Technische Universität Kaiserslautern

Department of Mathematics

Computing the Stiffness Tensor of Composite Materials by Interpolation Procedures

Lin, Chu He

Supervisor: Prof. Dr. Bernd Simeon

August 2022

Contents

Preface

> Mathematics is not about numbers, equations, computations, or algorithms: it is about understanding.
>
> *William Paul Thurston* **[COO09]**

After seven-year studies in mathematics, I start thinking about a problem, how could I show my deep understanding of mathematics? In the past few years, I must admit that I showed my understanding of mathematics by attending and passing exams with good marks at universities. However, in most of the writing exams at my bachelor's university, I had to only solve equations by certain known methods or algorithms, sometimes using a calculator was also allowed. For all oral exams at the Technical University of Kaiserslautern, I enjoyed showing off my understanding of mathematics by writing and explaining definitions, theorems, or proofs on the blackboard in front of professors.

At the final stage of my master's study, I am willing to present my deep understanding of mathematics by solving real-world problems with the assistance of mathematical modeling in the field of composite materials. But without a doubt, I do not merely list complicated mathematical concepts to show how elegant mathematics is. I genuinely hope that I could explain the application of mathematical knowledge in a logical and clear manner to pique the interest of readers in mathematics.

The first topic of this thesis is a general introduction to the composite materials and

an overview of previous research in Chapter 1. When I read the paper "Fiber orientation interpolation for the multiscale analysis of short fiber reinforced composite parts" by Jonathan et al. **[KSO+18]**, I was inspired by the wonderful implement of mathematical knowledge in mechanical properties of composites.

Chapter 2 is mainly about the classical mathematical concepts and numerical methods that are widely used in materials science. For me, I attempt to state every mathematical terminology using an understandable example or a vivid figure. In the last section of this chapter, we discuss two of the best-known mathematical methods in the one-dimensional case. Especially the finite element method plays an irreplaceable role in the area of mechanical engineering.

In Chapter 3, the key findings of my thesis are described. Fiber volume fraction and fiber orientation tensor both have significant influence on mechanical behaviour of composite materials. We are able to calculate numerically the stiffness tensor of a given fiber orientation tensor step by step using 3D finite elements and interpolation if we take both fiber volume fraction and fiber orientation tensor into consideration to complete the open topic in Jonathan's paper. To make each step of this procedure clear and straightforward, I will illustrate every step with numerous figures and a detailed algorithm.

Eventually, we have a quick look at computational examples and results in Chapter 4. Following the theoretical illustration from Chapter 3, we implement the final computation based on a real database from Fraunhofer ITWM in Python. Most importantly, we make a comparison between interpolation with prismatic and tetrahedral finite element from the aspect of accuracy and working efficiency, to tell our readers which method will be a better solution for their problems.

Acknowledgement

Words cannot express my gratitude to my supervisor Prof. Dr. Bernd Simeon for his invaluable feedback and prompt response all the time. The story began in the winter semester of 2021, I enjoyed deeply the lecture Numerical Methods for PDE I given by Prof. Dr. Bernd Simeon, who is always very patient to answer students' questions and shows how we could indeed implement mathematical methods in Matlab.

I would like to extend my sincere thanks to Dr. Hannes Grimm-Strele, who came up with such an interesting but challenging thesis topic for me at the beginning. In the past few months, he spent so much time answering my questions and checking my Python scripts. I could not have undertaken this journey without his help.

I would like to express my deepest appreciation to my parents, who raise me and support my study in Germany. Even though we have not seen each other for almost three years because of the Corona epidemic, their love for me will always be my biggest motivation wherever I am.

Lastly, I would like to mention my boyfriend Tim Huber for all our nice moments and his emotional support. From the day we met, he became the best listener in my life and will still be in the future.

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction to Composite Materials and Motivation

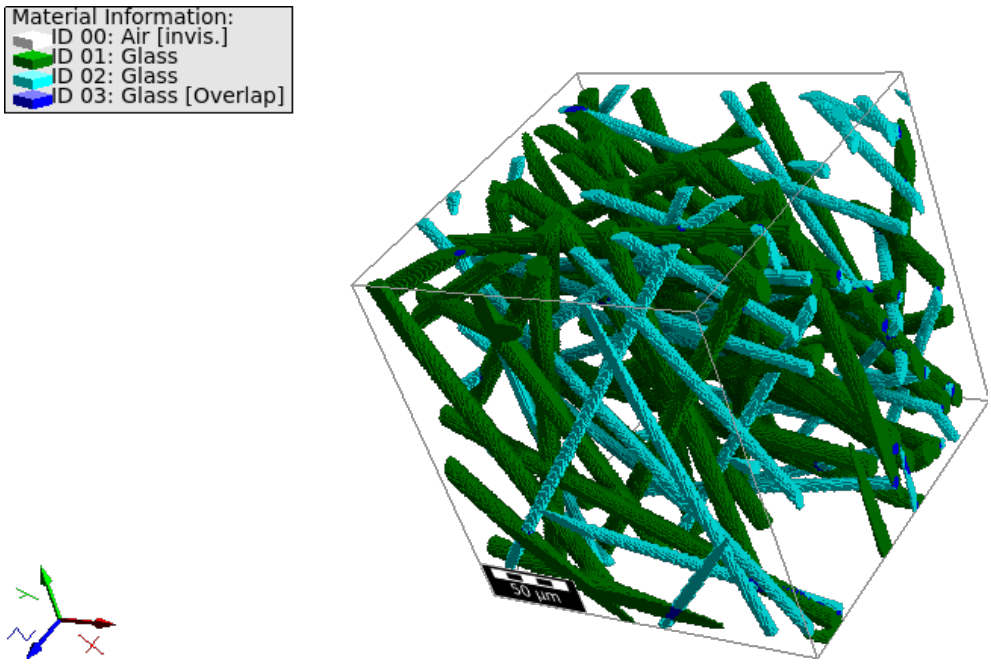

Figure 1: A 3D view of a composite material that made by Glass ID 01 (diameter: 10 µm) and Glass ID 02 (diameter: 6 µm) in software GeoDict **[BBG+21]**.

In this chapter we briefly explore the importance and general usage of composite materials in our daily life and then a quick review of the paper "Fiber orientation interpolation for the multiscale analysis of short fiber reinforced composite parts" by Jonathan et al. **[KSO+18]** will be presented as well. Eventually, a brief overview

of the research topic in this thesis will be given.

1.1 Composite Materials and Microstructure

1.1.1 A Short Introduction to Composite Materials

We mainly refer to "Structural Composite Materials" by F.C. Campbell **[Cam10]** for a precise and scientific introduction to composite materials.

Definition 1.1.1 (Composite material)**.** A *composite material* is produced as a combination of two or more constituent materials on a macroscopic scale that has better performances than those of the constituents used individually.

Nowadays, composite materials exist pervasively in our world. However, when we are talking about composite materials as non-experts, most of the time we are only referring to "man-made" composite materials from industrial manufacture. The truth is that composite materials have been used by mankind over thousands of years. Obviously, we ignore the existence of natural composites which played a significant role throughout the world in very early human civilization. For example, the first known usage of composite material was around 3400. B.C. by Mesopotamians. The ancient Mesopotamians glued pieces of natural wood together at different angles for better quality, which created the earliest prototype of plywood. In nature, wood is a composite consisting of wood fibers (cellulose) bound together by a matrix of lignin [**[Cam10]**, Preface].

Due to the outstanding advantages of composites, traditional materials such as steel and aluminum are replaced by composites year by year gradually. There are so many known benefits of composites, including corrosion resistance in an extreme environment, a higher strength-to-weight ratio and flexible design due to the wide range of constituent materials.

The applications of composites are widespread. In recent decades, the most common application is in the field of infrastructures such as construction and transportation. Moreover, wind power as the fastest-growing energy source, the blades of enormous

wind turbines are normally made of fiberglass to improve energy generation efficiency [**[Cam10]**, Chapter 1].

Figure 2: The blades of wind turbines are made by fiberglass which is one of the largest man-made composites. [**[Cam10], Fig. 1.32**]

1.1.2 Microstructure

In materials science, "microstructure" is used to describe the profile of material on the nm-cm length scale that has been revealed by an optical microscope with at least 25-time magnification. When we observe the microstructure of a material, we can see the internal arrangement of defects and phases.

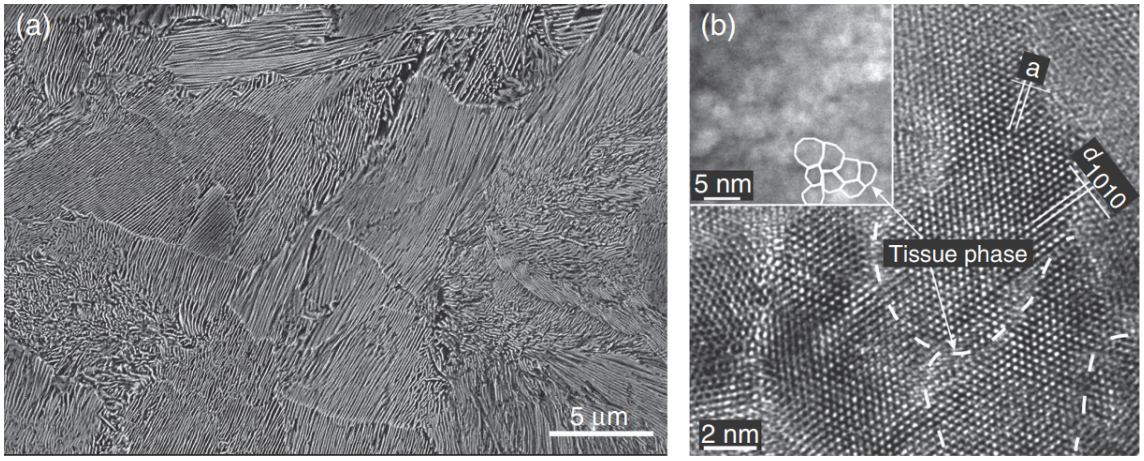

Figure 3: (a) "Conventional microstructure" of a pearlitic steel (scanning electron microscope image) and (b) "advanced nanostructure" of a superhard TiB_2 coating (high-resolution transmission electron microscope image). The grain size of the pearlitic steel is about 10 μm, whereas the grain size of the TiB_2 coating is below 5 nm. [**[CMS17], Figure 1.4**]

Most importantly, microstructures have a significant impact on the performance and physical properties of a material. For example, the microstructural parameters influence mechanical strength which is governed by the motion and quantity of *dislocations*. Furthermore, the microstructure of natural materials provides us with information about their complicated geological history.

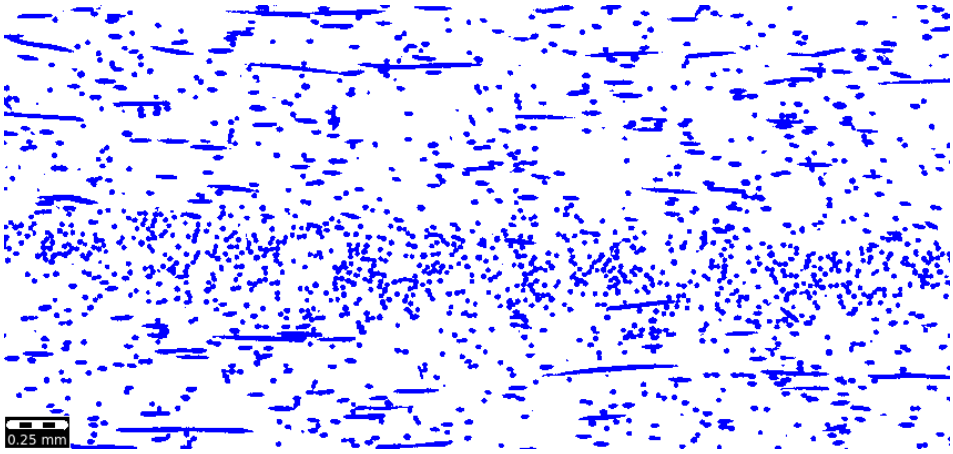

Figure 4: CT-scan of a material sample. [**[GSSA20], Figure 64**]

In Figure 4, we can see that the distribution of the composite material varies continuously in a 3D volume from point to point. Therefore, the corresponding material parameters such as fiber orientation tensor and fiber volume fraction will vary continuously as well from point to point.

1.2 Review of Previous Research and Motivation

We refer to **[KSO$^+$18]** in this section to present a review of previous research that we are working on its online phase in this thesis.

The local fiber orientation significantly affects the mechanical characteristics of the short fiber reinforced plastic parts. In **[KSO$^+$18]**, we provide a two-step identification strategy to enable the multiscale computations utilizing surrogate models. In the first step, using numerical methods described in **[KSO$^+$18]**, an effective model is derived for a number of sample orientations. In the second step, we create a fresh and effective strategy to carry out the interpolation of effective models in order to generalize the orientation state.

Consider a volume V that contains a number of fibers with N directions $p_1, ..., p_N \in S^2$, where S^2 represents the unit sphere, p_i is the orientation vector with three components and N can be quiet large. Referring to **[KSO$^+$18]**, we introduce the 3×3 symmetric positive semidefinite *Advani–Tucker (second order) fiber orientation tensor* A

$$A = \frac{1}{N}\sum_{i=1}^{N} p_i \otimes p_i \tag{1.1}$$

with $tr(A) = 1$.

Spectrally decomposing A leads to

$$A = R \cdot \Lambda \cdot R^T, \quad \Lambda = diag(\lambda_1, \lambda_2, \lambda_3) \tag{1.2}$$

where R is an orthogonal matrix and Λ is a diagonal matrix of sorted eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$. Particularly, the fiber orientation tensor A can be defined by two positive real eigenvalues λ_1, λ_2 that satisfy two inequalities up to an orthogonal transformation

$$1 - 2\lambda_1 \leq \lambda_2 \leq \lambda_1 \text{ and } \frac{1}{3} \leq \lambda_1 \leq 1, \tag{1.3}$$

which geometrically corresponds to a planar triangle, cf. Figure 5.

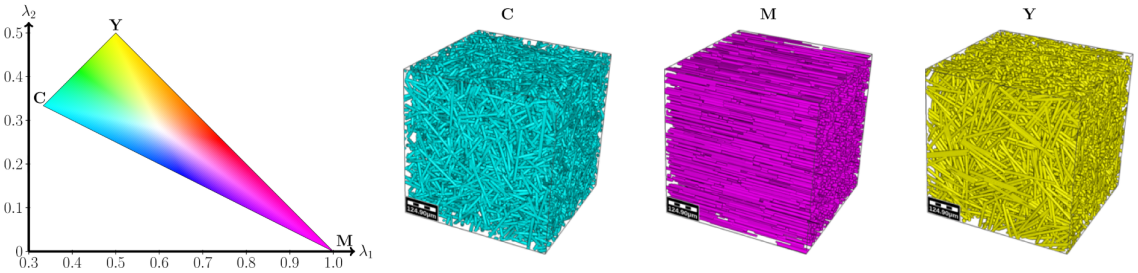

Figure 5: The fiber orientation reference triangle, showing the two largest principal eigenvalues λ_1, λ_2 of the fiber orientation tensor A. The extreme orientations are marked in cyan (isotropic), magenta (unidirectional) and yellow (planar), whereas the intermediate orientation states arise from combinations of the colors in CMY format, s.t. the center of mass of the triangle is white. **[[KSO+18], Fig. 2]**

In Figure 5, there are infinite points inside the fiber orientation reference triangle. From a physical perspective, the effective free energy W_A for each fiber orientation tensor A in equilibrium is continuous as a function of the orientation for fixed applied strain. As a result, if we discretize the triangle in Figure 5 using a sufficient fine triangulation mesh and assign a constitutive law (W_A, Φ_A) between an effective free energy W_A and an effective dissipation potential Φ_A to each node Λ of the mesh, a subsequent interpolation should accurately capture all effective materials law, see Figure 6 **[KSO+18]**.

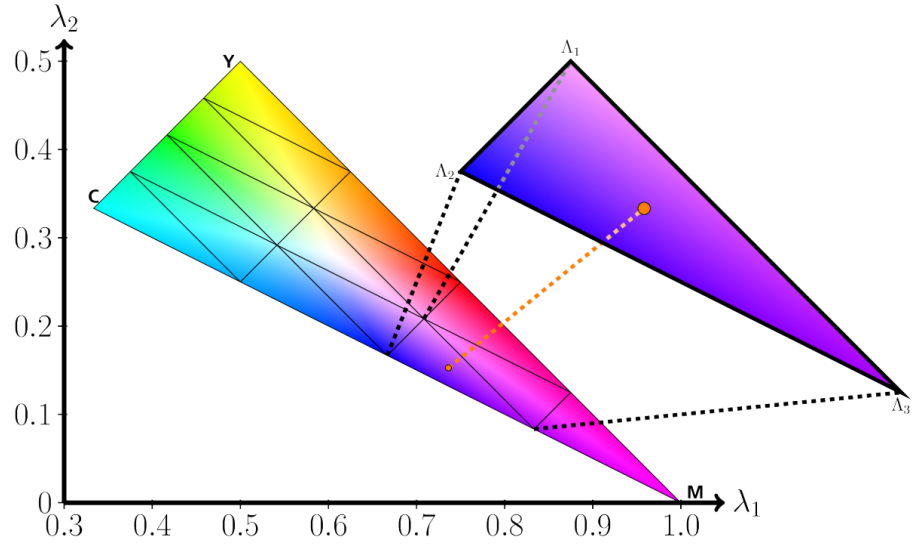

Figure 6: A triangulation of the fiber orientation triangle. For a given orientation state Λ (orange), the containing triangle is augmented, with nodes Λ_1, Λ_2 and Λ_3. The coloring encodes the orientation. **[[KSO+18], Fig. 3]**

Following Figure 6, the online phases from **[KSO+18]** that we will carry on in this master thesis are

1. Eigendecomposition of local fiber orientation tensor $A = R \cdot \Lambda \cdot R^T$ cf. Equation (1.2) and determine the two largest principal eigenvalues λ_1, λ_2;

2. Determine the local triangle T such that $\Lambda \in T = conv\{\Lambda_1, \Lambda_2, \Lambda_3\}$, then compute coefficients s_i such that $\Lambda = \sum_{i=1}^{3} s_i \Lambda_i$ and $\sum_{i=1}^{3} s_i = 1$ for all $s_i \geq 0$ in 2D triangular mesh cf. Figure 6;

3. Return eventually numerical approximation of local stiffness tensor C for a given fiber orientation tensor A from the first step using finite elements interpolation in the 3D principal prismatic space, cf. Figure 33.

