKey(fromName='Part-1', toName='FIBBI')
  a = mdb.models['Model-1'].rootAssembly
  a.DatumCsysByDefault(CARTESIAN)
  p = mdb.models['Model-1'].parts['FIBBI']
  a.Instance(name='FIBBI-1', part=p, dependent=OFF)
else:
#Assembly
  a = mdb.models['Model-1'].rootAssembly
  a.DatumCsysByDefault(CARTESIAN)
  p = mdb.models['Model-1'].parts['Part-1']
  a.Instance(name='Part-1-1', part=p, dependent=OFF)
#Pattern for more than one unit cell
  a.LinearInstancePattern(instanceList=('Part-1-1', ), direction1=(1.0, 0.0,
    0.0), direction2=(fa1/2./tan(anglerad), fa1/2., 0.0), number1=XNUC, number2
        =YNUC,
    spacing1=fa2, spacing2=fa1/sin(anglerad))
  a.LinearInstancePattern(
    instanceList=(a.instances.keys()),
    direction1=(0.0, 0.0, 1.0),
    direction2=(0.0, 1.0, 0.0),
    number1=ZNUC, number2=1, spacing1=drve, spacing2=0.0)
  a = mdb.models['Model-1'].rootAssembly
  a.InstanceFromBooleanMerge(name='FIBBI', instances=(a.instances.values()),
    keepIntersections=ON, originalInstances=DELETE, domain=GEOMETRY)
session.viewports['Viewport: 1'].setValues(displayedObject=a)
session.viewports['Viewport: 1'].view.fitView()
#Mesh
a.makeIndependent(instances=(a.instances['FIBBI-1'], ))
partInstances =(a.instances['FIBBI-1'], )
a.seedPartInstance(regions=partInstances, size=ele, deviationFactor=0.1)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
cel = c1.pointsOn
pickedRegions = c1.findAt(cel[0],)
for i in range(1,len(cel)):
  pickedRegions += c1.findAt(cel[i],)
a.setMeshControls(regions=pickedRegions, elemShape=HEX_DOMINATED,
  technique=SWEEP, algorithm=ADVANCING_FRONT)
```

```
if eleshape == 'TET':
  a.setMeshControls(regions=pickedRegions, elemShape=eval(eleshape), technique=
      FREE)
else:
  a.setMeshControls(regions=pickedRegions, elemShape=eval(eleshape), technique=
      SYSTEM_ASSIGN)
elemType1 = mesh.ElemType(eval(eletyp))
pickedRegions = (pickedRegions,)
a.setElementType(regions=(pickedRegions), elemTypes=(elemType1, ))
partInstances =(a.instances['FIBBI-1'], )
a.generateMesh(regions=partInstances)
##############################################################################
## Write input file
##############################################################################
#The name of the operating system dependent module imported.
#The following names have currently been registered:
#'posix', 'nt', 'os2', 'ce', 'java', 'riscos'.
if "posix" in os.name:
#    print 'LINUX'
##Cluster
#    mdb.Job(name=inpname, model='Model-1', type=ANALYSIS,
#         explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
#         description='', parallelizationMethodExplicit=DOMAIN,
#         multiprocessingMode=DEFAULT, numDomains=1, userSubroutine='',
#         numCpus=1, preMemory=16056.0, standardMemory=16056.0,
#         standardMemoryPolicy=MODERATE, scratch='', echoPrint=OFF,
#         modelPrint=OFF, contactPrint=OFF, historyPrint=OFF)
# Local
  mdb.Job(name=inpname, model='Model-1', description='', type= ANALYSIS,
    atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
    memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
    modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
    scratch='', parallelizationMethodExplicit=DOMAIN, numDomains=1,
    activateLoadBalancing=False, multiprocessingMode=DEFAULT, numCpus=1)
  mdb.jobs[inpname].writeInput(consistencyChecking=OFF)
elif "nt" in os.name:
# WINDOWS
  mdb.Job(name=inpname, model='Model-1', type=ANALYSIS,
    explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
    description='', parallelizationMethodExplicit=DOMAIN,
    multiprocessingMode=DEFAULT, numDomains=1, userSubroutine='',
    numCpus=1, memory=50, memoryUnits=PERCENTAGE, scratch='',
    echoPrint=OFF, modelPrint=OFF, contactPrint=OFF,
    historyPrint=OFF)
  mdb.jobs[inpname].setValues( \
    userSubroutine='C:\\SIMULIA\\Abaqus_Temp\\umat3\\rate_ana.for')
  mdb.jobs[inpname].writeInput(consistencyChecking=OFF)
##############################################################################
# Open input file
##############################################################################
f = open(inpname+".inp",'r+')
read = str(f.readline())
while read[0:5] != '*Node':
  read = str(f.readline())
##############################################################################
# Generate list of nodes
##############################################################################
print 'Read nodes'
nodes = []
data = []
read = str(f.readline())
while read[:8] != '*Element':
  data = read.split(",")
  data = map(float, data)
  data[0] = int(data[0])
```

```
    nodes.append(data)
    read = str(f.readline())
lennodes = len(nodes)
#print("Nodes")
#for line in nodes: print line
##############################################################################
# Generate list of elements
##############################################################################
print 'Read elements'
elements = []
data = []
if str(eletyp)[0:4]=='C3D8':
  read = str(f.readline())
elif str(eletyp)[0:5]=='C3D20':
  read = str(f.readline()) + str(f.readline())
else:
  print 'No such element type! Check input'
while read[:5] != '*Nset':
  data = read.split(",")
  elements.append(map(int, data))
  if eval(eletyp)==C3D8:
    read = str(f.readline())
  if eval(eletyp)==C3D20:
    read = str(f.readline()) + str(f.readline())
print len(elements), 'Elements'
print 'Sorting for LDBC and PBC ...'
##############################################################################
# Sorting of surface node couples
##############################################################################
tol = 0.1 #*ele
stellen = 3
ecken = []

kantenXpYpZ = []
kantenXmYpZ = []
kantenXpYmZ = []
kantenXmYmZ = []

kantenYpXpZ = []
kantenYmXpZ = []
kantenYpXmZ = []
kantenYmXmZ = []

kantenZpXpY = []
kantenZmXpY = []
kantenZpXmY = []
kantenZmXmY = []

flaechenXm = []
flaechenXp = []
flaechenYm = []
flaechenYp = []
flaechenZm = []
flaechenZp = []

innen = []

fta = YNUC*fa1/tan(anglerad)
for i in range(len(nodes)):
  #Ecke1
  if round(nodes[i][1],stellen) == round(XNUC*0.,stellen) and \
    round(nodes[i][2],stellen) == round(YNUC*0.,stellen) and \
    round(nodes[i][3],stellen) == round(ZNUC*0.,stellen):
    ecken.append(nodes[i][0])
  #Ecke2
  elif round(nodes[i][1],stellen) == round(XNUC*fa2,stellen) and \
```

```
        round(nodes[i][2],stellen) == round(YNUC*0.  ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*0.  ,stellen):
        ecken.append(nodes[i][0])
      #Ecke3
      elif round(nodes[i][1],stellen) == round(XNUC*fa2+fta,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*fa1       ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*0.        ,stellen):
        ecken.append(nodes[i][0])
      #Ecke4
      elif round(nodes[i][1],stellen) == round(fta,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*fa1       ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*0.        ,stellen):
        ecken.append(nodes[i][0])
      #Ecke5
      elif round(nodes[i][1],stellen) == round(fta,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*fa1       ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*drve      ,stellen):
        ecken.append(nodes[i][0])
      #Ecke6
      elif round(nodes[i][1],stellen) == round(XNUC*fa2+fta,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*fa1       ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*drve      ,stellen):
        ecken.append(nodes[i][0])
      #Ecke7
      elif round(nodes[i][1],stellen) == round(XNUC*fa2,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*0.      ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*drve      ,stellen):
        ecken.append(nodes[i][0])
      #Ecke8
      elif round(nodes[i][1],stellen) == round(XNUC*0.,stellen) and \
        round(nodes[i][2],stellen) == round(YNUC*0.      ,stellen) and \
        round(nodes[i][3],stellen) == round(ZNUC*drve      ,stellen):
        ecken.append(nodes[i][0])

      #Kanten X+Y-Z
      elif round(nodes[i][1],stellen) != round(fta,stellen)  and \
        round(nodes[i][1],stellen) != round(XNUC*fa2+fta,stellen)  and \
        round(nodes[i][2],stellen) == round(YNUC*fa1,stellen)  and \
        round(nodes[i][3],stellen) == round(0.,stellen):
        kantenXpYmZ.append(nodes[i][0])
      #Kanten X-Y-Z
      elif round(nodes[i][1],stellen) != round(fta,stellen)  and \
        round(nodes[i][1],stellen) != round(XNUC*fa2+fta,stellen)  and \
        round(nodes[i][2],stellen) == round(0.,stellen)  and \
        round(nodes[i][3],stellen) == round(0.,stellen):
        kantenXmYmZ.append(nodes[i][0])
      #Kanten X+Y+Z
      elif round(nodes[i][1],stellen) != round(fta,stellen)  and \
        round(nodes[i][1],stellen) != round(XNUC*fa2+fta,stellen)  and \
        round(nodes[i][2],stellen) == round(YNUC*fa1,stellen)  and \
        round(nodes[i][3],stellen) == round(ZNUC*drve,stellen):
        kantenXpYpZ.append(nodes[i][0])
      #Kanten X-Y+Z
      elif round(nodes[i][1],stellen) != round(fta,stellen)  and \
        round(nodes[i][1],stellen) != round(XNUC*fa2+fta,stellen)  and \
        round(nodes[i][2],stellen) == round(0.,stellen)  and \
        round(nodes[i][3],stellen) == round(ZNUC*drve,stellen):
        kantenXmYpZ.append(nodes[i][0])

      #Kanten Y+X-Z
      elif round(nodes[i][1],stellen) == round(XNUC*fa2+nodes[i][2]/tan(anglerad),
          stellen)  and \
        round(nodes[i][2],stellen) != round(0.,stellen)  and \
        round(nodes[i][2],stellen) != round(YNUC*fa1,stellen)  and \
        round(nodes[i][3],stellen) == round(0.,stellen):
        kantenYpXmZ.append(nodes[i][0])
```

```
        #Kanten Y-X-Z
        elif round(nodes[i][1],stellen) == round(nodes[i][2]/tan(anglerad),stellen)
            and \
          round(nodes[i][2],stellen) != round(0.,stellen)  and \
          round(nodes[i][2],stellen) != round(YNUC*fa1,stellen)  and \
          round(nodes[i][3],stellen) == round(0.,stellen):
          kantenYmXmZ.append(nodes[i][0])
        #Kanten Y+X+Z
        elif round(nodes[i][1],stellen) == round(XNUC*fa2+nodes[i][2]/tan(anglerad),
            stellen)  and \
          round(nodes[i][2],stellen) != round(0.,stellen)  and \
          round(nodes[i][2],stellen) != round(YNUC*fa1,stellen)  and \
          round(nodes[i][3],stellen) == round(ZNUC*drve,stellen):
          kantenYpXpZ.append(nodes[i][0])
        #Kanten Y-X+Z
        elif round(nodes[i][1],stellen) == round(nodes[i][2]/tan(anglerad),stellen)
            and \
          round(nodes[i][2],stellen) != round(0.,stellen)  and \
          round(nodes[i][2],stellen) != round(YNUC*fa1,stellen)  and \
          round(nodes[i][3],stellen) == round(ZNUC*drve,stellen):
          kantenYmXpZ.append(nodes[i][0])

        #Kanten  Z+X-Y
        elif round(nodes[i][1],stellen) == round(XNUC*fa2,stellen)  and \
          round(nodes[i][2],stellen) == round(0.,stellen)  and \
          round(nodes[i][3],stellen) != round(0.,stellen)        and \
          round(nodes[i][3],stellen) != round(ZNUC*drve,stellen):
          kantenZpXmY.append(nodes[i][0])
        #Kanten  Z-X-Y
        elif round(nodes[i][1],stellen) == round(0.,stellen)  and \
          round(nodes[i][2],stellen) == round(0.,stellen)  and \
          round(nodes[i][3],stellen) != round(0.,stellen)        and \
          round(nodes[i][3],stellen) != round(ZNUC*drve,stellen):
          kantenZmXmY.append(nodes[i][0])
        #Kanten  Z+X+Y
        elif round(nodes[i][1],stellen) == round(XNUC*fa2+fta,stellen)  and \
          round(nodes[i][2],stellen) == round(YNUC*fa1,stellen)  and \
          round(nodes[i][3],stellen) != round(0.,stellen)        and \
          round(nodes[i][3],stellen) != round(ZNUC*drve,stellen):
          kantenZpXpY.append(nodes[i][0])
        #Kanten  Z-X+Y
        elif round(nodes[i][1],stellen) == round(fta,stellen)  and \
          round(nodes[i][2],stellen) == round(YNUC*fa1,stellen)  and \
          round(nodes[i][3],stellen) != round(0.,stellen)        and \
          round(nodes[i][3],stellen) != round(ZNUC*drve,stellen):
          kantenZmXpY.append(nodes[i][0])
        #Minus (m) and plus (p) side
        #Flaechen Xp
        elif round(nodes[i][1],stellen) == round(nodes[i][2]/tan(anglerad),stellen):
          flaechenXp.append(nodes[i][0])
        #Flaechen Xm
        elif round(nodes[i][1],stellen) == round(XNUC*fa2+nodes[i][2]/tan(anglerad),
            stellen):
          flaechenXm.append(nodes[i][0])
        #Flaechen Yp
        elif round((nodes[i][2]),stellen) == round(YNUC*fa1,stellen):
          flaechenYp.append(nodes[i][0])
        #Flaechen Ym
        elif round((nodes[i][2]),stellen) == round(0.,stellen):
          flaechenYm.append(nodes[i][0])
        #Flaechen Zp
        elif round(nodes[i][3],stellen) == round(ZNUC*drve,stellen):
          flaechenZp.append(nodes[i][0])
        #Flaechen Zm
        elif round(nodes[i][3],stellen) == round(0.,stellen):
          flaechenZm.append(nodes[i][0])
```

```
    #Innen
    else:
      innen.append(nodes[i][0])
  ##################################################################
  # Additional sorting for PBC
  ##################################################################
  if rb == 'PBC':
    print 'Additional sorting PBC ...'
    #Sorting of opposite nodes within the list
    #Sorting corners
    eckensort = []

    for i in range(len(ecken)):
      #Ecke1
      if round(nodes[ecken[i]-1][1],stellen) == round(XNUC*0.,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(YNUC*0.,stellen) and \
        round(nodes[ecken[i]-1][3],stellen) == round(ZNUC*0.,stellen):
        eckensort.append(ecken[i])
        break
    for i in range(len(ecken)):
      #Ecke2
      if round(nodes[ecken[i]-1][1],stellen) == round(XNUC*fa2,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(YNUC*0. ,stellen) and \
        round(nodes[ecken[i]-1][3],stellen) == round(ZNUC*0. ,stellen):
        eckensort.append(ecken[i])
        break
    eckensort.append(ecken[i])
    for i in range(len(ecken)):
      #Ecke3
      if round(nodes[ecken[i]-1][1],stellen) == round(XNUC*fa2+fta,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(YNUC*fa1      ,stellen) and
            \
        round(nodes[ecken[i]-1][3],stellen) == round(ZNUC*0.       ,stellen):
        eckensort.append(ecken[i])
        break
    eckensort.append(ecken[i])
    for i in range(len(ecken)):
      #Ecke4
      if round(nodes[ecken[i]-1][1],stellen) == round(fta,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(YNUC*fa1      ,stellen) and
            \
        round(nodes[ecken[i]-1][3],stellen) == round(ZNUC*0.       ,stellen):
        eckensort.append(ecken[i])
        break
    eckensort.append(ecken[i])
    for i in range(len(ecken)):
      #Ecke5
      if round(nodes[ecken[i]-1][1],stellen) == round(fta,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(YNUC*fa1      ,stellen) and
            \
        round(nodes[ecken[i]-1][3],stellen) == round(ZNUC*drve     ,stellen):
        eckensort.append(ecken[i])
        break
    eckensort.append(ecken[i])
    for i in range(len(ecken)):
      #Ecke6
      if round(nodes[ecken[i]-1][1],stellen) == round(XNUC*fa2+fta,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(YNUC*fa1      ,stellen) and
            \
        round(nodes[ecken[i]-1][3],stellen) == round(ZNUC*drve     ,stellen):
        eckensort.append(ecken[i])
        break
    eckensort.append(ecken[i])
    for i in range(len(ecken)):
      #Ecke7
      if round(nodes[ecken[i]-1][1],stellen) == round(XNUC*fa2,stellen) and \
```

```
        round(nodes[ecken[i]-1][2],stellen) == round(YNUC*0.        ,stellen) and \
        round(nodes[ecken[i]-1][3],stellen) == round(ZNUC*drve      ,stellen):
        eckensort.append(ecken[i])
        break
    eckensort.append(ecken[i])
    for i in range(len(ecken)):
      #Ecke8
      if round(nodes[ecken[i]-1][1],stellen) == round(XNUC*0.,stellen) and \
        round(nodes[ecken[i]-1][2],stellen) == round(YNUC*0.        ,stellen) and \
        round(nodes[ecken[i]-1][3],stellen) == round(ZNUC*drve      ,stellen):
        eckensort.append(ecken[i])
        break
    # Sorting edges
    kantenXsort = []
    kantenYsort = []
    kantenZsort = []

    Xkanten = len(kantenXpYmZ)
    for i in range(Xkanten):
      Xvalue = round(nodes[kantenXpYmZ[i]-1][1]-fta,stellen)
      kantenXsort.append(kantenXpYmZ[i])

      for j in range(Xkanten):
        if Xvalue == round(nodes[kantenXmYmZ[j]-1][1],stellen):
          break
      kantenXsort.append(kantenXmYmZ[j])
      kantenXsort.append(kantenXmYmZ[j])

      for j in range(Xkanten):
        if Xvalue == round(nodes[kantenXmYpZ[j]-1][1],stellen):
          break
      kantenXsort.append(kantenXmYpZ[j])
      kantenXsort.append(kantenXmYpZ[j])

      for j in range(Xkanten):
        if Xvalue == round(nodes[kantenXpYpZ[j]-1][1]-fta,stellen):
          break
      kantenXsort.append(kantenXpYpZ[j])
    #print 'Kanten X sortiert'
    #print kantenXsort
    Ykanten = len(kantenYpXmZ)
    for i in range(Ykanten):
      Yvalue = round(nodes[kantenYpXmZ[i]-1][2],stellen)
      kantenYsort.append(kantenYpXmZ[i])

      for j in range(Ykanten):
        if Yvalue == round(nodes[kantenYmXmZ[j]-1][2],stellen):
          break
      kantenYsort.append(kantenYmXmZ[j])
      kantenYsort.append(kantenYmXmZ[j])

      for j in range(Ykanten):
        if Yvalue == round(nodes[kantenYmXpZ[j]-1][2],stellen):
          break
      kantenYsort.append(kantenYmXpZ[j])
      kantenYsort.append(kantenYmXpZ[j])

      for j in range(Ykanten):
        if Yvalue == round(nodes[kantenYpXpZ[j]-1][2],stellen):
          break
      kantenYsort.append(kantenYpXpZ[j])
    #print 'Kanten Y sortiert'
    #print kantenYsort
    Zkanten = len(kantenZpXmY)
    for i in range(Zkanten):
      Yvalue = round(nodes[kantenZpXmY[i]-1][3],stellen)
```

```
        kantenZsort.append(kantenZpXmY[i])

        for j in range(Zkanten):
          if Yvalue == round(nodes[kantenZmXmY[j]-1][3],stellen):
            break
        kantenZsort.append(kantenZmXmY[j])
        kantenZsort.append(kantenZmXmY[j])

        for j in range(Zkanten):
          if Yvalue == round(nodes[kantenZmXpY[j]-1][3],stellen):
            break
        kantenZsort.append(kantenZmXpY[j])
        kantenZsort.append(kantenZmXpY[j])

        for j in range(Zkanten):
          if Yvalue == round(nodes[kantenZpXpY[j]-1][3],stellen):
            break
        kantenZsort.append(kantenZpXpY[j])
      #print 'Kanten Z sortiert'
      #print kantenZsort
      flaechenXsort = []
      flaechenYsort = []
      flaechenZsort = []
      #stellen = 1
      Xflaechen = len(flaechenXp)
      for i in range(Xflaechen):
        Yvalue = nodes[flaechenXp[i]-1][2]
        Zvalue = nodes[flaechenXp[i]-1][3]
        flaechenXsort.append(flaechenXp[i])
        for j in range(Xflaechen):
          if abs(Yvalue - nodes[flaechenXm[j]-1][2]) <= tol*ele  and \
            abs(Zvalue - nodes[flaechenXm[j]-1][3]) <= tol*ele:
            break
        flaechenXsort.append(flaechenXm[j])
      #print 'Flaechen X sortiert'
      #print flaechenXsort
      Yflaechen = len(flaechenYp)
      for i in range(Yflaechen):
        Xvalue = nodes[flaechenYp[i]-1][1]-fta
        Zvalue = nodes[flaechenYp[i]-1][3]
        flaechenYsort.append(flaechenYp[i])
        for j in range(Yflaechen):
          if abs(Xvalue - nodes[flaechenYm[j]-1][1]) <= tol*ele and \
            abs(Zvalue - nodes[flaechenYm[j]-1][3]) <= tol*ele:
            break
        flaechenYsort.append(flaechenYm[j])
      #print 'Flaechen Y sortiert'
      #print flaechenYsort
      Zflaechen = len(flaechenZp)
      for i in range(Zflaechen):
        Xvalue = nodes[flaechenZp[i]-1][1]
        Yvalue = nodes[flaechenZp[i]-1][2]
        flaechenZsort.append(flaechenZp[i])
        for j in range(Zflaechen):
          if abs(Xvalue - nodes[flaechenZm[j]-1][1]) <= tol*ele and \
            abs(Yvalue - nodes[flaechenZm[j]-1][2]) <= tol*ele:
            break
        flaechenZsort.append(flaechenZm[j])
      #print 'Flaechen Z sortiert'
      #print flaechenZsort
    if rb=='LDBC':
      print 'LDBC: Sorting nodes list again'
      # Merge all lists
      nrf = []
      nrf.extend(ecken)
      nrf.extend(kantenXpYpZ)
```

```
    nrf.extend(kantenXmYpZ)
    nrf.extend(kantenXpYmZ)
    nrf.extend(kantenXmYmZ)
    nrf.extend(kantenYpXpZ)
    nrf.extend(kantenYmXpZ)
    nrf.extend(kantenYpXmZ)
    nrf.extend(kantenYmXmZ)
    nrf.extend(kantenZpXpY)
    nrf.extend(kantenZmXpY)
    nrf.extend(kantenZpXmY)
    nrf.extend(kantenZmXmY)
    nrf.extend(flaechenXm)
    nrf.extend(flaechenXp)
    nrf.extend(flaechenYm)
    nrf.extend(flaechenYp)
    nrf.extend(flaechenZm)
    nrf.extend(flaechenZp)
    #nrf.extend(innen)
    # Sorting the list of nodes corresponding to nrf
    ifortschritt = 0
    i = 0
    for i in range(len(nrf)):
      for k in range(len(nodes)):
        if nodes[k][0]==nrf[i]:
          break
      nodes[k],nodes[i] = nodes[i],nodes[k]
    letzteoberflaeche = len(nrf)-1
  elif rb=='PBC':
    print 'PBC: Sorting nodes list again'
    # Merge all lists
    nrf = []
    nrf.extend(eckensort)
    nrf.extend(kantenXsort)
    nrf.extend(kantenYsort)
    nrf.extend(kantenZsort)
    nrf.extend(flaechenXsort)
    nrf.extend(flaechenYsort)
    nrf.extend(flaechenZsort)
    nrf.extend(innen)
    nodesneu = []
    #print 'len nrf',len(nrf)
    #print 'len nodes',len(nodes)
    i=0
    while i != len(nrf):
    #     if i % 1000 == 0:
    #         print 'i',i
      if i < len(nrf)-1:
        if nrf[i] == nrf[i+1]:
          for k in range(len(nodes)):
            if nodes[k][0] == nrf[i]:
              break
          nodesneu.append(nodes[k])
          nodesneu.append(nodes[k])
          nodes.pop(k)
          i+=2
        else:
          for k in range(len(nodes)):
            if nodes[k][0] == nrf[i]:
              break
          nodesneu.append(nodes[k])
          nodes.pop(k)
          i+=1
      else:
        for k in range(len(nodes)):
          if nodes[k][0] == nrf[i]:
            break
```

```
            nodesneu.append(nodes[k])
            nodes.pop(k)
            i+=1
        letzteoberflaeche = len(nrf)-len(innen)-1
        nodes = nodesneu
          #print("Nodesneu")
          #for line in nodes: print line
    ##############################################################################
    # Write input file
    ##############################################################################
    shutil.copyfile(inpname+".inp", inpname+"_backup.inp")
    f.seek(0,0)
    while read[:5] != '*Node':
      read = str(f.readline())
    f.truncate()
    f.write('%d,0.0,0.0,0.0 \n' %(lennodes+1))
    f.write('%d,0.0,0.0,0.0 \n' %(lennodes+2))
    f.write('%d,0.0,0.0,0.0 \n' %(lennodes+3))

    b = open(inpname+"_backup.inp",'r+')
    lines1 = b.readlines()
    b.close()
    os.remove(inpname+"_backup.inp")
    for i in range(17,len(lines1)):
      f.write(lines1[i])

    f.seek(0,0)

    # Add additional nodes
    print 'Complete input file ...'
    while read[:11] != '** Section:':
      read = str(f.readline())
    f.truncate()
    if "posix" in os.name:
      f.seek(-17,2)
        #f.seek(-22,2)

    if XNUC==1 and YNUC==1 and ZNUC==1:
      f.write('** Section: Fiber1\n')
      f.write('*Solid Section, elset=_PickedSet4, material=Fiber1\n')
      f.write(',\n')
      f.write('** Section: Fiber2\n')
      f.write('*Solid Section, elset=_PickedSet7, material=Fiber2\n')
      f.write(',\n')
      f.write('** Section: Matrix\n')
      f.write('*Solid Section, elset=_PickedSet8, material=Matrix\n')
      f.write(',\n')
    elif XNUC==2 and YNUC==2 and ZNUC==2:
      f.write('** Section: Fiber1\n')
      f.write('*Solid Section, elset=_PickedSet4_#1, material=Fiber1\n')
      f.write(',\n')
      f.write('** Section: Fiber2\n')
      f.write('*Solid Section, elset=_PickedSet7_#2, material=Fiber2\n')
      f.write(',\n')
      f.write('** Section: Matrix\n')
      f.write('*Solid Section, elset=_PickedSet8_#3, material=Matrix\n')
      f.write(',\n')

    # Define equations
    f.write('**\n')
    f.write('** Constraint: Constraint-1 \n')
    f.write('**\n')
    f.write('*Equation \n')
    if rb == 'LDBC':
    ##############################################################################
    # Linear displacement boundary conditions (LDBC)
```

```
    ##############################################################################
      print 'Linear displacement boundary conditions'
      #DOF1
      for i in range(0,letzteoberflaeche+1):
        f.write("4\n")
        f.write("%d,1,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
          %(nodes[i][0],len(nodes)+1,-nodes[i][1],len(nodes)+1, \
          -nodes[i][2],len(nodes)+1,-nodes[i][3]))
      #DOF2
      for i in range(0,letzteoberflaeche+1):
        f.write("4\n")
        f.write("%d,2,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
          %(nodes[i][0],len(nodes)+2,-nodes[i][1],len(nodes)+2, \
          -nodes[i][2],len(nodes)+2,-nodes[i][3]))
      #DOF3
      for i in range(0,letzteoberflaeche+1):
        f.write("4\n")
        f.write("%d,3,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
          %(nodes[i][0],len(nodes)+3,-nodes[i][1],len(nodes)+3, \
          -nodes[i][2],len(nodes)+3,-nodes[i][3]))

    elif rb == 'PBC':
    ##############################################################################
    # Periodic boundary conditions (PBC)
    ##############################################################################
      print 'Periodic boundary conditions'
      #DOF1
      for i in range(0,letzteoberflaeche+1,2):
        f.write("5\n")
        f.write("%d,1,1.0\n%d,1,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
          %(nodes[i][0], \
          nodes[i+1][0], \
          lennodes+1,(nodes[i+1][1]-nodes[i][1]), \
          lennodes+1,(nodes[i+1][2]-nodes[i][2]), \
          lennodes+1,(nodes[i+1][3]-nodes[i][3])))
      #DOF2
      for i in range(0,letzteoberflaeche+1,2):
        f.write("5\n")
        f.write("%d,2,1.0\n%d,2,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
          %(nodes[i][0], \
          nodes[i+1][0], \
          lennodes+2,(nodes[i+1][1]-nodes[i][1]), \
          lennodes+2,(nodes[i+1][2]-nodes[i][2]), \
          lennodes+2,(nodes[i+1][3]-nodes[i][3])))
      #DOF3
      for i in range(0,letzteoberflaeche+1,2):
        f.write("5\n")
        f.write("%d,3,1.0\n%d,3,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
          %(nodes[i][0], \
          nodes[i+1][0], \
          lennodes+3,(nodes[i+1][1]-nodes[i][1]), \
          lennodes+3,(nodes[i+1][2]-nodes[i][2]), \
          lennodes+3,(nodes[i+1][3]-nodes[i][3])))
    ##prescribe H symmetric
    #f.write("2\n")
    #f.write("%d,2,1.0\n%d,1,-1.0\n" %(lennodes+1, lennodes+2))
    #f.write("2\n")
    #f.write("%d,3,1.0\n%d,1,-1.0\n" %(lennodes+1, lennodes+3))
    #f.write("2\n")
    #f.write("%d,3,1.0\n%d,2,-1.0\n" %(lennodes+2, lennodes+3))
    f.write('*End Instance \n')
    # Determine origin
    ursprung = 'kein'
    for i in range(0,len(nodes)):
      if abs(nodes[i][1] - (YNUC*fa1/tan(anglerad)/2.+XNUC*fa2/2.)) <= tol*ele and \
```

```
        abs(nodes[i][2] - (YNUC*fa1/2.)) <= tol*ele and \
        abs(nodes[i][3] - (ZNUC*drve/2.)) <= tol*ele:
        ursprung=nodes[i][0]
    if ursprung == 'kein':
      print 'No origin, check mesh'
    zusatz1 = 'kein'
    for i in range(0,len(nodes)):
      if abs(nodes[i][1] - (YNUC*fa1/tan(anglerad)/2.+XNUC*fa2)) <= tol*ele and \
        abs(nodes[i][2] - (YNUC*fa1/2.)) <= tol*ele and \
        abs(nodes[i][3] - (ZNUC*drve/2.)) <= tol*ele:
        zusatz1=nodes[i][0]
    if zusatz1 == 'kein':
      print 'Kein Zusatz1, Vernetzung pruefen'
    zusatz2 = 'kein'
    for i in range(0,len(nodes)):
      if abs(nodes[i][1] - (YNUC*fa1/tan(anglerad)/2.+XNUC*fa2/2.)) <= tol*ele and
          \
        abs(nodes[i][2] - (YNUC*fa1/2.)) <= tol*ele and \
        abs(nodes[i][3] - (ZNUC*drve)) <= tol*ele:
        zusatz2=nodes[i][0]
    if zusatz2 == 'kein':
      print 'Kein Zusatz2, Vernetzung pruefen'
    # Generate node sets for boundary of the artificial nodes
    f.write('*Nset, nset=K1, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+1))
    f.write('*Nset, nset=K2, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+2))
    f.write('*Nset, nset=K3, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(lennodes+3))
    if rb != 'LDBC':
        f.write('*Nset, nset=Ursprung, internal, instance=FIBBI-1 \n')
        f.write('%d, \n' %ursprung)

    f.write('*Nset, nset=zusatz1, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %zusatz1)
    f.write('*Nset, nset=zusatz2, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %zusatz2)

    f.write('*End Assembly \n')
    f.write('** \n')
    f.write('** MATERIALS\n')
    f.write('** \n')
# ==============================================================
# Element type U: 1 round fiber in z direction
#                 transveral-isotropic symmetry
# ==============================================================
if typ == 'U':
    ##############################################################################
    print '============================================================'
    print 'Element type U: 1 round fiber in z direction'
    print '============================================================'
    ##############################################################################
    # Volume fraction of fibers in %
    vi    = 60
    # Calculate fiber radius
    rf = sqrt(vi*sqrt(3.)/200./pi)
    print 'Radius fiber',rf
    # X-Number of unit cells
    XNUC=1
    # Y-Number of unit cells
    YNUC=1
    # Unit cell size
    xunitcell = 1.0
    yunitcell = sqrt(3.)
    zunitcell = 2.*ele
    # Calculate COOS
```

