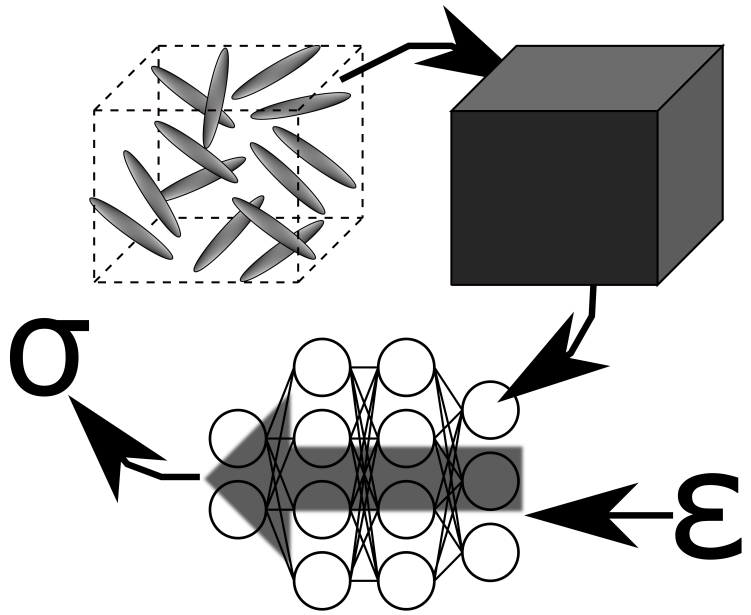
CHALMERS
UNIVERSITY OF TECHNOLOGY



Predicting the elasto-plastic response of short fiber composites using deep neural networks trained on micro-mechanical simulations

Master's thesis in Applied Mechanics

JOHAN FRIEMANN

Department of Industrial and Materials Science

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

MASTER'S THESIS 2021

Predicting the elasto-plastic response of short fiber composites using deep neural networks trained on micro-mechanical simulations

JOHAN FRIEMANN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Industrial and Materials Science
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Predicting the elasto-plastic response of short fiber composites using deep neural networks trained on micro-mechanical simulations

JOHAN FRIEMANN

Master's Thesis 2021
Department of Industrial and Materials Science
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 (0)31-772 1000

Cover: An abstraction of the proposed neural network based material model. A short fiber composite material is homogenized and used as training data for a neural network that predicts the stress given a strain history.

Predicting the elasto-plastic response of short fiber composites using deep neural networks trained on micro-mechanical simulations

JOHAN FRIEMANN

Department of Industrial and Materials Science

Chalmers University of Technology

Abstract

The mechanical modeling of short fiber composites has proven to be difficult. This is partly owing to the high degree of anisotropy and fiber discontinuity. Being able to accurately predict the behavior of short fiber composites with varying fiber orientations and fiber volume fractions is highly relevant in the design and production of injection molded parts. It has therefore become popular to abandon classical constitutive models in favor of data driven models. Artificial neural networks is a popular and efficient method of using large amounts of data to teach an algorithm the underlying rules of a phenomenon, enabling it to make predictions for data it has never before encountered. In this work a recurrent deep neural network model utilizing Gated Recurrent Units (GRU) is trained to predict the elasto-plastic stress response of a short fiber composite material given the strain path. The model is designed to have the ability to make predictions for arbitrary fiber orientations and varying fiber volume fractions. The training data is generated by performing micro-mechanical simulations utilizing mean field methods in the commercially available software DIGIMAT-MF. The strain data is generated by a random walk scheme in strain space. The finished model performs well, and the mean error for a typical load cycle usually stays below 10% of the matrix yield stress.

Keywords: Composite mechanics, Short fiber composites, Material mechanics, Micro mechanics, Elasto-plasticity, Mean field homogenization, Artificial neural networks, Deep neural networks, Recurrent neural networks

Acknowledgements

I want to begin by thanking my supervisor Mohsen Mirkhalaf, Assistant Professor at the University of Gothenburg, The Faculty of Science, Department of Physics, for his continuous advice and by always challenging me to improve my research. I also want to thank my co-supervisor Behdad Dasht Bozorg, University Researcher at Eindhoven University of Technology, Department of Biomedical Engineering, Medical Image Analysis group, for his advice concerning artificial neural networks. Many times when I could not make progress with my network implementation a discussion with Behdad cleared out the issues.

I want to express my gratitude to the examiner of this thesis, Martin Fagerström, Associate Professor at the Division of Material and Computational Mechanics, Department of Industrial and Materials Science, Chalmers University of Technology. He offered many helpful comments when revising my thesis drafts, and has been invaluable to the project by arranging for computational resources.

I want to offer my thanks to Aykut Argun of the University of Gothenburg, Department of Physics, for a fruitful discussion on recurrent neural networks.

I want to extend my most heartfelt thanks to Professor Jun Takahashi of the University of Tokyo, Graduate School of Engineering, Department of Systems Innovation, for hosting me as an exchange student between the fall of 2019 and summer of 2020. The seed of this project was sown while still attending his laboratory. Additionally, I learned many things during my time at the laboratory that has been of great use for this thesis.

I want to thank *e-Xstream* for supplying me with a DIGIMAT-license. They also deserve to be acknowledged for quickly transferring said license when my computer broke down at two different occasions.

Finally, I want to thank my family for their everlasting support. They valiantly endured listening to me explaining my work at best, and my complaints about the perils of academic writing at worst.

The training of the artificial neural network was enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at Chalmers Center for Computational Science and Engineering (C3SE) partially funded by the Swedish Research Council through grant agreement no. 2018-05973.

Johan Friemann 2021-01-31 Gothenburg, Sweden

Contents

1 Introduction

Classical laminate theory has long been proven to accurately model the elastic behavior of continuous laminated fiber composite materials. These composites have many excellent mechanical properties and have seen widespread application in the aerospace and automotive industry. Ease of modeling the material stiffness enables engineers to quickly test new designs without relying on computationally expensive and time consuming computer simulations. There are however drawbacks with long continuous fibers. For example, they are typically expensive and can be difficult to work with. Long fiber sheets are difficult to form into complex geometries, an issue that hampers their utilization for intricately designed car bodies, among others. It may be more suitable to use injection molding for such applications, where short fibers are used. In addition to be being easier to manufacture, short fiber composite production also have the potential to be more sustainable since they are recyclable. In an age where environmental concerns are a high priority, materials used in mass production must have further use at the end of their life cycle. Furthermore, recycled materials can be very cost effective fetching prices as low as a quarter of virgin material [1]. It is possible to reuse the materials in continuous composites, the fibers will however break into shorter pieces that can be used as short fiber reinforcement.

Short fiber composites have great potential to be used in high performance materials. As mentioned before, it is a suitable material for injection molding which enables faster and more cost efficient production [2]. The molding process itself also tends to align fibers with the material flow direction, creating a preferential fiber orientation. This gives engineers the ability to control the material properties. A useful computational model describing such a material must therefore be able to account for the varying orientation of fibers and their volume fraction.

Furthermore, short fibers are also suitable for use in additive manufacturing. Recent developments enables printing complex geometries with excellent mechanical properties [3]. One of the issues that stops the short fiber materials from being readily available to engineers is the difficulty of accurately modeling them. Homogenization schemes and subsequent experiments can give macroscopic material parameters such as stiffness and strength. However, the discontinuous and often random nature of the materials gives a large variance in test results. For design purposes, high accuracy is key and thus design engineers need accurate material data and models. Especially in industries such as the automotive industry where repeated prototyping isn't feasible.

In the quest to find constitutive material models describing the behavior of short fiber composites, homogenization schemes have been thoroughly investigated (see e.g. [4]). Homogenization seeks to

describe the composite material based on the mechanical properties of its constituents. Using material data for the matrix and the reinforcing elements, a macroscopic material property is calculated through homogenization techniques. These material models may be very case specific, complex, or computationally expensive. Currently, the most accurate homogenization technique is computational homogenization. One such approach is creating a numerical Representative Volume Element (RVE), which reflects the typical micro-structural features of the material, and subsequently performing finite element simulations to compute the macroscopic properties of the material. This approach faces several issues. To begin with, such computations can be very expensive to perform. Secondly, the generation of adequate RVE's themselves can be expensive and difficult [5][6]. As an alternative, data driven modeling approaches are being investigated. One such approach is machine learning utilizing deep neural networks.

Deep Neural Networks (DNN) have recently gained mainstream popularity for a wide variety of big data applications such as making stock market predictions [7], natural language processing, and image recognition [8]. DNN consists of a large network of artificial neurons whose individual properties are very simple, but their connected behavior can give rise to very complex phenomena. The network is trained (i.e. calibrated) to accomplish a task by feeding it large amounts of data and adjusting the internal parameters as to minimize the error between the predicted outcome and the data that it has been fed. Once trained, the computational speed of predictions is very high. A novel use is to train DNN to be used as constitutive models for composite materials. By performing micro-mechanical simulations, strain stress data can be generated. A DNN is then trained to learn the behavior of the material to give accurate stress predictions given the strain (and its history). This approach has been tested for nonlinear elasticity [9] and path dependent plasticity [10] and shows a great promise. While the referred works are great proofs of concept, they are developed with only one specific material in mind and do not necessarily generalize. To really show the benefit of the computational speed of the DNN approach when compared with classical constitutive modeling, generality is key. A model that can take micro-mechanical descriptors into account to be able to generalize to a family of materials needs to be developed. A general model could outperform standard methods where the micro-structure varies highly, such as for injection molded parts. For short fiber composites, descriptors of the micro-structure are fiber length and aspect ratio, the distribution of fiber orientations, and the volume fraction of fibers among others.

1.1 Purpose

The purpose of this project was to develop a DNN model that can replicate the path dependent general 3D elasto-plastic response of a Short Fiber Reinforced Composite (SFRC). The model is trained on the results of micro-mechanical simulations, utilizing a mean-field approach, of an SFRC with different fiber orientations and fiber volume fractions. The DNN model should be able to describe the elasto-plastic response of the SFRC for an arbitrary fiber distribution, and for fiber volume fractions within a small range.

The success of the DNN implementation relies on three major factors. First, the implementation needs to accurately capture the correct physical features of the underlying micro-mechanical model. This includes accurate yielding, hardening, and a clear difference of loading/unloading. Secondly,

it must possess an ability to display the proper rate independence that the underlying micro-mechanical model requires. Finally, it should be able to make elasto-plastic predictions of fairly complex loading with an acceptably small prediction error.

1.2 Limitations

The project was limited to implementing the model for only one set of composite constituents. Specifically, the model was developed for a material with a matrix that obeys isotropic J_2-hardening plasticity and has fibers that are linearly elastic. The volume fraction of fibers was kept below 15%. The model was only trained through results from mean field simulations, there were no comparison with a model trained from homogenized materials via finite element simulations of RVE's. No physical experiments was performed, neither for creating training data nor for validating the model. It was also outside the scope of this project to validate the developed model through structural simulations, for example through utilization of finite elements.

2 Theory

The underlying theory concerning the current work is divided into three parts. The first part entails describing the constitutive behavior of an elasto-plastic material based on fundamental thermodynamical considerations, while the second part consists of how to describe the complex micro-structure of an SFRC material as a continuum. The final part however, leaves the realm of mechanics and concerns the workings of artificial neural networks. The chapter concludes with a description of recurrent neural networks, whose ability to classify time dependent data make them suitable to predict path dependent plasticity.

The present work utilizes index notation with covariant and contravariant indices to describe tensors and variants. A reader unfamiliar with these concepts is referred to Appendix A for a comprehensive introduction. It is assumed in this work that the treated strains are small enough that infinitesimal strain theory may be applied. That is, the second order displacement gradient is negligible. For a review of topics in continuum solid mechanics and continuum thermodynamics, the reader is referred to Appendix B.

2.1 Material Mechanics

In linear elasticity the strain energy density has no dependence on any other internal variables except the mechanical strain, i.e. the strain energy density is equivalent to the free energy density. The energy density follows

$$\psi(\boldsymbol{\varepsilon}) = \frac{1}{2} C^{ijkl} \varepsilon_{ij} \varepsilon_{kl}, \tag{2.1}$$

and the stress is obtained easily through differentiation with respect to $\boldsymbol{\varepsilon}$. A material only obeying this energy density will always return to its original shape after unloading. Moreover, it would continue to behave linearly for infinite stresses and strains. In reality, most materials deform permanently, in process called *plasticity*, when stressed sufficiently. The constitutive model is extended to allow *yielding*, where permanent deformation starts taking place when the applied stress is greater or equal to some *yield stress*. Subsequently, internal variables need to be introduced that store the information of the deformation history. To account for any non-linearity, the energy density functional needs to be modified to depend on these new internal variables.

2.1.1 Flow Plasticity Theory

The fundamental assumption of flow plasticity theory is that the strain inside a material can be additively decomposed into an elastic component and plastic component as

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{\mathrm{e}} + \boldsymbol{\varepsilon}^{\mathrm{p}}. \tag{2.2}$$

The elastic strain $\boldsymbol{\varepsilon}^{\mathrm{e}}$ is the strain that results in the stress through Hooke's law. The stress is assumed to only depend on the elastic strain. The plastic strain $\boldsymbol{\varepsilon}^{\mathrm{p}}$ is a new internal variable that records the irreversible deformation due to plasticity. The condition that is chosen to control whether the elastic or plastic strain is changing for a given loading situation is a *yield function*. The yield function Φ is typically dependent on the stress and decides the current mode of deformation through

$$\begin{aligned} &\text{Elastic deformation if} \quad \Phi(\boldsymbol{\sigma}) < 0, \\ &\text{Plastic deformation if} \quad \Phi(\boldsymbol{\sigma}) = 0. \end{aligned} \tag{2.3}$$

The evolution of the elastic strain is coupled with the evolution of the stress state. However, the plastic strain needs its own equation of evolution. It is not sufficient to directly describe the current plastic strain as it does not include information whether the material is currently yielding or undergoing elastic deformation. Instead the plastic strain rate $\dot{\boldsymbol{\varepsilon}}^{\mathrm{p}}$, or plastic flow, is considered. This leads to the name flow plasticity theory. Similarly, the evolution of other internal variables is also viewed as flows. The rules that govern these flows can be found through thermodynamic considerations.

2.1.2 Flow Rules Derived from the Principle of Maximum Dissipation

The principle of maximum dissipation is a postulate that states that the dissipation rate is maximized with respect to the dissipative stresses [11]. It is an equivalent problem minimizing the negative dissipation rate. If the yield function Φ and the dissipation rate is convex, the *Karush-Kuhn-Tucker* theorem states that a global minimum of the negative dissipation rate $-\mathcal{D}(\boldsymbol{\kappa})$ that adheres to the yield function $\Phi \leq 0$ is found through solving [12]

$$\begin{aligned} &\nabla_{\boldsymbol{\kappa}}(\mathcal{L}(\boldsymbol{\kappa}, \lambda)) = \mathbf{0} \quad \text{where} \quad \mathcal{L} = -\mathcal{D}(\boldsymbol{\kappa}) + \lambda\Phi(\boldsymbol{\kappa}) \\ &\text{such that } \lambda \geq 0,\ \Phi(\boldsymbol{\kappa}) \leq 0 \quad \text{and} \quad \lambda\Phi = 0, \end{aligned} \tag{2.4}$$

where λ is a Lagrangian multiplier. Due to the appearance of $\mathcal{D}$, which is shown in the end of Appendix B, $\nabla_{\boldsymbol{\kappa}}\mathcal{L} = \mathbf{0}$ will give a direct relationship between the rate of change of the internal variables and λ and Φ. In other words, it is possible to find the complete flow equations and loading conditions for any admissible potential ψ and associated yield function Φ by solving Equation (2.4).

2.1.3 1-Dimensional Plasticity

Perfect plasticity is the most basic flow plasticity model. The model follows the familiar elastic stress strain relation with Young's modulus E until some critical yield stress σ_{y} is reached. When the yield stress is reached the plastic strain starts changing under constant stress. It can be visualized as a spring attached to a slider, as in Figure 2.1, where the slider is open if and only if $\sigma = \sigma_{\mathrm{y}}$. The free energy and yield function for such a material takes the form

$$\psi(\varepsilon) = \frac{1}{2}E(\varepsilon - \varepsilon^{\mathrm{p}})^2, \quad \Phi(\sigma) = |\sigma| - \sigma_{\mathrm{y}} \leq 0. \tag{2.5}$$

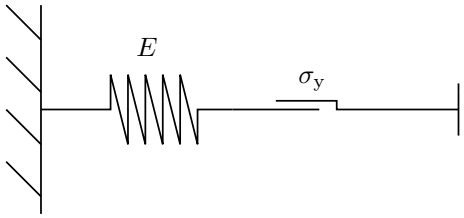


Figure 2.1: A schematic of a 1-dimensional perfect plasticity model is displayed.

The stress then follows as

$$\sigma = \frac{\partial \psi}{\partial \varepsilon} = E(\varepsilon - \varepsilon^{\mathrm{p}}) = E\varepsilon^{\mathrm{e}}, \tag{2.6}$$

and the dissipative stress as

$$\sigma_{\mathrm{d}} = -\frac{\partial \psi}{\partial \varepsilon^{\mathrm{p}}} = E(\varepsilon - \varepsilon^{\mathrm{p}}) = \sigma. \tag{2.7}$$

Then, Equation (2.4) implies that

$$\dot{\varepsilon}^{\mathrm{p}} = \lambda \frac{\partial}{\partial \sigma}(|\sigma| - \sigma_{\mathrm{y}}) = \lambda \ \mathrm{sgn}(\sigma), \tag{2.8}$$

where $\lambda \geq 0$ and $\lambda\Phi = 0$. To fully determine the evolution of the plastic strain it remains to find an expression for λ. As $\Phi\lambda = 0$, it is possible to draw the conclusion that λ must be 0 when $\Phi < 0$, i.e during elastic loading. The value of λ during plastic loading is determined by realizing that when plastic loading occurs $\Phi = 0$ and subsequently $(\dot{\lambda\Phi}) = \dot{0} = \dot{\Phi}\lambda + \dot{\lambda}\Phi = 0 \Rightarrow \dot{\Phi} = 0$ since $\lambda \neq 0$. Thus λ during plastic loading is found through

$$\dot{\Phi} = \frac{\partial \Phi}{\partial \sigma}\dot{\sigma} = \mathrm{sgn}(\sigma)E(\dot{\varepsilon} - \dot{\varepsilon}^{\mathrm{p}}) = \mathrm{sgn}(\sigma)E(\dot{\varepsilon} - \lambda \ \mathrm{sgn}(\sigma)) = 0 \Longrightarrow \lambda = \dot{\varepsilon} \ \mathrm{sgn}(\sigma). \tag{2.9}$$

It is thereby possible to state the complete equations of evolution in terms of the strain rate $\dot{\varepsilon}$ as

$$\begin{aligned} &\text{Elastic loading/unloading } (\Phi < 0): \ \dot{\varepsilon}^{\mathrm{p}} = 0, \quad \dot{\sigma} = E\dot{\varepsilon} \\ &\text{Plastic loading } (\Phi = 0): \ \dot{\varepsilon}^{\mathrm{p}} = \dot{\varepsilon}, \quad \dot{\sigma} = 0. \end{aligned} \tag{2.10}$$

Most real yielding engineering materials allow the stress to increase further after the yield stress is reached. This phenomenon is called hardening. A hardening material behaves linearly elastic with Young's modulus E until the yield stress σ_{y} is reached. After yielding the stiffness changes and the stress may increase further. This model can be visualized as a spring attached to a spring with stiffness H and a slider in parallel, which can be seen in Figure 2.2. When the yield stress is reached, the two springs are in series and the combined stiffness, called tangent stiffness, becomes $E_{\mathrm{T}} = EH/(E+H)$. For an isotropic hardening material, the yield stress increases when the material is stressed beyond its yield limit. In the context of the spring-slider model, it can be thought of that the slider friction increases the further it slides. The free energy and yield function for an isotropic hardening material is given by

$$\psi = \frac{1}{2}(\varepsilon - \varepsilon^{\mathrm{p}})^2 + \frac{1}{2}Hk^2, \quad \Phi(\sigma, \kappa) = |\sigma| - (\sigma_{\mathrm{y}} + \kappa) \leq 0. \tag{2.11}$$

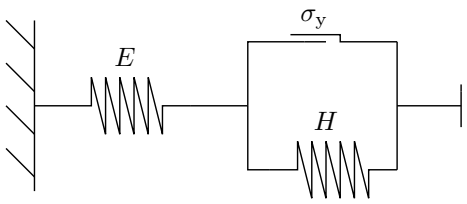


Figure 2.2: A schematic of a 1-dimensional hardening plasticity model is displayed.

Here k is a newly introduced internal variable that describes how the hardening of the material influences the stored free energy. The stress and dissipative stress related to ε^p are identical to the ones for perfect plasticity. However isotropic hardening plasticity has an additional dissipative stress, called micro hardening stress

$$\kappa = -\frac{\partial \psi}{\partial k} = -Hk. \tag{2.12}$$

Equation (2.4) then sets the flow rules for the internal variables as

$$\dot{\varepsilon}^p = \lambda \operatorname{sgn}(\sigma), \quad \dot{k} = -\lambda. \tag{2.13}$$

where $\lambda \geq 0$ and $\lambda\Phi = 0$. Using the exact same reasoning as for perfect plasticity, $\lambda = 0$ when the material behaves elastically. It also still holds that $\dot{\Phi} = 0$ when plastic loading occurs, so in plastic loading λ obeys

$$\dot{\Phi} = \frac{\partial \Phi}{\partial \sigma}\dot{\sigma} + \frac{\partial \Phi}{\partial \kappa}\dot{\kappa} = \operatorname{sgn}(\sigma)E(\dot{\varepsilon} - \dot{\varepsilon}^p) + H\dot{k} = \operatorname{sgn}(\sigma)E(\dot{\varepsilon} - \lambda \operatorname{sgn}(\sigma)) - H\lambda = 0. \tag{2.14}$$

Thus the plastic multiplier is obtained as

$$\lambda = \operatorname{sgn}(\sigma)\frac{E}{E+H}\dot{\varepsilon}. \tag{2.15}$$

The complete equations of evolution in terms of $\dot{\varepsilon}$ then follows

$$\begin{aligned} &\text{Elastic loading/unloading } (\Phi < 0): \ \dot{\varepsilon}^p = 0, \quad \dot{\sigma} = E\dot{\varepsilon}, \quad \dot{k} = 0 \\ &\text{Plastic loading } (\Phi = 0): \ \dot{\varepsilon}^p = \frac{E}{E+H}\dot{\varepsilon}, \quad \dot{\sigma} = \frac{EH}{E+H}\dot{\varepsilon}, \quad \dot{k} = -\operatorname{sgn}(\sigma)\frac{E}{E+H}. \end{aligned} \tag{2.16}$$

By considering plasticity in one dimension, the flow rules can be understood conceptually through the introduced spring-slider models. Based on the understanding of the equations of evolution in one dimension the models may be generalized to three dimensions. This is done by basing the yield condition on a scaled norm of the deviatoric stress. This is known as the equivalent *von Mises* stress.

2.1.4 Isotropic J_2-Hardening Plasticity

The deviatoric stress tensor is defined by

$$\boldsymbol{\sigma}_{\text{dev}} = \boldsymbol{\sigma} - \frac{1}{3}\operatorname{tr}(\boldsymbol{\sigma})\boldsymbol{I}, \tag{2.17}$$

where $\mathrm{tr}(\boldsymbol{\sigma})$ is the trace, defined by

$$\mathrm{tr}(\boldsymbol{\sigma}) = \mathrm{tr}(\sigma^{ij}\boldsymbol{e}_i\boldsymbol{e}_j) = \sigma^{ij}\boldsymbol{e}_i \cdot \boldsymbol{e}_j = \sigma^{ij}g_{ij} = \boldsymbol{\sigma} : \boldsymbol{I}. \tag{2.18}$$

In Equation (2.18), g_{ij} is the metric tensor and $\boldsymbol{I}$ is the second order identity. The trace consists of fully contracted tensors, and is as a consequence an invariant quantity (independent of the choice of coordinates). It follows from the definition given in Equation (2.17) that the trace of $\boldsymbol{\sigma}_{\mathrm{dev}}$ is zero, since the trace of the second order identity tensor is 3. The tensor

$$\boldsymbol{\sigma}_{\mathrm{vol}} = \frac{1}{3}\mathrm{tr}(\boldsymbol{\sigma})\boldsymbol{I} \tag{2.19}$$

is referred to as the volumetric stress. It follows from the definition that $\mathrm{tr}(\boldsymbol{\sigma}_{\mathrm{vol}}) = \mathrm{tr}(\boldsymbol{\sigma}) = \sqrt{3\boldsymbol{\sigma}^{\mathrm{vol}} : \boldsymbol{\sigma}^{\mathrm{vol}}}$. Another invariant quantity related to deviatoric stress is

$$\begin{aligned} &\frac{1}{2}\boldsymbol{\sigma}_{\mathrm{dev}} : \boldsymbol{\sigma}_{\mathrm{dev}} = (\boldsymbol{\sigma} - \frac{1}{3}\mathrm{tr}(\boldsymbol{\sigma})\boldsymbol{I}) : (\boldsymbol{\sigma} - \frac{1}{3}\mathrm{tr}(\boldsymbol{\sigma})\boldsymbol{I}) = \\ &\frac{1}{2}\boldsymbol{\sigma} : \boldsymbol{\sigma} - \frac{2}{6}\mathrm{tr}(\boldsymbol{\sigma})\boldsymbol{\sigma} : \boldsymbol{I} + \frac{1}{6}\mathrm{tr}(\boldsymbol{\sigma})^2 = \frac{1}{2}(\mathrm{tr}(\boldsymbol{\sigma}\cdot\boldsymbol{\sigma}) - \frac{1}{3}\mathrm{tr}(\boldsymbol{\sigma})^2) = J_2. \end{aligned} \tag{2.20}$$

J_2 is called the second invariant of the stress tensor. Experiments on the yielding behavior of ductile materials indicates that yielding is tied to a critical value of J_2. This gives the plasticity model its name. Typically the value of the yield stress of materials is investigated in a one dimensional setting where a uniaxial stress state arises. If a uni-axial stress σ_{y} leads to yielding, the second invariant at yielding is

$$2J_2 = \boldsymbol{\sigma}_{\mathrm{dev}} : \boldsymbol{\sigma}_{\mathrm{dev}} = \mathrm{tr}(\boldsymbol{\sigma}\cdot\boldsymbol{\sigma}) - \frac{1}{3}\mathrm{tr}(\boldsymbol{\sigma})^2 = \sigma_{\mathrm{y}}^2 - \frac{1}{3}\sigma_{\mathrm{y}}^2 = \frac{2}{3}\sigma_{\mathrm{y}}^2. \tag{2.21}$$

Thereby the yield stress can be related to the deviatoric stress and the von Mises equivalent stress is thus defined by

$$\sigma_{\mathrm{M}} = \sqrt{\frac{3}{2}\boldsymbol{\sigma}_{\mathrm{dev}} : \boldsymbol{\sigma}_{\mathrm{dev}}} = \sqrt{3J_2}. \tag{2.22}$$

If the von Mises equivalent stress reaches the yield stress σ_{y}, yielding occurs. This reproduces the correct yielding behavior in the uni-axial loading situation as required.

Inspired by the one-dimensional isotropic hardening yield function, the J_2-hardening yield function is stated using the von Mises equivalent stress as

$$\Phi(\boldsymbol{\sigma}, \kappa) = \sigma_{\mathrm{M}} - (\sigma_{\mathrm{y}} + \kappa) \leq 0. \tag{2.23}$$

The deviatoric and volumetric strain can be defined analogously as for the stress through

$$\boldsymbol{\varepsilon}^{\mathrm{dev}} = \boldsymbol{\varepsilon} - \frac{1}{3}\mathrm{tr}(\boldsymbol{\varepsilon})\boldsymbol{I}, \quad \boldsymbol{\varepsilon}^{\mathrm{vol}} = \frac{1}{3}\mathrm{tr}(\boldsymbol{\varepsilon})\boldsymbol{I}, \quad \boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{\mathrm{dev}} + \boldsymbol{\varepsilon}^{\mathrm{vol}}. \tag{2.24}$$

It follows directly from the definition that $\boldsymbol{\varepsilon}^{\mathrm{dev}} : \boldsymbol{\varepsilon}^{\mathrm{vol}} = 0$, and as for the stress $\mathrm{tr}(\boldsymbol{\varepsilon}^{\mathrm{dev}}) = 0$ and $\mathrm{tr}(\boldsymbol{\varepsilon}^{\mathrm{vol}}) = \mathrm{tr}(\boldsymbol{\varepsilon}) = \sqrt{3\boldsymbol{\varepsilon}^{\mathrm{vol}} : \boldsymbol{\varepsilon}^{\mathrm{vol}}}$. Using these properties and the isotropic Hooke's law,

$$C^{ijkl} = \lambda g^{ij}g^{kl} + \mu(g^{ik}g^{jl} + g^{il}g^{jk}), \tag{2.25}$$

the free energy is stated as

$$\psi(\boldsymbol{\sigma}, k) = \frac{1}{2} C^{ijkl} \varepsilon^{\mathrm{e}}_{ij} \varepsilon^{\mathrm{e}}_{kl} + \frac{1}{2} H k^2 = \mu \boldsymbol{\varepsilon}^{\mathrm{edev}} : \boldsymbol{\varepsilon}^{\mathrm{edev}} + \frac{1}{2} 3K \boldsymbol{\varepsilon}^{\mathrm{evol}} : \boldsymbol{\varepsilon}^{\mathrm{evol}} + \frac{1}{2} H k^2. \tag{2.26}$$

Here $K = (\lambda + \frac{2}{3}\mu)$ is the bulk modulus (in this equation λ is the first Lamé parameter, not to be confused with the Lagrange multiplier). Moreover $\varepsilon^{\mathrm{e}}_{ij} = \varepsilon_{ij} - \varepsilon^{\mathrm{p}}_{ij}$ is the elastic strain where $\varepsilon^{\mathrm{p}}_{ij}$ is the plastic strain. $\boldsymbol{\varepsilon}^{\mathrm{edev}}$ and $\boldsymbol{\varepsilon}^{\mathrm{evol}}$ are the deviatoric and volumetric parts of the elastic strain respectively. The stress can be computed as before through

$$\boldsymbol{\sigma} = \frac{\partial \psi}{\partial \boldsymbol{\varepsilon}} = C^{ijkl} \varepsilon^{\mathrm{e}}_{kl} \boldsymbol{e}_i \boldsymbol{e}_j = 2\mu \boldsymbol{\varepsilon}^{\mathrm{edev}} + 3K \boldsymbol{\varepsilon}^{\mathrm{evol}}. \tag{2.27}$$

The dissipative stress coupled to the plastic strain is computed analogously to the one-dimensional situation through

$$\boldsymbol{\sigma}_{\mathrm{d}} = -\frac{\partial \psi}{\partial \boldsymbol{\varepsilon}^{\mathrm{p}}} = \boldsymbol{\sigma}, \tag{2.28}$$

and the micro hardening stress is computed as

$$\kappa = -\frac{\partial \psi}{\partial k} = -Hk. \tag{2.29}$$

Now equation (2.4) leads to

$$\dot{\boldsymbol{\varepsilon}}^{\mathrm{p}} = \lambda \frac{\partial}{\partial \boldsymbol{\sigma}} (\sigma_{\mathrm{M}} - (\sigma_{\mathrm{y}} + \kappa)) = \lambda \frac{3}{2} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}}, \quad \dot{k} = -\lambda, \tag{2.30}$$

with the conditions $\lambda \geq 0$ and $\Phi\lambda = 0$. Equation (2.30) shows that the change in the plastic strain is purely deviatoric. By the exact same reasoning as for the one dimensional case $\lambda = 0$ when $\Phi \neq 0$. It also still holds that plastic loading requires that

$$\dot{\Phi} = \frac{\partial \Phi}{\partial \boldsymbol{\sigma}} : \dot{\boldsymbol{\sigma}} + \frac{\partial \Phi}{\partial \kappa} \dot{\kappa} = \frac{3}{2} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} : (2\mu(\dot{\boldsymbol{\varepsilon}}^{\mathrm{dev}} - \dot{\boldsymbol{\varepsilon}}^{\mathrm{p}}) + 3K \dot{\boldsymbol{\varepsilon}}^{\mathrm{vol}}) - H\lambda = 0. \tag{2.31}$$

Equation (2.31) can be rewritten as

$$\frac{3}{2} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} : (2\mu \dot{\boldsymbol{\varepsilon}}^{\mathrm{dev}} + 3K \dot{\boldsymbol{\varepsilon}}^{\mathrm{vol}} - 2\mu\lambda \frac{3}{2} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}}) - H\lambda = 0 \Rightarrow \lambda = \frac{3\mu}{(3\mu + H)} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} : \dot{\boldsymbol{\varepsilon}}, \tag{2.32}$$

where the fact that $\boldsymbol{\sigma}_{\mathrm{dev}} : \dot{\boldsymbol{\varepsilon}}^{\mathrm{vol}} = \boldsymbol{\sigma}_{\mathrm{dev}} : (\frac{1}{3} \boldsymbol{I} \dot{\boldsymbol{\varepsilon}} : \boldsymbol{I}) = 0$ and $\boldsymbol{\sigma}_{\mathrm{dev}} : \dot{\boldsymbol{\varepsilon}}^{\mathrm{dev}} = \boldsymbol{\sigma}_{\mathrm{dev}} : \dot{\boldsymbol{\varepsilon}}$ has been used. The complete equations of evolution for isotropic J_2-hardening plasticity in terms of $\dot{\boldsymbol{\varepsilon}}$ then becomes

$$\begin{aligned}
&\text{Elastic loading/unloading } (\Phi < 0): \ \dot{\boldsymbol{\varepsilon}}^{\mathrm{p}} = 0, \quad \dot{\boldsymbol{\sigma}} = 2\mu \dot{\boldsymbol{\varepsilon}}^{\mathrm{dev}} + 3K \dot{\boldsymbol{\varepsilon}}^{\mathrm{vol}}, \quad \dot{k} = 0 \\
&\text{Plastic loading } (\Phi = 0): \ \dot{\boldsymbol{\varepsilon}}^{\mathrm{p}} = \frac{9\mu}{2(3\mu + H)} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} : \dot{\boldsymbol{\varepsilon}}, \\
&\dot{\boldsymbol{\sigma}} = 2\mu \dot{\boldsymbol{\varepsilon}}^{\mathrm{dev}} + 3K \dot{\boldsymbol{\varepsilon}}^{\mathrm{vol}} - \frac{9\mu^2}{(3\mu + H)} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} : \dot{\boldsymbol{\varepsilon}}, \quad \dot{k} = -\frac{3\mu}{(3\mu + H)} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} : \dot{\boldsymbol{\varepsilon}}.
\end{aligned} \tag{2.33}$$

2.2 Micro-mechanics and Homogenization

Micro-mechanics is the study of heterogeneous materials where the microscopic structure of the material is taken into account. On the macro-level the material may seem to behave as a homogeneous material, but if the micro-structure is resolved the interplay between the material constituents becomes apparent. Examples could be the grain structure of steel, or the micro-structure of short fiber reinforced composites. By utilizing the fundamental solution of isotropic elasticity, it is possible to describe the mechanics of a microscopic inclusion in an otherwise homogeneous material. The mechanical response of one inclusion may later be generalized to yield the response of a general particulate micro-structure. Mean-field homogenization aims to describe the properties of the material through an averaging scheme of the properties of the individual constituents. After homogenizing a composite material the micro-structure does not need to be resolved in the constitutive equations explicitly, and the material may be treated as a solid continuum.