Once we find out the coefficients s_i for $i = 1, 2, 3$, we associate to Λ the free energy

$$\widehat{W}_\Lambda(E, Q_{\Lambda_1}, Q_{\Lambda_2}, Q_{\Lambda_3}) = \sum_{i=1}^{3} s_i W_{\Lambda_i}(E, Q_{\Lambda_i})$$

and the corresponding dissipation potential

$$\widehat{\Phi}_\Lambda(\dot{Q}_{\Lambda_1}, \dot{Q}_{\Lambda_2}, \dot{Q}_{\Lambda_3}) = \sum_{i=1}^{3} s_i \Phi_{\Lambda_i}(\dot{Q}_{\Lambda_i}).$$

The corresponding macroscopic constitutive law

$$\begin{cases} \Sigma = \sum_{i=1}^{3} s_i \dfrac{\partial W_{\Lambda_i}}{\partial E}(E, Q_{\Lambda_i}), \\ \dfrac{\partial W_{\Lambda_i}}{\partial Q_{\Lambda_i}}(E, Q_{\Lambda_i}) + \dfrac{\partial \Phi_{\Lambda_i}}{\partial \dot{Q}_{\Lambda_i}}(\dot{Q}_{\Lambda_i}) = 0, \end{cases}$$

where Σ is the macroscopic stress, E is the macroscopic strain, Q_Λ is a vector of internal variables for Λ and $\dot{Q}_\Lambda$ is the rate of change of internal variables **[KSO+18]**.

Chapter 2

Linear Elasticity Theory and Mathematical Methods

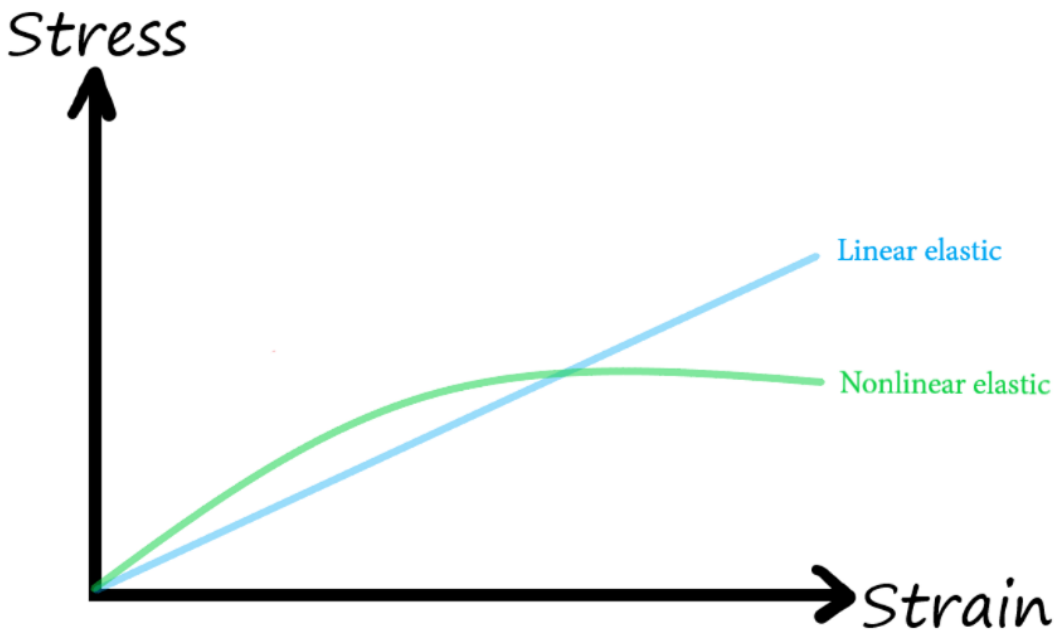

Figure 7: Stress-strain relationships for linear and non-linear elasticity

In this chapter, we introduce firstly the mathematical terms that are used widely in materials science. In 1676, British physicist Robert Hooke initially announced remarkable Hooke's law which provides us nowadays a possibility to discover further linear-elastic material and its deformation.

The historical story of materials science is different from most branches of scientific studies that have a founder, such as Archimedes in mathematics. In some ways, we could not even pinpoint the precise time as the starting point of materials science. Despite many uncertainties in materials science, we could still establish a strong connection between mathematics and materials science.

A comprehensive introduction to mathematical terms with an emphasis on classical linear elasticity theory and elastic motion can be found in [**[Bra13]**, Kapitel VI] and [**[Sim13]**, Chapter 3].

2.1 Classical Linear Elasticity Theory

2.1.1 Kinematics

Classical elasticity theory states the relationship between applied forces to an elastic body and resulting deformation in three-dimensional space, more accurately, the relationship between stresses and strains due to deformations. When we use mathematical items, we could explain the motion of an elastic body in following way from [**[Bra13]**, Kapitel VI] and [**[Sim13]**, Chapter 3]:

- **reference configuration** $\overline{\Omega}$ as the closure of a bounded open set $\Omega \subset \mathbb{R}^3$ that describes natural state of an elastic body without external forces;

- the **deformation** is denoted by the map
$$\varphi(x,t) : \overline{\Omega} \times [t_0, t_1] \to \mathbb{R}^3, \quad (x,t) \mapsto \varphi(x,t)$$
$\varphi(x,t)$ denotes the location of material point x at time t in deformed body;

- define **deformation gradient** $\nabla\varphi$, and it can be presented by matrix
$$\nabla\varphi = \begin{bmatrix} \frac{\partial\varphi_1}{\partial x_1} & \frac{\partial\varphi_1}{\partial x_2} & \frac{\partial\varphi_1}{\partial x_3} \\ \frac{\partial\varphi_2}{\partial x_1} & \frac{\partial\varphi_2}{\partial x_2} & \frac{\partial\varphi_2}{\partial x_3} \\ \frac{\partial\varphi_3}{\partial x_1} & \frac{\partial\varphi_3}{\partial x_2} & \frac{\partial\varphi_3}{\partial x_3} \end{bmatrix};$$

- assume deformation $\varphi(x,t)$ is sufficiently smooth, such that $det(\nabla\varphi) > 0$;

- we could also characterize the deformation by **displacement** $u(x,t)$

$$u(x,t) = \varphi(x,t) - x, \quad u(x,t) \in \mathbb{R}^3.$$

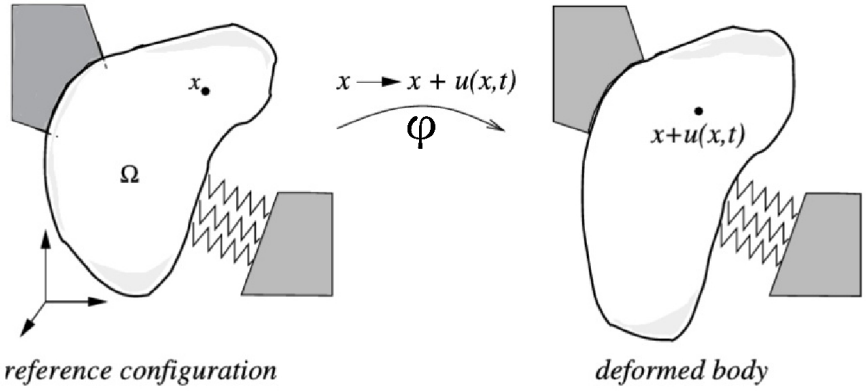

Figure 8: Sketch of reference configuration and deformed body with external applied forces. [**[Sim13], Fig. 3.1**]

2.1.2 Linear Elasticity Theory

Following Section 2.1.1, we now introduce the most significant definitions of linear elasticity theory.

Definition 2.1.1 (Green-Lagrangian strain tensor, [**[Sim13]**, Section 3.1.1])**.** The changes of local length element in deformed body can be described by second-order *Green-Lagrangian strain tensor*

$$E(x,t) = \frac{1}{2}(\nabla\varphi^T\nabla\varphi - I) = \frac{1}{2}(\nabla u + \nabla u^T + \nabla u^T\nabla u),$$

equivalently in matrix representation

$$E(x,t) = (E_{ij}(x,t))_{i,j=1,2,3} \in \mathbb{R}^{3\times 3}.$$

Remark 2.1.1 ([**[Sim13]**, Section 3.1.1] and [**[Bra13]**, Kapitel VI. §1])**.** Specifically, we could only take small displacements and small distortions into consideration in linear elasticity theory, consequently the strain tensor E could be simplified as

$$\epsilon = \epsilon(u) = \frac{1}{2}(\nabla u + \nabla u^T), \quad \epsilon_{ij} := \frac{1}{2}(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}).$$

In this case, strain tensor ϵ consists of the symmetric part of the displacement gradient ∇u.

Definition 2.1.2 (Stress tensor, [**[Sim13]**, Section 3.1.1])**.** Consider identity matrix $I \in \mathbb{R}^{3\times 3}$ and two Lamé constants $\lambda, \mu \in \mathbb{R}$, such that we define the symmetric *stress*

tensor

$$\Sigma(x,t) = \lambda(trace\ E)I + 2\mu E, \quad \Sigma(x,t) \in \mathbb{R}^{3\times 3}.$$

Remark 2.1.2 ([**[Sim13]**, Section 3.1.1])**.** For sake of simplification, we similarly reduce to linear elasticity, therefore by Hooke's law

$$\sigma = \sigma(u) = \lambda(trace\ \epsilon(u))I + 2\mu\epsilon(u).$$

We still introduce several essential materials parameters by following:

- **Lamé first constant** λ describes the stresses due to change in density;
- **Lamé second constant** μ is also called *shear modulus* in engineering to measure elastic shear stiffness of a material;
- **Poisson ratio** $\nu = \dfrac{\lambda}{2(\lambda+\mu)}$, measures how the stresses influence displacements in the orthogonal direction;
- **Young's modulus** E= $\dfrac{\mu(3\lambda+2\mu)}{\lambda+\mu}$, describes the undergoing tensile or compressive stiffness of a solid material in only one direction.

Generally, due to physical consideration it follows that E, $\lambda, \mu > 0$ and $0 < \nu < \dfrac{1}{2}$. With assistance of those quantities, we could introduce *Voigt notation* for Hooke's law and then lead to a vector and matrix representation. Let

$$\underline{\epsilon} := (\epsilon_{11}, \epsilon_{22}, \epsilon_{33}, 2\epsilon_{12}, 2\epsilon_{23}, 2\epsilon_{13})^T, \quad \underline{\sigma} := (\sigma_{11}, \sigma_{22}, \sigma_{33}, \sigma_{12}, \sigma_{23}, \sigma_{13})^T,$$

then replace Lamé constants by *Poisson ratio* ν and *Young's modulus* E. The Hooke's law in Remark 2.1.2 now reads $\underline{\sigma} = C\underline{\epsilon}$ with matrix $C \in \mathbb{R}^{6\times 6}$

$$C = \frac{\mathrm{E}}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & & & \\ \nu & 1-\nu & \nu & & & \\ \nu & \nu & 1-\nu & & & \\ & & & (1-2\nu)/2 & & \\ & & & & (1-2\nu)/2 & \\ & & & & & (1-2\nu)/2 \end{bmatrix}.$$

Introduction to Voigt Notation

Understanding the transformation between tensor notation and Voigt notation is very essential for programming in Python to compute the numerical approximation of stiffness tensor. Referring to [**[Giu16]**, Chapter 2], we discuss how to transfer reversibly between tensor notation and Voigt notation from a theoretical perspective.

The stress-strain relation in tensor notations can be written as, i.e.,

$$\sigma_{ij} = c_{ijkl}\epsilon_{kl}, \quad i,j = 1,2,3,$$

here we have *stress tensor* σ_{ij} and *strain tensor* ϵ_{kl} in an array form

$$[\sigma_{ij}] = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{12} & \sigma_{22} & \sigma_{23} \\ \sigma_{13} & \sigma_{23} & \sigma_{33} \end{bmatrix}, [\epsilon_{ij}] = \begin{bmatrix} \epsilon_{11} & \epsilon_{12} & \epsilon_{13} \\ \epsilon_{12} & \epsilon_{22} & \epsilon_{23} \\ \epsilon_{13} & \epsilon_{23} & \epsilon_{33} \end{bmatrix}.$$

The term c_{ijkl} is called the *stiffness tensor*, i.e.,

$$c_{ijkl}, \quad i,j,k,l = 1,2,3.$$

Like stress and strain tensor, the stiffness tensor is also symmetric, i.e.,

$$c_{ijkl} = c_{jikl} = c_{ijlk} = c_{klij} = c_{jilk}, \quad i,j,k,l = 1,2,3.$$

The corresponding strain-stress relation is given by

$$\epsilon_{ij} = s_{ijkl}\sigma_{kl}, \quad i,j = 1,2,3,$$

where the term s_{ijkl} is called the *compliance tensor* that enjoys the same symmetry properties as the stiffness tensor, i.e.,

$$s_{ijkl} = s_{jikl} = s_{ijlk} = s_{klij} = s_{jilk}, \quad i,j,k,l = 1,2,3.$$

For stiffness tensor with four indices, i.e., it is a fourth order tensor and it cannot be represented in a plane array. To overcome this difficulty, Voigt notation has been created to allow the stiffness tensor to be written as a matrix. Voigt notation consists of a replacing ij or kl by p or q, where $i,j,k,l = 1,2,3$ and $p,q = 1,2,3,4,5,6$ according to Table 1.

Table 1: Conversion from tensor to matrix indices for the Voigt notation [**[Giu16]**, TABLE 1]

ij or kl	p or q
11	1
22	2
33	3
23 or 32	4
31 or 13	5
12 or 21	6

Taking symmetry properties of the stiffness tensor into consideration and using the indexing rule of Table 1, we denote the elements of the 6×6 *stiffness matrix* as follows:

$$C = \begin{bmatrix} c_{1111} \to C_{11} & c_{1122} \to C_{12} & c_{1133} \to C_{13} & c_{1123} \to C_{14} & c_{1131} \to C_{15} & c_{1112} \to C_{16} \\ & c_{2222} \to C_{22} & c_{2233} \to C_{23} & c_{2223} \to C_{24} & c_{2231} \to C_{25} & c_{2212} \to C_{26} \\ & & c_{3333} \to C_{33} & c_{3323} \to C_{34} & c_{3331} \to C_{35} & c_{3312} \to C_{36} \\ & & & c_{2323} \to C_{44} & c_{2331} \to C_{45} & c_{2312} \to C_{46} \\ & sym. & & & c_{3131} \to C_{55} & c_{3112} \to C_{56} \\ & & & & & c_{1212} \to C_{66} \end{bmatrix}.$$

2.2 Constitutive Equation and Hyperelasticity

2.2.1 Cauchy's Stress Theorem

Cauchy's stress theorem states that there exists a **Cauchy stress tensor $\boldsymbol{T}(\boldsymbol{x})$** (or $\boldsymbol{\sigma}$) which maps the **unit normal vector $\boldsymbol{n}$** through a particle P to a surface to the **traction vector or stress vector $\boldsymbol{t}$** acting on that surface element dA [**[Irg19]**, Section 2.2.4]. For the most general statement, we restrict ourselves to three-dimensional Cartesian coordinate system with base vectors $\{e_i\}_{i=1,2,3}$ in Figure 9.

Then by *Cauchy's stress theorem*

$$\boldsymbol{t} = \boldsymbol{T}(\boldsymbol{x})\boldsymbol{n}.$$

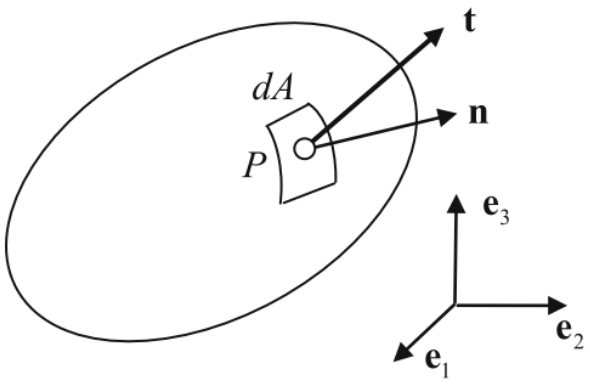

Figure 9: Illustration in 3D Cartesian coordinate system. [**[Irg19], Fig. 2.10**]

2.2.2 Constitutive Equation

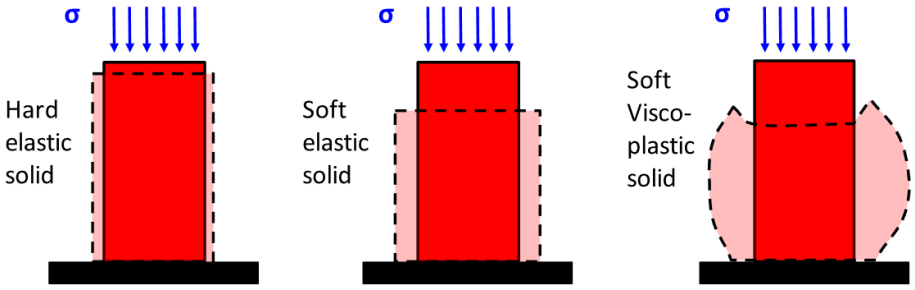

Figure 10: Example of three types deformations imparted by an applied stress. [**[Esp21], Figure 3.9**]

In physics and engineering, the constitutive equation states how an elastic body deforms under stresses from applied external forces. In [**[Bra13]**, Kapitel VI], we find a very detailed description of the constitutive equation by mathematical items.

For simplicity, we ignore the effect of temperature on materials and restrict ourselves to *homogeneous* materials. For a sake of convenience, we firstly introduce following notations:

- $\mathbb{M}^3_+$ the set of 3×3 matrices with positive determinants;
- $\mathbb{S}^3_+$ the set of symmetric 3×3 matrices with positive determinants;
- $\mathbb{O}^3_+$ the set of orthogonal 3×3 matrices with positive determinants;
- $\varphi(x_R)$, subscript R refers to the position in reference configuration, in particular $x = \varphi(x_R)$.

Definition 2.2.1 (elastic, response function, constitutive equation, [**[Bra13]**, 1.4 Definition]). A material is called *elastic* if the Cauchy stress tensor $T(x)$ is the function of deformation gradient $\nabla\varphi(x_R)$ for every deformed state. Set *response function* $\hat{T}$ for the Cauchy stress

$$\hat{T} : \mathbb{M}^3_+ \to \mathbb{S}^3_+$$

and corresponding *constitutive equation* is

$$T(x) = \hat{T}(\nabla\varphi(x_R)).$$

For a better understanding from physical perspective, elastic materials can recover to original state once their external load disappears because there is no internal energy dissipation during the deforming process. Therefore, we called the deforming process is *reversible* [**[Cha13]**. Section 8.1].