```
movex = -0.5*(XNUC-1.)
movey = -sqrt(3.)/2.*(YNUC-1.)
# Calculate RVE Size
xsize = xunitcell*XNUC
ysize = yunitcell*YNUC
zsize = zunitcell
# Area A0
#A0 = fa1*fa2*sin(anglerad)
A0 = xsize*ysize
# Volume V0
V0 = A0*zsize
# Obsolet, only for name
seed = []
seed.append(XNUC)
##############################################################################
# Generate faserinput.txt
##############################################################################
inpname = 'RVE_%s_%s_%s_ele%s_%s' %(typ,'X'+str(XNUC)+'Y'+str(YNUC),rb,str(ele)
    .replace('.',''),extname)
print inpname
f = open(inpname+'_fiberinput.txt','w' )
f.write('CPU number:\n')
f.write('%s\n' %(cpunumber))
f.write('RVE type:\n')
f.write('%s\n' %(typ))
f.write('Boundary condition:\n')
f.write('%s\n' %(rb))
f.write('Element size [ele]:\n')
f.write('%.3f\n' %(ele))
f.write('Element shape:\n')
f.write('%s\n' %(eleshape))
f.write('Element type:\n')
f.write('%s\n' %(eletyp))
f.write('Volumefraction of fibers (%):\n')
f.write('%.3f\n' %(vi))
f.write('Resulting fiber radius:\n')
f.write('%.3f\n' %(rf))
f.write('Displacement Gradient\n')
f.write('H0:\n')
f.write('%s,%s,%s\n' %(str(H0[0][0]),str(H0[0][1]),str(H0[0][2])))
f.write('%s,%s,%s\n' %(str(H0[1][0]),str(H0[1][1]),str(H0[1][2])))
f.write('%s,%s,%s\n' %(str(H0[2][0]),str(H0[2][1]),str(H0[2][2])))
f.write('Materials\n')
mat = open('0_material.py','r')
for line in mat:
    f.write('%s' %line)
mat.close()
f.write('Name of input file\n')
f.write(inpname)
f.write('\n ')
f.close()
##############################################################################
# Build RVE model
##############################################################################
# Close open models
Mdb()
session.viewports['Viewport: 1'].setValues(displayedObject=None)
session.journalOptions.setValues(replayGeometry=COORDINATE,recoverGeometry=
    COORDINATE)
# Section
mdb.models['Model-1'].HomogeneousSolidSection(name='Fiber1', material='Fiber1',
     thickness=None)
mdb.models['Model-1'].HomogeneousSolidSection(name='Matrix', material='Matrix',
     thickness=None)
# Build Fiber-11
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
```

```
    sheetSize=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.ArcByCenterEnds(center=(0.0+movex, 0.0+movey),
                   point1=(-rf+movex, 0.0+movey),
                   point2=(0.0+movex, rf+movey),
                direction=CLOCKWISE)
session.viewports['Viewport: 1'].view.fitView()
s1.Line(point1=(0.0+movex, rf+movey), point2=(0.0+movex, 0.0+movey))
s1.Line(point1=(0.0+movex, 0.0+movey), point2=(-rf+movex, 0.0+movey))
p = mdb.models['Model-1'].Part(name='Part-11', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-11']
p.BaseSolidExtrude(sketch=s1, depth=zunitcell)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-11']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-11']
c = p.cells
cells = c.findAt(((0.0+movex, 0.0+movey, zunitcell/2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Build Fiber-12
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.ArcByCenterEnds(center=(0.0+movex, 0.0+movey),
                   point1=(0.0+movex, rf+movey),
                   point2=(rf+movex, 0.0+movey),
                direction=CLOCKWISE)
session.viewports['Viewport: 1'].view.fitView()
s1.Line(point1=(0.0+movex, rf+movey), point2=(0.0+movex, 0.0+movey))
s1.Line(point1=(0.0+movex, 0.0+movey), point2=(rf+movex, 0.0+movey))
p = mdb.models['Model-1'].Part(name='Part-12', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-12']
p.BaseSolidExtrude(sketch=s1, depth=zunitcell)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-12']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-12']
c = p.cells
cells = c.findAt(((0.0+movex, 0.0+movey, zunitcell/2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Build Fiber-13
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.ArcByCenterEnds(center=(0.0+movex, 0.0+movey),
                   point1=(rf+movex, 0.0+movey),
                   point2=(0.0+movex, -rf+movey),
                direction=CLOCKWISE)
session.viewports['Viewport: 1'].view.fitView()
s1.Line(point1=(0.0+movex, -rf+movey), point2=(0.0+movex, 0.0+movey))
s1.Line(point1=(0.0+movex, 0.0+movey), point2=(rf+movex, 0.0+movey))
```

```
p = mdb.models['Model-1'].Part(name='Part-13', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-13']
p.BaseSolidExtrude(sketch=s1, depth=zunitcell)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-13']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-13']
c = p.cells
cells = c.findAt(((0.0+movex, 0.0+movey, zunitcell/2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Build Fiber-14
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.ArcByCenterEnds(center=(0.0+movex, 0.0+movey),
                   point1=(0.0+movex, -rf+movey),
                   point2=(-rf+movex, 0.0+movey),
                direction=CLOCKWISE)
session.viewports['Viewport: 1'].view.fitView()
s1.Line(point1=(0.0+movex, -rf+movey), point2=(0.0+movex, 0.0+movey))
s1.Line(point1=(0.0+movex, 0.0+movey), point2=(-rf+movex, 0.0+movey))
p = mdb.models['Model-1'].Part(name='Part-14', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-14']
p.BaseSolidExtrude(sketch=s1, depth=zunitcell)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-14']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-14']
c = p.cells
cells = c.findAt(((0.0+movex, 0.0+movey, zunitcell/2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Build Matrix-21
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.ArcByCenterEnds(center=(0.0+movex, 0.0+movey),
                   point1=(-rf+movex, 0.0+movey),
                   point2=(0.0+movex, rf+movey),
                direction=CLOCKWISE)
s1.ArcByCenterEnds(center=(-0.5+movex, sqrt(3.)/2.+movey),
                   point1=(-0.5+movex+rf, sqrt(3.)/2.+movey),
                   point2=(-0.5+movex,sqrt(3.)/2.-rf+movey),
                direction=CLOCKWISE)
session.viewports['Viewport: 1'].view.fitView()
s1.Line(point1=(-0.5+rf+movex, sqrt(3.)/2.+movey), point2=(0.0+movex, sqrt(3.)
    /2.+movey))
s1.Line(point1=(0.0+movex, sqrt(3.)/2.+movey), point2=(0.0+movex, rf+movey))
s1.Line(point1=(-rf+movex, 0.0+movey), point2=(-0.5+movex, 0.0+movey))
s1.Line(point1=(-0.5+movex, 0.0+movey), point2=(-0.5+movex, sqrt(3.)/2.-rf+
    movey))
p = mdb.models['Model-1'].Part(name='Part-21', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
```

```
p = mdb.models['Model-1'].parts['Part-21']
p.BaseSolidExtrude(sketch=s1, depth=zunitcell)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-21']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-21']
c = p.cells
cells = c.findAt(((-(0.5-rf)/2.-rf+movex, +(sqrt(3.)/2.-rf)/2.+movey, zunitcell
    /2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Matrix', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Build Matrix-22
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.ArcByCenterEnds(center=(0.0+movex, 0.0+movey),
                   point1=(0.0+movex, rf+movey),
                   point2=(rf+movex, 0.0+movey),
                direction=CLOCKWISE)
s1.ArcByCenterEnds(center=(0.5+movex, sqrt(3.)/2.+movey),
                   point1=(0.5+movex, sqrt(3.)/2.-rf+movey),
                   point2=(0.5-rf+movex, sqrt(3.)/2.+movey),
                direction=CLOCKWISE)
session.viewports['Viewport: 1'].view.fitView()
s1.Line(point1=(0.5-rf+movex, sqrt(3.)/2.+movey), point2=(0.0+movex, sqrt(3.)
    /2.+movey))
s1.Line(point1=(0.0+movex, sqrt(3.)/2.+movey), point2=(0.0+movex, rf+movey))
s1.Line(point1=(rf+movex, 0.0+movey), point2=(0.5+movex, 0.0+movey))
s1.Line(point1=(0.5+movex, 0.0+movey), point2=(0.5+movex, sqrt(3.)/2.-rf+movey)
    )
p = mdb.models['Model-1'].Part(name='Part-22', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-22']
p.BaseSolidExtrude(sketch=s1, depth=zunitcell)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-22']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-22']
c = p.cells
cells = c.findAt(((+(0.5-rf)/2.+rf+movex, +(sqrt(3.)/2.-rf)/2.+movey, zunitcell
    /2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Matrix', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Build Matrix-23
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.ArcByCenterEnds(center=(0.0+movex, 0.0+movey),
                   point1=(rf+movex, 0.0+movey),
                   point2=(0.0+movex,-rf+movey),
                direction=CLOCKWISE)
s1.ArcByCenterEnds(center=(0.5+movex, -sqrt(3.)/2.+movey),
                   point1=(0.5-rf+movex, -sqrt(3.)/2.+movey),
                   point2=(0.5+movex, -sqrt(3.)/2.+rf+movey),
                direction=CLOCKWISE)
session.viewports['Viewport: 1'].view.fitView()
```

```
s1.Line(point1=(0.5-rf+movex, -sqrt(3.)/2.+movey), point2=(0.0+movex, -sqrt(3.)
    /2.+movey))
s1.Line(point1=(0.0+movex, -sqrt(3.)/2.+movey), point2=(0.0+movex, -rf+movey))
s1.Line(point1=(rf+movex, 0.0+movey), point2=(0.5+movex, 0.0+movey))
s1.Line(point1=(0.5+movex, 0.0+movey), point2=(0.5+movex, -sqrt(3.)/2.+rf+movey
    ))
p = mdb.models['Model-1'].Part(name='Part-23', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-23']
p.BaseSolidExtrude(sketch=s1, depth=zunitcell)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-23']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-23']
c = p.cells
cells = c.findAt(((+(0.5-rf)/2.+rf+movex, -(sqrt(3.)/2.-rf)/2.+movey, zunitcell
    /2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Matrix', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Build Matrix-24
s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=STANDALONE)
s1.ArcByCenterEnds(center=(0.0+movex, 0.0+movey),
                   point1=(0.0+movex,-rf+movey),
                   point2=(-rf+movex, 0.0+movey),
                   direction=CLOCKWISE)
s1.ArcByCenterEnds(center=(-0.5+movex, -sqrt(3.)/2.+movey),
                   point1=(-0.5+movex, -sqrt(3.)/2.+rf+movey),
                   point2=(-0.5+rf+movex, -sqrt(3.)/2.+movey),
                   direction=CLOCKWISE)
session.viewports['Viewport: 1'].view.fitView()
s1.Line(point1=(-0.5+rf+movex, -sqrt(3.)/2.+movey), point2=(0.0+movex, -sqrt
    (3.)/2.+movey))
s1.Line(point1=(0.0+movex, -sqrt(3.)/2.+movey), point2=(0.0+movex, -rf+movey))
s1.Line(point1=(-rf+movex, 0.0+movey), point2=(-0.5+movex, 0.0+movey))
s1.Line(point1=(-0.5+movex, 0.0+movey), point2=(-0.5+movex, -sqrt(3.)/2.+rf+
    movey))
p = mdb.models['Model-1'].Part(name='Part-24', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-24']
p.BaseSolidExtrude(sketch=s1, depth=zunitcell)
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-24']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-24']
c = p.cells
cells = c.findAt(((-(0.5-rf)/2.-rf+movex, -(sqrt(3.)/2.-rf)/2.+movey, zunitcell
    /2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Matrix', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Build Fiber-31
s0 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s0.geometry, s0.vertices, s0.dimensions, s0.constraints
s0.setPrimaryObject(option=STANDALONE)
s0.ArcByCenterEnds(center=(-0.5+movex, sqrt(3.)/2.+movey),
```

```
                    point1=(-0.5+rf+movex, sqrt(3.)/2.+movey),
                    point2=(-0.5+movex, sqrt(3.)/2.-rf+movey),
    direction=CLOCKWISE)
s0.Line(point1=(-0.5+movex, sqrt(3.)/2.+movey), point2=(-0.5+rf+movex, sqrt(3.)
    /2.+movey))
s0.Line(point1=(-0.5+movex, sqrt(3.)/2.+movey), point2=(-0.5+movex, sqrt(3.)
    /2.-rf+movey))
p = mdb.models['Model-1'].Part(name='Part-31', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-31']
p.BaseSolidExtrude(sketch=s0, depth=zunitcell)
s0.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-31']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-31']
c = p.cells
cells = c.findAt(((-0.5+1./sqrt(2.)*rf+movex, sqrt(3.)/2.-1./sqrt(2.)*rf+movey,
     zunitcell/2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Build Fiber-32
s0 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s0.geometry, s0.vertices, s0.dimensions, s0.constraints
s0.setPrimaryObject(option=STANDALONE)
s0.ArcByCenterEnds(center=(0.5+movex, sqrt(3.)/2.+movey),
                    point1=(0.5+movex, sqrt(3.)/2.-rf+movey),
                    point2=(0.5-rf+movex, sqrt(3.)/2.+movey),
    direction=CLOCKWISE)
s0.Line(point1=(0.5+movex, sqrt(3.)/2.+movey), point2=(0.5+movex, sqrt(3.)/2.-
    rf+movey))
s0.Line(point1=(0.5+movex, sqrt(3.)/2.+movey), point2=(0.5-rf+movex, sqrt(3.)
    /2.+movey))
p = mdb.models['Model-1'].Part(name='Part-32', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-32']
p.BaseSolidExtrude(sketch=s0, depth=zunitcell)
s0.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-32']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-32']
c = p.cells
cells = c.findAt(((0.5-1./sqrt(2.)*rf+movex, sqrt(3.)/2.-1./sqrt(2.)*rf+movey,
    zunitcell/2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Build Fiber-33
s0 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s0.geometry, s0.vertices, s0.dimensions, s0.constraints
s0.setPrimaryObject(option=STANDALONE)
s0.ArcByCenterEnds(center=(0.5+movex, -sqrt(3.)/2.+movey),
                    point1=(0.5+movex, -sqrt(3.)/2.+rf+movey),
                    point2=(0.5-rf+movex, -sqrt(3.)/2.+movey),
    direction=COUNTERCLOCKWISE)
s0.Line(point1=(0.5+movex, -sqrt(3.)/2.+movey), point2=(0.5+movex, -sqrt(3.)
    /2.+rf+movey))
```

```
s0.Line(point1=(0.5+movex, -sqrt(3.)/2.+movey), point2=(0.5-rf+movex, -sqrt(3.)
    /2.+movey))
p = mdb.models['Model-1'].Part(name='Part-33', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-33']
p.BaseSolidExtrude(sketch=s0, depth=zunitcell)
s0.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-33']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-33']
c = p.cells
cells = c.findAt(((0.5-1./sqrt(2.)*rf+movex, -sqrt(3.)/2.+1./sqrt(2.)*rf+movey,
     zunitcell/2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Build Fiber-34
s0 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=200.0)
g, v, d, c = s0.geometry, s0.vertices, s0.dimensions, s0.constraints
s0.setPrimaryObject(option=STANDALONE)
s0.ArcByCenterEnds(center=(-0.5+movex, -sqrt(3.)/2.+movey),
                   point1=(-0.5+movex, -sqrt(3.)/2.+rf+movey),
                   point2=(-0.5+rf+movex, -sqrt(3.)/2.+movey),
    direction=CLOCKWISE)
s0.Line(point1=(-0.5+movex, -sqrt(3.)/2.+movey), point2=(-0.5+movex, -sqrt(3.)
    /2.+rf+movey))
s0.Line(point1=(-0.5+movex, -sqrt(3.)/2.+movey), point2=(-0.5+rf+movex, -sqrt
    (3.)/2.+movey))
p = mdb.models['Model-1'].Part(name='Part-34', dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
p = mdb.models['Model-1'].parts['Part-34']
p.BaseSolidExtrude(sketch=s0, depth=zunitcell)
s0.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-34']
session.viewports['Viewport: 1'].setValues(displayedObject=p)
# Assign Section
p = mdb.models['Model-1'].parts['Part-34']
c = p.cells
cells = c.findAt(((-0.5+1./sqrt(2.)*rf+movex, -sqrt(3.)/2.+1./sqrt(2.)*rf+movey
    , zunitcell/2.), ))
region = regionToolset.Region(cells=cells)
p.SectionAssignment(region=region, sectionName='Fiber1', offset=0.0,
    offsetType=MIDDLE_SURFACE, offsetField='',
    thicknessAssignment=FROM_SECTION)
del mdb.models['Model-1'].sketches['__profile__']
# Assembly
a = mdb.models['Model-1'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
a1 = mdb.models['Model-1'].rootAssembly
a1.DatumCsysByDefault(CARTESIAN)
p = mdb.models['Model-1'].parts['Part-11']
a1.Instance(name='Part-11-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-12']
a1.Instance(name='Part-12-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-13']
a1.Instance(name='Part-13-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-14']
a1.Instance(name='Part-14-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
```

```
p = mdb.models['Model-1'].parts['Part-21']
a1.Instance(name='Part-21-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-22']
a1.Instance(name='Part-22-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-23']
a1.Instance(name='Part-23-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-24']
a1.Instance(name='Part-24-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-31']
a1.Instance(name='Part-31-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-32']
a1.Instance(name='Part-32-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-33']
a1.Instance(name='Part-33-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
p = mdb.models['Model-1'].parts['Part-34']
a1.Instance(name='Part-34-1', part=p, dependent=OFF)
a1 = mdb.models['Model-1'].rootAssembly
# Pattern for more than one unique cell
a = mdb.models['Model-1'].rootAssembly
a.LinearInstancePattern(instanceList=('Part-11-1', 'Part-12-1', 'Part-13-1',
    'Part-14-1', 'Part-21-1', 'Part-22-1', 'Part-23-1', 'Part-24-1',
    'Part-31-1', 'Part-32-1', 'Part-33-1', 'Part-34-1'), direction1=(1.0,
    0.0, 0.0), direction2=(0.0, 1.0, 0.0), number1=XNUC, number2=YNUC,
    spacing1=1.0, spacing2=sqrt(3.))
session.viewports['Viewport: 1'].view.fitView()
# InstanceFromBooleanMerge
a1 = mdb.models['Model-1'].rootAssembly
a1.InstanceFromBooleanMerge(name='FIBBI', instances=(a1.instances.values()),
  keepIntersections=ON, originalInstances=DELETE, domain=GEOMETRY)
a = mdb.models['Model-1'].rootAssembly
# Mesh
a1.makeIndependent(instances=(a1.instances['FIBBI-1'], ))
partInstances =(a.instances['FIBBI-1'], )
a1.seedPartInstance(regions=partInstances, size=ele, deviationFactor=0.1)
a = mdb.models['Model-1'].rootAssembly
c1 = a.instances['FIBBI-1'].cells
cel = c1.pointsOn
pickedRegions = c1.findAt(cel[0],)
for i in range(1,len(cel)):
    pickedRegions += c1.findAt(cel[i],)
a.setMeshControls(regions=pickedRegions, elemShape=eval(eleshape), technique=
    SWEEP)
elemType1 = mesh.ElemType(eval(eletyp))
pickedRegions = (pickedRegions,)
a.setElementType(regions=(pickedRegions), elemTypes=(elemType1, ))
partInstances =(a.instances['FIBBI-1'], )
a.generateMesh(regions=partInstances)
###########################################################################
# Write input file
###########################################################################
#The name of the operating system dependent module imported.
#The following names have currently been registered:
#'posix', 'nt', 'os2', 'ce', 'java', 'riscos'.
if "posix" in os.name:
#    print 'LINUX'
# Cluster
#    mdb.Job(name=inpname, model='Model-1', type=ANALYSIS,
#        explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
#        description='', parallelizationMethodExplicit=DOMAIN,
```

```
#          multiprocessingMode=DEFAULT, numDomains=1, userSubroutine='',
#          numCpus=1, preMemory=16056.0, standardMemory=16056.0,
#          standardMemoryPolicy=MODERATE, scratch='', echoPrint=OFF,
#          modelPrint=OFF, contactPrint=OFF, historyPrint=OFF)
# Local
    mdb.Job(name=inpname, model='Model-1', description='', type= ANALYSIS,
        atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
        memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
        explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, echoPrint=OFF,
        modelPrint=OFF, contactPrint=OFF, historyPrint=OFF, userSubroutine='',
        scratch='', parallelizationMethodExplicit=DOMAIN, numDomains=1,
        activateLoadBalancing=False, multiprocessingMode=DEFAULT, numCpus=1)
    mdb.jobs[inpname].writeInput(consistencyChecking=OFF)
elif "nt" in os.name:
#    print 'WINDOWS'
    mdb.Job(name=inpname, model='Model-1', type=ANALYSIS,
        explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
        description='', parallelizationMethodExplicit=DOMAIN,
        multiprocessingMode=DEFAULT, numDomains=1, userSubroutine='',
        numCpus=1, memory=50, memoryUnits=PERCENTAGE, scratch='',
        echoPrint=OFF, modelPrint=OFF, contactPrint=OFF,
        historyPrint=OFF)
    mdb.jobs[inpname].setValues( \
        userSubroutine='C:\\SIMULIA\\Abaqus_Temp\\umat3\\rate_ana.for')
    mdb.jobs[inpname].writeInput(consistencyChecking=OFF)
##############################################################################
# Open input file
##############################################################################
f = open(inpname+".inp",'r+')
read = str(f.readline())
while read[0:5] != '*Node':
    read = str(f.readline())
##############################################################################
# Generate list of nodes
##############################################################################
print 'Read nodes'
nodes = []
data = []
read = str(f.readline())
while read[:8] != '*Element':
    data = read.split(",")
    data = map(float, data)
    data[0] = int(data[0])
    nodes.append(data)
    read = str(f.readline())
print len(nodes), 'Nodes'
##############################################################################
# Generate list of elements
##############################################################################
# read elements until element definition is reached
# for C3D20 element definition need two rows: if necessary
print 'Elemente einlesen!'
elements = []
data = []
# HEX
if str(eletyp)[0:4]=='C3D8':
  print 'HEX C3D8'
    read = str(f.readline())
elif str(eletyp)[0:5]=='C3D20':
  read = str(f.readline()) + str(f.readline())
  print 'HEX C3D20'
# WEDGE
elif str(eletyp)[0:4]=='C3D6':
  read = str(f.readline())
  print 'WEDGE C3D6'
elif str(eletyp)[0:5]=='C3D15':
```

```
    read = str(f.readline())
    print 'WEDGE C3D15'
  else:
    print 'Elementtyp nicht vorgesehen! Code ueberpruefen'
  while read[:5] != '*Nset':
    data = read.split(",")
    elements.append(map(int, data))
    if eval(eletyp)==C3D8:
      read = str(f.readline())
    if eval(eletyp)==C3D20:
      read = str(f.readline()) + str(f.readline())
    if eval(eletyp)==C3D6:
      read = str(f.readline())
    if eval(eletyp)==C3D15:
      read = str(f.readline())
  print len(elements), 'Elements'
  print 'Sort for LDBC and PBC ...'
  ##############################################################################
  # Sorting node couples of the surface
  ##############################################################################
  stellen = 6
  ecken = []

  kantenXpYpZ = []
  kantenXmYpZ = []
  kantenXpYmZ = []
  kantenXmYmZ = []

  kantenYpXpZ = []
  kantenYmXpZ = []
  kantenYpXmZ = []
  kantenYmXmZ = []

  kantenZpXpY = []
  kantenZmXpY = []
  kantenZpXmY = []
  kantenZmXmY = []

  flaechenXm = []
  flaechenXp = []
  flaechenYm = []
  flaechenYp = []
  flaechenZm = []
  flaechenZp = []

  innen = []

  for i in range(len(nodes)):
      #Ecke1
      if round(nodes[i][1],stellen) == +round(xsize/2.,stellen) and \
         round(nodes[i][2],stellen) == +round(ysize/2.,stellen) and \
         round(nodes[i][3],stellen) == +round(    0.    ,stellen):
          ecken.append(nodes[i][0])
      #Ecke2
      elif round(nodes[i][1],stellen) == +round(xsize/2.,stellen) and \
           round(nodes[i][2],stellen) == -round(ysize/2.,stellen) and \
           round(nodes[i][3],stellen) == +round(    0.    ,stellen):
          ecken.append(nodes[i][0])
      #Ecke3
      elif round(nodes[i][1],stellen) == -round(xsize/2.,stellen) and \
           round(nodes[i][2],stellen) == -round(ysize/2.,stellen) and \
           round(nodes[i][3],stellen) == +round(    0.    ,stellen):
          ecken.append(nodes[i][0])
      #Ecke4
      elif round(nodes[i][1],stellen) == -round(xsize/2.,stellen) and \
           round(nodes[i][2],stellen) == +round(ysize/2.,stellen) and \
```

```
        round(nodes[i][3],stellen) == +round(    0.    ,stellen):
        ecken.append(nodes[i][0])
    #Ecke5
    elif round(nodes[i][1],stellen) == -round(xsize/2.,stellen) and \
         round(nodes[i][2],stellen) == +round(ysize/2.,stellen) and \
         round(nodes[i][3],stellen) == +round(zsize    ,stellen):
        ecken.append(nodes[i][0])
    #Ecke6
    elif round(nodes[i][1],stellen) == +round(xsize/2.,stellen) and \
         round(nodes[i][2],stellen) == +round(ysize/2.,stellen) and \
         round(nodes[i][3],stellen) == +round(zsize    ,stellen):
        ecken.append(nodes[i][0])
    #Ecke7
    elif round(nodes[i][1],stellen) == +round(xsize/2.,stellen) and \
         round(nodes[i][2],stellen) == -round(ysize/2.,stellen) and \
         round(nodes[i][3],stellen) == +round(zsize    ,stellen):
        ecken.append(nodes[i][0])
    #Ecke8
    elif round(nodes[i][1],stellen) == -round(xsize/2.,stellen) and \
         round(nodes[i][2],stellen) == -round(ysize/2.,stellen) and \
         round(nodes[i][3],stellen) == +round(zsize    ,stellen):
        ecken.append(nodes[i][0])

    #Kanten X+Y-Z
    elif round(abs(nodes[i][1]),stellen) != round(xsize/2.,stellen)  and \
         round(    (nodes[i][2]),stellen) == +round(ysize/2.,stellen)  and \
         round(abs(nodes[i][3]),stellen) == round(0.,stellen):
        kantenXpYmZ.append(nodes[i][0])
    #Kanten X-Y-Z
    elif round(abs(nodes[i][1]),stellen) != round(xsize/2.,stellen)  and \
         round(    (nodes[i][2]),stellen) == -round(ysize/2.,stellen)  and \
         round(abs(nodes[i][3]),stellen) == round(0.,stellen):
        kantenXmYmZ.append(nodes[i][0])
    #Kanten X+Y+Z
    elif round(abs(nodes[i][1]),stellen) != round(xsize/2.,stellen)  and \
         round(    (nodes[i][2]),stellen) == +round(ysize/2.,stellen)  and \
         round(abs(nodes[i][3]),stellen) == round(zsize,stellen):
        kantenXpYpZ.append(nodes[i][0])
    #Kanten X-Y+Z
    elif round(abs(nodes[i][1]),stellen) != round(xsize/2.,stellen)  and \
         round(    (nodes[i][2]),stellen) == -round(ysize/2.,stellen)  and \
         round(abs(nodes[i][3]),stellen) == round(zsize,stellen):
        kantenXmYpZ.append(nodes[i][0])

    #Kanten Y+X-Z
    elif round(    (nodes[i][1]),stellen) == +round(xsize/2.,stellen)  and \
         round(abs(nodes[i][2]),stellen) != round(ysize/2.,stellen)  and \
         round(abs(nodes[i][3]),stellen) == round(0.,stellen):
        kantenYpXmZ.append(nodes[i][0])
    #Kanten Y-X-Z
    elif round(    (nodes[i][1]),stellen) == -round(xsize/2.,stellen)  and \
         round(abs(nodes[i][2]),stellen) != round(ysize/2.,stellen)  and \
         round(abs(nodes[i][3]),stellen) == round(0.,stellen):
        kantenYmXmZ.append(nodes[i][0])
    #Kanten Y+X+Z
    elif round(    (nodes[i][1]),stellen) == +round(xsize/2.,stellen)  and \
         round(abs(nodes[i][2]),stellen) != round(ysize/2.,stellen)  and \
         round(abs(nodes[i][3]),stellen) == round(zsize,stellen):
        kantenYpXpZ.append(nodes[i][0])
    #Kanten Y-X+Z
    elif round(    (nodes[i][1]),stellen) == -round(xsize/2.,stellen)  and \
         round(abs(nodes[i][2]),stellen) != round(ysize/2.,stellen)  and \
         round(abs(nodes[i][3]),stellen) == round(zsize,stellen):
        kantenYmXpZ.append(nodes[i][0])

    #Kanten  Z+X-Y
```

```
        elif round(    (nodes[i][1]),stellen) == +round(xsize/2.,stellen)  and \
             round(    (nodes[i][2]),stellen) == -round(ysize/2.,stellen)  and \
             round(abs(nodes[i][3]),stellen) != round(0.,stellen)          and \
             round(abs(nodes[i][3]),stellen) != round(zsize,stellen):
            kantenZpXmY.append(nodes[i][0])
         #Kanten  Z-X-Y
        elif round(    (nodes[i][1]),stellen) == -round(xsize/2.,stellen)  and \
             round(    (nodes[i][2]),stellen) == -round(ysize/2.,stellen)  and \
             round(abs(nodes[i][3]),stellen) != round(0.,stellen)          and \
             round(abs(nodes[i][3]),stellen) != round(zsize,stellen):
            kantenZmXmY.append(nodes[i][0])
        #Kanten  Z+X+Y
        elif round(    (nodes[i][1]),stellen) == +round(xsize/2.,stellen)  and \
             round(    (nodes[i][2]),stellen) == +round(ysize/2.,stellen)  and \
             round(abs(nodes[i][3]),stellen) != round(0.,stellen)          and \
             round(abs(nodes[i][3]),stellen) != round(zsize,stellen):
            kantenZpXpY.append(nodes[i][0])
        #Kanten  Z-X+Y
        elif round(    (nodes[i][1]),stellen) == -round(xsize/2.,stellen)  and \
             round(    (nodes[i][2]),stellen) == +round(ysize/2.,stellen)  and \
             round(abs(nodes[i][3]),stellen) != round(0.,stellen)          and \
             round(abs(nodes[i][3]),stellen) != round(zsize,stellen):
            kantenZmXpY.append(nodes[i][0])

        #Minus (m) and Plus (p) side
        #Flaechen Xp
        elif round((nodes[i][1]),stellen) == round(xsize/2.,stellen)  and \
             round(abs(nodes[i][2]),stellen) != round(ysize/2.,stellen)  and \
             round(abs(nodes[i][3]),stellen) != round(0.,stellen):
            flaechenXp.append(nodes[i][0])
        #Flaechen Xm
        elif round((nodes[i][1]),stellen) == -round(xsize/2.,stellen)  and \
             round(abs(nodes[i][2]),stellen) != round(ysize/2.,stellen)  and \
             round(abs(nodes[i][3]),stellen) != round(zsize,stellen):
            flaechenXm.append(nodes[i][0])
        #Flaechen Yp
        elif round(abs(nodes[i][1]),stellen) != round(xsize/2.,stellen)  and \
             round((nodes[i][2]),stellen) == round(ysize/2.,stellen)  and \
             round(abs(nodes[i][3]),stellen) != round(0.,stellen):
            flaechenYp.append(nodes[i][0])
        #Flaechen Ym
        elif round(abs(nodes[i][1]),stellen) != round(xsize/2.,stellen)  and \
             round((nodes[i][2]),stellen) == -round(ysize/2.,stellen)  and \
             round(abs(nodes[i][3]),stellen) != round(zsize,stellen):
            flaechenYm.append(nodes[i][0])
        #Flaechen Zp
        elif round(abs(nodes[i][1]),stellen) != round(xsize/2.,stellen)  and \
             round(abs(nodes[i][2]),stellen) != round(ysize/2.,stellen)  and \
             round(abs(nodes[i][3]),stellen) == round(zsize,stellen):
            flaechenZp.append(nodes[i][0])
        #Flaechen Zm
        elif round(abs(nodes[i][1]),stellen) != round(xsize/2.,stellen)  and \
             round(abs(nodes[i][2]),stellen) != round(ysize/2.,stellen)  and \
             round(abs(nodes[i][3]),stellen) == round(0.,stellen):
            flaechenZm.append(nodes[i][0])
        #Innen
        else:
            innen.append(nodes[i][0])
        #define first internal node as origin

#    ursprung = innen[0]
    ##################################################################
    # Additional sorting for PBC
    ##################################################################
    if rb == 'PBC':
        print 'Zusatzsortierung PBC ...'
```