2.2.1 The Eshelby Tensor

Imagine an infinite homogeneous isotropic elastic medium that contains a subregion which changes its shape or size, for example due to a phase change or thermal expansion. The Eshelby tensor is a mechanical tensor that relates the strain the subregion would posses if it was unconstrained by the rest of the medium to the elastic strain of the constrained subregion. The problem can be further generalized to let the subregion posses different elastic properties to its parent medium. The tensor was first introduced by J.D Eshelby in his to composite mechanics legendary 1957 paper: *The determination of the elastic field of an ellipsoidal inclusion, and related problems* [13].

Let the infinite medium have the linear elastic constitutive tensor $\boldsymbol{C}_0$, and let the subregion have the surface Γ with outward facing unit normal $\boldsymbol{n}$ and surface element dS. Consider an imaginary loading cycle acting on the subregion, hereby called the inclusion. To begin with the inclusion that has undergone a change is extracted from the infinite medium, hereby referred to as the matrix, by making a virtual cut. Since the inclusion now is unconstrained by the matrix, it will be subject to a constant strain $\boldsymbol{\varepsilon}^{\text{eig}}$ referred to as the *eigenstrain* (for example free thermal expansion). By removing the inclusion the matrix becomes stress- and strain-free. In the next step of the cycle, the inclusion is restored to its actual shape by applying a surface traction $\boldsymbol{t}$ on its boundary, that gives rise to an elastic strain that is equal but opposite to the eigenstrain. The strain inside the inclusion is now $\mathbf{0}$ and the stress, only dependent on the elastic strain, is

$$\boldsymbol{\sigma} = -\boldsymbol{C}_0 : \boldsymbol{\varepsilon}^{\text{eig}} = -\boldsymbol{\sigma}_{\text{eig}}. \tag{2.34}$$

The stress $\boldsymbol{\sigma}_{\text{eig}}$ in Equation (2.34) is referred to as the *eigenstress*. The traction on the surface can then be expressed as $\boldsymbol{t} = -\boldsymbol{\sigma}_{\text{eig}} \cdot \boldsymbol{n}$. Next, the inclusion is reattached to the matrix. Finally, the inclusion is restored to its original state by applying an opposite body force per unit volume $\boldsymbol{F}$ such that $\boldsymbol{F}dV = -\boldsymbol{t}dS$ inside the now combined matrix and inclusion. A schematic of the imagined cycle can be seen in Figure 2.3. The applied traction gives rise to the *constrained* strain and stress fields in the matrix denoted by $\boldsymbol{\varepsilon}^{\text{c}}$ and $\boldsymbol{\sigma}_{\text{c}}$. Equally, the principle of superposition carries with it that the constrained fields are superimposed on pre-existing strains and stresses inside the inclusion. It is now possible to express the complete strain and stress state of the matrix and the

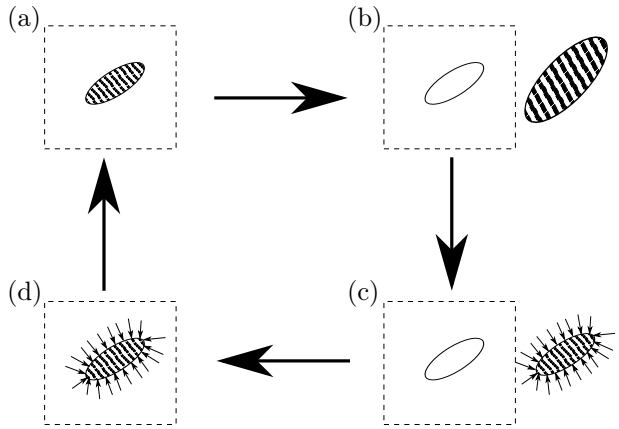


Figure 2.3: The imaginary loading cycle is displayed, beginning in the schematic (a). The first step of the cycle displays an imaginary extraction of the inclusion. Unconstrained by the matrix the inclusion takes a constant *eigenstrain* $\boldsymbol{\varepsilon}^{\text{eig}}$, as seen in (b). In the second step a traction $\boldsymbol{t}$ is applied to the boundary of the inclusion as to restore it to its original shape, which is displayed in (c). In the third step, which is shown in (d), the inclusion is reattached to the matrix. The final step entails applying an equal but opposite force to the combined inclusion and matrix thus restoring the system to its original state as seen in (a).

inclusion as

$$\begin{aligned} &\text{Matrix: } \boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{\text{c}},\ \boldsymbol{\sigma} = \boldsymbol{\sigma}_{\text{c}} \\ &\text{Inclusion: } \ \boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{\text{c}}, \boldsymbol{\sigma} = \boldsymbol{\sigma}_{\text{c}} - \boldsymbol{\sigma}_{\text{eig}}. \end{aligned} \tag{2.35}$$

The displacement of an arbitrary point $\boldsymbol{x}$ can be stated by using the fundamental solution of isotropic elasticity of a point force applied in the point $\boldsymbol{x'}$

$$G_{jk}(\boldsymbol{x}, \boldsymbol{x'}) = \frac{1}{4\pi\mu|\boldsymbol{x} - \boldsymbol{x'}|} g_{jk} - \frac{\mu + \lambda}{8\pi(2\mu + \lambda)\mu} |\boldsymbol{x} - \boldsymbol{x'}|_{,jk}, \tag{2.36}$$

and by taking the integral

$$u_i(\boldsymbol{x}) = \int_{-\infty}^{\infty} G_{ij}(\boldsymbol{x}, \boldsymbol{x'}) b^j(\boldsymbol{x'}) d\boldsymbol{x'}. \tag{2.37}$$

For a derivation of the fundamental solution refer to Appendix (C). Since the applied traction $-\boldsymbol{t} = \boldsymbol{\sigma}_{\text{eig}} \cdot \boldsymbol{n}$ is non-zero only on the boundary Γ between the inclusion and the matrix, the integral turns into the surface integral

$$u_i^{\text{c}}(\boldsymbol{x}) = \int_{\Gamma} \left(\frac{1}{4\pi\mu|\boldsymbol{x} - \boldsymbol{x'}|} g_{ij} - \frac{\mu + \lambda}{8\pi(2\mu + \lambda)\mu} |\boldsymbol{x} - \boldsymbol{x'}|_{,ij} \right) \sigma_{\text{eig}}^{jk} n_k dS', \tag{2.38}$$

where u_i^{c} is the displacement field giving rise to the constrained strain through

$$\varepsilon_{ij}^{\text{c}} = \frac{1}{2}(u_{i,j}^{\text{c}} + u_{j,i}^{\text{c}}). \tag{2.39}$$

By using Gauss' divergence theorem, the fact that the derivative with respect to $\boldsymbol{x}$ is equal to the negative derivative with respect to $\boldsymbol{x'}$ for both the functions, and by utilizing that $\boldsymbol{\sigma}^{\text{eig}}$ is constant,

Equation (2.38) can be rewritten as

$$u_i^{\mathrm{c}}(\boldsymbol{x}) = \frac{\mu+\lambda}{8\pi(2\mu+\lambda)\mu}\sigma_{\mathrm{eig}}^{jk}\left(\int_\Omega |\boldsymbol{x}-\boldsymbol{x}'|dV'\right)_{,ijk} - \frac{1}{4\pi\mu}g_{ij}\sigma_{\mathrm{eig}}^{jk}\left(\int_\Omega \frac{1}{|\boldsymbol{x}-\boldsymbol{x}'|}dV'\right)_{,k}, \tag{2.40}$$

where Ω is the domain of the inclusion. By naming the first integral I and the second integral J and by rewriting ${\sigma^{\mathrm{eig}}}^{jk} = C_0^{jkmn}\varepsilon_{mn}^{\mathrm{eig}}$ the displacement gradient becomes

$$u_{i,j}^{\mathrm{c}}(\boldsymbol{x}) = \frac{\mu+\lambda}{8\pi(2\mu+\lambda)\mu}C_0^{klmn}\varepsilon_{mn}^{\mathrm{eig}}I_{,ijkl} - \frac{1}{4\pi\mu}g_{il}C^{klmn}\varepsilon_{mn}^{\mathrm{eig}}J_{,kj}. \tag{2.41}$$

The constrained strain easily follows

$$\varepsilon_{ij}^{\mathrm{c}} = \left(\frac{\mu+\lambda}{8\pi(2\mu+\lambda)\mu}I_{,ijkl} - \frac{1}{8\pi\mu}(g_{il}J_{,jk}+g_{jl}J_{,ik})\right)C^{klmn}\varepsilon_{mn}^{\mathrm{eig}}. \tag{2.42}$$

It becomes apparent that there is a fourth order tensor that maps the eigenstrain on the constrained strain. This is the *Eshelby* tensor

$$\varepsilon_{ij}^{\mathrm{c}} = \mathcal{S}_{ij}^{mn}\varepsilon_{mn}^{\mathrm{eig}}. \tag{2.43}$$

It is clear that $\boldsymbol{\mathcal{S}}$ is minor symmetric, it is however not major symmetric. Eshelby showed that if the inclusion is an ellipsoid, the strain- and stress fields are uniform inside the inclusion. Additionally an ellipsoid inclusion results in a closed form solution for $\boldsymbol{\mathcal{S}}$. The integrals in Equation (2.42) can be be rewritten on a form where closed form solutions exist, where explicit expressions for the components of $\boldsymbol{\mathcal{S}}$ for ellipsoidal inclusions has been stated by *Mura* [14].

2.2.2 Equivalent Inclusions and the Strain Concentration Tensor

Consider another inclusion inside an infinite matrix. This time the inclusion has elastic properties $\boldsymbol{C}$ that is different to the the stiffness $\boldsymbol{C}_0$ of the matrix. A constant traction is applied in the far field of the matrix giving rise to the far field stress $\boldsymbol{\sigma}_0$, and strain $\boldsymbol{\varepsilon}^0$. Is there a possibility to determine the stress and strain state inside the inclusion? If the inclusion is swapped for an inclusion with stiffness $\boldsymbol{C}_0$ and an imagined eigenstrain $\boldsymbol{\varepsilon}^{\mathrm{eig}}$, it may be possible to pick an $\boldsymbol{\varepsilon}^{\mathrm{eig}}$ such that the strain and stress state matches the one of the inclusion with stiffness $\boldsymbol{C}$. In that case the solution may be stated in terms of the known Eshelby tensor. Since the solid has constant stiffness in the case of the swapped inclusion, the solution can be viewed as the superposition of the far field stress and the constrained stress state from the previous section. The stress inside the swapped inclusion using Equation (2.43) becomes

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{\mathrm{c}} - \boldsymbol{\sigma}_{\mathrm{eig}} + \boldsymbol{\sigma}_0 = \boldsymbol{C}_0 : ((\boldsymbol{\mathcal{S}} - \boldsymbol{\mathcal{I}}) : \boldsymbol{\varepsilon}^{\mathrm{eig}} + \boldsymbol{\varepsilon}^0), \tag{2.44}$$

where $\boldsymbol{\mathcal{I}}$ is the fourth order identity tensor. In the original problem the stress inside the inclusion of stiffness $\boldsymbol{C}$ is the superposition of the far field stress and some disturbance due to the jump in stiffness

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_0 + \boldsymbol{\sigma}_{\mathrm{d}} = \boldsymbol{C} : \boldsymbol{\varepsilon}. \tag{2.45}$$

The strain inside the swapped inclusion, using the Eshelby tensor, is

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^0 + \boldsymbol{\mathcal{S}} : \boldsymbol{\varepsilon}^{\mathrm{eig}}. \tag{2.46}$$

The strain inside the original inclusion is like for the stress, the far field contribution and a disturbance

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^0 + \boldsymbol{\varepsilon}^{\mathrm{d}}. \tag{2.47}$$

If the imagined eigenstrain, called an *equivalent eigenstrain*, is picked adequately the stress and the strain states inside the inclusion must be identical for both situations. By equating Equation (2.46) and (2.47) it immediately follows that the strain disturbance must be equal to the constrained strain from the modified problem. For ellipses the strain and stress states are constant inside the inclusion. As consequence, since $\boldsymbol{\varepsilon}_0$ is constant, $\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{\mathrm{d}} + \boldsymbol{\varepsilon}^0 = \boldsymbol{\mathcal{S}} : \boldsymbol{\varepsilon}^{\mathrm{eig}} + \boldsymbol{\varepsilon}^0$ must be constant inside the inclusion. It is therefore possible to express the strain inside the original inclusion as a tensor relation

$$\boldsymbol{\varepsilon} = \boldsymbol{\mathcal{A}} : \boldsymbol{\varepsilon}^0, \tag{2.48}$$

where $\boldsymbol{\mathcal{A}}$ is fourth order tensor called the *strain concentration tensor*. The equivalence of the strain states further requires that $\boldsymbol{\mathcal{A}} : \boldsymbol{\varepsilon}^0 = \boldsymbol{\varepsilon}^0 + \boldsymbol{\mathcal{S}} : \boldsymbol{\varepsilon}^{\mathrm{eig}}$, so the correct choice of $\boldsymbol{\varepsilon}^{\mathrm{eig}}$ is

$$\boldsymbol{\varepsilon}^{\mathrm{eig}} = \boldsymbol{\mathcal{S}}^{-1} : (\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{I}}) : \boldsymbol{\varepsilon}^0. \tag{2.49}$$

For this equivalent eigenstrain to be valid, the stresses must also be identical in the original and modified problem. This is ensured by equating Equations (2.44) and (2.45)

$$\boldsymbol{C} : \boldsymbol{\mathcal{A}} : \boldsymbol{\varepsilon}^0 = \boldsymbol{C}_0 : ((\boldsymbol{\mathcal{S}} - \boldsymbol{\mathcal{I}}) : \boldsymbol{\varepsilon}^{\mathrm{eig}} + \boldsymbol{\varepsilon}^0) = \boldsymbol{C}_0 : (\boldsymbol{\mathcal{A}} + \boldsymbol{\mathcal{S}}^{-1} - \boldsymbol{\mathcal{S}}^{-1} : \boldsymbol{\mathcal{A}}) : \boldsymbol{\varepsilon}^0. \tag{2.50}$$

Equation (2.50) can be rewritten as

$$((\boldsymbol{C} - \boldsymbol{C}_0) + \boldsymbol{C}_0\boldsymbol{\mathcal{S}}^{-1}) : \boldsymbol{\mathcal{A}} : \boldsymbol{\varepsilon}^0 = \boldsymbol{C}_0 : \boldsymbol{\mathcal{S}}^{-1} : \boldsymbol{\varepsilon}^0, \tag{2.51}$$

which in turn is rewritten as

$$(\boldsymbol{\mathcal{S}} : \boldsymbol{C}_0^{-1} : (\boldsymbol{C} - \boldsymbol{C}_0) + \boldsymbol{\mathcal{I}}) : \boldsymbol{\mathcal{A}} : \boldsymbol{\varepsilon}^0 = \boldsymbol{\varepsilon}^0. \tag{2.52}$$

The only way for equation (2.52) to hold is if the two fourth order tensors on the left hand side are each others inverses. The strain concentration tensor then follows as

$$\boldsymbol{\mathcal{A}} = \left(\boldsymbol{\mathcal{I}} + \boldsymbol{\mathcal{S}} : \boldsymbol{C}_0^{-1} : (\boldsymbol{C} - \boldsymbol{C}_0)\right)^{-1}. \tag{2.53}$$

The complete strain and stress response of the inclusion with stiffness $\boldsymbol{C}$ due to the applied far field traction then follows by insertion into Equations (2.45) and (2.47).

2.2.3 Mean Field Homogenization

The Eshelby solution, and subsequently the equivalent inclusion method, gives an exact closed form stress-strain relation for a single ellipsoidal inclusion perfectly bonded to an infinite matrix. A natural next step is to study materials with many inclusions, but then the assumptions of Eshelby no longer hold. In order to find analytical descriptions of general composite materials consisting of many inclusions mean field approaches need to be utilized.

From here on index M denotes a matrix property, index I denotes an inclusion property and lack of index denotes a property of the entire material. The total volume of the composite is given by the sum of the matrix- and inclusion volumes as

$$V = V_{\mathrm{M}} + V_{\mathrm{I}}. \tag{2.54}$$

The volume fraction v of a constituent is defined as the ratio between the constituent volume to the total volume. It follows directly from Equation (2.54) that the volume fractions must adhere to

$$v_{\mathrm{M}} + v_{\mathrm{I}} = 1. \tag{2.55}$$

The average of a quantity f over a volume V is defined by

$$\overline{f} = \langle f \rangle = \frac{1}{V} \int_V f dV. \tag{2.56}$$

If the volume V is divided into n subvolumes, the linearity of the integral implies that Equation (2.56) takes the form

$$\langle f \rangle = v_1 \overline{f}_1 + ... v_n \overline{f}_n, \tag{2.57}$$

where $\overline{f}_1, ..., \overline{f}_n$ are the averages over the respective subvolumes. In other words, the average is the sum of averages of the different constituents weighted by their respective volume fractions.

The goal of mean field homogenization is to relate the average stress of a material to its average strain, i.e

$$\overline{\boldsymbol{\sigma}} = \overline{\boldsymbol{C}} : \overline{\boldsymbol{\varepsilon}}. \tag{2.58}$$

By adequately choosing the volume to average over, the micro-structure can be viewed as a homogeneous continuum on the macro scale. It is said that the material has been homogenized. Refer to Figure 2.4 for a visualization. In order to be able to explicitly state a homogenized stiffness

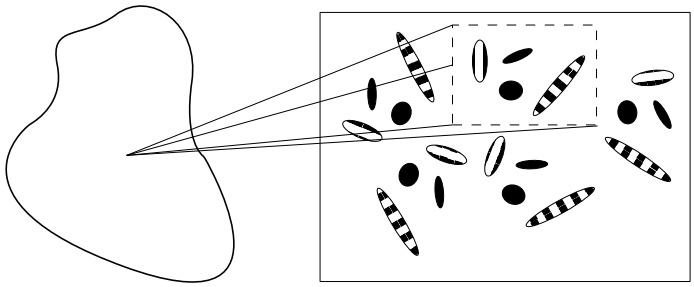


Figure 2.4: A representative volume element that captures the micro-structure of the material is shown as a dashed box. The material properties on the macro level can be computed point-wise by averaging over such elements.

$\overline{\boldsymbol{C}}$, certain assumptions on the mean stress and/or strain fields needs to be made. The two most simple models are the *Voigt*- and *Reuss* estimates. The Voigt estimate assumes that the mean strain is constant throughout the composite. The homogenized stiffness tensor of a composite with n constituents then simply becomes

$$\overline{\boldsymbol{C}} = \boldsymbol{C}_1 + ... + \boldsymbol{C}_n. \tag{2.59}$$

On the other hand, the Reuss estimate assumes that the mean stress is constant. Under that assumption the stiffness tensor is given by

$$\overline{\boldsymbol{C}} = \left(\boldsymbol{C}_1^{-1} + ... + \boldsymbol{C}_n^{-1} \right)^{-1} \tag{2.60}$$

The accuracy of these estimates is very poor, however it turns out that the Voigt and Reuss estimates will give upper and lower bounds respectively of the stiffness [15]. It is possible to make assumptions that lead to more sophisticated and accurate homogenized stiffness tensors. One such model is the *Mori-Tanaka* model.

2.2.4 Mori-Tanaka Theory

Consider a composite material consisting of an isotropic matrix and elliptical inclusions. The inclusions and matrix are perfectly bonded. The inclusions have identical shape, size, and material properties, but their orientation varies. *Mori* and *Tanaka* [16] showed that the average stress inside the matrix $\overline{\boldsymbol{\sigma}}_{\mathrm{M}}$ is uniform, under the assumption that there is no direct interaction between the inclusions but only through the mean fields. *Benveniste* [17] realized that it is possible to reformulate the Mori-Tanaka result as a direct relationship between the mean strain of one of the inclusions and the matrix mean strain. Under the assumption of no interaction between the inclusions, the constant average strain in the matrix carries with it that the average strain inside one arbitrarily picked inclusion can be related to the average matrix strain through the procedure of Section 2.2.2. As the average strain is constant in the matrix, it must be equal to the matrix far field strain. As a consequence

$$\overline{\boldsymbol{\varepsilon}}^{\mathrm{I}} = \boldsymbol{\mathcal{A}} : \overline{\boldsymbol{\varepsilon}}^{\mathrm{M}}. \tag{2.61}$$

The inclusion may be picked arbitrarily and therefore any result that follows must apply to all the inclusions. The average strain over the entire material (including the inclusions) is equal to the far field strain $\overline{\boldsymbol{\varepsilon}} = \boldsymbol{\varepsilon}^0$ since it is assumed that the average of the disturbances of all inclusions cancel. A new strain concentration tensor $\boldsymbol{A}$ that relates the inclusion average strain to the total average strain is introduced

$$\overline{\boldsymbol{\varepsilon}}^{\mathrm{I}} = \boldsymbol{A} : \boldsymbol{\varepsilon}^0 = \boldsymbol{A} : \overline{\boldsymbol{\varepsilon}}. \tag{2.62}$$

The total average strain can be expressed in terms of the matrix and inclusion averages

$$\overline{\boldsymbol{\varepsilon}} = v_{\mathrm{M}} \overline{\boldsymbol{\varepsilon}}^{M} + v_{\mathrm{I}} \left\langle \overline{\boldsymbol{\varepsilon}}^{\mathrm{I}} \right\rangle, \tag{2.63}$$

where the average of the mean inclusion strain is computed over all inclusion orientations. Combining Equations (2.61) and (2.63) results in

$$\overline{\boldsymbol{\varepsilon}} = (v_{\mathrm{M}} \boldsymbol{\mathcal{I}} + v_{\mathrm{I}} \langle \boldsymbol{\mathcal{A}} \rangle) : \overline{\boldsymbol{\varepsilon}}^{\mathrm{M}}. \tag{2.64}$$

By inverting the tensor in Equation (2.64) and inserting Equation (2.61) again, comparison with Equation (2.62) yields a relationship between the strain concentration tensors

$$\boldsymbol{A} = \boldsymbol{\mathcal{A}} : (v_{\mathrm{M}} \boldsymbol{\mathcal{I}} + v_{\mathrm{I}} \langle \boldsymbol{\mathcal{A}} \rangle)^{-1}. \tag{2.65}$$

The stiffness tensor that relates the total average strain to the total average stress can be expressed as [15]

$$\overline{\boldsymbol{C}} = \boldsymbol{C}_{\mathrm{M}} + v_{\mathrm{I}} (\boldsymbol{C}_{\mathrm{I}} - \boldsymbol{C}_{\mathrm{M}}) : \langle \boldsymbol{A} \rangle. \tag{2.66}$$

A closed expression of the homogenized stiffness tensor is then finally found by combining Equations (2.65) and (2.66)

$$\overline{\boldsymbol{C}} = \boldsymbol{C}_{\mathrm{M}} + v_{\mathrm{I}} (\boldsymbol{C}_{\mathrm{I}} - \boldsymbol{C}_{\mathrm{M}}) : \langle \boldsymbol{\mathcal{A}} \rangle : (v_{\mathrm{M}} \boldsymbol{\mathcal{I}} + v_{\mathrm{I}} \langle \boldsymbol{\mathcal{A}} \rangle)^{-1}. \tag{2.67}$$

It remains to compute the average over inclusion orientations. In order to achieve this an *orientation tensor* is introduced.

2.2.5 Orientation Tensor

Imagine a matrix material with many ellipsoid inclusions of different orientations. Let the composite material be dilute, i.e the volume fraction of inclusions is small enough to neglect inclusions from interacting with each other. Furthermore the inclusions are restricted to being long and thin and possess cylindrical symmetry.

Define the vector $\boldsymbol{p}$ as the unit vector aligned with an inclusion. In Cartesian coordinates

$$\boldsymbol{p} = \sin\theta\cos\varphi\boldsymbol{e}_1 + \sin\theta\sin\varphi\boldsymbol{e}_2 + \cos\theta\boldsymbol{e}_3, \tag{2.68}$$

where θ is the angle measured from the axis aligned with $\boldsymbol{e}_3$ and φ is the angle in the plane spanned by $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ measured from the axis aligned with $\boldsymbol{e}_1$. Refer to Figure 2.5 for a visualization. The

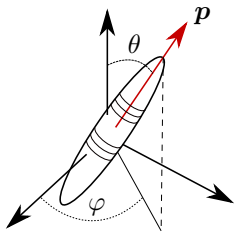


Figure 2.5: The vector $\boldsymbol{p}$ for an ellipsoidal inclusion is shown.

fiber orientation distribution $\psi(\boldsymbol{p}(\varphi,\theta))$ is defined as the probability of finding an inclusion with a direction between $\boldsymbol{p}$ and $\boldsymbol{p}+d\boldsymbol{p}$. As every fiber must have some orientation, ψ must satisfy the normality condition

$$\int_0^{2\pi}\int_0^{\pi}\psi(\varphi,\theta)d\theta d\varphi = \oint\psi(\boldsymbol{p})d\boldsymbol{p} = 1. \tag{2.69}$$

Additionally, an inclusion is indistinguishable if it is flipped. That is, if $\boldsymbol{p}$ is swapped for $-\boldsymbol{p}$. Therefore ψ must be π periodic for single axis rotations of $\boldsymbol{p}$ on the unit sphere and can thus be expanded in a Fourier series of spherical harmonics of even degree. It can be shown that it in turn is possible to rewrite the Fourier series expansion of ψ as a series of the deviatoric part of the even powers of $\boldsymbol{p}$ [18]. The Fourier coefficients of this series expression is given by the deviatoric part of the so called *orientation tensors*. The orientation tensor of rank $2n$ ($n = 1, 2, 3, ...$) is defined by

$$\boldsymbol{a} = \oint \boldsymbol{p}...\boldsymbol{p}\psi(\boldsymbol{p})d\boldsymbol{p}, \tag{2.70}$$

where there are $2n$ factors of $\boldsymbol{p}$ in the integral.

Advani and *Tucker* introduced the concept of orientation averaging [19]. The orientation average of a tensor $\boldsymbol{T}$ is defined by

$$< \boldsymbol{T} >= \oint \boldsymbol{T}(\boldsymbol{p})\psi(\boldsymbol{p})d\boldsymbol{p}. \tag{2.71}$$

Advani and Tucker showed that the orientation average of a symmetric tensor of even rank is fully determined by the corresponding orientation tensor of similar rank. This is due to the fact that

$\boldsymbol{T}$ can, similarly to ψ, be expanded as a series in terms of powers of $\boldsymbol{p}$, where the orthogonality of Fourier basis functions may be exploited. It is thereby enough to determine the orientation tensors of appropriate rank to enable the computation of the average over inclusion orientations.

2.3 Machine Learning

Machine learning, or more particularly, supervised machine learning, is the study of using vast amounts of data to train computers to make classifications or predictions. Machine learning has recently exploded in popularity and is now used in an array of fields. The theoretical framework of the artificial neural networks that constitute contemporary machine learning has been understood for many years, but the large computational cost thereof has made widespread application unfeasible. However, in the present day, powerful Graphical Processing Units (GPU) that can perform many parallel computations are readily available. This has enabled the rapid growth of the field of machine learning. The underlying mode of operation of artificial neural networks is simple, and is an excellent example of an emergent algorithm.

2.3.1 Artificial Neurons

The artificial neuron is the fundamental building block of an artificial neural network. Its mode of operation is inspired by the neurons of the human brain. Every artificial neuron has a number of inputs and an output which in turn connects to other neurons. The output signal is decided by studying the sum of the inputs. The neuron of the brain is thought to fire only if a certain threshold potential is reached at its input. The threshold potential is modeled in the artificial neuron by letting one of the inputs be a fixed value called a *bias*. To determine if the artificial neuron fires or not, the sum z of the bias and the inputs is fed through an *activation function*. One of the most common activation functions is the *logistic* function

$$f(z) = \frac{1}{1 + e^{-z}}. \tag{2.72}$$

The logistic function in Equation (2.72) takes values between 0 and 1. The derivative of the logistic function is easily found to be

$$\frac{df(z)}{dz} = f(z)(1 - f(z)). \tag{2.73}$$

If the bias is a large negative value, the logistic function will return values close to zero as long as the sum isn't large enough. In the analogy of the human brain, the neuron won't fire if the total signal from the connecting input neurons is smaller than the threshold potential. A schematic of an artificial neuron can be seen in Figure 2.6.

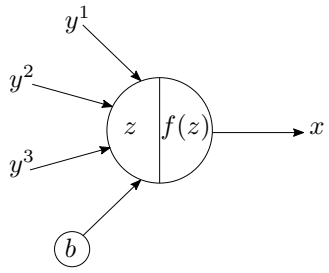


Figure 2.6: A schematic of an artificial neuron with 4 inputs (of which one is a bias), and one output. The output signal $x = f(z)$ is determined through the sum of the inputs $z = y^1 + y^2 + y^3 + b$ passed through the logistic activation function f.

2.3.2 Feed Forward Neural Networks

An Artificial Neural Network (ANN) consists of many connected artificial neurons. One of the simplest, but very useful, network structures is the Feed Forward Artificial Neural Network (FFANN). The FFANN consists of an ordered layer structure where outputs from the neurons of one layer makes up the inputs of the subsequent layer's neurons, hence the name feed forward. The first layer is the input layer and consists of n_{in} neurons. Following the input layer, N_{h} so called hidden layers follow. Each hidden layer has $n_1, ..., n_{N_{\text{h}}}$ number of neurons respectively. The final layer is the output layer and has n_{out} number of neurons. Each neuron of any one layer is connected to every neuron of the following layer. Every connection in the network is supplied with a weight. The weights scale the respective outputs before they are being fed as input to the following neurons. The final component that makes up the basic structure of the network is the bias signals. There is a set of bias signal between every layer of the network. Every set of biases connects to the neurons of the layer directly following it, and is added to the inputs. As an example, a simple network structure with $n_{\text{in}} = 4$, $N_{\text{h}} = 2$, $n_1 = 3$, $n_2 = 3$, and $n_{\text{out}} = 2$ is shown in Figure 2.7.

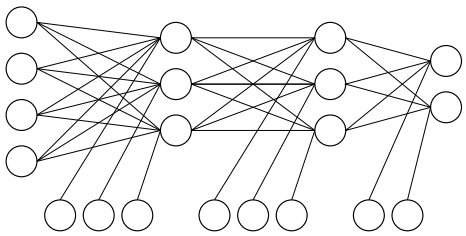


Figure 2.7: The structure of a simple Feed Forward Artificial Neural Network (FFANN) is displayed. This network has an input layer with four neurons, two hidden layers with three neurons each, and an output layer with two neurons. There are also bias signals being introduced between every layer. The input signal enters on the left side and the output signal exits on the right side of the diagram.

The output from every neuron in every layer is described with a single variable. The information can be presented even more compactly if every layer is represented by a vector where the length is the number of neurons in that layer. If the weights between every pair of subsequent layers are

gathered in matrices and the biases are gathered as vectors, the structure of the network allows the feed-forward of signals to be treated as repeated matrix-vector multiplications and additions. After every multiplication followed by a summation, the resulting vector is furthermore passed through the activation function. The weight matrices and vectors of neural networks are often incorrectly referred to as tensors. The matrices and vectors in the basic formulation of neural networks doesn't necessarily transform in the required way under a change of coordinates. It would be more correct to call them arrays, but index notation may be used to describe them nonetheless.

When using index notation, and the Einstein summation convention, to describe the network it is important to note that the range of the indices is not always the same. Let the vector representing the input layer be denoted by x_0^i, then $i = 1, ..., n_{\text{in}}$. On the other hand, the vector representing the output layer is x_{out}^j, and $j = 1, ..., n_{\text{out}}$. The remaining layers are represented by the vectors $x_1^i, ..., x_{N_\text{h}}^i$ and i ranges between 1 and $n_1, ..., n_{N_\text{h}}$ respectively. Let w_{0j}^i be the matrix of weights connecting the input layer and the first hidden layer and let b_0^i be the bias signal. The input to the first hidden layer is then given by

$$y_1^i = w_{0j}^i x_0^j + b_0^i, \tag{2.74}$$

and the first layers output follows as

$$x_1^i = f(y_1^i) = f(w_{0j}^i x_0^j + b_0^i), \tag{2.75}$$

where f is the logistic activation function. Let $w_{k'j}^i$ be the matrix connecting the k'th and $k'+1$th layers[1], and let $b_{k'}^i$ be the corresponding vector of biases. The inputs and outputs of the hidden layers then follow the form

$$y_{k'+1}^i = w_{k'j}^i x_{k'}^j + b_{k'}^i \quad \text{and} \quad x_{k'+1}^i = f(y_{k'+1}^i), \tag{2.76}$$

where $k' = 1, ..., N_\text{h} - 1$. Finally, the matrix connecting the last hidden layer and the output layer is $w_{N_\text{h}j}^i$ and the bias is $b_{N_\text{h}}^i$. The network output is thus computed as

$$x_{\text{out}}^i = y_{\text{out}}^i \quad \text{where} \quad y_{\text{out}}^i = w_{N_\text{h}j}^i x_{N_\text{h}}^j + b_{N_\text{h}}^i. \tag{2.77}$$

Note that there is no activation function for the output layer, which would constrict the components of x_{out}^i between 0 and 1. This choice is made to enable the output signal to span any range of outputs.

It has been shown by *Cybenko* [20] that an FFANN with one hidden layer, containing a sufficient amount of neurons, and logistic activation functions can approximate any n-dimensional continuous function with support in the unit n-hypercube, i.e. the function is only 0 on a set of measure 0 inside the unit hypercube. This fact is often called the *Universal approximation theorem*. Recent mathematical developments have resulted in a similar result but for networks with a sufficient number of hidden layers each containing a set amount of neurons [21]. These types of networks, with more than one hidden layer, are what is referred to as Deep Neural Networks (DNN). The consequences of the approximation theorems is that it is possible to map any input on any chosen output. This enables the network to make data classification. There is however an issue with how to chose all the weights and biases to achieve the desired mapping. Luckily, it is possible to train a network with arbitrary weights that is making incorrect predictions. By training the network it learns how to adjust the weights and biases to make accurate predictions.

[1] It is important to note that k' is not an index in the index-notation sense. It is distinguished from other indices by the prime symbol.

2.3.3 Training Feed Forward Neural Networks

Consider an FFANN as stated in the previous section where all the weights and biases have some initial arbitrary values. Let $\hat{x}^i_{\text{out}}$ be the desired output for a given input x^i_0. However, since the weights and biases are arbitrary, the actual output x^i_{out} most certainly differs from the desired output. The error-, or cost function, is defined by

$$C = (\hat{x}^i_{\text{out}} - x^i_{\text{out}})(\hat{x}^j_{\text{out}} - x^j_{\text{out}})g_{ij}, \tag{2.78}$$

and measures how much the calculated output differs from the correct output. Here g_{ij} is a positive definite symmetric matrix of weights inspired by the metric tensor, and allows to put emphasis on the importance of certain entries of the output. If the accuracy of all outputs are equally important, g_{ij} simply is the identity matrix. Since the square of the error is computed, $C \geq 0$. Therefore if one were to minimize C the actual output would be as close as possible to the desired one.