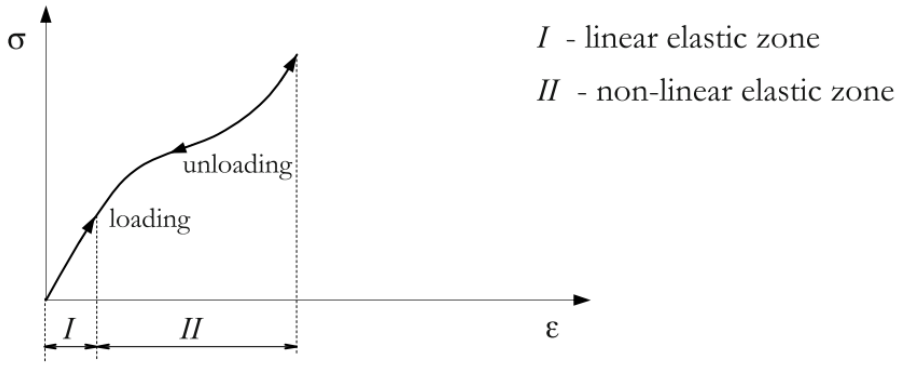

Figure 11: The stress-strain curve for elastic materials. [**[Cha13], Fig. 8.1**]

Definition 2.2.2 (isotropic, [**[Bra13]**, 1.7 Definition]). A material is called *isotropic* if

$$\hat{T}(F) = \hat{T}(FQ), \quad \text{for all } Q \in \mathbb{O}^3_+.$$

Formulas with variable F will generally be applied with $F := \nabla\varphi(x)$.

In simple terms, isotropic materials remain identical values of a property when tested in different directions. In contrast, *anisotropic materials* allow changes of properties in different directions. Isotropic materials include glass and metals, which we encounter every day. Wood and composites are anisotropic.

2.2.3 Hyperelasticity

For many materials such as robber and biological materials, the classical linear elastic models cannot precisely describe their mechanical behaviors. Such materials do not have any internal energy dissipation, even if they have extremely huge deformations. Those materials that have no memory of motion history and only depend on the current state are called *hyperelastic* or *purely hyperelastic* [**[Cha13]**, Section 8.1]. Being different from linear elastic materials, the stress-strain relation of a *hyperelastic* material is derived from a strain energy density function rather than a simple constant factor.

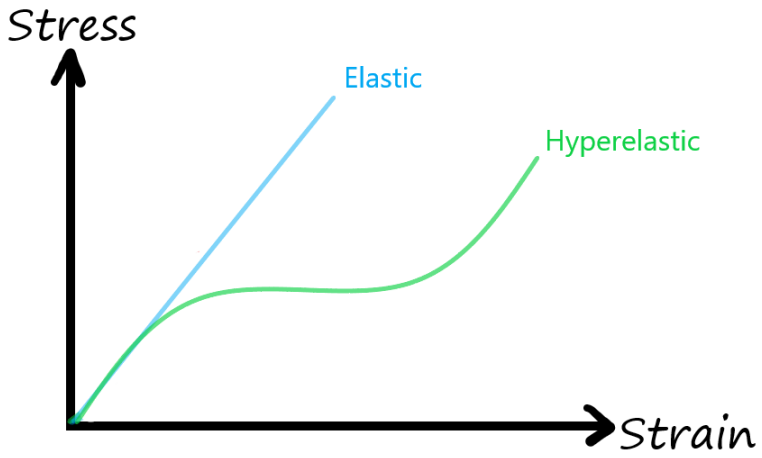

Figure 12: The stress-strain curve of elastic and hyperelastic material.

Definition 2.2.3 (hyperelastic, [**[Bra13]**, 2.1 Definition])**.** An elastic material is called *hyperelastic* if there exists an energy functional $\hat{W} : \Omega \times \mathbb{M}^3_+ \to \mathbb{R}$ such that

$$\hat{T}(x, F) = \frac{\partial \hat{W}}{\partial F}(x, F) \qquad \text{for } x \in \Omega, F \in \mathbb{M}^3_+.$$

In mathematics, a material with Poisson ratio $\nu = 0.5$ is a perfectly *incompressible* material, such as rubber. When a body undergoes deformation, the volume of a *incompressible* material remains the same.

Definition 2.2.4 (incompressible, [**[ADH$^+$14]**, Section 2.2])**.** From energy functional $\hat{W}$, we assume that nominal stress tensor S and symmetric Cauchy stress tensor σ are given respectively

$$S = \frac{\partial \hat{W}}{\partial F} - pF^{-1}, \quad \sigma = F\frac{\partial \hat{W}}{\partial F} - pI$$

where p is the Lagrange multiplier. Consequently, an *incompressible* material satisfies

$$\sigma = FS.$$

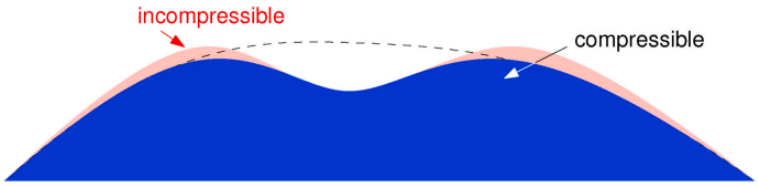

Figure 13: The difference between an incompressible material and a compressible material during indentation is illustrated. If the material is incompressible, material is pushed aside to fulfill the volume preserving constraint. [**[SCT+20], Fig. 1**]

2.3 State of the Art

2.3.1 Finite Element Method

In mathematics, the finite element method (FEM) is an efficient numerical approach for solving partial differential equations in integral form for physical analysis. When we apply FEM in materials science, FEM is a powerful tool to deal with complicated geometries that could be used to discretize a domain with irregular shapes and more natural imposition of boundary conditions [**[RBD03]**, Section 3.1].

Using FEM, we decompose the spatial domain into a set of **elements** with arbitrary size and shape to generate a **mesh** inside the domain but without any overlaps. For each element inside the domain, a certain number of **nodes** must be defined to construct the numerical approximation of original equations over the entire domain by *interpolation* [**[RBD03]**, Section 3.2.1].

In this thesis, we do not use FEM to solve a differential equation numerically. However, the *finite element* plays an important role in the interpolation of analytic function and stiffness tensor.

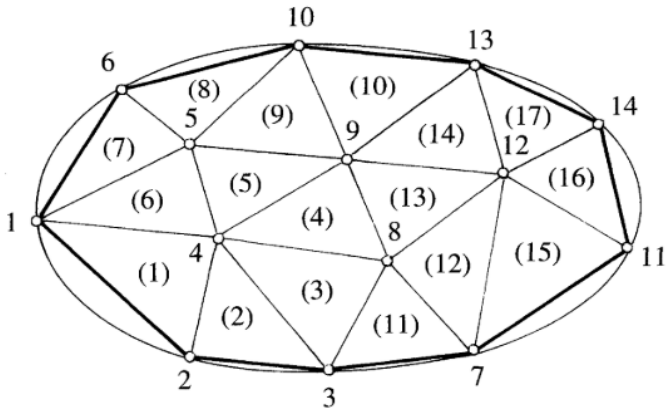

Figure 14: Discretization of an elliptical domain into 17 triangular elements and 14 nodes. The indices of nodes are "n" or "i", "j", "k", whereas the corresponding element will be noted with "e". **[[RBD03], Fig. 3.1]**

2.3.2 Finite Element Interpolation

Interpolation is a frequently used numerical estimation in the field of mathematical computation that can determine the value of new data points by using the discrete set of existing data points. Together with FEM, continuous models can be estimated by using known information from a finite number of discrete data points after discretization.

For a theoretical foundation of interpolation, we refer to **[[EG04]**, Section 1.1.1]. As an introductory section, we only discuss formula statements and examples in the one-dimensional case.

Introduction to Interpolation

Suppose interval $\Omega = [a, b]$, a *mesh* of Ω is an indexed set of intervals with non-zero measure $\{I_i = [x_{1,i}, x_{2,i}]\}$ for $0 \leq i \leq N$, i.e.,

$$\overline{\Omega} = \bigcup_{i=0}^{N} I_i \qquad \text{and} \qquad \mathring{I}_i \cap \mathring{I}_j = \emptyset \qquad \text{for } i \neq j.$$

For simplification, we construct a mesh by using $(N + 2)$ nodes from $\overline{\Omega}$ such that

$$a = x_0 < x_1 < ... < x_N < x_{N+1} = b$$

and let $x_{1,i} = x_i$ and $x_{2,i} = x_{i+1}$ for $0 \leq i \leq N$. Nodes from the collection $\{x_0, x_1, ..., x_{N+1}\}$ are called *vertices* of the mesh. The variable *step size* of the mesh for $0 \leq i \leq N$ is

$$h_i = x_{i+1} - x_i$$

and define

$$h = \max_{0 \le i \le N} h_i.$$

The intervals I_i are also called *elements* and the mesh is denoted by $\tau_h = \{I_i\}_{0 \le i \le N}$.

The process of computing the value of $u = u(x)$ based on the corresponding value $x \in [a, b]$ is called *interpolation*. Once we start compute the value of u whose corresponding value x lies out of the given interval $[a, b]$, the process is called *extrapolation*.

Finite Element Interpolation

In most of applications, the form of function $u(x)$ is implicit, consequently it is very hard to determine the explicit form of $u(x)$ with only limited known values of (x_i, u_i). Therefore, we refer to **[[RBD03]**, Section 3.2] to introduce to the approximation of functions by interpolation with finite elements in this section and following section "One-dimensional Finite Element" for one-dimensional case. For the subscripts and superscripts, we recall Figure 14 for the mathematical notation.

For such cases, the original function $u(x)$(x is a vector of spatial coordinates) can be estimated by a linear combination of *interpolation functions* or *shape functions* $\psi_n(x)$ that are associated with the local finite element on the meshed geometric domain. We assume that the shape functions $\psi_n(x)$ are known as priori, therefore the approximation $u_h(x)$ of $u(x)$ is the sum over Nn nodes of domain

$$u_h(x) = u^n \psi_n(x).$$

However, we have to notice that the shape functions are *defined locally at the level of each element* in the standard FEM. For example, node n belongs to certain element e, when we use ψ_n^e to denote the restriction of ψ_n inside element e, we have for every coordinate vector x strictly outside element e

$$\psi_n^e(x) = 0.$$

In contrast, if x is inside or on the boundary of element e, we have

$$u_h(x) = \sum_{n=1}^{Nn} u^n \psi_n(x) = \sum_{n \in e} u^n \psi_n^e(x).$$

In addition to condition $\psi_n^e(x) = 0$, the shape functions must satisfy following two conditions:

- If nodes n, p belong to same element e, x^p denotes the position vector of node p,
$$\psi_n^e(x^p) = \delta_{np},$$
where δ_{np} is the Kronecker delta function.

- For all position vectors x inside or on the boundary of element e,
$$\sum_{n \in e} \psi_n^e(x) = 1.$$

One-dimensional Finite Element

In one-dimensional cases, we use directly the straight line segment as the simplest finite element. Each straight line segment has two nodes at the extremities. We restrict ourselves to linear shape functions on all those individual segments.

In Figure 15, segment g contains the nodes $i-1$, i and has the node i in common with adjacent segment d. Similarly, segment d contains nodes i and $i+1$. Recalling three conditions from section "Finite Element Interpolation", furthermore we take the linearity of shape functions into consideration, overall we express shape functions for element g and element d respectively

$$\psi_{i-1}^g(x) = \frac{x^i - x}{h_g} \text{ and } \psi_i^g(x) = \frac{x - x^{i-1}}{h_g} \text{ for element } g : [x^{i-1}, x^i],$$
$$\psi_i^d(x) = \frac{x^{i+1} - x}{h_d} \text{ and } \psi_{i+1}^d(x) = \frac{x - x^i}{h_d} \text{ for element } d : [x^i, x^{i+1}],$$

here h_g, h_d denote the lengths of element g, h respectively.

We pay our attention to common node i, its shape function ψ_i is defined globally on the entire grid and as an assembling of shape functions ψ_i^g and ψ_i^d as illustrated in Figure 15. Referring to Figure 15, we notice that ψ_i is zero in every other element that does not contain the node i, i.e,

$$\psi_i(x) = 0, \text{ for } x \leq x^{i-1} \text{ or } x \geq x^{i+1}.$$

To construct a transformation between original function u and shape functions ψ_i, let ξ be the local coordinate inside element g

$$\xi = \frac{x - x^{i-1}}{h_g}.$$

Defining shape functions of element g

$$\psi_{i-1}^g = 1 - \xi, \qquad \psi_i^g = \xi.$$

To generalize the shape functions and local element ξ to every straight linear element with the form $[x^i, x^{i+1}]$, we could define

$$\xi = \frac{x - x^i}{x^{i+1} - x^i}.$$

Overall, for each element e, we define a map E_e for transformation between physical space and unit segment $[0, 1]$ (Figure 16). With the help of E_e, we can define the shape functions universally for diverse elements regardless of their original coordinates in physical space. This is the fundamental notion of the *reference element*.

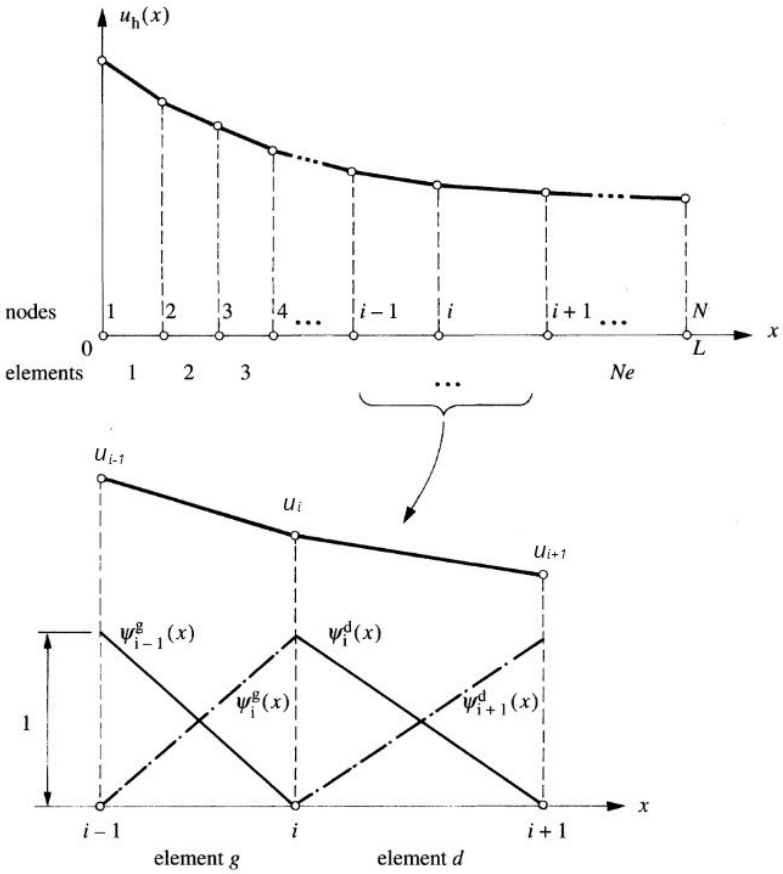

Figure 15: Linear finite elements with two nodes and linear interpolation functions. [**[RBD03], Fig. 3.3**]

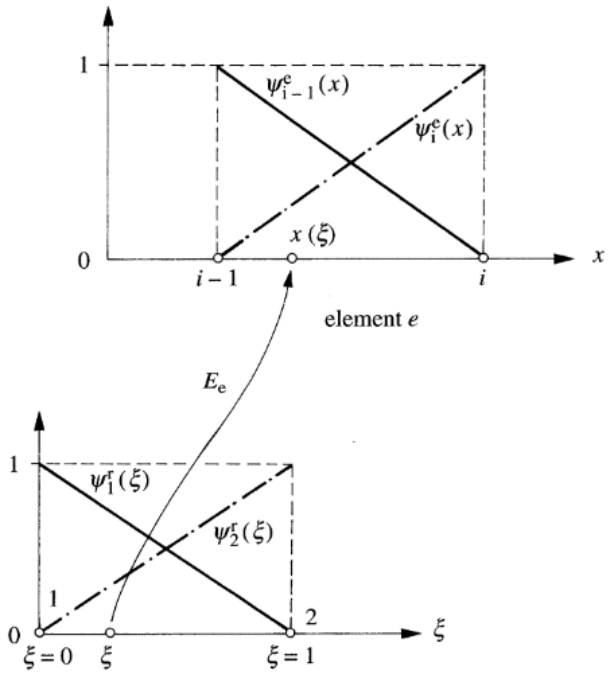

Figure 16: Map Ee and the notion of reference element. [**[RBD03], Fig. 3.4**]

Example: 1D Linear Interpolation of Explicit Function

Let

$$u(x) = cos(\frac{x^2}{8}) + 1, \quad x \in [0, 10].$$

We assume simply uniform step size $h_i = 1$ for $0 \leq i \leq 9$. Therefore, we have the set of vertices $\{x_0 = 0, x_1 = 1, ..., x_{10} = 10\}$ for $N = 9$.

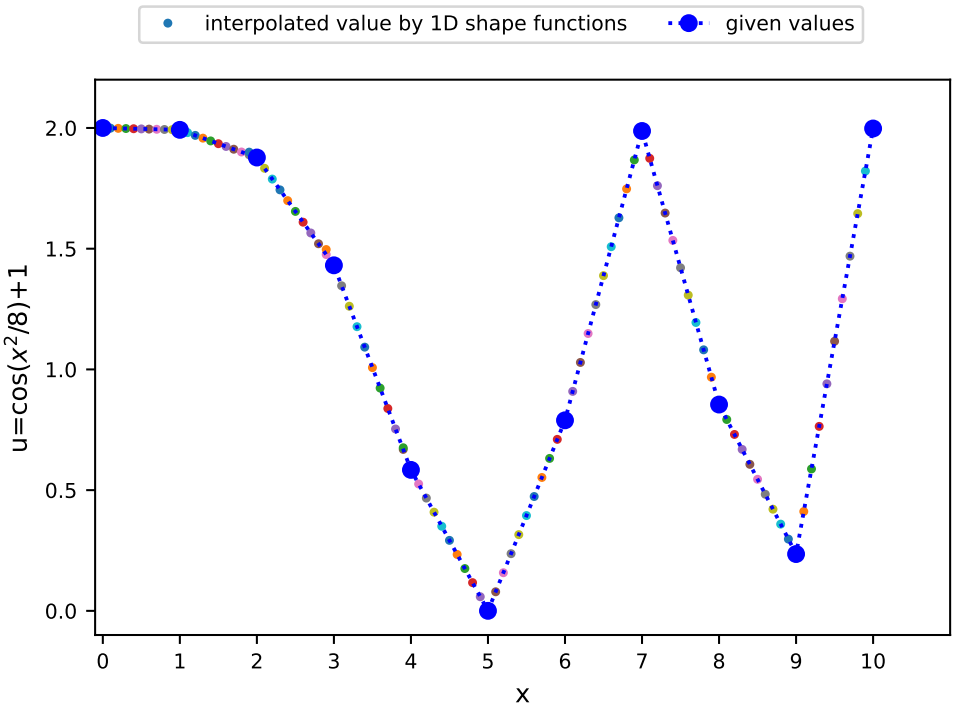

Figure 17: Interpolated value $x \in [0, 10]$ and $x_i < x = x_i + 0.1 \cdot k < x_{i+1}$ for $0 \leq i \leq 9$, $1 \leq k \leq 9$.