```
# Sorting opposite nodes in the list
# Sorting corners
eckensort = []

for i in range(len(ecken)):
    #Ecke1
    if round(nodes[ecken[i]-1][1],stellen) == +round(xsize/2.,stellen) and \
       round(nodes[ecken[i]-1][2],stellen) == +round(ysize/2.,stellen) and \
       round(nodes[ecken[i]-1][3],stellen) == +round(    0.    ,stellen):
        eckensort.append(ecken[i])
        break
for i in range(len(ecken)):
    #Ecke2
    if round(nodes[ecken[i]-1][1],stellen) == +round(xsize/2.,stellen) and \
       round(nodes[ecken[i]-1][2],stellen) == -round(ysize/2.,stellen) and \
       round(nodes[ecken[i]-1][3],stellen) == +round(    0.    ,stellen):
        eckensort.append(ecken[i])
        break
eckensort.append(ecken[i])
for i in range(len(ecken)):
    #Ecke3
    if round(nodes[ecken[i]-1][1],stellen) == -round(xsize/2.,stellen) and \
       round(nodes[ecken[i]-1][2],stellen) == -round(ysize/2.,stellen) and \
       round(nodes[ecken[i]-1][3],stellen) == +round(    0.    ,stellen):
        eckensort.append(ecken[i])
        break
eckensort.append(ecken[i])
for i in range(len(ecken)):
    #Ecke4
    if round(nodes[ecken[i]-1][1],stellen) == -round(xsize/2.,stellen) and \
       round(nodes[ecken[i]-1][2],stellen) == +round(ysize/2.,stellen) and \
       round(nodes[ecken[i]-1][3],stellen) == +round(    0.    ,stellen):
        eckensort.append(ecken[i])
        break
eckensort.append(ecken[i])
for i in range(len(ecken)):
    #Ecke5
    if round(nodes[ecken[i]-1][1],stellen) == -round(xsize/2.,stellen) and \
       round(nodes[ecken[i]-1][2],stellen) == +round(ysize/2.,stellen) and \
       round(nodes[ecken[i]-1][3],stellen) == +round(zsize    ,stellen):
        eckensort.append(ecken[i])
        break
eckensort.append(ecken[i])
for i in range(len(ecken)):
    #Ecke6
    if round(nodes[ecken[i]-1][1],stellen) == +round(xsize/2.,stellen) and \
       round(nodes[ecken[i]-1][2],stellen) == +round(ysize/2.,stellen) and \
       round(nodes[ecken[i]-1][3],stellen) == +round(zsize    ,stellen):
        eckensort.append(ecken[i])
        break
eckensort.append(ecken[i])
for i in range(len(ecken)):
    #Ecke7
```

```
            if round(nodes[ecken[i]-1][1],stellen) == +round(xsize/2.,stellen) and \
               round(nodes[ecken[i]-1][2],stellen) == -round(ysize/2.,stellen) and \
               round(nodes[ecken[i]-1][3],stellen) == +round(zsize    ,stellen):
                eckensort.append(ecken[i])
                break
        eckensort.append(ecken[i])
        for i in range(len(ecken)):
            #Ecke8
            if round(nodes[ecken[i]-1][1],stellen) == -round(xsize/2.,stellen) and \
               round(nodes[ecken[i]-1][2],stellen) == -round(ysize/2.,stellen) and \
               round(nodes[ecken[i]-1][3],stellen) == +round(zsize    ,stellen):
                eckensort.append(ecken[i])
                break
        # Sorting edges

        kantenXsort = []
        kantenYsort = []
        kantenZsort = []

        Xkanten = len(kantenXpYmZ)
        for i in range(Xkanten):
            Xvalue = round(nodes[kantenXpYmZ[i]-1][1],stellen)
            kantenXsort.append(kantenXpYmZ[i])

            for j in range(Xkanten):
                if Xvalue == round(nodes[kantenXmYmZ[j]-1][1],stellen):
                    break
            kantenXsort.append(kantenXmYmZ[j])
            kantenXsort.append(kantenXmYmZ[j])

            for j in range(Xkanten):
                if Xvalue == round(nodes[kantenXmYpZ[j]-1][1],stellen):
                    break
            kantenXsort.append(kantenXmYpZ[j])
            kantenXsort.append(kantenXmYpZ[j])

            for j in range(Xkanten):
                if Xvalue == round(nodes[kantenXpYpZ[j]-1][1],stellen):
                    break
            kantenXsort.append(kantenXpYpZ[j])
#         print 'Kanten X sortiert'
#         print kantenXsort

        Ykanten = len(kantenYpXmZ)
        for i in range(Ykanten):
            Yvalue = round(nodes[kantenYpXmZ[i]-1][2],stellen)
            kantenYsort.append(kantenYpXmZ[i])

            for j in range(Ykanten):
                if Yvalue == round(nodes[kantenYmXmZ[j]-1][2],stellen):
                    break
            kantenYsort.append(kantenYmXmZ[j])
            kantenYsort.append(kantenYmXmZ[j])

            for j in range(Ykanten):
                if Yvalue == round(nodes[kantenYmXpZ[j]-1][2],stellen):
                    break
            kantenYsort.append(kantenYmXpZ[j])
            kantenYsort.append(kantenYmXpZ[j])

            for j in range(Ykanten):
                if Yvalue == round(nodes[kantenYpXpZ[j]-1][2],stellen):
```

```
                    break
                kantenYsort.append(kantenYpXpZ[j])
#           print 'Kanten Y sortiert'
#           print kantenYsort

            Zkanten = len(kantenZpXmY)
            for i in range(Zkanten):
                Yvalue = round(nodes[kantenZpXmY[i]-1][3],stellen)
                kantenZsort.append(kantenZpXmY[i])

                for j in range(Zkanten):
                    if Yvalue == round(nodes[kantenZmXmY[j]-1][3],stellen):
                        break
                kantenZsort.append(kantenZmXmY[j])
                kantenZsort.append(kantenZmXmY[j])

                for j in range(Zkanten):
                    if Yvalue == round(nodes[kantenZmXpY[j]-1][3],stellen):
                        break
                kantenZsort.append(kantenZmXpY[j])
                kantenZsort.append(kantenZmXpY[j])

                for j in range(Zkanten):
                    if Yvalue == round(nodes[kantenZpXpY[j]-1][3],stellen):
                        break
                kantenZsort.append(kantenZpXpY[j])
#           print 'Kanten Z sortiert'
#           print kantenZsort

            flaechenXsort = []
            flaechenYsort = []
            flaechenZsort = []

            Xflaechen = len(flaechenXp)
            for i in range(Xflaechen):
                Yvalue = round(nodes[flaechenXp[i]-1][2],stellen)
                Zvalue = round(nodes[flaechenXp[i]-1][3],stellen)
                flaechenXsort.append(flaechenXp[i])
                for j in range(Xflaechen):
                    if Yvalue == round(nodes[flaechenXm[j]-1][2],stellen) and \
                       Zvalue == round(nodes[flaechenXm[j]-1][3],stellen):
                        break
                flaechenXsort.append(flaechenXm[j])
#           print 'Flaechen X sortiert'
#           print flaechenXsort

            Yflaechen = len(flaechenYp)
            for i in range(Yflaechen):
                Xvalue = round(nodes[flaechenYp[i]-1][1],stellen)
                Zvalue = round(nodes[flaechenYp[i]-1][3],stellen)
                flaechenYsort.append(flaechenYp[i])
                for j in range(Yflaechen):
                    if Xvalue == round(nodes[flaechenYm[j]-1][1],stellen) and \
                       Zvalue == round(nodes[flaechenYm[j]-1][3],stellen):
                        break
                flaechenYsort.append(flaechenYm[j])
#           print 'Flaechen Y sortiert'
#           print flaechenYsort

            Zflaechen = len(flaechenZp)
            for i in range(Zflaechen):
                Xvalue = round(nodes[flaechenZp[i]-1][1],stellen)
                Yvalue = round(nodes[flaechenZp[i]-1][2],stellen)
                flaechenZsort.append(flaechenZp[i])
                for j in range(Zflaechen):
                    if Xvalue == round(nodes[flaechenZm[j]-1][1],stellen) and \
```

```
                    Yvalue == round(nodes[flaechenZm[j]-1][2],stellen):
                    break
            flaechenZsort.append(flaechenZm[j])
#         print 'Flaechen Z sortiert'
#         print flaechenZsort
        # Merge all lists
        nrf = []
        nrf.extend(eckensort)
        nrf.extend(kantenXsort)
        nrf.extend(kantenYsort)
        nrf.extend(kantenZsort)
        nrf.extend(flaechenXsort)
        nrf.extend(flaechenYsort)
        nrf.extend(flaechenZsort)
        nrf.extend(innen)

        nodesneu = []
        i=0
        while i != len(nrf):
        #     if i % 1000 == 0:
        #         print 'i',i
            if i < len(nrf)-1:
                if nrf[i] == nrf[i+1]:
                    for k in range(len(nodes)):
                        if nodes[k][0] == nrf[i]:
                            break
                    nodesneu.append(nodes[k])
                    nodesneu.append(nodes[k])
                    nodes.pop(k)
                    i+=2
                else:
                    for k in range(len(nodes)):
                        if nodes[k][0] == nrf[i]:
                            break
                    nodesneu.append(nodes[k])
                    nodes.pop(k)
                    i+=1
            else:
                for k in range(len(nodes)):
                    if nodes[k][0] == nrf[i]:
                        break
                nodesneu.append(nodes[k])
                nodes.pop(k)
                i+=1

        letzteoberflaeche = len(nrf)-len(innen)-1
        nodes = nodesneu

    elif rb=='LDBC':
        # Merge all lists
        nrf = []
        nrf.extend(ecken)
        nrf.extend(kantenXpYpZ)
        nrf.extend(kantenXmYpZ)
        nrf.extend(kantenXpYmZ)
        nrf.extend(kantenXmYmZ)
        nrf.extend(kantenYpXpZ)
        nrf.extend(kantenYmXpZ)
        nrf.extend(kantenYpXmZ)
        nrf.extend(kantenYmXmZ)
        nrf.extend(kantenZpXpY)
        nrf.extend(kantenZmXpY)
        nrf.extend(kantenZpXmY)
        nrf.extend(kantenZmXmY)
        nrf.extend(flaechenXm)
        nrf.extend(flaechenXp)
```

```
        nrf.extend(flaechenYm)
        nrf.extend(flaechenYp)
        nrf.extend(flaechenZm)
        nrf.extend(flaechenZp)
        nrf.extend(innen)
        # Sorting nodes list in correspondance to nrf
        ifortschritt = 0
        i = 0
        for i in range(len(nrf)):
            for k in range(len(nodes)):
                if nodes[k][0]==nrf[i]:
                    break
            nodes[k],nodes[i] = nodes[i],nodes[k]
        letzteoberflaeche = len(nrf)-1

    ##############################################################################
    # Write input file
    ##############################################################################
    shutil.copyfile(inpname+".inp", inpname+"_backup.inp")
    f.seek(0,0)
    while read[:5] != '*Node':
        read = str(f.readline())
    f.truncate()
    f.write('%d,0.0,0.0,0.0 \n' %(len(nodes)+1))
    f.write('%d,0.0,0.0,0.0 \n' %(len(nodes)+2))
    f.write('%d,0.0,0.0,0.0 \n' %(len(nodes)+3))

    b = open(inpname+"_backup.inp",'r+')
    lines1 = b.readlines()
    b.close()
    os.remove(inpname+"_backup.inp")
    for i in range(17,len(lines1)):
        f.write(lines1[i])

    f.seek(0,0)

    # Append additional nodes
    print 'Inputdatei verfollstaendigen ...'
    while read[:11] != '** Section:':
        read = str(f.readline())
    f.truncate()
    if "posix" in os.name:
        f.seek(-17,2)

    f.write('** Section: Fiber1\n')
    f.write('*Solid Section, elset=_PickedSet2_#1, material=Fiber1\n')
    f.write(',\n')
    f.write('** Section: Fiber1\n')
    f.write('*Solid Section, elset=_PickedSet2_#2, material=Fiber1\n')
    f.write(',\n')
    f.write('** Section: Fiber1\n')
    f.write('*Solid Section, elset=_PickedSet2_#3, material=Fiber1\n')
    f.write(',\n')
    f.write('** Section: Fiber1\n')
    f.write('*Solid Section, elset=_PickedSet2_#4, material=Fiber1\n')
    f.write(',\n')
    f.write('** Section: Matrix\n')
    f.write('*Solid Section, elset=_PickedSet2_#5, material=Matrix\n')
    f.write(',\n')
    f.write('** Section: Matrix\n')
    f.write('*Solid Section, elset=_PickedSet2_#6, material=Matrix\n')
    f.write(',\n')
    f.write('** Section: Matrix\n')
    f.write('*Solid Section, elset=_PickedSet2_#7, material=Matrix\n')
    f.write(',\n')
    f.write('** Section: Matrix\n')
```

```
    f.write('*Solid Section, elset=_PickedSet2_#8, material=Matrix\n')
    f.write(',\n')
    f.write('** Section: Fiber1\n')
    f.write('*Solid Section, elset=_PickedSet2_#9, material=Fiber1\n')
    f.write(',\n')
    f.write('** Section: Fiber1\n')
    f.write('*Solid Section, elset=_PickedSet2_#10, material=Fiber1\n')
    f.write(',\n')
    f.write('** Section: Fiber1\n')
    f.write('*Solid Section, elset=_PickedSet2_#11, material=Fiber1\n')
    f.write(',\n')
    f.write('** Section: Fiber1\n')
    f.write('*Solid Section, elset=_PickedSet2_#12, material=Fiber1\n')
    f.write(',\n')
    # Define equation
    f.write('**\n')
    f.write('** Constraint: Constraint-1 \n')
    f.write('**\n')
    f.write('*Equation \n')
    if rb == 'LDBC':
    ##########################################################################
    # Linear displacement boundary conditions
    ##########################################################################
        print 'Linear displacement boundary conditions'
        #DOF1
        for i in range(0,letzteoberflaeche+1):
            f.write("4\n")
            f.write("%d,1,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
                %(nodes[i][0],len(nodes)+1,-nodes[i][1],len(nodes)+1, \
                -nodes[i][2],len(nodes)+1,-nodes[i][3]))
        #DOF2
        for i in range(0,letzteoberflaeche+1):
            f.write("4\n")
            f.write("%d,2,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
                %(nodes[i][0],len(nodes)+2,-nodes[i][1],len(nodes)+2, \
                -nodes[i][2],len(nodes)+2,-nodes[i][3]))
        #DOF3
        for i in range(0,letzteoberflaeche+1):
            f.write("4\n")
            f.write("%d,3,1\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n" \
                %(nodes[i][0],len(nodes)+3,-nodes[i][1],len(nodes)+3, \
                -nodes[i][2],len(nodes)+3,-nodes[i][3]))

    elif rb == 'PBC':
    ##########################################################################
    # Periodic boundary counditions
    ##########################################################################
        print 'Periodic boundary counditions'
        #DOF1
        for i in range(0,letzteoberflaeche+1,2):
            f.write("5\n")
            f.write("%d,1,1.0\n%d,1,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
              %(nodes[i][0], \
                nodes[i+1][0], \
                len(nodes)+1,(nodes[i+1][1]-nodes[i][1]), \
                len(nodes)+1,(nodes[i+1][2]-nodes[i][2]), \
                len(nodes)+1,(nodes[i+1][3]-nodes[i][3])))
        #DOF2
        for i in range(0,letzteoberflaeche+1,2):
            f.write("5\n")
            f.write("%d,2,1.0\n%d,2,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
              %(nodes[i][0], \
                nodes[i+1][0], \
                len(nodes)+2,(nodes[i+1][1]-nodes[i][1]), \
                len(nodes)+2,(nodes[i+1][2]-nodes[i][2]), \
                len(nodes)+2,(nodes[i+1][3]-nodes[i][3])))
```

```
        #DOF3
        for i in range(0,letzteoberflaeche+1,2):
            f.write("5\n")
            f.write("%d,3,1.0\n%d,3,-1.0\n%d,1,%.9f\n%d,2,%.9f\n%d,3,%.9f\n"
              %(nodes[i][0], \
                nodes[i+1][0], \
                len(nodes)+3,(nodes[i+1][1]-nodes[i][1]), \
                len(nodes)+3,(nodes[i+1][2]-nodes[i][2]), \
                len(nodes)+3,(nodes[i+1][3]-nodes[i][3])))
    f.write('*End Instance \n')
    # Determine origin
    ursprung = 'kein'
    for i in range(0,len(nodes)):
        if nodes[i][1]==0. and nodes[i][2]==0. and round(nodes[i][3],stellen)==
            round(zsize/2.,stellen):
            ursprung=nodes[i][0]
    if ursprung == 'kein':
        print 'No origin, check mesh'
    # Generate node sets for boundary of the artificial nodes
    f.write('*Nset, nset=K1, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(len(nodes)+1))
    f.write('*Nset, nset=K2, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(len(nodes)+2))
    f.write('*Nset, nset=K3, internal, instance=FIBBI-1 \n')
    f.write('%d, \n' %(len(nodes)+3))

    if rb != 'LDBC':
        f.write('*Nset, nset=Ursprung, internal, instance=FIBBI-1 \n')
        f.write('%d, \n' %ursprung)

    f.write('*End Assembly \n')
    f.write('** \n')
    f.write('** MATERIALS\n')
    f.write('** \n')
# ==============================================================
# Read material parameters
mat = open('0_material.py','r')
for line in mat:
    #read = str(mat.readline())
    f.write('%s' %line)
mat.close()
print '=============================================================',
print 'End RVE subroutine'
print '=============================================================',
```

Appendix B

Python scripts to calculate the stiffness tetrads

B.1 Start-up calculation "0_A.py"

```
# ======================================================================
# Start-up calculation 1
# ======================================================================
# ======================================================================
# Start with: nohup /opt/abaqus/Commands/abaqus cae noGUI=*.py &
# ======================================================================
from abaqus import *
from abaqusConstants import *
import __main__
import sys
import string
import fileinput
import shutil
import os
import subprocess
import time
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import odbAccess
import random
import csv
import resource
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
```

```
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
# ======================================================================
execfile('0_subroutines.py')
# ======================================================================
# 0_A.py PART1
# ======================================================================
extname = 'A'
H2 = nullmatrix(3,3)
vH2 = nullvektor(6)
# Define displacement gradient H
H0 = nullmatrix(3,3)
H0[0][0] = 0.
H0[0][1] = 0.5
H0[0][2] = 0.
H0[1][0] = 0.
H0[1][1] = 0.
H0[1][2] = 0.
H0[2][0] = 0.
H0[2][1] = 0.
H0[2][2] = 0.
print 'H0'
for line in H0: print line
ausH0 = open('K0_ANLAUFRECHNUNG_H0.txt','w')
for i in range(3):
    for j in range(3):
      ausH0.write('%s' %(H0[i][j]))
      if j != 2: ausH0.write(',')
    ausH0.write('\n')
ausH0.close()
# ======================================================================
execfile('0_RVE.py')
# ======================================================================
# 0_A.py  PART2
# ======================================================================
# STEP 1 Large Deformation
# ======================================================================
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step1.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: Kuenstliche Knoten Type: Displacement/Rotation\n")
f.write("*Boundary \n")
if H0[0][0] != '-':
    f.write("K1, 1, 1, %.9f \n" %(float(H0[0][0])))
if H0[0][1] != '-':
    f.write("K1, 2, 2, %.9f \n" %(float(H0[0][1])))
if H0[0][2] != '-':
    f.write("K1, 3, 3, %.9f \n" %(float(H0[0][2])))
if H0[1][0] != '-':
```

```
    f.write("K2, 1, 1, %.9f \n" %(float(H0[1][0])))
if H0[1][1] != '-':
    f.write("K2, 2, 2, %.9f \n" %(float(H0[1][1])))
if H0[1][2] != '-':
    f.write("K2, 3, 3, %.9f \n" %(float(H0[1][2])))
if H0[2][0] != '-':
    f.write("K3, 1, 1, %.9f \n" %(float(H0[2][0])))
if H0[2][1] != '-':
    f.write("K3, 2, 2, %.9f \n" %(float(H0[2][1])))
if H0[2][2] != '-':
    f.write("K3, 3, 3, %.9f \n" %(float(H0[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
# Output
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==========================================================
# Unload in case of uni-directional tension
# ==========================================================
if H0[0][0] == '-' or H0[1][1] == '-' or H0[2][2] == '-':
    # ======================================================
    # STEP 2 Unload Remove H11 H22 H33
    # ======================================================
    f.write('** ----------------------------------------------------------------\n')
    #Step Infos einlesen
    step = open('0_step2.py','r')
    for line in step:
        f.write('%s' %line)
    step.close()
    #Boundary Conditions
    f.write('** \n')
    f.write('** BOUNDARY CONDITIONS\n')
    f.write('** \n')
    f.write('*Boundary, op=NEW\n')
    #if H0[0][0] != '-':
        #f.write("K1, 1, 1, %.9f \n" %(float(H0[0][0])))
    if H0[0][1] != '-':
        f.write("K1, 2, 2, %.9f \n" %(float(H0[0][1])))
    if H0[0][2] != '-':
        f.write("K1, 3, 3, %.9f \n" %(float(H0[0][2])))
    if H0[1][0] != '-':
        f.write("K2, 1, 1, %.9f \n" %(float(H0[1][0])))
    #if H0[1][1] != '-':
        #f.write("K2, 2, 2, %.9f \n" %(float(H0[1][1])))
    if H0[1][2] != '-':
        f.write("K2, 3, 3, %.9f \n" %(float(H0[1][2])))
    if H0[2][0] != '-':
        f.write("K3, 1, 1, %.9f \n" %(float(H0[2][0])))
```

```
    if H0[2][1] != '-':
        f.write("K3, 2, 2, %.9f \n" %(float(H0[2][1])))
    #if H0[2][2] != '-':
        #f.write("K3, 3, 3, %.9f \n" %(float(H0[2][2])))
    if rb != 'LV':
        f.write("** Name: Ursprung Type: Displacement/Rotation\n")
        f.write("Ursprung, 1, 1, 0 \n" )
        f.write("Ursprung, 2, 2, 0 \n" )
        f.write("Ursprung, 3, 3, 0 \n" )
    #Output
    f.write('** \n')
    f.write('** OUTPUT REQUESTS\n')
    f.write('** \n')
    f.write('*Restart, write, frequency=0\n')
    f.write('** \n')
    f.write('** FIELD OUTPUT: F-Output-1\n')
    f.write('**\n')
    f.write('*Output, field, frequency=1\n')
    f.write('*Node Output\n')
    f.write('CF, RF, U\n')
    f.write('*Element Output, directions=YES\n')
    f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
    f.write('** \n')
    f.write('** HISTORY OUTPUT: H-Output-1\n')
    f.write('** \n')
    f.write('*Output, history, variable=PRESELECT\n')
    f.write('*End Step\n')
    # ==============================================================
    # STEP 3  ENTLASTEN      #H12 H21 entfernen
    # ==============================================================
    f.write('** ----------------------------------------------------------------\n')
    #Step Infos einlesen
    step = open('0_step3.py','r')
    for line in step:
        f.write('%s' %line)
    step.close()
    #Boundary Conditions
    f.write('** \n')
    f.write('** BOUNDARY CONDITIONS\n')
    f.write('** \n')
    f.write('*Boundary, op=NEW\n')
    #if H0[0][0] != '-':
        #f.write("K1, 1, 1, %.9f \n" %(float(H0[0][0])))
    #if H0[0][1] != '-':
        #f.write("K1, 2, 2, %.9f \n" %(float(H0[0][1])))
    if H0[0][2] != '-':
        f.write("K1, 3, 3, %.9f \n" %(float(H0[0][2])))
    if H0[1][0] != '-':
        f.write("K2, 1, 1, %.9f \n" %(float(H0[1][0])))
    #if H0[1][1] != '-':
        #f.write("K2, 2, 2, %.9f \n" %(float(H0[1][1])))
    if H0[1][2] != '-':
        f.write("K2, 3, 3, %.9f \n" %(float(H0[1][2])))
    if H0[2][0] != '-':
        f.write("K3, 1, 1, %.9f \n" %(float(H0[2][0])))
    if H0[2][1] != '-':
        f.write("K3, 2, 2, %.9f \n" %(float(H0[2][1])))
    #if H0[2][2] != '-':
        #f.write("K3, 3, 3, %.9f \n" %(float(H0[2][2])))
    if rb != 'LV':
        f.write("** Name: Ursprung Type: Displacement/Rotation\n")
        f.write("Ursprung, 1, 1, 0 \n" )
        f.write("Ursprung, 2, 2, 0 \n" )
        f.write("Ursprung, 3, 3, 0 \n" )
    #Output
    f.write('** \n')
```

```
    f.write('** OUTPUT REQUESTS\n')
    f.write('** \n')
    f.write('*Restart, write, frequency=0\n')
    f.write('** \n')
    f.write('** FIELD OUTPUT: F-Output-1\n')
    f.write('**\n')
    f.write('*Output, field, frequency=1\n')
    f.write('*Node Output\n')
    f.write('CF, RF, U\n')
    f.write('*Element Output, directions=YES\n')
    f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
    f.write('** \n')
    f.write('** HISTORY OUTPUT: H-Output-1\n')
    f.write('** \n')
    f.write('*Output, history, variable=PRESELECT\n')
    f.write('*End Step\n')
# ==========================================================================
# Unload in any other case
# ==========================================================================
else:
    # ======================================================================
    # STEP 2  Unload - remove H0
    # ======================================================================
    f.write('** ----------------------------------------------------------------\n')
    # Read step information
    step = open('0_step2.py','r')
    for line in step:
        f.write('%s' %line)
    step.close()
    # Boundary conditions
    f.write('** \n')
    f.write('** BOUNDARY CONDITIONS\n')
    f.write('** \n')
    f.write('*Boundary, op=NEW\n')
    if rb != 'LDBC':
        f.write("** Name: Ursprung Type: Displacement/Rotation\n")
        f.write("Ursprung, 1, 1, 0 \n" )
        f.write("Ursprung, 2, 2, 0 \n" )
        f.write("Ursprung, 3, 3, 0 \n" )
    f.write('** \n')
    f.write('** OUTPUT REQUESTS\n')
    f.write('** \n')
    f.write('*Restart, write, frequency=0\n')
    f.write('** \n')
    f.write('** FIELD OUTPUT: F-Output-1\n')
    f.write('**\n')
    f.write('*Output, field, frequency=1\n')
    f.write('*Node Output\n')
    f.write('CF, RF, U\n')
    f.write('*Element Output, directions=YES\n')
    f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
    f.write('** \n')
    f.write('** HISTORY OUTPUT: H-Output-1\n')
    f.write('** \n')
    f.write('*Output, history, variable=PRESELECT\n')
    f.write('*End Step\n')
f.close()
# Save CAE file
mdb.saveAs(pathName=inpname+'.cae')
# ==========================================================================
#     Run Job
# ==========================================================================
print '----------------------------------------------------------------'
print 'Run job'
print '----------------------------------------------------------------'
if "posix" in os.name:
```

```
    execfile('0_job.py')
# Wait for lck file, then wait for lck file is deleted -> job finished
t0 = time.time();
while not os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
while os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
# Check if calculation was successful, else STOP
time.sleep(5)
f = open(inpname+'.sta','r')
sta = []
for line in f:
    sta.append(str(line.rstrip()))
f.close()
if sta[len(sta)-1] == ' THE ANALYSIS HAS COMPLETED SUCCESSFULLY':
    print sta[len(sta)-1]
else:
    print sta[len(sta)-1]
    f = open('0_Auswertung_ERROR_in_'+inpname+'.txt','w')
    f.close()
    time.sleep(5)
    sys.exit("Calculation cancelled")
# ==========================================================
#     Open ODB file, read RF1,RF2,RF3 and K1, K2, K3
# ==========================================================
# Generate path
pfad = inpname
# Search for node number K1NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K1':
    read = str(f.readline())
read = str(f.readline())
K1NR = int(read[0:len(read)-3])
f.close()
# Search for node number K2NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K2':
    read = str(f.readline())
read = str(f.readline())
K2NR = int(read[0:len(read)-3])
f.close()
# Search for node number K3NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K3':
    read = str(f.readline())
read = str(f.readline())
K3NR = int(read[0:len(read)-3])
f.close()
o1 = session.openOdb(
    name=inpname+'.odb')
session.viewports['Viewport: 1'].setValues(displayedObject=o1)
odb = session.odbs[inpname+'.odb']
session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('RF',
    NODAL, ((COMPONENT, 'RF1'), (COMPONENT, 'RF2'), (COMPONENT, 'RF3'), )),
    ), nodeSets=('K1', 'K2', 'K3', ))
session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('U',
    NODAL, ((COMPONENT, 'U1'), (COMPONENT, 'U2'), (COMPONENT, 'U3'), )), ),
    nodeSets=('K1', 'K2', 'K3', ))
increm = len(session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data) - 1

x0 = session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K1NR]
```

```
x1 = session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K1NR]
x2 = session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K1NR]
x3 = session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K2NR]
x4 = session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K2NR]
x5 = session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K2NR]
x6 = session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K3NR]
x7 = session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K3NR]
x8 = session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K3NR]
session.writeXYReport(fileName='K0_ANLAUFRECHNUNG.rpt', appendMode=OFF, xyData=(x0,
    x1,
    x2, x3, x4, x5, x6, x7, x8))

# Generate matrix H1
H1 = nullmatrix(3,3)
H1[0][0] = x0[increm][1]
H1[0][1] = x1[increm][1]
H1[0][2] = x2[increm][1]
H1[1][0] = x3[increm][1]
H1[1][1] = x4[increm][1]
H1[1][2] = x5[increm][1]
H1[2][0] = x6[increm][1]
H1[2][1] = x7[increm][1]
H1[2][2] = x8[increm][1]

del session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K3NR]


# Generate matrix T1PK
T1PK = nullmatrix(3,3)
T1PK[0][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK[0][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK[0][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK[1][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK[1][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK[1][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK[2][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK[2][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK[2][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0

del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR]
```

```
odb.close()

# Calculate T2PK
# F1 = H1 + 1
F1 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F1[i][j] = H1[i][j]

F1[0][0] += 1.
F1[1][1] += 1.
F1[2][2] += 1.
F1i = invert3(F1)
T2PK = matmul(F1i,T1PK)

print 'H1 end'
for line in H1: print line
print 'T1PK end'
for line in T1PK: print line
print 'T2PK end'
for line in T2PK: print line

#ausH.write('H\n')
ausH = open('K0_ANLAUFRECHNUNG_H1.txt','w')
for i in range(3):
    for j in range(3):
      ausH.write('%e' %(H1[i][j]))
      if j != 2: ausH.write(',')
    ausH.write('\n')
ausH.close()

print 'FINISHED'
```

B.2 Start-up calculation "0_A2.py"

```
# ======================================================================
# Start-up calculation 2
# ======================================================================
# ======================================================================
# Start with: nohup /opt/abaqus/Commands/abaqus cae noGUI=*.py &
# ======================================================================
from abaqus import *
from abaqusConstants import *
import __main__
import sys
import string
import fileinput
import shutil
import os
import subprocess
import time
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
```

```
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import odbAccess
import random
import csv
import resource
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
# ==========================================================================
execfile('0_subroutines.py')
# ==========================================================================
# 0_A.py PART1
# ==========================================================================
extname = 'A2'
# Read H0 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H0.txt','r')
H0 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    data[2] = data[2].replace('\n','')
    H0.append(data)
f.close()
print('H0')
for line in H0: print line
# Read H1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H1.txt','r')
H1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    H1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        H1[i][j] = float(H1[i][j])
print('H1')
for line in H1: print line
# Calculate deformation gradient F1
F1 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F1[i][j] = H1[i][j]
F1[0][0] += 1.
F1[1][1] += 1.
F1[2][2] += 1.
```

```
print('F1')
for line in F1: print line
# Calculate EGREEN1 = 1/2 (F1T*F1-1)
F1T = matrixTranspose(F1)
EG1 = matmul(F1T,F1)
EG1[0][0] -= 1.
EG1[1][1] -= 1.
EG1[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG1[i][j] *= 0.5
print('EG1')
for line in EG1: print line
# ============================================================
execfile('0_RVE.py')
# ============================================================
# 0_A.py  PART2
# ============================================================
# STEP 1 Large Deformation
# ============================================================
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step1.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
if H0[0][0] != '-':
    f.write("*Boundary \n")
    f.write("K1, 1, 1, %.9f \n" %(float(H0[0][0])))
if H0[0][1] != '-':
    f.write("*Boundary \n")
    f.write("K1, 2, 2, %.9f \n" %(float(H0[0][1])))
if H0[0][2] != '-':
    f.write("*Boundary \n")
    f.write("K1, 3, 3, %.9f \n" %(float(H0[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
if H0[1][0] != '-':
    f.write("*Boundary \n")
    f.write("K2, 1, 1, %.9f \n" %(float(H0[1][0])))
if H0[1][1] != '-':
    f.write("*Boundary \n")
    f.write("K2, 2, 2, %.9f \n" %(float(H0[1][1])))
if H0[1][2] != '-':
    f.write("*Boundary \n")
    f.write("K2, 3, 3, %.9f \n" %(float(H0[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
if H0[2][0] != '-':
    f.write("*Boundary \n")
    f.write("K3, 1, 1, %.9f \n" %(float(H0[2][0])))
if H0[2][1] != '-':
    f.write("*Boundary \n")
    f.write("K3, 2, 2, %.9f \n" %(float(H0[2][1])))
if H0[2][2] != '-':
    f.write("*Boundary \n")
    f.write("K3, 3, 3, %.9f \n" %(float(H0[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
```