For any fixed input, the cost function C is a function of all the weights and biases of the entire network. Accordingly, the error is minimized if C is minimized with respect to $w^i_{0j}, b^i_0, w^i_{1j}, b^i_1, ..., w^i_{N_{\text{h}}j}$, and $b^i_{N_{\text{h}}}$. The naive approach would to simply solve for $\text{grad}_{w,b}C = 0$ by Newton iteration. However the data vectors of ANN are usually very large and would result in enormous data structures of second derivatives which simply is not feasible to store. As an alternative, gradient descent methods are employed.

The change in the cost function with respect to the weights $w^i_{N_{\text{h}}j}$ is computed through the chain rule

$$\frac{\partial C}{\partial w^k_{N_{\text{h}}l}} = -2\frac{\partial y^i_{\text{out}}}{\partial w^k_{N_{\text{h}}l}}(\hat{x}^j_{\text{out}} - x^j_{\text{out}})g_{ij}. \tag{2.79}$$

Similarly, the change with respect to the bias $b_{N^i_{\text{h}}}$ is computed as

$$\frac{\partial C}{\partial b^k_{N_{\text{h}}}} = -2\frac{\partial y^i_{\text{out}}}{\partial w^k_{N_{\text{h}}}}(\hat{x}^j_{\text{out}} - x^j_{\text{out}})g_{ij}. \tag{2.80}$$

All the remaining partial derivatives can be computed iteratively through repeated application of the chain rule. The change with respect to the weight matrix $w^i_{k'j}$ is

$$\frac{\partial C}{\partial w^k_{k'l}} = -2\frac{\partial y^i_{\text{out}}}{\partial x^{k_1}_{N_{\text{h}}}}\frac{\partial x^{k_1}_{N_{\text{h}}}}{\partial y^{l_1}_{N_{\text{h}}}}\frac{\partial y^{l_1}_{N_{\text{h}}}}{\partial x^{k_2}_{N_{\text{h}}-1}}...\frac{\partial x^{k_m}_{k'+1}}{\partial y^{l_m}_{k'+1}}\frac{\partial y^{l_m}_{k'+1}}{\partial w^k_{k'l}}(\hat{x}^j_{\text{out}} - x^j_{\text{out}})g_{ij}, \tag{2.81}$$

where $m = N_{\text{h}} - k'$. The change with respect to bias $b^i_{k'}$ is

$$\frac{\partial C}{\partial b^k_{k'}} = -2\frac{\partial y^i_{\text{out}}}{\partial x^{k_1}_{N_{\text{h}}}}\frac{\partial x^{k_1}_{N_{\text{h}}}}{\partial y^{l_1}_{N_{\text{h}}}}\frac{\partial y^{l_1}_{N_{\text{h}}}}{\partial x^{k_2}_{N_{\text{h}}-1}}...\frac{\partial x^{k_m}_{k'+1}}{\partial y^{l_m}_{k'+1}}\frac{\partial y^{l_m}_{k'+1}}{\partial b^k_{k'}}(\hat{x}^j_{\text{out}} - x^j_{\text{out}})g_{ij}. \tag{2.82}$$

It is thereby clear that if one has computed the change with respect to $w^i_{k'j}$ and $b^i_{k'}$, the change with respect to $w^k_{(k'-1)l}$ and $b^k_{k'-1}$ can be found by computing just 4 new derivatives. Specifically

$$\frac{\partial y^{k_{m+1}}_{k'+1}}{\partial x^{l_{m+1}}_{k'}}, \quad \frac{\partial x^{l_{m+1}}_{k'}}{\partial y^{k_{m+2}}_{k'}}, \quad \frac{\partial y^{k_{m+2}}_{k'}}{\partial w^k_{(k'-1)l}}, \quad \text{and} \quad \frac{\partial y^{k_{m+2}}_{k'}}{\partial b^k_{k'-1}}. \tag{2.83}$$

The individual factors in the chains of derivatives above are easily evaluated. The derivative of the activation may be computed by using Equation (2.73):

$$\frac{\partial x^i_{k'}}{\partial y^j_{k'}} = \frac{\partial f(y^i_{k'})}{\partial y^j_{k'}} = f(y^{i'}_{k'})(1 - f(y^{i'}_{k'}))\delta^i_j = x^{i'}_{k'}(1 - x^{i'}_{k'})\delta^i_j. \tag{2.84}$$

Here a slight deviation is made from the established index notation. Namely, the index i' in Equation (2.84) always takes the same value as the index i of the Kronecker delta. All the remaining factors follow the forms

$$\frac{\partial y^i_{k'+1}}{\partial x^j_{k'}} = w^i_{k'j}, \quad \frac{\partial y^i_{k'+1}}{\partial w^k_{k'l}} = \delta^i_k x^l_{k'}, \quad \text{and,} \quad \frac{\partial y^i_{k'+1}}{\partial b^j_{k'}} = \delta^i_j. \tag{2.85}$$

The workflow for calculating all the weight and bias derivatives at their current value is called *Backpropagation*. A forward pass is first made by letting the input signal pass through the network, and all the x-vector components are computed. Thereafter the derivatives are propagated backwards through the network following the procedure established in Equations (2.80) through (2.85).

Once the gradient $\nabla_{w,b}C$ is computed, it is possible to take a step in a gradient descent scheme. One of the simplest available methods is to simply change all the w^i_j and b^i in the gradient direction. A parameter α, called the *learning rate*, is introduced. The updated weights and biases become

$$w^i_{\text{new}j} = w^i_{\text{old}j} - \alpha\ \nabla_w(C_{\text{old}})^i_j \quad \text{and} \quad b^i_{\text{new}} = b^i_{\text{old}} - \alpha\ \nabla_b(C_{\text{old}})^i. \tag{2.86}$$

It's possible to iterate on this scheme until the parameters converge and C is minimized. However, there is a major issue. All the steps in the algorithm stated in this section build on choosing a fixed x_{in} with a corresponding $\hat{x}_{\text{out}}$. Minimizing the error doesn't necessarily imply that the error is minimized for other inputs. The network might be *over-fit* in order to ensure that x_{in} maps to $\hat{x}_{\text{out}}$, an analogy would be using an nth degree polynomial as a curve fit for n data points. The remedy is to modify the cost function to account for several different x_{in} with corresponding $\hat{x}_{\text{out}}$.

Let N_{data} be the total number of available input vectors x^i_{in} with corresponding desired outputs $\hat{x}^i_{\text{out}}$. The data points can further be divided into N_{batch} *batches* of size $N = N_{\text{data}}/N_{\text{batch}}$. The cost function is modified to be the average of the batch cost functions:

$$C(w_0, b_0, w_1, b_1, ..., w_{N_{\text{h}}}, b_{N_{\text{h}}}) = \frac{1}{N}\sum_{n=1}^{N} C_n, \tag{2.87}$$

where C_n is the cost function for one individual data point in the current batch. The gradient of this modified cost function follows directly from the previously derived gradient due to the linearity of the derivative. The iteration scheme is changed into the following algorithm:

$$\begin{aligned} &1: \quad \text{Compute gradient of } C \text{ at the current value of } w^i_j \text{ and } b^i \text{ for the first batch.} \\ &2: \quad \text{Take a gradient descent step according to Equation (2.86).} \\ &3: \quad \text{Repeat 1 and 2 for all remaining batches.} \end{aligned} \tag{2.88}$$

Running the algorithm presented in Equation (2.88) one time (one gradient step per batch) is called training the network for one *epoch*. The network may need to be trained for many epochs before the cost function, and subsequently the prediction error, is satisfyingly small.

The optimization algorithm gradient descent is quite primitive and has several shortcomings. One major issue is the fact that it quite easily converges to local minima instead of the global minimum. A possible remedy for convergence to early local minima is to add momentum to the gradient. Simply put, the actual gradient step is calculated by averaging the current gradient with the gradient of one or several of the previous steps. By possesing momentum the gradient more easily avoids local minima by making it less sensitive to small local fluctuations. Additionally, momentum can mitigate to rapid gradient updates due to noisy data. A very popular momentum algorithm is *ADAM* [22]. Using ADAM can increase the probability of convergence to an appropriate minimum. Another technique that can improve the prospects of convergence is the normalization of input data. Convergence is usually faster if the average of the input variables is close to zero and that the input variance is close to one [23]. Therefore it might be beneficial to pre-process the input data by subtracting the input mean and dividing by the input variance. The normalization of the inputs may be carried out using the population statistics. A more novel and possibly superior method is to introduce a few extra trainable parameters and normalize the inputs batch-wise [24].

2.3.4 Network Hyperparameters

In the previous sections, several parameters that are not directly trainable with the gradient descent method have been introduced. Some examples are the number of hidden layers N_h, the number of nodes per layer $n_1, ..., n_{N_\mathrm{h}}$, and the learning rate α. These parameters are referred to as *hyper-parameters* which add a new layer of complexity in network optimization. A lot of contemporary research is dedicated to finding out how to choose hyperparameters optimally. Having said that, hyperparameters are very often picked heuristically and a considerable amount of valid network structures are discovered through trial and error.

2.3.5 Recurrent Neural Networks

The architecture of the FFANN described in Sections 2.3.2 and 2.3.3 has one major limitation. Any given network is designed and trained for a fixed number of input- and output neurons. Ideally, some time dependent phenomenon could be modeled by an FFANN with $N \cdot N_T$ input neurons, where N is the number of variables that are fed to the network and N_T is the number of time steps. However, this network would not be able to predict what happens at time step $N_T + 1$, or even predicting the behavior for a time series shorter than N_T since that would require a different number of input neurons. For many areas of application the phenomenon that is to be predicted has a changing number of time steps. Examples could be stock market predictions where the amount of available market data grows with time, or path-dependent plasticity with different possible loading conditions. One solution is introducing ANN with internal variables that update every time data is fed to the network. Such networks are called Recurrent Neural Networks (RNN).

A layer of an RNN is very similar to those of FFANN. The major difference is that it has two inputs that vary with time. One input is the regular input of the FFANN, the second input is however dependent on the output of the previous time step. Let an index inside square brackets $[t]$ denote the time step, where $t = 1$ is the first time step. A new hidden state variable $h^i_{k'[t]}$ for the k'th hidden layer and time step $[t]$ is introduced. As before, let $w^i_{k'j}$ be the matrix of weights connecting the k'th and the $k'+1$th hidden layers, and let $b^i_{k'}$ be the bias between the k'th and $k'+1$th layers. In addition, let $u^i_{k'j}$ be the matrix of weights connecting the k'th layer in the current time step to

the previous time step. While $u^i_{k'j}$ connects layers through time it is independent of time itself. The input, output, and updated hidden state variable of such a layer is then given by

$$x^i_{k'+1[t]} = h^i_{k'[t]} = f(y^i_{k'+1[t]}) \quad \text{and} \quad y^i_{k'+1[t]} = w^i_{k'j}x^j_{k'[t]} + u^i_{k'j}h^j_{k'[t-1]} + b^i_{k'}. \tag{2.89}$$

Here $k' = 0, 1, ..., N_\text{h} - 1$. The weights and biases following the input layer corresponds to $k' = 0$. The weights and biases connecting the final hidden layer to the output is in a conventional RNN chosen to have no dependence on a hidden state, but more novel implementations may use recurrent output layers [25]. The activation before the output layer is furthermore chosen as the identity function like for the FFANN. This is to allow the components of the output vector to span an arbitrary range. The output of the RNN is given by

$$x^i_{\text{out}[t]} = y^i_{\text{out}[t]} = w^i_{N_\text{h}j}x^j_{N_\text{h}[t]} + b^i_{N_\text{h}}. \tag{2.90}$$

The hidden state of acting as input in the first time step i.e. $h^i_{k'[0]}$, is most often chosen as the zero vector.

The function f in Equation (2.89) is generally not chosen as the logistic function due to the vanishing gradient problem, which will be explained in more detail later. As an alternative activation the hyperbolic tangent is used, which is defined by

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.91}$$

The hyperbolic tangent has a similar shape to the logistic, but has an output range of -1 to 1. The derivative of the hyperbolic tangent is

$$\frac{df(x)}{dx} = 1 - f(x)^2. \tag{2.92}$$

The derivative of the hyperbolic tangent typically has a larger magnitude than the derivative of the logistic activation function. This motivates its use in RNN, as one of the countermeasures to the vanishing gradient problem.

The weights and biases of an RNN is like for a regular FFANN decided by minimizing a cost function. However, the minimization problem needs to be modified to account for the fact that the outputs of previous time steps influence the future predictions. This modified procedure is called Backpropagation Through Time (BPTT) [26].

2.3.6 Training Recurrent Neural Networks

The RNN as stated here takes an input, and gives an output for every time step. The total output will be time series data. As the desired output also is a time series, the cost function needs to be formulated as the average of the discrepancies at every time step. Let $\hat{x}^i_{[t]}$ be the desired output at time t and let N_T be the total number of time steps (), the cost function for one time series is then given by

$$C = \frac{1}{N_T}\sum_{t=1}^{N_T} C_{[t]} = \frac{1}{N_T}\sum_{t=1}^{N_T}(\hat{x}^i_{\text{out}[t]} - x^i_{\text{out}[t]})(\hat{x}^j_{\text{out}[t]} - x^j_{\text{out}[t]})g_{ij}. \tag{2.93}$$

To investigate how the error changes with the adjustment of the weights and biases of the network, backpropagation through time is employed. The derivative of the cost function with respect to the weight $w^i_{k'j}$ is found through the chain rule as before. However, every instance of $y^i_{k'+1[t]}$ not only has a dependence on the previous layers through $x^i_{k'[t]}$ but also on the previous time step through $h^i_{k'[t-1]}$. The derivatives are branching. The derivatives with respect to $u^i_{k'j}$ and $b^i_{k'}$ are found analogously and are not stated here explicitly.

The change in the cost functions $C_{[t]}$ due to the change in $w^i_{N_\mathrm{h}j}$ and $b^i_{N_\mathrm{h}}$ is computed the same way as in the FFANN case through equations (2.79) and (2.80). The derivatives with respect to some arbitrary layer's weights $w^i_{k'j}$ however displays a more complicated behavior. The first branching follows as

$$\frac{\partial C_{[t]}}{\partial w^k_{k'l}} = -2\frac{\partial y^i_{\mathrm{out}[t]}}{\partial x^{k_1}_{N_\mathrm{h}[t]}}\frac{\partial x^{k_1}_{N_\mathrm{h}[t]}}{\partial y^{l_1}_{N_\mathrm{h}[t]}}\left(\frac{\partial y^{l_1}_{N_\mathrm{h}[t]}}{\partial x^{k_2}_{N_\mathrm{h}-1[t]}}\frac{\partial x^{k_2}_{N_\mathrm{h}-1[t]}}{\partial w^k_{k'l}} + \frac{\partial y^{l_1}_{N_\mathrm{h}[t]}}{\partial h^{k_2}_{N_\mathrm{h}-1[t-1]}}\frac{\partial h^{k_2}_{N_\mathrm{h}-1[t-1]}}{\partial w^k_{k'l}}\right)(\hat{x}^j_{\mathrm{out}[t]} - x^j_{\mathrm{out}[t]})g_{ij}. \tag{2.94}$$

The derivatives inside the parentheses are expanded as

$$\frac{\partial x^{k_2}_{N_\mathrm{h}-1[t]}}{\partial w^k_{k'l}} = \frac{\partial x^{k_2}_{N_\mathrm{h}-1[t]}}{\partial y^{l_2}_{N_\mathrm{h}-1[t]}}\left(\frac{\partial y^{l_2}_{N_\mathrm{h}-1[t]}}{\partial x^{k_3}_{N_\mathrm{h}-2[t]}}\frac{\partial x^{k_3}_{N_\mathrm{h}-2[t]}}{\partial w^k_{k'l}} + \frac{\partial y^{l_2}_{N_\mathrm{h}-1[t]}}{\partial h^{k_3}_{N_\mathrm{h}-2[t-1]}}\frac{\partial h^{k_3}_{N_\mathrm{h}-2[t-1]}}{\partial w^k_{k'l}}\right), \tag{2.95}$$

and as

$$\frac{\partial h^{k_2}_{N_\mathrm{h}-1[t-1]}}{\partial w^k_{k'l}} = \frac{\partial h^{k_2}_{N_\mathrm{h}-1[t-1]}}{\partial y^{l_2}_{N_\mathrm{h}[t-1]}}\left(\frac{\partial y^{l_2}_{N_\mathrm{h}[t-1]}}{\partial x^{k_3}_{N_\mathrm{h}-1[t-1]}}\frac{\partial x^{k_3}_{N_\mathrm{h}-1[t-1]}}{\partial w^k_{k'l}} + \frac{\partial y^{l_2}_{N_\mathrm{h}[t-1]}}{\partial h^{k_3}_{N_\mathrm{h}-1[t-2]}}\frac{\partial h^{k_3}_{N_\mathrm{h}-1[t-2]}}{\partial w^k_{k'l}}\right). \tag{2.96}$$

The derivative splits into two branches at every layer. The first branch is the recognized derivative of regular backpropagation. The second branch is new to RNN. It is a backpropagation through time. It is evident that the first type of branching keeps occurring until the k'th layer is reached, and that second type repeats until $[t] = [0]$. Once the derivatives pertaining one set of $w^i_{k'j}$ are computed the derivatives for $b^i_{k'}$ follow by just computing a few additional derivatives. However, the amount of derivatives that needs to be evaluated increase by a multiplication of t. Similarly, once the $w^i_{k'j}$ derivative of the cost function $C_{[t]}$ is known the derivative of $C_{[t+1]}$ is received by the additional evaluation of the derivatives along the first type of branching only. The newly presented BPTT scheme can be summarized as follows: Start backpropagating $C_{[1]}$ and then use previous information as the derivatives for following $C_{[t]}$ are computed.

Identically to the FFANN case, the cost function needs to be some type of average of the available data to yield representative results. The procedure of dividing the desired outputs (now time series) into batches is the same as stated previously. That is, the cost function in Equation (2.93) is computed for all points in a batch and averaged through the scheme presented in Equation (2.87). The length of the different time series N_T making up the data do not need to have the same length. Once the cost function is received gradient descent can be carried out as presented in Equation (2.88), with the modification that the gradient with respect to the $u^i_{k'j}$ is furthermore computed. Due to the rapid branching in the computation of the RNN cost function derivatives it becomes evident that RNN are computationally expensive. Therefore, the number of RNN layers are usually kept to a minimum. The network doesn't need to be shallow though, as regular time independent FFANN layers can be inserted into an RNN.

An issue that optimization of RNN faces is the vanishing gradient problem. The derivative of the activation function is less than 1, so repeated differentiation through the chain rule tends toward zero. For an RNN that is propagating through many time steps this problem makes training the network very difficult. This problem applies to both the logistic function and the hyperbolic tangent, but the derivative of the hyperbolic tangent tends to be of greater magnitude and therefore a little bit less affected. As a way to circumvent the issue of vanishing gradients the RNN architecture is modified to allow for derivative truncation. The idea is to let the network only pass information a finite distance in time before *forgetting* it. A successful implementation of a mechanism enabling forgetting information is Long Short Term Memory (LSTM) networks introduced by *Hochreiter* and *Schmidhuber* [27], and later refined by *Gers* and *Cummins* [28]. The LSTM architecture introduces a new hidden state which keeps track of which information is relevant for long term behavior. By forgetting things that have no influence on long term behavior, the predicament of vanishing gradients is mitigated. Another RNN implementation that remedies the vanishing gradient problem is the Gated Recurrent Unit (GRU) introduced by *Cho et al.* [29]. The GRU contains two gate functions that determine how much of the information contained in the hidden state from previous time steps remains, and how much new information from the current time step is introduced respectively. The GRU implementation is computationally quicker than the LSTM one, however it is difficult to know which one suits a certain task better. In general both methods need to be tried.

3 Methodology

The method adopted in this project consists of three parts. The first part entails generating a large number of stress-strain curves that are representative of the mechanical response of the composite being studied. One pair of composite constituents is chosen for modeling, while the volume fraction of fibers and their orientation is left as free variables used for prediction. The second part consists of designing an artificial neural network, and training said network to predict the macroscopic stress inside a composite material given the strain history. The final part comprises the validation of the created model on a representative group of loading conditions. The first and third part of the work contains all the physical considerations, while the second part is purely data-scientific. The first part contains the material modeling considerations and the final part encompasses investigating the viability of the generated model by considering the physicality of model responses. The composite materials investigated in the current work are assumed to be completely rate independent, therefore all simulations are quasi-static. The strains are assumed small enough to be able to use small strain theory, i.e there is a linear relation between displacement gradient and stress for materials in their elastic regime.

3.1 Generation of Training Data

The generation of training data for the artificial neural network is divided into two main parts. The first part consists of generating strain paths that cover a representative spectrum of loading conditions. Additionally, for each strain path a second order orientation tensor with a corresponding fiber volume fraction is also generated. The second part entails in using the strain paths as input data to perform strain controlled micro-mechanical simulations of various fiber distributions using a suitable material model. The simulations result in the macroscopic stress of the investigated composites. When training the network, the generated strain paths, orientation data, and fiber volume fractions are fed as inputs, and the corresponding stress responses pose as the quantity being predicted.

3.1.1 Generation of Strain Paths

In order to be able to build a network that can learn the proper path dependency of plasticity, representative strain paths need to be generated. The paths used for training need to be varied

enough for the network to be able to recognize complicated paths without losing the ability to predict common simple situations such as uni-axial stress. As an example, if the strain fed to the network is white noise, it is very likely that the network won't consider uni-axial stress states as a possibility. Randomly sampled data is however useful in ensuring a good distribution of data. The problem transforms into generating data that is random enough to capture a large enough family of load paths without sacrificing accuracy in typical load cases. Random sampling has been used by other authors to generate adequate strain data. One example is sampling a set of random points in strain space and subsequently using an interpolator to connect the points to form paths radiating from the origin [30]. In this present work, an alternative but similar approach is taken.

First, the components of a 6-dimensional vector representing a direction in the space of independent strain are sampled independently from each other. The samples are normal distributed with mean $\mu = 0$ and standard deviation $\sigma = 1$. The vector is normalized such that its magnitude is 1. The resulting probability density function of the vector will be uniform on the 6-dimensional unit sphere. The direction of this vector is hereby referred to as the drift direction.

Secondly, a number of steps n_{step} is chosen, and an ordered set of n_{step} additional random vectors are generated in the same fashion. These new random vectors are scaled by some factor γ between 0 and 1. The additional vectors are from now on called noise vectors. As a third step, the drift direction is added to the noise vectors. The result is n_{step} randomly distributed vectors with a clear bias in the drift direction. The strain path may then be generated by taking the cumulative sum of the 6-dimensional vectors. The cumulative sum will be a time series consisting of small steps in 6-dimensional strain space.

As a final part of the algorithm, a number of drift directions n_{drift} are chosen. The previously described process of picking a drift direction, adding noise, and taking the cumulative sum is repeated n_{drift} times. The total strain path is taken as the cumulative sum of all the individual paths. The length of the time series is $(n_{\text{step}} \cdot n_{\text{drift}}) + 1$, where the first entry is the zero vector. As a normalizing measure, the largest entry of the time series, denoted by M, is identified and the entire time series is then scaled by $\varepsilon_{\max}/M$ where $\varepsilon_{\max}$ is the largest admissible strain component. In order to ensure that also uni-axial strain states may arise, an option is added to isolate one strain component before the scaling step and set all the other time series components to 0. A flowchart clarifying the steps of the generation algorithm is presented in Figure 3.1.

The presented algorithm is motivated by several considerations. To begin with, the algorithm is quite simple and can be implemented with just a few lines of code. Moreover, it allows for a large range of possible strain paths by varying the parameters, n_{step}, n_{drift}, γ, and $\varepsilon_{\max}$. By generating the strain paths from picking drift directions and adding noise, the strain paths are enabled to consist of long term trends with local fluctuations. It is thus assumed to be able to capture any type of reasonable loading path. The strain paths can thereby be generated randomly without forgoing the ability to capture behavior such as uni-axial loading. The normal components of a typical strain path generated by the presented algorithm can be seen in Figure 3.2. Since the shear components are generated identically to the normal components they are left out of the visualization for readability.

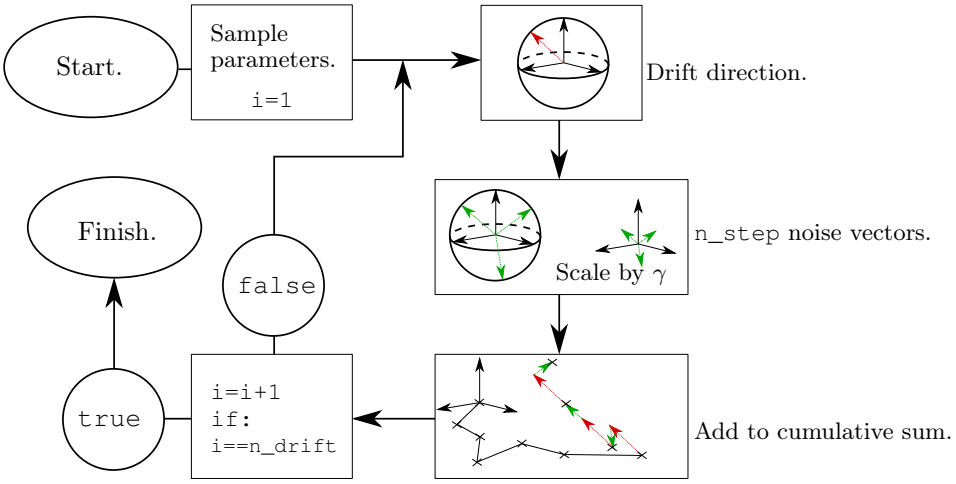
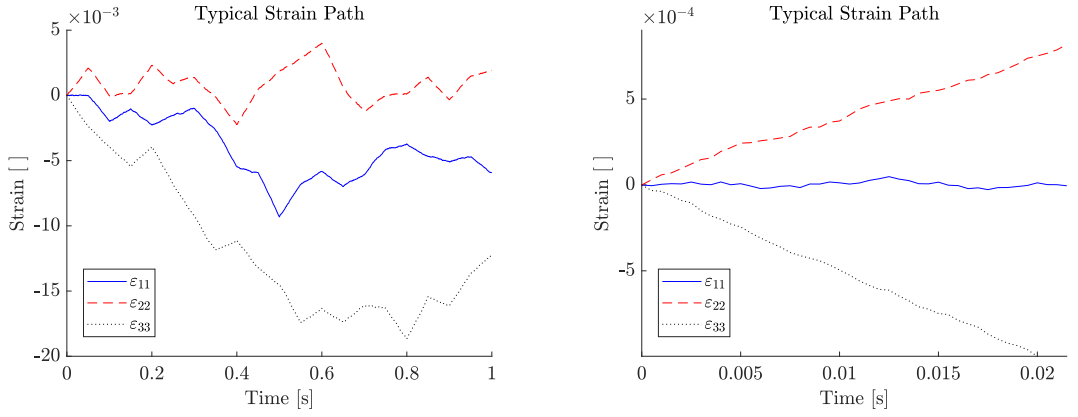


Figure 3.1: A flowchart demonstrating the proposed method of generating strain paths.



(a) Influence of drift

(b) Influence of noise

Figure 3.2: The normal strain components of a typical strain path generated according to the presented algorithm is displayed. Subfigure (a) shows the complete time evolution of the strain path and demonstrates how the drift directions influence the paths. Subfigure (b) on the other hand, which contains only the first 20 ms of the path, demonstrates the influence of the added noise.

3.1.2 Random Sampling of Orientation Tensors

The trace of the second order orientation tensor must be unity [19], and its eigenvalues must be non-negative. To uniformly sample from the set of all possible orientation tensors, one may uniformly sample triplets of eigenvalues and thereafter perform a coordinate transformation through a randomly chosen rotation. Uniform sampling of three eigenvalues between 0 and 1 that sum to 1 can be achieved by sampling uniformly from the standard 2-simplex. Algorithmically, this can be

implemented by sampling 2 points uniformly in $(0, 1)$ and using the length of the three segments these two points make up together with 0 and 1 as the three eigenvalues [31]. To ensure the generation of single axis fiber distributions a special case is added to the algorithm where one eigenvalue is set to 1 randomly while the others are set to 0. Arbitrary rotations may be sampled uniformly in a computationally efficient manner through an algorithm proposed by *Arvo* [32]. First, a single-axis rotation with the rotation matrix $\boldsymbol{R}(\theta)$ is performed, where the angle θ is sampled uniformly in $[0, 2\pi]$. Thereafter, the axis of rotation is in turn reflected to an arbitrary point on the unit sphere with uniform probability through a negatively scaled Householder transformation

$$-\boldsymbol{H} = 2\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}} - \boldsymbol{I}. \tag{3.1}$$

The vector $\boldsymbol{v}$ is given by

$$\boldsymbol{v} = \begin{bmatrix} \cos\varphi\sqrt{z} \\ \sin\varphi\sqrt{z} \\ \sqrt{1-z} \end{bmatrix}, \tag{3.2}$$

where φ is sampled uniformly from $[0, 2\pi]$ and z is sampled uniformly from $[0, 1]$. The total rotation matrix $\boldsymbol{M} = -\boldsymbol{H}\boldsymbol{R}$, which is sampled uniformly from the set of all possible rotations, is the composition of the two transformations. As the orientation tensor is of the second order, it requires two transformation tensors to achieve a change of coordinates. In matrix notation this is written as

$$\boldsymbol{a} = \boldsymbol{M}\tilde{\boldsymbol{a}}\boldsymbol{M}^{\mathrm{T}}, \tag{3.3}$$

where $\tilde{\boldsymbol{a}}$ is the matrix representation of the diagonal tensor whose entries are the uniformly sampled eigenvalues, and $\boldsymbol{a}$ is the matrix representation of the resulting uniformly sampled orientation tensor. A flowchart visualizing the steps of the generation algorithm is shown in Figure 3.3.

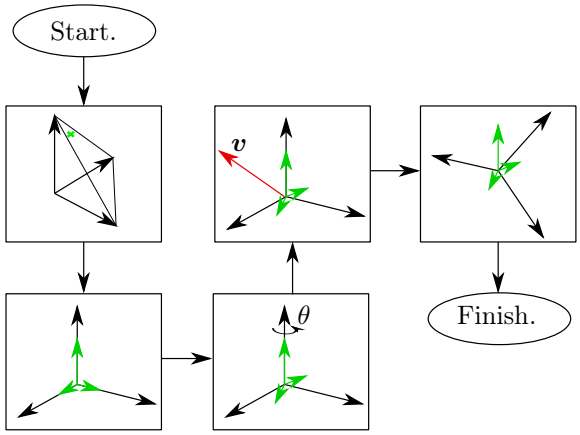


Figure 3.3: A flowchart demonstrating the proposed algorithm for uniform sampling of orientation tensors.

3.1.3 Chosen Material Model

The material chosen for modeling has linearly elastic fibers and an isotropic J_2 elasto-plastic matrix. The hardening is however of a linear exponential type. The free energy of such a material is given

by

$$\psi(\boldsymbol{\sigma}, k) = \frac{1}{2} C^{ijkl} \varepsilon^{\mathrm{e}}_{ij} \varepsilon^{\mathrm{e}}_{kl} + \frac{1}{2} H k^2 + H_\infty \left(\frac{1}{m} e^{mk} - k - \frac{1}{m} \right), \tag{3.4}$$

where H_∞ and m are new material parameters. From now on H_∞ is is referred to as the hardening modulus while H is called the linear hardening modulus. The parameter m is called the hardening exponent. The yield function is still given by

$$\Phi(\boldsymbol{\sigma}, \kappa) = \sigma_{\mathrm{M}} - (\sigma_{\mathrm{y}} + \kappa), \tag{3.5}$$

and the expressions for $\dot{k}$ and $\dot{\boldsymbol{\varepsilon}}^{\mathrm{p}}$ also remains unchanged. However, the micro hardening stress is modified and takes the appearance

$$\kappa = -\frac{\partial \psi}{\partial k} = -Hk + H_\infty \left(1 - e^{mk}\right). \tag{3.6}$$

The plastic multiplier is as before received from the plastic loading condition

$$\dot{\Phi} = \frac{\partial \Phi}{\partial \boldsymbol{\sigma}} : \dot{\boldsymbol{\sigma}} + \frac{\partial \Phi}{\partial \kappa} \dot{\kappa} = 0, \tag{3.7}$$

but since κ is modified λ takes the form

$$\lambda = \frac{3\mu}{(3\mu + H + H_\infty m e^{mk})} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} : \dot{\boldsymbol{\varepsilon}}. \tag{3.8}$$

The complete equations of evolution for the state variables become

$$\begin{aligned}
&\text{Elastic loading/unloading } (\Phi < 0): \ \dot{\boldsymbol{\varepsilon}}^{\mathrm{p}} = 0, \quad \dot{\boldsymbol{\sigma}} = 2\mu \dot{\boldsymbol{\varepsilon}}^{\mathrm{dev}} + 3K \dot{\boldsymbol{\varepsilon}}^{\mathrm{vol}}, \quad \dot{k} = 0 \\
&\text{Plastic loading } (\Phi = 0): \ \dot{\boldsymbol{\varepsilon}}^{\mathrm{p}} = \frac{9\mu}{2(3\mu + H + H_\infty m e^{mk})} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} : \dot{\boldsymbol{\varepsilon}}, \\
&\dot{\boldsymbol{\sigma}} = 2\mu \dot{\boldsymbol{\varepsilon}}^{\mathrm{dev}} + 3K \dot{\boldsymbol{\varepsilon}}^{\mathrm{vol}} - \frac{9\mu^2}{(3\mu + H + H_\infty m e^{mk})} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} : \dot{\boldsymbol{\varepsilon}}, \\
&\dot{k} = -\frac{3\mu}{(3\mu + H + H_\infty m e^{mk})} \frac{\boldsymbol{\sigma}_{\mathrm{dev}}}{\sigma_{\mathrm{M}}} : \dot{\boldsymbol{\varepsilon}}.
\end{aligned} \tag{3.9}$$

The material model has several layers of complexity which is a good stress test for the machine learning model. If the model can capture phenomena like saturation of the micro hardening stress (the exponential part vanishes when $|k|$ grows) it is a good indications that machine learning is a suitable method to accurately model complex composite materials.

In an article by *Kammoun et al.* [33], the proposed hardening law was used to accurately describe a Polyamide 6,6 matrix reinforced with short glass fibers. This composite is manufactured through injection molding which makes it relevant in the context of the current project. The values of the elastic parameters and inclusion dimensions are here taken directly from the mentioned article. The plastic material parameters however are only stated in the article as the relative values in terms of initial yield stress. The numerical value of the initial yield stress is classified information. In the present work the initial yield stress is therefore taken as $\sigma_{\mathrm{y}} = 25$ MPa which is a typical value for

the used type of material, for example see [34]. All the numerical values used for modeling can be seen in Table 3.1.

Table 3.1: The material parameters used for modeling the SFRC.

	Parameter	**Value**
Fiber	E	76 GPa
	ν	0.22
Matrix	E	3.1 GPa
	ν	0.35
	σ_y	25 MPa
	H	150 MPa
	H_∞	20 MPa
	m	325

3.1.4 Computing the Stress Response

The stress response pertaining a given strain path and fiber distribution is computed via micro-mechanical simulations. The simulations are performed using a mean-field approach where Mori-Tanaka theory, as described in Section 2.2.4, is used for homogenization. The volume averages are computed using orientation averaging via orientation tensors as presented in Section 2.2.5. The software DIGIMAT-MF [35] is utilized for the simulations.