For a better understanding, we introduce 1D linear interpolation by Algorithm 1. We assume interpolated value x between two nodes x_i, x_{i+1} and want to find out its corresponding value $u(x)$. Moreover, we introduce the transformation between original function $u(x)$ and shape functions $\psi_i(\xi)$ by using parameter ξ.

As a simple example, we use an explicit function $u(x)$. However, the "ShapeFunction" in Algorithm 1 can be used in the interpolation of an implicit function as well. In the case of implicit function, several values of x_i and corresponding u_i will be given. If we want to compute a numerical approximation by interpolation method in $x \in (x_i, x_{i+1})$, Algorithm 1 is still available with only small modifications.

Algorithm 1: 1D Linear Interpolation of Explicit Function $u(x)$

Input: x_i, x, x_{i+1}

// u_i and u_{i+1} are also Input in the case of an implicit function, for $x \in (x_i, x_{i+1})$

Output: u

1 **Def** `ShapeFunction`(x_i, x, x_{i+1})**:**

2 $\quad \xi = \dfrac{x - x_i}{x_{i+1} - x_i}$

3 $\quad \psi_1 = 1 - \xi$

4 $\quad \psi_2 = \xi$

5 $\quad$ print ψ_1, ψ_2

6 $\quad$ **return** ψ_1, ψ_2

7

// constants a, b, c

8 **Function** `Main`**:**

9 $\quad x_i = a$

10 $\quad x = b$

11 $\quad x_{i+1} = c$

12 $\quad$ ShapeFunction(a, b, c)

// the line 13 - 14 can be deleted in the case of an implicit function

13 $\quad u_i = u(a)$

14 $\quad u_{i+1} = u(c)$

15 $\quad u = \psi_1 \cdot u_i + \psi_2 \cdot u_{i+1}$

16 $\quad$ print u

17 $\quad$ **return** u

Chapter 3

Finite Elements in Interpolation Procedures

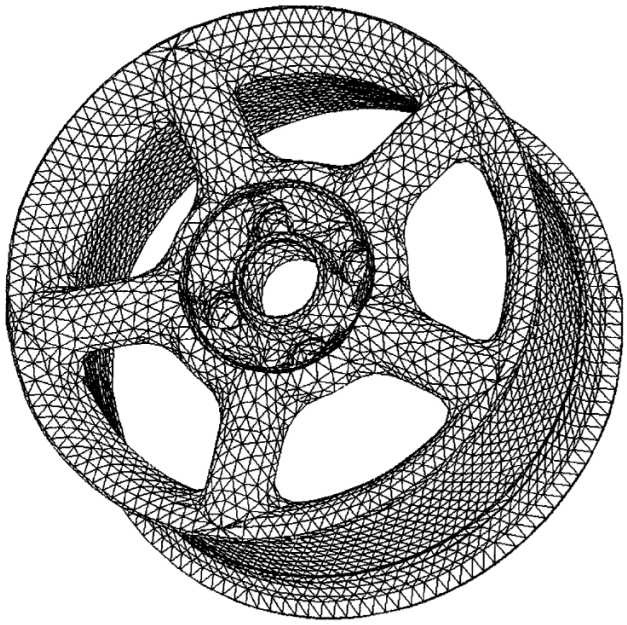

Figure 18: 3D finite element mesh of an automobile wheel rim. [**[RBD03], Figure 3.2**]

From Chapter 2, we obtain a comprehensive overview of classical linear elasticity and interpolation procedure. Naturally, we would like to present a much more detailed procedure in three-dimensional space to illustrate how to implement interpolation with finite elements in Chapter 3.

As mathematicians, we are well aware of how complicated advanced mathematical modeling is for non-experts. Therefore, we refer to [**[TAMC11]**, Section 2] and [**[DO10]**, Section 4.1.3] for a brief summary of 3D finite elements and shape functions from a theoretical perspective. After that, we illustrate in detail the 3D numerical approach and algorithm.

This chapter primarily involves my own work, all implementations in Python were completed by myself. The original data set for constructing triangular mesh as well as the mechanical and geometric information about reinforced composite PBT are provided by Fraunhofer ITWM and based on [**[ADS$^+$19]**, Chapter 3].

3.1 A Specific Introduction to Research Background

To specify the certain material that we are modeling in Chapter 3, we introduce shortly the scientific background of this material with several given material mechanical parameters and geometric settings.

In this thesis, we are focusing on computation and modeling based on the local fiber orientation tensor of thermoplastic engineering polymer *Polybutylene Terephthalate (PBT)*. In the field of electrical and electronics industries, PBT is commonly utilized as insulators. Due to some excellent characteristics of PBT such as high resistance to fuels, oil and fats, PBT is frequently used in the automotive industry as well. Back to our normal daily life, PBT is easily found in keycaps and has been often regarded as the most durable material for keyboard production.

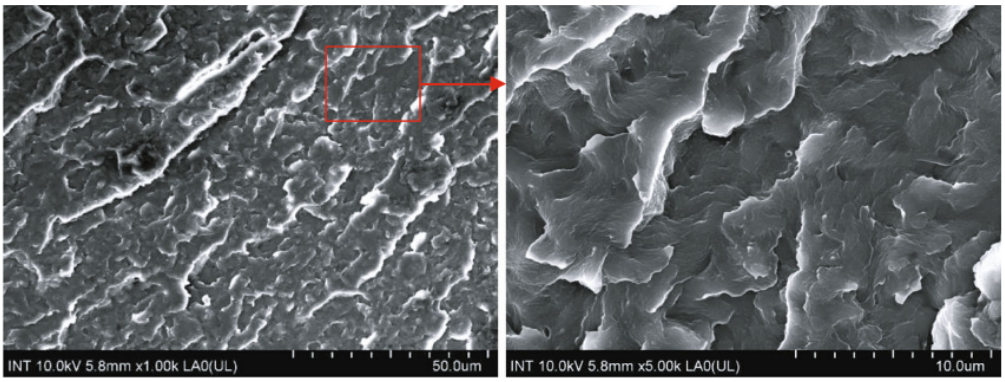

Figure 19: Neat PBT. [**[TH21], Fig. 5(a)**]

Out of certain consideration and based on the given experiment data from Fraunhofer ITWM, we construct 3D mathematical models for a specific composite PBT material with following mechanical properties of the constituents referring to [**[ADS+19]**, Chapter 3]:

- Young's Modulus of glass 72 Gpa
- Young's Modulus of PBT 2.47 Gpa
- Poisson ratio of glass 0.22
- Poisson ratio of PBT 0.4

and geometric setting:

- Fiber diameter 10 μm
- Fiber length 250 μm
- Fiber volume fractions of glass 10%, 14% and 18%
- Voxel length 1.9531 μm
- Domain size 2 $mm \times 2\ mm \times 2\ mm$

We pay our attention to the third geometric setting "fiber volume fraction". For different PBT materials, different "fiber volume fraction" reveals the constituent ratio of the glass component. For example, 18% fiber volume fraction of glass is corresponding to PBT GF 30 which is extensively used to manufacture airflow sensors.

3.2 Finite Elements in Two- and Three-dimensional Spaces

In Section 2.3.2, we have seen a short introduction to the simplest one-dimensional finite element straight line segment. However, in practice two- and three-dimensional finite elements have been used much more frequently in numerical computation.

3.2.1 Two-dimensional Triangular Finite Element

In two-dimensional spaces, triangle is one of the most common finite elements. Any triangle with a nonzero area can be transformed into a unit standard triangle which has been recognized as a *reference element*, *cf.* the right-hand side diagram in Figure 20 [**[LQT17]**, Section 9.4].

Therefore, we define three nonzero shape functions to complete the linear transformation, namely,

$$\begin{cases} \psi_1(\xi,\eta) = 1-\xi-\eta, \\ \psi_2(\xi,\eta) = \xi, \\ \psi_3(\xi,\eta) = \eta. \end{cases} \tag{3.1}$$

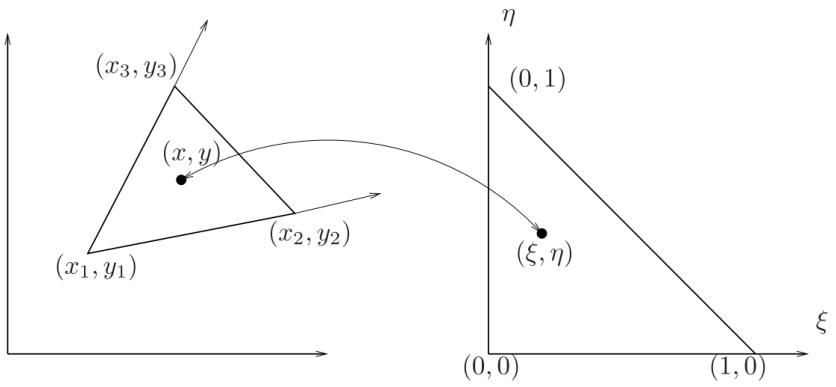

Figure 20: The linear transform from an arbitrary triangle to the unit standard triangle and the inverse map. **[[LQT17], Figure 9.8]**

Referring to Figure 20, the linear transformation from left-hand side triangle with vertices $(x_i, y_i)_{i=1,2,3}$ can be arranged in the counter-clockwise direction to the unit standard triangle, such that

$$\begin{cases} x = \sum_{i=1}^{3} x_i \psi_i(\xi,\eta), \\ y = \sum_{i=1}^{3} y_i \psi_i(\xi,\eta), \end{cases} \tag{3.2}$$

alternatively

$$\begin{cases} \xi = \dfrac{1}{2A_e}\Big((y_3 - y_1)(x - x_1) - (x_3 - x_1)(y - y_1)\Big), \\ \eta = \dfrac{1}{2A_e}\Big((x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)\Big), \end{cases} \tag{3.3}$$

where A_e is the area of left-hand side triangle that can be calculated using the vertices $(x_i, y_i)_{i=1,2,3}$ via

$$A_e = det \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}. \tag{3.4}$$

3.2.2 Three-dimensional Finite Elements

Prismatic Finite Element

In this thesis, 3D prismatic finite element plays a significant role. Being similar to Section 3.2.1, we introduce briefly prismatic finite element with six nodes (Figure 21) based on **[TAMC11]**.

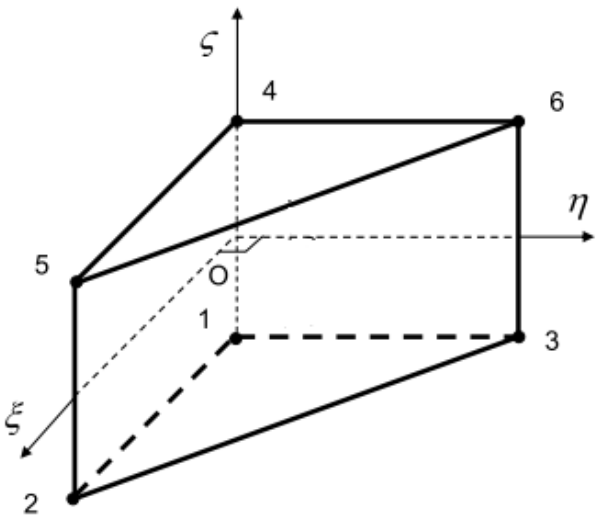

Figure 21: Reference prismatic element with 6 nodes. **[[TAMC11], Fig. 1]**

For an arbitrary prism located in $xyz-$space, let x, y, z directions are parallel to ξ, η and ζ axes respectively. Therefore, we define isoparametric shape functions for six-node prismatic element

$$\begin{cases} \psi_1(\xi,\eta,\zeta) = \frac{1}{2}(1-\zeta)(1-\xi-\eta), \\ \psi_2(\xi,\eta,\zeta) = \frac{1}{2}(1-\zeta)\xi, \\ \psi_3(\xi,\eta,\zeta) = \frac{1}{2}(1-\zeta)\eta, \\ \psi_4(\xi,\eta,\zeta) = \frac{1}{2}(1+\zeta)(1-\xi-\eta), \\ \psi_5(\xi,\eta,\zeta) = \frac{1}{2}(1+\zeta)\xi, \\ \psi_6(\xi,\eta,\zeta) = \frac{1}{2}(1+\zeta)\eta, \end{cases} \tag{3.5}$$

with certain restriction in reference element for $\xi \in [0,1], \eta \in [0, 1-\xi], \zeta \in [-1,1]$.

To compute the values of ξ, η and ζ, we consider the relation between spatial coordinates x_i of point x and reference nodal coordinates x_{iJ} of nodes J. Taking six shape functions into consideration, we have

$$x_i(\xi,\eta,\zeta) = x_{iJ}\psi_J(\xi,\eta,\zeta) = \sum_{J=1}^{6} x_{iJ}\psi_J(\xi,\eta,\zeta), \quad i = 1,2,3. \tag{3.6}$$

Tetrahedral Finite Element

In [**[PH08]**, Section 4.1], we find six different methods to divide a prism into three tetrahedra. We highly recommend **[DLVC99]** to readers who are interested in the subdivision of common 3D finite elements.

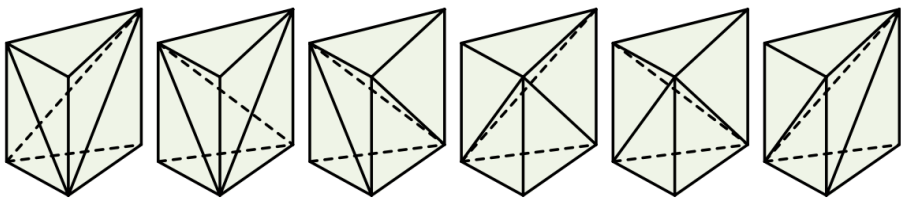

Figure 22: Six different methods to divide a prism into three tetrahedra. [**[PH08], Fig. 5**]

Referring to [**[DO10]**, Section 4.1.3] and Figure 23(a), a linear shape function of vertex P_i in an arbitrary tetrahedron in Cartesian system has the general form

$$\psi_i(x, y, z) = a_i + b_i x + c_i y + d_i z, \quad i = 1, 2, 3, 4. \tag{3.7}$$

The coefficients a_i, b_i, c_i and d_i can be calculated by considering the condition

$$\psi_j(\overrightarrow{r_i}) = \delta_{ij}, \quad i, j = 1, 2, 3, 4. \tag{3.8}$$

As a result, a system of four equations for the four unknown coefficients is obtained.

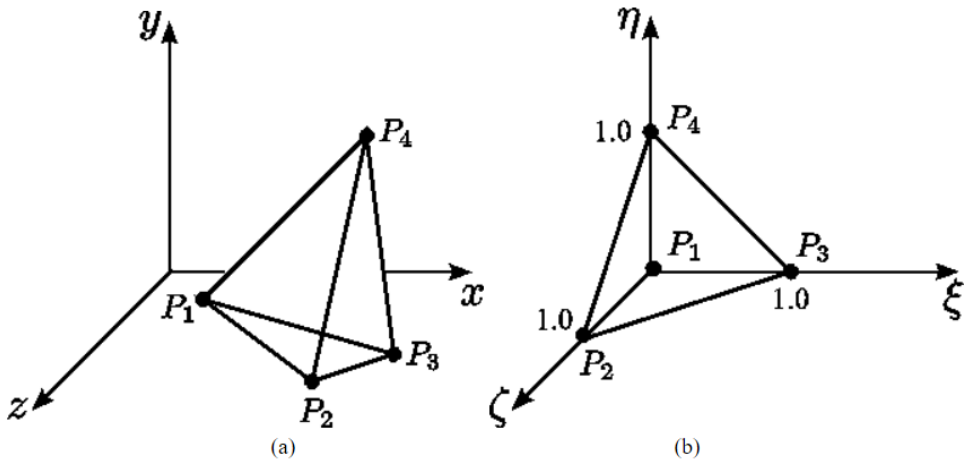

Figure 23: Tetrahedral finite element. (a) Original spatial coordinate system. (b) Reference coordinate system. [**[DO10]**, Section 4.1.3]

Being similar to prismatic finite element, an arbitrary point (x, y, z) from originally spatial coordinate system can be mapped to a corresponding point (ξ, η, ζ) in the reference coordinate system. Additionally, we can also compute the numerical values of parameters ξ, η and ζ by

$$\begin{cases} x = x_1 + (x_2 - x_1)\xi + (x_3 - x_1)\eta + (x_4 - x_1)\zeta, \\ y = y_1 + (y_2 - y_1)\xi + (y_3 - y_1)\eta + (y_4 - y_1)\zeta, \\ z = z_1 + (z_2 - z_1)\xi + (z_3 - z_1)\eta + (z_4 - z_1)\zeta. \end{cases} \tag{3.9}$$

In this way, we define shape functions for four vertices in the reference coordinate

system

$$\begin{cases} \psi_1(\xi,\eta,\zeta) = 1-\xi-\eta-\zeta, \\ \psi_2(\xi,\eta,\zeta) = \xi, \\ \psi_3(\xi,\eta,\zeta) = \eta, \\ \psi_4(\xi,\eta,\zeta) = \zeta. \end{cases} \tag{3.10}$$

3.3 Algorithm: Prismatic Finite Element in Interpolation Procedure

Out of consideration of the certain construction of 2D triangular mesh (Figure 6), we begin with the 3D interpolation with prismatic finite element. First of all, we sketch briefly the computing procedure:

- Step 1: Construct the two-dimensional triangular mesh results Figure 6;
- Step 2: Generate an arbitrary 3×3 second order fiber orientation tensor A, recall Equation (1.2), the coordinate of point $P : (\lambda_1, \lambda_2)$ as the given orientation state (orange) in two-dimensional mesh, see Figure 6;
- Step 3: Determine the smallest local triangle T of P in two-dimensional triangular mesh, such that $\Lambda \in T = conv\{\Lambda_1, \Lambda_2, \Lambda_3\}$, see Figure 6;
- Step 4: Extend the third coordinate in vertical direction using random fiber fraction volume $z' \in [0.10, 0.18]$ (see Section 3.1), construct prisms and then determine the smallest local prismatic finite element for $P : (\lambda_1, \lambda_2, z')$;
- Step 5: Recall Equations (3.5) and (3.6), compute values of shape functions $(\psi_i)_{i=1:6}$;
- Step 6: Validation and visualization.