```
# Output
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==========================================================================
# STEP 2  Unload - prescribe H1
# ==========================================================================
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step2.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary conditions
f.write('** \n')
f.write('** BOUNDARY CONDITIONS\n')
f.write('** \n')
f.write('*Boundary, op=NEW\n')
f.write("K1, 1, 1, %.9f \n" %(float(H1[0][0])))
f.write("K1, 2, 2, %.9f \n" %(float(H1[0][1])))
f.write("K1, 3, 3, %.9f \n" %(float(H1[0][2])))
f.write("K2, 1, 1, %.9f \n" %(float(H1[1][0])))
f.write("K2, 2, 2, %.9f \n" %(float(H1[1][1])))
f.write("K2, 3, 3, %.9f \n" %(float(H1[1][2])))
f.write("K3, 1, 1, %.9f \n" %(float(H1[2][0])))
f.write("K3, 2, 2, %.9f \n" %(float(H1[2][1])))
f.write("K3, 3, 3, %.9f \n" %(float(H1[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')

f.close()
# ==========================================================================
```

```
#      Run Job
# ==========================================================================
print '=============================================================';
print 'Run job'
print '=============================================================';
if "posix" in os.name:
    execfile('0_job.py')
# Wait for lck file, then wait for lck file is deleted -> job finished
t0 = time.time();
while not os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
while os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
# Check if calculation was successful, else STOP
time.sleep(5)
f = open(inpname+'.sta','r')
sta = []
for line in f:
    sta.append(str(line.rstrip()))
f.close()
if sta[len(sta)-1] == ' THE ANALYSIS HAS COMPLETED SUCCESSFULLY':
    print sta[len(sta)-1]
else:
    print sta[len(sta)-1]
    f = open('0_Auswertung_ERROR_in_'+inpname+'.txt','w')
    f.close()
    time.sleep(5)
    sys.exit("print sta[len(sta)-1]")
# ==========================================================================
#    Open ODB file, read RF1,RF2,RF3 and K1, K2, K3
# ==========================================================================
# Generate path
pfad = inpname
# Search for node number K1NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K1':
    read = str(f.readline())
read = str(f.readline())
K1NR = int(read[0:len(read)-3])
f.close()
# Search for node number K2NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K2':
    read = str(f.readline())
read = str(f.readline())
K2NR = int(read[0:len(read)-3])
f.close()
# Search for node number K2NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K3':
    read = str(f.readline())
read = str(f.readline())
K3NR = int(read[0:len(read)-3])
f.close()

o1 = session.openOdb(
    name=inpname+'.odb')
session.viewports['Viewport: 1'].setValues(displayedObject=o1)
odb = session.odbs[inpname+'.odb']
session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('RF',
    NODAL, ((COMPONENT, 'RF1'), (COMPONENT, 'RF2'), (COMPONENT, 'RF3'), )),
```

```
    ), nodeSets=('K1', 'K2', 'K3', ))
session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('U',
    NODAL, ((COMPONENT, 'U1'), (COMPONENT, 'U2'), (COMPONENT, 'U3'), )), ),
    nodeSets=('K1', 'K2', 'K3', ))
increm = len(session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data) - 1

x0 = session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K1NR]
x1 = session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K1NR]
x2 = session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K1NR]
x3 = session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K2NR]
x4 = session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K2NR]
x5 = session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K2NR]
x6 = session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K3NR]
x7 = session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K3NR]
x8 = session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K3NR]
session.writeXYReport(fileName='K0_ANLAUFRECHNUNG.rpt', appendMode=OFF, xyData=(x0,
     x1,
    x2, x3, x4, x5, x6, x7, x8))

# Generate matrix H
H1 = nullmatrix(3,3)
H1[0][0] = x0[increm][1]
H1[0][1] = x1[increm][1]
H1[0][2] = x2[increm][1]
H1[1][0] = x3[increm][1]
H1[1][1] = x4[increm][1]
H1[1][2] = x5[increm][1]
H1[2][0] = x6[increm][1]
H1[2][1] = x7[increm][1]
H1[2][2] = x8[increm][1]

del session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['U:U1 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['U:U2 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['U:U3 PI: FIBBI-1 N: %d' %K3NR]

# Generate matrix T1PK
T1PK = nullmatrix(3,3)
T1PK[0][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK[0][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK[0][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK[1][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK[1][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK[1][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK[2][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK[2][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK[2][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0

del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR]
```

```
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR]

odb.close()

# Calculate T2PK
# F1 = H1 + 1
F1 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F1[i][j] = H1[i][j]

F1[0][0] += 1.
F1[1][1] += 1.
F1[2][2] += 1.
F1i = invert3(F1)
T2PK = matmul(F1i,T1PK)

print 'H1 end'
for line in H1: print line
print 'T1PK end'
for line in T1PK: print line
print 'T2PK end'
for line in T2PK: print line

#ausT.write('T2PK\n')
ausT = open('K0_ANLAUFRECHNUNG_T1.txt','w')
for i in range(3):
    for j in range(3):
        ausT.write('%e' %(T2PK[i][j]))
        if j != 2: ausT.write(',')
    ausT.write('\n')
ausT.close()

if 'abq' in os.path.abspath("."):
    ausPINTERN = open('K0_ANLAUFRECHNUNG_PINTERN.txt','w')
    ausPINTERN.write('1.,0.,0.\n')
    ausPINTERN.write('0.,1.,0.\n')
    ausPINTERN.write('0.,0.,1.\n')
    ausPINTERN.close()
    print 'FERTIG'
else:
    print 'Read Pintern'
    # Wait until log file is finished
    time.sleep(5)
    f = open(inpname+'.log','r')
    pintern = []
    for line in f:
        pintern.append(str(line.rstrip()))
    f.close()
    PI = nullmatrix(3,3)
    PI[0][0] = pintern[len(pintern)-20]
    PI[0][1] = pintern[len(pintern)-19]
    PI[0][2] = pintern[len(pintern)-18]
    PI[1][0] = pintern[len(pintern)-17]
    PI[1][1] = pintern[len(pintern)-16]
    PI[1][2] = pintern[len(pintern)-15]
    PI[2][0] = pintern[len(pintern)-14]
    PI[2][1] = pintern[len(pintern)-13]
    PI[2][2] = pintern[len(pintern)-12]
    for i in range(3):
        for j in range(3):
            PI[i][j] = float(PI[i][j])
```

```
    #ausT.write('T2PK\n')
    ausPINTERN = open('K0_ANLAUFRECHNUNG_PINTERN.txt','w')
    for i in range(3):
        for j in range(3):
            ausPINTERN.write('%e' %(PI[i][j]))
            if j != 2: ausPINTERN.write(',')
        ausPINTERN.write('\n')
    ausPINTERN.close()
print 'FINISHED'
```

B.3 Elastic test strain "0_1.py"

```
# ======================================================================
# Elastic test strain calculation 1 of 6
# ======================================================================
# ======================================================================
# Start with: nohup /opt/abaqus/Commands/abaqus cae noGUI=*.py &
# ======================================================================
from abaqus import *
from abaqusConstants import *
import __main__
import sys
import string
import fileinput
import shutil
import os
import subprocess
import time
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import odbAccess
import random
import csv
import resource
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
```

```
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
# ==========================================================
execfile('0_subroutines.py')
# ==========================================================
# 0_1.py PART1
# ==========================================================
extname = '1'
# Define delta value (small elastic test strain)
delta = 1.e-6
matheE = open('K0_matheEGREEN_einzeln.txt','w')
matheT = open('K0_matheT2PK_einzeln.txt','w')
# Read H0 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H0.txt','r')
H0 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    data[2] = data[2].replace('\n','')
    H0.append(data)
f.close()
print('H0')
for line in H0: print line
# Read H1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H1.txt','r')
H1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    H1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        H1[i][j] = float(H1[i][j])
print('H1')
for line in H1: print line
# Read T2PK1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_T1.txt','r')
T2PK1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    T2PK1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        T2PK1[i][j] = float(T2PK1[i][j])
print('T2PK1')
for line in T2PK1: print line
# Calculate F1
F1 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F1[i][j] = H1[i][j]
F1[0][0] += 1.
F1[1][1] += 1.
F1[2][2] += 1.
print('F1')
for line in F1: print line
# Calculate EGREEN1 = 1/2 (F1T*F1-1)
F1T = matrixTranspose(F1)
EG1 = matmul(F1T,F1)
EG1[0][0] -= 1.
```

```
EG1[1][1] -= 1.
EG1[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG1[i][j] *= 0.5
print('EG1')
for line in EG1: print line
# H2 = H1 + delta
H2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        H2[i][j] = H1[i][j]
H2[0][0] = H1[0][0] + delta
print('H2')
for line in H2: print line
# ============================================================
execfile('0_RVE.py')
# ============================================================
# 0_1.py  PART2
# ============================================================
# STEP 1 Large Deformation
# ============================================================
print 'STEP-1'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step1.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
if H0[0][0] != '-':
    f.write("*Boundary \n")
    f.write("K1, 1, 1, %.9f \n" %(float(H0[0][0])))
if H0[0][1] != '-':
    f.write("*Boundary \n")
    f.write("K1, 2, 2, %.9f \n" %(float(H0[0][1])))
if H0[0][2] != '-':
    f.write("*Boundary \n")
    f.write("K1, 3, 3, %.9f \n" %(float(H0[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
if H0[1][0] != '-':
    f.write("*Boundary \n")
    f.write("K2, 1, 1, %.9f \n" %(float(H0[1][0])))
if H0[1][1] != '-':
    f.write("*Boundary \n")
    f.write("K2, 2, 2, %.9f \n" %(float(H0[1][1])))
if H0[1][2] != '-':
    f.write("*Boundary \n")
    f.write("K2, 3, 3, %.9f \n" %(float(H0[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
if H0[2][0] != '-':
    f.write("*Boundary \n")
    f.write("K3, 1, 1, %.9f \n" %(float(H0[2][0])))
if H0[2][1] != '-':
    f.write("*Boundary \n")
    f.write("K3, 2, 2, %.9f \n" %(float(H0[2][1])))
if H0[2][2] != '-':
    f.write("*Boundary \n")
    f.write("K3, 3, 3, %.9f \n" %(float(H0[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
```

```
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
# Output
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==============================================================
# STEP 2  Unload - prescribe H1
# ==============================================================
print 'STEP-2'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step2.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write('** \n')
f.write('** BOUNDARY CONDITIONS\n')
f.write('** \n')
f.write('*Boundary, op=NEW\n')
f.write("K1, 1, 1, %.9f \n" %(float(H1[0][0])))
f.write("K1, 2, 2, %.9f \n" %(float(H1[0][1])))
f.write("K1, 3, 3, %.9f \n" %(float(H1[0][2])))
f.write("K2, 1, 1, %.9f \n" %(float(H1[1][0])))
f.write("K2, 2, 2, %.9f \n" %(float(H1[1][1])))
f.write("K2, 3, 3, %.9f \n" %(float(H1[1][2])))
f.write("K3, 1, 1, %.9f \n" %(float(H1[2][0])))
f.write("K3, 2, 2, %.9f \n" %(float(H1[2][1])))
f.write("K3, 3, 3, %.9f \n" %(float(H1[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
```

```
f.write('*End Step\n')
# ==========================================================================
# STEP 3  H2 -> Column 1
# ==========================================================================
print 'STEP-3'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step3.py','r')
for line in step:
    f.write('%s' %line)
step.close()
#Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K1, 1, 1, %.9f \n" %(float(H2[0][0])))
f.write("*Boundary \n")
f.write("K1, 2, 2, %.9f \n" %(float(H2[0][1])))
f.write("*Boundary \n")
f.write("K1, 3, 3, %.9f \n" %(float(H2[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K2, 1, 1, %.9f \n" %(float(H2[1][0])))
f.write("*Boundary \n")
f.write("K2, 2, 2, %.9f \n" %(float(H2[1][1])))
f.write("*Boundary \n")
f.write("K2, 3, 3, %.9f \n" %(float(H2[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K3, 1, 1, %.9f \n" %(float(H2[2][0])))
f.write("*Boundary \n")
f.write("K3, 2, 2, %.9f \n" %(float(H2[2][1])))
f.write("*Boundary \n")
f.write("K3, 3, 3, %.9f \n" %(float(H2[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
f.close()
# ==========================================================================
#       Run Job
# ==========================================================================
#print '=============================================================='
#print 'Job Spalte', spalte+1
#print '=============================================================='
```

```
if "posix" in os.name:
    execfile('0_job.py')
# Wait for lck file, then wait for lck file is deleted -> job finished
t0 = time.time();
while not os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
while os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
# Check if calculation was successful, else STOP
time.sleep(5)
f = open(inpname+'.sta','r')
sta = []
for line in f:
    sta.append(str(line.rstrip()))
f.close()
if sta[len(sta)-1] == ' THE ANALYSIS HAS COMPLETED SUCCESSFULLY':
    print sta[len(sta)-1]
else:
    print sta[len(sta)-1]
    f = open('0_Auswertung_ERROR_in_'+inpname+'.txt','w')
    f.close()
    time.sleep(5)
    sys.exit("print sta[len(sta)-1]")
# ================================================================
#     ODB oeffnen und RF1,RF2,RF3 an K1, K2, K3 rausschreiben
# ================================================================
# Generate path
pfad = inpname
# Search for node number K1NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K1':
    read = str(f.readline())
read = str(f.readline())
K1NR = int(read[0:len(read)-3])
f.close()
# Search for node number K2NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K2':
    read = str(f.readline())
read = str(f.readline())
K2NR = int(read[0:len(read)-3])
f.close()
# Search for node number K3NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K3':
    read = str(f.readline())
read = str(f.readline())
K3NR = int(read[0:len(read)-3])
f.close()
# ================================================================
#     Column 1 Step-3
# ================================================================
o1 = session.openOdb(
    name=inpname+'.odb')
session.viewports['Viewport: 1'].setValues(displayedObject=o1)
odbName=session.viewports[session.currentViewportName].odbDisplay.name
session.odbData[odbName].setValues(activeFrames=(('Step-3', ('0:-1', )), ))
odb = session.odbs[inpname+'.odb']
session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('RF',
    NODAL, ((COMPONENT, 'RF1'), (COMPONENT, 'RF2'), (COMPONENT, 'RF3'), )),
```

```
    ), nodeSets=('K1', 'K2', 'K3', ))
increm = len(session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data) - 1

# Generate matrix T1PK
T1PK2 = nullmatrix(3,3)
T1PK2[0][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[1][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[2][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR]
print('T1PK2')
for line in T1PK2: print line
odb.close()
# Calculate T2PK2
# F2 = H2 + 1
F2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F2[i][j] = H2[i][j]
F2[0][0] += 1.
F2[1][1] += 1.
F2[2][2] += 1.
F2i = invert3(F2)
T2PK2 = matmul(F2i,T1PK2)
# Calculate E2_green berechnen
# E2_green = 1/2 (F2T*F-1)
FT2 = matrixTranspose(F2)
EG2 = matmul(FT2,F2)
EG2[0][0] -= 1.
EG2[1][1] -= 1.
EG2[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG2[i][j] *= 0.5
# Calculate delta EG and delta T2PK
DEG = nullmatrix(3,3)
DT2PK = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        DEG[i][j]   = EG2[i][j]-EG1[i][j]
        DT2PK[i][j] = T2PK2[i][j]-T2PK1[i][j]
print('EG1')
for line in EG1: print line
print('EG2')
```

```
for line in EG2: print line
print('Delta EG')
for line in DEG: print line

print('T2PK1')
for line in T2PK1: print line
print('T2PK2')
for line in T2PK2: print line
print('Delta T2PK')
for line in DT2PK: print line
# ==========================================================
# Generate stress and strain output for Mathematica post processing
# ==========================================================
# Generate vector of symmetric Green strain tensor EG
matheE.write('%e,' %(DEG[0][0]))
matheE.write('%e,' %(DEG[1][1]))
matheE.write('%e,' %(DEG[2][2]))
matheE.write('%e,' %(DEG[0][1]))
matheE.write('%e,' %(DEG[0][2]))
matheE.write('%e,' %(DEG[1][2]))
# Generate vector of symmetric 2nd Piola-Kirchhoff stress tensor T2PK
matheT.write('%e,' %(DT2PK[0][0]))
matheT.write('%e,' %(DT2PK[1][1]))
matheT.write('%e,' %(DT2PK[2][2]))
matheT.write('%e,' %(DT2PK[0][1]))
matheT.write('%e,' %(DT2PK[0][2]))
matheT.write('%e,' %(DT2PK[1][2]))
matheE.close()
matheT.close()
print 'FINISHED'
```

B.4 Elastic test strain "0_2.py"

```
# ==========================================================
# Elastic test strain calculation 2 of 6
# ==========================================================
# ==========================================================
# Start with: nohup /opt/abaqus/Commands/abaqus cae noGUI=*.py &
# ==========================================================
from abaqus import *
from abaqusConstants import *
import __main__
import sys
import string
import fileinput
import shutil
import os
import subprocess
import time
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
```

```
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import odbAccess
import random
import csv
import resource
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
# ======================================================================
execfile('0_subroutines.py')
# ======================================================================
# 0_2.py PART1
# ======================================================================
extname = '2'
# Define delta value (small elastic test strain)
delta = 1.e-6
matheE = open('K0_matheEGREEN_einzeln.txt','w')
matheT = open('K0_matheT2PK_einzeln.txt','w')
# Read H0 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H0.txt','r')
H0 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    data[2] = data[2].replace('\n','')
    H0.append(data)
f.close()
print('H0')
for line in H0: print line
# Read H1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H1.txt','r')
H1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    H1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        H1[i][j] = float(H1[i][j])
print('H1')
for line in H1: print line
# Read T2PK1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_T1.txt','r')
T2PK1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
```

```
        T2PK1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        T2PK1[i][j] = float(T2PK1[i][j])
print('T2PK1')
for line in T2PK1: print line
# Calculate F1
F1 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F1[i][j] = H1[i][j]
F1[0][0] += 1.
F1[1][1] += 1.
F1[2][2] += 1.
print('F1')
for line in F1: print line
# Calculate EGREEN1 = 1/2 (F1T*F1-1)
F1T = matrixTranspose(F1)
EG1 = matmul(F1T,F1)
EG1[0][0] -= 1.
EG1[1][1] -= 1.
EG1[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG1[i][j] *= 0.5
print('EG1')
for line in EG1: print line
# H2 = H1 + delta
H2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        H2[i][j] = H1[i][j]
H2[1][1] = H1[1][1] + delta
print('H2')
for line in H2: print line
# ============================================================================
execfile('0_RVE.py')
# ============================================================================
# 0_2.py  PART2
# ============================================================================
# STEP 1 Large Deformation
# ============================================================================
print 'STEP-1'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step1.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
if H0[0][0] != '-':
    f.write("*Boundary \n")
    f.write("K1, 1, 1, %.9f \n" %(float(H0[0][0])))
if H0[0][1] != '-':
    f.write("*Boundary \n")
    f.write("K1, 2, 2, %.9f \n" %(float(H0[0][1])))
if H0[0][2] != '-':
    f.write("*Boundary \n")
    f.write("K1, 3, 3, %.9f \n" %(float(H0[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
if H0[1][0] != '-':
```

```
    f.write("*Boundary \n")
    f.write("K2, 1, 1, %.9f \n" %(float(H0[1][0])))
if H0[1][1] != '-':
    f.write("*Boundary \n")
    f.write("K2, 2, 2, %.9f \n" %(float(H0[1][1])))
if H0[1][2] != '-':
    f.write("*Boundary \n")
    f.write("K2, 3, 3, %.9f \n" %(float(H0[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
if H0[2][0] != '-':
    f.write("*Boundary \n")
    f.write("K3, 1, 1, %.9f \n" %(float(H0[2][0])))
if H0[2][1] != '-':
    f.write("*Boundary \n")
    f.write("K3, 2, 2, %.9f \n" %(float(H0[2][1])))
if H0[2][2] != '-':
    f.write("*Boundary \n")
    f.write("K3, 3, 3, %.9f \n" %(float(H0[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
# Output
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==========================================================================
# STEP 2  Unload - prescribe H1
# ==========================================================================
print 'STEP-2'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step2.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write('** \n')
f.write('** BOUNDARY CONDITIONS\n')
f.write('** \n')
f.write('*Boundary, op=NEW\n')
f.write("K1, 1, 1, %.9f \n" %(float(H1[0][0])))
f.write("K1, 2, 2, %.9f \n" %(float(H1[0][1])))
f.write("K1, 3, 3, %.9f \n" %(float(H1[0][2])))
f.write("K2, 1, 1, %.9f \n" %(float(H1[1][0])))
f.write("K2, 2, 2, %.9f \n" %(float(H1[1][1])))
f.write("K2, 3, 3, %.9f \n" %(float(H1[1][2])))
f.write("K3, 1, 1, %.9f \n" %(float(H1[2][0])))
f.write("K3, 2, 2, %.9f \n" %(float(H1[2][1])))
f.write("K3, 3, 3, %.9f \n" %(float(H1[2][2])))
```

```
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==============================================================
# STEP 3  H2 -> Column 1
# ==============================================================
print 'STEP-3'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step3.py','r')
for line in step:
    f.write('%s' %line)
step.close()
#Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K1, 1, 1, %.9f \n" %(float(H2[0][0])))
f.write("*Boundary \n")
f.write("K1, 2, 2, %.9f \n" %(float(H2[0][1])))
f.write("*Boundary \n")
f.write("K1, 3, 3, %.9f \n" %(float(H2[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K2, 1, 1, %.9f \n" %(float(H2[1][0])))
f.write("*Boundary \n")
f.write("K2, 2, 2, %.9f \n" %(float(H2[1][1])))
f.write("*Boundary \n")
f.write("K2, 3, 3, %.9f \n" %(float(H2[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K3, 1, 1, %.9f \n" %(float(H2[2][0])))
f.write("*Boundary \n")
f.write("K3, 2, 2, %.9f \n" %(float(H2[2][1])))
f.write("*Boundary \n")
f.write("K3, 3, 3, %.9f \n" %(float(H2[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
```

```
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
f.close()
# ==========================================================================
#     Run Job
# ==========================================================================
#print '============================================================';
#print 'Job Spalte',spalte+1
#print '============================================================';

if "posix" in os.name:
    execfile('0_job.py')
# Wait for lck file, then wait for lck file is deleted -> job finished
t0 = time.time();
while not os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
while os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
# Check if calculation was successful, else STOP
time.sleep(5)
f = open(inpname+'.sta','r')
sta = []
for line in f:
    sta.append(str(line.rstrip()))
f.close()
if sta[len(sta)-1] == ' THE ANALYSIS HAS COMPLETED SUCCESSFULLY':
    print sta[len(sta)-1]
else:
    print sta[len(sta)-1]
    f = open('0_Auswertung_ERROR_in_'+inpname+'.txt','w')
    f.close()
    time.sleep(5)
    sys.exit("print sta[len(sta)-1]")
# ==========================================================================
#     ODB oeffnen und RF1,RF2,RF3 an K1, K2, K3 rausschreiben
# ==========================================================================
# Generate path
pfad = inpname
# Search for node number K1NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K1':
    read = str(f.readline())
read = str(f.readline())
K1NR = int(read[0:len(read)-3])
f.close()
# Search for node number K2NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K2':
    read = str(f.readline())
read = str(f.readline())
```

```
K2NR = int(read[0:len(read)-3])
f.close()
# Search for node number K3NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K3':
    read = str(f.readline())
read = str(f.readline())
K3NR = int(read[0:len(read)-3])
f.close()
# ============================================================================
#       Column 1 Step-3
# ============================================================================
o1 = session.openOdb(
    name=inpname+'.odb')
session.viewports['Viewport: 1'].setValues(displayedObject=o1)
odbName=session.viewports[session.currentViewportName].odbDisplay.name
session.odbData[odbName].setValues(activeFrames=(('Step-3', ('0:-1', )), ))
odb = session.odbs[inpname+'.odb']
session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('RF',
    NODAL, ((COMPONENT, 'RF1'), (COMPONENT, 'RF2'), (COMPONENT, 'RF3'), )),
    ), nodeSets=('K1', 'K2', 'K3', ))
increm = len(session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data) - 1

# Generate matrix T1PK
T1PK2 = nullmatrix(3,3)
T1PK2[0][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[1][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[2][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR]
print('T1PK2')
for line in T1PK2: print line
odb.close()
# Calculate T2PK2
# F2 = H2 + 1
F2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F2[i][j] = H2[i][j]
F2[0][0] += 1.
F2[1][1] += 1.
F2[2][2] += 1.
F2i = invert3(F2)
```

```
T2PK2 = matmul(F2i,T1PK2)
# Calculate E2_green berechnen
# E2_green = 1/2 (F2T*F-1)
FT2 = matrixTranspose(F2)
EG2 = matmul(FT2,F2)
EG2[0][0] -= 1.
EG2[1][1] -= 1.
EG2[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG2[i][j] *= 0.5
# Calculate delta EG and delta T2PK
DEG = nullmatrix(3,3)
DT2PK = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        DEG[i][j]   = EG2[i][j]-EG1[i][j]
        DT2PK[i][j] = T2PK2[i][j]-T2PK1[i][j]
print('EG1')
for line in EG1: print line
print('EG2')
for line in EG2: print line
print('Delta EG')
for line in DEG: print line

print('T2PK1')
for line in T2PK1: print line
print('T2PK2')
for line in T2PK2: print line
print('Delta T2PK')
for line in DT2PK: print line
# ================================================================
# Generate stress and strain output for Mathematica post processing
# ================================================================
# Generate vector of symmetric Green strain tensor EG
matheE.write('%e,' %(DEG[0][0]))
matheE.write('%e,' %(DEG[1][1]))
matheE.write('%e,' %(DEG[2][2]))
matheE.write('%e,' %(DEG[0][1]))
matheE.write('%e,' %(DEG[0][2]))
matheE.write('%e,' %(DEG[1][2]))
# Generate vector of symmetric 2nd Piola-Kirchhoff stress tensor T2PK
matheT.write('%e,' %(DT2PK[0][0]))
matheT.write('%e,' %(DT2PK[1][1]))
matheT.write('%e,' %(DT2PK[2][2]))
matheT.write('%e,' %(DT2PK[0][1]))
matheT.write('%e,' %(DT2PK[0][2]))
matheT.write('%e,' %(DT2PK[1][2]))
matheE.close()
matheT.close()
print 'FINISHED'
```

B.5 Elastic test strain "0_3.py"

```
# ================================================================
# Elastic test strain calculation 3 of 6
# ================================================================
# ================================================================
# Start with: nohup /opt/abaqus/Commands/abaqus cae noGUI=*.py &
```

```
# ======================================================================
from abaqus import *
from abaqusConstants import *
import __main__
import sys
import string
import fileinput
import shutil
import os
import subprocess
import time
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import odbAccess
import random
import csv
import resource
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
# ======================================================================
execfile('0_subroutines.py')
# ======================================================================
# 0_3.py PART1
# ======================================================================
extname = '3'
# Define delta value (small elastic test strain)
delta = 1.e-6
matheE = open('K0_matheEGREEN_einzeln.txt','w')
matheT = open('K0_matheT2PK_einzeln.txt','w')
# Read H0 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H0.txt','r')
H0 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    data[2] = data[2].replace('\n','')
    H0.append(data)
f.close()
```

```
print('H0')
for line in H0: print line
# Read H1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H1.txt','r')
H1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    H1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        H1[i][j] = float(H1[i][j])
print('H1')
for line in H1: print line
# Read T2PK1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_T1.txt','r')
T2PK1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    T2PK1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        T2PK1[i][j] = float(T2PK1[i][j])
print('T2PK1')
for line in T2PK1: print line
# Calculate F1
F1 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F1[i][j] = H1[i][j]
F1[0][0] += 1.
F1[1][1] += 1.
F1[2][2] += 1.
print('F1')
for line in F1: print line
# Calculate EGREEN1 = 1/2 (F1T*F1-1)
F1T = matrixTranspose(F1)
EG1 = matmul(F1T,F1)
EG1[0][0] -= 1.
EG1[1][1] -= 1.
EG1[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG1[i][j] *= 0.5
print('EG1')
for line in EG1: print line
# H2 = H1 + delta
H2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        H2[i][j] = H1[i][j]
H2[2][2] = H1[2][2] + delta
print('H2')
for line in H2: print line
# ================================================================
execfile('0_RVE.py')
# ================================================================
# 0_3.py  PART2
# ================================================================
# STEP 1 Large Deformation
# ================================================================
print 'STEP-1'
f.write('** ----------------------------------------------------------\n')
```

```
# Read step information
step = open('0_step1.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
if H0[0][0] != '-':
    f.write("*Boundary \n")
    f.write("K1, 1, 1, %.9f \n" %(float(H0[0][0])))
if H0[0][1] != '-':
    f.write("*Boundary \n")
    f.write("K1, 2, 2, %.9f \n" %(float(H0[0][1])))
if H0[0][2] != '-':
    f.write("*Boundary \n")
    f.write("K1, 3, 3, %.9f \n" %(float(H0[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
if H0[1][0] != '-':
    f.write("*Boundary \n")
    f.write("K2, 1, 1, %.9f \n" %(float(H0[1][0])))
if H0[1][1] != '-':
    f.write("*Boundary \n")
    f.write("K2, 2, 2, %.9f \n" %(float(H0[1][1])))
if H0[1][2] != '-':
    f.write("*Boundary \n")
    f.write("K2, 3, 3, %.9f \n" %(float(H0[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
if H0[2][0] != '-':
    f.write("*Boundary \n")
    f.write("K3, 1, 1, %.9f \n" %(float(H0[2][0])))
if H0[2][1] != '-':
    f.write("*Boundary \n")
    f.write("K3, 2, 2, %.9f \n" %(float(H0[2][1])))
if H0[2][2] != '-':
    f.write("*Boundary \n")
    f.write("K3, 3, 3, %.9f \n" %(float(H0[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
# Output
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==========================================================
# STEP 2  Unload - prescribe H1
# ==========================================================
```

```
print 'STEP-2'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step2.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write('** \n')
f.write('** BOUNDARY CONDITIONS\n')
f.write('** \n')
f.write('*Boundary, op=NEW\n')
f.write("K1, 1, 1, %.9f \n" %(float(H1[0][0])))
f.write("K1, 2, 2, %.9f \n" %(float(H1[0][1])))
f.write("K1, 3, 3, %.9f \n" %(float(H1[0][2])))
f.write("K2, 1, 1, %.9f \n" %(float(H1[1][0])))
f.write("K2, 2, 2, %.9f \n" %(float(H1[1][1])))
f.write("K2, 3, 3, %.9f \n" %(float(H1[1][2])))
f.write("K3, 1, 1, %.9f \n" %(float(H1[2][0])))
f.write("K3, 2, 2, %.9f \n" %(float(H1[2][1])))
f.write("K3, 3, 3, %.9f \n" %(float(H1[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==============================================================
# STEP 3  H2 -> Column 1
# ==============================================================
print 'STEP-3'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step3.py','r')
for line in step:
    f.write('%s' %line)
step.close()
#Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K1, 1, 1, %.9f \n" %(float(H2[0][0])))
f.write("*Boundary \n")
f.write("K1, 2, 2, %.9f \n" %(float(H2[0][1])))
f.write("*Boundary \n")
f.write("K1, 3, 3, %.9f \n" %(float(H2[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
```

```
f.write("K2, 1, 1, %.9f \n" %(float(H2[1][0])))
f.write("*Boundary \n")
f.write("K2, 2, 2, %.9f \n" %(float(H2[1][1])))
f.write("*Boundary \n")
f.write("K2, 3, 3, %.9f \n" %(float(H2[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K3, 1, 1, %.9f \n" %(float(H2[2][0])))
f.write("*Boundary \n")
f.write("K3, 2, 2, %.9f \n" %(float(H2[2][1])))
f.write("*Boundary \n")
f.write("K3, 3, 3, %.9f \n" %(float(H2[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
f.close()
# ==========================================================================
#      Run Job
# ==========================================================================
#print '============================================================';
#print 'Job Spalte',spalte+1
#print '============================================================';

if "posix" in os.name:
    execfile('0_job.py')
# Wait for lck file, then wait for lck file is deleted -> job finished
t0 = time.time();
while not os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
while os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
# Check if calculation was successful, else STOP
time.sleep(5)
f = open(inpname+'.sta','r')
sta = []
for line in f:
    sta.append(str(line.rstrip()))
f.close()
if sta[len(sta)-1] == ' THE ANALYSIS HAS COMPLETED SUCCESSFULLY':
    print sta[len(sta)-1]
else:
    print sta[len(sta)-1]
    f = open('0_Auswertung_ERROR_in_'+inpname+'.txt','w')
    f.close()
```

```
    time.sleep(5)
    sys.exit("print sta[len(sta)-1]")
# ================================================================
#     ODB oeffnen und RF1,RF2,RF3 an K1, K2, K3 rausschreiben
# ================================================================
# Generate path
pfad = inpname
# Search for node number K1NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K1':
    read = str(f.readline())
read = str(f.readline())
K1NR = int(read[0:len(read)-3])
f.close()
# Search for node number K2NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K2':
    read = str(f.readline())
read = str(f.readline())
K2NR = int(read[0:len(read)-3])
f.close()
# Search for node number K3NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K3':
    read = str(f.readline())
read = str(f.readline())
K3NR = int(read[0:len(read)-3])
f.close()
# ================================================================
#     Column 1 Step-3
# ================================================================
o1 = session.openOdb(
    name=inpname+'.odb')
session.viewports['Viewport: 1'].setValues(displayedObject=o1)
odbName=session.viewports[session.currentViewportName].odbDisplay.name
session.odbData[odbName].setValues(activeFrames=(('Step-3', ('0:-1', )), ))
odb = session.odbs[inpname+'.odb']
session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('RF',
    NODAL, ((COMPONENT, 'RF1'), (COMPONENT, 'RF2'), (COMPONENT, 'RF3'), )),
    ), nodeSets=('K1', 'K2', 'K3', ))
increm = len(session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data) - 1