Eshelby's theory, and by extension Mori-Tanaka theory, makes the assumption of linear elasticity. When the prerequisites for these theories are fulfilled, the strain and stress fields are uniform inside inclusions. This is not necessarily true for non-linear material models. In order to generalize the Mori-Tanaka homogenization scheme, the inclusion state variables first need to be homogenized. From there the non-linear model may be linearized and mean field homogenization can be carried out. DIGIMAT handles eventual non-uniformities in the relevant fields by introducing a comparison material at every time step. The comparison material is defined through volume averaging of the composite phases, such that the tangent operators used for solving the equations of elasto-plasticity numerically are uniform inside every phase. DIGIMAT allows for a first and second order homogenization. In first order homogenization the actual material model is used, but the volume average of the strain field is used in computing the tangent operators. Second order homogenization uses higher order information of the strain field to evaluate the comparison material. Second order homogenization shows a significant improvement in accuracy over first order homogenization when there is a large difference between matrix and inclusion stiffness and the matrix exhibits little hardening [35]. In the present work these conditions are fulfilled and second order homogenization is therefore chosen.

In DIGIMAT the material micro-structure and loading is defined by the user, and the software subsequently computes the response of the material under study. Each phase of the micro-structure is defined through picking a material model and entering the appropriate material parameters. The micro-structure is built by entering the volume fractions of the different phases and the geometry of inclusions. In the present work only one type of inclusion is used, that is, all inclusions use the

same material and geometry but may have different orientations. The inclusions are are set as an isotropic linearly elastic material and the matrix is set to an isotropic elasto-plastic (J_2-plasticity) material with an exponential linear hardening law. The geometry of the inclusions is determined by defining an ellipsoid radius and length to diameter aspect ratio. As a final step in defining the microstructure, the orientation distribution of the inclusions is determined by defining the second order orientation tensor. The fourth order orientation tensor is also required to compute the orientation average of fourth order tensors like the stiffness tensor. Since the user only enters the second order tensor (which in general is the information available after an injection molding simulation) the fourth order tensor needs to be approximated. In DIGIMAT, the fourth order orientation tensor is approximated from the second order orientation tensor via the orthotropic closure approximation following *Cintra* and *Tucker* [36]. Strain controlled loading is chosen for the simulations and is entered as a user defined strain history. After a DIGIMAT simulation is completed, each of the macroscopic stress components corresponding to the time stamps of the input strain path is saved as training data.

3.1.5 Implementation

The generation of training data is controlled via a small program written in the programming language julia [37]. The process of running DIGIMAT simulations can be automated by calling a pre-made `.bat` file through Windows PowerShell, with the path to a `.mat` file containing all the analysis parameters as an argument.

The julia program first generates the strain data and orientation tensors used as input according to the presented algorithms. The total number of steps in the strain paths are set to $N = 2000$. A pseudo-time variable is also created that starts at 0 s and ends at 1 s with N increments. The time range is completely arbitrary due to the rate independence of the plasticity, and is simply chosen to match the default settings of DIGIMAT.

The motivation for long time sequences as training data is that an LSTM network trained on long sequences would be able to make accurate predictions on shorter input sequences. This follows from the fact that the short sequences are contained in the long ones owing to the construction of the generation algorithm. However, the length of the sequences is a trade-off between generality and computational cost. The chosen number of time steps results in a Δt of 0.5 ms, which may be considered fine enough for many engineering applications. As pointed out, the mechanics are rate independent. It is therefore possible to re-scale any engineering problem to be contained between the bounds 0 and 1 s. The Δt should accordingly be interpreted as the level of dynamics in the load paths that the model can handle.

For each generated strain path the generation parameters are sampled randomly. The noise multiplicative factor γ is chosen between 0 and 1 from a uniform distribution. The number of drift directions n_{drift} is chosen uniformly from the set $(1, 2, 5, 10, 20, 25, 50, 100, 200)$, and the number of steps n_{step} per direction is subsequently chosen such that $n_{\text{drift}} \cdot n_{\text{step}} = N$. This set of number of drift directions is chosen arbitrarily, motivated only by allowing a fairly large range of path complexities, where $n_{\text{drift}} = 1$ would mean essentially a perturbed linear function and $n_{\text{drift}} = 200$ would result in a highly complex path. Furthermore, the maximum admissible strain component $\varepsilon_{\max}$ is sampled uniformly between 0.01 and 0.05. This range captures a full range of reasonable

loading conditions. The lower bound allows loading that does not lead to plasticity and the upper range of 0.05 is kept since failure is extremely likely to occur beyond that point.

For every strain path a corresponding orientation tensor is generated. The case of uni-axial fiber alignment is chosen with a probability of 10% during generation. As an additional parameter associated with the fiber distribution, the fiber volume fraction v_{F} is chosen randomly between 10 and 15%. The volume fraction is allowed to vary as volume fraction will be dependent on position after injection molding. The range of volume fraction is kept small to make training the network easier.

Finally, the strain paths and orientation tensors are written into `.mat` files together with the material parameters. All the remaining required parameters are left as their default values. The `.mat` analysis files are subsequently sent as argument to the `.bat` program through a PowerShell extension to julia. The output stress data is parsed for the data points that matches the time stamps of the input strain signal and saved. As a final measure the `.mat` files and the unprocessed solution files are deleted to conserve hard drive space. The full workflow for the generation of training data is as follows:

$$\begin{aligned} &1: \quad \text{Generate strain paths and orientation tensors} \\ &2: \quad \text{Write } \texttt{.mat} \text{ files} \\ &3: \quad \text{Run DIGIMAT simulations} \\ &4: \quad \text{Post-process and clean-up.} \end{aligned} \tag{3.10}$$

A data set with 40,000 samples is generated. Every sample consists of one strain path, one orientation tensor, and the resulting stress path. The computations are carried out on a personal computer with an 8 core 3.8 GHz processor and 16 GB of memory. The entire generation process takes about two weeks. The vast majority of the computational time is dedicated to the micro-mechanical simulations that compute the stresses. The file size of the generated data set becomes approximately 14 GB. The complete julia code used for data generation can be found in Appendix D.

3.2 Selection of Artificial Neural Network

To capture the path dependency of plasticity, RNN are utilized. Both LSTM and GRU network architectures are investigated. The input signals of the network consist of time series of length $N_T = 2001$, with $F = 13$ different features $\hat{x}^i_{\mathrm{out}[t]}$, where $i = 1, ..., F$. The features comprise the 6 independent components of the second order orientation tensor, the fiber volume fraction, and the 6 independent components of the strain tensor. The time independent features are simply fed to the network at every time step without modification, while the strain tensor components are changing with time. The output signals are the 6 independent components of the stress tensor. The cost is computed by comparing the output to the stress time series corresponding to the input signal. For the optimization of the cost function, the *ADAM* optimizer is employed. The optimizer specific parameters are left as MATLAB's default values, and are not considered as hyperparameters to be optimized in this work. The default optimizer settings are known to be good for machine learning

applications [22]. The number of epochs N_{epoch} on the other hand, is one of the hyperparameters that are optimized. A list of all the hyperparameters optimized in this work is displayed in Table 3.2.

Table 3.2: An exhaustive list of the hyperparameters that are subject to optimization is displayed.

Parameter	**Symbol**
Initial learning rate	α_0
Learning rate decay period	τ
Learning rate decay factor	γ
Number of hidden layers	N_{h}
Number of neurons per hidden layer	$n_1, n_2, ..., n_{N_{\text{h}}}$
Mini-batch size	N_{batch}
Number of epochs	N_{epoch}

Both network types were tested with a varying number of hidden layers and number of neurons per layer.

In order to improve the prospects of convergence piecewise learning rate decay is used. Three hyperparameters are introduced: the initial learning rate $\alpha_0 > 0$, the decay period $\tau \geq 1$, and the learning rate decay factor $0 < \gamma < 1$. The learning rate is made to decrease as training progresses through updating: $\alpha_{n+1} = \alpha_n \cdot \gamma$ every τ epochs. The scheduling is motivated by the following: By setting the learning rate too low from the beginning, convergence may be to slow, or the optimizer gets stuck in a local minimum early. On the other hand, setting a high learning rate might help the optimizer arrive in the vicinity of the global minimum. However, the large step size may prevent the optimizer from making the final stretch without overstepping the minimum, that may be in a small dip in an otherwise flat area of the cost function. By scheduling the learning rate one may get the best of both worlds.

The 40,000 generated data samples are randomly split into three groups; a training set, a validation set, and a test set. The training set receives 80% ($N_{\text{train}} = 32,000$) of the data, whereas the validation and test sets make up 19.75% ($N_{\text{valid}} = 7900$) and 0.25% ($N_{\text{test}} = 100$) of the data respectively. The training set is the actual data that is used for training. The training data is shuffled every epoch during training to ensure that the training process is independent of the structure of the training set.

The validation set on the other hand, is used to monitor if the network over-fits or not. The validation set is not used for training. The validation set is fed through the network every epoch, and a cost is calculated. However, backpropagation is not performed, and the weights and biases are subsequently not affected by this data. The validation set cost indicates if the model generalizes its prediction capability to data outside the training set. A validation cost that stops decreasing, or starts to increase, indicates that the network is over-fit.

The test set is used to evaluate the network performance after the training has finished. The test set does like the validation set not affect the weights and biases. The test set is intentionally kept very small, since it is only to be used as a final check up of the results before proper model testing

is carried out. The random nature of the generated paths is not representative of the model use cases, instead the model is rigorously tested on more meaningful data specifically generated for testing. The batch size N_{batch} is a very important hyperparameter that influences the convergence rate substantially. Simply taking as large mini-batch size as possible doesn't necessarily guarantee convergence to a good minimum [38]. It is therefore optimized to suit the task at hand.

The input features are subjected to z-score normalization to improve odds of convergence. Thereby, the feature mean μ_i is subtracted from the feature and the difference is divided by the feature standard deviation σ_i. The mean and standard deviation of each of the 13 features is computed individually by summing through time and over all the time series in the training set. If $\hat{x}^i_{\text{out}[t]}(n)$ is the value of the ith feature at time step t in sample number n, then

$$\mu_i = \frac{1}{N_T N_{\text{train}}} \sum_{n=1}^{N_{\text{train}}} \sum_{t=1}^{N_T} \hat{x}^i_{\text{out}[t]}(n), \quad \text{and} \quad \sigma_i = \sqrt{\frac{1}{N_T N_{\text{train}}} \sum_{n=1}^{N_{\text{train}}} \sum_{t=1}^{N_T} (\hat{x}^i_{\text{out}[t]}(n) - \mu_i)^2}. \tag{3.11}$$

As a measure to avoid numerical difficulties, the output stress signal is divided by 10^6, effectively changing the units from Pa to MPa.

The network structure begins with an input layer of time sequence data. After the input layer a number of recurrent layers follows. Networks consisting of either LSTM or GRU neurons are investigated. All the network weights and biases use the default initialization. The number of hidden layers N_{h}, and the number of neurons in the hidden layers $n_1, n_2, ..., n_{N_{\text{h}}}$ are hyperparameters that are investigated. The output layer consists of a standard feed forward layer, without an activation function, with 6 neurons. The role of this layer is to ensure that the output signal is of correct size, namely 6. In addition, the layer also scales the output to match the magnitude of the target data. To prevent overfitting a *Dropout layer* is placed between the final RNN layer and the feed forward layer at the end. A dropout layer temporarily cuts a random set of connections between its neighboring layers for one training iteration with a predetermined probability. They thus introduce noise into the network and is a simple but efficient method for preventing overfitting. The probability is chosen as 50% and is not optimized as a hyperparameter. A 50% dropout probability is empirically shown to have the best performance for many applications [39], possibly due to the fact that it maximizes the regularization effects of the layer [40].

3.2.1 Network Implementation

The computational programming language MATLAB [41] is used for the implementation of artificial neural networks. Specifically, the *Deep Learning* and *Parallel Computing* Toolboxes are employed. The program is designed to perform parallel computations on a GPU. The complete MATLAB code utilized for the network implementation is found in Appendix E.

A function first reads all the text files containing the generated data. The data is saved in cell arrays as required for use in MATLAB's deep learning routines. Every member of the input data cell array is formatted as F by N_T matrices, where the first 7 rows are the orientation tensor components and fiber volume fraction repeated N_T times. The remaining rows are the different strain components' time evolution. The output data cell array is populated by 6 by N_T matrices where every row is the stress components' time evolution. The script shuffles the data to remove dependence on file

reading order. The script furthermore divides the data into training, validation, and test sets as defined earlier. The network structure is easily defined by listing the layers in an array, where the number of neurons per layer is given as arguments. Options for training are defined through an options object. The input is defined as a sequence input layer, the output is defined as a regression output layer, and the network is configured as sequence to sequence regression by flagging the RNN layers accordingly. The cost function that MATLAB uses for sequence to sequence regression is a mean squared error:

$$C = \frac{1}{N} C_n, \quad \text{where} \quad C_n = \frac{1}{2N_T} \sum_{t=1}^{N_T} \sum_{i=1}^{F} (\hat{x}^i_{\text{out}[t]}(n) - x^i_{\text{out}[t]}(n))^2, \tag{3.12}$$

where $N = N_{\text{train}}$, N_{valid}, or N_{test} depending on the situation.

MATLAB by default uses L_2-regularization to prevent overfitting. L_2-regularization modifies the cost function such that it penalizes weights of large magnitude. The penalty term is the squared sum of all the weights multiplied by a factor λ that controls the amount of regularization. The default value of $\lambda = 0.0001$ is used in the training of the current network. The idea is that weights with large magnitude make the model more complex and reduces its ability to generalize and is therefore penalized. The MATLAB documentation suggests using gradient clipping when training RNN for regression in order to counteract exploding gradients that lead to numerical issues. Gradient clipping re-scales a gradient if the gradient norm is larger than a set threshold value. The default norm that MATLAB employs is the L_2 norm computed on each gradient individually. That is, all gradients are not scaled the same way. The gradient threshold is in the current work set to 1.

3.3 Model Testing

It is of great importance to have an understanding of the model accuracy. Future users of the model need to know the limitations on use cases in order to not draw erroneous conclusions. The measure of the model accuracy should be based on properly chosen evaluation metrics that bear some physical significance. In order to guarantee that the model accurately predicts a wide range of feasible use cases, representative loading situations need to be explored. The range of time independent inputs (orientation tensor and fiber volume fraction) that yield reasonable results also needs to be investigated. Neural network models are generally good at interpolating, but lack ability to extrapolate. It is therefore of interest to investigate how the error behaves when the parameters approaches, and crosses, the bounds of the range used for training. Moreover, in order to be widely applicable, the model needs to be functional for a wide range of sequence lengths. That is, it shouldn't only be accurate for loading of similar length to the training data. This condition is equivalent to rate independence.

Testing of the model comprises simulating the mechanical response of different material samples in DIGIMAT-MF. The strain from the simulation and the orientation tensor and fiber volume fraction of sample is subsequently used as input for the neural network model. The predicted stress is compared with the stress output from DIGIMAT-MF, and the prediction error can be computed. Four types of tests are conducted, which are presented below.

3.3.1 General Testing

General testing of the model is performed by running representative load cases against uniformly sampled fiber orientations and volume fractions within the training range. In order to test the rate independence, the general tests are also performed with varying input sequence length.

Five orientation tensors are uniformly sampled according to the previously proposed rules, and corresponding fiber volume fractions are uniformly sampled from the admissible parameter range. The six independent components of the generated tensors, together with the corresponding fiber volume fractions, are shown in Table 3.3.

Table 3.3: The second order orientation tensors and fiber volume fractions for the general testing samples are presented.

Sample	a_{11}	a_{22}	a_{33}	a_{12}	a_{13}	a_{23}	v_{F}
1	0.477	0.188	0.335	-0.080	-0.071	-0.183	0.130
2	0.094	0.692	0.214	-0.103	0.012	-0.255	0.144
3	0.649	0.139	0.212	0.011	-0.117	-0.154	0.131
4	0.392	0.225	0.382	-0.142	0.080	0.152	0.139
5	0	0.919	0.081	0.015	0.005	0.273	0.109

Five different types of loading conditions are simulated for every sample. All loading's are strain controlled load cycles where the control strains are set to change piecewise monotonically from 0 to 0.035 to -0.035 to 0, where DIGIMAT-MF computes the other strain components such that the desired stress state is achieved. However, for plane strain loading, the out of plane strain components simply are set to constant 0. The first type of loading is a uniaxial stress in the σ_{11}-direction, controlled by imposing ε_{11}. The second type of loading is a pure shear stress state in the σ_{12}-direction, controlled by imposing ε_{12}. The third type is a biaxial stress state in the σ_{11}-σ_{22}-directions, controlled by ε_{11} and ε_{22}. The fourth type of loading is a bi-axial stress state in the σ_{11}-σ_{23}-directions, controlled by ε_{11} and ε_{23}. The final loading condition is plane strain in the ε_{11}-ε_{22}-plane.

The rate independence of the model is investigated by repeating all the above described tests on the five samples with different degrees of linear interpolation/extrapolation.

3.3.2 Repeated Cyclical Loading

To test how complex the load histories can be without forgoing too much accuracy, a test with an increasing number of load cycles is performed.

Cyclical uniaxial loading of three different samples is simulated. The three investigated samples are: unidirectional fibers along the loading axis, a uniform 2D fiber distribution in the loading plane, and a uniform random 3D fiber distribution. All the samples have a fiber volume fraction of 12%. A load cycle consists of changing the strain along the load axis piecewise monotonically from 0 to 0.04 to -0.04 to 0, where DIGIMAT-MF computes the other strain components such that uni-axial stress is achieved. Tests consisting of 1 to 5 cycles are performed.

3.3.3 Testing of Extrapolation Ability

As a test of the extrapolation ability of the created model, uniaxial loads beyond the maximum strain of the training data are applied. Similarly, the volume fraction of fibers is pushed outside the training range in order to investigate the interdependence on the parameter extrapolation.

Uniaxial loading of a uniform 3D fiber distribution for volume fractions and maximum strains beyond the permissible range is simulated. Fiber volume fractions of 0.1% 2.5%, 5%, 7.5%, 10%, 12.5%, 15%, 17.5%, and 20% were investigated. For every volume fraction 3 tests consisting of one load cycle is performed, where the maximum strain ε_{M} is set to 5%, 7.5%, and 10% respectively. A load cycle consists of changing the strain along the load axis piecewise monotonically from 0 to ε_{M} to $-\varepsilon_{\mathrm{M}}$ to 0, where DIGIMAT-MF computes the other strain components such that uni-axial stress is achieved.

3.3.4 Hydrostatic Loading

The final test consists of an investigation of the model performance under hydrostatic loading conditions. This test is performed to test if the correct yield behavior has been learned by the model, i.e zero deviatoric stress results in no yielding.

The mechanical response of a uniform 3D fiber distribution with 12% fiber volume fraction is simulated. The simulation is stress controlled where the pressure is set to change monotonically starting at 0 and ending at 1 GPa.

3.3.5 Evaluation Metrics

Simply referring to the cost as an error metric doesn't give a fair estimate. A numerically small cost doesn't necessarily imply a small error if the scale of the output is small. To give the error physical significance, some dimensionless quantity needs to be introduced. To remove the dimensionality, the root mean square error is divided by the yield stress of the matrix σ_{y}. This is done for every feature individually. This metric is from here on referred to as the Mean Relative Error (MeRE). This metric gives an overall estimate of the model accuracy, it does however not account for large localized errors. As a consequence the Maximum Relative Error (MaRE) is introduced. The MaRE is defined by the maximum absolute error divided by the yield stress of the matrix σ_{y}. As for the MeRE, the MaRE is also computed over the features individually. The two error metrics of a time series of length T are computed through the equations

$$\mathrm{MeRE} = \frac{\sqrt{\frac{1}{N_T}\sum_{t=1}^{N_T}(\sigma_t - \hat{\sigma}_t)^2}}{\sigma_{\mathrm{y}}} \quad \text{and} \quad \mathrm{MaRE} = \frac{\max_t |\sigma_t - \hat{\sigma}_t|}{\sigma_{\mathrm{y}}} \tag{3.13}$$

where σ_t is the predicted stress at step t and $\hat{\sigma}_t$ is the desired stress at step t.

4 Results

The result is divided into two parts. The first part explains the design and training results of the DNN model. Contrarily, the second part describes the results concerning the model performance during elasto-plastic prediction of representative test cases.

4.1 Model Design and Training

For the problem at hand it turned out that GRU neurons outperformed LSTM neurons. The GRU neurons both displayed better convergence and numerical stability. LSTM networks had a tendency to experience GPU errors when using the entire training set on larger networks (the order of magnitude 3 hidden layers containing above 400 neurons each). The inclusion of the dropout layer assisted convergence significantly. For networks without dropout the validation cost stopped decreasing even though the training cost kept decreasing, which is a typical sign of over fitting. Normalization also proved to improve the chances of convergence.

The initial learning rate was decreased from the default value $\alpha_0 = 0.001$ until satisfyingly steady convergence was achieved. It proved to be more effective to let the learn rate decay in larger chunks rather than setting the period to $\tau = 1$ and having a slow but continuous decay. For example setting the learn rate decay factor to $\gamma = 0.9$ and the decay period to $\tau = 10$ proved more effective than setting $\gamma = 0.99$ and $\tau = 1$, even though $0.99^{10} \approx 0.9$.

As there is an interaction between the number of layers and the neurons per layer, it is impossible to investigate the effects of these hyperparameters independently from each other. A systematic optimization procedure would therefore require testing a large number of combinations of N_h, and $n_1, n_2, ..., n_{N_\mathrm{h}}$ to be certain that the network is optimal. This was deemed unfeasible for this project since the network training time was very long, and access to computational hardware was limited. From the tests that were carried out the following results were arrived at: There was a significant increase in performance when adding a second and third GRU layer. However, more than 3 GRU layers did not result in any significant performance increases in contrast to the increase in training time. Once the number of layers was decided, different configuration of the numbers of neurons were tested. The results of the test were that simply using more neurons yielded better performance. Training time was thus the limiting factor, as adding more neurons make forward- and backpropagation take more time. There was no noticeable influence of having a different number

of neurons in the different hidden layers.

Very large batch sizes often resulted in a steady decrease in validation error at first, but led to premature convergence to a sub-optimal minimum. A very small batch size on the other hand made the gradient vary too much between batches such that convergence was unfeasible. Batch sizes that gave promising results were seen in the range starting from approximately 20 and ending with a few hundred. These batch sizes yielded steady enough convergence, but with enough gradient variation to prevent getting stuck in local minima. The network did not suffer from overfitting when trained for many epochs. The only constraint imposed on the number of epochs was therefore the time it would take to train the network.

To summarize: the network structure and the hyperparameters were chosen quite arbitrarily, and performance was improved mostly through trial and error. It can not be claimed that the network architecture and set of hyperparameters that were chosen as the final result were the optimal ones. However, after many iterations of different designs, the performance was deemed good enough for the intents of this project.

4.1.1 Final Network Used as Model

The final network architecture consisted of three hidden layers with 500 GRU neurons each. This resulted in a total of 3,777,006 trainable parameters. The initial learn rate was set to $\alpha_0 = 0.0005$, the learn rate decay period was set to $\tau = 10$, and the learn rate decay factor was set to $\gamma = 0.9$. The network was trained for $N_{\text{epoch}} = 500$ epochs with a mini-batch size of $N_{\text{batch}} = 32$. The model was trained on an *Nvidia V100* GPU with 32 GB of VRAM. The training took just above 81 hours.

At the first training epoch, the model yielded a cost of 12981.8057 MPa^2 on the validation set, which reduced to a final cost of 7.4021 MPa^2 when the training session terminated. A log-log plot of the training- and validation set cost functions is displayed in Figure 4.1. It may look pathological that the validation cost becomes smaller than the training cost. It is however correct and has to do with the way MATLAB computes the cost. It computes the cost with dropout on the training data but without dropout on the validation data. If a quadratic fit ($y = 0.1542x^2 - 1.98x + 8.36$, with coefficient of determination $R^2 = 0.9905$) is utilized on the log-log data it is possible to conclude that the validation cost function, for the current set of hyperparameters, decreases like

$$C(E) = E^{(0.1542 \ln E - 1.98)} e^{8.36}, \tag{4.1}$$

where E is the epoch number. This function has a minimum at $E \approx 614$.

The average of the MeRE and MaRE is computed over the 100 time series inside the test set. The time to make a prediction was less than 1 s for all time series, with the typical time to make a prediction being around 0.1 s. The averages for all 6 stress components are displayed in Table 4.1. The errors are small, which indicates that the model doesn't suffer from overfitting and generalizes well.

Table 4.1: This table displays the average MeRE and the average MaRE computed over the test set.

	σ_{11}	σ_{22}	σ_{33}	σ_{12}	σ_{23}	σ_{13}
MeRE	0.0582	0.0528	0.0487	0.0469	0.0378	0.0431
MaRE	0.1255	0.1137	0.1121	0.1020	0.0874	0.0980

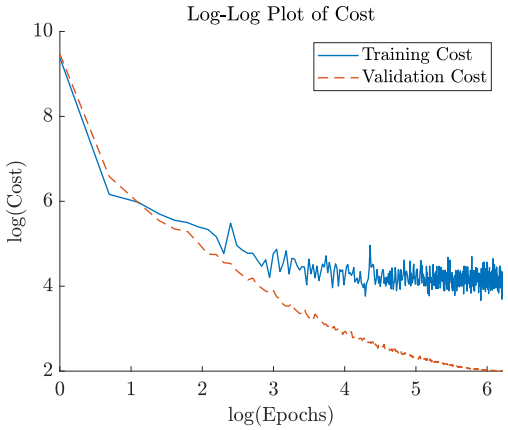


Figure 4.1: A log-log plot of the training- and validation cost functions is shown. The training cost flattens out quite early while the validation cost keeps decreasing.

4.2 Model Performance

The tests described in Section 3.3 were carried out using the network architecture presented in Section 4.1.1. The MATLAB-code used for testing and visualization is found in it's entirety in Appendix E. The results for the different test types are presented in the same order as they are introduced in Section 3.3, starting with the performance of the model when subject to representative load cases, where the rate independence of the model also is demonstrated.

4.2.1 General Testing

The resulting stress-strain curves from the tests described in Section 3.3.1 show good agreement between the network prediction and the output of DIGIMAT-MF. Two stress-strain curves representative of the results are presented in Figure 4.2. These curves highlight how the developed neural network model capture the fundamental characteristics of the underlying model well. Specifically, the model correctly displays a hardening process that saturates for large plastics strains, and the correct loading/unloading behavior.

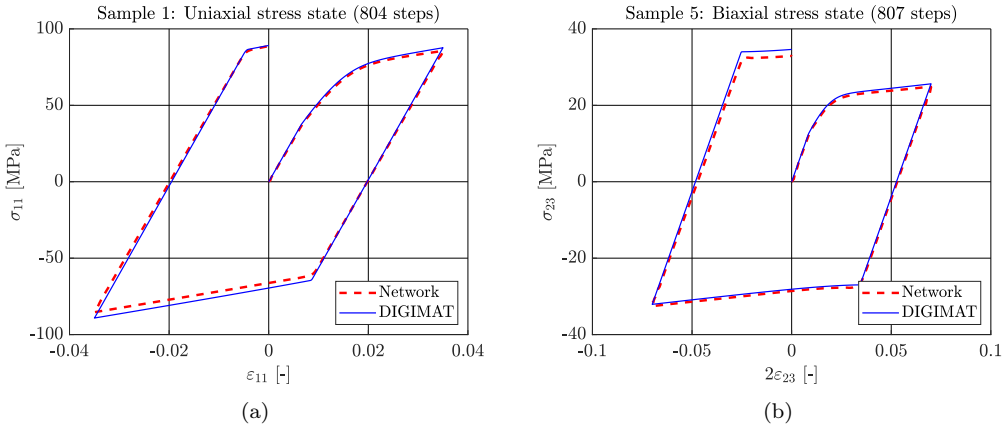


Figure 4.2: Two stress-strain curves consisting of the network prediction compared to the output of DIGIMAT-MF. Subfigure (a) shows the the results from the uniaxial test performed on sample 1, while subfigure (b) shows the results of the σ_{11}-σ_{23} biaxial test performed on sample 5.

A demonstrative stress time series of all six stress components for a plane strain test is presented in Figure 4.3. This figure also shows good agreement between the network prediction and the DIGIMAT-MF output. The prediction of the σ_{13} looks poor, however by noting the scale of the stress it is apparent that the relative error still is small.

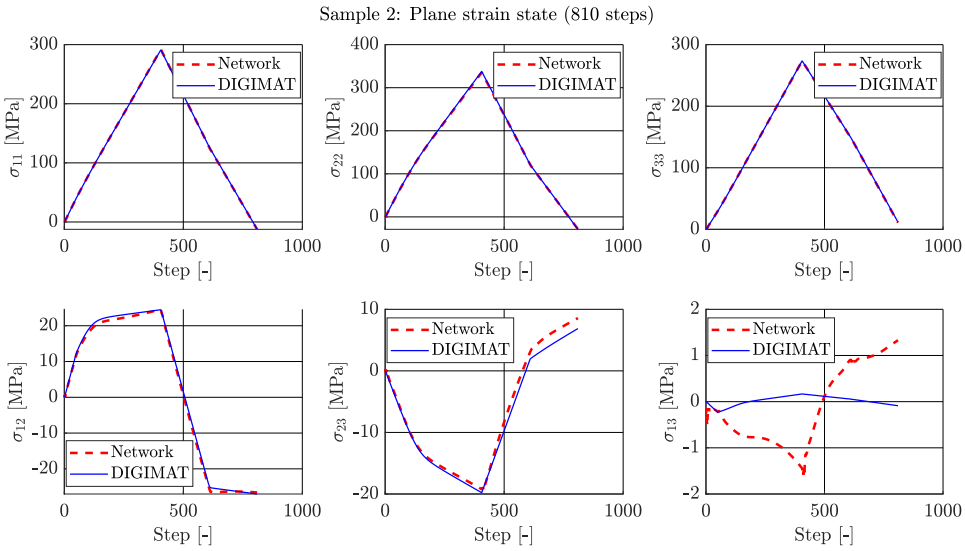


Figure 4.3: The time series data of all 6 stress components for a plane strain state load cycle on sample 5 is shown. The output of DIGIMAT is compared to the network output.

The exhaustive results of all the tests of the five samples is found in the form of the MeRE in Table 4.2 and MaRE in Table 4.3. The model shows great promise with a MeRE that never surpasses 25% of the matrix yield stress during the tests. The feature wise average MeRE is also low, less than 7% of the matrix yield stress. The largest recorded MaRE was less than 50% of the matrix yield stress, while the feature wise average was less than 15%.

Figure 4.4 shows the typical behavior of the MeRE and MaRE when the number of steps in the time series is varied. As is seen in the Figure the errors seem to stay constant in the range 200 to 20000, but quickly growing for sequences shorter or longer than these bounds. This holds true for most of the tests, but some tests showed stricter bounds for constant error. None of the performed tests showed errors that varied to a significant degree when loaded with sequences of lengths between 500 and 8000 steps.

The limits on the sequence lengths can be explained by the structure of the training data. As the algorithm that was used to generate training data has a theoretical maximum strain rate, and an average strain rate that is considerably lower than the maximum, it's reasonable that there is a minimum sequence length for which the neural network model can make accurate predictions. This explains the lower bound. Since the sequence length of the training data is fixed, the network is not trained to remember the accumulated plastic strain forever. The loss of accuracy for very long sequences can possibly be explained by the network forgetting about plastic strain accumulated early in the sequence when it approaches the sequence's end. Bearing this in mind, it may still be said that the network model displays rate independence for a large range of sequence lengths.

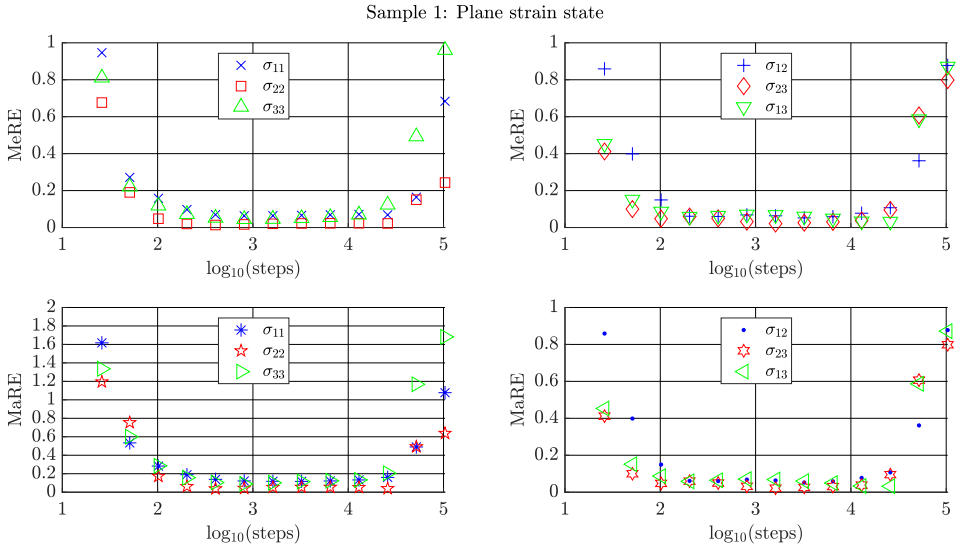


Figure 4.4: The figure displays the dependency between sequence length and the error. The horizontal axis consists of the 10-logarithm of the sequence length. The data corresponds to a plane strain load cycle on sample 1.

Table 4.2: The MeRE of every stress component for the five samples and five loading conditions. The feature wise average and maximum MeRE over all tests is shown in the bottom of the table.

Sample	**Load**	**Steps**	σ_{11}	σ_{22}	σ_{33}	σ_{12}	σ_{23}	σ_{13}
1	Uniaxial	804	0.0971	0.0301	0.0391	0.0124	0.0080	0.0155
	Pure shear	808	0.0117	0.0123	0.0177	0.0247	0.0068	0.0077
	Biaxial (σ_{11}-σ_{22})	804	0.0624	0.0510	0.0667	0.0403	0.0481	0.0297
	Biaxial (σ_{11}-σ_{23})	810	0.0955	0.0384	0.0571	0.0805	0.0553	0.1232
	Plane strain	810	0.1217	0.0424	0.0889	0.0683	0.0318	0.0709
2	Uniaxial	804	0.0308	0.0346	0.0288	0.0299	0.0273	0.0148
	Pure shear	802	0.0163	0.0258	0.0226	0.0410	0.0106	0.0329
	Biaxial (σ_{11}-σ_{22})	802	0.0612	0.0781	0.0881	0.0584	0.0456	0.0331
	Biaxial (σ_{11}-σ_{23})	812	0.0552	0.1168	0.0731	0.0443	0.1030	0.0566
	Plane strain	810	0.0143	0.0417	0.0130	0.0285	0.0440	0.0357
3	Uniaxial	808	0.1515	0.0359	0.0503	0.0144	0.0151	0.0144
	Pure shear	804	0.0279	0.0101	0.0139	0.0225	0.0167	0.0209
	Biaxial (σ_{11}-σ_{22})	803	0.1136	0.0515	0.0515	0.0250	0.0357	0.0381
	Biaxial (σ_{11}-σ_{23})	806	0.0953	0.0373	0.0457	0.0449	0.0449	0.1059
	Plane strain	810	0.1252	0.0324	0.0370	0.0792	0.0197	0.0428
4	Uniaxial	804	0.0644	0.0249	0.0265	0.0309	0.0147	0.0190
	Pure shear	805	0.0237	0.0131	0.0208	0.0415	0.0119	0.0062
	Biaxial (σ_{11}-σ_{22})	803	0.0666	0.0630	0.1209	0.0423	0.0549	0.0577
	Biaxial (σ_{11}-σ_{23})	805	0.1073	0.0624	0.0475	0.2442	0.0091	0.0806
	Plane strain	810	0.0473	0.0356	0.0255	0.0320	0.0336	0.0279
5	Uniaxial	802	0.0459	0.0301	0.0349	0.0142	0.0278	0.0079
	Pure shear	808	0.0140	0.0121	0.0220	0.0242	0.0100	0.0237
	Biaxial (σ_{11}-σ_{22})	809	0.0862	0.2027	0.1096	0.0922	0.0825	0.0347
	Biaxial (σ_{11}-σ_{23})	807	0.0417	0.0307	0.0297	0.0180	0.0377	0.0119
	Plane strain	810	0.0423	0.0570	0.0323	0.0262	0.0725	0.0134
Average			0.0648	0.0468	0.0465	0.0472	0.0347	0.0370
Maximum			0.1515	0.2027	0.1209	0.2442	0.1030	0.1232

Table 4.3: The MaRE of every stress component for the five samples and five loading conditions. The feature wise average and maximum MaRE over all tests is shown in the bottom of the table.