To illustrate each step in the above procedure clearly, we would state corresponding algorithm firstly and then present the result by figure or table in this section.

Step 1: Construct 2D orientation triangular mesh

From Fraunhofer ITWM, we receive the coordinates of 12 nodes to construct the triangular mesh, we use a tuple $\Lambda_i : (\lambda_{1,i}, \lambda_{2,i})$ to denote the given nodes, where subscript i is the index of a given node.

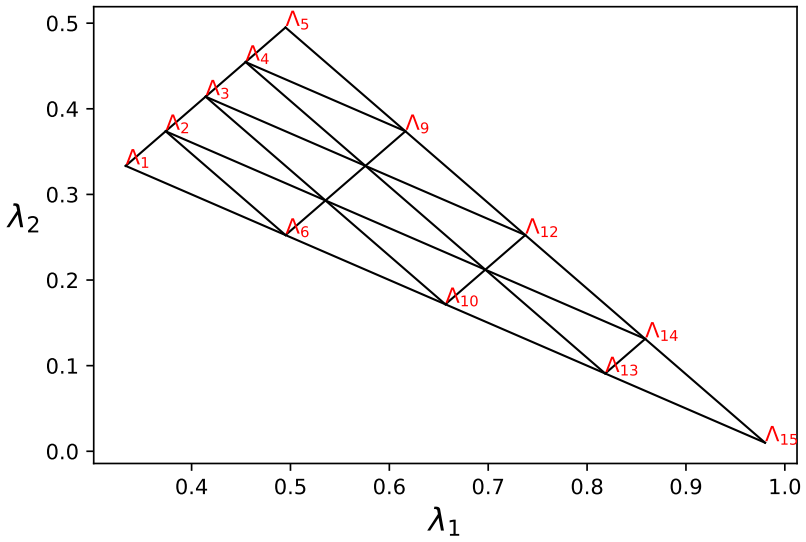

Figure 24: Orientation triangle based on given nodes Λ_i from Fraunhofer ITWM.

Referring to Figure 24, we notice that the nodes Λ_7, Λ_8 and Λ_{11} are still missing, and therefore, we must figure out an algorithm to determine the coordinate of those three nodes.

We consider the intersection $I : (I_x, I_y)$ of two lines L_1 and L_2 in a two-dimensional $xy-$plane with L_1 being defined by two distinct nodes (x_1, y_1) and (x_2, y_2), L_2 being defined by two distinct nodes (x_3, y_3) and (x_4, y_4). For example, we let Λ_2, Λ_{14} be the two nodes of L_1, let Λ_3, Λ_{10} be the two nodes of L_2, such that we can find out intersection Λ_7 in $\lambda_1\lambda_2-$ plane. Implementing similar procedure, we can find out the intersections Λ_8 and Λ_{11}. For simplification, we only discuss the algorithm that two lines L_1, L_2 are not parallel or coincident, i.e. $D \neq 0$ in Algorithm 2.

Algorithm 2: Determine intersection I of two lines L_1, L_2

Input: x_i, y_i for $i = 1, 2, 3, 4$

Output: I_x, I_y

Def `LineIntersection`$(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$**:**

$$D = (x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)$$

$$I_x = \frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{D}$$

$$I_y = \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{D}$$

return I_x, I_y

Function `Main`**:**

$x_1, y_1 = \Lambda_2[0], \Lambda_2[1]$

$x_2, y_2 = \Lambda_{14}[0], \Lambda_{14}[1]$

$x_3, y_3 = \Lambda_3[0], \Lambda_3[1]$

$x_4, y_4 = \Lambda_{10}[0], \Lambda_{10}[1]$

LineIntersection$(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$

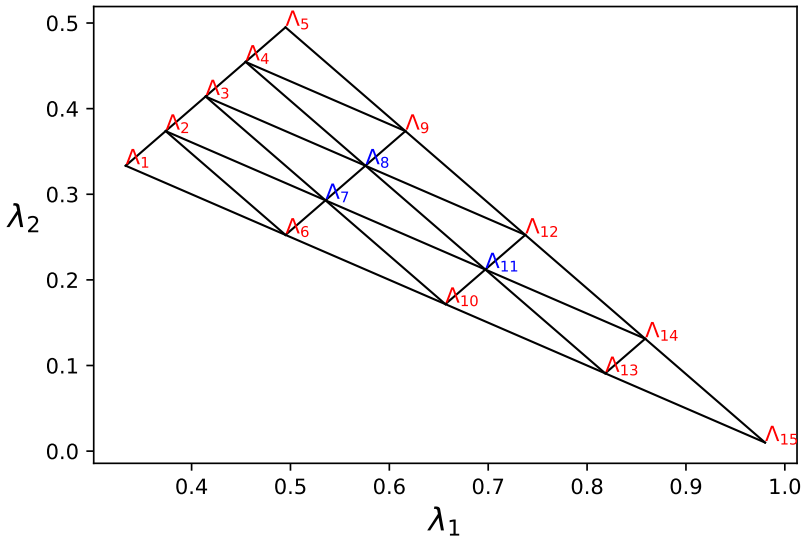

Figure 25: Complete reference orientation triangle contains 15 nodes.

In Figure 25, we notice that actually there exist 16 small triangles in two different directions. When we use triangular finite element, we normally annotate the three vertices of a triangular element in a counterclockwise direction.

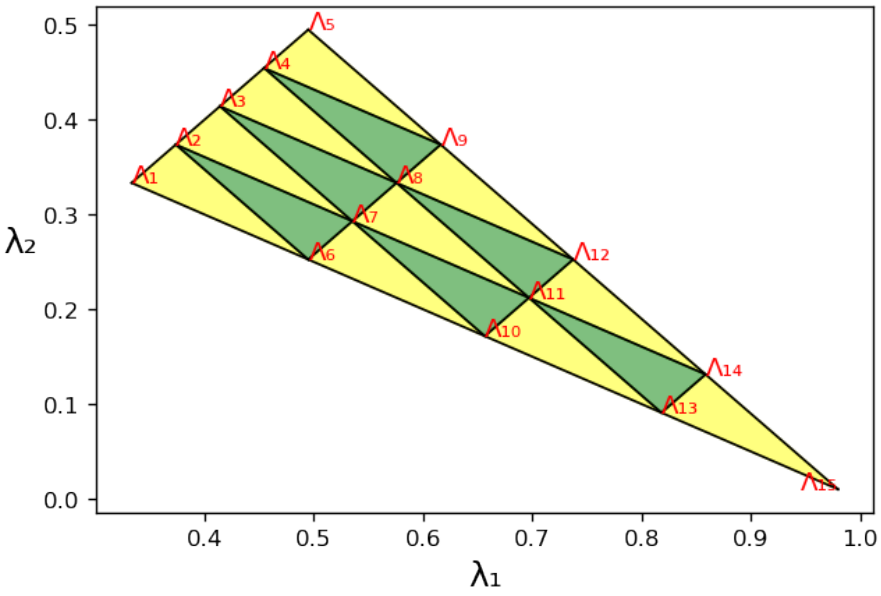

Figure 26: Color yellow denotes the orientation triangle in direction 1, color green denotes the orientation triangle in direction 2.

For the 1-directional triangles in Figure 26, for instance, the triangle to which node Λ_1 belongs, we define an ordered tuple to contain three vertices, i.e. $(\Lambda_2, \Lambda_1, \Lambda_6)$. For the adjacent 2-directional triangle to $\triangle\Lambda_2\Lambda_1\Lambda_6$, we define its vertices tuple $(\Lambda_2, \Lambda_6, \Lambda_7)$. Those two principles for annotating three vertices will be applied to all orientation triangles in Figure 26.

Table 2: Collections of triangles in two directions in Figure 26

Collection of 1-directional triangles	Collection of 2-directional triangles
$\triangle\Lambda_2\Lambda_1\Lambda_6$	$\triangle\Lambda_2\Lambda_6\Lambda_7$
$\triangle\Lambda_3\Lambda_2\Lambda_7$	$\triangle\Lambda_3\Lambda_7\Lambda_8$
$\triangle\Lambda_4\Lambda_3\Lambda_8$	$\triangle\Lambda_4\Lambda_8\Lambda_9$
$\triangle\Lambda_5\Lambda_4\Lambda_9$	$\triangle\Lambda_7\Lambda_{10}\Lambda_{11}$
$\triangle\Lambda_7\Lambda_6\Lambda_{10}$	$\triangle\Lambda_8\Lambda_{11}\Lambda_{12}$
$\triangle\Lambda_8\Lambda_7\Lambda_{11}$	$\triangle\Lambda_{11}\Lambda_{13}\Lambda_{14}$
$\triangle\Lambda_9\Lambda_8\Lambda_{12}$	
$\triangle\Lambda_{11}\Lambda_{10}\Lambda_{13}$	
$\triangle\Lambda_{12}\Lambda_{11}\Lambda_{14}$	
$\triangle\Lambda_{14}\Lambda_{13}\Lambda_{15}$	

Step 2: Generate fiber orientation tensor A and determine $P : (\lambda_1, \lambda_2)$

Recall Equations (1.2) and (1.3), we discuss now how to define practically a 3×3 matrix A that satisfies several certain conditions:

- A is a symmetric positive semidefinite matrix and $A = R \cdot \Lambda \cdot R^T$;
- $tr(A) = 1$ for sorted eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$;
- $P : (\lambda_1, \lambda_2)$ lies inside the largest $\triangle\Lambda_5\Lambda_1\Lambda_{15}$.

We notice that package `Numpy` in Python provides many useful built-in functions relevant to linear algebra, therefore we present the Python code here directly to generate the random $P : (\lambda_1, \lambda_2)$ in two-dimensional $\lambda_1\lambda_2-$plane. In the following code, we use w instead of Λ.

```
import numpy as np

# Generate a random 3 x 3 matrix
A_random = np.random.rand(3,3)
# Generate a symmetric 3 x 3 matrix
A_symmetric = 0.5 * (A_random + np.transpose(A_random))
# Compute eigenvalues w, eigenvectors R of A_symmetric
w,R = np.linalg.eig(A_symmetric)
# Compute transpose of eigenvectors matrix R
Rt = np.transpose(R)
# Given an interval, values outside the interval are clipped to the
                                    interval edge(s)
w = w.clip(min=0.01)
# Return a matrix norm with order 1, such that we scale the sum of
                                    three components in w to 1
w /= np.linalg.norm(w, 1)
# Construct diagonal matrix
wDiagonal = np.diag(w)
# Define the fiber orientation tensor A
A_positiveDefinite = np.real(np.dot(np.dot(R, wDiagonal), Rt))
# Compute eigenvalues and eigenvectors of A
```

```
evalue, evect = np.linalg.eig(A_positiveDefinite)
# Sort three eigenvalues in descending numerical order
evalue1 = np.sort(evalue)[::-1]
# Generate 2D coordinate by using eigenvalues
EV_coor_2D = [evalue1[0], evalue1[1]]
```

Step 3: Determine the smallest local triangle of $P : (\lambda_1, \lambda_2)$ from Table 2

To determine whether an arbitrary point lies inside a triangle or not by using three vertices of a triangle, we could actually find various classical algorithms. In this thesis, we introduce classical "*Same Side Technique*" in two-dimensional $xy-$plane.

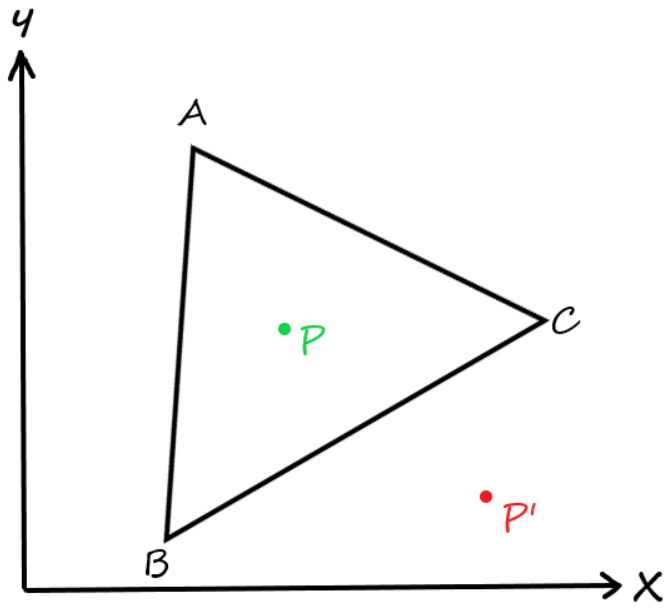

Figure 27: Reference coordinates $A : (a_x, a_y)$, $B : (b_x, b_y)$, $C : (c_x, c_y)$, $P : (x, y)$, $P' : (x', y')$.

Using Algorithm 3, we determine the smallest local triangle from Table 2 for $P : (\lambda_1, \lambda_2)$. For example, for $\triangle\Lambda_2\Lambda_1\Lambda_6$ from Table 2, we let vertices $\Lambda_2, \Lambda_1, \Lambda_6$ be vertices A, B, C with reference coordinates respectively. Moreover, we use Algorithm 3 to verify whether $P : (\lambda_1, \lambda_2)$ is inside $\triangle\Lambda_5\Lambda_1\Lambda_{15}$ or not. If $P : (\lambda_1, \lambda_2)$ is not inside the largest $\triangle\Lambda_5\Lambda_1\Lambda_{15}$, we should abandon the corresponding fiber orientation tensor A and generate a new A.

Algorithm 3: Determine whether point P is inside $\triangle ABC$ or not based on Figure 27

Input: $a_x, a_y, b_x, b_y, c_x, c_y, x, y$

Output: **True** or **False**

Def `InTriangle`$(a_x, a_y, b_x, b_y, c_x, c_y, x, y)$**:**

`// Compute three cross products`

$Product1 = (x - b_x)(a_y - b_y) - (a_x - b_x)(y - b_y)$

$Product2 = (x - c_x)(b_y - c_y) - (b_x - c_x)(y - c_y)$

$Product3 = (x - a_x)(c_y - a_y) - (c_x - a_x)(y - a_y)$

`// Check signs of three cross products`

if $Product1 > 0$ **and** $Product2 > 0$ **and** $Product3 > 0$ **then**:

    return True

    print("Point P is inside triangle ABC")

else if $Product1 < 0$ **and** $Product2 < 0$ **and** $Product3 < 0$ **then**:

    return True

    print("Point P is inside triangle ABC")

else:

    return False

    print("Point P is outside triangle ABC")

Function `Main`**:**

`// Example: check` $P : (\lambda_1, \lambda_2)$ `is inside` $\triangle \Lambda_2 \Lambda_1 \Lambda_6$

$a_x, a_y = \Lambda_2[0], \Lambda_2[1]$

$b_x, b_y = \Lambda_1[0], \Lambda_1[1]$

$c_x, c_y = \Lambda_6[0], \Lambda_6[1]$

$x, y = \lambda_1, \lambda_2$

InTriangle$(a_x, a_y, b_x, b_y, c_x, c_y, x, y)$

Step 4: Extend the third coordinate z' of $P : (\lambda_1, \lambda_2)$ and determine the smallest local prismatic finite element for $P : (\lambda_1, \lambda_2, z')$

Recall Section 3.1, in this thesis we extend the third coordinate z' of P by using fiber volume fraction. Therefore, we generate a uniformly arbitrary value z' from interval $[0.10, 0.18]$ by using `random` library in Python. The Python code below presents the complete 3D coordinate of $P : (\lambda_1, \lambda_2, z')$ following the Python code from Page 46.

```
import random
random_point_3D = [EV_coor_2D[0], EV_coor_2D[1], random.uniform(0.1
                                                , 0.18)]
```

Once we obtain the 3D coordinate, we could eventually construct prismatic finite element mesh. Recall Figure 25, we have 15 nodes on 2D $\lambda_1\lambda_2-$plane, and λ_1, λ_2 denote two largest eigenvalues of 3×3 second order fiber orientation tensor A. In the vertical direction, we use z' to denote the fiber volume fraction where z' lies in interval $[0.10, 0.18]$.

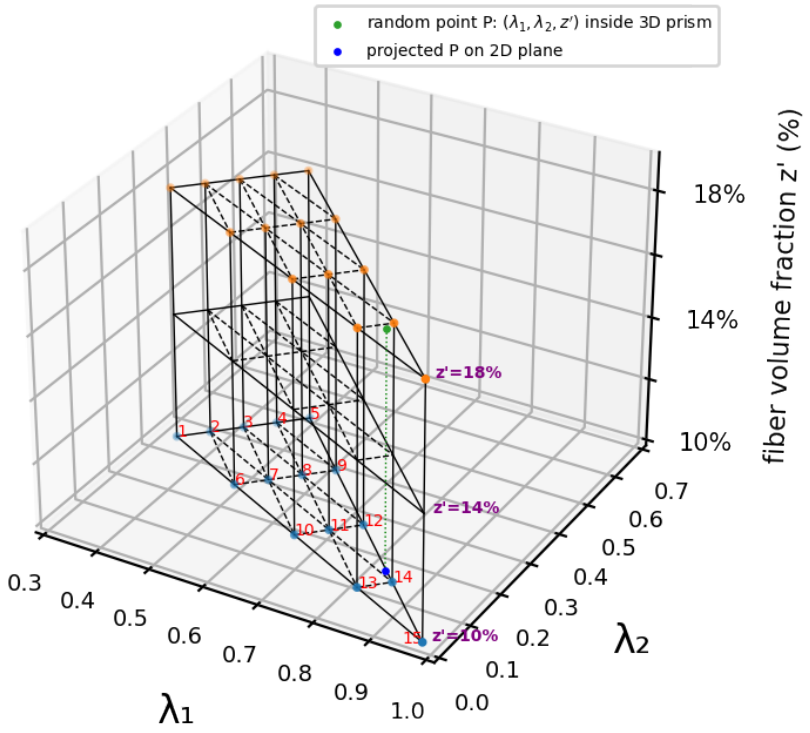

Figure 28: Two-layer prismatic finite element mesh with demarcation points 10%, 14% and 18%. For simplification, we use constant i to denote Λ_i on $\lambda_1\lambda_2-$plane, for $i = 1 : 15$.