# Generate matrix T1PK
T1PK2 = nullmatrix(3,3)
T1PK2[0][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[1][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[2][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR]
```

```
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR]
print('T1PK2')
for line in T1PK2: print line
odb.close()
# Calculate T2PK2
# F2 = H2 + 1
F2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F2[i][j] = H2[i][j]
F2[0][0] += 1.
F2[1][1] += 1.
F2[2][2] += 1.
F2i = invert3(F2)
T2PK2 = matmul(F2i,T1PK2)
# Calculate E2_green berechnen
# E2_green = 1/2 (F2T*F-1)
FT2 = matrixTranspose(F2)
EG2 = matmul(FT2,F2)
EG2[0][0] -= 1.
EG2[1][1] -= 1.
EG2[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG2[i][j] *= 0.5
# Calculate delta EG and delta T2PK
DEG = nullmatrix(3,3)
DT2PK = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        DEG[i][j]   = EG2[i][j]-EG1[i][j]
        DT2PK[i][j] = T2PK2[i][j]-T2PK1[i][j]
print('EG1')
for line in EG1: print line
print('EG2')
for line in EG2: print line
print('Delta EG')
for line in DEG: print line

print('T2PK1')
for line in T2PK1: print line
print('T2PK2')
for line in T2PK2: print line
print('Delta T2PK')
for line in DT2PK: print line
# ==========================================================
# Generate stress and strain output for Mathematica post processing
# ==========================================================
# Generate vector of symmetric Green strain tensor EG
matheE.write('%e,' %(DEG[0][0]))
matheE.write('%e,' %(DEG[1][1]))
matheE.write('%e,' %(DEG[2][2]))
matheE.write('%e,' %(DEG[0][1]))
matheE.write('%e,' %(DEG[0][2]))
matheE.write('%e,' %(DEG[1][2]))
# Generate vector of symmetric 2nd Piola-Kirchhoff stress tensor T2PK
matheT.write('%e,' %(DT2PK[0][0]))
matheT.write('%e,' %(DT2PK[1][1]))
matheT.write('%e,' %(DT2PK[2][2]))
```

```
matheT.write('%e,' %(DT2PK[0][1]))
matheT.write('%e,' %(DT2PK[0][2]))
matheT.write('%e,' %(DT2PK[1][2]))
matheE.close()
matheT.close()
print 'FINISHED'
```

B.6 Elastic test strain "0_4.py"

```
# ======================================================================
# Elastic test strain calculation 4 of 6
# ======================================================================
# ======================================================================
# Start with: nohup /opt/abaqus/Commands/abaqus cae noGUI=*.py &
# ======================================================================
from abaqus import *
from abaqusConstants import *
import __main__
import sys
import string
import fileinput
import shutil
import os
import subprocess
import time
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import odbAccess
import random
import csv
import resource
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
```

```
import displayGroupOdbToolset as dgo
import connectorBehavior
# ======================================================================
execfile('0_subroutines.py')
# ======================================================================
# 0_4.py PART1
# ======================================================================
extname = '4'
# Define delta value (small elastic test strain)
delta = 1.e-6
matheE = open('K0_matheEGREEN_einzeln.txt','w')
matheT = open('K0_matheT2PK_einzeln.txt','w')
# Read H0 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H0.txt','r')
H0 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    data[2] = data[2].replace('\n','')
    H0.append(data)
f.close()
print('H0')
for line in H0: print line
# Read H1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H1.txt','r')
H1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    H1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        H1[i][j] = float(H1[i][j])
print('H1')
for line in H1: print line
# Read T2PK1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_T1.txt','r')
T2PK1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    T2PK1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        T2PK1[i][j] = float(T2PK1[i][j])
print('T2PK1')
for line in T2PK1: print line
# Calculate F1
F1 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F1[i][j] = H1[i][j]
F1[0][0] += 1.
F1[1][1] += 1.
F1[2][2] += 1.
print('F1')
for line in F1: print line
# Calculate EGREEN1 = 1/2 (F1T*F1-1)
F1T = matrixTranspose(F1)
EG1 = matmul(F1T,F1)
EG1[0][0] -= 1.
EG1[1][1] -= 1.
EG1[2][2] -= 1.
for i in range(3):
```

```
    for j in range(3):
        EG1[i][j] *= 0.5
print('EG1')
for line in EG1: print line
# H2 = H1 + delta
H2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        H2[i][j] = H1[i][j]
H2[0][1] = H1[0][1] + delta
H2[1][0] = H1[1][0] + delta
print('H2')
for line in H2: print line
# ============================================================
execfile('0_RVE.py')
# ============================================================
# 0_4.py  PART2
# ============================================================
# STEP 1 Large Deformation
# ============================================================
print 'STEP-1'
f.write('** ------------------------------------------------------------\n')
# Read step information
step = open('0_step1.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
if H0[0][0] != '-':
    f.write("*Boundary \n")
    f.write("K1, 1, 1, %.9f \n" %(float(H0[0][0])))
if H0[0][1] != '-':
    f.write("*Boundary \n")
    f.write("K1, 2, 2, %.9f \n" %(float(H0[0][1])))
if H0[0][2] != '-':
    f.write("*Boundary \n")
    f.write("K1, 3, 3, %.9f \n" %(float(H0[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
if H0[1][0] != '-':
    f.write("*Boundary \n")
    f.write("K2, 1, 1, %.9f \n" %(float(H0[1][0])))
if H0[1][1] != '-':
    f.write("*Boundary \n")
    f.write("K2, 2, 2, %.9f \n" %(float(H0[1][1])))
if H0[1][2] != '-':
    f.write("*Boundary \n")
    f.write("K2, 3, 3, %.9f \n" %(float(H0[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
if H0[2][0] != '-':
    f.write("*Boundary \n")
    f.write("K3, 1, 1, %.9f \n" %(float(H0[2][0])))
if H0[2][1] != '-':
    f.write("*Boundary \n")
    f.write("K3, 2, 2, %.9f \n" %(float(H0[2][1])))
if H0[2][2] != '-':
    f.write("*Boundary \n")
    f.write("K3, 3, 3, %.9f \n" %(float(H0[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
```

```
    f.write("Ursprung, 3, 3, 0 \n" )
# Output
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==========================================================
# STEP 2  Unload - prescribe H1
# ==========================================================
print 'STEP-2'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step2.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write('** \n')
f.write('** BOUNDARY CONDITIONS\n')
f.write('** \n')
f.write('*Boundary, op=NEW\n')
f.write("K1, 1, 1, %.9f \n" %(float(H1[0][0])))
f.write("K1, 2, 2, %.9f \n" %(float(H1[0][1])))
f.write("K1, 3, 3, %.9f \n" %(float(H1[0][2])))
f.write("K2, 1, 1, %.9f \n" %(float(H1[1][0])))
f.write("K2, 2, 2, %.9f \n" %(float(H1[1][1])))
f.write("K2, 3, 3, %.9f \n" %(float(H1[1][2])))
f.write("K3, 1, 1, %.9f \n" %(float(H1[2][0])))
f.write("K3, 2, 2, %.9f \n" %(float(H1[2][1])))
f.write("K3, 3, 3, %.9f \n" %(float(H1[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==========================================================
```

```
# STEP 3  H2 -> Column 1
# ======================================================================
print 'STEP-3'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step3.py','r')
for line in step:
    f.write('%s' %line)
step.close()
#Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K1, 1, 1, %.9f \n" %(float(H2[0][0])))
f.write("*Boundary \n")
f.write("K1, 2, 2, %.9f \n" %(float(H2[0][1])))
f.write("*Boundary \n")
f.write("K1, 3, 3, %.9f \n" %(float(H2[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K2, 1, 1, %.9f \n" %(float(H2[1][0])))
f.write("*Boundary \n")
f.write("K2, 2, 2, %.9f \n" %(float(H2[1][1])))
f.write("*Boundary \n")
f.write("K2, 3, 3, %.9f \n" %(float(H2[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K3, 1, 1, %.9f \n" %(float(H2[2][0])))
f.write("*Boundary \n")
f.write("K3, 2, 2, %.9f \n" %(float(H2[2][1])))
f.write("*Boundary \n")
f.write("K3, 3, 3, %.9f \n" %(float(H2[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
f.close()
# ======================================================================
#      Run Job
# ======================================================================
#print '=============================================================='
#print 'Job Spalte', spalte+1
#print '=============================================================='

if "posix" in os.name:
```

```
    execfile('0_job.py')
# Wait for lck file, then wait for lck file is deleted -> job finished
t0 = time.time();
while not os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
while os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
# Check if calculation was successful, else STOP
time.sleep(5)
f = open(inpname+'.sta','r')
sta = []
for line in f:
    sta.append(str(line.rstrip()))
f.close()
if sta[len(sta)-1] == ' THE ANALYSIS HAS COMPLETED SUCCESSFULLY':
    print sta[len(sta)-1]
else:
    print sta[len(sta)-1]
    f = open('0_Auswertung_ERROR_in_'+inpname+'.txt','w')
    f.close()
    time.sleep(5)
    sys.exit("print sta[len(sta)-1]")
# ==========================================================================
#     ODB oeffnen und RF1,RF2,RF3 an K1, K2, K3 rausschreiben
# ==========================================================================
# Generate path
pfad = inpname
# Search for node number K1NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K1':
    read = str(f.readline())
read = str(f.readline())
K1NR = int(read[0:len(read)-3])
f.close()
# Search for node number K2NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K2':
    read = str(f.readline())
read = str(f.readline())
K2NR = int(read[0:len(read)-3])
f.close()
# Search for node number K3NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K3':
    read = str(f.readline())
read = str(f.readline())
K3NR = int(read[0:len(read)-3])
f.close()
# ==========================================================================
#     Column 1 Step-3
# ==========================================================================
o1 = session.openOdb(
    name=inpname+'.odb')
session.viewports['Viewport: 1'].setValues(displayedObject=o1)
odbName=session.viewports[session.currentViewportName].odbDisplay.name
session.odbData[odbName].setValues(activeFrames=(('Step-3', ('0:-1', )), ))
odb = session.odbs[inpname+'.odb']
session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('RF',
    NODAL, ((COMPONENT, 'RF1'), (COMPONENT, 'RF2'), (COMPONENT, 'RF3'), )),
    ), nodeSets=('K1', 'K2', 'K3', ))
increm = len(session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data) - 1
```

```
# Generate matrix T1PK
T1PK2 = nullmatrix(3,3)
T1PK2[0][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[1][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[2][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR]
print('T1PK2')
for line in T1PK2: print line
odb.close()
# Calculate T2PK2
# F2 = H2 + 1
F2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F2[i][j] = H2[i][j]
F2[0][0] += 1.
F2[1][1] += 1.
F2[2][2] += 1.
F2i = invert3(F2)
T2PK2 = matmul(F2i,T1PK2)
# Calculate E2_green berechnen
# E2_green = 1/2 (F2T*F-1)
FT2 = matrixTranspose(F2)
EG2 = matmul(FT2,F2)
EG2[0][0] -= 1.
EG2[1][1] -= 1.
EG2[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG2[i][j] *= 0.5
# Calculate delta EG and delta T2PK
DEG = nullmatrix(3,3)
DT2PK = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        DEG[i][j]   = EG2[i][j]-EG1[i][j]
        DT2PK[i][j] = T2PK2[i][j]-T2PK1[i][j]
print('EG1')
for line in EG1: print line
print('EG2')
for line in EG2: print line
print('Delta EG')
```

```
for line in DEG: print line

print('T2PK1')
for line in T2PK1: print line
print('T2PK2')
for line in T2PK2: print line
print('Delta T2PK')
for line in DT2PK: print line
# ==========================================================
# Generate stress and strain output for Mathematica post processing
# ==========================================================
# Generate vector of symmetric Green strain tensor EG
matheE.write('%e,' %(DEG[0][0]))
matheE.write('%e,' %(DEG[1][1]))
matheE.write('%e,' %(DEG[2][2]))
matheE.write('%e,' %(DEG[0][1]))
matheE.write('%e,' %(DEG[0][2]))
matheE.write('%e,' %(DEG[1][2]))
# Generate vector of symmetric 2nd Piola-Kirchhoff stress tensor T2PK
matheT.write('%e,' %(DT2PK[0][0]))
matheT.write('%e,' %(DT2PK[1][1]))
matheT.write('%e,' %(DT2PK[2][2]))
matheT.write('%e,' %(DT2PK[0][1]))
matheT.write('%e,' %(DT2PK[0][2]))
matheT.write('%e,' %(DT2PK[1][2]))
matheE.close()
matheT.close()
print 'FINISHED'
```

B.7 Elastic test strain "0_5.py"

```
# ==========================================================
# Elastic test strain calculation 5 of 6
# ==========================================================
# ==========================================================
# Start with: nohup /opt/abaqus/Commands/abaqus cae noGUI=*.py &
# ==========================================================
from abaqus import *
from abaqusConstants import *
import __main__
import sys
import string
import fileinput
import shutil
import os
import subprocess
import time
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
```

```
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import odbAccess
import random
import csv
import resource
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
#==========================================================================
execfile('0_subroutines.py')
#==========================================================================
# 0_5.py PART1
#==========================================================================
extname = '5'
# Define delta value (small elastic test strain)
delta = 1.e-6
matheE = open('K0_matheEGREEN_einzeln.txt','w')
matheT = open('K0_matheT2PK_einzeln.txt','w')
# Read H0 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H0.txt','r')
H0 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    data[2] = data[2].replace('\n','')
    H0.append(data)
f.close()
print('H0')
for line in H0: print line
# Read H1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H1.txt','r')
H1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    H1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        H1[i][j] = float(H1[i][j])
print('H1')
for line in H1: print line
# Read T2PK1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_T1.txt','r')
T2PK1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    T2PK1.append(data)
f.close()
```

```
for i in range(3):
    for j in range(3):
        T2PK1[i][j] = float(T2PK1[i][j])
print('T2PK1')
for line in T2PK1: print line
# Calculate F1
F1 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F1[i][j] = H1[i][j]
F1[0][0] += 1.
F1[1][1] += 1.
F1[2][2] += 1.
print('F1')
for line in F1: print line
# Calculate EGREEN1 = 1/2 (F1T*F1-1)
F1T = matrixTranspose(F1)
EG1 = matmul(F1T,F1)
EG1[0][0] -= 1.
EG1[1][1] -= 1.
EG1[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG1[i][j] *= 0.5
print('EG1')
for line in EG1: print line
# H2 = H1 + delta
H2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        H2[i][j] = H1[i][j]
H2[0][2] = H1[0][2] + delta
H2[2][0] = H1[2][0] + delta
print('H2')
for line in H2: print line
# ================================================================
execfile('0_RVE.py')
# ================================================================
# 0_5.py  PART2
# ================================================================
# STEP 1 Large Deformation
# ================================================================
print 'STEP-1'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step1.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
if H0[0][0] != '-':
    f.write("*Boundary \n")
    f.write("K1, 1, 1, %.9f \n" %(float(H0[0][0])))
if H0[0][1] != '-':
    f.write("*Boundary \n")
    f.write("K1, 2, 2, %.9f \n" %(float(H0[0][1])))
if H0[0][2] != '-':
    f.write("*Boundary \n")
    f.write("K1, 3, 3, %.9f \n" %(float(H0[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
if H0[1][0] != '-':
    f.write("*Boundary \n")
```

```
    f.write("K2, 1, 1, %.9f \n" %(float(H0[1][0])))
if H0[1][1] != '-':
    f.write("*Boundary \n")
    f.write("K2, 2, 2, %.9f \n" %(float(H0[1][1])))
if H0[1][2] != '-':
    f.write("*Boundary \n")
    f.write("K2, 3, 3, %.9f \n" %(float(H0[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
if H0[2][0] != '-':
    f.write("*Boundary \n")
    f.write("K3, 1, 1, %.9f \n" %(float(H0[2][0])))
if H0[2][1] != '-':
    f.write("*Boundary \n")
    f.write("K3, 2, 2, %.9f \n" %(float(H0[2][1])))
if H0[2][2] != '-':
    f.write("*Boundary \n")
    f.write("K3, 3, 3, %.9f \n" %(float(H0[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
# Output
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==========================================================================
# STEP 2  Unload - prescribe H1
# ==========================================================================
print 'STEP-2'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step2.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write('** \n')
f.write('** BOUNDARY CONDITIONS\n')
f.write('** \n')
f.write('*Boundary, op=NEW\n')
f.write("K1, 1, 1, %.9f \n" %(float(H1[0][0])))
f.write("K1, 2, 2, %.9f \n" %(float(H1[0][1])))
f.write("K1, 3, 3, %.9f \n" %(float(H1[0][2])))
f.write("K2, 1, 1, %.9f \n" %(float(H1[1][0])))
f.write("K2, 2, 2, %.9f \n" %(float(H1[1][1])))
f.write("K2, 3, 3, %.9f \n" %(float(H1[1][2])))
f.write("K3, 1, 1, %.9f \n" %(float(H1[2][0])))
f.write("K3, 2, 2, %.9f \n" %(float(H1[2][1])))
f.write("K3, 3, 3, %.9f \n" %(float(H1[2][2])))
if rb != 'LDBC':
```

```
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==========================================================================
# STEP 3  H2 -> Column 1
# ==========================================================================
print 'STEP-3'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step3.py','r')
for line in step:
    f.write('%s' %line)
step.close()
#Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K1, 1, 1, %.9f \n" %(float(H2[0][0])))
f.write("*Boundary \n")
f.write("K1, 2, 2, %.9f \n" %(float(H2[0][1])))
f.write("*Boundary \n")
f.write("K1, 3, 3, %.9f \n" %(float(H2[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K2, 1, 1, %.9f \n" %(float(H2[1][0])))
f.write("*Boundary \n")
f.write("K2, 2, 2, %.9f \n" %(float(H2[1][1])))
f.write("*Boundary \n")
f.write("K2, 3, 3, %.9f \n" %(float(H2[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K3, 1, 1, %.9f \n" %(float(H2[2][0])))
f.write("*Boundary \n")
f.write("K3, 2, 2, %.9f \n" %(float(H2[2][1])))
f.write("*Boundary \n")
f.write("K3, 3, 3, %.9f \n" %(float(H2[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
```

```
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
f.close()
# ==========================================================================
#      Run Job
# ==========================================================================
#print '=============================================================',
#print 'Job Spalte',spalte+1
#print '=============================================================',

if "posix" in os.name:
    execfile('0_job.py')
# Wait for lck file, then wait for lck file is deleted -> job finished
t0 = time.time();
while not os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
while os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
# Check if calculation was successful, else STOP
time.sleep(5)
f = open(inpname+'.sta','r')
sta = []
for line in f:
    sta.append(str(line.rstrip()))
f.close()
if sta[len(sta)-1] == ' THE ANALYSIS HAS COMPLETED SUCCESSFULLY':
    print sta[len(sta)-1]
else:
    print sta[len(sta)-1]
    f = open('0_Auswertung_ERROR_in_'+inpname+'.txt','w')
    f.close()
    time.sleep(5)
    sys.exit("print sta[len(sta)-1]")
# ==========================================================================
#      ODB oeffnen und RF1,RF2,RF3 an K1, K2, K3 rausschreiben
# ==========================================================================
# Generate path
pfad = inpname
# Search for node number K1NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K1':
    read = str(f.readline())
read = str(f.readline())
K1NR = int(read[0:len(read)-3])
f.close()
# Search for node number K2NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K2':
    read = str(f.readline())
read = str(f.readline())
K2NR = int(read[0:len(read)-3])
```

```
f.close()
# Search for node number K3NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K3':
    read = str(f.readline())
read = str(f.readline())
K3NR = int(read[0:len(read)-3])
f.close()
#=============================================================
#      Column 1 Step-3
#=============================================================
o1 = session.openOdb(
    name=inpname+'.odb')
session.viewports['Viewport: 1'].setValues(displayedObject=o1)
odbName=session.viewports[session.currentViewportName].odbDisplay.name
session.odbData[odbName].setValues(activeFrames=(('Step-3', ('0:-1', )), ))
odb = session.odbs[inpname+'.odb']
session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('RF',
    NODAL, ((COMPONENT, 'RF1'), (COMPONENT, 'RF2'), (COMPONENT, 'RF3'), )),
    ), nodeSets=('K1', 'K2', 'K3', ))
increm = len(session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data) - 1

# Generate matrix T1PK
T1PK2 = nullmatrix(3,3)
T1PK2[0][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[1][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[2][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR]
print('T1PK2')
for line in T1PK2: print line
odb.close()
# Calculate T2PK2
# F2 = H2 + 1
F2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F2[i][j] = H2[i][j]
F2[0][0] += 1.
F2[1][1] += 1.
F2[2][2] += 1.
F2i = invert3(F2)
T2PK2 = matmul(F2i,T1PK2)
```

```
# Calculate E2_green berechnen
# E2_green = 1/2 (F2T*F-1)
FT2 = matrixTranspose(F2)
EG2 = matmul(FT2,F2)
EG2[0][0] -= 1.
EG2[1][1] -= 1.
EG2[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG2[i][j] *= 0.5
# Calculate delta EG and delta T2PK
DEG = nullmatrix(3,3)
DT2PK = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        DEG[i][j]   = EG2[i][j]-EG1[i][j]
        DT2PK[i][j] = T2PK2[i][j]-T2PK1[i][j]
print('EG1')
for line in EG1: print line
print('EG2')
for line in EG2: print line
print('Delta EG')
for line in DEG: print line

print('T2PK1')
for line in T2PK1: print line
print('T2PK2')
for line in T2PK2: print line
print('Delta T2PK')
for line in DT2PK: print line
# ============================================================
# Generate stress and strain output for Mathematica post processing
# ============================================================
# Generate vector of symmetric Green strain tensor EG
matheE.write('%e,' %(DEG[0][0]))
matheE.write('%e,' %(DEG[1][1]))
matheE.write('%e,' %(DEG[2][2]))
matheE.write('%e,' %(DEG[0][1]))
matheE.write('%e,' %(DEG[0][2]))
matheE.write('%e,' %(DEG[1][2]))
# Generate vector of symmetric 2nd Piola-Kirchhoff stress tensor T2PK
matheT.write('%e,' %(DT2PK[0][0]))
matheT.write('%e,' %(DT2PK[1][1]))
matheT.write('%e,' %(DT2PK[2][2]))
matheT.write('%e,' %(DT2PK[0][1]))
matheT.write('%e,' %(DT2PK[0][2]))
matheT.write('%e,' %(DT2PK[1][2]))
matheE.close()
matheT.close()
print 'FINISHED'
```

B.8 Elastic test strain "0_6.py"

```
# ============================================================
# Elastic test strain calculation 6 of 6
# ============================================================
# ============================================================
# Start with: nohup /opt/abaqus/Commands/abaqus cae noGUI=*.py &
# ============================================================
```

```
from abaqus import *
from abaqusConstants import *
import __main__
import sys
import string
import fileinput
import shutil
import os
import subprocess
import time
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import odbAccess
import random
import csv
import resource
import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
#==========================================================
execfile('0_subroutines.py')
#==========================================================
# 0_6.py PART1
#==========================================================
extname = '6'
# Define delta value (small elastic test strain)
delta = 1.e-6
matheE = open('K0_matheEGREEN_einzeln.txt','w')
matheT = open('K0_matheT2PK_einzeln.txt','w')
# Read H0 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H0.txt','r')
H0 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    data[2] = data[2].replace('\n','')
    H0.append(data)
f.close()
print('H0')
```

```
for line in H0: print line
# Read H1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_H1.txt','r')
H1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    H1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        H1[i][j] = float(H1[i][j])
print('H1')
for line in H1: print line
# Read T2PK1 from Start-up calculation 1
f = open('K0_ANLAUFRECHNUNG_T1.txt','r')
T2PK1 = []
for i in range(0,3):
    read = str(f.readline())
    data = read.split(",")
    T2PK1.append(data)
f.close()
for i in range(3):
    for j in range(3):
        T2PK1[i][j] = float(T2PK1[i][j])
print('T2PK1')
for line in T2PK1: print line
# Calculate F1
F1 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F1[i][j] = H1[i][j]
F1[0][0] += 1.
F1[1][1] += 1.
F1[2][2] += 1.
print('F1')
for line in F1: print line
# Calculate EGREEN1 = 1/2 (F1T*F1-1)
F1T = matrixTranspose(F1)
EG1 = matmul(F1T,F1)
EG1[0][0] -= 1.
EG1[1][1] -= 1.
EG1[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG1[i][j] *= 0.5
print('EG1')
for line in EG1: print line
# H2 = H1 + delta
H2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        H2[i][j] = H1[i][j]
H2[1][2] = H1[1][2] + delta
H2[2][1] = H1[2][1] + delta
print('H2')
for line in H2: print line
# ================================================================
execfile('0_RVE.py')
# ================================================================
# 0_6.py  PART2
# ================================================================
# STEP 1 Large Deformation
# ================================================================
print 'STEP-1'
f.write('** ----------------------------------------------------------\n')
```

```
# Read step information
step = open('0_step1.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
if H0[0][0] != '-':
    f.write("*Boundary \n")
    f.write("K1, 1, 1, %.9f \n" %(float(H0[0][0])))
if H0[0][1] != '-':
    f.write("*Boundary \n")
    f.write("K1, 2, 2, %.9f \n" %(float(H0[0][1])))
if H0[0][2] != '-':
    f.write("*Boundary \n")
    f.write("K1, 3, 3, %.9f \n" %(float(H0[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
if H0[1][0] != '-':
    f.write("*Boundary \n")
    f.write("K2, 1, 1, %.9f \n" %(float(H0[1][0])))
if H0[1][1] != '-':
    f.write("*Boundary \n")
    f.write("K2, 2, 2, %.9f \n" %(float(H0[1][1])))
if H0[1][2] != '-':
    f.write("*Boundary \n")
    f.write("K2, 3, 3, %.9f \n" %(float(H0[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
if H0[2][0] != '-':
    f.write("*Boundary \n")
    f.write("K3, 1, 1, %.9f \n" %(float(H0[2][0])))
if H0[2][1] != '-':
    f.write("*Boundary \n")
    f.write("K3, 2, 2, %.9f \n" %(float(H0[2][1])))
if H0[2][2] != '-':
    f.write("*Boundary \n")
    f.write("K3, 3, 3, %.9f \n" %(float(H0[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
# Output
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==========================================================
# STEP 2  Unload - prescribe H1
# ==========================================================
```

```
print 'STEP-2'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step2.py','r')
for line in step:
    f.write('%s' %line)
step.close()
# Boundary Conditions
f.write('** \n')
f.write('** BOUNDARY CONDITIONS\n')
f.write('** \n')
f.write('*Boundary, op=NEW\n')
f.write("K1, 1, 1, %.9f \n" %(float(H1[0][0])))
f.write("K1, 2, 2, %.9f \n" %(float(H1[0][1])))
f.write("K1, 3, 3, %.9f \n" %(float(H1[0][2])))
f.write("K2, 1, 1, %.9f \n" %(float(H1[1][0])))
f.write("K2, 2, 2, %.9f \n" %(float(H1[1][1])))
f.write("K2, 3, 3, %.9f \n" %(float(H1[1][2])))
f.write("K3, 1, 1, %.9f \n" %(float(H1[2][0])))
f.write("K3, 2, 2, %.9f \n" %(float(H1[2][1])))
f.write("K3, 3, 3, %.9f \n" %(float(H1[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
# ==============================================================
# STEP 3  H2 -> Column 1
# ==============================================================
print 'STEP-3'
f.write('** ----------------------------------------------------------------\n')
# Read step information
step = open('0_step3.py','r')
for line in step:
    f.write('%s' %line)
step.close()
#Boundary Conditions
f.write("**\n")
f.write("**BOUNDARY CONDITIONS\n")
f.write("**\n")
f.write("** Name: BC-K1 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K1, 1, 1, %.9f \n" %(float(H2[0][0])))
f.write("*Boundary \n")
f.write("K1, 2, 2, %.9f \n" %(float(H2[0][1])))
f.write("*Boundary \n")
f.write("K1, 3, 3, %.9f \n" %(float(H2[0][2])))
f.write("** Name: BC-K2 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
```

```
f.write("K2, 1, 1, %.9f \n" %(float(H2[1][0])))
f.write("*Boundary \n")
f.write("K2, 2, 2, %.9f \n" %(float(H2[1][1])))
f.write("*Boundary \n")
f.write("K2, 3, 3, %.9f \n" %(float(H2[1][2])))
f.write("** Name: BC-K3 Type: Displacement/Rotation\n")
f.write("*Boundary \n")
f.write("K3, 1, 1, %.9f \n" %(float(H2[2][0])))
f.write("*Boundary \n")
f.write("K3, 2, 2, %.9f \n" %(float(H2[2][1])))
f.write("*Boundary \n")
f.write("K3, 3, 3, %.9f \n" %(float(H2[2][2])))
if rb != 'LDBC':
    f.write("** Name: Ursprung Type: Displacement/Rotation\n")
    f.write("*Boundary \n")
    f.write("Ursprung, 1, 1, 0 \n" )
    f.write("Ursprung, 2, 2, 0 \n" )
    f.write("Ursprung, 3, 3, 0 \n" )
f.write('** \n')
f.write('** OUTPUT REQUESTS\n')
f.write('** \n')
f.write('*Restart, write, frequency=0\n')
f.write('** \n')
f.write('** FIELD OUTPUT: F-Output-1\n')
f.write('**\n')
f.write('*Output, field, frequency=1\n')
f.write('*Node Output\n')
f.write('CF, RF, U\n')
f.write('*Element Output, directions=YES\n')
f.write('E, EE, ELEN, ENER, EVOL, IE, LE, PE, PEEQ, PEMAG, S, SDV\n')
f.write('** \n')
f.write('** HISTORY OUTPUT: H-Output-1\n')
f.write('** \n')
f.write('*Output, history, variable=PRESELECT\n')
f.write('*End Step\n')
f.close()
# ==========================================================================
#      Run Job
# ==========================================================================
#print '=============================================================';
#print 'Job Spalte',spalte+1
#print '=============================================================';

if "posix" in os.name:
    execfile('0_job.py')
# Wait for lck file, then wait for lck file is deleted -> job finished
t0 = time.time();
while not os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
while os.path.isfile(inpname+".lck"):
  time.sleep(0.1)
  tn = time.time();
# Check if calculation was successful, else STOP
time.sleep(5)
f = open(inpname+'.sta','r')
sta = []
for line in f:
    sta.append(str(line.rstrip()))
f.close()
if sta[len(sta)-1] == ' THE ANALYSIS HAS COMPLETED SUCCESSFULLY':
    print sta[len(sta)-1]
else:
    print sta[len(sta)-1]
    f = open('0_Auswertung_ERROR_in_'+inpname+'.txt','w')
    f.close()
```

```
    time.sleep(5)
    sys.exit("print sta[len(sta)-1]")
# ================================================================
#     ODB oeffnen und RF1,RF2,RF3 an K1, K2, K3 rausschreiben
# ================================================================
# Generate path
pfad = inpname
# Search for node number K1NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K1':
    read = str(f.readline())
read = str(f.readline())
K1NR = int(read[0:len(read)-3])
f.close()
# Search for node number K2NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K2':
    read = str(f.readline())
read = str(f.readline())
K2NR = int(read[0:len(read)-3])
f.close()
# Search for node number K3NR
f = open(inpname + '.inp','r')
read = str(f.readline())
while read[0:14] != '*Nset, nset=K3':
    read = str(f.readline())
read = str(f.readline())
K3NR = int(read[0:len(read)-3])
f.close()
# ================================================================
#     Column 1 Step-3
# ================================================================
o1 = session.openOdb(
    name=inpname+'.odb')
session.viewports['Viewport: 1'].setValues(displayedObject=o1)
odbName=session.viewports[session.currentViewportName].odbDisplay.name
session.odbData[odbName].setValues(activeFrames=(('Step-3', ('0:-1', )), ))
odb = session.odbs[inpname+'.odb']
session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('RF',
    NODAL, ((COMPONENT, 'RF1'), (COMPONENT, 'RF2'), (COMPONENT, 'RF3'), )),
    ), nodeSets=('K1', 'K2', 'K3', ))
increm = len(session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data) - 1

# Generate matrix T1PK
T1PK2 = nullmatrix(3,3)
T1PK2[0][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[0][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR].data[increm
    ][1]/V0
T1PK2[1][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[1][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR].data[increm
    ][1]/V0
T1PK2[2][0] = session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][1] = session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
T1PK2[2][2] = session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR].data[increm
    ][1]/V0
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K1NR]
```