Sample	**Load**	**Steps**	σ_{11}	σ_{22}	σ_{33}	σ_{12}	σ_{23}	σ_{13}
1	Uniaxial	804	0.1764	0.0541	0.0843	0.0375	0.0336	0.0503
	Pure shear	808	0.0225	0.0305	0.0477	0.0614	0.0166	0.0230
	Biaxial (σ_{11}-σ_{22})	804	0.2109	0.1195	0.1520	0.0795	0.1030	0.0643
	Biaxial (σ_{11}-σ_{23})	810	0.1340	0.0980	0.1809	0.1605	0.1789	0.2246
	Plane strain	808	0.0225	0.0305	0.0477	0.0614	0.0166	0.0230
2	Uniaxial	804	0.0621	0.0706	0.0534	0.0640	0.0426	0.0394
	Pure shear	802	0.0434	0.0689	0.0434	0.0823	0.0254	0.0699
	Biaxial (σ_{11}-σ_{22})	802	0.1998	0.1842	0.2319	0.1377	0.0841	0.0963
	Biaxial (σ_{11}-σ_{23})	812	0.1140	0.2093	0.1177	0.1073	0.1977	0.1199
	Plane strain	810	0.0418	0.1026	0.0335	0.0485	0.0746	0.0709
3	Uniaxial	808	0.2652	0.0795	0.0968	0.0386	0.0370	0.0267
	Pure shear	804	0.0524	0.0309	0.0418	0.0532	0.0298	0.0397
	Biaxial (σ_{11}-σ_{22})	803	0.3613	0.1071	0.1443	0.0512	0.0738	0.0792
	Biaxial (σ_{11}-σ_{23})	806	0.1742	0.0764	0.0936	0.0974	0.1523	0.1741
	Plane strain	810	0.2292	0.0751	0.0804	0.1409	0.0357	0.0822
4	Uniaxial	804	0.1288	0.0513	0.0686	0.0523	0.0305	0.0336
	Pure shear	804	0.0525	0.0374	0.0443	0.0948	0.0224	0.0108
	Biaxial (σ_{11}-σ_{22})	803	0.2014	0.1659	0.3256	0.0822	0.1505	0.1411
	Biaxial (σ_{11}-σ_{23})	805	0.2184	0.1374	0.0942	0.4278	0.0304	0.1527
	Plane strain	810	0.0982	0.0631	0.0672	0.0580	0.0766	0.0559
5	Uniaxial	802	0.0849	0.0991	0.1197	0.0313	0.0592	0.0164
	Pure shear	808	0.0262	0.0378	0.0488	0.0700	0.0174	0.0556
	Biaxial (σ_{11}-σ_{22})	809	0.1745	0.4861	0.2483	0.1759	0.2170	0.0649
	Biaxial (σ_{11}-σ_{23})	807	0.1111	0.1050	0.0705	0.0455	0.1024	0.0250
	Plane strain	810	0.1129	0.1654	0.0640	0.0783	0.1540	0.0355
Average			0.1327	0.1074	0.1040	0.0935	0.0785	0.0712
Maximum			0.3613	0.4861	0.3256	0.4278	0.2170	0.2246

4.2.2 Cyclical Loading

The calculated MeRE corresponding to each of the six strain components as a function of the number of load cycles is displayed in Figure 4.5, while the MaRE is found in Figure 4.6. The error is found to approximately increase linearly with the number of load cycles. It can be seen that the uniaxial fiber distribution typically suffers from the greatest error. The sequence length is proportional to the number of cycles, where one cycle consists of approximately 800 steps. Since it was shown in the previous section that the error of sequences between approximately 500 and 8000 steps is not affected by changing the sequence length, it can be concluded that the error stems from the complexity of the load path. Two representative stress-strain curves from the conducted test are displayed in Figure 4.7. It is possible to see how the prediction error increases slightly for every additional cycle. It interesting to see that once yielding has occurs, the error seems to say

fairly constant. This speaks in favor of the model accurately capturing material properties such as the tangent modulus.

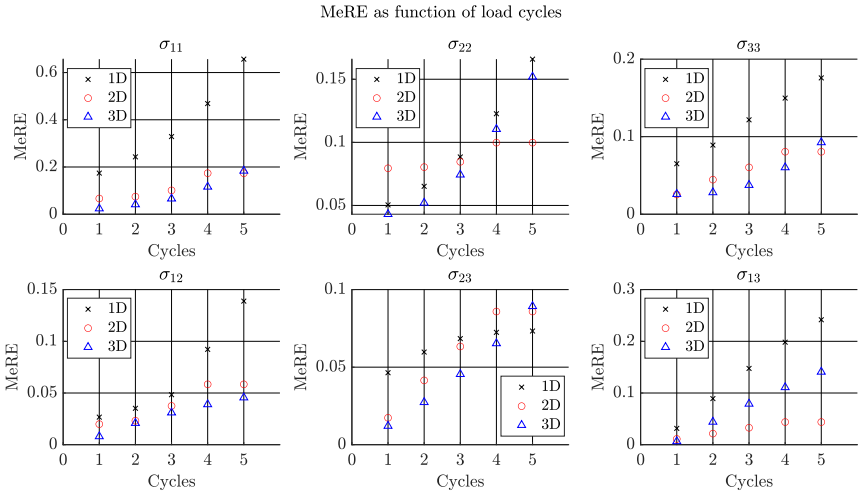


Figure 4.5: The MeRE of a cyclical test as a function of the number of cycles is presented. The component wise errors for a uniaxial fiber distribution, a uniform 2D distribution, and a uniform 3D distributions are given.

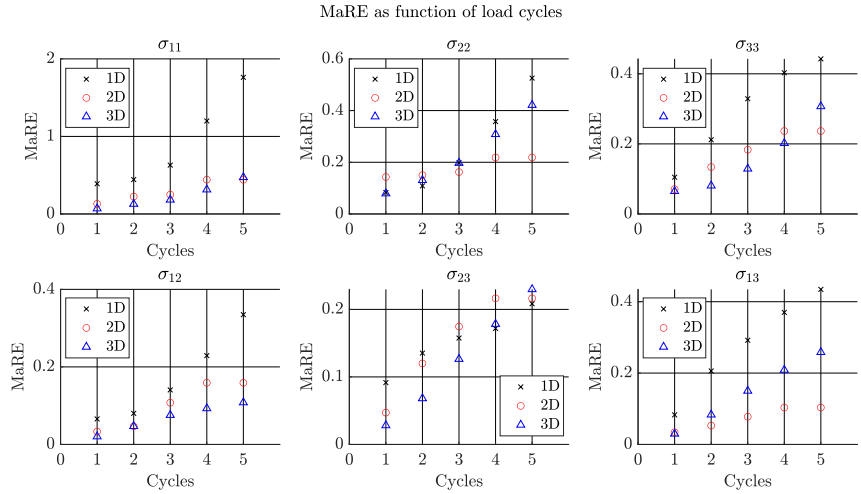


Figure 4.6: The MaRE of a cyclical test as a function of the number of cycles is presented. The component wise errors for a uniaxial fiber distribution, a uniform 2D distribution, and a uniform 3D distributions are given.

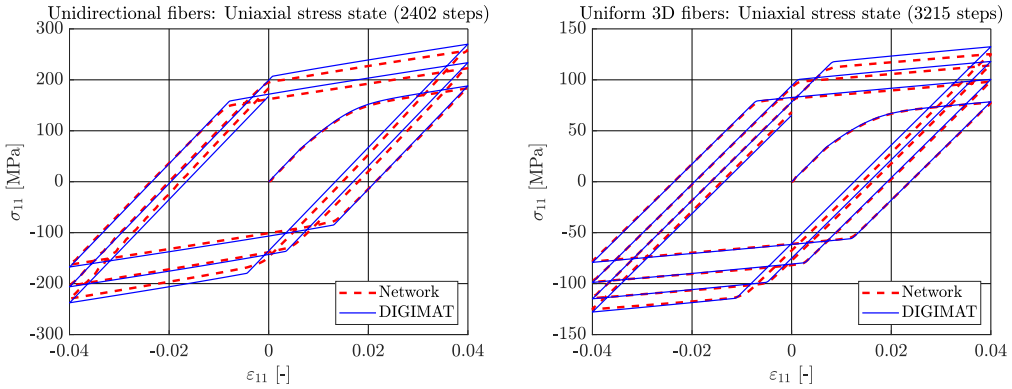


Figure 4.7: Two stress-strain curves consisting of the network prediction compared to the output of DIGIMAT-MF. Subfigure (a) shows the the results from a uniaxial test performed on a sample consisting of unidirectional fibers, while subfigure (b) shows the results of a uniaxial test performed on a uniform 3D fiber distribution.

4.2.3 Testing of Extrapolation Ability

The computed error in the prediction of the σ_{11} component is found in Figure 4.8. It seems like for fiber volume fractions in the permissible range (10%-15%), strains between 5% and 7.5% do not lead to a very significant increase in the error. However, strains between 7.5% and 10% leads to more significant increase in the error for admissible fiber volume fractions. It can also be seen that the model generalizes very well to fiber volume fractions lower than 10% and between 15% and 20% for strains in the admissible range ($\leq 5\%$). The impressive extrapolation ability of the model for the admissible strains can be seen in the plot of two representative stress-strain curves in Figure 4.9. On the other hand, the detrimental effects on the prediction error for maximum strains larger than 5% are amplified for fiber volume fractions outside the admissible range.

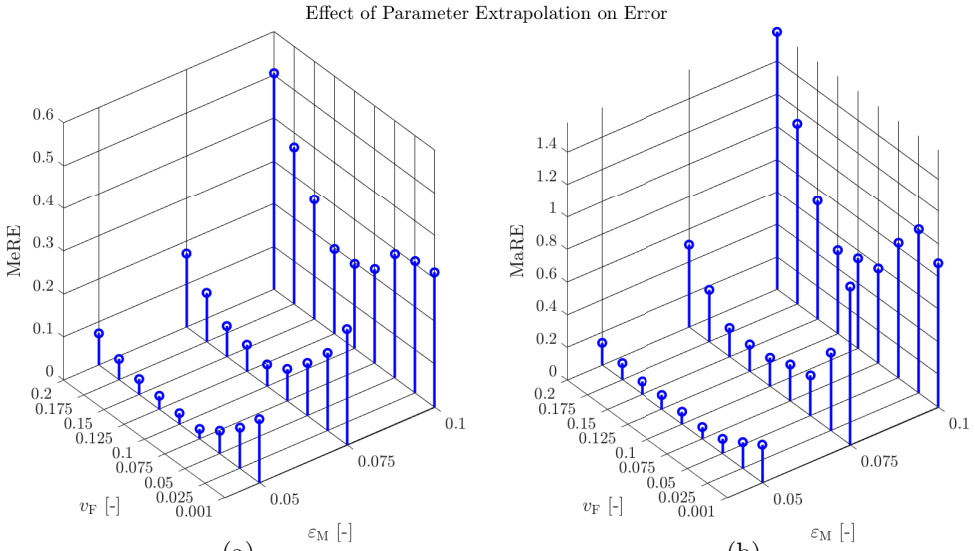


Figure 4.8: The error in the σ_{11} component is plotted as a function of strains and fiber volume fractions beyond the admissible range. The graph in subfigure (a) shows the MeRE, while the graph in subfigure (b) shows the MaRE.

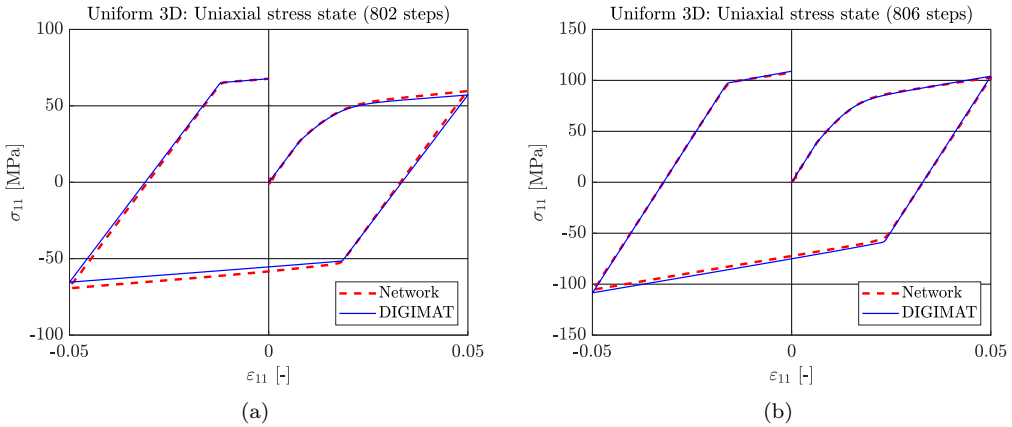


Figure 4.9: Two stress-strain curves consisting of the network prediction compared to the output of DIGIMAT-MF. Subfigure (a) shows the the results from a uniaxial test performed on a sample with fiber volume fraction 2.5%, while subfigure (b) shows the results of a uniaxial test performed on a sample with fiber volume fraction 20%.

4.2.4 Hydrostatic Loading

AThe difference in the σ_{11} component is presented in Figure 4.10. The predicted stress seems to agree with the expected result up until about 560 MPa, where the model erroneously predicts what looks like perfect plasticity. This behavior starts at almost exactly 4% strain. It can therefore probably be explained by the fact that the network was trained with a maximum allowed strain component of 5%. Since hydrostatic loading was not explicitly included in the training data it is reasonable that the breakpoint occurs earlier than 5% since only a quarter of the training data had a maximum strain in the range 4% to 5%.

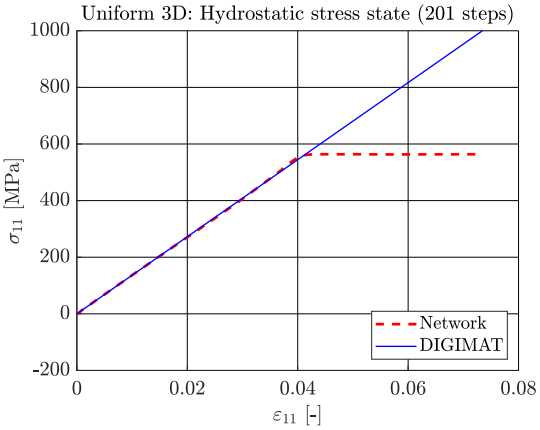


Figure 4.10: The stress-strain curve of a hydrostatic loading test conducted in DIGIMAT-MF is compared with the network prediction. The knee in the dashed curve occurs at a stress of approximately 560 MPa.

5 Discussion

The acquired neural network, albeit exhibiting good performance, can not without doubt be said to be optimal. In order to achieve an optimal network architecture the hyperparameters would need to be investigated more systematically. The complexity of the problem however results in very long network training times. A detailed survey of the hyperparameters would require more dedicated hardware to enable parallel training of networks with multiple sets of parameters. Nevertheless, the number of epochs that the network was trained for can be consider sufficient. This is motivated by the fact that the loss decreased according to Equation (4.1). The trend of the cost had a minimum at epoch $E = 614$, and the cost at the minimum compared to the cost at epoch $E = 500$ is insignificant. It is also possible to argue for the number of layers and neurons being appropriate. Previous attempts at predicting elasto-plasticity with GRU's showed that networks with more than 3 layers and 500 neurons per layer didn't display any significant improvement in performance [30].

One possibility of reducing the prediction error of the network is modifying the cost function used during training. The cost as stated in Equation (3.12) puts the same emphasis on every time step. Since stresses largely vary in magnitude, it becomes apparent that the cost function used in this project puts more emphasis on large stresses. An alternative would be to use a cost function with some sort of normalization. One possible cost function to investigate in a future work would be

$$C = \frac{1}{N} C_n, \quad \text{where} \quad C_n = \frac{1}{2N_T} \sum_{t=1}^{N_T} \sum_{i=1}^{F} \left(\frac{\hat{x}^i_{\text{out}[t]}(n) - x^i_{\text{out}[t]}(n)}{\hat{x}^i_{\text{out}[t]}(n)} \right)^2, \tag{5.1}$$

where the same notation as the one used in (3.12) is employed.

The chosen method of generating training data may be deemed successful. The created network shows good performance, and it seems that it is able to accurately predict the response to a wide variety of mechanical loads. There is however room for improvement. One obvious thing to investigate is the inclusion of characteristic training data that represent physically relevant phenomena such as hydrostatic loading. It was shown that the model struggled in predicting hydrostatic loading beyond a certain stress. It is possible that introducing hydrostatic loading in the training data could remedy this. Similarly, the model displayed a surprisingly large MeRE when predicting uniaxial stress in sample 3 of the general testing samples (see Tables 3.3 and 4.2). While uniaxial strain states were included in the training data, uniaxial stress states were not. Any subsequent iteration of the network should therefore include specific stress states in addition to strain states to properly capture the mechanics. Alternatively, in future investigations it would be of

interest to consider more smooth strain paths for training. This has been successfully implemented by others, using Gaussian processes [42].

The tests on cyclical loading shows that the network model has a propensity to accurately predict complex load paths. Seeing that the error approximately increases linearly with the increase in complexity, it is possible for the user to estimate the error of complex loading. Subsequently the user can make a decision if the results are accurate enough. This would however need to be investigated in more detail before any definite conclusion can be drawn. In light of the poor extrapolation ability of DNN models, it was also a positively surprising result that the model could make accurate predictions for fiber volume fractions outside the range it was trained on (10%-15%). As is always the case, the user of a model needs to be aware of the model's limitations. It is however reassuring that the model manages to operate with success outside it's intended area of use. A similar statement can to some extent be made about strains beyond 5%. This is however less useful since failure/damage most definitely would start to occur beyond this point, rendering the model useless anyway.

A natural question to ask is what the proposed neural network model offers that the traditional constitutive model that it was trained on does not. To begin with, the proposed model offers the ability to change fiber orientation, and fiber volume fraction without having to perform additional simulations to homogenize the composite for the new set of parameters. This is important in the context of injection molding where the fiber density and orientation varies highly throughout the molded part. Secondly, the neural network model offers computational speed. As an example, using DIGIMAT-MF to compute the response to complex general 3D-loading (like the ones in the training data) may take up to a minute. In contrast, the network model makes the prediction in less than a second. Finally, the network contains all the information on loading/unloading and accumulated plastic strain/hardening purely through the strain history. Therefore, the process of determining these quantities repeatedly through Equation (3.9) is not necessary. As a consequence, a lot of time is saved if the model is used in a framework where iterative calculations are required. This is important when implementing the network as a constitutive model in an FEM setting.

5.1 Prospects of FEM Implementation

Consider the domain Ω with boundary $\Gamma = \Gamma_h \bigcup \Gamma_g$. Let $\boldsymbol{t}$ be the traction acting on Γ and $\boldsymbol{u}$ be the displacement of a material point in Ω. In addition, a body force $\boldsymbol{b}$ is acting inside Ω. The displacement is imposed as $\boldsymbol{u} = \boldsymbol{g}$ on Γ_g, while the traction is imposed as $\boldsymbol{t} = \boldsymbol{h}$ on $\boldsymbol{\Gamma}_h$. The FE formulation of the equilibrium boundary value problem over the domain is given by [43]

$$\int_\Omega \boldsymbol{B}^{\mathrm{T}} \boldsymbol{\sigma}(\boldsymbol{B}\boldsymbol{a}) d\Omega = \int_\Omega \boldsymbol{N}^{\mathrm{T}} \boldsymbol{b} d\Omega + \int_{\Gamma_h} \boldsymbol{N}^{\mathrm{T}} \boldsymbol{h} d\Gamma + \int_{\Gamma_g} \boldsymbol{N}^{\mathrm{T}} \boldsymbol{t} d\Gamma, \tag{5.2}$$

where $\boldsymbol{N}$ is a matrix of the basis shape functions ($\boldsymbol{u} \approx \boldsymbol{N}\boldsymbol{a}$), and $\boldsymbol{B}$ is a matrix containing the spatial derivatives of $\boldsymbol{N}$. The objective is to solve for the vector $\boldsymbol{a}$ such that the equation holds. Since the relation between the stress and strain, and in extension the stress and displacement, is non-linear for plasticity, the FE equation needs to be solved through some iterative scheme such as Newton-Raphson. By calling the left hand side the internal force $\boldsymbol{f}_{\mathrm{int}}$, and the right hand side the

external force $\boldsymbol{f}_{\text{ext}}$, the problem can be reformulated as:

$$\text{Find } \boldsymbol{a} \text{ such that:} \quad \boldsymbol{f}_{\text{int}}(a) - \boldsymbol{f}_{\text{ext}} = 0. \tag{5.3}$$

In order to solve this equation through Newton-Raphson, the derivative of $\boldsymbol{f}_{\text{int}}$ with respect to $\boldsymbol{a}$ is required. This quantity is called the tangent stiffness matrix. Using the fact that $\boldsymbol{\varepsilon} \approx \boldsymbol{B}\boldsymbol{a}$, the derivative is computed as

$$\frac{d}{d\boldsymbol{a}}\boldsymbol{f}_{\text{int}} = \int_{\Omega} \frac{d}{d\boldsymbol{a}}\boldsymbol{\sigma}(\boldsymbol{B}\boldsymbol{a})d\Omega = \int_{\Omega} \frac{\partial \boldsymbol{\sigma}}{\partial(\boldsymbol{B}\boldsymbol{a})}\frac{d\boldsymbol{B}\boldsymbol{a}}{d\boldsymbol{a}}d\Omega = \int_{\Omega} \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\varepsilon}}d\Omega\boldsymbol{a}. \tag{5.4}$$

The neural network constitutive model takes the orientation tensor, volume fraction, and current strain state as inputs and uses the strain history to predict the current stress. The neural network model is differentiable with respect to the input signal, so it is differentiable with respect to the strain. The strain history is a known quantity, it is therefore always possible to carry out the computation in Equation 5.4 and receive the tangent stiffness. The derivative is easily evaluated with the help of automatic differentiation as is standard for machine learning applications. Automatic differentiation has already successfully been used for DNN based constitutive models to find tangent stiffness matrices [9].

Solving Equation 5.3 iteratively requires that the constitutive model can handle a wide variety of step sizes in $\boldsymbol{a}$, and by extension $\boldsymbol{\varepsilon}$, to ensure convergence. It has been shown that the procured ANN model is rate independent for a quite large range of loading rates. It can therefore be considered robust enough to handle FEM implementation. Additionally, as the model is time-independent there is no inherent limitations on adjusting the loading to match the network. The limiting factor that decides the maximum time step in a non-linear FEM solver is getting the solution of Equation 5.3 to converge. It is therefore always possible to interpolate/extrapolate the load-stepping path such that the step size matches the admissible range of the network.

It has here been made plausible that the developed model is suitable for FEM implementation. There is however no guarantee that derived quantities from a FEM simulation, such as energy or work rate, will adhere to fundamental physical constraints. For instance, there are situations where the model erroneously predicts and increase in stress when the strain is decreasing, which in turn leads to a negative work rate. In order to rectify this, an alternative neural network approach may be taken.

5.2 Physics Aware Neural Networks

In the present work, a DNN model has been trained on micro-mechanical simulations with the hope of the model being able to infer the physics elasto-plasticity purely by observing stress-strain relations. As an alternative, a network may be trained with physically motivated constraints in order to ensure laws such as energy conservation. *Linka* et al. [44] introduced *Constitutive Artificial Neural Networks* (CANN). These networks first compute the variants of the strain tensor and combine them with micro-mechanical descriptors to compute a set of generalized invariants. The generalized invariants are in turn used to compute a strain energy functional. The stress and

constitutive tensor are then easily obtained from the strain energy through differentiation. By computing the stress from an energy functional it is ensured that certain physical laws remain unviolated. The network is trained as usual with a strain input, with the goal of matching a predetermined stress-strain relation. One perk with this approach in addition to certain laws of physics already being built into the network structure is the reduced need for training data. The results in the referred study look incredibly promising and it is very likely that it will inspire many similar research projects. As the model proposed in the present work yield some clearly physically dubious results, such as accumulating plastic strain during hydrostatic loading, it would be of interest investigating the possibility of introducing mechanically motivated constraints into a future implementation of the network.

6 Conclusions

In this project a deep neural network model that predicts the elasto-plastic response of a short fiber composite material was developed. The material consists of elastic fibers with a J_2 elasto-plastic matrix obeying a linear exponential hardening law. The network was trained on data from micro-mechanical simulations utilizing the Mori-Tanaka mean field method for homogenization. The strain-paths used as input to the simulations were generated by utilizing a random walks in 6D-strain space with bias directions. The created model allows for an arbitrary fiber distribution by defining the second order fiber orientation tensor. Additionally, it is possible to vary the fiber volume fraction between 10% and 15%, but it is possible to go slightly outside these bounds without introducing significant error.

The proposed model shows promise and seems to give accurate predictions for a wide range of fiber orientation distributions and loading types. The average Mean Relative Error (MeRE) in the stress prediction for typical applications is below 10% of the matrix yield stress most of the time. In addition the MeRE never exceeded 25% of the matrix yield stress during one cycle tests. The highest recorded Maximum Relative Error (MaRE) in the general testing was below 50% of the matrix yield stress. The model displays accurate yielding behavior with the appropriate hardening law (including saturation), and shows a clear difference in loading/unloading. It was furthermore demonstrated that the model correctly possesses rate independence for a large range of loading rates. Judging from these properties of the created artificial neural network model, it may be stated that the purpose of this project has been fulfilled.

It was determined that GRU based networks outperformed LSTM networks for the current application. Another very important takeaway from the design process was the importance of adding dropout to the network. It was one of the network components that improved convergence significantly. An additional valuable lesson was the fact that a bigger batch size didn't necessarily improve convergence, which may seem counter-intuitive and thus be overlooked by inexperienced network designers.

As a future outlook, a finite element implementation of the model is of interest. A true test of the model validity is the ability to make predictions of the behavior of a structural component under operating load. It is also of interest to implement the current model for different types of hardening laws to further test the applicability of DNN's on constitutive modeling. Finally, the possibility of improving the modeling by utilizing physical constraints in the model seems very promising.

References

[1] M. Holmes, "Recycled carbon fiber composites become a reality," *Reinforced Plastics*, vol. 62, no. 3, pp. 148 – 153, 2018.

[2] B. Agarwal, L. Broutman, and K. Chandrashekhara, *Analysis and Performance of Fiber Composites*. Wiley, 2017.

[3] N. Nawafleh and E. Celik, "Additive manufacturing of short fiber reinforced thermoset composites with unprecedented mechanical performance.," *Additive Manufacturing*, vol. 33, 2020.

[4] S. Mirkhalaf, T. van Beurden, M. Ekh, F. Larsson, and M. Fagerström, "A finite element based orientation averaging model for predicting elasto-plastic behaviour of short fiber reinforced composites," 2021. (under review).

[5] S. Mirkhalaf, E. Eggels, A. Anantharanga, F. Larsson, and M. Fagerström, "Short fiber composites: Computational homogenization vs orientation averaging," in *Proceedings of the 2019 International Conference on Composite Materials* (A. Mouritz, C. Wang, and B. Fox, eds.), RMIT University, Melbourne, Australia, Aug 2019.

[6] S. Mirkhalaf, E. Eggels, T. van Beurden, F. Larsson, and M. Fagerström, "A finite element based orientation averaging method for predicting elastic properties of short fiber reinforced composites," *Composites Part B: Engineering*, vol. 202, p. 108388, 2020.

[7] H. Liu and Z. Long, "An improved deep learning model for predicting stock market price time series," *Digital Signal Processing*, vol. 102, p. 102741, July 2020.

[8] M. Alam, M. Samad, L. Vidyaratne, A. Glandon, and K. Iftekharuddin, "Survey on deep neural networks in speech and vision systems," *Neurocomputing*, vol. 417, pp. 302–321, Dec. 2020.

[9] D. Huang, J. N. Fuhg, C. Weißenfels, and P. Wriggers, "A machine learning based plasticity model using proper orthogonal decomposition.," *Computer Methods in Applied Mechanics and Engineering*, vol. 365, 2020.

[10] A. Zhang and D. Mohr, "Using neural networks to represent von mises plasticity with isotropic hardening.," *International Journal of Plasticity*, vol. 132, 2020.

[11] J. C. Simo, *Computational inelasticity*. New York: Springer, 1998.

[12] L. Råde and B. Westergren, *Mathematics handbook for science and engineering*. Lund: Studentlitteratur, 2004.

[13] J. D. Eshelby and R. E. Peierls, "The determination of the elastic field of an ellipsoidal inclusion, and related problems," *Proc. R. Soc. Lond. A*, vol. 241, no. 1226, p. 376–396, 1957.

[14] T. Mura, *Micromechanics of defects in solids*. Dordrecht, Netherlands Boston Hingham, MA, USA: M. Nijhoff Distributors for the U.S. and Canada, Kluwer Academic Publishers, 1987.

[15] R. Hill, "Elastic properties of reinforced solids: Some theoretical principles," *Journal of the Mechanics and Physics of Solids*, vol. 11, pp. 357–372, Sept. 1963.

[16] T. Mori and K. Tanaka, "Average stress in matrix and average elastic energy of materials with misfitting inclusions," *Acta Metallurgica*, vol. 21, pp. 571–574, May 1973.

[17] Y. Benveniste, "A new approach to the application of mori-tanaka's theory in composite materials," *Mechanics of Materials*, vol. 6, pp. 147–157, June 1987.

[18] E. T. Onat and F. A. Leckie, "Representation of mechanical behavior in the presence of changing internal structure," *Journal of Applied Mechanics*, vol. 55, pp. 1–10, Mar. 1988.

[19] S. G. Advani and C. L. Tucker, "The use of tensors to describe and predict fiber orientation in short fiber composites," *Journal of Rheology*, vol. 31, pp. 751–784, Nov. 1987.

[20] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 303–314, Dec. 1989.

[21] P. Kidger and T. Lyons, "Universal approximation with deep narrow networks," in *Proceedings of Machine Learning Research*, vol. 125, pp. 2306–2327, PMLR, 09–12 Jul 2020.

[22] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[23] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade*, vol. 7700 of *Lecture Notes in Computer Science*, pp. 9–48, Springer Berlin Heidelberg, 2012.

[24] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015.

[25] H. Zen and H. Sak, "Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, Apr. 2015.

[26] R. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Back-propagation: Theory, Architectures and Applications* (Y. Chauvin and D. Rumelhart, eds.), pp. 433—-486, Lawrence Erlbaum Publishers, Hillsdale, NJ, 1995.

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997.

[28] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation*, vol. 12, pp. 2451–2471, Oct. 2000.

[29] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.

[30] M. Mozaffar, R. Bostanabad, W. Chen, K. Ehmann, J. Cao, and M. A. Bessa, "Deep learning predicts path-dependent plasticity," *Proceedings of the National Academy of Sciences*, vol. 116, pp. 26414–26420, Dec. 2019.

[31] L. Devroye, *Non-uniform random variate generation*. New York: Springer-Verlag, 1986.

[32] J. Arvo, "FAST RANDOM ROTATION MATRICES," in *Graphics Gems III (IBM Version)*, pp. 117–120, Elsevier, 1992.

[33] S. Kammoun, I. Doghri, L. Adam, G. Robert, and L. Delannay, "First pseudo-grain failure model for inelastic composites with misaligned short fibers," *Composites Part A: Applied Science and Manufacturing*, vol. 42, pp. 1892–1902, Dec. 2011.

[34] S. Wu, B. Wang, G. Zheng, S. Liu, K. Dai, C. Liu, and C. Shen, "Preparation and characterization of macroscopically electrospun polyamide 66 nanofiber bundles," *Materials Letters*, vol. 124, pp. 77–80, June 2014.

[35] eX stream, DIGIMAT USER'S MANUAL. MSC Software, Belgium SA., 2020.

[36] J. S. Cintra and C. L. Tucker, "Orthotropic closure approximations for flow-induced fiber orientation," *Journal of Rheology*, vol. 39, pp. 1095–1122, Nov. 1995.

[37] "The julia programming language." https://julialang.org/. Accessed: 2020-01-27.

[38] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," 2017.

[39] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[40] P. Baldi and P. J. Sadowski, "Understanding dropout," in *Advances in Neural Information Processing Systems* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), vol. 26, pp. 2814–2822, Curran Associates, Inc., 2013.

[41] The Mathworks, Inc., Natick, Massachusetts, MATLAB *version 9.9.0.1538559 (R2020b) Update 3*, 2020.

[42] H. J. Logarzo, G. Capuano, and J. J. Rimoli, "Smart constitutive laws: Inelastic homogenization through machine learning," *Computer Methods in Applied Mechanics and Engineering*, vol. 373, p. 113482, Jan. 2021.

[43] N. Saabye Ottosen and H. Petersson, *Introduction to the finite element method*. New York: Prentice Hall, 1992.

[44] K. Linka, M. Hillgärtner, K. P. Abdolazizi, R. C. Aydin, M. Itskov, and C. J. Cyron, "Constitutive artificial neural networks: A fast and general approach to predictive data-driven constitutive modeling by deep learning," *Journal of Computational Physics*, p. 110010, Nov. 2020.

[45] P. Grinfeld, *Introduction to tensor analysis and the calculus of moving surfaces*. New York: Springer, 2013.