Using Algorithm 3, we could determine the smallest local triangle of 2D $P : (\lambda_1, \lambda_2)$. To determine the smallest local prism of the 3D $P : (\lambda_1, \lambda_2, z')$, we require one more algorithm to determine the local layer. Namely, we have to determine that the z' belongs to interval [10%,14%] or interval [14%,18%] in vertical direction.

We suppose that in the future we may use a multiple-layer prismatic finite element model, namely, the number of layers is greater than 2, therefore we require the Python code adapts to the flexible change of number of layers.

```
# Set the number of layers
layers = int(2)
print('We have now '+str(layers)+' layer(s):')

layers_list = []
difference = 0.18 - 0.10

# we add layers as tuple into the layer_list, i.e. (0.10,0.14), (0.
                                    14,0.18)
for i in range(0, layers):
    layers_list.append((i*(difference/layers)+0.10, (i+1)*(
                                        difference/layers)+0.10))
    print('The layer {0} from'.format(i+1), layers_list[i])

# determine the located layer of third coordinate z'
for start, stop in layers_list:
    if start <= random_point_new_3d[2] <= stop:
        located_layer = (start, stop)
        for pair in layers_list:
            if located_layer == pair:
                global index
                index = layers_list.index(pair)
                print('We found the random point in layer '+str(
                                                    index+1))
```

Step 5: Compute values of shape functions $(\psi_i)_{i=1:6}$

Based on the theoretical introduction to prismatic finite element in Section 3.2.2, we present how to transfer the mathematical items from theory to computer language in Step 5.

Before we present our Python code for defining and computing six shape functions, we explain briefly the basic steps in our Python code:

1. Input the coordinates of a random point $P : (\lambda_1, \lambda_2, z')$ and vertices of local prismatic finite element $P_i : (\lambda_{1,i}, \lambda_{2,i}, z'_i)$ for $i = 1 : 6$, in Python code we use x_i, y_i, z_i instead of $\lambda_{1,i}, \lambda_{2,i}, z'_i$;

2. Define six shape functions $\psi_i(\xi, \eta, \zeta)$ by Equation (3.5), in Python code we use $N_i(x, y, z)$ instead of $\psi_i(\xi, \eta, \zeta)$;

3. Solve the system of three linear equations:

$$\begin{cases} \lambda_1 = \sum_{i=1}^{6} \psi_i \lambda_{1,i}, \\ \lambda_2 = \sum_{i=1}^{6} \psi_i \lambda_{2,i}, \\ z' = \sum_{i=1}^{6} \psi_i z'_i, \end{cases} \tag{3.11}$$

   such that we find the numerical values of ξ, η, ζ. Alternatively, the system (3.11) of three linear equations in Python will be formulated as

$$\begin{cases} \lambda_1 = \sum_{i=1}^{6} N_i x_i, \\ \lambda_2 = \sum_{i=1}^{6} N_i y_i, \\ z' = \sum_{i=1}^{6} N_i z_i, \end{cases} \tag{3.12}$$

   in Python we use library `sympy.solvers` to find out the numerical values of x, y, z.

4. Plug the numerical values of ξ, η, ζ into ψ_i, let $s_i = \psi_i$, the numerical values of s_i will be the outputs of Python code.

Recall Figure 26 and Table 2, we have two-directional triangles. Therefore, we consider the annotated order of six vertices as well according to Table 2.

(a) Standard triangulation prism 1 (b) Standard triangulation prism 2

Figure 29: Recall Table 2, (a) The prism is constructed using one of the Collection of 1-directional triangles; (b) The prism is constructed using one of the Collection of 2-directional triangles. We illustrate the annotating order of six vertices in this figure.

```
import sympy as sy
from sympy.solvers import solve

def ShapeFunction_Prism(random_pt=[], V1=[], V2=[], V3=[], V4=[],
                                    V5=[], V6=[]):
# Step 1: Referring to Figure 29, we input the 3D coordinates of
                                    random point P and six vertices
                                    of the smallest local prismatic
                                    finite element
    lam1_coor = random_pt[0]
    lam2_coor = random_pt[1]
    z_RanPt = random_pt[2]

    global x1, x2, x3, x4, x5, x6
    x1 = V1[0]
    x2 = V2[0]
    x3 = V3[0]
    x4 = V4[0]
    x5 = V5[0]
    x6 = V6[0]
    global y1, y2, y3, y4, y5, y6
```

```
    y1 = V1[1]
    y2 = V2[1]
    y3 = V3[1]
    y4 = V4[1]
    y5 = V5[1]
    y6 = V6[1]
    global z1, z2, z3, z4, z5, z6
    z1 = V1[2]
    z2 = V2[2]
    z3 = V3[2]
    z4 = V4[2]
    z5 = V5[2]
    z6 = V6[2]
# Step 2: Declare variables x,y,z and define six shape functions
    x,y,z = sy.symbols("x,y,z", real=True)
    N1 = 0.5*(1-x-y)*(1-z)
    N2 = 0.5*x*(1-z)
    N3 = 0.5*y*(1-z)
    N4 = 0.5*(1-x-y)*(1+z)
    N5 = 0.5*x*(1+z)
    N6 = 0.5*y*(1+z)
# Step 3: Solve the system of three linear equations and find out
                                    the numerical values of x,y,z
    solution = solve([lam1_coor-(x1*N1+x2*N2+x3*N3+x4*N4+x5*N5+x6*
                                        N6), lam2_coor-(y1*N1+y2*N2+
                                        y3*N3+y4*N4+y5*N5+y6*N6),
                                        z_RanPt-(z1*N1+z2*N2+z3*N3+z4
                                        *N4+z5*N5+z6*N6)], (x, y, z),
                                        dict = True)
    key, val = zip(*solution[0].items())
# Step 4: Assign and return the values of coefficients s_i
    global s1, s2, s3, s4, s5, s6
    s1 = 0.5*(1-val[0]-val[1])*(1-val[2])
    s2 = 0.5*val[0]*(1-val[2])
    s3 = 0.5*val[1]*(1-val[2])
    s4 = 0.5*(1-val[0]-val[1])*(1+val[2])
    s5 = 0.5*val[0]*(1+val[2])
```

```
    s6 = 0.5*val[1]*(1+val[2])

    return s1, s2, s3, s4, s5, s6
```

Step 6: Validation and visualization

Once we obtain six coefficients s_i, we could verify the correctness by comparing the $P^{byPFE} : (\lambda_1^{byPFE}, \lambda_2^{byPFE}, z'^{byPFE})$ and $P : (\lambda_1, \lambda_2, z')$. We set

$$\begin{cases} \lambda_1^{byPFE} = \sum_{i=1}^{6} s_i \lambda_{1,i}, \\ \lambda_2^{byPFE} = \sum_{i=1}^{6} s_i \lambda_{2,i}, \\ z'^{byPFE} = \sum_{i=1}^{6} s_i z'_i. \end{cases} \tag{3.13}$$

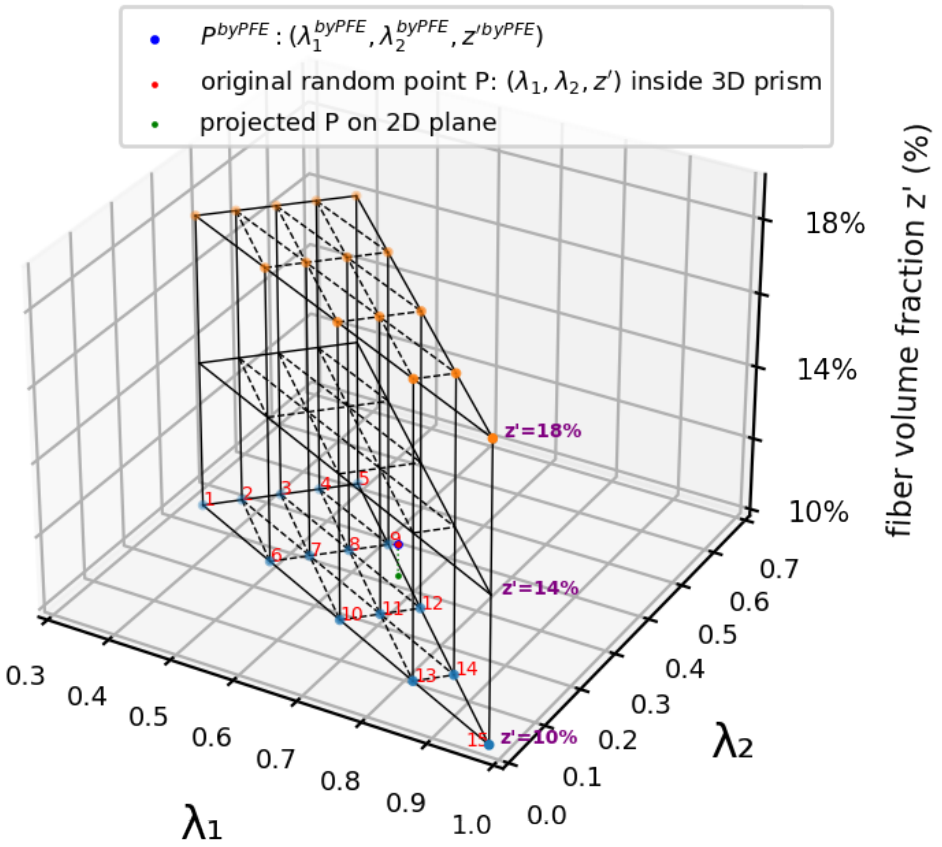

Figure 30: Once we see the overlap of P^{byPFE} and P, we claim the correctness of using prismatic finite element in interpolation procedure.

3.4 Algorithm: Tetrahedral Finite Element in Interpolation Procedure

Referring to [**[PH08]**, Section 4.1] and Figure 22, we notice that we have six different methods to divide a prism into three tetrahedra. In this thesis, we introduce only one dividing method that we see from the left-hand side second prism in Figure 22.

The computing procedure in this section is highly similar with previous section except two small modifications in Step 4 and Step 5 respectively:

- Step 4: Extend the third coordinate in vertical direction using random fiber fraction volume $z' \in [0.10, 0.18]$, construct prisms and divide all prisms into three tetrahedra using same method; then determine the local tetrahedral finite element for $P : (\lambda_1, \lambda_2, z')$;
- Step 5: Recall Equations (3.9) and (3.10), compute values of shape functions $(\psi_i)_{i=1:4}$.

To avoid meaningless repetition, we only state the algorithms that have not been discussed in the previous section. Namely, we will only focus on the specific algorithms in Step 4 and Step 5.

Step 4: Extend the third coordinate z' of $P : (\lambda_1, \lambda_2)$ and determine the smallest local tetrahedral finite element for $P : (\lambda_1, \lambda_2, z')$

Recall Figure 29, we have prisms in two directions in this thesis. Therefore, we must distinguish the direction of a prism when we try to divide an individual prismatic element into three tetrahedra and moreover we define a set for collecting the resulted three tetrahedra with vertices respectively. We must notice an important principle, once we annotated the first vertex of a tetrahedron, we must annotate the remaining three vertices on a triangular plane in anticlockwise direction. We illustrate this principle in the caption of following Figure 31.

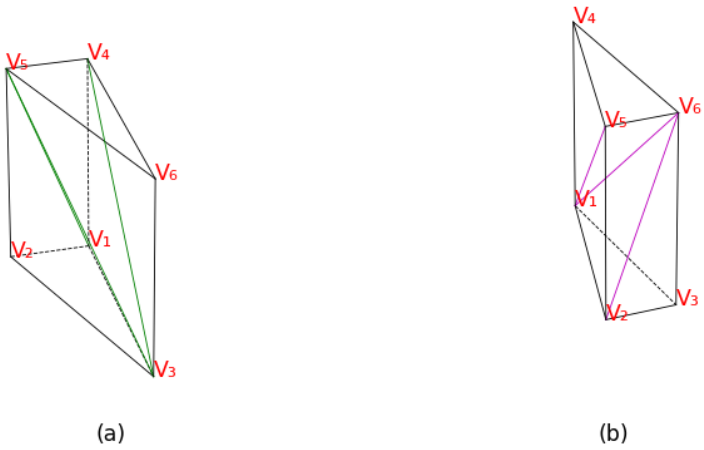

Figure 31: Recall Figure 29: (a) Dividing a standard triangulation prism 1 into three tetrahedra. The set of three resulted tetrahedra $\{V_5V_1V_2V_3, V_3V_6V_4V_5, V_4V_1V_5V_3\}$; (b) Dividing a standard triangulation prism 2 into three tetrahedra. The set of three resulted tetrahedra $\{V_6V_2V_3V_1, V_1V_4V_5V_6, V_5V_2V_6V_1\}$.

In previous Section 3.3, we have introduced how to determine the local prismatic finite element for $P : (\lambda_1, \lambda_2, z')$ in a three-dimensional space. We recall this process quickly: we firstly use Algorithm 3 to determine (λ_1, λ_2) is inside an orientation triangle from Table 2 or not on 2D $\lambda_1\lambda_2-$plane; then we determine the local subinterval in vertical direction by checking the third coordinate $z' \in [0.10, 0.14]$ or $z' \in [0.14, 0.18]$.

Therefore, if we try to determine the local tetrahedral finite element for $P : (\lambda_1, \lambda_2, z')$, first of all we would determine the local prismatic finite element and then we would check which tetrahedral finite element of local prismatic finite element $P : (\lambda_1, \lambda_2, z')$ inside is using "*Same Side Technique*" again.

For instance, with four given vertices $\{V_A, V_B, V_C, V_D\}$ we define a non-degenerate tetrahedron $V_AV_BV_CV_D$, we furthermore have an arbitrary point $P : (x, y, z)$ to test in a three-dimensional space. One method would be to transform the coordinate of $P : (x, y, z)$ into the coordinate system of tetrahedron $V_AV_BV_CV_D$, for example, setting V_A as the origin, then defining three unit vectors $\overrightarrow{V_AV_B}$, $\overrightarrow{V_AV_C}$ and $\overrightarrow{V_AV_D}$.

In this coordinate system, the coordinates of $P : (x, y, z)$ are all between 0 and 1 if $P : (x, y, z)$ is inside tetrahedron $V_AV_BV_CV_D$, but $P : (x, y, z)$ could be in anywhere in the transformed cube which is defined by the origin V_A and three unit vectors

$\overrightarrow{V_AV_B}, \overrightarrow{V_AV_C}$ and $\overrightarrow{V_AV_D}$. One way to assert that $P : (x, y, z)$ is inside tetrahedron $V_AV_BV_CV_D$ is by taking in turn as origin from set $\{V_A, V_B, V_C, V_D\}$ and other three vertices to define another new coordinate system.

The most efficient way is to transform the coordinate system only once and then reuse the *sameside* function four times. For example, we take V_A as the origin, transforming into $V_AV_BV_CV_D$ coordinates system, $P : (x, y, z)$ and V_A must lie on the same side of the $V_BV_CV_D-$plane.

```
# Cite from: https://stackoverflow.com/questions/25179693/how-to-
                                    check-whether-the-point-is-in-the
                                    -tetrahedron-or-not

# Define two functions to verify an arbitrary point P is inside
                                    tetrahedron {V_A V_B V_C V_D} or
                                    not, if yes, the second function
                                    returns True
def tetraCoord(A,B,C,D):
    v1 = B-A ; v2 = C-A ; v3 = D-A
    # mat defines an affine transform from the tetrahedron to the
                                        orthogonal system
    mat = np.concatenate((np.array((v1,v2,v3,A)).T, np.array([[0,0,
                                        0,1]])))
    # The inverse matrix does the opposite (from orthogonal system
                                        to tetrahedron)
    M1 = np.linalg.inv(mat)
    return(M1)

def pointInsideT(v1,v2,v3,v4,p):
    # Find the transform matrix from orthogonal to tetrahedron
                                        system
    M1=tetraCoord(v1,v2,v3,v4)
    # apply the transform to P
    p1 = np.append(p,1)
    newp = M1.dot(p1)
    # perform test
    return(np.all(newp>=0) and np.all(newp <=1) and sameside(v2,v3,
```

```
                                        v4,v1,p))
```

Step 5: Compute values of shape functions $(\psi_i)_{i=1:4}$

Recall Step 5 (Page 50) in Section 3.3, the basic steps for defining shape functions of tetrahedral finite elements are highly similar with defining shape functions of prismatic finite elements and moreover we will use the same short-hand notation for parameters in Python code.

```
import sympy as sy
from sympy . solvers import solve

def ShapeFunction_tetrahedral(V_A=[],V_B=[],V_C=[],V_D=[],P=[]):
# Step 1: Referring to Figure 31, we input the 3D coordinates of
                                    random point P and four vertices
    lam1_coor = P[0]
    lam2_coor = P[1]
    z_RanPt = P[2]

    global x1, x2, x3, x4
    x1 = V_A[0]
    x2 = V_B[0]
    x3 = V_C[0]
    x4 = V_D[0]
    global y1, y2, y3, y4
    y1 = V_A[1]
    y2 = V_B[1]
    y3 = V_C[1]
    y4 = V_D[1]
    global z1, z2, z3, z4
    z1 = V_A[2]
    z2 = V_B[2]
    z3 = V_C[2]
    z4 = V_D[2]
# Step 2: Declare variables x,y,z
    x,y,z = sy.symbols("x,y,z", real=True)
# Step 3: Solve the system of three linear equations and find out
```

```
                                    the numerical values of x,y,z
    solution = solve([lam1_coor-(x1+(x2-x1)*x+(x3-x1)*y+(x4-x1)*z),
                                        lam2_coor-(y1+(y2-y1)*x+(y3-
                                        y1)*y+(y4-y1)*z),z_RanPt-(z1+
                                        (z2-z1)*x+(z3-z1)*y+(z4-z1)*z
                                        )], (x, y, z), dict = True)
    key, val = zip(*solution[0].items())
# Step 4: Assign and return the values of coefficients s_i
    global s1, s2, s3, s4
    s1 = 1-val[0]-val[1]-val[2]
    s2 = val[0]
    s3 = val[1]
    s4 = val[2]

    return s1, s2, s3, s4
```