```
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K1NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K2NR]
del session.xyDataObjects['RF:RF1 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF2 PI: FIBBI-1 N: %d' %K3NR]
del session.xyDataObjects['RF:RF3 PI: FIBBI-1 N: %d' %K3NR]
print('T1PK2')
for line in T1PK2: print line
odb.close()
# Calculate T2PK2
# F2 = H2 + 1
F2 = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        F2[i][j] = H2[i][j]
F2[0][0] += 1.
F2[1][1] += 1.
F2[2][2] += 1.
F2i = invert3(F2)
T2PK2 = matmul(F2i,T1PK2)
# Calculate E2_green berechnen
# E2_green = 1/2 (F2T*F-1)
FT2 = matrixTranspose(F2)
EG2 = matmul(FT2,F2)
EG2[0][0] -= 1.
EG2[1][1] -= 1.
EG2[2][2] -= 1.
for i in range(3):
    for j in range(3):
        EG2[i][j] *= 0.5
# Calculate delta EG and delta T2PK
DEG = nullmatrix(3,3)
DT2PK = nullmatrix(3,3)
for i in range(3):
    for j in range(3):
        DEG[i][j]   = EG2[i][j]-EG1[i][j]
        DT2PK[i][j] = T2PK2[i][j]-T2PK1[i][j]
print('EG1')
for line in EG1: print line
print('EG2')
for line in EG2: print line
print('Delta EG')
for line in DEG: print line

print('T2PK1')
for line in T2PK1: print line
print('T2PK2')
for line in T2PK2: print line
print('Delta T2PK')
for line in DT2PK: print line
# ==========================================================
# Generate stress and strain output for Mathematica post processing
# ==========================================================
# Generate vector of symmetric Green strain tensor EG
matheE.write('%e,' %(DEG[0][0]))
matheE.write('%e,' %(DEG[1][1]))
matheE.write('%e,' %(DEG[2][2]))
matheE.write('%e,' %(DEG[0][1]))
matheE.write('%e,' %(DEG[0][2]))
matheE.write('%e,' %(DEG[1][2]))
# Generate vector of symmetric 2nd Piola-Kirchhoff stress tensor T2PK
matheT.write('%e,' %(DT2PK[0][0]))
matheT.write('%e,' %(DT2PK[1][1]))
matheT.write('%e,' %(DT2PK[2][2]))
```

```
matheT.write('%e, ' %(DT2PK[0][1]))
matheT.write('%e, ' %(DT2PK[0][2]))
matheT.write('%e, ' %(DT2PK[1][2]))
matheE.close()
matheT.close()
print 'FINISHED'
```

B.9 Subroutines "0_subroutines.py"

```
# ============================================================
# Define subroutines
# ============================================================
def nullmatrix(m,n):
    # Zero matrix
    matrix = [[0 for row in range(n)] for col in range(m)]
    return matrix

def nullvektor(m):
    # Zero vector
    vektor = [0 for row in range(m)]
    return vektor

def matmul(matrix1,matrix2):
    # Matrix multiplication
    if len(matrix1[0]) != len(matrix2):
        # Check matrix dimension
        print 'm*n and n*p multiply!'
    else:
        # Multiply if correct dimensions
        matrix = nullmatrix(len(matrix1),len(matrix2[0]))
        for i in range(len(matrix1)):
            for j in range(len(matrix2[0])):
                for k in range(len(matrix2)):
                    matrix[i][j] += matrix1[i][k]*matrix2[k][j]
        return matrix

def det3(m33):
    # Determinant det of 3x3 matrix m33
    det  = m33[0][0]*m33[1][1]*m33[2][2]
    det += m33[0][1]*m33[1][2]*m33[2][0]
    det += m33[0][2]*m33[1][0]*m33[2][1]
    det += m33[2][0]*m33[1][1]*m33[0][2]
    det += m33[2][1]*m33[1][2]*m33[0][0]
    det += m33[2][2]*m33[1][0]*m33[0][1]
    return det

def invert3(m):
    #Inverse matrix mi of 3x3 matrix m
    hi3m = det3(m)
    mi = nullmatrix(3,3)
    if(abs(hi3m) > 0.000000001):
        mi[0][0]=m[1][1]*m[2][2]/hi3m-m[2][1]*m[1][2]/hi3m
        mi[1][0]=m[2][0]*m[1][2]/hi3m-m[1][0]*m[2][2]/hi3m
        mi[2][0]=m[1][0]*m[2][1]/hi3m-m[2][0]*m[1][1]/hi3m
        mi[0][1]=m[2][1]*m[0][2]/hi3m-m[0][1]*m[2][2]/hi3m
        mi[1][1]=m[0][0]*m[2][2]/hi3m-m[2][0]*m[0][2]/hi3m
        mi[2][1]=m[2][0]*m[0][1]/hi3m-m[0][0]*m[2][1]/hi3m
        mi[0][2]=m[0][1]*m[1][2]/hi3m-m[1][1]*m[0][2]/hi3m
        mi[1][2]=m[1][0]*m[0][2]/hi3m-m[0][0]*m[1][2]/hi3m
```

```
        mi[2][2]=m[0][0]*m[1][1]/hi3m-m[1][0]*m[0][1]/hi3m
    return mi

def clearall():
    #Clear all variables
    all = [var for var in globals() if (var[:2], var[-2:]) != ("__", "__")]
    for var in all:
        del globals()[var]

def lastfall(delta):
    #Load case in dependence of delta
    N = nullmatrix(3,3)
    N[0][0] =sqrt(2./3.)*cos(delta+2./3.*pi  )
    N[0][1] = 0.
    N[0][2] = 0.
    N[1][0] = 0.
    N[1][1] =sqrt(2./3.)*cos(delta+4./3.*pi  )
    N[1][2] = 0.
    N[2][0] = 0.
    N[2][1] = 0.
    N[2][2] =sqrt(2./3.)*cos(delta )
    return N

def matrixTranspose(anArray):
    #Transposed matrix transposed of matrix anArray
    transposed = [None]*len(anArray[0])
    for t in range(len(anArray)):
        transposed[t] = [None]*len(anArray)
        for tt in range(len(anArray[t])):
            transposed[t][tt] = anArray[tt][t]
    return transposed
```

B.10 Start Abaqus job "0_job.py"

```
# ==========================================================================
# Start Abaqus job
# ==========================================================================
#UMAT
job = '/home/martin/Commands/abaqus job='+inpname+' user=0_umat.f cpu='+cpunumber+'
      ask_delete=OFF'
os.system(job)
```

B.11 Material parameters "0_material.py"

```
# ==========================================================================
# Material definitions
# ==========================================================================
*Material, name=Fiber1
*Depvar
      10,
*User Material, constants=3
```

```
100000.,0.3,200.
*INITIAL CONDITIONS, TYPE=SOLUTION, USER
**
*Material, name=Fiber2
*Depvar
      10,
*User Material, constants=3
100000.,0.3,200.
*INITIAL CONDITIONS, TYPE=SOLUTION, USER
**
*Material, name=Matrix
*Depvar
      10,
*User Material, constants=3
10000.,0.3,100.
*INITIAL CONDITIONS, TYPE=SOLUTION, USER
```

B.12 Settings step 1 "0_step1.py"

```
** #=============================================================
** #Step 1
** #=============================================================
**
** STEP: Step-1
**
*Step, name=Step-1, nlgeom=YES, inc=10000000
*Static
**START,GESAMT,MIN,MAX
0.1, 100., 1e-10, 5.
*CONTROLS, ANALYSIS=DISCONTINUOUS
```

B.13 Settings step 2 "0_step2.py"

```
** #=============================================================
** #Step 2
** #=============================================================
**
** STEP: Step-2
**
*Step, name=Step-2, nlgeom=YES, inc=10000000
*Static
**START,GESAMT,MIN,MAX
0.1, 100., 1e-10, 5.
*CONTROLS, ANALYSIS=DISCONTINUOUS
```

B.14 Settings step 3 “0_step3.py”

```
** #============================================================
** #Step 3
** #============================================================
**
** STEP: Step-3
**
*Step, name=Step-3, nlgeom=YES, inc=10000000
*Static
**START,GESAMT,MIN,MAX
100., 100., 1e-10, 100.
*CONTROLS, ANALYSIS=DISCONTINUOUS
```

Appendix C

Fortran source codes for the user material

C.1 Subroutine UMAT

```
c ===================================================================
c ===================================================================
c = Subroutine UMAT
c ===================================================================
c ===================================================================
      SUBROUTINE UMAT(STRESS,STATEV,DDSDDE,SSE,SPD,SCD,
     1 RPL,DDSDDT,DRPLDE,DRPLDT,
     2 STRAN,DSTRAN,TIME,DTIME,TEMP,DTEMP,PREDEF,DPRED,CMNAME,
     3 NDI,NSHR,NTENS,NSTATV,PROPS,NPROPS,COORDS,DROT,PNEWDT,
     4 CELENT,DFGRD0,DFGRD1,NOEL,NPT,LAYER,KSPT,KSTEP,KINC)
      IMPLICIT NONE
c ===================================================================
c     UMAT definitions
c ===================================================================
      INTEGER kstep,kspt,layer,npt,noel,nprops,nstatv,ntens,
     1 nshr,ndi,kinc
      DOUBLE PRECISION sse,spd,scd,rpl,drpldt,dtime,temp,dtemp,
     1 pnewdt,celent
      DOUBLE PRECISION dfgrd0(3,3),dfgrd1(3,3),time(2),stress(ntens),
     1 statev(nstatv),ddsdde(ntens,ntens),ddsddt(ntens),drplde(ntens),
     2 stran(ntens),dstran(ntens),predef(1),dpred(1),props(nprops),
     3 coords(3),drot(3,3)
      CHARACTER*80 cmname
c ===================================================================
c     Own definitions
c ===================================================================
      integer size,npassin,i,j,k,l
      parameter (size = 10)
      parameter (npassin = 18)
      double precision Pn(3,3),Pnv(9)
      double precision x(size),x2(size),xalt(size),passin(npassin)
      double precision T2PK(3,3),T(3,3)
```

```
      double precision Tdev(3,3),Tdevnorm,sigmav
      double precision Pn1(3,3),Pn1T(3,3)
      double precision emo,nu,dreik,zweig,rd,rd1,rd2
      double precision id(3,3),K0(3,3,3,3)
      double precision dfgrd1T(3,3),CR(3,3),CRv(6)
      double precision Jac,det,ddsddedelta
      double precision T2PKdelta(3,3),m6(6,6),ddsdde2(6,6)
c      write(*,*) "UMAT start"
c      if(npt == 1 .and. noel == 1) then
c      write(*,*) "=================================================="
c      write(*,*) "= Step",kstep,"Inc",kinc,"npt",npt,"noel",noel
c      write(*,*) "=================================================="
c      write(*,*) "dfgrd1="
c      call printmatrix(dfgrd1,3)
c      end if
c ================================================================
c      Starting values Pn
c ================================================================
      Pn(1,1) = statev(1)
      Pn(1,2) = statev(2)
      Pn(1,3) = statev(3)
      Pn(2,1) = statev(4)
      Pn(2,2) = statev(5)
      Pn(2,3) = statev(6)
      Pn(3,1) = statev(7)
      Pn(3,2) = statev(8)
      Pn(3,3) = statev(9)
c ================================================================
c      Build Pnv vector from tensor PN
c ================================================================
      CALL TENSOR_VEKTOR(Pn,Pnv)
c ================================================================
c      Starting values 1...9 for x=pn+1 are pn
c ================================================================
      x(1) = statev(1)
      x(2) = statev(2)
      x(3) = statev(3)
      x(4) = statev(4)
      x(5) = statev(5)
      x(6) = statev(6)
      x(7) = statev(7)
      x(8) = statev(8)
      x(9) = statev(9)
c ================================================================
c      Starting value lambda
c ================================================================
      x(10) = statev(10)
c ================================================================
c      Store starting value of lambda as x2
c ================================================================
      x2 = x
c ================================================================
```

```
c      Calculate right Cauchy-Green tensor CR and store as vector
c ===================================================================
       dfgrd1T = transpose(dfgrd1)
       CR = matmul(dfgrd1T,dfgrd1)
       call sym_tensor_vektor(CR,CRv)
c       if(npt == 1 .and. noel == 1) then
c       write(*,*) "CR"
c       call printmatrix(CR,3)
c       end if
c       write(*,*) "CRv"
c       call printvektor(CRv,6)
c ===================================================================
c      Build passin vector
c ===================================================================
       passin(1)=CRv(1)
       passin(2)=CRv(2)
       passin(3)=CRv(3)
       passin(4)=CRv(4)
       passin(5)=CRv(5)
       passin(6)=CRv(6)
       passin(7)=999999999.d0
       passin(8)=999999999.d0
       passin(9)=999999999.d0
c ===================================================================
c       passin(10..18) = pn
c ===================================================================
       passin(10)=Pnv(1)
       passin(11)=Pnv(2)
       passin(12)=Pnv(3)
       passin(13)=Pnv(4)
       passin(14)=Pnv(5)
       passin(15)=Pnv(6)
       passin(16)=Pnv(7)
       passin(17)=Pnv(8)
       passin(18)=Pnv(9)
c       if(npt == 1 .and. noel == 1) then
c       write(*,*) "passin= START UMAT vor stresses"
c       call printvektor(passin,npassin)
c       end if
c ===================================================================
c      Call subroutine stresses to calculate the stresses T2PK
c ===================================================================
       call stresses(passin,npassin,x,size,T2PK,npt,noel,nprops,props)
c ===================================================================
c      Statev update
c ===================================================================
       statev(1) = x(1)
       statev(2) = x(2)
       statev(3) = x(3)
       statev(4) = x(4)
       statev(5) = x(5)
       statev(6) = x(6)
```

```
      statev(7) = x(7)
      statev(8) = x(8)
      statev(9) = x(9)
      statev(10) = x(10)
c ===================================================================
c     Stress update
c ===================================================================
      Jac = det(dfgrd1)
      T = 1./Jac*matmul(matmul(dfgrd1,T2PK),dfgrd1T)
      stress(1) = T(1,1)
      stress(2) = T(2,2)
      stress(3) = T(3,3)
      stress(4) = T(1,2)
      stress(5) = T(1,3)
      stress(6) = T(2,3)
c      if(npt == 1 .and. noel == 1) then
c      write(*,*) "stress"
c      call printvektor(stress,6)
c      write(*,*) "P"
c      call printvektor(statev(1:9),9)
c      end if
c ===================================================================
c     DDSDDE update numerical calculation
c ===================================================================
c     Test strain ddsddedelta
      ddsddedelta = 1.d-7
c      if(npt == 1 .and. noel == 1) then
c      write(*,*) "Start DDSDDE numerical calculation"
c      end if
c ===================================================================
c     Do-Loop applying 6 test strains, call subroutine stresses
c ===================================================================
      do i=1,6
      passin(i) = passin(i) + ddsddedelta
c      if(npt == 1 .and. noel == 1) then
c      write(*,*) "i=",i,"x2"
c      call printvektor(x2,10)
c      write(*,*) "passin="
c      call printvektor(passin,npassin)
c      end if
      xalt = x2
      call stresses(passin,npassin,x2,size,T2PKdelta,npt,noel,
     & nprops,props)
c      write(*,*) "T2PKdelta"
c      call printvektor(T2PKdelta,6)
      x2 = xalt
c ===================================================================
c     Difference quotient to calculate the new ddsdde2
c ===================================================================
      passin(i) = passin(i) - ddsddedelta
      ddsdde2(1,i)=(T2PKdelta(1,1)-T2PK(1,1))/ddsddedelta
      ddsdde2(2,i)=(T2PKdelta(2,2)-T2PK(2,2))/ddsddedelta
```

```
      ddsdde2(3,i)=(T2PKdelta(3,3)-T2PK(3,3))/ddsddedelta
      ddsdde2(4,i)=(T2PKdelta(1,2)-T2PK(1,2))/ddsddedelta
      ddsdde2(5,i)=(T2PKdelta(1,3)-T2PK(1,3))/ddsddedelta
      ddsdde2(6,i)=(T2PKdelta(2,3)-T2PK(2,3))/ddsddedelta
      end do
c ================================================================
c     Normalization
c ================================================================
      rd = dsqrt(2.d0)
      ddsdde2(4:6,1:3)=ddsdde2(4:6,1:3)*rd
      rd = dsqrt(0.5d0)
      ddsdde2(1:3,4:6)=ddsdde2(1:3,4:6)*rd
c     no normalisation, because sqrt2*sqrt2 is already 2
c      rd = 2.d0
c      ddsdde2(4:6,4:6)=ddsdde2(4:6,4:6)*rd
c      write(*,*) "ddsdde2"
c ================================================================
c      Soubroutine m3to6_cowin to transform F from 3x3 to 6x6
c ================================================================
      call m3to6_cowin(dfgrd1,m6)
      ddsdde = matmul(matmul(m6,ddsdde2),transpose(m6))*2.d0/Jac
c ================================================================
c     remove the normalization because abaqus requires unnormalized
c ================================================================
      rd=dsqrt(0.5d0)
      ddsdde(1,4)=ddsdde(1,4)*rd
      ddsdde(1,5)=ddsdde(1,5)*rd
      ddsdde(1,6)=ddsdde(1,6)*rd
      ddsdde(2,4)=ddsdde(2,4)*rd
      ddsdde(2,5)=ddsdde(2,5)*rd
      ddsdde(2,6)=ddsdde(2,6)*rd
      ddsdde(3,4)=ddsdde(3,4)*rd
      ddsdde(3,5)=ddsdde(3,5)*rd
      ddsdde(3,6)=ddsdde(3,6)*rd
      ddsdde(4,1)=ddsdde(4,1)*rd
      ddsdde(5,1)=ddsdde(5,1)*rd
      ddsdde(6,1)=ddsdde(6,1)*rd
      ddsdde(4,2)=ddsdde(4,2)*rd
      ddsdde(5,2)=ddsdde(5,2)*rd
      ddsdde(6,2)=ddsdde(6,2)*rd
      ddsdde(4,3)=ddsdde(4,3)*rd
      ddsdde(5,3)=ddsdde(5,3)*rd
      ddsdde(6,3)=ddsdde(6,3)*rd
      ddsdde(4,4)=ddsdde(4,4)*0.5d0
      ddsdde(4,5)=ddsdde(4,5)*0.5d0
      ddsdde(4,6)=ddsdde(4,6)*0.5d0
      ddsdde(5,4)=ddsdde(5,4)*0.5d0
      ddsdde(5,5)=ddsdde(5,5)*0.5d0
      ddsdde(5,6)=ddsdde(5,6)*0.5d0
      ddsdde(6,4)=ddsdde(6,4)*0.5d0
      ddsdde(6,5)=ddsdde(6,5)*0.5d0
      ddsdde(6,6)=ddsdde(6,6)*0.5d0
```

```
c       if(npt == 1 .and. noel == 1) then
c       write(*,*) "ddsdde numerisch",i
c       call printmatrix(ddsdde,6)
c       end if
       end
c ================================================================
c ================================================================
c === Include SDVINI
c ================================================================
c ================================================================
       include '0_sdvini'
```

C.2 Subroutine STRESSES

```
c ================================================================
c ================================================================
c === Subroutine STRESSES
c ================================================================
c ================================================================
       subroutine stresses(passin,npassin,x,size,T2PK,npt,noel,
      & nprops,props)
       implicit none
       integer i,j,size,npassin,npt,noel,it,nprops
       integer maxit,msglinesearch,msgjac,msgquasi
       double precision eps,dmp(size),residuum0(size),props(nprops)
       double precision CR(3,3),CRv(6),passin(npassin),Pn(3,3)
       double precision T2PKtilde(3,3),T(3,3),CRtilde(3,3)
       double precision Pn1(3,3),x(size),T2PK(3,3)
c       if(npt == 1 .and. noel == 1) then
c       write(*,*) "stress start"
c       end if
c ================================================================
c       Call subroutine resmaterial to calculate residuum0
c ================================================================
       call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
      & nprops,props)
c       if(npt == 1 .and. noel == 1) then
c       write(*,*) "PRAEDIKTORSCHRITT"
c       write(*,*) "LAMBDA=",x(10),"sigmav-sigmaf=",residuum0(10)
c       end if
c ================================================================
c       ELASTIC if residuum0 = sigmav-sigmaf is lower than 0
c ================================================================
       if(residuum0(10).LT.0.d0) then
c       if(npt == 1 .and. noel == 1) then
c       write(*,*)"elastic"
c       end if
```

```
c ==========================================================================
c       Calculate stresses
c ==========================================================================
      CRv(1) = passin(1)
      CRv(2) = passin(2)
      CRv(3) = passin(3)
      CRv(4) = passin(4)
      CRv(5) = passin(5)
      CRv(6) = passin(6)
      call sym_vektor_tensor(CRv,CR)
      Pn(1,1) = passin(10)
      Pn(1,2) = passin(11)
      Pn(1,3) = passin(12)
      Pn(2,1) = passin(13)
      Pn(2,2) = passin(14)
      Pn(2,3) = passin(15)
      Pn(3,1) = passin(16)
      Pn(3,2) = passin(17)
      Pn(3,3) = passin(18)
c~        write(*,*) "passin in ellaw="
c~        call printvektor(passin,npassin)
c ==========================================================================
c     Call subroutine ellaw to calculate stresses T2PKtilde
c ==========================================================================
      call ellaw(CR,Pn,T2PKtilde,CRtilde,npt,noel,nprops,props)
      T2PK = matmul(matmul(Pn,T2PKtilde),transpose(Pn))
c       if(npt == 1 .and. noel == 1) then
c       write(*,*) "passin= ende elastisch stresses"
c       call printvektor(passin,npassin)
c       end if
c ==========================================================================
c       PLASTIC if residuum0 = sigmav-sigmaf is greater than or equal 0
c ==========================================================================
      else
c       if(npt == 1 .and. noel == 1) then
c       write(*,*)"plastic"
c       end if
c ==========================================================================
c       Settings for the Newton solver
c ==========================================================================
      eps=1.d-10
      maxit=500
      msglinesearch=0
      msgjac=1
      msgquasi=0
      dmp=1.d0
c       if(npt == 1 .and. noel == 1) then
c       write(*,*) "passin= vor solven"
c       call printvektor(passin,npassin)
c       end if
c ==========================================================================
c       Call subroutine solven - Newton solver
```

```
c ==========================================================================
      call solven(x,size,passin,npassin,eps,maxit,dmp,
     &  msglinesearch,msgjac,msgquasi,npt,noel,it,nprops,props)
c     x in Matrix schreiben (Pn1)
c      if(npt == 1 .and. noel == 1) then
c      write(*,*) "it=",it
c      call printvektor(passin,npassin)
c      end if
c ==========================================================================
c      Results from the Newton solver Pn1 and passin
c ==========================================================================
      Pn1(1,1) = x(1)
      Pn1(1,2) = x(2)
      Pn1(1,3) = x(3)
      Pn1(2,1) = x(4)
      Pn1(2,2) = x(5)
      Pn1(2,3) = x(6)
      Pn1(3,1) = x(7)
      Pn1(3,2) = x(8)
      Pn1(3,3) = x(9)
      CRv(1) = passin(1)
      CRv(2) = passin(2)
      CRv(3) = passin(3)
      CRv(4) = passin(4)
      CRv(5) = passin(5)
      CRv(6) = passin(6)
      call sym_vektor_tensor(CRv,CR)
c ==========================================================================
c     Calculate T2PK stresses after plastic deformation
c ==========================================================================
      call ellaw(CR,Pn1,T2PKtilde,CRtilde,npt,noel,nprops,props)
c      if(npt == 1 .and. noel == 1) then
c      write(*,*) "stress ende"
c      end if
      T2PK = matmul(matmul(Pn1,T2PKtilde),transpose(Pn1))
      end if
      end
```

C.3 Subroutine RESMATERIAL

```
c ==========================================================================
c ==========================================================================
c ==== Subroutine RESMATERIAL
c ==========================================================================
c ==========================================================================
      subroutine resmaterial(x,r,passin,npassin,size,npt,noel,
     1 nprops,props)
```

```
      implicit none
      integer size,npassin,i,j,npt,noel,nprops
      double precision x(size),r(size),passin(npassin),lambdacp,det
      double precision dfgrd1(3,3),Pn(3,3),T2PKtilde(3,3),Ftilde(3,3)
      double precision FTtilde(3,3),EGtilde(3,3),id(3,3),CRtilde(3,3)
      double precision emo,nu,mu,lambda,spurEGtilde,T(3,3),Jac,sigmav
      double precision Tdev(3,3),Tdevnorm,sigmaf,TC(3,3)
      double precision TCdev(3,3),TCdevexp(3,3),Pn1(3,3),Pexp(3,3)
      double precision TCdevnorm,Pn1i(3,3),Pn1Pn(3,3),LNPn1Pn(3,3)
      double precision CTdev(3,3),CTdevnorm,JC,CR(3,3),CRv(6)
      double precision props(nprops)
c      write(*,*) "resmaterial start"
c ================================================================
c     Write x in matrix Pn1
c ================================================================
      Pn1(1,1) = x(1)
      Pn1(1,2) = x(2)
      Pn1(1,3) = x(3)
      Pn1(2,1) = x(4)
      Pn1(2,2) = x(5)
      Pn1(2,3) = x(6)
      Pn1(3,1) = x(7)
      Pn1(3,2) = x(8)
      Pn1(3,3) = x(9)
      lambdacp = x(10)
c     CRv = passin(1..6)
      CRv(1) = passin(1)
      CRv(2) = passin(2)
      CRv(3) = passin(3)
      CRv(4) = passin(4)
      CRv(5) = passin(5)
      CRv(6) = passin(6)
      call sym_vektor_tensor(CRv,CR)
c      passin(10..18) = pn
      Pn(1,1) = passin(10)
      Pn(1,2) = passin(11)
      Pn(1,3) = passin(12)
      Pn(2,1) = passin(13)
      Pn(2,2) = passin(14)
      Pn(2,3) = passin(15)
      Pn(3,1) = passin(16)
      Pn(3,2) = passin(17)
      Pn(3,3) = passin(18)
c ================================================================
c      Call subroutine ellaw to calculate stresses
c ================================================================
      call ellaw(CR,Pn1,T2PKtilde,CRtilde,npt,noel,nprops,props)
c ================================================================
c      Build parts of yield function
c ================================================================
      JC = sqrt(det(CRtilde))
      call DEVIATOR(matmul(CRtilde,T2PKtilde),CTdev)
```

```
      call NORMT(CTdev,CTdevnorm)
      sigmav=dsqrt(3.d0/2.d0)/JC*CTdevnorm
      TC = matmul(T2PKtilde,CRtilde)
      call DEVIATOR(TC,TCdev)
      call invert3(Pn1,Pn1i)
      Pn1Pn = matmul(Pn1i,Pn)
      call LN(Pn1Pn,LNPn1Pn)
c      R=Pn-Pn1*TCdevexp
c      R=Pn-x*TCdevexp
c ===================================================================
c     Build final residuum of yield function
c ===================================================================
      sigmaf=props(3)
      r(1)=LNPn1Pn(1,1)-lambdacp*TCdev(1,1)
      r(2)=LNPn1Pn(1,2)-lambdacp*TCdev(1,2)
      r(3)=LNPn1Pn(1,3)-lambdacp*TCdev(1,3)
      r(4)=LNPn1Pn(2,1)-lambdacp*TCdev(2,1)
      r(5)=LNPn1Pn(2,2)-lambdacp*TCdev(2,2)
      r(6)=LNPn1Pn(2,3)-lambdacp*TCdev(2,3)
      r(7)=LNPn1Pn(3,1)-lambdacp*TCdev(3,1)
      r(8)=LNPn1Pn(3,2)-lambdacp*TCdev(3,2)
      r(9)=LNPn1Pn(3,3)-lambdacp*TCdev(3,3)
      r(10)=sigmav-sigmaf
c      write(*,*) "resmaterial ende"
      return
      end
```

C.4 Subroutine ELLAW

```
c ===================================================================
c ===================================================================
c ==== Subroutine ELLAW
c ===================================================================
c ===================================================================
      subroutine ellaw(CR,P,T2PKtilde,CRtilde,npt,noel,nprops,props)
      implicit none
      integer i,j,npt,noel,nprops
      double precision id(3,3),CR(3,3),P(3,3),PT(3,3),props(nprops)
      double precision Ftilde(3,3),T2PKtilde(3,3),T(3,3)
      double precision FTtilde(3,3),EGtilde(3,3),CRtilde(3,3)
      double precision emo,nu,mu,lambda,spurEGtilde,det,Jac
c      write(*,*) "ellaw start"
c ===================================================================
c     Read material parameters from Abaqus input
c ===================================================================
      emo = props(1)
      nu = props(2)
```

```
      mu = emo/(2.d0*(1.d0+nu))
      lambda = nu*emo/(1.d0 + nu)/(1.d0 - 2.d0*nu);
c ===================================================================
c     Identity
c ===================================================================
      id =0.D0
      id(1,1) = 1.D0
      id(2,2) = 1.D0
      id(3,3) = 1.D0
c ===================================================================
c      Calculate EGREEN from CR
c ===================================================================
      PT = transpose(P)
      CRtilde = matmul(matmul(PT,CR),P)
      EGtilde = 0.5d0*(CRtilde-id)
c ===================================================================
c      Calculate stresses T2PKtilde
c ===================================================================
      spurEGtilde = EGtilde(1,1)+EGtilde(2,2)+EGtilde(3,3)
      T2PKtilde = lambda*spurEGtilde*id + 2.d0*mu*EGtilde
c      write(*,*) "ellaw ende"
      end
```

C.5 Subroutine SOLVEN

```
c ===================================================================
c ===================================================================
c === Subroutine SOLVEN
c ===================================================================
c ===================================================================
      subroutine solven(x,size,passin,
     &    npassin,tol,maxit,damp,msglinesearch,msgjac,
     &    msgquasi,npt,noel,it,nprops,props)
      implicit none
c     Newton Solver
c     Arguments:
c     x(size) ... independent variables
c     residuumsfunktion ... name of the subroutine which provides the
c        residuum vector (in our case: resmaterial)
c     Form: residuumsfkt(x(size),residuum0(size),passin(npassin),size)
c     passin(npassin)
c     size ... INTEGER: Dimension of the system
c     npassin ... INTEGER: number of variables in resmaterial
c     tol ... end if tolerance > norm of residuum vector
c     maxit ... maximum number of iterations
c     damp(size) ... vector of damping values each for one equation of
c         the residuum equation (everything = 1 for standard Newton)
```

```
c     msglinesearch ... INTEGER: Linesearch parameter:
c        0 - no linesearch
c          n - n linesearch bisection
c     msgjac ... Jacobi matrix parameter:
c        INTEGER: determine Jacobian all n steps
c        1 for standard Newton,
c        else Broydens update algorithm is used
c     msgquasi ... INTEGER: quasi newton parameter:
c        < 0: no quasi newton iterations
c        0 - quasi newton iterations till norm new residuum is less than
c            10% better than old residuum
c        n - n quasi newton iterations
      integer size,error,npassin,maxit,it,msglinesearch,msgjac
      integer msgquasi,npt,noel,nprops
      double precision x(size),residuum0(size),tol,rest,rdummy
      double precision residuum1(size),dx(size)
      double precision jac(size,size),eps
      double precision passin(npassin),damp(size)
      double precision rdummyalt,xalt(size),n(size)
      double precision beta,alpha,residuumnachit(size)
      double precision xnachit(size),rdummynachit,alpha2
      double precision residuumalt(size),rdummynachit2
      double precision dr(size),hvek(size),xnachit2(size)
      double precision residuumnachit2(size),jacalt(size,size)
      double precision rdummyquasi,props(nprops)
      integer i,j,k,nachitz,nachitp,opjac
      integer indx(size),quasicount
      double precision a,b,c,fl1,fl2,fl3,fl4
      double precision ls(3,2),alphalinks,alpharechts
c      if(npt == 1 .and. noel == 1) then
c      write(*,*) "Start Solven"
c      write(*,*) "x="
c      call printvektor(x,size)
c      end if
c ===================================================================
c      Call subroutine resmaterial
c ===================================================================
      call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
     & nprops,props)
c ===================================================================
c      Call subroutine res
c ===================================================================
      call res(residuum0,size,rdummyalt)
      rdummy=rdummyalt
      eps=1.d-7
      it=0
      nachitz=0
      nachitp=0
      do while((rdummy.ge.tol).and.(it.lt.maxit))
     if(mod(it,msgjac).eq.0) then
     opjac=1
     else
```

```
        opjac=0
      end if
        it=it+1
c       JACOBIMATRIX
        if(opjac.eq.1) then
      do i=1,size
        x(i)=x(i)+eps
        call resmaterial(x,residuum1,passin,npassin,size,npt,noel,
     & nprops,props)
        x(i)=x(i)-eps
        do j=1,size
        jac(j,i)=(residuum1(j)-residuum0(j))/eps
        end do
        end do
c       BROYDONS BAD UPDATE
      else
      rdummy=0.d0
      do i=1,size
        dx(i)=x(i)-xalt(i)
        rdummy=rdummy+dx(i)**2.d0
      end do
      rdummy=1.d0/rdummy
      do i=1,size
      do j=1,size
        jac(i,j)=jacalt(i,j)+rdummy*residuum0(i)*dx(j)
      end do
      end do
        end if
c         write(*,*) "END JACOBI, 10x resmaterial"
        do i=1,size
      residuumalt(i)=residuum0(i)
        xalt(i)=x(i)
      do j=1,size
      jacalt(i,j)=jac(i,j)
      end do
      end do
c         write(*,*) "X ALT",xalt
c ======================================================================
c       Call subroutine solvelu
c ======================================================================
        call solvelu(size,jac,residuum0,dx,error,indx)
        do i=1,size
      x(i)=x(i)-damp(i)*dx(i)
        end do
c           write(*,*) "X NEU",x
      if(msgquasi.gt.0) then
        call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
     & nprops,props)
        call res(residuum0,size,rdummy)
      k=0
      write(*,*)rdummy
      do while((rdummy.gt.tol).and.(k.lt.msgquasi))
```