A Tensors and Index Notation

Let $\mathcal{V}$ be an n-dimensional vector space equipped with the scalar product $(\cdot)$. The scalar product is a symmetric bilinear form. To be a symmetric bilinear form $(\cdot)$ has the requirement that for any three vectors $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathcal{V}$ and scalar λ the following must hold true:

$$\begin{aligned} &(\boldsymbol{u}+\boldsymbol{v})\cdot\boldsymbol{w} = \boldsymbol{u}\cdot\boldsymbol{w} + \boldsymbol{v}\cdot\boldsymbol{w} \\ &(\lambda\boldsymbol{u})\cdot\boldsymbol{v} = \lambda\boldsymbol{u}\cdot\boldsymbol{v} \\ &\boldsymbol{u}\cdot\boldsymbol{v} = \boldsymbol{v}\cdot\boldsymbol{u}. \end{aligned} \tag{A.1}$$

The magnitude of a vector $\boldsymbol{u} \in \mathcal{V}$ is defined by

$$|\boldsymbol{u}|^2 = \boldsymbol{u}\cdot\boldsymbol{u}, \tag{A.2}$$

Consequently, the angle θ between any two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{V}$ is defined as

$$\cos(\theta) = \frac{\boldsymbol{u}\cdot\boldsymbol{v}}{|\boldsymbol{u}||\boldsymbol{v}|}. \tag{A.3}$$

Any element of $\mathcal{V}$ can be expressed as a linear combination of n linearly independent basis vectors $\boldsymbol{e}_i$, where $i = 1, 2, ..., n$. The element $\boldsymbol{v}$ can be accordingly be written as

$$\boldsymbol{v} = \sum_{i=1}^{n} v^i \boldsymbol{e}_i, \tag{A.4}$$

where v^i is a scalar and is called the ith coordinate of $\boldsymbol{v}$ in the basis $\boldsymbol{e}_i$. Using the properties from Equation (A.1), the scalar product between two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ in $\mathcal{V}$ expressed in the basis $\boldsymbol{e}_i$ is computed as

$$\boldsymbol{u}\cdot\boldsymbol{v} = \left(\sum_{i=1}^{n} u^i \boldsymbol{e}_i\right)\cdot\left(\sum_{j=1}^{n} v^j \boldsymbol{e}_j\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} u^i v^j \boldsymbol{e}_i\cdot\boldsymbol{e}_j = \sum_{i=1}^{n}\sum_{j=1}^{n} u^i v^j g_{ij}. \tag{A.5}$$

Here $\boldsymbol{e}_i\cdot\boldsymbol{e}_j$ is defined to be g_{ij} and is called the metric tensor. The scalar product is positive definite and symmetric. As a consequence the metric tensor also possess those properties, and by extension has an inverse. For a basis that is orthonormal the metric tensor is the identity.

A.1 Index Notation

The summands in Equation (A.4) contains two of the indices i, while the ones in Equation (A.5) contains two of indices i and two of j. What these equations have in common is that every summation symbol has two counts of its index associated with it, one superindex and one subindex. Therefore it superfluous to use the summation symbol. Instead index-notation, also often called Einstein-notation, is used. Whenever an index appears twice in a term, once as superindex and once as a subindex, a sum is implied according to

$$\sum_{i=1}^{n} a^i b_i = a^i b_i. \tag{A.6}$$

Moreover, this notation requires that every index only can appear twice in every term to avoid ambiguities. The components of the vector $\boldsymbol{v} = v^i \boldsymbol{e}_i$ can be written on its own as v_i. The index i is then called a free index and takes the value $i = 1, 2, ..., n$. When an object with a free index is multiplied with an object containing the same letter index in the opposite position, summation is carried out as described above. This sum arising from multiplying a subindex factor with a superindex factor is called taking the contraction of the index.

A.2 Change of Basis

The vector $\boldsymbol{v} \in \mathcal{V}$ can be expressed in a second basis $\tilde{\boldsymbol{e}}_i$ with coordinates $\tilde{v}^i$. Since the vector is independent on which basis it is expressed in the following relationship between the bases is established

$$v^i \boldsymbol{e}_i = \tilde{v}^i \tilde{\boldsymbol{e}}_i. \tag{A.7}$$

Similarly, the ith basis vector in the old basis can be expressed as a linear combination of the new basis vectors as

$$\boldsymbol{e}_i = J_i^j \tilde{\boldsymbol{e}}_j, \tag{A.8}$$

where J_i^j is the jth component in the new basis of the ith basis vector in the old basis. Multiplying both sides of Equation (A.8) with $\tilde{\boldsymbol{e}}_k$ yields

$$\boldsymbol{e}_i \cdot \tilde{\boldsymbol{e}}_k = J_i^j \tilde{g}_{jk}, \tag{A.9}$$

where $\tilde{g}_{jk}$ is the metric tensor for the basis $\tilde{\boldsymbol{e}}_i$. Multiplying both sides of Equation (A.7) with $\tilde{\boldsymbol{e}}_k$ and inserting the result from Equation (A.9) the following is obtained

$$v^i J_i^j \tilde{g}_{jk} = \tilde{v}^i \tilde{g}_{ik}. \tag{A.10}$$

Since all the indices are summation indices, the specific symbol that is used is irrelevant and can be modified freely as long as it is modified in all places simultaneously ($a^i b_i = a^j b_j$). Equation (A.10) is thus rewritten as

$$v^i J_i^j \tilde{g}_{jk} = \tilde{v}^j \tilde{g}_{jk}. \tag{A.11}$$

As stated previously the metric tensor has an inverse, so

$$\tilde{v}^j = v^i J_i^j. \tag{A.12}$$

In Equation (A.8) the change from the new basis to the old is carried out by the tensor J_i^j, on the other hand the coordinates are transformed in the reverse direction by the same tensor J_i^j in Equation (A.12). This fact shows that the coordinates and the bases are of a different nature. The basis vectors are called covariant tensors and denoted by a subindex. As the coordinates transform opposite to the basis vectors they are called contravariant and are denoted with a superindex.

A.3 The Bases $\boldsymbol{e}_i$, $\tilde{\boldsymbol{e}}_j$ and their Jacobian

Let $\boldsymbol{R} \in \mathcal{V}$ be the position vector. For some system of coordinates x^i that describes $\boldsymbol{R}$ the basis vectors are defined as

$$\boldsymbol{e}_i = \frac{\partial \boldsymbol{R}}{\partial x^i}, \tag{A.13}$$

i.e. the direction $\boldsymbol{R}$ shifts when one of the coordinates are perturbed. At the same time $\boldsymbol{R}$ is also able to be expressed in the alternative system of coordinates $\tilde{x}^j$. The derivative in Equation (A.13) is rewritten via the chain rule as

$$\boldsymbol{e}_i = \frac{\partial \boldsymbol{R}(\tilde{x}(x))}{\partial x^j} = \frac{\partial \boldsymbol{R}}{\partial \tilde{x}^j}\frac{\partial \tilde{x}^j}{\partial x^i} = \tilde{\boldsymbol{e}}_j J_i^j. \tag{A.14}$$

The derivative of $\boldsymbol{R}$ with respect to $\tilde{x}^j$ in Equation (A.14) is the basis vector $\tilde{\boldsymbol{e}}_j$ by definition. Thereby it follows that the coordinates of one system derived with the respect to the coordinate of another system is the object that transforms the basis vector. This is the object J_i^j introduced earlier and it is recognizable as the Jacobian of the basis change $\tilde{x}$ to x. Similarly, by beginning with the vector $\tilde{\boldsymbol{e}}_j$ the following must hold

$$\tilde{\boldsymbol{e}}_j = \frac{\partial \boldsymbol{R}(x(\tilde{x}))}{\partial \tilde{x}^j} = \frac{\partial \boldsymbol{R}}{\partial x^i}\frac{\partial x^i}{\partial \tilde{x}^j} = \boldsymbol{e}_i \tilde{J}_j^i, \tag{A.15}$$

where $\tilde{J}_j^i$ is the Jacobian of the basis change x to $\tilde{x}$. By comparing Equation (A.14) and (A.15) it becomes evident that J_i^j and $\tilde{J}_j^i$ must be each others inverses. This can be expressed as

$$J_k^i \tilde{J}_j^k = \tilde{J}_k^i J_j^k = \delta_j^i, \tag{A.16}$$

where δ_j^i is the object that simply renames an index when contracted as follows

$$\delta_j^i T_i = T_j \text{ or } \delta_j^i T^j = T^i. \tag{A.17}$$

δ_j^i is called the *Kronecker* delta and is analogous to the identity matrix in linear algebra.

A.4 The Dual Basis e^i

As remarked earlier the metric tensor g_{ij} pertaining the basis $\boldsymbol{e}_i$ has an inverse. Following the formalism in Equation (A.16) the inverse should be of the form

$$g^{ik}g_{kj} = \delta^i_j. \tag{A.18}$$

The tensor g^{ik} in Equation (A.18) is called the inverse, or contravariant metric tensor. Similarly, g_{jk} is also called the covariant metric tensor. As the covariant metric tensor, the contravariant metric tensor is positive definite and symmetric ($g^{ij} = g^{ji}$). Using the contravariant metric tensor the following definition can be made:

$$\boldsymbol{e}^i = g^{ij}\boldsymbol{e}_j. \tag{A.19}$$

The object $\boldsymbol{e}^i$ is called the dual basis of $\boldsymbol{e}_i$, or the contravariant basis. Analogously, the original basis $\boldsymbol{e}_i$ is referred to as the covariant basis. A benefit of introducing the dual basis is that the scalar product of a basis vector and its dual always results in the Kronecker delta, even for non-orthogonal coordinate systems. This easily follows from inserting the definition from Equation (A.19),

$$\boldsymbol{e}_j \cdot \boldsymbol{e}^i = \boldsymbol{e}_j \cdot \boldsymbol{e}_k g^{ik} = g_{jk}g^{ik} = \delta^i_j. \tag{A.20}$$

Furthermore, by similar manipulations of Equation (A.19), the contravariant metric tensor can be computed through the scalar product $\boldsymbol{e}^i \cdot \boldsymbol{e}^j = g^{ij}$ through

$$\boldsymbol{e}^i \cdot \boldsymbol{e}^j = g^{ik}\boldsymbol{e}_k \cdot \boldsymbol{e}_l g^{lj} = g^{ik}g_{kl}g^{lj} = \delta^i_l g^{lj} = g^{ij}. \tag{A.21}$$

The result in Equation (A.21) is analogous to the definition of the covariant metric tensor. One useful consequence of Equation (A.20) is that the coordinates v^i of a vector $\boldsymbol{v}$ can be simply extracted through scalar multiplication with the dual basis elements as

$$\boldsymbol{v} \cdot \boldsymbol{e}^i = v^j \boldsymbol{e}_j \cdot \boldsymbol{e}^i = v^j \delta^i_j = v^i. \tag{A.22}$$

The exact same reasoning can be used when extracting the covariant coordinates v_j from the tensor $\boldsymbol{v} = v_j\boldsymbol{e}^j$ through scalar multiplication with the covariant basis $\boldsymbol{e}_i$.

A.5 Formal Definition of Tensors

A quantity $\tilde{T}^{i_1\ldots i_n}_{j_1\ldots j_m}$ is said to be a tensor of covariant rank m and and contravariant rank n if it transforms to a different coordinate system through the rule

$$T^{k_1\ldots k_n}_{l_1\ldots l_m} = \tilde{T}^{i_1\ldots i_n}_{j_1\ldots j_m} J^{j_1}_{l_1}\ldots J^{j_m}_{l_m}\tilde{J}^{k_1}_{i_1}\ldots\tilde{J}^{k_n}_{i_n}. \tag{A.23}$$

In other words, an object is a tensor of covariant rank m and contravariant rank n (more briefly rank(m, n)) if it in a change of coordinates needs to be multiplied by m forward transforming Jacobians and n inverse Jacobians. Since it is completely unambiguous what type of Jacobian that is required to transform the object based on the position of the indices, the tilde can be dropped

without loss of clarity. For example, the metric tensor of rank$(2,0)$ would transform with two forward transforms (contracting the upper index of the Jacobian) like

$$g_{ij} = \tilde{g}_{kl} J_i^k J_j^l, \tag{A.24}$$

while the inverse metric tensor of rank$(0,2)$ would transform with two inverse Jacobians (contracting the lower index of the Jacobian) like

$$g^{ij} = \tilde{g}^{kl} J_k^i J_l^j. \tag{A.25}$$

A.6 Raising and Lowering Indices

If the components of some covariant tensor T_j are known, the components of the contravariant tensor T^i is defined by

$$T^i = g^{ij} T_j. \tag{A.26}$$

The opposite holds for the known components of a contravariant tensor T'^j,

$$T'_i = g_{ij} T'^j. \tag{A.27}$$

This act is known as raising and lowering indices, and makes the notation short and concise since it erases the need to explicitly write down the metric tensors.

A.7 Tensors and Invariance

One of the most important concepts in physics is invariance, the property of independence of the choice of coordinates. An example of invariant objects are vectors. The actual position of an object in space is independent on the way its position is described. The position vector is given by the contraction of the chosen coordinates and their basis vectors as

$$\boldsymbol{R} = R^i \boldsymbol{e}_i = R_i \boldsymbol{e}^i. \tag{A.28}$$

However, since this must hold for every possible choice of valid coordinates, the act of contracting two tensors seem to generate an invariant object. To show this is a fact for the contraction of two tensors of arbitrary rank consider the contraction of the rank(m,n) tensor T and the rank(n,m) tensor S:

$$T^{i_1 \dots i_n}_{j_1 \dots j_m} S^{j_1 \dots j_m}_{i_1 \dots i_n} = \tilde{T}^{k_1 \dots k_n}_{l_1 \dots l_m} J^{i_1}_{k_1} \dots J^{i_n}_{k_n} J^{l_1}_{j_1} \dots J^{l_m}_{j_m} S^{j_1 \dots j_m}_{i_1 \dots i_n} = \tilde{T}^{i_1 \dots i_n}_{j_1 \dots j_m} \tilde{S}^{j_1 \dots j_m}_{i_1 \dots i_n}. \tag{A.29}$$

The act of contracting two tensors thus gives the same result in every system of coordinates. This fact carries some tremendous consequences. If the contraction of two tensors give the correct result in one system of coordinates, the result is correct in all systems. By extension, it is sufficient to prove tensor relations involving contractions in the most convenient set of coordinates. Furthermore, if a tensor $T^{k_1 \dots k_n}_{l_1 \dots l_m}$ is shown to be 0 in on set of coordinates it must be 0 in all sets of coordinates. This is true since

$$\tilde{T}^{i_1 \dots i_n}_{j_1 \dots j_m} = T^{k_1 \dots k_n}_{l_1 \dots l_m} J^{i_1}_{k_1} \dots J^{i_n}_{k_n} J^{l_1}_{j_1} \dots J^{l_m}_{j_m} = 0^{k_1 \dots k_n}_{l_1 \dots l_m} J^{i_1}_{k_1} \dots J^{i_n}_{k_n} J^{l_1}_{j_1} \dots J^{l_m}_{j_m} = 0^{i_1 \dots i_n}_{j_1 \dots j_m}. \tag{A.30}$$

A.8 The Christoffel Symbol Γ^i_{jk}

Consider the set of basis vectors $\boldsymbol{e}_i$. In general, these vectors are dependent on the position in space, like in the case of cylindrical, spherical, or toroidal coordinates. To get a measure of how each of the $i = 1, ..., n$ basis vectors varies with every different coordinate, the derivative with respect to a new independent index $j = 1, ..., n$ is computed. Every resulting vector from this derivative can in turn be expressed as combination of $k = 1, ..., n$ components in the treated basis. The following holds

$$\frac{\partial \boldsymbol{e}_i}{\partial x^j} = \Gamma^k_{ij} \boldsymbol{e}_k, \tag{A.31}$$

where Γ^k_{ij} is called the *Christoffel* symbol of the coordinate system x^j. The Christoffel symbol given explicitly by taking the scalar product with $\boldsymbol{e}^l$ on both sides of Equation (A.31) and renaming the indices,

$$\Gamma^k_{ij} = \frac{\partial \boldsymbol{e}_i}{x^j} \cdot \boldsymbol{e}^k. \tag{A.32}$$

The Christoffel symbol is symmetric with respect to the lower indices. This is easily seen through the definition of the basis vectors,

$$\Gamma^k_{ij} = \frac{\partial \boldsymbol{e}_i}{x^j} \cdot \boldsymbol{e}^k = \frac{\partial^2 \boldsymbol{R}}{\partial x^j \partial x^i} \cdot \boldsymbol{e}^k = \frac{\partial \boldsymbol{e}_j}{\partial x^i} \cdot \boldsymbol{e}^k = \Gamma^k_{ji}. \tag{A.33}$$

The derivative of the contravariant basis $\boldsymbol{e}^k$ is computed through

$$-\Gamma^k_{ij} \boldsymbol{e}^i = -(\frac{\partial \boldsymbol{e}_i}{x^j} \cdot \boldsymbol{e}^k)\boldsymbol{e}^i = -\left(\frac{\partial (\boldsymbol{e}_i \cdot \boldsymbol{e}^k)}{\partial x^j} - \frac{\partial \boldsymbol{e}^k}{\partial x^j} \cdot \boldsymbol{e}_i \right) \boldsymbol{e}^i = \frac{\partial \boldsymbol{e}^k}{\partial x^j} \cdot \boldsymbol{e}_i \boldsymbol{e}^i = \frac{\partial \boldsymbol{e}^k}{\partial x^j}. \tag{A.34}$$

where the derivative of the product of basis vectors is zero since the Kronecker delta is a constant. The final equality in Equation (A.34) holds since

$$\frac{\partial \boldsymbol{e}^k}{\partial x^j} \cdot \boldsymbol{e}_i \boldsymbol{e}^i = \left(\frac{\partial \boldsymbol{e}^k}{\partial x^j} \right)_i \boldsymbol{e}^i. \tag{A.35}$$

The Christoffel symbols are incredibly important in defining a derivative that returns tensors when acting on tensors. It is important to note that the Christoffel symbols are not tensors themselves since they transform under a change of coordinates according to

$$\Gamma^k_{ij} = \frac{\partial \boldsymbol{e}_i}{\partial x_j} \cdot \boldsymbol{e}^k = \frac{(\tilde{\boldsymbol{e}}_l J^l_i)}{\partial x^j} \cdot \tilde{\boldsymbol{e}}^m J^k_m = \frac{\partial \tilde{\boldsymbol{e}}_l}{\partial \tilde{x}^n} \cdot \tilde{\boldsymbol{e}}^m J^k_m J^n_j J^l_i + \delta^m_l J^k_m \frac{\partial J^l_i}{\partial x^j} = \tilde{\Gamma}^m_{ln} J^k_m J^n_j J^l_i + \frac{\partial J^l_i}{\partial x^j} J^k_l. \tag{A.36}$$

A.9 The Covariant Derivative

The usual derivative of a tensor is generally not a tensor. One example is the naive definition of the divergence of a tensor T^i given by

$$\frac{\partial T^i}{\partial x^i} = \frac{\partial (\tilde{T}^k J^i_k)}{\partial x^i} = \frac{\partial (\tilde{T}^k J^i_k)}{\partial \tilde{x}^l} J^l_i = \frac{\partial \tilde{T}^k}{\partial \tilde{x}^l} J^i_k J^l_i + \frac{\partial J^i_k}{\partial \tilde{x}^l} J^l_i \tilde{T}^k = \frac{\partial \tilde{T}^i}{\partial \tilde{x}^i} + \frac{\partial J^i_l}{\partial \tilde{x}^k} J^l_i \tilde{T}^k, \tag{A.37}$$

which doesn't transform as a tensor due to the extra term. The given definition of the divergence obviously isn't valid in all choices of coordinates. Thus there arises a need to redefine the derivative for tensors in general. Consider the derivative of the vector $\boldsymbol{v} = v^i \boldsymbol{e}_i$. Through the product rule of derivatives the following is obtained

$$\frac{\partial \boldsymbol{v}}{\partial x^j} = \frac{\partial v^i}{\partial x^j}\boldsymbol{e}_i + v^i \frac{\partial \boldsymbol{e}_i}{\partial x^j} = \frac{\partial v^i}{\partial x^j}\boldsymbol{e}_i + v^i \Gamma^k_{ij}\boldsymbol{e}_k = \left(\frac{\partial v^i}{\partial x^j} + \Gamma^i_{jk} v^k\right)\boldsymbol{e}_i. \tag{A.38}$$

The quantity in the parenthesis in Equation (A.38) keeps track of the rate of change of both the coordinates and the basis vectors. Using the result of Equation (A.38) as inspiration, a new derivative ∇_j of the tensor T^i is defined by

$$\nabla_j T^i = \frac{\partial T^i}{\partial x^j} + \Gamma^i_{jk} T^k, \tag{A.39}$$

and is called the covariant derivative. The covariant derivative coincides with the classical derivative in cartesian coordinates. Contracting the indices i and j in Equation (A.39) gives a new correct divergence operation

$$\nabla_i T^i = \frac{\partial T^i}{\partial x^i} + \Gamma^i_{ik} T^k. \tag{A.40}$$

Combining the results from Equation (A.36) and Equation (A.37) a change of coordinates of the properly defined divergence takes the form

$$\begin{aligned} &\nabla_i T^i = \frac{\partial \tilde{T}^i}{\partial \tilde{x}^i} + \frac{\partial J^i_l}{\partial \tilde{x}^k} J^l_i \tilde{T}^k + \left(\tilde{\Gamma}^m_{ln} J^i_m J^n_k J^l_i + \frac{\partial J^l_i}{\partial x^j} J^k_l\right) T^k = \\ &\frac{\partial \tilde{T}^i}{\partial \tilde{x}^i} + \tilde{\Gamma}^i_{ik}\tilde{T}^k + \frac{\partial J^i_l}{\partial \tilde{x}^k} J^l_i \tilde{T}^k + \frac{\partial J^l_i}{\partial x^k} J^i_l T^k = \frac{\partial \tilde{T}^i}{\partial \tilde{x}^i} + \tilde{\Gamma}^i_{ik}\tilde{T}^k = \tilde{\nabla}_i \tilde{T}^i. \end{aligned} \tag{A.41}$$

The last equality in Equation (A.41) follows from the fact that

$$\frac{\partial J^l_i}{\partial x^k} J^i_l T^k = \frac{\partial J^l_i J^i_l}{\partial x^k} T^k - \frac{\partial J^i_l}{\partial x^k} J^l_i T^k = -\frac{\partial J^i_l}{\partial \tilde{x}^n} J^l_i J^n_k T^k = -\frac{\partial J^i_l}{\partial \tilde{x}^k} \tilde{T}^k J^l_i. \tag{A.42}$$

These calculations show that the divergence defined in Equation (A.40) has the same appearance in all coordinate systems, and therefore motivates the use of the covariant derivative. The covariant derivative of an arbitrary tensor of rank(m, n) is defined as

$$\nabla_k T^{i_1 \ldots i_n}_{j_1 \ldots j_m} = \frac{\partial T^{i_1 \ldots i_n}_{j_1 \ldots j_m}}{\partial x^k} + \Gamma^{i_1}_{kl} T^{l \ldots i_n}_{j_1 \ldots j_m} + \ldots + \Gamma^{i_n}_{kl} T^{i_1 \ldots l}_{j_1 \ldots j_m} - \Gamma^l_{kj_1} T^{i_1 \ldots i_n}_{l \ldots j_m} - \Gamma^l_{kj_m} T^{i_1 \ldots i_n}_{j_1 \ldots l}, \tag{A.43}$$

and is also a tensor [45]. The covariant derivative has all the properties one expects of a derivative such as product-, sum-, and chain-rule and it commutes with contraction. In addition the covariant derivative of several common tensors is zero, a property called the metrinilic property. These tensors include the covariant and contravariant basis vectors, the covariant and contravariant metric tensors, and the Kronecker delta [45]. To make notation more compact, a subscript comma followed by an index denotes covariant differentiation and is written as

$$\nabla_j T^i = T^i_{,j}\ , \quad \nabla_j T_i = T_{i,j}. \tag{A.44}$$

A.10 The Levi-Civita Tensor ε_{ijk}

The permutation symbol of order n is defined by

$$e_{i_1 \ldots i_n} = e^{i_1 \ldots i_n} = \begin{cases} 1 \text{ if } i_1, ..., i_n \text{ is an even permutation of } 1, ..., n \\ -1 \text{ if } i_1, ..., i_n \text{ is an odd permutation of } 1, ..., n \\ 0 \text{ if any indices are repeated,} \end{cases} \tag{A.45}$$

Using the permutation symbol the determinant of a second order tensor T^{ij} can be concisely defined as

$$T = \frac{1}{n!} e_{i_1 \ldots i_n} e_{j_1 \ldots j_n} T^{i_1 j_1} T^{i_2 j_2} ... T^{i_n j_n}, \tag{A.46}$$

where the determinant of a covariant tensor is computed analogously using the permutation symbol with superindices. The permutation symbol is itself not a tensor, but it can be turned into the *Levi-Civita* tensor by scaling with the square root of the determinant of the metric tensor as [45]

$$\varepsilon_{i_1 \ldots i_n} = \sqrt{g} e_{i_1 \ldots i_n} \tag{A.47}$$

for the covariant version or

$$\varepsilon^{i_1 \ldots i_n} = \frac{1}{\sqrt{g}} e^{i_1 \ldots i_n} \tag{A.48}$$

for the contravariant version. In three dimensions the vector-, or cross-product between two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ can be defined with the Levi-Civita tensor as

$$\boldsymbol{u} \times \boldsymbol{v} = \varepsilon_{ijk} u^i v^j \boldsymbol{e}^k. \tag{A.49}$$

An interest consequence of this definition is that it makes it apparent that the cross product yields a pseudo-vector since it turns two vectors with contravariant coordinates into a vector with covariant coordinates.

B Continuum Solid Mechanics

In solid mechanics there is a need to relate the displacements inside a continuous body to the forces acting on it to be able to solve the equations of equilibrium. However, simply trying to find a direct relationship between the displacement and forces contradicts the physics of reality, a rigid body displacement doesn't give rise to internal forces. Not even the displacement gradient is a sufficient measure as rigid body rotations also requires a state of non-existent internal forces. Thus a more sophisticated concept, which is called strain, is required. The concept of force also requires generalization. Simply considering the force balance of a point inside the solid will lead to ambiguous results. Therefore stress, which is more general, is introduced. Elasticity is a special, but very common, material behavior where a deforming body will return to its original shape when any external forces or boundary displacements are removed. The equations of elasticity is the bridge between strain and stress. From now on, all tensors are three dimensional. In other words, indices have the range $i = 1, 2, 3$.

B.1 Strain

Consider a continuum body occupying some region in space Ω with boundary Γ. Let the body be deformed and denote the region and its boundary after deformation by Ω' and Γ' respectively. Refer to Figure B.1 to see a visualization of the deforming process. Pick an arbitrary point inside Ω and call it $\boldsymbol{r}$. Furthermore, pick a point in the vicinity of $\boldsymbol{r}$, and let its position be denoted by $\boldsymbol{r} + \Delta\boldsymbol{r}$. After deformation the first point takes the location $\boldsymbol{r}' = \boldsymbol{r} + \boldsymbol{u}(\boldsymbol{r})$, while the second point moves to $\boldsymbol{r}' + \Delta\boldsymbol{r}' = \boldsymbol{r} + \Delta\boldsymbol{r} + \boldsymbol{u}(\boldsymbol{r} + \Delta\boldsymbol{r})$. In this context $\boldsymbol{u}$ is the displacement field that shows how every point of Ω moves to its place in Ω' seen from a Lagrangian viewpoint. The line segment that connects the two points in Ω is $\Delta\boldsymbol{r}$ by construction, while it is $\Delta\boldsymbol{r}' = \Delta\boldsymbol{r} + \boldsymbol{u}(\boldsymbol{r} + \Delta\boldsymbol{r}) - \boldsymbol{u}(\boldsymbol{r})$ in Ω'. If the two points are sufficiently close together during the course of the displacement, $\boldsymbol{u}(\boldsymbol{r} + \Delta(\boldsymbol{r}))$ can be Taylor expanded to the first order as

$$\Delta\boldsymbol{r}' = \Delta\boldsymbol{r} + \boldsymbol{u}(\boldsymbol{r}) + \nabla\boldsymbol{u}(\boldsymbol{r}) \cdot \Delta\boldsymbol{r} - \boldsymbol{u}(\boldsymbol{r}) = (\boldsymbol{I} + \nabla\boldsymbol{u}(\boldsymbol{r})) \cdot \Delta\boldsymbol{r}, \tag{B.1}$$

where $\boldsymbol{I}$ is the second order identity tensor. It is thus evident that $\Delta\boldsymbol{r}'$ can be expressed as a linear mapping of $\Delta\boldsymbol{r}$ involving the second order displacement gradient tensor. This tensor is often called the deformation tensor $\boldsymbol{F}$. Written on index notation it reads

$$\boldsymbol{F} = F_{ij}\boldsymbol{e}^i\boldsymbol{e}^j = (g_{ij} + u_{i,j})\boldsymbol{e}^i\boldsymbol{e}^j. \tag{B.2}$$

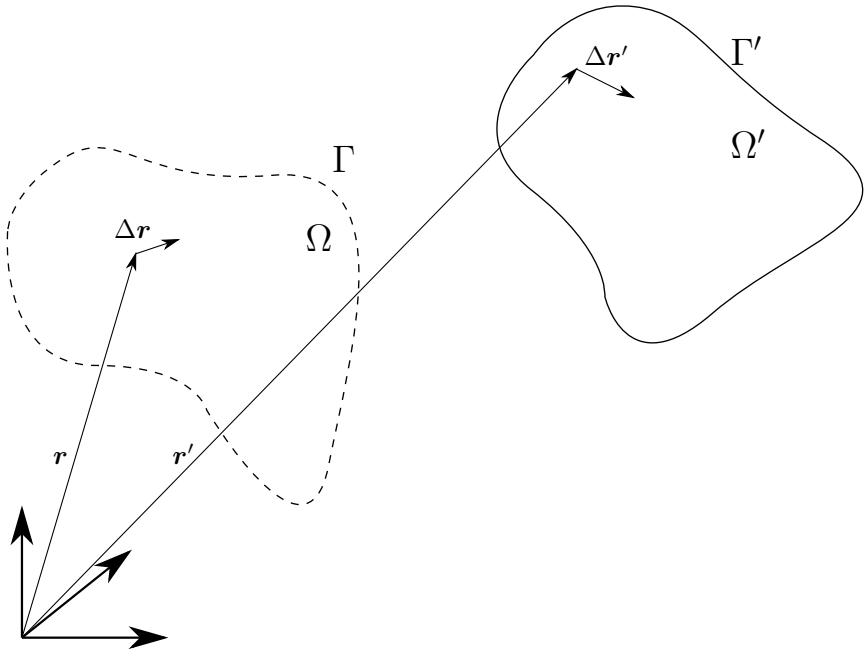


Figure B.1: A continuum body occupying a region Ω with boundary Γ is deformed and until it occupies the region Ω' with boundary Γ'. An arbitrary pair of points $\boldsymbol{r}$ and $\boldsymbol{r} + \Delta\boldsymbol{r}$ in Ω is mapped on the points $\boldsymbol{r}'$ and $\boldsymbol{r}' + \Delta\boldsymbol{r}'$ inside Ω' respectively.

The line segment in Ω' is written in index notation as

$$\Delta r'^i \boldsymbol{e}_i = F_{ij}\boldsymbol{e}^i\boldsymbol{e}^j \cdot \Delta r^k \boldsymbol{e}_k = F_{ij}\delta^j_k \Delta r^k \boldsymbol{e}^i = F_{ij}\Delta r^j \boldsymbol{e}^i, \tag{B.3}$$

and its squared length of thus follows as

$$|\Delta \boldsymbol{r}'|^2 = \Delta \boldsymbol{r}' \cdot \Delta \boldsymbol{r}' = F_{ij}\Delta r^j \boldsymbol{e}^i \cdot F_{kl}\Delta r^l \boldsymbol{e}^k = F_{ij}F_{kl}g^{ik}\Delta r^j \Delta r^l. \tag{B.4}$$

The ratio of the length of the line segment after deformation to its length before deformation is defined as the stretch and is denoted by $\lambda = |\Delta \boldsymbol{r}'|/|\Delta \boldsymbol{r}|$. The normal strain is defined by the ratio between the change in length to the original length as

$$\varepsilon_n = \frac{|\Delta \boldsymbol{r}'| - |\Delta \boldsymbol{r}|}{|\Delta \boldsymbol{r}|} = \lambda - 1. \tag{B.5}$$

The difference of the squared length before and after the deformation can be expressed as

$$\begin{aligned} |\Delta \boldsymbol{r}'|^2 - |\Delta \boldsymbol{r}|^2 = (F_{ij}F_{kl}g^{ik} - g_{jl})\Delta r^j \Delta r^l = ((g_{ij} + u_{i,j})(g_{kl} + u_{k,l})g^{ik} - g_{jl})\Delta r^j \Delta r^l = \\ ((g_{ij}g_{kl} + g_{ij}u_{k,l} + u_{i,j}g_{kl} + u_{i,j}u_{k,l})g^{ik} - g^{jl})\Delta r^j \Delta r^l = (u_{j,l} + u_{l,j} + u_{i,j}u_{k,l}g^{ik})\Delta r^j \Delta r^l. \end{aligned} \tag{B.6}$$

If $\boldsymbol{n} = n^i \boldsymbol{e}_i$ is the unit vector in the direction of $\Delta \boldsymbol{r} = \Delta r^i \boldsymbol{e}_i$ the strain of the line segment is written as

$$\frac{|\Delta \boldsymbol{r}'|^2 - |\Delta \boldsymbol{r}|^2}{|\Delta \boldsymbol{r}|^2} = (u_{i,j} + u_{j,i} + u_{k,i}u_{l,j}g^{kl})n^i n^j = n^i \boldsymbol{e}_i \cdot \boldsymbol{e}^k 2E_{kl}\boldsymbol{e}^l \cdot n^j \boldsymbol{e}_j. \tag{B.7}$$

The strain tensor is followingly defined as

$$\boldsymbol{E} = E_{ij}\boldsymbol{e}^i\boldsymbol{e}^j = \frac{1}{2}(u_{i,j} + u_{j,i} + u_{k,i}u_{l,j}g^{kl})\boldsymbol{e}^i\boldsymbol{e}^j. \tag{B.8}$$

The strain tensor is related to the normal strain of a line with direction $\boldsymbol{n}$ through

$$\varepsilon_n = \sqrt{\boldsymbol{n} \cdot 2\boldsymbol{E} \cdot \boldsymbol{n} + 1} - 1, \tag{B.9}$$

and to the stretch through

$$\lambda = \sqrt{\boldsymbol{n} \cdot 2\boldsymbol{E} \cdot \boldsymbol{n} + 1}. \tag{B.10}$$

$\boldsymbol{E}$ is a second order tensor and is symmetric. Now consider two line segments $\Delta \boldsymbol{r}$ and $\Delta \boldsymbol{s}$ with unit directions $\boldsymbol{n}$ and $\boldsymbol{m}$ respectively, which can be seen in Figure B.2. If the line segments share their origin the angle that between them is given by

$$\theta = \arccos(\boldsymbol{n} \cdot \boldsymbol{m}) \tag{B.11}$$

After deformation the two line segments satisfy the scalar product

$$\cos(\theta)|\Delta \boldsymbol{r}'||\Delta \boldsymbol{s}'| = g^{il}\Delta r^j \Delta s^k F_{ij}F_{lk} = (2E_{jk} + g_{jk})\Delta r^j \Delta s^k = \boldsymbol{n} \cdot (\boldsymbol{E} + \boldsymbol{I}) \cdot \boldsymbol{m}|\Delta \boldsymbol{r}||\Delta \boldsymbol{s}|, \tag{B.12}$$

and the angle between by the deformed segments is thus given by

$$\theta' = \arccos\left(\frac{\boldsymbol{n} \cdot (2\boldsymbol{E} + \boldsymbol{I}) \cdot \boldsymbol{m}}{\sqrt{\boldsymbol{n} \cdot 2\boldsymbol{E} \cdot \boldsymbol{n} + 1}\sqrt{\boldsymbol{m} \cdot 2\boldsymbol{E} \cdot \boldsymbol{m} + 1}}\right). \tag{B.13}$$

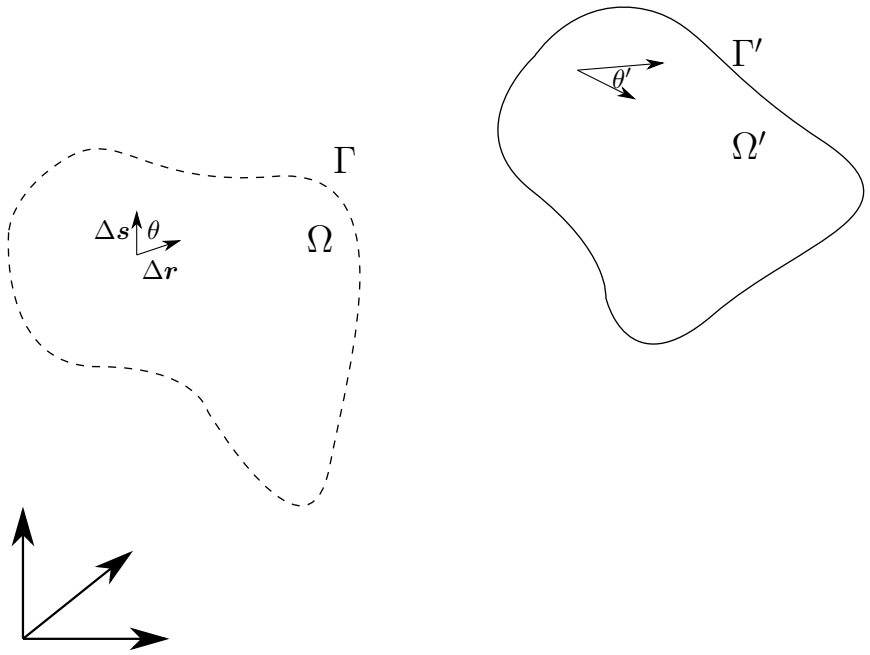


Figure B.2: The shearing of the angle θ between by the line segments $\Delta \boldsymbol{r}$ and $\Delta \boldsymbol{s}$ is displayed. After deformation the angle between by the line segments change into θ'.