Step 6: Validation and visualization

Once we obtain four coefficients s_i, we could verify the correctness by comparing $P^{byTFE} : (\lambda_1^{byTFE}, \lambda_2^{byTFE}, z'^{byTFE})$ and $P : (\lambda_1, \lambda_2, z')$. We set

$$\begin{cases} \lambda_1^{byTFE} = \sum_{i=1}^{4} s_i \lambda_{1,i}, \\ \lambda_2^{byTFE} = \sum_{i=1}^{4} s_i \lambda_{2,i}, \\ z'^{byTFE} = \sum_{i=1}^{4} s_i z'_i. \end{cases} \tag{3.14}$$

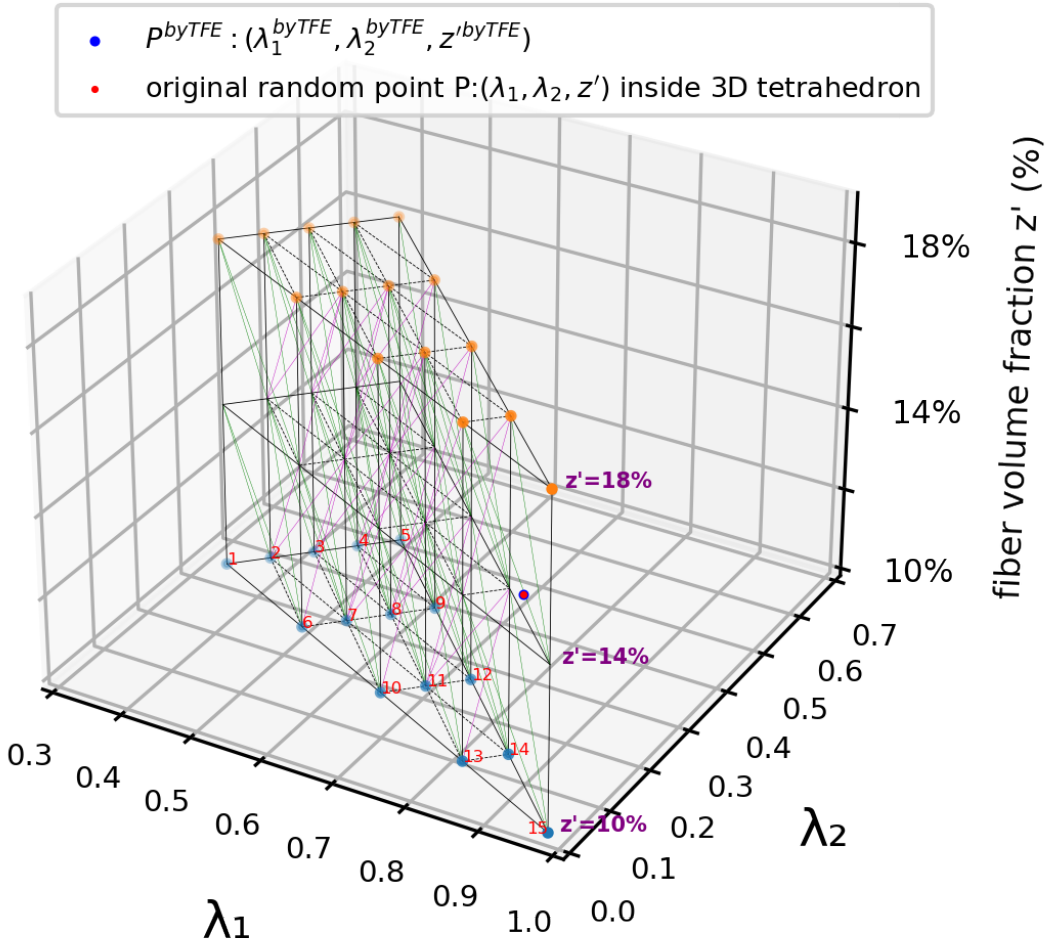

Figure 32: Once we see the overlap of P^{byTFE} and P, we claim the correctness of using tetrahedral finite element in interpolation procedure.

Chapter 4

Compare: Interpolation with Prismatic and Tetrahedral Finite Element

In the previous chapter, we have introduced prismatic and tetrahedral finite element respectively. Naturally, we would like to have a discussion about how to compare interpolation with those two finite elements from the aspects of accuracy and computing efficiency in this chapter.

From a mathematical theoretical perspective, we often use the item "convergence" when we try to evaluate the accuracy of a numerical method. However, as we have seen from the previous chapters in this thesis, we were not interested in the discussion from theoretical background so much because we have already a completely real-world data set that can be used in numerical experiments. Most importantly, we want to compare the interpolation with prismatic and tetrahedral finite element in the fixed size and shape of mesh. A refinement of finite elements won't be discussed in this thesis.

Since we completed all algorithms and computations in this thesis by Python, naturally we will care about how much time Python needs to execute the interpolation

procedure. Therefore, we use the "executing time" of Python as the evaluation of computing efficiency.

4.1 Discussion: How to Compare?

4.1.1 Comparison by Using Analytical Function

Method 1: Using 100 Random Points

Referring to Figure 33, we implement our interpolation procedure in a 3D prismatic $\lambda_1\lambda_2 z'-$space which we have seen in previous chapters with discretization as well.

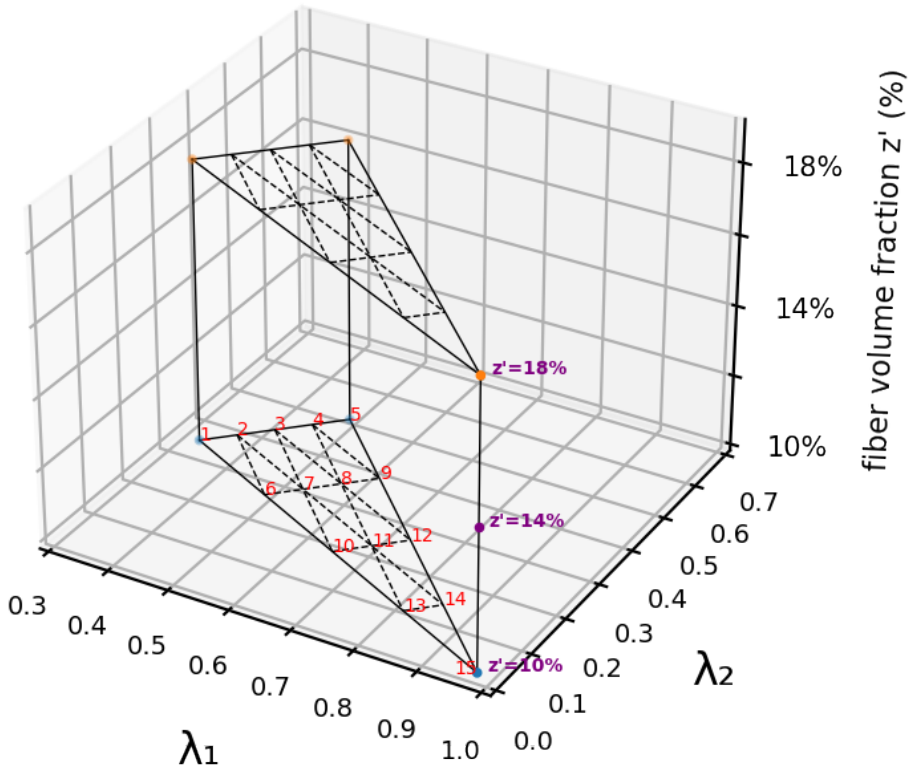

Figure 33: The 3D principal prismatic $\lambda_1\lambda_2 z'-$ space without prismatic and tetrahedral finite elements.

The computing procedure to compare accuracy of interpolation with prismatic and tetrahedral finite elements of an analytical function as following:

- Step 1: Define analytical function $f(\lambda_1, \lambda_2, z') = sin(\alpha\lambda_1)\cdot sin(\alpha\lambda_2)\cdot sin(\alpha z')+ 2$, for $\alpha = 1, 10, 100$;

- Step 2: Generate 100 random points $P_i : (\lambda_{1,i}, \lambda_{2,i}, z'_i)$ inside principal prism in Figure 33, for $i = 1 : 100$;

- Step 3: Determine the smallest local prismatic (see Figure 30) and the smallest local tetrahedral finite element (see Figure 32) for each P_i, compute the exact values of analytical function in six vertices $(\lambda_{1,i,j}, \lambda_{2,i,j}, z'_{i,j})$ of local prismatic- and in four vertices $(\lambda_{1,i,k}, \lambda_{2,i,k}, z'_{i,k})$ of local tetrahedral finite element for P_i, namely $f_{i,j} = f(\lambda_{1,i,j}, \lambda_{2,i,j}, z'_{i,j})$ and $f_{i,k} = f(\lambda_{1,i,k}, \lambda_{2,i,k}, z'_{i,k})$, for $i = 1 : 100$, $j = 1 : 6$ and $k = 1 : 4$;

- Step 4: Use prismatic shape functions (Equations (3.5) and (3.6)) and tetrahedral shape functions (Equations (3.9) and (3.10)) in local finite element of P_i, compute coefficients $s_{i,j}$ and $s_{i,k}$, for $i = 1 : 100$, $j = 1 : 6$ and $k = 1 : 4$;

- Step 5: Compute exact values $f_i = sin(\alpha\lambda_{1,i}) \cdot sin(\alpha\lambda_{2,i}) \cdot sin(\alpha z'_i) + 2$ in P_i, for $\alpha = 1, 10, 100$, $i = 1 : 100$;

- Step 6: Compute numerical approximation by interpolation in P_i using prismatic and tetrahedral finite element: $f_i^{PFE} = \sum_{j=1}^{6} s_{i,j} f_{i,j}$, $f_i^{TFE} = \sum_{k=1}^{4} s_{i,k} f_{i,k}$, for $i = 1 : 100$, $j = 1 : 6$ and $k = 1 : 4$;

- Step 7: Compute absolute interpolation error in P_i: $e_{a,i}^{PFE} = ||f_i^{PFE} - f_i||$ and $e_{a,i}^{TFE} = ||f_i^{TFE} - f_i||$, for $i = 1 : 100$;

- Step 8: Compute relative interpolation error in P_i: $e_{r,i}^{PFE} = \dfrac{e_{a,i}^{PFE}}{||f_i||}$ and $e_{r,i}^{TFE} = \dfrac{e_{a,i}^{TFE}}{||f_i||}$, for $i = 1 : 100$.

In Section 4.1.1, $||\cdot||$ denotes absolute-value norm. To have a better overview of performances of those two finite elements in interpolation procedures, we generated three databases and each database contains 100 random points $P_i : (\lambda_{1,i}, \lambda_{2,i}, z'_i)_{i=1:100}$ inside principal prism, see Figure 33. We introduce four notations:

- $R_a = \dfrac{a}{100 - a}$, a =in how many points P_i, the interpolation of analytic function $f(\lambda_1, \lambda_2, z')$ with prismatic finite element leads to a smaller absolute error, namely, $e_{a,i}^{PFE} < e_{a,i}^{TFE}$;

- $R_r = \dfrac{r}{100 - r}$, r =in how many points P_i, the interpolation of analytic function $f(\lambda_1, \lambda_2, z')$ with prismatic finite element leads to a smaller relative error, namely, $e_{r,i}^{PFE} < e_{r,i}^{TFE}$;

- t^{PFE}, how many seconds Python requires to compute all resulting parameters (see Table 3-5) of each database using prismatic finite element in interpolation of analytic function $f(\lambda_1, \lambda_2, z')$;

- t^{TFE}, how many seconds Python requires to compute all resulting parameters (see Table 3-5) of each database using tetrahedral finite element in interpolation of analytic function $f(\lambda_1, \lambda_2, z')$.

Table 3: Compare interpolation with prismatic and tetrahedral finite element of analytic function $f(\lambda_1, \lambda_2, z') = sin(\lambda_1) \cdot sin(\lambda_2) \cdot sin(z') + 2$ in $P_i : (\lambda_{1,i}, \lambda_{2,i}, z'_i)$, for $i = 1 : 100$ in each Database

(a) Compare interpolation absolute and relative errors

Resulting parameters	Database 1	Database 2	Database 3
min $e_{a,i}^{PFE}$	2.024e-06	2.335e-06	5.681e-06
min $e_{a,i}^{TFE}$	1.153e-05	2.75e-06	6.87e-06
max $e_{a,i}^{PFE}$	0.0006353	0.0006181	0.0005622
max $e_{a,i}^{TFE}$	0.0009151	0.000725	0.0007748
min $e_{r,i}^{PFE}$	1.004e-06	1.15e-06	2.799e-06
min $e_{r,i}^{TFE}$	5.731e-06	1.363e-06	3.418e-06
max $e_{r,i}^{PFE}$	0.0003125	0.0003033	0.0002775
max $e_{r,i}^{TFE}$	0.0004538	0.0003569	0.0003842
mean e_a^{PFE}	0.0002916	0.0003131	0.000276
mean e_a^{TFE}	0.0003181	0.000353	0.0003147
mean e_r^{PFE}	0.0001443	0.0001549	0.0001365
mean e_r^{TFE}	0.0001574	0.0001746	0.0001557
R_a	$\frac{66}{34}$	$\frac{69}{31}$	$\frac{70}{30}$
R_r	$\frac{66}{34}$	$\frac{69}{31}$	$\frac{70}{30}$

(b) Compare executing time (sec) in Python

	Database 1	Database 2	Database 3
t^{PFE}	24.9146	25.3564	25.0794
t^{TFE}	14.9566	14.6919	14.8621

Table 4: Compare interpolation with prismatic and tetrahedral finite element of analytic function $f(\lambda_1, \lambda_2, z') = sin(10\lambda_1) \cdot sin(10\lambda_2) \cdot sin(10z') + 2$ in $P_i : (\lambda_{1,i}, \lambda_{2,i}, z'_i)$, for $i = 1 : 100$ in each Database

(a) Compare interpolation absolute and relative errors

Resulting parameters	Database 1	Database 2	Database 3
min $e_{a,i}^{PFE}$	0.001094	0.0003383	0.0006921
min $e_{a,i}^{TFE}$	0.0007274	0.001833	0.000732
max $e_{a,i}^{PFE}$	0.3532	0.3535	0.3655
max $e_{a,i}^{TFE}$	0.3536	0.3534	0.3652
min $e_{r,i}^{PFE}$	0.0004681	0.0001858	0.0002939
min $e_{r,i}^{TFE}$	0.0004347	0.001139	0.0003109
max $e_{r,i}^{PFE}$	0.1694	0.1716	0.1729
max $e_{r,i}^{TFE}$	0.1647	0.1581	0.1736
mean e_a^{PFE}	0.1322	0.1359	0.1421
mean e_a^{TFE}	0.1309	0.1346	0.1406
mean e_r^{PFE}	0.05602	0.05887	0.06236
mean e_r^{TFE}	0.05537	0.05819	0.06169
R_a	$\frac{46}{54}$	$\frac{48}{52}$	$\frac{50}{50}$
R_r	$\frac{46}{54}$	$\frac{48}{52}$	$\frac{50}{50}$

(b) Compare executing time (sec) in Python

	Database 1	Database 2	Database 3
t^{PFE}	25.2687	26.4115	25.7659
t^{TFE}	15.1039	15.4574	15.0886

Table 5: Compare interpolation with prismatic and tetrahedral finite element of analytic function $f(\lambda_1, \lambda_2, z') = sin(100\lambda_1) \cdot sin(100\lambda_2) \cdot sin(100z') + 2$ in $P_i : (\lambda_{1,i}, \lambda_{2,i}, z'_i)$, for $i = 1 : 100$ in each Database

(a) Compare interpolation absolute and relative errors

Resulting parameters	Database 1	Database 2	Database 3
min $e_{a,i}^{PFE}$	0.002993	0.021	0.0009533
min $e_{a,i}^{TFE}$	0.01299	0.001319	0.003283
max $e_{a,i}^{PFE}$	0.9774	1.099	1.056
max $e_{a,i}^{TFE}$	1.088	1.082	1.15
min $e_{r,i}^{PFE}$	0.001607	0.01085	0.0004572
min $e_{r,i}^{TFE}$	0.006755	0.0006842	0.001582
max $e_{r,i}^{PFE}$	0.8206	0.6614	0.7269
max $e_{r,i}^{TFE}$	0.8636	0.6449	0.7691
mean e_a^{PFE}	0.2988	0.3508	0.3066
mean e_a^{TFE}	0.3194	0.351	0.3426
mean e_r^{PFE}	0.1601	0.1803	0.1597
mean e_r^{TFE}	0.1716	0.1785	0.1773
R_a	$\frac{61}{39}$	$\frac{48}{52}$	$\frac{62}{38}$
R_r	$\frac{61}{39}$	$\frac{48}{52}$	$\frac{62}{38}$

(b) Compare executing time (sec) in Python

	Database 1	Database 2	Database 3
t^{PFE}	25.7694	25.9804	26.3926
t^{TFE}	14.6795	15.4125	14.8729

Method 2: Using Centroids

Recall Figure 26, we have introduced the 2D discretization mesh of the reference orientation triangle on $\lambda_1\lambda_2-$ plane. Therefore, we could easily find out the centroid $(\lambda_1^j, \lambda_2^j)$ of each triangular finite element.

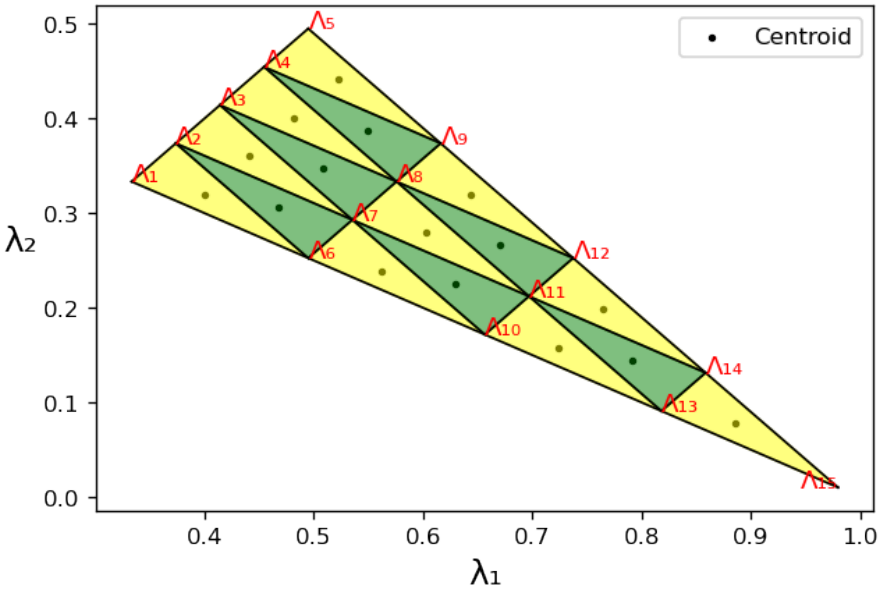

Figure 34: 16 fiber orientation triangles with centroids $(\lambda_1^j, \lambda_2^j)_{j=1:16}$.