```
k=k+1
call LUBKSB(jac,size,size,indx,residuum0)
  do i=1,size
  x(i)=x(i)-damp(i)*residuum0(i)
  end do
   call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
 & nprops,props)
   call res(residuum0,size,rdummy)
write(*,*)rdummy
end do
write(*,*) x
end if
   if(msgquasi.eq.0) then
   call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
 & nprops,props)
call res(residuum0,size,rdummyquasi)
rdummy=rdummyquasi*0.8999d0
do while((rdummy.lt.(rdummyquasi*0.9d0)).and.(rdummy.gt.tol))
rdummyquasi=rdummy
call LUBKSB(jac,size,size,indx,residuum0)
  do i=1,size
  x(i)=x(i)-damp(i)*residuum0(i)
  end do
   call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
 & nprops,props)
call res(residuum0,size,rdummy)
end do
end if
   if(msglinesearch.gt.0) then
write(*,*)"=================================================="
do i=1,size
dx(i)=x(i)-xalt(i)
end do
   call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
 & nprops,props)
call res(residuum0,size,rdummy)
if(rdummy.lt.rdummyalt) then
ls(1,1)=0.d0
ls(1,2)=rdummyalt
ls(2,1)=1.d0
ls(2,2)=rdummy
alpha=1.d0
rdummyalt=0.d0
do while(rdummyalt.lt.rdummy)
alpha=alpha+1.d0
do i=1,size
x(i)=xalt(i)+alpha*dx(i)
end do
   call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
 & nprops,props)
call res(residuum0,size,rdummyalt)
end do
```

```
ls(3,1)=alpha
ls(3,2)=rdummyalt
else
ls(2,1)=0.d0
ls(2,2)=rdummyalt
ls(3,1)=1.d0
ls(3,2)=rdummy
alpha=0.d0
rdummy=0.d0
do while(rdummy.lt.rdummyalt)
alpha=alpha-1.d0
do i=1,size
x(i)=xalt(i)+alpha*dx(i)
end do
   call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
 & nprops,props)
call res(residuum0,size,rdummy)
end do
ls(1,1)=alpha
ls(1,2)=rdummy
end if
do j=1,msglinesearch
write(*,'(3F20.10)') ls(1,1),ls(2,1),ls(3,1)
write(*,'(3F20.10)') ls(1,2),ls(2,2),ls(3,2)
   write(*,*)"=================================================="
alphalinks=0.5d0*(ls(1,1)+ls(2,1))
   do i=1,size
x(i)=xalt(i)+alphalinks*dx(i)
end do
   call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
 & nprops,props)
call res(residuum0,size,rdummy)
if (rdummy.gt.ls(2,2)) then
ls(1,1)=alphalinks
ls(1,2)=rdummy
else
ls(3,1)=ls(2,1)
ls(3,2)=ls(2,2)
ls(2,1)=alphalinks
ls(2,2)=rdummy
end if
   alpharechts=0.5d0*(ls(2,1)+ls(3,1))
   do i=1,size
x(i)=xalt(i)+alpharechts*dx(i)
end do
   call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
 & nprops,props)
call res(residuum0,size,rdummy)
if (rdummy.gt.ls(2,2)) then
ls(3,1)=alpharechts
ls(3,2)=rdummy
else
```

```
      ls(1,1)=ls(2,1)
      ls(1,2)=ls(2,2)
      ls(2,1)=alpharechts
      ls(2,2)=rdummy
      end if
      end do
      if(ls(2,1).eq.0.d0) ls(2,1)=1.d0
        do i=1,size
        dx(i)=dx(i)*ls(2,1)
      x(i)=xalt(i)+dx(i)
      end do
      end if
        call resmaterial(x,residuum0,passin,npassin,size,npt,noel,
     & nprops,props)
c       write(*,*)x
c       write(*,*)residuum0
        call res(residuum0,size,rdummy)
        rdummyalt=rdummy
c       write(*,*) it,"te Iteration"
        end do
c       if(npt == 1 .and. noel == 1) then
c       write(*,*) "Ende Solven"
c       write(*,*) "Das waren ",it," Newtonschritte!"
c       write(*,*) "x="
c       call printvektor(x,size)
c       write(*,*) "passin="
c       call printvektor(passin,size)
c       write(*,*) "residuum0="
c       call printvektor(residuum0,size)
c       end if
        return
        end
```

C.6 Subroutine RES

```
c ==========================================================================
c ==========================================================================
c ==== Subroutine RES
c ==========================================================================
c ==========================================================================
        subroutine res(r,size,erg)
        implicit none
        integer size,i
        double precision r(size),erg
        erg=0.d0
        do i=1,size
          if(dabs(r(i)).gt.erg) erg=dabs(r(i))
```

```
      end do
c       erg=0.d0
c       do i=1,size
c         erg=erg+r(i)**2.d0
c       end do
c       erg=dsqrt(erg)
      return
      end
```

C.7 Subroutine SOLVELU

```
c =======================================================================
c =======================================================================
c ==== Subroutine SOLVELU
c =======================================================================
c =======================================================================
      subroutine solvelu(size,A,b,dx,error,indx)
      implicit none
      integer size,error,i
      double precision A(size,size),b(size),dx(size),d
      integer indx(size),np
      do i=1,size
        dx(i)=b(i)
      end do
c =======================================================================
c       Call subroutines LUDCMP and LUBKSB
c =======================================================================
c     J in, decomposed J out
      call LUDCMP(A,size,size,indx,d)
c     dx in(right, dx (solution)out
      call LUBKSB(A,size,size,indx,dx)
      return
      end
```

C.8 Subroutine LUDCMP

```
c =======================================================================
c =======================================================================
c ==== Subroutine LUDCMP
c =======================================================================
c =======================================================================
      subroutine LUDCMP(a,n,np,indx,d)
```

```
      implicit double precision (a-h,o-z)
c
c Given an NxN matrix A, with dimension NP, this routine replaces it by
c the LU decomposition ofa rowwise permutation of itself. A and N are
c input. A is the output, arranged in equation 2.3.14 on p. 34. INDX
    is
c an output vector which records the row permutation resulting from
c partial pivoting. D is output as +/- 1 depending on whether the
    number
c of row interchanges was even or odd, respectively. Used in
    combination
c with LUBKSB to solve linear equations or invert a matrix.
c
c Numerical Recipes, p. 38-9.
c
      INTEGER n,np,indx(n),NMAX,i,imax,j,k
      DOUBLE PRECISION d,a(np,np),TINY
      PARAMETER(nmax=500, tiny=1.d-40)
      DOUBLE PRECISION aamax,dum,sum,vv(NMAX)
      d=1.0
      do 12 i=1,n
        aamax=0.d0
        do 11 j=1,n
          if (abs(a(i,j)).gt.aamax) aamax=abs(a(i,j))
11      continue
        if (aamax.eq.0.d0) PAUSE ' Singular matrix in LUDCMP'
        vv(i)=1.d0/aamax
12    continue
      do 19 j=1,n
        do 14 i=1,j-1
          sum=a(i,j)
          do 13 k=1,i-1
            sum=sum-a(i,k)*a(k,j)
13        continue
          a(i,j)=sum
14      continue
        aamax=0.d0
        do 16 i=j,n
          sum=a(i,j)
          do 15 k=1,j-1
            sum=sum-a(i,k)*a(k,j)
15        continue
          a(i,j)=sum
          dum=vv(i)*abs(sum)
          if (dum.ge.aamax) then
            imax=i
            aamax=dum
          endif
16      continue
        if (j.ne.imax) then
          do 17 k=1,n
            dum=a(imax,k)
```

```
              a(imax,k)=a(j,k)
              a(j,k)=dum
17          continue
            d=-d
            vv(imax)=vv(j)
          endif
          indx(j)=imax
          if (a(j,j).eq.0.d0) a(j,j)=tiny
          if (j.ne.n) then
            dum=1.d0/a(j,j)
            do 18 i=j+1,n
              a(i,j)=a(i,j)*dum
18          continue
          endif
19      continue
        return
        end
```

C.9 Subroutine LUBKSB

```
c ==========================================================================
c ==========================================================================
c ==== Subroutine LUBKSB
c ==========================================================================
c ==========================================================================
        subroutine LUBKSB(a,n,np,indx,b)
        implicit double precision (a-h,o-z)
c
c Solves the set of N linear equations A.X=B. Here A is input, not as
c the matrix A but rather its LU decomposition, determined by the
    routine
c LUDCMP. INDX is input as the permutation vector returned by LUDCMP.
    B
c is input as teh right hand side vector B and returns with the
    solution
c vector X. A, N, NP, and INDX are not modified by this routine and
    can
c be left in place for successive calls with different right-hand sides
    B.
c This routie takes into account the possibility that B will begin with
c many zero elements, so it is efficient for use in matrix inversion.
c
c Numerical Recipes, p. 39
c
        INTEGER n,np,indx(n)
        DOUBLE PRECISION a(np,np),b(n)
        INTEGER i,ii,j,ll
```

```
      DOUBLE PRECISION sum
       ii=0
      do 12 i=1,n
         ll=indx(i)
        sum=b(ll)
        b(ll)=b(i)
        if (ii.ne.0) then
          do 11 j=ii,i-1
            sum=sum-a(i,j)*b(j)
11        continue
        else if (sum.ne.0.d0) then
          ii=i
        endif
        b(i)=sum
12    continue
      do 14 i=n,1,-1
        sum=b(i)
        do 13 j=i+1,n
          sum=sum-a(i,j)*b(j)
13      continue
        b(i)=sum/a(i,i)
14    continue
      return
      end
```

C.10 Subroutine SDVINI

```
c ===================================================================
c ===================================================================
c = Subroutine SDVINI to set the initial values DEPVARS
c ===================================================================
c ===================================================================
      SUBROUTINE SDVINI(STATEV,COORDS,NSTATV,
     & NCRDS,NOEL,NPT,LAYER,KSPT)
      IMPLICIT NONE
c     Definitions ABAQUS
      INTEGER nstatv,ncrds,kspt,layer,npt,noel
      DOUBLE PRECISION statev(nstatv),coords(ncrds)
c     Own definitions
c      write(*,*) "SDVINI"
c     P initial value
      statev(1) = 1.d0
      statev(2) = 0.d0
      statev(3) = 0.d0
      statev(4) = 0.d0
      statev(5) = 1.d0
      statev(6) = 0.d0
```

```
      statev(7) = 0.d0
      statev(8) = 0.d0
      statev(9) = 1.d0
c      lambda initial value
      statev(10) = 0.d0
c      call printvektor(statev,10)
      RETURN
      END
```

Appendix D

Fortran utility subroutines

D.1 Subroutine VEKTOR_TENSOR

```
c ==========================================================================
c ==========================================================================
c ==== Subroutine VEKTOR_TENSOR
c ==== Input: 9x1 Vector
c ==== Output: Tensor 3x3
c ==========================================================================
c ==========================================================================
      SUBROUTINE VEKTOR_TENSOR(v,T)
      implicit none
      integer i
      double precision v(9),T(3,3)
      T(1,1) = v(1)
      T(1,2) = v(2)
      T(1,3) = v(3)
      T(2,1) = v(4)
      T(2,2) = v(5)
      T(2,3) = v(6)
      T(3,1) = v(7)
      T(3,2) = v(8)
      T(3,3) = v(9)
      return
      END SUBROUTINE VEKTOR_TENSOR
```

D.2 Subroutine TENSOR_VEKTOR

```
c ==========================================================================
c ==========================================================================
c ==== TENSOR_VEKTOR
c ==== Input: Tensor 3x3
```

```
c ==== Output: 9x1 Vector
c ================================================================
c ================================================================
      SUBROUTINE TENSOR_VEKTOR(T,v)
      implicit none
      integer i
      double precision v(9),T(3,3)
      v(1) = T(1,1)
      v(2) = T(1,2)
      v(3) = T(1,3)
      v(4) = T(2,1)
      v(5) = T(2,2)
      v(6) = T(2,3)
      v(7) = T(3,1)
      v(8) = T(3,2)
      v(9) = T(3,3)
      return
      END SUBROUTINE TENSOR_VEKTOR
```

D.3 Subroutine SYM_VEKTOR_TENSOR

```
c ================================================================
c ================================================================
c ==== Subroutine SYM_VEKTOR_TENSOR
c ==== Input: 6x1 Vector
c ==== Output: Symmetric Tensor 3x3
c ================================================================
c ================================================================
      SUBROUTINE SYM_VEKTOR_TENSOR(v,T)
      implicit none
      integer i
      double precision v(6),T(3,3)
      T(1,1) = v(1)
      T(2,2) = v(2)
      T(3,3) = v(3)
      T(1,2) = v(4)
      T(2,1) = v(4)
      T(1,3) = v(5)
      T(3,1) = v(5)
      T(2,3) = v(6)
      T(3,2) = v(6)
      return
      END SUBROUTINE SYM_VEKTOR_TENSOR
```

D.4 Subroutine SYM_TENSOR_VEKTOR

```
c ==========================================================
c ==========================================================
c ==== Subroutine SYM_TENSOR_VEKTOR
c ==== Input: Symmetric Tensor 3x3
c ==== Output: 6x1 Vector
c ==========================================================
c ==========================================================
      SUBROUTINE SYM_TENSOR_VEKTOR(T,v)
      implicit none
      integer i
      double precision v(6),T(3,3)
      v(1) = T(1,1)
      v(2) = T(2,2)
      v(3) = T(3,3)
      v(4) = T(1,2)
      v(5) = T(1,3)
      v(6) = T(2,3)
      return
      END SUBROUTINE SYM_TENSOR_VEKTOR
```

D.5 Subroutine PRINTMATRIX

```
c ==========================================================
c ==========================================================
c = Subroutine PRINTMATRIX
c = Input: m matrix, sm dimension
c = Output: screen print
c ==========================================================
c ==========================================================
      SUBROUTINE PRINTMATRIX(m,sm)
      IMPLICIT NONE
      INTEGER i,j,sm
      DOUBLE PRECISION m(sm,sm)
      DO i=1,sm
        PRINT 113,(m(i,j),j=1,sm)
      END DO
      PRINT*
 113  FORMAT(30F30.18)
      RETURN
      END
```

D.6 Subroutine PRINTMATRIX2

```
c ==========================================================
c ==========================================================
c = Subroutine PRINTMATRIX2
c = Input: m matrix, sm dimension
c = Output: screen print
c ==========================================================
c ==========================================================
      subroutine printmatrix2(m,sm)
c     Matrix m output on screen
      implicit none
      integer i,j,sm
      double precision m(sm,sm)
      do i=1,sm
        print 113,(m(i,j),j=1,sm)
      end do
      print*
 113  format(10f16.9)
      return
      end
```

D.7 Subroutine PRINTVEKTOR

```
c ==========================================================
c ==========================================================
c === Subroutine PRINTVEKTOR
c === Input: vector v, dimension n
c === Output: screen print
c ==========================================================
c ==========================================================
      SUBROUTINE PRINTVEKTOR(v,n)
      implicit none
      integer i, n
      double precision v(n)
      do i=1,n
          write(*,*) v(i)
      end do
      END SUBROUTINE PRINTVEKTOR
```

D.8 Subroutine DEVIATOR

```
c ==========================================================================
c ==========================================================================
c ==== SUBROUTINE DEVIATOR
c ==== Input: T 3x3 matrix
c ==== Output: Tdev 3x3 deviatoric part of matrix T
c ==========================================================================
c ==========================================================================
      SUBROUTINE DEVIATOR(T,Tdev)
      implicit none
      integer i,j
      double precision T(3,3), Tdev(3,3),spur
      spur = 0.D0
      do i=1,3
          spur = spur + T(i,i)
      end do
      do i=1,3
      do j=1,3
          if(i==j) then
              Tdev(i,j) = T(i,j) - 1.D0/3.D0*spur
          else
              Tdev(i,j) = T(i,j)
          end if
      end do
      end do
      return
      END SUBROUTINE DEVIATOR
```

D.9 Subroutine NORMT

```
c ==========================================================================
c ==========================================================================
c ==== SUBROUTINE NORMT
c ==== Input: T 3x3 matrix
c ==== Output: norm of T as scalar value
c ==========================================================================
c ==========================================================================
      SUBROUTINE NORMT(T,norm)
      implicit none
      integer i,j
      double precision T(3,3), norm
      norm = 0.D0
      do i=1,3
      do j=1,3
          norm = norm + T(i,j)*T(i,j)
      end do
      end do
      norm = dsqrt(norm)
```

```
      return
      END SUBROUTINE NORMT
```

D.10 Subroutine MEXP

```
c ==========================================================================
c ==========================================================================
c ==== SUBROUTINE MEXP
c ==== Input: m1 3x3 matrix
c ==== Output: msum as exponent of m1
c ==========================================================================
c ==========================================================================
      subroutine mexp(m1,msum)
      implicit none
      double precision m1(3,3),m2(3,3),msum(3,3),m3(3,3),rdummy
      integer count
      double precision fak
      msum(1,1)=1.d0+m1(1,1)
      msum(1,2)=0.d0+m1(1,2)
      msum(1,3)=0.d0+m1(1,3)
      msum(2,1)=0.d0+m1(2,1)
      msum(2,2)=1.d0+m1(2,2)
      msum(2,3)=0.d0+m1(2,3)
      msum(3,1)=0.d0+m1(3,1)
      msum(3,2)=0.d0+m1(3,2)
      msum(3,3)=1.d0+m1(3,3)
      m2(1,1)=m1(1,1)
      m2(1,2)=m1(1,2)
      m2(1,3)=m1(1,3)
      m2(2,1)=m1(2,1)
      m2(2,2)=m1(2,2)
      m2(2,3)=m1(2,3)
      m2(3,1)=m1(3,1)
      m2(3,2)=m1(3,2)
      m2(3,3)=m1(3,3)
      count=1
      fak=1.d0
      rdummy=1.d0
      do while (rdummy.gt.1.d-16)
      count=count+1
      fak=fak*count
      call multiply233f1(m2,m1,m3)
      msum(1,1)=msum(1,1)+m3(1,1)/fak
      msum(1,2)=msum(1,2)+m3(1,2)/fak
      msum(1,3)=msum(1,3)+m3(1,3)/fak
      msum(2,1)=msum(2,1)+m3(2,1)/fak
      msum(2,2)=msum(2,2)+m3(2,2)/fak
```

```
      msum(2,3)=msum(2,3)+m3(2,3)/fak
      msum(3,1)=msum(3,1)+m3(3,1)/fak
      msum(3,2)=msum(3,2)+m3(3,2)/fak
      msum(3,3)=msum(3,3)+m3(3,3)/fak
      m2(1,1)=m3(1,1)
      m2(1,2)=m3(1,2)
      m2(1,3)=m3(1,3)
      m2(2,1)=m3(2,1)
      m2(2,2)=m3(2,2)
      m2(2,3)=m3(2,3)
      m2(3,1)=m3(3,1)
      m2(3,2)=m3(3,2)
      m2(3,3)=m3(3,3)
      call norm33(m2,rdummy)
      rdummy=rdummy/fak
c       write(*,*)rdummy,count
      end do
      return
      end
```

D.11 Subroutine MULTIPLY233F1

```
c ===========================================================================
c ===========================================================================
c ==== SUBROUTINE MULTIPLY233F1 product: m3 = m1 * m2
c ==== Input: m1,m2 as 3x3 matrices
c ==== Output: m3 as 3x3 matrix
c ===========================================================================
c ===========================================================================
      subroutine multiply233f1(m1,m2,m3)
      implicit none
      double precision m1(3,3),m2(3,3),m3(3,3)
      m3(1,1) = (m1(1,1)*m2(1,1)+m1(1,2)*m2(2,1)+m1(1,3)*m2(3,1))
      m3(1,2) = (m1(1,1)*m2(1,2)+m1(1,2)*m2(2,2)+m1(1,3)*m2(3,2))
      m3(1,3) = (m1(1,1)*m2(1,3)+m1(1,2)*m2(2,3)+m1(1,3)*m2(3,3))
      m3(2,1) = (m1(2,1)*m2(1,1)+m1(2,2)*m2(2,1)+m1(2,3)*m2(3,1))
      m3(2,2) = (m1(2,1)*m2(1,2)+m1(2,2)*m2(2,2)+m1(2,3)*m2(3,2))
      m3(2,3) = (m1(2,1)*m2(1,3)+m1(2,2)*m2(2,3)+m1(2,3)*m2(3,3))
      m3(3,1) = (m1(3,1)*m2(1,1)+m1(3,2)*m2(2,1)+m1(3,3)*m2(3,1))
      m3(3,2) = (m1(3,1)*m2(1,2)+m1(3,2)*m2(2,2)+m1(3,3)*m2(3,2))
      m3(3,3) = (m1(3,1)*m2(1,3)+m1(3,2)*m2(2,3)+m1(3,3)*m2(3,3))
      return
      end
```

D.12 Subroutine NORM33

```
c ==========================================================================
c ==========================================================================
c ==== SUBROUTINE NORM33
c ==== Input: m1 as 3x3 matrix
c ==== Output: norm as scalar value
c ==========================================================================
c ==========================================================================
      subroutine norm33(m1,norm)
      implicit none
      double precision m1(3,3),norm
      norm=m1(1,1)*m1(1,1)+
     &     m1(1,2)*m1(1,2)+
     &     m1(1,3)*m1(1,3)+
     &     m1(2,1)*m1(2,1)+
     &     m1(2,2)*m1(2,2)+
     &     m1(2,3)*m1(2,3)+
     &     m1(3,1)*m1(3,1)+
     &     m1(3,2)*m1(3,2)+
     &     m1(3,3)*m1(3,3)
      norm=dsqrt(norm)
      return
      end
```

D.13 Subroutine LN

```
c ==========================================================================
c ==========================================================================
c ==== SUBROUTINE LN calculates ln of a 3x3 matrix mi with the series
c ==== ln(X)= 2 * sum(k=0..unend.) 1/(2k+1) * ((X–I)(X+I)^-1)^(2k+1)
c ==== Input: mi as 3x3 matrix
c ==== Output: mo as 3x3 matrix
c ==========================================================================
c ==========================================================================
      subroutine LN(mi,mo)
      implicit none
      double precision mi(3,3),mo(3,3),faktor,h1(3,3),h2(3,3),h3(3,3)
      double precision h4(3,3),h5(3,3),h6(3,3),h7(3,3),rdummy,h8(3,3)
      integer i,j,k
      double precision restnorm
      do i=1,3
      do j=1,3
         h1(i,j)=mi(i,j)
         h2(i,j)=mi(i,j)
         if(i.eq.j) then
            h1(i,j)=mi(i,j)-1.d0
```

```
            h2(i,j)=mi(i,j)+1.d0
          end if
        end do
        end do
        call invert3(h2,h3)
        call multiply233f1(h1,h3,h4)
        do i=1,3
        do j=1,3
          mo(i,j)=h4(i,j)
          h5(i,j)=h4(i,j)
          h6(i,j)=h4(i,j)
        end do
        end do
        restnorm=1.d0
        call multiply233f1(h5,h6,h7)
        k=0
        do while ((restnorm.gt.1.d-16).and.(k.lt.10000))
          k=k+1
          rdummy=2.d0*k+1.d0
          call multiply233f1(h4,h7,h8)
          restnorm=0.d0
        do i=1,3
        do j=1,3
          restnorm=restnorm+(h8(i,j)/rdummy)**2.d0
          mo(i,j)=mo(i,j)+h8(i,j)/rdummy
          h4(i,j)=h8(i,j)
        end do
        end do
          restnorm=dsqrt(restnorm)
        end do
c        write(*,*) k,restnorm
        do i=1,3
        do j=1,3
        mo(i,j)=mo(i,j)*2.d0
        end do
        end do
        return
        end
```

D.14 Subroutine INVERT3

```
c ==========================================================
c ==========================================================
c ==== SUBROUTINE INVERT3
c ==== Input: m as 3x3 matrix
c ==== Output: mi as 3x3 matrix being the inverse of m
c ==========================================================
```

```
c ==========================================================================
      subroutine invert3(m,mi)
      implicit none
      double precision m(3,3),mi(3,3),hi3m
      call det3(m,hi3m)
      if(abs(hi3m).gt.1.d-12)then
        mi(1,1)=(m(2,2)*m(3,3)-m(3,2)*m(2,3))/hi3m
        mi(2,1)=(m(3,1)*m(2,3)-m(2,1)*m(3,3))/hi3m
        mi(3,1)=(m(2,1)*m(3,2)-m(3,1)*m(2,2))/hi3m
        mi(1,2)=(m(3,2)*m(1,3)-m(1,2)*m(3,3))/hi3m
        mi(2,2)=(m(1,1)*m(3,3)-m(3,1)*m(1,3))/hi3m
        mi(3,2)=(m(3,1)*m(1,2)-m(1,1)*m(3,2))/hi3m
        mi(1,3)=(m(1,2)*m(2,3)-m(2,2)*m(1,3))/hi3m
        mi(2,3)=(m(2,1)*m(1,3)-m(1,1)*m(2,3))/hi3m
        mi(3,3)=(m(1,1)*m(2,2)-m(2,1)*m(1,2))/hi3m
      end if
      return
      end
```

D.15 Subroutine DET3

```
c ==========================================================================
c ==========================================================================
c ==== SUBROUTINE DET3
c ==== Input: m33 as 3x3 matrix
c ==== Output: det as scalar value
c ==========================================================================
c ==========================================================================
      subroutine det3(m33,det)
      implicit none
      double precision m33(3,3),det
      det=m33(1,1)*m33(2,2)*m33(3,3)
      det=det+m33(1,2)*m33(2,3)*m33(3,1)
      det=det+m33(1,3)*m33(2,1)*m33(3,2)
      det=det-m33(3,1)*m33(2,2)*m33(1,3)
      det=det-m33(3,2)*m33(2,3)*m33(1,1)
      det=det-m33(3,3)*m33(2,1)*m33(1,2)
      return
      end
```

D.16 Subroutine SHRINK333266

```
c ==========================================================================
c ==========================================================================
c ==== Subroutine SHRINK333266 Tetrade (3,3,3,3) in (6,6) Notation
c ==== Input: m as 3x3x3x3 fourth order tensor
c ==== Output: m66 as 6x6 matrix in voigt notation
c ==========================================================================
c ==========================================================================
      subroutine shrink3333266(m,m66)
c     only subsymmetries implied
      implicit none
      double precision m66(6,6),m(3,3,3,3),rd
      integer i1(6),i2(6),i,j
      i1(1)=1
      i1(2)=2
      i1(3)=3
      i1(4)=1
      i1(5)=1
      i1(6)=2
      i2(1)=1
      i2(2)=2
      i2(3)=3
      i2(4)=2
      i2(5)=3
      i2(6)=3
      do i=1,6
      do j=1,6
      m66(i,j)=(
     & m(i1(i),i2(i),i1(j),i2(j))+
     & m(i2(i),i1(i),i1(j),i2(j))+
     & m(i1(i),i2(i),i2(j),i1(j))+
     & m(i2(i),i1(i),i2(j),i1(j))
     &          )/4.d0
      end do
      end do
      rd=dsqrt(2.d0)
      m66(1,4)=m66(1,4)*rd
      m66(1,5)=m66(1,5)*rd
      m66(1,6)=m66(1,6)*rd
      m66(2,4)=m66(2,4)*rd
      m66(2,5)=m66(2,5)*rd
      m66(2,6)=m66(2,6)*rd
      m66(3,4)=m66(3,4)*rd
      m66(3,5)=m66(3,5)*rd
      m66(3,6)=m66(3,6)*rd
      m66(4,1)=m66(4,1)*rd
      m66(5,1)=m66(5,1)*rd
      m66(6,1)=m66(6,1)*rd
      m66(4,2)=m66(4,2)*rd
      m66(5,2)=m66(5,2)*rd
      m66(6,2)=m66(6,2)*rd
      m66(4,3)=m66(4,3)*rd
      m66(5,3)=m66(5,3)*rd
```

```
      m66(6,3)=m66(6,3)*rd
      m66(4,4)=m66(4,4)*2.d0
      m66(4,5)=m66(4,5)*2.d0
      m66(4,6)=m66(4,6)*2.d0
      m66(5,4)=m66(5,4)*2.d0
      m66(5,5)=m66(5,5)*2.d0
      m66(5,6)=m66(5,6)*2.d0
      m66(6,4)=m66(6,4)*2.d0
      m66(6,5)=m66(6,5)*2.d0
      m66(6,6)=m66(6,6)*2.d0
      return
      end
```

D.17 Subroutine M3TO6_COWIN

```
c ==========================================================
c ==========================================================
c ==== Subroutine M3TO6_COWIN to cowin with nomalized base
c ==== Input: m3 as 3x3 matrix
c ==== Output: m6 as 6x6 matrix
c ==========================================================
c ==========================================================
      subroutine m3to6_cowin(m3,m6)
      implicit none
      double precision m3(3,3),m6(6,6),rd
      integer i,j
      rd=dsqrt(2.d0)
      do i=1,3
      do j=1,3
      m6(i,j)=m3(i,j)**2.d0
      end do
      end do
      m6(1,4)=rd*m3(1,1)*m3(1,2)
      m6(2,4)=rd*m3(2,1)*m3(2,2)
      m6(3,4)=rd*m3(3,1)*m3(3,2)
      m6(1,5)=rd*m3(1,1)*m3(1,3)
      m6(2,5)=rd*m3(2,1)*m3(2,3)
      m6(3,5)=rd*m3(3,1)*m3(3,3)
      m6(1,6)=rd*m3(1,2)*m3(1,3)
      m6(2,6)=rd*m3(2,2)*m3(2,3)
      m6(3,6)=rd*m3(3,2)*m3(3,3)
      m6(4,1)=rd*m3(1,1)*m3(2,1)
      m6(4,2)=rd*m3(1,2)*m3(2,2)
      m6(4,3)=rd*m3(1,3)*m3(2,3)
      m6(5,1)=rd*m3(1,1)*m3(3,1)
      m6(5,2)=rd*m3(1,2)*m3(3,2)
      m6(5,3)=rd*m3(1,3)*m3(3,3)
```

```
      m6(6,1)=rd*m3(2,1)*m3(3,1)
      m6(6,2)=rd*m3(2,2)*m3(3,2)
      m6(6,3)=rd*m3(2,3)*m3(3,3)
      m6(4,4)=m3(1,1)*m3(2,2)+m3(1,2)*m3(2,1)
      m6(4,5)=m3(1,1)*m3(2,3)+m3(1,3)*m3(2,1)
      m6(4,6)=m3(1,2)*m3(2,3)+m3(1,3)*m3(2,2)
      m6(5,4)=m3(1,1)*m3(3,2)+m3(1,2)*m3(3,1)
      m6(5,5)=m3(1,1)*m3(3,3)+m3(1,3)*m3(3,1)
      m6(5,6)=m3(1,2)*m3(3,3)+m3(1,3)*m3(3,2)
      m6(6,4)=m3(2,1)*m3(3,2)+m3(2,2)*m3(3,1)
      m6(6,5)=m3(2,1)*m3(3,3)+m3(2,3)*m3(3,1)
      m6(6,6)=m3(2,2)*m3(3,3)+m3(2,3)*m3(3,2)
      return
      end
```

D.18 Function DET

```
c ==========================================================================
c ==========================================================================
c ==== FUNCTION DET Determinant of a 3x3 Matrix
c ==== Input: matrix m
c ==== Output: scalar value of the determinant
c ==========================================================================
c ==========================================================================
      function det(m)
      IMPLICIT NONE
      double precision det, m(3,3)
      det=+m(1,1)*m(2,2)*m(3,3)
     &    +m(1,2)*m(2,3)*m(3,1)
     &    +m(1,3)*m(2,1)*m(3,2)
     &    -m(3,1)*m(2,2)*m(1,3)
     &    -m(3,2)*m(2,3)*m(1,1)
     &    -m(3,3)*m(2,1)*m(1,2)
      return
      end
```

Appendix E

Example output files

The example files are from the calculation of the x-y-shear of the uni-directionally reinforced material.

E.1 Outfile K0_ANLAUFRECHNUNG.rpt

```
U:U1 PI: FIBBI-1   U:U2 PI: FIBBI-1   U:U3 PI: FIBBI-1   U:U1 PI: FIBBI
    -1   U:U2 PI: FIBBI-1   U:U3 PI: FIBBI-1   U:U1 PI: FIBBI-1   U:U2
    PI: FIBBI-1   U:U3 PI: FIBBI-1\\

X N: 2638 N: 2638 N: 2638 N: 2639 N: 2639 N: 2639 N: 2640 N: 2640 N:
    2640

0.0. 0. 0. 0. 0. 0. 0. 0. 0.

100.E-03 -0. 500.E-06 -0. -1.84155E-36 -0. 0. -0. 0. -0.