In Equation (B.13) the fact that $|\Delta \boldsymbol{r}'| = |\Delta \boldsymbol{r}|\lambda$ is utilized. The change of the angle can then be easily computed as $\Delta\theta = \theta' - \theta$. It is evident that the strain tensor $\boldsymbol{E}$ contains all information required to describe the stretch of line segments through Equation (B.10) and the shearing of angles through Equations (B.11) and (B.13). Since these processes are measures of actual deformation in a material, i.e. excluding rigid body rotations, the strain is a natural choice of material kinematics. To conclude the discussion on strain the strain tensor is adapted for linear elasticity. As long as the studied deformation is small, the second order term $u_{k,i}u_{l,j}g^{lk}$ contained in $\boldsymbol{E}$ becomes negligible and the strain tensor becomes the small strain tensor which is defined by

$$\boldsymbol{\varepsilon} = \varepsilon_{ij}\boldsymbol{e}^i\boldsymbol{e}^j = \frac{1}{2}(u_{i,j} + u_{j,i})\boldsymbol{e}^i\boldsymbol{e}^j. \tag{B.14}$$

This is also often refereed to as engineering strain and is the most common strain measure for most engineering applications where the deformations may be assumed small.

B.2 Stress

An infinitesimal tetrahedral region of a solid in equilibrium is constructed by intersecting three mutually orthogonal intersecting lines with a plane. Let the largest of the four faces be called dA_0 and have the normal vector $\boldsymbol{n}_0$. Let the three remaining faces have areas dA_1, dA_2, and dA_3, and

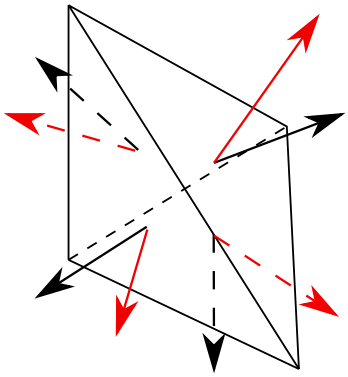


Figure B.3: An infinitesimal right angled tetrahedral sub-region of some solid material. The tetrahedron is in force equilibrium. The black arrows denote the normals of the tetrahedron faces while the red arrows represent the surface traction acting on each of the faces.

let them have the normals $\boldsymbol{n}_1$, $\boldsymbol{n}_2$, and $\boldsymbol{n}_3$ respectively. The tetrahedron can be seen in Figure (B.3). Any region of the solid must be in equilibrium, so the tractions acting on the tetrahedron must be self equilibrating if no other forces are acting on the body. As the faces of the tetrahedron are very small, the traction acting on the tetrahedron may be considered constant on each face. Equilibrium of the tetrahedron requires that

$$dA_0\boldsymbol{t}(\boldsymbol{n}_0) + dA_1\boldsymbol{t}(\boldsymbol{n}_1) + dA_2\boldsymbol{t}(\boldsymbol{n}_2) + dA_3\boldsymbol{t}(\boldsymbol{n}_3) = \boldsymbol{0}, \tag{B.15}$$

where $\boldsymbol{t}$ is the traction (force per unit area) as a function of the outward facing normal. The relationship between the different face areas and the normals can be obtained by considering Gauss divergence theorem where the function being integrated is the normal vectors:

$$\begin{aligned} &\int_S \boldsymbol{n}_\alpha \cdot \boldsymbol{n} dS = \int_V \boldsymbol{\nabla} \cdot \boldsymbol{n}_i dV \quad \alpha = 1,2,3 \\ &dA_0\boldsymbol{n}_1 \cdot \boldsymbol{n}_0 + dA_1 = 0 \\ &dA_0\boldsymbol{n}_2 \cdot \boldsymbol{n}_0 + dA_2 = 0 \\ &dA_0\boldsymbol{n}_3 \cdot \boldsymbol{n}_0 + dA_3 = 0. \end{aligned} \tag{B.16}$$

By combining Equations (B.15) and (B.16) and canceling dA_0 the following equilibrium condition is received

$$\boldsymbol{t}(\boldsymbol{n}_0) + \boldsymbol{n}_1 \cdot \boldsymbol{n}_0\boldsymbol{t}(\boldsymbol{n}_1) + \boldsymbol{n}_2 \cdot \boldsymbol{n}_0\boldsymbol{t}(\boldsymbol{n}_2) + \boldsymbol{n}_3 \cdot \boldsymbol{n}_0\boldsymbol{t}(\boldsymbol{n}_3) = \boldsymbol{0}. \tag{B.17}$$

Newtons third law requires that $\boldsymbol{t}(\boldsymbol{n}) = -\boldsymbol{t}(-\boldsymbol{n})$, thus the traction on the surface dA_0 is given by

$$\boldsymbol{t}(\boldsymbol{n}_0) = \boldsymbol{t}(-\boldsymbol{n}_1)\boldsymbol{n}_1 \cdot \boldsymbol{n}_0 + \boldsymbol{t}(-\boldsymbol{n}_2)\boldsymbol{n}_2 \cdot \boldsymbol{n}_0 + \boldsymbol{t}(-\boldsymbol{n}_3)\boldsymbol{n}_3 \cdot \boldsymbol{n}_0, \tag{B.18}$$

or written more compactly as

$$\boldsymbol{t}(\boldsymbol{n}_0) = (\boldsymbol{t}(-\boldsymbol{n}_1)\boldsymbol{n}_1 + \boldsymbol{t}(-\boldsymbol{n}_2)\boldsymbol{n}_2 + \boldsymbol{t}(-\boldsymbol{n}_3)\boldsymbol{n}_3) \cdot \boldsymbol{n}_0. \tag{B.19}$$

Let the traction be written on index notation as

$$\boldsymbol{t}(-\boldsymbol{n}_\alpha) = t^i_\alpha \boldsymbol{e}_i \quad \alpha = 1,2,3 \tag{B.20}$$

and let the normal vectors be written as

$$\boldsymbol{n}_\alpha = n^i_\alpha \boldsymbol{e}_i \quad \alpha = 1, 2, 3. \tag{B.21}$$

Using Equations (B.20) and (B.21) the equilibrium condition is written on index form as

$$\boldsymbol{t}(\boldsymbol{n}_0) = (t^i_1 n^j_1 \boldsymbol{e}_i \boldsymbol{e}_j + t^i_2 n^j_2 \boldsymbol{e}_i \boldsymbol{e}_j + t^i_3 n^j_3 \boldsymbol{e}_i \boldsymbol{e}_j) \cdot n^k_0 \boldsymbol{e}_k. \tag{B.22}$$

Following this the *Cauchy* stress tensor is defined as

$$\boldsymbol{\sigma} = \sigma^{ij} \boldsymbol{e}_i \boldsymbol{e}_j = (t^i_1 n^j_1 + t^i_2 n^j_2 + t^i_3 n^j_3) \boldsymbol{e}_i \boldsymbol{e}_j, \tag{B.23}$$

which reduces to a simple expression in a coordinate system aligned with the lines used to construct the tetrahedron. In such a coordinate system the ijth component of the stress tensor becomes the ith component of the reaction to the traction vector acting on the jth face. The diagonal components become the normal tractions and the off-diagonal components becomes the shears. In other words, if the traction is known on three orthogonal surfaces intersecting in a point, the traction on an arbitrary surface intersecting that point is known. By letting the tetrahedron shrink to a point, the stress tensor extracts the traction on an arbitrary cross section with normal $\boldsymbol{n}$ anywhere inside the solid through

$$\boldsymbol{t}(\boldsymbol{n}) = \boldsymbol{\sigma} \cdot \boldsymbol{n} = \sigma^{ij} n_j \boldsymbol{e}_i. \tag{B.24}$$

Consider force equilibrium inside an arbitrary subregion of a solid. The forces acting on the region is some force per unit volume $\boldsymbol{b} = b^i \boldsymbol{e}_i$ and a surface traction. The equilibrium condition reads

$$\int_V b^i \boldsymbol{e}_i dV + \int_S \sigma^{ij} n_j \boldsymbol{e}_i dS = \boldsymbol{0}. \tag{B.25}$$

Using Gauss' divergence theorem yields

$$\int_V (b^i + \sigma^{ij}_{,j}) \boldsymbol{e}_i dV = \boldsymbol{0}. \tag{B.26}$$

However, since this must be true for any subregion, the quantity under the integral sign must be zero. Thus, the general equilibrium relation reads

$$\sigma^{ij}_{,j} = -b^i. \tag{B.27}$$

Now consider moment equilibrium. Moment equilibrium is stated by integrating the cross product of forces and position like

$$\int_V b^i x^j \varepsilon_{ijk} \boldsymbol{e}^k dV + \int_S \sigma^{il} n_l x^j \varepsilon_{ijk} \boldsymbol{e}^k dS = \boldsymbol{0}, \tag{B.28}$$

which by using Gauss' theorem turns into

$$\int_V (b^i x^j + \sigma^{il}_{,l} x^j + \sigma^{il} x^j_{,l}) \varepsilon_{ijk} \boldsymbol{e}^k dV = \boldsymbol{0}. \tag{B.29}$$

By inserting the result from Equation (B.27) into Equation (B.29) and using the fact that $x^j_{,l} = \delta^j_l$ it follows that

$$\int_V \sigma^{ij} \varepsilon_{ijk} \boldsymbol{e}^k dV = \boldsymbol{0}. \tag{B.30}$$

Using the same reasoning as for the force equilibrium, the quantity under the integral sign must equate to zero. The Levi-Civita tensor is anti-symmetric, thus it follows that the stress tensor must be symmetric since the product of a symmetric and anti-symmetric tensor is zero.

B.3 Linear Elasticity

The reason of introducing the concepts of strain and stress is to be able to relate displacements of a continuum to the internal forces. The forces are subsequently used to establish equations of equilibrium. One final concept is required to bridge strain and stress, a constitutive relationship. In linear elasticity the linear small strain tensor is related to the stress. Since the small strain tensor is linear in displacements by definition, the equations of elasticity inherits this property.

The most general assumption is that every component of the strain tensor influences every component of the stress tensor. This is accomplished by a linear transformation through a fourth order tensor

$$\boldsymbol{\sigma} = \boldsymbol{C} : \boldsymbol{\varepsilon}, \tag{B.31}$$

where the : symbol refers to the double contraction $\boldsymbol{e}_i\boldsymbol{e}_j : \boldsymbol{e}_k\boldsymbol{e}_l = \boldsymbol{e}_i \cdot \boldsymbol{e}_k\boldsymbol{e}_j \cdot \boldsymbol{e}_l$. In index notation Equation (B.31) becomes

$$\sigma^{ij}\boldsymbol{e}_i\boldsymbol{e}_j = C^{ijkl}\boldsymbol{e}_i\boldsymbol{e}_j\boldsymbol{e}_k \cdot \boldsymbol{e}^m\boldsymbol{e}_l \cdot \boldsymbol{e}^n\varepsilon_{mn}. \tag{B.32}$$

The tensor $\boldsymbol{C}$ is called the stiffness tensor and the relation in Equation (B.31) is called the generalized Hooke's law. The stiffness tensor seem to have $3^4 = 81$ independent components, but that can immediately be reduced to 36 by using the symmetries of $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$. That is $C^{ijkl} = C^{jikl} = C^{jilk} = C^{ijlk}$, and this is called the minor symmetry of $\boldsymbol{C}$. It is possible to reduce the degrees of freedom of $\boldsymbol{C}$ further to 21. If it assumed that there is a strain energy density ψ, differentiation yields

$$\frac{\partial \psi}{\partial \varepsilon_{ij}} = \sigma^{ij}, \tag{B.33}$$

By inserting Hooke's law the stifness tensor can be derived from the strain energy like

$$C^{ijkl} = \frac{\partial^2 \psi}{\partial \varepsilon_{ij} \partial \varepsilon_{kl}}. \tag{B.34}$$

Since order of differentiation is arbitrary, so called major symmetry of the stiffness tensor $C^{ijkl} = C^{klij}$ becomes apparent.

There is a particularly easy way to express the Hooke's law if the solid being treated is isotropic. In orthogonal coordinate systems the stiffness tensor then takes the form

$$C^{ijkl} = \lambda g^{ij} g^{kl} + \mu(g^{ik} g^{jl} + g^{il} g^{jk}), \tag{B.35}$$

which only have 9 non-zero components. Here λ and μ are the *Lamé* parameters and are related to the engineering elasticity constants (Young's modulus E and Poisson's ratio ν) through

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \quad \mu = \frac{E}{2(1+\nu)}. \tag{B.36}$$

The presented theory on linear elasticity relies on the fact that the stress is given by the derivative of an energy density with respect to the strain. To support this claim a thermodynamic inquiry is necessary.

B.4 Continuum Thermodynamics

Consider a subregion V with boundary S of a solid under deformation. The region has an original configuration V_0 and S_0 before it is deformed. Let M be the mass of the region. Conservation of mass requires that the integral of the density is constant for any configuration of the solid,

$$M = \int_V \rho dV = \int_{V_0} \rho_0 dV, \tag{B.37}$$

where $\rho(t)$ is the mass density of the current configuration and ρ_0 is the time independent mass density of the original configuration. The integral over the current configuration V can be mapped on the initial configuration V_0 through a change of coordinates. Therefore,

$$M - M = 0 = \int_{V_0} \rho_0 - J\rho dV, \tag{B.38}$$

where J is the Jacobian of the coordinate change. This must be true for any initial configuration V_0, which implies that

$$\rho_0 = \rho J. \tag{B.39}$$

The kinetic energy of V is given by

$$K = \frac{1}{2}\int_V \dot{\boldsymbol{u}} \cdot \dot{\boldsymbol{u}} \rho dV, \tag{B.40}$$

where ρ is the mass density of the solid and the dot denotes time differentiation. The change in kinetic energy follows as

$$\dot{K} = \int_V \ddot{\boldsymbol{u}} \cdot \dot{\boldsymbol{u}} \rho dV = \int_V \ddot{u}_i \dot{u}_j g^{ij} \rho dV. \tag{B.41}$$

The time derivative doesn't act on ρ since conservation of mass requires that ρdV is constant. If it assumed that there is an internal energy per unit mass e, then the change of internal energy is

$$\dot{U} = \int_V \rho \dot{e} dV. \tag{B.42}$$

The external work rate acting on the solid is

$$W = \int_V \boldsymbol{b} \cdot \dot{\boldsymbol{u}} dV + \int_V \boldsymbol{t} \cdot \dot{\boldsymbol{u}} = \int_V b^i \dot{u}_i dV + \int_S \sigma^{ij} n_j \dot{u}_i dS. \tag{B.43}$$

Using the divergence theorem, (B.43) is rewritten on the form

$$W = \int_V (b^i + \sigma^{ij}_{,j})\dot{u}_i + \sigma^{ij}\dot{u}_{i,j} dV. \tag{B.44}$$

The heat transfer is given by

$$Q = \int_V \rho h dV - \int_S \boldsymbol{q} \cdot \boldsymbol{n} dS, \tag{B.45}$$

where h is the heat production per unit mass and $\boldsymbol{q}$ is the heat flow vector. Gauss theorem once again yields

$$Q = \int_V (\rho h - q_{i,i}) dV. \tag{B.46}$$

The first law of thermodynamics states that the total energy must be conserved, therefore the change of kinetic energy and internal energy summed must be equal to the sum external workrate and heat transfer

$$\dot{K} + \dot{U} = W + Q. \tag{B.47}$$

All the quantities from Equations (B.41), (B.42), (B.44), and (B.46) are written as volume integrals, so their sum can be written as

$$\int_V \rho \dot{e} + \ddot{u}_i \dot{u}_j g^{ij} \rho - (b^i + \sigma^{ij}_{,j}) \dot{u}_i - \sigma^{ij} \dot{u}_{i,j} - \rho h + q^i_{,i} dV = 0. \tag{B.48}$$

By using Newton's second law, the second and third terms of (B.48) cancel. Besides, the integral can be mapped on the initial configuration through a change of coordinates and becomes

$$\int_{V_0} \rho_0 \dot{e} - \sigma^{ij} \dot{u}_{i,j} J - \rho_0 h + J q^i_{,i} dV = 0 \tag{B.49}$$

Using the argument that this relation must hold for arbitrary regions of the solid, it is localized and Equation (B.49) becomes the balance law for energy:

$$\rho_0 \dot{e} = \sigma^{ij} \dot{u}_{i,j} J + \rho_0 h - J q^i_{,i}. \tag{B.50}$$

The second law of thermodynamics states that the global change in entropy is non-negative

$$dS - \frac{dQ}{T} \geq 0, \tag{B.51}$$

where T is the absolute temperature of the surroundings of the studied piece of solid. The change in entropy therefore obeys

$$\dot{S} - Q_T = \int_V \rho \dot{s} dV - \int_V \frac{\rho h}{T} dV + \int_S \frac{1}{T} \boldsymbol{q} \cdot \boldsymbol{n} dS \geq 0, \tag{B.52}$$

where s is the entropy per unit mass of the solid. By Gauss' divergence theorem Equation (B.52) transforms into

$$\dot{S} - Q_T = \int_V \rho \dot{s} - \frac{\rho h}{T} + \frac{1}{T} q^i_{,i} - \frac{1}{T^2} T_{,i} q_j g^{ij} dV \geq 0. \tag{B.53}$$

The final term in the integral of Equation (B.53) can absorbed by the inequality since it is always greater than 0. This follows from the fact that the heat flow is always parallel to the temperature gradient. By using the localization argument as for the other balance equations and by exploiting Equation (B.39), the *energy dissipation rate* is received

$$\rho_0 \dot{s} T - \rho_0 h + J q_{i,i} \geq 0. \tag{B.54}$$

Combining Equations (B.50) and (B.54) results in

$$\rho_0 \dot{s} T - \rho_0 \dot{e} + \sigma^{ij} \dot{u}_{i,j} J \geq 0. \tag{B.55}$$

If the strains are assumed small the product of the *Kirchoff* stress $J\sigma^{ij}$ and the gradient of the displacement rate $\dot{u}_{i,j}$, will be equal to the product of the Cauchy stress σ^{ij} and the small strain rate tensor $\dot{\varepsilon}_{ij}$. Morover, if the *Helmholtz* free energy density is defined as

$$\psi = \rho_0 e - \rho_0 s T, \tag{B.56}$$

Equation (B.55) then reads

$$\sigma^{ij}\dot{\varepsilon}_{ij} - \rho_0 s\dot{T} - \dot{\psi} \geq 0. \tag{B.57}$$

If the Helmholtz free energy is taken as a function of the strain, temperature, and an arbitrary number of other internal variables $\psi(\boldsymbol{\varepsilon}, T, k)$ Equation (B.57) takes the form

$$(\sigma^{ij} - \frac{\partial\psi}{\partial\varepsilon_{ij}})\dot{\varepsilon}_{ij} - (\rho_0 s + \frac{\partial\psi}{\partial T})\dot{T} - \frac{\partial\psi}{\partial k}\dot{k} \geq 0. \tag{B.58}$$

This inequality needs to be valid for any strain rate $\dot{\varepsilon}_{ij}$, rate of temperature change $\dot{T}$, and rate of change relating the internal variables $\dot{k}$, regardless of the current stress- or strain state or entropy density. The only possibility for this to hold is that the expressions in the parenthesis are zero, and that the final term is non-negative. Therefore the assumption on the existence of an energy density made in section (B.3) is motivated through

$$\sigma^{ij} = \frac{\partial\psi}{\partial\varepsilon_{ij}}. \tag{B.59}$$

In addition, the entropy density is given by

$$s = -\frac{1}{\rho_0}\frac{\partial\psi}{\partial T}, \tag{B.60}$$

and the change in the other internal variables must adhere to

$$-\frac{\partial\psi}{\partial k}\dot{k} \geq 0. \tag{B.61}$$

As this inequality pertains to the energy dissipation rate, it follows that the dissipation rate for a solid where the Helmholtz free energy is dependent on n internal variables (excluding elastic strain and temperature) is

$$\mathcal{D} = \dot{S} - Q_T = \sum_{i=1}^{n} -\frac{\partial\psi}{\partial k_i}\dot{k}_i = \sum_{i=1}^{n} \kappa_i \dot{k}_i \geq 0. \tag{B.62}$$

The factors

$$\kappa_i = -\frac{\partial\psi}{\partial k_i}, \tag{B.63}$$

look very similar to how the stress was computed in Equation (B.59). As a consequence they are called dissipative stresses. The dissipation rate is important for the theoretical basis of plasticity.

C Fundamental Solution of Elasticity

A fundamental solution G pertaining a linear differential operator $\mathcal{L}$ is the function that satisfies

$$\mathcal{L}G(\boldsymbol{x}, \boldsymbol{x}') = -\delta(\boldsymbol{x} - \boldsymbol{x}'). \tag{C.1}$$

Here δ is the *Dirac* delta-distribution that is zero for all non-zero arguments and satisfies

$$\int_{-\infty}^{\infty} \delta(\boldsymbol{x}) dV = 1. \tag{C.2}$$

An important property of the Dirac distribution is that it is the identity element of convolution, i.e.

$$\int_{-\infty}^{\infty} f(\boldsymbol{x}')\delta(\boldsymbol{x} - \boldsymbol{x}')dV' = f(\boldsymbol{x}). \tag{C.3}$$

Equation (C.3) implies that if $G(\boldsymbol{x}, \boldsymbol{x}')$ is the fundamental solution of the operator $\mathcal{L}$ then

$$F(\boldsymbol{x}) = \int_{-\infty}^{\infty} G(\boldsymbol{x}, \boldsymbol{x}')f(\boldsymbol{x}')dV' \tag{C.4}$$

is the solution of $\mathcal{L}F(\boldsymbol{x}) = -f(\boldsymbol{x})$. This is easily verified by differentiation under the integral sign in Equation (C.4) and using the definition of G given in Equation (C.1).

The concept of fundamental solutions can be generalized to vector-valued differential equations. If the linear differential operator $\mathcal{L}^{ik}$ acts on a vector object F_k, the fundamental solution is taken to be a second order tensor as follows:

$$\mathcal{L}^{ik}G_{jk}(\boldsymbol{x}, \boldsymbol{x'}) = -\delta(\boldsymbol{x} - \boldsymbol{x}')\delta^i_j. \tag{C.5}$$

Note that one of the deltas in Equation (C.5) is the Dirac delta, while the other is the Kronecker delta. Analogously to Equation (C.4) the solution to the equation $\mathcal{L}^{ik}F_k(\boldsymbol{x}) = -f^i(\boldsymbol{x})$ is then given by

$$F_k(\boldsymbol{x}) = \int_{-\infty}^{\infty} G_{jk}(\boldsymbol{x}, \boldsymbol{x}')f^j(\boldsymbol{x}')dV', \tag{C.6}$$

and is verified in the exact same fashion as earlier through differentiation under the integral sign.

C.1 Response of an Infinite Medium Due to a Point Force

The response of an infinite, linearly elastic medium due to a point force is the fundamental solution of the Cauchy-Navier equation of a linearly elastic isotropic medium. The Cauchy-Navier equation is easily derived by combining the equilibrium condition of a continuum, the definition of the infinitesimal strain, and Hooke's law. The computations in this section assumes a coordinate system whose metric tensor g_{ij} is constant in space. The equilibrium of a continuum is stated by the relation

$$\sigma^{ij}_{,j} = -b^i. \tag{C.7}$$

The infinitesimal strain tensor is defined by

$$\varepsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right). \tag{C.8}$$

Hooke's law is given by

$$\sigma^{ij} = C^{ijkl}\varepsilon_{kl}, \tag{C.9}$$

where C^{ijkl} has major and minor symmetry. By insering Equation (C.8) into Equation (C.9) and by exploiting the minor symmetry of C^{ijkl} the stress is expressed by

$$\sigma_{ij} = C^{ijkl}u_{k,l}. \tag{C.10}$$

By inserting Equation (C.10) into Equation (C.7) the Cauchy-Navier equation is attained, and takes the appearance

$$\left(C^{ijkl}u_{k,l}\right)_{,j} = -b^i. \tag{C.11}$$

By finally using the fact that the stiffness tensor of a linearly elastic isotropic medium is given by

$$C^{ijkl} = \lambda g^{ij}g^{kl} + \mu(g^{ik}g^{jl} + g^{il}g^{jk}), \tag{C.12}$$

the isotropic Cauchy-Navier is then ultimately obtained as

$$\left(\lambda g^{ij}g^{kl} + \mu(g^{ik}g^{jl} + g^{il}g^{jk})\right)u_{k,lj} = -b^i. \tag{C.13}$$

The differential operator of the isotropic Cauchy-Navier equation is

$$\mathcal{L}^{ik} = \left(\lambda g^{in}g^{kl} + \mu(g^{ik}g^{nl} + g^{il}g^{nk})\right)\nabla_l\nabla_n, \tag{C.14}$$

where the summation index j has been switched to n to free up the symbol j. Insertion of this differential operator into defining Equation (C.5) yields the equation

$$\left(\lambda g^{in}g^{kl} + \mu(g^{ik}g^{nl} + g^{il}g^{nk})\right)G_{jk,ln} = -\delta(\boldsymbol{x} - \boldsymbol{x}')\delta^i_j. \tag{C.15}$$

This equation is most easily solved in the Fourier domain. The Fourier transform is defined by

$$\hat{f}(\boldsymbol{k}) = \mathcal{F}(f(\boldsymbol{x})) = \int_{-\infty}^{\infty} f(\boldsymbol{x})e^{-i\boldsymbol{k}\cdot\boldsymbol{x}}d\boldsymbol{x}, \tag{C.16}$$

while the inverse Fourier transform is defined through

$$f(\boldsymbol{x}) = \mathcal{F}^{-1}(\hat{f}(\boldsymbol{k})) = \frac{1}{(2\pi)^3}\int_{-\infty}^{\infty} \hat{f}(\boldsymbol{k}) e^{i\boldsymbol{k}\cdot\boldsymbol{x}} d\boldsymbol{k}, \tag{C.17}$$

where $i^2 = -1$ is the imaginary unit. A Fourier transformed quantity f is denoted with a hat like $\hat{f}$. The following convenient properties of the Fourier transform are employed [12]:

$$\begin{aligned} &\mathcal{F}(f(\boldsymbol{x}-\boldsymbol{x'})) = e^{-i\boldsymbol{k}\cdot\boldsymbol{x}'} \\ &\mathcal{F}(\nabla_j f(\boldsymbol{x})) = ik_j\hat{f}(\boldsymbol{k}) \\ &\mathcal{F}(\delta(\boldsymbol{x})) = 1 \\ &\mathcal{F}(\mathcal{F}(f(\boldsymbol{x}))) = (2\pi)^3 f(-\boldsymbol{x}). \end{aligned} \tag{C.18}$$

The Fourier transform of Equation (C.15) (remembering that g^{ij} is assumed constant in space) leads to

$$\left(\lambda g^{in}g^{kl} + \mu(g^{ik}g^{nl} + g^{il}g^{nk})\right) k_l k_n \hat{G}_{jk} = \delta^i_j e^{-i\boldsymbol{k}\cdot\boldsymbol{x}'}. \tag{C.19}$$

This simplifies considerably to

$$\left((\mu+\lambda)k^i k^k + \mu g^{ik}k^n k_n\right)\hat{G}_{jk} = \delta^i_j e^{-i\boldsymbol{k}\cdot\boldsymbol{x}'}, \tag{C.20}$$

where the tensor multiplied with $\hat{G}_{jk}$ often is called the *Acoustic* tensor. To proceed, an inverse of the acoustic tensor needs to be found. Equation (C.20) can be further simplified to reveal some useful properties of the acoustic tensor. The equation becomes

$$\mu k^2\left(\frac{2\mu+\lambda}{\mu}n^i n^k + (g^{ik} - n^i n^k)\right)\hat{G}_{jk} = \delta^i_j e^{-i\boldsymbol{k}\cdot\boldsymbol{x}'}, \tag{C.21}$$

where $n^i = k^i/k$ and k is the norm of k^i. The tensor $n^i n^k$ called the *projection* tensor has the property

$$n^i n^k n_i n_l = n^k n_l \tag{C.22}$$

while tensor $(g^{ik} - n^i n^k)$ has the property

$$(g^{ik} - n^i n^k)(g_{il} - n_i n_l) = \delta^k_l - n^k n_l - n^k n_l + n^k n_l = (\delta^k_l - n^k n_l) \tag{C.23}$$

In addition, they are orthogonal through

$$n^i n^k (g_{kl} - n_k n_l) = n^k n_l - n^k n_l = 0^k_l. \tag{C.24}$$

Exploiting the properties presented in Equations (C.22), (C.23), and (C.24) it is easy to see that the inverse of the acoustic tensor is

$$\frac{1}{\mu k^2}\left(\frac{\mu}{2\mu+\lambda}n_i n_l + (g_{il} - n_i n_l)\right), \tag{C.25}$$

and the solution in the Fourier domain follows as

$$\hat{G}_{jk} = \frac{g_{jk}}{\mu}\frac{e^{-i\boldsymbol{k}\cdot\boldsymbol{x}'}}{k^2} - \frac{\mu+\lambda}{(2\mu+\lambda)\mu}\frac{e^{-i\boldsymbol{k}\cdot\boldsymbol{x}'}k_j k_k}{k^4}. \tag{C.26}$$

Taking the inverse Fourier transform of Equation (C.26) leads to the fundamental solution of the Cauchy-Navier equation.

To compute the inverse Fourier transform of k^{-2} one utilizes the fundamental solution of the Laplace operator [12]

$$G_{\Delta}(\boldsymbol{x}) = \frac{1}{4\pi|\boldsymbol{x}|}, \tag{C.27}$$

and the derivative property of the Fourier transform:

$$\mathcal{F}(\nabla_k \nabla^k G_{\Delta}) = \mathcal{F}(-\delta(\boldsymbol{x})) \Longrightarrow \mathcal{F}(G_{\Delta}) = \frac{1}{k^2} = \mathcal{F}\left(\frac{1}{4\pi|\boldsymbol{x}|}\right). \tag{C.28}$$

The inverse of $k_j k_k k^{-4}$ is slighly more involved to derive. Using the final property listen in Equation (C.18) on the Laplacian fundamental solution one obtains

$$\mathcal{F}\left(\frac{1}{|\boldsymbol{x}|^2}\right) = \frac{2\pi^2}{|\boldsymbol{k}|}. \tag{C.29}$$

By applying the derivative property on the result in Equation (C.29) the following is produced

$$\mathcal{F}\left(\frac{x_j}{|\boldsymbol{x}|^4}\right) = -\frac{i\pi^2 k_j}{|\boldsymbol{k}|}, \tag{C.30}$$

which in turn can be rewritten (using the final property of Equation (C.18) once again) as

$$\mathcal{F}\left(\frac{x_j}{8\pi|\boldsymbol{x}|}\right) = -\frac{ik_j}{k^4}. \tag{C.31}$$

By applying the derivative property once again on Equation (C.31) and using the first property of Equation (C.18) the final result follows:

$$G_{jk}(\boldsymbol{x}, \boldsymbol{x}') = \frac{1}{4\pi\mu|\boldsymbol{x} - \boldsymbol{x}'|} g_{jk} - \frac{\mu + \lambda}{8\pi(2\mu + \lambda)\mu} |\boldsymbol{x} - \boldsymbol{x}'|_{,jk}, \tag{C.32}$$

where the chain rule of differentiation has been used backwards on the final term.

To recapitulate: the fundamental solution G_{ij} will return the displacement response u_i due to an applied point force $b^j(\boldsymbol{x}) = \delta(\boldsymbol{x} - \boldsymbol{x}')P^j$ in the point $\boldsymbol{x}'$ of an infinite isotropic elastic medium through

$$u_i(\boldsymbol{x}) = G_{ij}(\boldsymbol{x}, \boldsymbol{x}')P^j. \tag{C.33}$$

This is easily seen by applying the Cauchy-Navier differential operator to Equation (C.33) and recalling the definition of the vector valued fundamental solution. By integrating the contribution of an arbitrary distribution of point forces, i.e. any type of body force, the general solution of the infinite linearly elastic isotropic body is given by

$$u_i(\boldsymbol{x}) = \int_{-\infty}^{\infty} G_{ij}(\boldsymbol{x}, \boldsymbol{x}')b^j(\boldsymbol{x}')d\boldsymbol{x}'. \tag{C.34}$$

This is very important result in micro-mechanics and can be utilized in the treatment of inhomogeneities.

D julia code

Note that file paths have to be modified to match the file structure of the current user.

D.1 `generateTrainingData_main.jl`

```
using Distributions, Random, DelimitedFiles, Shell, LinearAlgebra
include("C:/Users/Johan/Documents/Julia/strainPathGenerator.jl")
include("C:/Users/Johan/Documents/Julia/writemat.jl")
include("C:/Users/Johan/Documents/Julia/stressPostprocess.jl")
include("C:/Users/Johan/Documents/Julia/orientationTensorGenerator.jl")

##----Parameters----#############################################################
# Number of files is n_end - n_start
n_start = 1 # Start number
n_end = 1000 # End number
# Ensure that counter is increasing!
if n_start > n_end
    n_start = 1
    n_end = 10
end


timeSteps = 2000;
# Number of time steps in strain paths must be divisible by 1000!
if timeSteps%1000 != 0
    timeSteps = 1000
end

t = LinRange(0,1,timeSteps+1) # Time variable

strainPath = "C:/Users/Johan/Documents/Julia/StrainData/"
strainNameFormat = "strainData_"
stressPath = "C:/Users/Johan/Documents/Julia/StrainData/"
stressNameFormat = "stressData_"
orientationPath = "C:/Users/Johan/Documents/Julia/StrainData/"
orientationNameFormat = "orientationData_"

matPath = "C:/Users/Johan/Documents/Julia/StrainData/"
matNameFormat = "analysis_"
outputNameFormat = "output_"

finalTime = "1.0e+00"
maxTimeInc = "1.0e-01"
minTimeInc = "1.0e-02"
integrationParameter = "5.0e-01"
```

```
numberAngleIncrements = "12"
outputPrecision = "5"

PlaneConditionInitialGuess = "off"
OTtraceTol = "1.0e-01"

youngFiber = "76e+09"
poissonFiber = "2.2e-01"
fiberVolumefraction = "1e-01" # This value if random_3D orientation.
fiberAspectratio = "2.4e+01"
fiberOrientation = "tensor" # "tensor" or "random_3D"

youngMatrix = "3.1e+09"
poissonMatrix = "3.5e-01"
yieldMatrix = "2.5e+07"
hardeningModulusMatrix = "2e+07"
hardeningExponentMatrix = "3.25e+02"
hardeningModulus2Matrix = "1.5e+08"
matrixVolumefraction = "9.0e-01" # This value if random_3D orientation.
                                 # Must be equal to 1-fiberVolumefraction!
################################################################################

# IMPORTANT! Match significant digits of time vector to outputPrecision!
t = round.(t,sigdigits=parse(Int,outputPrecision))

# Generate strain path data. Optionally also generates orientation tensors.
for i = n_start:n_end
    noise = rand()
    n_drift = rand([1 2 5 10 20 25 50 100 200])
    # Calculate n_step by dividing timeStep by n_drift to ensure
    # constant length of time series
    n_step = div(timeSteps,n_drift) # div(x,y) to ensure int.
    EPS = 0.01 + rand()*0.04 # Maximum admissible component in 0.01 to 0.05
    if rand() > 0.9
        uni = true
    else
        uni = false
    end
    strainPathGenerator(noise,n_step,n_drift,EPS,t,
    strainPath*strainNameFormat*string(i),uni)
    if fiberOrientation != "random_3D"
        if rand() > 0.9
            fullyAligned = true
        else
            fullyAligned = false
        end
        orientationTensorGenerator(fullyAligned, fiberVolumefraction,
        orientationPath*orientationNameFormat*string(i))
    end
end

# Write *.mat files for every strain path.
for i = n_start:n_end
    local inPath = strainPath*strainNameFormat*string(i)*".txt"
    local outPath = matPath*matNameFormat*string(i)*".mat"
    outputName = outputNameFormat*string(i)
    if fiberOrientation == "random_3D"
        writemat(inPath,outPath,outputName,finalTime,maxTimeInc,minTimeInc,
        integrationParameter,numberAngleIncrements,outputPrecision,
        PlaneConditionInitialGuess,OTtraceTol,youngFiber,poissonFiber,
        fiberVolumefraction,fiberAspectratio,fiberOrientation,
        youngMatrix,poissonMatrix,yieldMatrix,hardeningModulusMatrix,
        hardeningExponentMatrix,hardeningModulus2Matrix,matrixVolumefraction)
    else
        local inPath2 = orientationPath*orientationNameFormat*string(i)*".txt"
        writemat(inPath,outPath,outputName,finalTime,maxTimeInc,minTimeInc,
        integrationParameter,numberAngleIncrements,outputPrecision,
        PlaneConditionInitialGuess,OTtraceTol,youngFiber,poissonFiber,
        fiberVolumefraction,fiberAspectratio,fiberOrientation,
```

```
        youngMatrix,poissonMatrix,yieldMatrix,hardeningModulusMatrix,
        hardeningExponentMatrix,hardeningModulus2Matrix,matrixVolumefraction,
        inPath2)
    end
end

# Run DIGIMAT simulation for every *.mat file.
for i = n_start:n_end
    digimatPath = "C:/MSC.Software/Digimat/2020.0/DigimatMF/exec/"
    digimatInput = "input="*matPath*matNameFormat*string(i)*".mat"

    Shell.run(digimatPath*"DigimatMFPlugin.bat"*" "*digimatInput)

    # Delete unnecessary files to save space!
    rm(strainPath*"analysis_"*string(i)*".log")
    rm(strainPath*"analysis_"*string(i)*".eng")
end

# Postprocess DIGIMAT output to match the generated strain data.
for i = n_start:n_end
    local inPath = matPath*matNameFormat*string(i)*".mac"
    local outPath = stressPath*stressNameFormat*string(i)*".txt"
    stressPostprocess(inPath,outPath,t)

    # Remove *.mac and *.mat files to save space!
    rm(matPath*matNameFormat*string(i)*".mac")
    rm(matPath*matNameFormat*string(i)*".mat")
end
```

D.2 strainPathGenerator.jl

```
#
# Generates a path in 6D strain space and prints to a tab delimited file.
# Picks n_drift directions and walks n_step steps in those directions with
# added noise. The noise vector is multiplied by the variable noise.
# The directions and noise is sampled from a normal distribution with mean
# 0 and standard deviation 1. The output data file is formatted column wise
# given in the order "time" "e11" "e22" "e33" "2*e12" "2*e23" "2*e13".
#
# noise:    Noise multiplicative factor
# n_step:   Number of random steps per directions
# n_drift:  Number of drift directions
# EPS:      Maximum admissible strain component
# t:        Time variable
# path:     File path including file name and number (will append .txt).
# uni:      If true; sets all but one random strain component to 0.

function strainPathGenerator(noise,n_step,n_drift,EPS,t,path,uni)
    X = repeat(rand(Normal(0,1),(n_drift,6)),inner=[n_step, 1])
    X = X./sqrt.(sum((X).^2,dims=2))

    Y = rand(Normal(0,1),(n_drift*n_step,6))
    Y = noise.*Y./sqrt.(sum((Y).^2,dims=2))

    Eps = vcat(zeros(1,6),cumsum(X+Y,dims=1))

    # For unidirectional strain, delete all components except for one.
    if uni
        zeroIndex = [1;2;3;4;5;6]
        deleteat!(zeroIndex,rand(1:6))
        Eps[:,zeroIndex] .= 0
```

```
    end

    s = maximum(abs.(Eps))
    eps = Eps.*EPS./s
    # Input format is shearstrain*2, therefore those components are scaled.
    eps[:,4:end] = eps[:,4:end]*2

    header = ["time" "e11" "e22" "e33" "2*e12" "2*e23" "2*e13"]
    currentPath =path*".txt"
    open(currentPath, "w") do io
        writedlm(io, vcat(header,hcat(t,eps)))
    end

    return nothing
end
```

D.3 writemat.jl

```
#
# Writes *.mat for analysis of linear exponential isotropic hardening matrix
# with elastic fibers. The loading conditions are loaded from a strain path
# file. The loading conditions are general 3D loading, prescribed 6D strain.
#
# inPath:                       The path of the input strain file.
# outPath:                      The path of the output *.mat file.
# finalTime:                    Final time of the analysis.
# maxTimeInc:                   Maximum integration time step.
# minTimeInc:                   Minimum integration time step.
# integrationParameter:         Integration parameter.
# numberAngleIncrements:        Number of angle increments.
# outputPrecision:              Output number of significant figures.
# PlaneConditionInitialGuess:   Is the initial guess plane conditions?
# OTtraceTol:                   Orientation tensor trace tolerance.
# youngFiber:                   Young's modulus of fiber.
# poissonFiber:                 Poisson's ratio of fiber.
# fiberVolumefraction:          Volume fraction of fiber.
# fiberAspectratio:             Fiber aspect ratio.
# fiberOrientation:             Fiber orientation.
# youngMatrix:                  Young's modulus of matrix.
# poissonMatrix:                Poisson's ratio of matrix.
# yieldMatrix:                  Yield stress of matrix.
# hardeningModulusMatrix:       Hardening modulus R_inf of matrix.
# hardeningExponentMatrix:      Hardening exponent m of matrix.
# hardeningModulus2Matrix:      Linear hardening modulus of matrix.
# matrixVolumefraction:         Volume fraction of matrix.
# orientationTensor:            2nd order orientation tensor on Voight form.[OPT]
#
function writemat(inPath,outPath,outputName,finalTime,maxTimeInc,minTimeInc,
    integrationParameter,numberAngleIncrements,outputPrecision,
    PlaneConditionInitialGuess,OTtraceTol,youngFiber,poissonFiber,
    fiberVolumefraction,fiberAspectratio,fiberOrientation,
    youngMatrix,poissonMatrix,yieldMatrix,hardeningModulusMatrix,
    hardeningExponentMatrix,hardeningModulus2Matrix,matrixVolumefraction,
    inPath2 = "")

analysisName = outPath[findlast("/",outPath)[1]+1:end-4]

separator = "########################################"

# Begin by reading strain data file.
strainData = open(inPath, "r") do io
```

```
    readdlm(io, '\t')
end
# Read orientation tensor if applicable.
if fiberOrientation == "random_3D"
    a = ["" "" "" "" "" ""]
else
    orientationTensor = open(inPath2, "r") do io
        readdlm(io, '\t')
    end
    a = string.(orientationTensor[2,:])
end

# Define entries for sections of *.mat
fiberMATERIAL = [""; separator; "MATERIAL"; "name = Fiber"; "type = elastic";
"elastic_model = isotropic"; "Young = "*youngFiber; "Poisson = "*poissonFiber;
""]

matrixMATERIAL = [separator; "MATERIAL"; "name = Matrix";
"type = J2_plasticity"; "consistent_tangent = on"; "elastic_model = isotropic";
"Young = "*youngMatrix; "Poisson = "*poissonMatrix;
"yield_stress = "*yieldMatrix; "hardening_model = exponential_linear";
"hardening_modulus = "*hardeningModulusMatrix;
"hardening_exponent = "*hardeningExponentMatrix;
"hardening_modulus2 = "*hardeningModulus2Matrix; "isotropic_method = spectral";
""]

if fiberOrientation == "random_3D"
    fiberPHASE = [separator; "PHASE";"name = InclusionPhase";
    "type = inclusion"; "volume_fraction = "*fiberVolumefraction;
    "behavior = deformable_solid"; "material = Fiber";
    "aspect_ratio = "*fiberAspectratio;
    "orientation = "*fiberOrientation; "coated = no"; ""]
else
    fiberPHASE = [separator; "PHASE";"name = InclusionPhase";
    "type = inclusion"; "volume_fraction = "*a[7];
    "behavior = deformable_solid"; "material = Fiber";
    "aspect_ratio = "*fiberAspectratio;
    "orientation = "*fiberOrientation;
    "orientation_11 = "*a[1]; "orientation_22 = "*a[2];
    "orientation_33 = "*a[3]; "orientation_12 = "*a[4];
    "orientation_13 = "*a[5]; "orientation_23 = "*a[6];
    "closure = orthotropic"; "coated = no"; ""]
end

if fiberOrientation == "random_3D"
    matrixPHASE = [separator; "PHASE";"name = MatrixPhase"; "type = matrix";
    "volume_fraction = "*matrixVolumefraction; "material = Matrix"; ""]
else
    matrixPHASE = [separator; "PHASE";"name = MatrixPhase"; "type = matrix";
    "volume_fraction = "*string(1-parse(Float64,a[7])); "material = Matrix"; ""]
end

MICROSTRUCTURE = [separator; "MICROSTRUCTURE"; "name = TheMicrostructure";
"phase = MatrixPhase"; "phase = InclusionPhase"; ""]

LOADING = [separator; "LOADING"; "name = Mechanical"; "type = strain";
"load = General_3D"; "initial_strain_11 = 0.0e+00"; "strain_11 = 1.0e+00";
"initial_strain_22 = 0.0e+00"; "strain_22 = 1.0e+00";
"initial_strain_33 = 0.0e+00"; "strain_33 = 1.0e+00";
"initial_strain_12 = 0.0e+00"; "strain_12 = 1.0e+00";
"initial_strain_23 = 0.0e+00"; "strain_23 = 1.0e+00";
"initial_strain_13 = 0.0e+00"; "strain_13 = 1.0e+00";
"history = user_defined"; "history_component_11 = e11";
"history_component_11_value = relative";
"history_component_22 = e22"; "history_component_22_value = relative";
"history_component_33 = e33"; "history_component_33_value = relative";
"history_component_12 = e12"; "history_component_12_value = relative";
"history_component_23 = e23"; "history_component_23_value = relative";
"history_component_13 = e13"; "history_component_13_value = relative";
```

```
"quasi_static = on"; ""]

RVE = [separator; "RVE"; "type = classical";
"microstructure = TheMicrostructure"; ""]

ANALYSIS = [separator; "ANALYSIS"; "name = "*analysisName; "type = mechanical";
"loading_name = Mechanical";
"final_time = "*finalTime; "max_time_inc = "*maxTimeInc;
"min_time_inc = "*minTimeInc; "finite_strain = off";
"output_name = "*outputName; "load = DIGIMAT"; "homogenization = on";
"homogenization_model = Mori_Tanaka"; "second_order = on";
"integration_parameter = "*integrationParameter;
"number_angle_increments = "*numberAngleIncrements;
"output_precision = "*outputPrecision; "stiffness = off";
"plane_condition_initial_guess = "*PlaneConditionInitialGuess;
"OT_trace_tol = "*OTtraceTol; "hybrid_methodology = off";
"hybrid_failure_criteria = off"; "FPGF_refinement = on"; ""; ""; ""]

OUTPUT = [separator; "OUTPUT"; "name = "*outputName;
"RVE_data = Custom,time,S.11,S.22,S.33,S.12,S.13,S.23";
"Phase_data = InclusionPhase,None"; "Phase_data = MatrixPhase,None";
"Engineering_data = Default"; "Log_data = Default"; "Dependent_data = Default";
"Fatigue_data = Default"; "Composite_data = None"; ""]

point = repeat(["point = "],outer=length(strainData[2:end,1]))

e11FUNCTION = [separator; "FUNCTION"; "name = e11"; "type = piecewise_linear";
point.*string.(strainData[2:end,1]).*","*string.(strainData[2:end,2]); ""]

e22FUNCTION = [separator; "FUNCTION"; "name = e22"; "type = piecewise_linear";
point.*string.(strainData[2:end,1]).*","*string.(strainData[2:end,3]); ""]

e33FUNCTION = [separator; "FUNCTION"; "name = e33"; "type = piecewise_linear";
point.*string.(strainData[2:end,1]).*","*string.(strainData[2:end,4]); ""]

e12FUNCTION = [separator; "FUNCTION"; "name = e12"; "type = piecewise_linear";
point.*string.(strainData[2:end,1]).*","*string.(strainData[2:end,5]); ""]

e23FUNCTION = [separator; "FUNCTION"; "name = e23"; "type = piecewise_linear";
point.*string.(strainData[2:end,1]).*","*string.(strainData[2:end,6]); ""]

e13FUNCTION = [separator; "FUNCTION"; "name = e13"; "type = piecewise_linear";
point.*string.(strainData[2:end,1]).*","*string.(strainData[2:end,7])]

open(outPath, "w") do io
    writedlm(io,fiberMATERIAL)
    writedlm(io,matrixMATERIAL)
    writedlm(io,matrixPHASE)
    writedlm(io,fiberPHASE)
    writedlm(io,MICROSTRUCTURE)
    writedlm(io,LOADING)
    writedlm(io,RVE)
    writedlm(io,ANALYSIS)
    writedlm(io,OUTPUT)
    writedlm(io,e11FUNCTION)
    writedlm(io,e22FUNCTION)
    writedlm(io,e33FUNCTION)
    writedlm(io,e12FUNCTION)
    writedlm(io,e23FUNCTION)
    writedlm(io,e13FUNCTION)
end
end
```

D.4 stressPostprocess.jl

```
#
# Postprocesses the *.mac file to match the timestamps of the input
# strainData_*.txt file.
#
# inPath:       File path for *.mac file.
# outPath:      File path for output file.
# t:            Vector of time stamps.
# precision:    Number of significant figures of indata.
#

function stressPostprocess(inPath,outPath,t)

    # Read stress data file
    preData = readdlm(inPath,skipstart=2)

    # Find indices matching timestamps available in strainData files.
    index = zeros(length(t),1)
    for i = 1:length(t)
        index[i] = findall(preData[:,1].==t[i])[1]
    end
    index = convert.(Int64,index)[:]

    # Extract matching stress data
    postData = preData[index,:]

    header = ["time" "s11" "s22" "s33" "s12" "s23" "s13"]
    open(outPath, "w") do io
        writedlm(io,header)
        writedlm(io,postData)
    end
end
```

D.5 orientationTensorGenerator.jl

```
#
# Generates the second order orientation tensor from a random uniform
# distribution. Will also randomly sample the base volume fraction + 0-50%.
#
# fullyAligned:     Special case to ensure representation of fully aligned.
# fibervf:          Base volume fraction of fiber in the composite material.
# path:             File path including file name and number (will append .txt).
#
function orientationTensorGenerator(fullyAligned,fibervf,path)
    # Generate a random diagonal matrix with trace 1 where components
    # uniformly randomly sampled. See Non-uniform random variate generation by
    # Devroye page 568 for details.
    if fullyAligned
        x = [0;0;0]
        index = rand([1 2 3])
        x[index] = 1
        L = diagm(x)
    else
        x = diff(sort([0;rand(2);1]))
        L = diagm(x)
    end
```

```
    # Random sampling of rotation parameters.
    theta = 2*pi*rand()
    phi = 2*pi*rand()
    z = rand()

    # Generate rotation matrix around z-axis.
    R = [cos(theta) sin(theta) 0;
    -sin(theta) cos(theta) 0;
    0 0 1]

    # Generate mirrored householder matrix.
    v = [cos(phi)*sqrt(z); sin(phi)*sqrt(z); sqrt(1-z)]
    P = 2*v*v' - Matrix(I,3,3)

    # Total rotation matrix which is sampled uniformly from SO(3)
    M = P*R
    # Uniformly random sampled orientation tensor.
    a = M*L*M'

    # Randomly sample a volume fraction between initial volume fraction + 0-50%
    vf = string(parse(Float64,fibervf) + parse(Float64,fibervf)*0.5*rand())
    # Write to file!
    header = ["a11" "a22" "a33" "a12" "a13" "a23" "vf"]
    currentPath =path*".txt"
    open(currentPath, "w") do io
        writedlm(io,vcat(header,hcat(a[1,1],a[2,2],a[3,3],a[1,2],a[1,3],a[2,3],
        vf)))
    end
    return nothing
end
```

E MATLAB code

Note that file paths have to be modified to match the file structure of the current user.

E.1 `trainGRU.m`

```
% LOAD DATA
load('GRU_inData_1.mat')

% DEFINE PARAMETERS
maxEpochs = 500; % Maximum number of training epochs.
miniBatchSize = 32; % Mini batch size.
alpha0 = 0.0005;
tau = 10;
gamma = 0.9;
VBfreq = floor(size(X_train,1)/miniBatchSize); % Verbose frequency

% CREATE NEURAL NETWORK
layers = [ ...
    sequenceInputLayer(13,'Normalization','zscore')
    gruLayer(500,'OutputMode','sequence')
    gruLayer(500,'OutputMode','sequence')
    gruLayer(500,'OutputMode','sequence')
    dropoutLayer(0.5)
    fullyConnectedLayer(6)
    regressionLayer];

options = trainingOptions('adam', ...
    'ExecutionEnvironment','gpu', ...
    'MaxEpochs',maxEpochs, ...
    'Shuffle','every-epoch', ...
    'ResetInputNormalization', false,...
    'MiniBatchSize',miniBatchSize, ...
    'GradientThreshold',1, ...
    'InitialLearnRate',alpha0, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',tau, ...
    'LearnRateDropFactor',gamma, ...
    'Verbose',true, ...
    'VerboseFrequency',VBfreq, ...
    'Plots','training-progress', ...
    'ValidationFrequency',VBfreq, ...
'ValidationData',{X_valid,Y_valid});

% TRAIN NETWORK!
net = trainNetwork(X_train,Y_train,layers,options);

% SAVE NETWORK
save('gru500Net.mat','net')
```

E.2 testNetwork.m

```
load('gru500Net.mat')
load('GRU_inDATA_1.mat')

% CALCULATE ERROR!
L = length(X_test);
MaRE = zeros(L,6);
MeRE = zeros(L,6);
T = length(X_test{1});

for i = 1:L
    pred = predict(net,X_test{i});
    error = pred-Y_test{i};
    MeRE(i,:) = sqrt(sum(error.^2,2)/T)/25;
    MaRE(i,:) = max(abs(error),[],2)/25;
end

AvMeRE = sum(MeRE,1)/L;
AvMaRE = sum(MaRE,1)/L;
```

E.3 testPerformance_main.m

```
% LOAD NET
load('gru500Net.mat')
% LOAD CURRENT TEST
load('sample_5_uni1_strain.mat')
load('sample_5_uni1_stress.mat')

% SET PARAMETERS
a_11 = 0;
a_22 = 0.919;
a_33 = 0.081;
a_12 = 0.015;
a_13 = 0.005;
a_23 = 0.273;

v = 0.109;

% PUT PLOT TITLES HERE
txt1 = 'Sample 5: Uni-axial stress state';
txt2 = 'Sample 5: Uni-axial stress state (804 steps)';

% CHOSE WHICH STRESS-STRAIN CURVE TO PLOT!
sig = 1;

% r>0 INTERPOLATE OR r<0 EXTRAPOLATE, r=0 USE DEFAULT
r = 0;

% RUN TEST
testRateDependency(net, DefaultJobNameanalysis1, DefaultJobNameanalysis2,...
    a_11, a_22, a_33, a_12, a_13, a_23, v, txt1);

[MeRE, MaRE] = modelValidator(net,...
    DefaultJobNameanalysis1, DefaultJobNameanalysis2,...
    r, a_11, a_22, a_33, a_12, a_13, a_23, v, sig, txt2);
```

E.4 modelValidator.m

```
function [MeRE,MaRE,L]=modelValidator(net,...
    strain,stress,r,a_11,a_22,a_33, a_12,a_13,a_23,v,n,txt)

     strain = strain';
     stress = stress'/10^6;
     % INTERPOLATE OR EXTRAPOLATE!?
     if r > 0
         L = length(strain);
         STRAIN = zeros(6,L*r);
         STRESS = zeros(6,L*r);
         t = linspace(0,1,L);
         s = linspace(0,1,L*r);
         for i = 1:6
             STRAIN(i,:) = interp1(t,strain(i,:),s);
             STRESS(i,:) = interp1(t,stress(i,:),s);
         end
         strain = STRAIN;
         stress = STRESS;
     elseif r < 0
         strain = strain(:,1:-r:end);
         stress = stress(:,1:-r:end);
     else
         % DO NOTHING!
     end

     L = length(strain);
     % INSERT ORIENTATION TENSOR AND VOLUME FRACTION!
     strain = [repmat(a_11,1,L);repmat(a_22,1,L);repmat(a_33,1,L);...
         repmat(a_12,1,L);repmat(a_13,1,L);repmat(a_23,1,L);...
         repmat(v,1,L); strain];

     % DO PREDICTION!
     pred = predict(net,strain);
     error = pred-stress;

     % COMPUTE ERRORS!
     MeRE = sqrt(sum(error.^2,2)/L)/25;
     MaRE = max(abs(error),[],2)/25;

     if nargin == 13
         % INFORMATION FOR PRETTY PLOTS!
         switch n
             case 1
                 eps = strcat('$\varepsilon_{11}\ $','[-]');
                 sig = strcat('$\sigma_{11}\ $','[MPa]');
             case 2
                 eps = strcat('$\varepsilon_{22}\ $','[-]');
                 sig = strcat('$\sigma_{22}\ $','[MPa]');
             case 3
                 eps = strcat('$\varepsilon_{33}\ $','[-]');
                 sig = strcat('$\sigma_{33}\ $','[MPa]');
             case 4
                 eps = strcat('$2\varepsilon_{12}\ $','[-]');
                 sig = strcat('$\sigma_{12}\ $','[MPa]');
             case 5
                 eps = strcat('$2\varepsilon_{23}\ $','[-]');
                 sig = strcat('$\sigma_{23}\ $','[MPa]');
             case 6
                 eps = strcat('$2\varepsilon_{13}\ $','[-])');
                 sig = strcat('$\sigma_{13}\ $','[MPa]');
         end

         % PLOT!
         figure(3);
         hold on;
         plot(strain(7+n,:),pred(n,:),'--r','LineWidth',2);
         plot(strain(7+n,:),stress(n,:),'-b','LineWidth',1);

         ax = gca;
         ax.GridLineStyle = '-';
         ax.GridColor = 'k';
         ax.GridAlpha = 1;
         grid on;

         set(gca,'TickLabelInterpreter', 'latex','fontsize',15);
         title(txt,'interpreter','latex','fontsize',15);
         xlabel(eps,'interpreter','latex','fontsize',15);
         ylabel(sig,'interpreter','latex','fontsize',15);
         legend('Network','DIGIMAT','Interpreter','latex',...
             'Location','southeast');

         figure(4);
         set(gcf,'Position', [100, 100, 1200, 600])
         sgtitle(txt,'interpreter','latex','fontsize',15);

         labels = ['$\sigma_{11}$';'$\sigma_{22}$';'$\sigma_{33}$';...
             '$\sigma_{12}$';'$\sigma_{23}$';'$\sigma_{13}$'];
```

```
        for i = 1:6
            subplot(2,3,i);
            hold on;
            plot(pred(i,:),'--r','LineWidth',2);
            plot(stress(i,:),'-b','LineWidth',1);
            ax = gca;
            ax.GridLineStyle = '-';
            ax.GridColor = 'k';
            ax.GridAlpha = 1;
            grid on;
            set(gca,'TickLabelInterpreter', 'latex','fontsize',15);
            xlabel('Step [-]','interpreter','latex','fontsize',15);
            ylabel(strcat(labels(i,:),' [MPa]'),...
                'interpreter','latex','fontsize',15);
            legend('Network','DIGIMAT',...
                'Interpreter','latex','Location','best');
        end
end
```

E.5 testRateDependency.m

```
function testRateDependency(net, strain, stress,...
    a_11, a_22, a_33, a_12, a_13, a_23, v, txt)
    r = [-32 -16 -8 -4 -2 0 2 4 8 16 32 64 128];

    % COMPUTE
    l = length(r);
    MeRE = zeros(6,l);
    MaRE = zeros(6,l);
    steps = zeros(1,l);
    for i = 1:l
        [Me,Ma,L]=modelValidator(net, strain, stress, r(i), a_11, a_22,...
        a_33, a_12, a_13, a_23, v);
        MeRE(:,i) = Me;
        MaRE(:,i) = Ma;
        steps(i) = L;
    end

    % ---PLOT ERROR---
    figure(1);
    set(gcf,'Position', [100, 100, 1200, 600])
    sgtitle(txt,'interpreter','latex','fontsize',15);
    for i = 1:4
        subplot(2,2,i);
        hold on;
        plot(log10(steps),MeRE(1,:),'xb','MarkerSize',10);
        plot(log10(steps),MeRE(2,:),'sr','MarkerSize',10);
        plot(log10(steps),MeRE(3,:),'^g','MarkerSize',10);
        ax = gca;
        ax.GridLineStyle = '-';
        ax.GridColor = 'k';
        ax.GridAlpha = 1;
        grid on;
        set(gca,'TickLabelInterpreter', 'latex','fontsize',15);
        set(gca,'YTick',0:0.2:3);
        xlabel('log$_{10}$(steps)','interpreter','latex','fontsize',15);
        if i<3
            ylabel('MeRE','interpreter','latex','fontsize',15);
        else
            ylabel('MaRE','interpreter','latex','fontsize',15);
        end
        if i == 1 | i == 3
            legend('$\sigma_{11}$','$\sigma_{22}$','$\sigma_{33}$',...
                'Interpreter','latex','Location','north');
        else
            legend('$\sigma_{12}$','$\sigma_{23}$','$\sigma_{13}$',...
                'Interpreter','latex','Location','north');
        end
    end
end
```

E.6 cycleTest.m

```
load('gru500Net.mat')

a_11 = 0;
a_22 = 0;
a_33 = 0;
a_12 = 0;
a_13 = 0;
a_23 = 0;

v = 0.12;

r = 0;

labels = ['$\sigma_{11}$';'$\sigma_{22}$';'$\sigma_{33}$';...
    '$\sigma_{12}$';'$\sigma_{23}$';'$\sigma_{13}$'];


MeRE = zeros(6,15);
MaRE = zeros(6,15);
k=1;

for i = 1:3
    switch i
        case 1
            a_11 = 1;
            a_22 = 0;
            a_33 = 0;
        case 2
            a_11 = 0.5;
            a_22 = 0.5;
            a_33 = 0;
        case 3
            a_11 = 0.33;
            a_22 = 0.33;
            a_33 = 0.33;
    end
    for j = 1:5
        load(strcat(num2str(i),'D_cycletest',num2str(j),'_strain.mat'))
        load(strcat(num2str(i),'D_cycletest',num2str(j),'_stress.mat'))

        [Me, Ma] = modelValidator(net,...
            DefaultJobNameanalysis1, DefaultJobNameanalysis2,...
            r, a_11, a_22, a_33, a_12, a_13, a_23, v);

        MeRE(:,k) = Me;
        MaRE(:,k) = Ma;

        k = k +1;
    end
end

figure(1)
set(gcf,'Position', [100, 100, 1200, 600]);
sgtitle('MeRE as function of load cycles','interpreter','latex','fontsize',15);

figure(2)
set(gcf,'Position', [100, 100, 1200, 600]);
sgtitle('MaRE as function of load cycles','interpreter','latex','fontsize',15);
for p = 1:6
    figure(1)
    subplot(2,3,p)
    hold on
    ax = gca;
    ax.GridLineStyle = '-';
    ax.GridColor = 'k';
    ax.GridAlpha = 1;
    grid on;
    title(labels(p,:),'interpreter','latex','fontsize',15);
    plot(MeRE(p,1:5),'xk')
    plot(MeRE(p,6:10),'or')
    plot(MeRE(p,11:15),'b^')
    set(gca,'XTick',0:1:5);
    set(gca,'TickLabelInterpreter', 'latex','fontsize',15);
    xlabel('Cycles','interpreter','latex','fontsize',15);
    ylabel('MeRE','interpreter','latex','fontsize',15);
    legend('1D','2D','3D','Interpreter','latex','Location','best');

    figure(2)
    subplot(2,3,p)
    hold on
    ax = gca;
    ax.GridLineStyle = '-';
    ax.GridColor = 'k';
    ax.GridAlpha = 1;
    grid on;
    title(labels(p,:),'interpreter','latex','fontsize',15);
    plot(MaRE(p,1:5),'xk')
```

```
88      plot(MaRE(p,6:10),'or')
89      plot(MaRE(p,11:15),'b^')
90      set(gca,'XTick',0:1:5);
91      set(gca,'TickLabelInterpreter', 'latex','fontsize',15);
92      xlabel('Cycles','interpreter','latex','fontsize',15);
93      ylabel('MaRE','interpreter','latex','fontsize',15);
94      legend('1D','2D','3D','Interpreter','latex','Location','best');
95  end
```

E.7 extrapolateTest.m

```
1   load('gru500Net.mat')
2
3   a_11 = 0.33;
4   a_22 = 0.33;
5   a_33 = 0.33;
6   a_12 = 0;
7   a_13 = 0;
8   a_23 = 0;
9
10  v = 0;
11
12  r = 0;
13
14
15  str1 = '';
16  str2 = '';
17
18  MeRE = zeros(6,27);
19  MaRE = zeros(6,27);
20  k=1;
21
22  for i = 1:3
23      switch i
24          case 1
25              str2 = 'e5_';
26          case 2
27              str2 = 'e7-5_';
28          case 3
29              str2 = 'e10_';
30      end
31      for j = 1:9
32          switch j
33              case 1
34                  v = 0.001;
35                  str1 = '0-1_';
36              case 2
37                  str1 = '2-5_';
38                  v = 0.025;
39              case 3
40                  v = 0.05;
41                  str1 = '5_';
42              case 4
43                  v = 0.075;
44                  str1 = '7-5_';
45              case 5
46                  v = 0.1;
47                  str1 = '10_';
48              case 6
49                  v = 0.125;
50                  str1 = '12-5_';
51              case 7
52                  v = 0.15;
53                  str1 = '15_';
54              case 8
55                  v = 0.175;
56                  str1 = '17-5_';
57              case 9
58                  str1 = '20_';
59                  v = 0.2;
60          end
61          load(strcat('extrapolatetest_v',str1,str2,'strain.mat'))
62          load(strcat('extrapolatetest_v',str1,str2,'stress.mat'))
63
64          [Me, Ma] = modelValidator(net,...
65              DefaultJobNameanalysis1, DefaultJobNameanalysis2,...
66              r, a_11, a_22, a_33, a_12, a_13, a_23, v);
67
68          MeRE(:,k) = Me;
69          MaRE(:,k) = Ma;
70
71          k = k +1;
72      end
73  end
74
75  figure(1)
```

```
set(gcf,'Position', [100, 100, 1200, 600]);
sgtitle('Effect of Parameter Extrapolation on Error','interpreter',...
    'latex','fontsize',15);

X = [0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05;
    0.075 0.075 0.075 0.075 0.075 0.075 0.075 0.075 0.075;
    0.10 0.10 0.10 0.10 0.10 0.10 0.10 0.10 0.10];
Y = [0.001 0.025 0.05 0.075 0.10 0.125 0.15 0.175 0.2;
    0.001 0.025 0.05 0.075 0.10 0.125 0.15 0.175 0.2;
    0.001 0.025 0.05 0.075 0.10 0.125 0.15 0.175 0.2];

Z1 = [MeRE(1,1:9); MeRE(1,10:18); MeRE(1,19:27)];
Z2 = [MaRE(1,1:9); MaRE(1,10:18); MaRE(1,19:27)];

subplot(1,2,1)
stem3(X,Y,Z1,'b','LineWidth',2)
ax = gca;
ax.GridLineStyle = '-';
ax.GridColor = 'k';
ax.GridAlpha = 1;
grid on;

set(gca,'XTick',0.05:0.025:0.1);
set(gca,'YTick',[0.001 0.025 0.05 0.075 0.1 0.125 0.15 0.175 0.2]);
set(gca,'TickLabelInterpreter', 'latex','fontsize',13);
xlabel('$\varepsilon_{\mathrm{M}}$ [-]','interpreter','latex',...
    'fontsize',15);
ylabel('$v_{\mathrm{F}}$ [-]','interpreter','latex','fontsize',15);
zlabel('MeRE','interpreter','latex','fontsize',15);

subplot(1,2,2)
stem3(X,Y,Z2,'b','LineWidth',2)
ax = gca;
ax.GridLineStyle = '-';
ax.GridColor = 'k';
ax.GridAlpha = 1;
grid on;

set(gca,'XTick',0.05:0.025:0.1);
set(gca,'YTick',[0.001 0.025 0.05 0.075 0.1 0.125 0.15 0.175 0.2]);
set(gca,'ZTick',0:0.2:2);
set(gca,'TickLabelInterpreter', 'latex','fontsize',13);
xlabel('$\varepsilon_{\mathrm{M}}$ [-]','interpreter','latex',...
    'fontsize',15);
ylabel('$v_{\mathrm{F}}$ [-]','interpreter','latex','fontsize',15);
zlabel('MaRE','interpreter','latex','fontsize',15);
```