200.E-03 -0. 1.E-03 -0. -3.68311E-36 -12.2265E-39 0. -0. 0. -0.

                    .
                    .
                    .

200. -1.05954E-03 500.975E-03 -6.67E-09 -9.0195E-03 -3.25134E-03 75.946
    E-09 7.33338E-09 -72.3482E-09 59.6508E-06
```

E.2 Outfile K0_ANLAUFRECHNUNG_H0.txt

```
0.000000, 0.500000, 0.000000
0.000000, 0.000000, 0.000000
```

```
0.000000, 0.000000, 0.000000
```

E.3 Outfile K0_ANLAUFRECHNUNG_H1.txt

```
-1.070704e-03,5.005839e-01,5.720507e-08
-8.625158e-03,-3.066690e-03,6.329396e-08
-5.659331e-08,-9.172872e-08,7.050904e-05
```

E.4 Outfile K0_ANLAUFRECHNUNG_T1.txt

```
3.822745e-04,1.910864e-04,-4.442093e-06
1.889818e-04,-5.147681e-04,-4.607616e-06
-4.403416e-06,-4.574429e-06,1.927272e-04
```

E.5 Outfile K0_matheEGREEN_einzeln.txt

```
9.989298e-07,0.000000e+00,0.000000e+00,
2.502919e-07,2.860254e-14,0.000000e+00,

0.000000e+00,9.969338e-07,0.000000e+00,
-4.312579e-09,0.000000e+00,3.164698e-14,

0.000000e+00,0.000000e+00,1.000071e-06,
0.000000e+00,-2.829666e-14,-4.586436e-14,

-8.624658e-09,5.005844e-07,0.000000e+00,
9.979313e-07,3.164698e-14,2.860254e-14,

4.434231e-13,0.000000e+00,5.572209e-13,
-4.587997e-14,9.994999e-07,2.502919e-07,

0.000000e+00,4.082290e-13,5.632161e-13,
-2.831069e-14,-4.312579e-09,9.985019e-07,
```

E.6 Outfile K0_matheT2PK_einzeln.txt

```
4.215586e-02,1.195762e-02,1.417150e-02,
-9.914443e-03,3.929877e-08,3.309542e-08,

2.240174e-02,4.021668e-02,1.560775e-02,
-2.075727e-02,3.711311e-08,3.647988e-08,

1.808729e-02,1.558635e-02,7.347116e-02,
-7.824693e-03,3.808867e-08,3.311851e-08,

-2.880589e-02,-2.130365e-02,-7.949692e-03,
2.920955e-02,4.023180e-08,3.172748e-08,

-1.299025e-05,-8.807129e-06,-1.412913e-06,
5.136588e-06,2.466441e-02,-6.220451e-03,

-1.295309e-05,-8.849800e-06,-1.422639e-06,
5.140203e-06,-1.259238e-02,2.504515e-02,
```

Appendix F

Mathematica post-processing

F.1 Material Plasticity

```
Remove["Global`*"];
(*********************************************************************)
(*Subroutines*)
(*********************************************************************)
SetDirectory[NotebookDirectory[]];
Translate66to3333[M66_]:=(Indexmatrix={{1,4,5},{4,2,6},{5,6,3}};
Faktormatrix={{1,1/Sqrt[2],1/Sqrt[2]},{1/Sqrt[2],1,1/Sqrt[2]},{1/Sqrt
    [2],1/Sqrt[2],1}};
Table[M66[[Indexmatrix[[i,j]],Indexmatrix[[k,l]]]]*Faktormatrix[[i,j]]*
    Faktormatrix[[k,l]],{i,1,3},{j,1,3},{k,1,3},{l,1,3}]);
Translate3333to66[C3333_]:=(Ivek1={1,2,3,1,1,2};
Ivek2={1,2,3,2,3,3};
Faktvek={1,1,1,Sqrt[2],Sqrt[2],Sqrt[2]};
Table[C3333[[Ivek1[[i]],Ivek2[[i]],Ivek2[[j]],Ivek1[[j]]]]*Faktvek[[i
    ]]*Faktvek[[j]],{i,1,6},{j,1,6}]);
Rayleigh[A33_,B3333_]:=
(
C3333=Table[0,{i,3},{j,3},{k,3},{l,3}] ;
For[i=1,i<=3,i++,
For[j=1,j<=3,j++,
For[k=1,k<=3,k++,
For[l=1,l<=3,l++,
C3333[[i,j,k,l]]=0;
For[m=1,m<=3,m++,
For[n=1,n<=3,n++,
For[o=1,o<=3,o++,
For[p=1,p<=3,p++,
C3333[[i,j,k,l]]=C3333[[i,j,k,l]]+B3333[[m,n,o,p]]*A33[[i,m]]*A33[[j,n
    ]]*A33[[k,o]]*A33[[l,p]]
];
];
];
];
];
```

```
];
];
];
C3333
)
(*********************************************************************)
(*Read ABAQUS example output - uni-directional material, shear test
    13*)
(*********************************************************************)
string0="K0_ANLAUFRECHNUNG_H1.txt";
data0=ReadList[string0,Number,RecordSeparators->","];
string1="K0_matheEGREEN_einzeln.txt";
data1=ReadList[string1,Number,RecordSeparators->","];
string2="K0_matheT2PK_einzeln.txt";
data2=ReadList[string2,Number,RecordSeparators->","];
string4="K0_ANLAUFRECHNUNG_H0.txt";
data4=ReadList[string4,Number,RecordSeparators->","];

(*read H0*)
H0=Table[0,{i,3},{j,3}] ;
F0=Table[0,{i,3},{j,3}] ;
z = 0;
For[i=1,i<=3,i++,
For[j=1,j<=3,j++,
z+=1;
H0[[i,j]] = data4[[z]];
F0[[i,j]]=H0[[i,j]];
If[i==j,F0[[i,j]]+=1];
];
];

(*read H1*)
H1=Table[0,{i,3},{j,3}] ;
F1=Table[0,{i,3},{j,3}] ;
z = 0;
For[i=1,i<=3,i++,
For[j=1,j<=3,j++,
z+=1;
H1[[i,j]] = data0[[z]];
F1[[i,j]]=H1[[i,j]];
If[i==j,F1[[i,j]]+=1];
];
];

(*read E Green*)
EG=Table[0,{i,6},{j,6}] ;
z = 0;
For[i=1,i<=6,i++,
For[j=1,j<=6,j++,
z+=1;
EG[[j,i]] = data1[[z]];
If[j>3,EG[[j,i]]*=Sqrt[2]];
```

```
];
];

(*read T2PK*)
T=Table[0,{i,6},{j,6}] ;
z = 0;
For[i=1,i<=6,i++,
For[j=1,j<=6,j++,
z+=1;
T[[j,i]] = data2[[z]];
If[j>3,T[[j,i]]*=Sqrt[2]];
];
];

(*Calculation of K1*)
K=Table[Kparam[i,j],{i,1,6},{j,1,6}];
eqs1=Flatten[Table [{T[[i,k]]==Sum [K[[i,j]]*EG[[j,k]],{j,6}]},{i
    ,1,6},{k,1,6}] ];
(*Print["eqs1 = ",MatrixForm[eqs1]]*)
sol=Solve[eqs1,Flatten[K]];
Set@@@sol[[1]];

EINS=Table[0,{i,3},{j,3}] ;
EINS[[1,1]]=1.;
EINS[[2,2]]=1.;
EINS[[3,3]]=1.;
Pext=Inverse[EINS+H1];
CPext=Transpose[Pext].Pext;
evals=Eigenvalues[CPext];
eveks=Eigenvectors[CPext];
Uext=Sum[Sqrt[evals[[i]]]*Outer[Times,Normalize[eveks[[i]]],Normalize[
    eveks[[i]]]],{i,1,3}];
Rext=Pext.Inverse[Uext];
Print["PC = ",MatrixForm[Pext]];
F=Inverse[Pext];
Print["F = ",MatrixForm[F]];
Print["H0 = ",MatrixForm[H0]];
Print["F*K1 = ",MatrixForm[Translate3333to66[Rayleigh[F,
    Translate66to3333[K]]]]];
K1 = 1/2(K+Transpose[K]);
Print["symF*K1 = ",MatrixForm[Translate3333to66[Rayleigh[F,
    Translate66to3333[K1]]]]];
(*********************************************************************)
(*Calculation of PK*)
(*********************************************************************)
K0=
{{34828.1,13568.3,14519.,-0.0314602,-0.0319863,-0.0102192},
{13568.3,34826.,14518.3,-0.030704,-0.0102193,-0.0319844},
{14519.,14518.3,72710.5,-0.00962756,-0.0342346,-0.0342332},
{-0.0314602,-0.030704,-0.00962756,21264.4,1.57501*10^-9,-6.31069*10^-9},

{-0.0319863,-0.0102193,-0.0342346,1.57501*10^-9,22598.,-0.000766524},
```

```
{-0.0102192,-0.0319844,-0.0342332,-6.31069*10^-9,-0.000766524,22597.2}};

(*********************************************************************)
(*Analytical calculation of PK (three free values)*)
(*********************************************************************)
time=100;
PC=Pext;
Fpunkt = H1 / time;
Print["Fpunkt = ",MatrixForm[Fpunkt]];
(*Determination of factors k1, k2 and k3 from seperate starting
    calculation*)
k1=(0.999837-1)/(0.5-0);
k2=(1.03216-1)/(0.5-0);
k3=(0.916138-1)/(0.5-0);
Print["k1 = ",k1]
Print["k2 = ",k2]
Print["k3 = ",k3]
P11={};
P22={};
P33={};
P3list={};
Klist={};
FSTART = {};
AppendTo[FSTART,{{1,0,0},{0,1,0},{0,0,1}}];
AppendTo[P11,1];
AppendTo[P22,1];
AppendTo[P33,1];
Print["=== Euler explicit ==="];
Print["=== with evolution of K ==="];
h=1;
For[z=1,z<=100,z++,
AppendTo[FSTART,FSTART[[z]]+h*Fpunkt];
L=Fpunkt.Inverse[FSTART[[z+1]]];
Lref=(Inverse[FSTART[[z+1]]].L).FSTART[[z+1]];
AppendTo[P11,P11[[z]]+h*k1 (Sqrt[Lref[[2,1]]^2+Lref[[3,1]]^2])];
AppendTo[P22,P22[[z]]+h*k2 (Sqrt[Lref[[1,2]]^2+Lref[[3,2]]^2])];
AppendTo[P33,P33[[z]]+h*k3 (Sqrt[Lref[[1,3]]^2+Lref[[2,3]]^2])];
AppendTo[P3list,{{P11[[z]],0,0},{0,P22[[z]],0},{0,0,P33[[z]]}}];
(*Print["z = ",z,"    P3list = ",P3list[[z]]];*)
AppendTo[Klist,Translate3333to66[Rayleigh[Inverse[FSTART[[z]]].P3list[[
    z]],Translate66to3333[K0]]]];
(*Print["z = ",z,"    Klist = ",MatrixForm[Klist[[z]]]];*)
];
P3={{P11[[-1]],0,0},{0,P22[[-1]],0},{0,0,P33[[-1]]}};
(*********************************************************************)
 (*Numerical calculation of Pk = PMIN3 with three free values*)
(*********************************************************************)
Clear[P11,P22,P33]
(*PC=Pext;*)
P={{P11,0.,0.},{0.,P22,0.},{0.,0.,P33}};
expr=K1-Translate3333to66[Rayleigh[PC.P,Translate66to3333[K0]]];
expr2=Flatten[Table[Table[expr[[i,j]]^2,{i,j,6}],{j,1,6}]];
```

```
minimi=Expand[Sum[expr2[[i]],{i,1,21}]];
erg=FindMinimum[minimi,{{P11},{P22},{P33}},MaxIterations->800];
Print["erg ",erg]
(*erg2=NMinimize[{minimi,-1\[LessEqual]P11\[LessEqual]1,-1\[LessEqual]
    P22\[LessEqual]1,-1\[LessEqual]P33\[LessEqual]1},{P11,P22,P33}];
Print["erg2 ",erg2]*)
PMIN3={{erg[[2,1,2]],0,0},{0,erg[[2,2,2]],0},{0,0,erg[[2,3,2]]}}
(*****************************************************************)
(*Numerical calculation of Pk = PMIN9 with nine free values*)
(*****************************************************************)
P={{P11,P12,P13},{P21,P22,P23},{P31,P32,P33}};
expr=K1-Translate3333to66[Rayleigh[PC.P,Translate66to3333[K0]]];
expr2=Flatten[Table[Table[expr[[i,j]]^2,{i,j,6}],{j,1,6}]];
minimi=Expand[Sum[expr2[[i]],{i,1,21}]];
(*EINSTENSOR als Startwert*)
PSTART={{1,0,0},{0,1,0},{0,0,1}};
start=Table[{Riffle[Flatten[P],Flatten[PSTART]][[2 i+1]],Riffle[Flatten
    [P],Flatten[PSTART]][[2 i+2]]},{i,0,8}];
erg=FindMinimum[minimi,start,MaxIterations->800];
Print["erg ",erg];
PMIN9={{erg[[2,1,2]],erg[[2,2,2]],erg[[2,3,2]]},{erg[[2,4,2]],erg
    [[2,5,2]],erg[[2,6,2]]},{erg[[2,7,2]],erg[[2,8,2]],erg[[2,9,2]]}};

SetDirectory[NotebookDirectory[]]
(*****************************************************************)
(*Write Output*)
(*****************************************************************)
Print["H0 = ",MatrixForm[H0]]
Print["F0 = ",MatrixForm[F0]]
Print["H1 = ",MatrixForm[H1]]
Print["F1 = ",MatrixForm[F1]]

Print["DeltaEG = ",MatrixForm[EG]]
Print["DeltaT2PK = ",MatrixForm[T]]

Print["F=",MatrixForm[Inverse[Pext]]];
Print["P3 = ",MatrixForm[P3]];
Print["PMIN3 = ",MatrixForm[PMIN3]];
Print["PMIN9 = ",MatrixForm[PMIN9]];

Print["K0 = ",MatrixForm[K0]];
Print["K1 = ",MatrixForm[K1]];
Print["F*K1 = ",MatrixForm[Translate3333to66[Rayleigh[F,
    Translate66to3333[K1]]]]];
Print["PC*PMIN3*K1 = ",MatrixForm[Translate3333to66[Rayleigh[PC.PMIN3,
    Translate66to3333[K0]]]]];
Print["||K1 - K0|| / ||K1|| = ",          MatrixForm[Norm[K1-K0,"
    Frobenius"]/Norm[K0,"Frobenius"]]];
Print["||K1 - PC*K0|| / ||K1|| = ",          MatrixForm[Norm[K1-
    Translate3333to66[Rayleigh[PC,Translate66to3333[K0]]],"Frobenius"]/
    Norm[K1,"Frobenius"]]];
```

```
Print["||K1 - PC*P3*K0|| / ||K1|| = ",MatrixForm[Norm[K1-
    Translate3333to66[Rayleigh[PC.P3,Translate66to3333[K0]]],"Frobenius
    "]/Norm[K1,"Frobenius"]]];
Print["||K1 - PC*PMIN3*K0|| / ||K1|| = ",MatrixForm[Norm[K1-
    Translate3333to66[Rayleigh[PC.PMIN3,Translate66to3333[K0]]],"
    Frobenius"]/Norm[K1,"Frobenius"]]];
Print["||K1 - PC*PMIN9*K0|| / ||K1|| = ",MatrixForm[Norm[K1-
    Translate3333to66[Rayleigh[PC.PMIN9,Translate66to3333[K0]]],"
    Frobenius"]/Norm[K1,"Frobenius"]]];
L=F-EINS;
LP=PMIN3-EINS;
Print["L=F-1=",MatrixForm[L]];
Print["LSYM = ",MatrixForm[1/2*(L+Transpose[L])]];
Print["LSKW = ",MatrixForm[1/2*(L-Transpose[L])]];
Print["LP=PMIN3-1=",MatrixForm[LP]];
Print["LPSYM = ",MatrixForm[1/2*(LP+Transpose[LP])]];
Print["LPSKW = ",MatrixForm[1/2*(LP-Transpose[LP])]];
Print["H0 = ",MatrixForm[H0]];
Print["H1 = ",MatrixForm[H1]];
Print["F0 = ",MatrixForm[F0]];
Print["F1 = ",MatrixForm[F1]];
Print["F = ",MatrixForm[F]];
```

F.2 Distance of stiffness tetrads to symmetry classes

```
Remove["Global`*"]
(* This script calculates the distances of a given stiffness tetrad \
to its projections onto the 7 non-trivial symmetry classes of \
elasticity based on Weber, Gluege Bertram: Distances of stiffness \
tetrads to the
symmetry groups of elasticity. International Journal of Solids and \
Structures 2018. *)
(* Input: 6x6 stiffness Matrix w.r.t. the \
6-dimensional non-normalized symmetric tensor basis with the ordering \
following Cowin (11,22,33,12,13,23) *)
(* Run with "Evaluation" \
\[Rule] "Evaluate Notebook" *)
(* Output: Bar-chart of the distances *)


(* The Input *)

K66 = {{47344.5, 14639.4, 14639.4, -622.211, -622.175, -1439.37},
   {14639.4, 39515.5, 12773.5, 4385.7, 668.875, -2304.24},
   {14639.4, 12773.5, 39515.5, 668.931, 4385.75, -2304.15},
   {-622.211, 4385.7, 668.931, 15690., -3312.97, -1811.98},
```

```
   {-622.175, 668.875, 4385.75, -3312.97, 15690.1, -1811.92},
   {-1439.37, -2304.24, -2304.15, -1811.98, -1811.92, 18908.3}};

(* Utility functions *)

T8Rayleigh[T2_] :=
  Table[T2[[i, m]] T2[[j, n]] T2[[k, o]] T2[[l, p]], {i, 1, 3}, {j, 1,
     3}, {k, 1, 3}, {l, 1, 3}, {m, 1, 3}, {n, 1, 3}, {o, 1, 3}, {p, 1,
     3}];
T8T4[T8_, T4_] :=
  Table[Sum[
    T8[[m, n, o, p, i, j, k, l]] T4[[i, j, k, l]], {i, 1, 3}, {j, 1,
     3}, {k, 1, 3}, {l, 1, 3}],  {m, 1, 3}, {n, 1, 3}, {o, 1, 3}, {p,
    1, 3}];
T8T8[T8_, T82_] :=
  Table[Sum[
    T8[[m, n, o, p, i, j, k, l]] T82[[i, j, k, l, q, r, s, t]], {i, 1,
      3}, {j, 1, 3}, {k, 1, 3}, {l, 1, 3}],  {m, 1, 3}, {n, 1, 3}, {o,
     1, 3}, {p, 1, 3}, {q, 1, 3}, {r, 1, 3}, {s, 1, 3}, {t, 1, 3}];
T4T4[T4_, T42_] :=
  Sum[T4[[i, j, k, l]] T42[[i, j, k, l]], {i, 1, 3}, {j, 1, 3}, {k, 1,
     3}, {l, 1, 3}];
ScalarProdT4[T4_] := T4T4[T4, T4];
Translate66to3333[
  M66_] := (Indexmatrix = {{1, 4, 5}, {4, 2, 6}, {5, 6, 3}};
  Faktormatrix = {{1, 1/Sqrt[2], 1/Sqrt[2]}, {1/Sqrt[2], 1,
     1/Sqrt[2]}, {1/Sqrt[2], 1/Sqrt[2], 1}};
  Table[M66[[Indexmatrix[[i, j]], Indexmatrix[[k, l]]]]*
    Faktormatrix[[i, j]]*Faktormatrix[[k, l]], {i, 1, 3}, {j, 1,
    3}, {k, 1, 3}, {l, 1, 3}])
Translate3333to66[C3333_] := (Ivek1 = {1, 2, 3, 1, 1, 2};
  Ivek2 = {1, 2, 3, 2, 3, 3};
  Faktvek = {1, 1, 1, Sqrt[2], Sqrt[2], Sqrt[2]};
  Table[C3333[[Ivek1[[i]], Ivek2[[i]], Ivek2[[j]], Ivek1[[j]]]]*
    Faktvek[[i]]*Faktvek[[j]], {i, 1, 6}, {j, 1, 6}])
id = IdentityMatrix[3];
(* Parametrizing the rotation *)
(* R: 2nd order rotation tensor \
depending on three Euler angles *)
(* We choose the extrinsic \
rotations (i.e. w.r.t. fixed axes) in the order z,y,x around the \
angles alpha, beta, gamma, respectively. *)
(* The intervals of the \
Euler angles are:
  -Pi < alpha \[LessEqual] Pi
-Pi/2 \[LessEqual] beta \[LessEqual] Pi/2
-Pi < gamma < Pi*)
(* With this choice, simplifications regarding the \
angle gamma are possible for the monoclinic and transversal \
symmetries
with the generators that are used here. The distance function becomes \
gamma-invariant, and a minimization over alpha and beta
```

```
is sufficient, see the appending cells and also Cagri, Kochetov, \
Slawinski: Identifying Symmetry Classes of Elasticity Tensors
Using Monoclinic Distance Function, Journal of Elasticity (2011), \
102(2):175-190. *)

R = RotationMatrix[gamma, {1, 0, 0}].RotationMatrix[
    beta, {0, 1, 0}].RotationMatrix[alpha, {0, 0, 1}];

(* Rot: 8th order rotation tensor that, when applied to a fourth \
order tensor K, gives the generalized rotation (Rayleigh product) of \
R with K *)
Rot = T8Rayleigh[R];
(* The generators of the 6 discrete symmetry groups monoclinic, \
orthotropic, tetragonal, trigonal, hexagonal, cubic*)
SymGroups = {};
AppendTo[SymGroups, {-id + {{2, 0, 0}, {0, 0, 0}, {0, 0,
      0}}}]; (*1 monoclinic *)

AppendTo[SymGroups, {-id + {{2, 0, 0}, {0, 0, 0}, {0, 0,
      0}}, -id + {{0, 0, 0}, {0, 2, 0}, {0, 0,
      0}}}]; (*2 orthotropic *)

AppendTo[SymGroups, {{{0, 1, 0}, {-1, 0, 0}, {0, 0,
     1}}, -id + {{2, 0, 0}, {0, 0, 0}, {0, 0, 0}}}]; (*3 tetragonal *)

AppendTo[SymGroups, {RotationMatrix[2*Pi/3, {0, 0, 1}],
   RotationMatrix[Pi, {0, 1, 0}]}];  (*4 trigonal *)

AppendTo[SymGroups, {RotationMatrix[
    Pi/3, {1, 0,
     0}], -id + {{0, 0, 0}, {0, 0, 0}, {0, 0, 2}}}];  (*5 hexagonal *)

AppendTo[SymGroups, {{{1, 0, 0}, {0, 0, 1}, {0, -1, 0}}, {{0, 0,
     1}, {1, 0, 0}, {0, 1, 0}}}]; (*6 cubic *)

(* Prepare empty list to hold the projectors onto the symmetry groups*)

P8s = {};

(* We combine the group elements until no new group members appear \
and build then the projectors by averaging over all group members *)

For[iouter = 1, iouter <= 6, iouter++,
  Print["Building symmetry group ", iouter, " and projector"];
  list = SymGroups[[iouter]];
  init = Length[list];
  outit = 100000;
  (* Generate all elements in symmetry group,
  proceed while new elements appear *)
  While[init != outit,
   init = Length[list];
   For[i = 1, i <= init, i += 1,
```

```
    For[j = 1, j <= init, j += 1,
      list = Append[list, FullSimplify[list[[i]].list[[j]]]
      ];
    ];
   list = DeleteDuplicates[list];
   outit = Length[list];
   ];
  (* Build 8th order projector by summing over all group elements *)

   AppendTo[P8s,
   FullSimplify[
    1/Length[list] Sum[
      T8Rayleigh[list[[x]]], {x, 1, Length[list]}]]];
  ];
(* Initialize empty list that holds the distances later on *)

distances = {};
(* Cast input to fourth order tensor *)
K = Translate66to3333[K66];

(* Loop over the 6 non-trivial discrete symmetry groups *)

For[iouter = 1, iouter <= 6, iouter++,
  Print["Minimization ", iouter , " of 6."];
  (* Combine P8 und Rot *)
  SuperP = T8T8[P8s[[iouter]], Rot];
  objectivefunction = ScalarProdT4[T8T4[Rot, K] - T8T4[SuperP, K]];

  (* The method "RandomSearch" starts a local minimization from all \
grid points.
  With the grid spacing no larger than Pi/4,
  the global minimum is found, since
  no symmetries higher than a four-fold rotational symmetry
  can be distinguished in linear elasticity. *)

  ni = 4; nj = 2; nk = 4; (* alpha (2Pi),
  beta (Pi) and gamma (2Pi) interval division *)

  If[iouter == 1  || iouter == 5, (*
   in case of monoclinic and transversal isotropic symmetry a two-
   dimensional search domain suffices, see appending cells *)

   res = NMinimize[{objectivefunction /. {gamma -> 0}, -Pi <= alpha <=
        Pi, -Pi/2 <= beta <= Pi/2}, {alpha, beta},
     Method -> {"RandomSearch",
       "InitialPoints" ->
        Flatten[Table[{-Pi + i*2 Pi/ni, -Pi/2 + j*Pi/nj}, {i, 1,
           ni}, {j, 1, nj}], 1]}],
   res = NMinimize[{objectivefunction, -Pi <= alpha <= Pi, -Pi/2 <=
       beta <= Pi/2, -Pi <= gamma <= Pi}, {alpha, beta, gamma},
     Method -> {"RandomSearch",
       "InitialPoints" ->
```

```
        Flatten[Table[{-Pi + i*2 Pi/ni, -Pi/2 + j*Pi/nj, -Pi +
            k*2 Pi/nk}, {i, 1, ni}, {j, 1, nj}, {k, 1, nk}], 2]}]
   ];

  (* The method "NelderMead" is much faster,
  but may not give the global minimum. *)
  (*erg=
  NMinimize[{objectivefunction,-Pi\[LessEqual]alpha\[LessEqual]Pi,-Pi/
  2\[LessEqual]beta\[LessEqual]Pi/
  2,-Pi\[LessEqual]gamma\[LessEqual]Pi},{alpha,beta,gamma},
  Method\[Rule]{"NelderMead"}];*)

  AppendTo[distances, Sqrt[res[[1]]]/Sqrt[ScalarProdT4[K]]];
  ];
(* Add distance to isotropy using isotropic projectors *)

P1 = Table[
   1/3 id[[i, j]] id[[k, l]], {i, 1, 3}, {j, 1, 3}, {k, 1, 3}, {l, 1,
    3}];
P2 = Table[
    1/2 (id[[i, k]] id[[j, l]] + id[[i, l]] id[[j, k]]), {i, 1,
     3}, {j, 1, 3}, {k, 1, 3}, {l, 1, 3}] - P1;
KISO = T4T4[K, P1] P1 + 1/5 T4T4[K, P2] P2;
AppendTo[distances, Sqrt[ScalarProdT4[K - KISO]/ScalarProdT4[K]]];
(* Draw chart *)
Print[distances];
BarChart[distances, BarSpacing -> {0.5, 2},
 ChartLabels -> {Placed[{"monoclinic", "orthotropic", "tetragonal",
     "trigonal", "hex./trans.-iso.", "cubic", "isotropic"}, Below]},
 ChartStyle -> {White}]
(* A P P E N D I X  C E L L  1  -  Demonstration of the \
interaction of R and P in special cases - part 1 - monoclinic \
symmetry *)

(* Simplifications regarding the angle gamma are possible for the \
monoclinic generator that has been used here.
The distance function becomes gamma-invariant, and a minimization \
over alpha and beta is sufficient. *)

SuperP = T8T8[P8s[[1]], Rot]; (* Load monoclinic projector *)

K66 = {{6, 5, 4, 3, 2, 1}, {5, 9, 5, 4, 3, 2}, {4, 5, 7, 1, 2, 3}, {3,
    4, 1, 5, 3, 2}, {2, 3, 2, 3, 8, 1}, {1, 2, 3, 2, 1,
    7}}; (* Triclinic stiffness *)
K = Translate66to3333[K66];
K = K/Sqrt[ScalarProdT4[K]]; (* Normalize for simplicity *)

objectivefunction = ScalarProdT4[T8T4[Rot, K] - T8T4[SuperP, K]];

ContourPlot3D[objectivefunction, {alpha, -Pi, Pi}, {beta, -Pi/2,
  Pi/2}, {gamma, -Pi, Pi}, Contours -> {0.04, 0.08, 0.12},
 ContourStyle -> {Directive[Opacity[0.3], Red],
```

```
   Directive[Opacity[0.2], Red], Directive[Opacity[0.1], Red]},
 Mesh -> None, PlotPoints -> {20, 10, 2}, MaxRecursion -> 0,
 BoxRatios -> {2, 1, 2}]

(* APPENDIX CELL 2 - Demonstration of the \
interaction of R and P in special cases - part 2 - transversal \
isotropy *)

(* Simplifications regarding the angle gamma are possible for the \
hexagonal generator (indistinguishable from transversal isotropy in \
elasticity) that has been used here.
The distance function becomes gamma-invariant, and a minimization \
over alpha and beta is sufficient. *)

SuperP = T8T8[P8s[[5]],
   Rot]; (* Load transversal isotropic projector *)

K66 = {{6, 5, 4, 3, 2, 1}, {5, 9, 5, 4, 3, 2}, {4, 5, 7, 1, 2, 3}, {3,
     4, 1, 5, 3, 2}, {2, 3, 2, 3, 8, 1}, {1, 2, 3, 2, 1,
    7}}; (* Triclinic stiffness *)
K = Translate66to3333[K66];
K = K/Sqrt[ScalarProdT4[K]];(* Normalize for simplicity *)

objectivefunction = ScalarProdT4[T8T4[Rot, K] - T8T4[SuperP, K]];

ContourPlot3D[objectivefunction, {alpha, -Pi, Pi}, {beta, -Pi/2,
  Pi/2}, {gamma, -Pi, Pi}, Contours -> 3*{0.04, 0.08, 0.12},
 ContourStyle -> {Directive[Opacity[0.3], Red],
   Directive[Opacity[0.2], Red], Directive[Opacity[0.1], Red]},
 Mesh -> None, PlotPoints -> {10, 5, 10}, MaxRecursion -> 0,
 BoxRatios -> {2, 1, 2}]

(* APPENDIX CELL 3 - Number of grid points \
necessary to obtain the global minimum *)

(* The worst possible case is the determination of the distance of a \
triclinic cell to the monoclinic symmetry.
Then the global minimum is only two-fold, as the monoclinic symmetry \
group is of order two. Less global minima
are not possible. Using the result from Appendix Cell 1, we draw grid \
points and the distance landscape in one plot to see that a grid
spacing of Pi/4 is sufficient. Then, each local minimum has at least \
one grid point, such that each local minimum is attained
at least once. However, a better discretization of the orientation \
space than with two Euler angles would be through a fair distribution \
of points on the sphere
(see second plot). In case of three Euler angles, unit quaternions or \
platonic solids may give a good starting point distribution (see, \
e.g. Nawratil and Pottmann:
Subdivision Schemes for the fair Discretization of the Spherical \
Motion Group, Journal of Computational and Applied Mathematics \
(2008), 222(2), 574-591). *)
```

```
SuperP = T8T8[P8s[[1]], Rot]; (* Load monoclinic projector *)

K66 = {{47344.5, 14639.4,
    14639.4, -622.211, -622.175, -1439.37}, {14639.4, 39515.5,
    12773.5, 4385.7, 668.875, -2304.24}, {14639.4, 12773.5, 39515.5,
    668.931, 4385.75, -2304.15}, {-622.211, 4385.7, 668.931,
    15690., -3312.97, -1811.98}, {-622.175, 668.875,
    4385.75, -3312.97,
    15690.1, -1811.92}, {-1439.37, -2304.24, -2304.15, -1811.98, \
-1811.92, 18908.3}}; (* Triclinic stiffness *)

K = Translate66to3333[K66];
K = K/Sqrt[ScalarProdT4[K]];(* Normalize for simplicity *)

objectivefunction = ScalarProdT4[T8T4[Rot, K] - T8T4[SuperP, K]];
gridspacing = Pi/4;
Show[ContourPlot[
  objectivefunction /. {gamma -> 0}, {alpha, -Pi, Pi}, {beta, -Pi/2,
   Pi/2}, Mesh -> None, Contours -> 10, PlotPoints -> {40, 20},
  MaxRecursion -> 1, AspectRatio -> 1/2,
  ColorFunction -> "ThermometerColors"],
 Graphics[{PointSize[Large],
   Point[Flatten[
     Table[{-Pi + i*gridspacing, -Pi/2 + j*gridspacing}, {i, 0,
       2 Pi/gridspacing}, {j, 0, Pi/gridspacing}], 1]]}]]
o[alpha_, beta_] = 30*objectivefunction /. {gamma -> 0};
SphericalPlot3D[1, {beta, 0, Pi}, {alpha, 0, 2 Pi},
 ColorFunction ->
  Function[{x, y, z, beta, alpha, r},
   ColorData["ThermometerColors"][o[alpha, beta - Pi/2]]],
 Mesh -> None, PlotPoints -> {80, 40}, Boxed -> True, Axes -> True,
 ColorFunctionScaling -> False]
```