The computing procedure and analytic function in Method 2 is the same as in Method 1 (see Page 62); but, instead of using coordinates $P_i : (\lambda_{1,i}, \lambda_{2,i}, z_i')_{i=1:100}$, we use coordinates $P_{j,k} : (\lambda_1^j, \lambda_2^j, z_k')_{j=1:16,k=1:9}$ from Step 2 to Step 8.

The idea in Method 2 is that, we use the third coordinate z_k' from collection $z' =\{10\%, 11\%, 12\%, 13\%, 14\%, 15\%, 16\%, 17\%, 18\%\}$, for each selected z_k', we further use the coordinate of a centroid from Figure 34 as the first and the second coordinate to construct a 3D coordinate. Therefore, we have 9 different values from z' collection and for each selected z_k', we have 16 centroids, namely we have totally 16×9 coordinates now inside 3D principal prism, see Figure 35, i.e. $P_{j,k} : (\lambda_1^j, \lambda_2^j, z_k')$, for $j = 1 : 16$, $k = 1 : 9$.

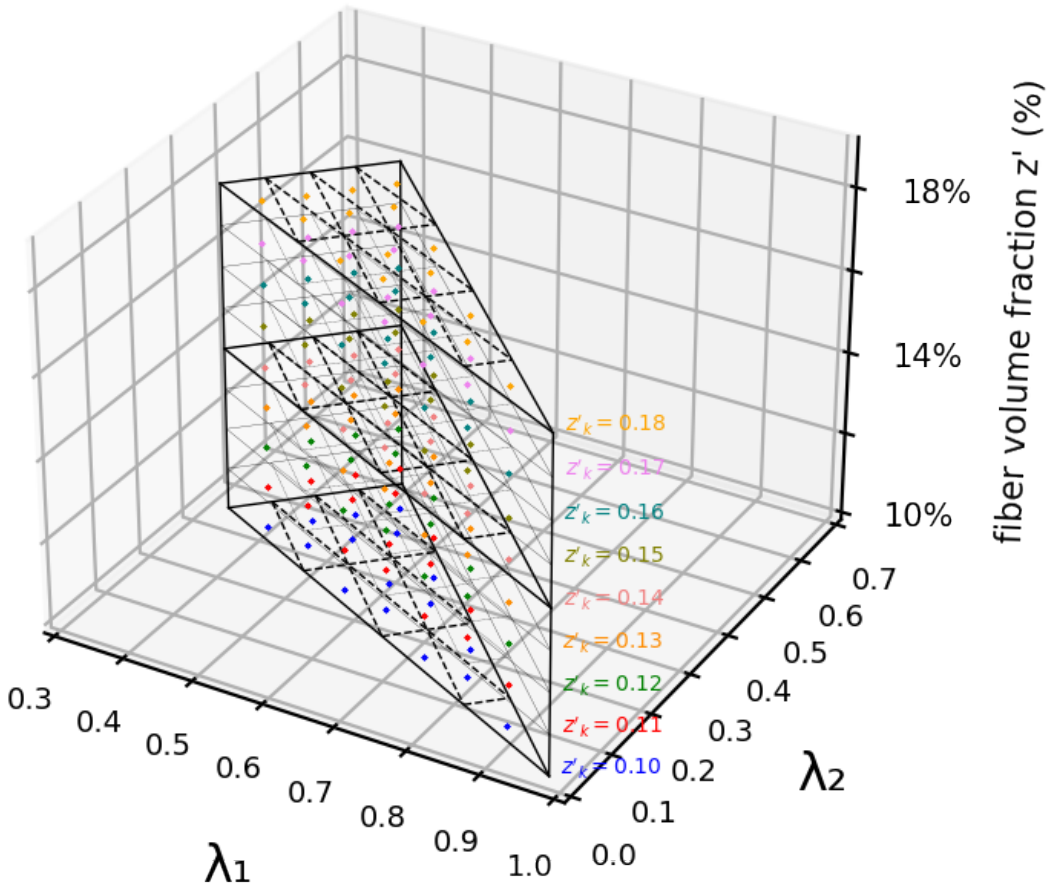

Figure 35: Visualization: $P_{j,k} : (\lambda_1^j, \lambda_2^j, z'_k)_{j=1:16,k=1:9}$ inside principal prismatic space.

In Figure 36, vertical axis $\overline{e_{r,j}}$ denotes the average relative interpolation errors with prismatic and tetrahedral finite element of analytical function $f(\lambda_1, \lambda_2, z') = sin(\alpha\lambda_1) \cdot sin(\alpha\lambda_2) \cdot sin(\alpha z') + 2$ in 16 points $P_{j,k} : (\lambda_1^j, \lambda_2^j, z'_k)$ with the same third coordinate z'_k, for $z'_k \in z'$, $k = 1 : 9$. In Table 6, we compare the executing time t^{PFE} and t^{TFE}.

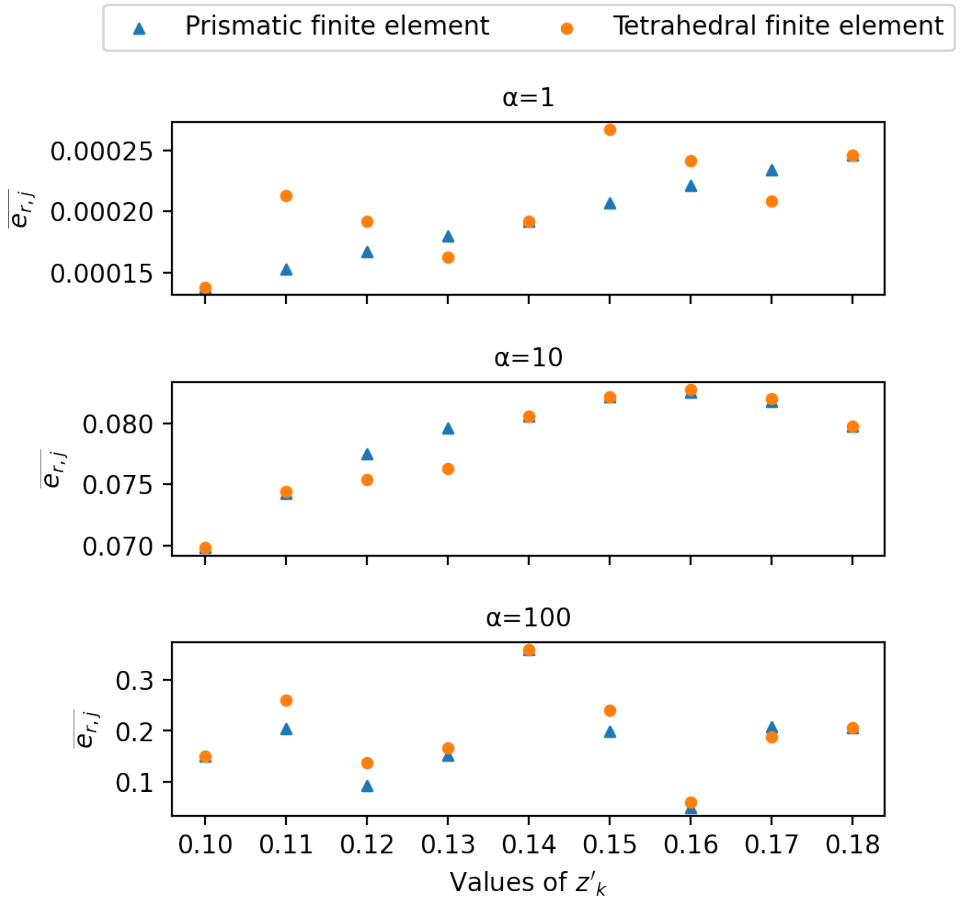

Figure 36: Compare average relative interpolation errors of analytic function $f(\lambda_1, \lambda_2, z') = sin(\alpha\lambda_1) \cdot sin(\alpha\lambda_2) \cdot sin(\alpha z') + 2$ for a fixed z'_k in coordinates $P_{j,k} : (\lambda_1^j, \lambda_2^j, z'_k)$ using prismatic and tetrahedral finite elements, for $\alpha = 1, 10, 100$, $z'_k \in z'$, $j = 1 : 16$, $k = 1 : 9$.

Table 6: Compare executing time (sec) that computing relative interpolation errors of analytic function $f(\lambda_1, \lambda_2, z') = sin(\alpha\lambda_1) \cdot sin(\alpha\lambda_2) \cdot sin(\alpha z') + 2$ in coordinates $P_{j,k} : (\lambda_1^j, \lambda_2^j, z'_k)$ using prismatic and tetrahedral finite elements in Python, for $\alpha = 1, 10, 100$, $z'_k \in z'$, $j = 1 : 16$, $k = 1 : 9$.

	$\alpha = 1$	$\alpha = 10$	$\alpha = 100$
t^{PFE}	28.9305	29.8196	30.0026
t^{TFE}	17.5982	18.4397	17.5903

4.1.2 Comparison by Using Stiffness Tensor in Voigt Notation

From Fraunhofer ITWM, we obtain 12 random local fiber orientation tensors $A_i = (A^i_{xx}, A^i_{yy}, A^i_{zz}, A^i_{yz}, A^i_{xz}, A^i_{xy})$, each A_i is equipped with nine different fiber volume fraction z'_k from collect $z' = \{10\%, 11\%, 12\%, 13\%, 14\%, 15\%, 16\%, 17\%, 18\%\}$. Therefore, we have in total 12×9 stiffness tensors $C_{i,k}$ in Voigt notation that correspond to fiber orientation tensor A_i with equipped fiber volume fraction z'_k, for $i = 1:12$, $k = 1:9$. The stiffness tensors $C_{i,k}$ are generated by Fraunhofer software FeelMath **[fTuWI13]**. The relevant numerical methods implemented in FeelMath can be found in **[SOK16]**, **[KFS16]** and **[GSK19]**. The geometries were generated for FeelMath calculation using the algorithm from the paper **[Sch17]**.

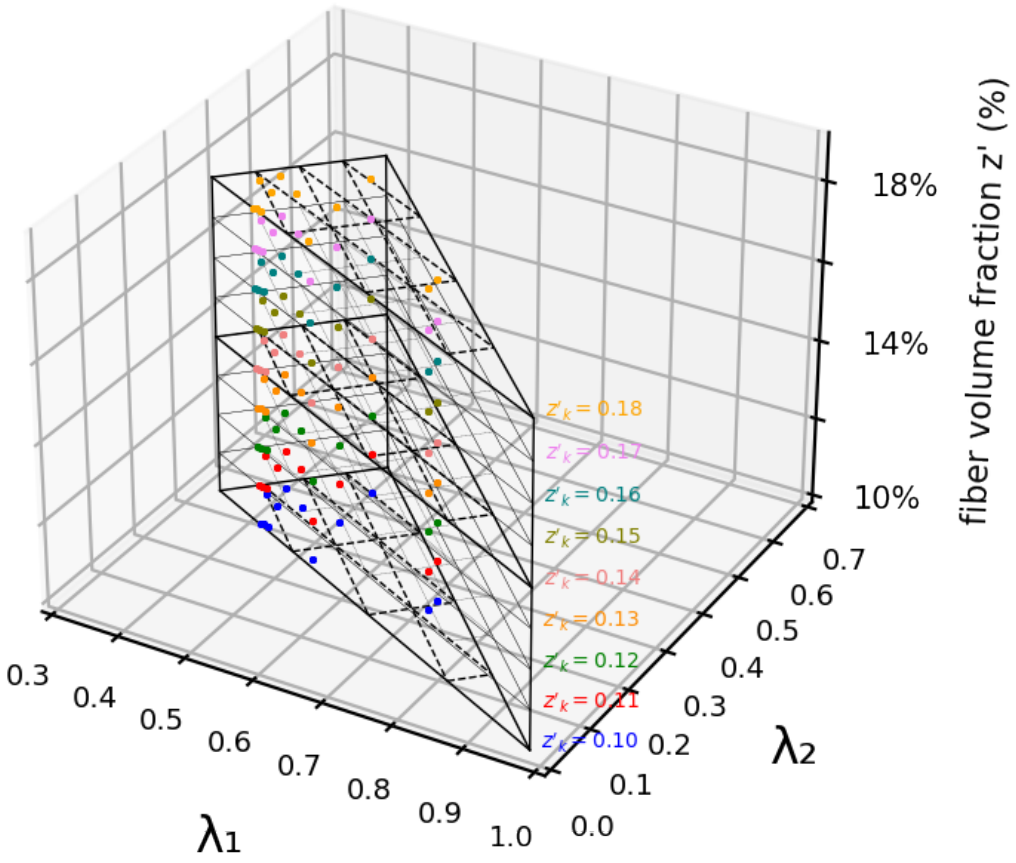

Figure 37: Visualization: $P_{i,k} : (\lambda_{1,i}, \lambda_{1,i}, z'_k)_{i=1:12,k=1:9}$ inside principal prismatic space.

The interpolation procedure of given stiffness tensors $C_{i,k}$ with prismatic and tetrahedral finite element follows:

- Step 1: Rewrite given fiber orientation tensor $A_i = (A^i_{xx}, A^i_{yy}, A^i_{zz}, A^i_{yz}, A^i_{xz}, A^i_{xy})$ to

$$A_i = \begin{bmatrix} A^i_{xx} & A^i_{xy} & A^i_{xz} \\ A^i_{xy} & A^i_{yy} & A^i_{yz} \\ A^i_{xz} & A^i_{yz} & A^i_{zz} \end{bmatrix} \in \mathbb{R}^{3\times 3}, \quad i = 1:12;$$

- Step 2: Decompose spectrally $A_i \in \mathbb{R}^{3\times 3}$ using Equation (1.2)

$$A_i = R_i \cdot \Lambda_i \cdot R_i^T,$$

where $R_i, R_i^T \in \mathbb{R}^{3\times 3}$ and $\Lambda_i = diag(\lambda_{1,i}, \lambda_{2,i}, \lambda_{3,i})$ for $\lambda_{1,i} \geq \lambda_{2,i} \geq \lambda_{3,i}$; Construct coordinates $P_{i,k} : (\lambda_{1,i}, \lambda_{1,i}, z'_k)$, for $z'_k \in z'$, $i = 1:12$, $k = 1:9$;

- Step 3: Determine the smallest local prismatic finite element M (see Figure 30) and the smallest tetrahedral finite element N (see Figure 32) of each point $P_{i,k}$, for $i = 1:12$, $k = 1:9$;

- Step 4: Use prismatic shape functions ((Equations (3.5) and (3.6)) in local element M and tetrahedral shape functions (Equations (3.9) and (3.10)) in local element N for each point $P_{i,k}$, compute coefficients $s_{i,k,m}$ and $s_{i,k,n}$, for $i = 1:12$, $k = 1:9$, $m = 1:6$, $n = 1:4$;

- Step 5: Compute the interpolated stiffness tensor in Voigt notation in $P_{i,k}$ by prismatic and tetrahedral finite element, $C^{IP}_{i,k} = \sum_{m=1}^{6} s_{i,k,m} C_{i,k,m}$ and $C^{IT}_{i,k} = \sum_{n=1}^{4} s_{i,k,n} C_{i,k,n}$, here $C_{i,k,m}$ and $C_{i,k,n}$ denote the stiffness tensors in vertices of element M and element N, for $i = 1:12$, $k = 1:9$, $m = 1:6$, $n = 1:4$;

- Step 6: Compute the numerical approximation of stiffness tensor in Voigt notation in $P_{i,k}$ by orthogonal transformation, $C^{PFE}_{i,k} = R_i \cdot C^{IP}_{i,k} \cdot R_i^T$ and $C^{TFE}_{i,k} = R_i \cdot C^{IT}_{i,k} \cdot R_i^T$, where $R_i, R_i^T \in \mathbb{R}^{6\times 6}$ (see Remark 4.1.1.), for $i = 1:12$, $k = 1:9$;

- Step 7: Referring to **[ABB$^+$99]**, compute the relative interpolation errors in matrices representation in $P_{i,k}$: $E^{PFE}_{i,k} = \frac{||C^{PFE}_{i,k} - C_{i,k}||}{||C_{i,k}||}$ and $E^{TFE}_{i,k} = \frac{||C^{TFE}_{i,k} - C_{i,k}||}{||C_{i,k}||}$, for $i = 1:12$, $k = 1:9$.

In this Section 4.1.2, $||\cdot||$ denotes norms that are used to evaluate a matrix. We use here Frobenius norm $||\cdot||_F$, infinity norm $||\cdot||_\infty$ and one norm $||\cdot||_1$ to compare the relative errors.

Remark 4.1.1. ([**[Aul73]**, Volume I, Chapter 3]) In Step 2, we have matrices R_i and R_i^T from the eigendecomposition of A_i, R_i is the matrix of sorted eigenvectors of A_i based on Voigt notation order $A_i = (A^i_{xx}, A^i_{yy}, A^i_{zz}, A^i_{yz}, A^i_{xz}, A^i_{xy})$. In Step 2, the 3×3 matrix R_i in orthonormal coordinates is given by

$$R_i = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{bmatrix}.$$

The stiffness tensor $C_{i,k}$ maintains symmetry if it does not change when subjected to a corresponding orthogonal transformation. In other words, the elastic properties of a material do not change under orthogonal transformation. Therefore, in Step 6, the transformation matrix for stiffness tensor in Voigt notation can be expressed as a 6×6 matrix R_i via

$$\begin{bmatrix} R^2_{xx} & R^2_{xy} & R^2_{xz} & 2R_{xy}R_{xz} & 2R_{xx}R_{xz} & 2R_{xx}R_{xy} \\ R^2_{yx} & R^2_{yy} & R^2_{yz} & 2R_{yy}R_{yz} & 2R_{yx}R_{yz} & 2R_{yx}R_{yy} \\ R^2_{zx} & R^2_{zy} & R^2_{zz} & 2R_{zy}R_{zz} & 2R_{zx}R_{zz} & 2R_{zx}R_{zy} \\ R_{yx}R_{zx} & R_{yy}R_{zy} & R_{yz}R_{zz} & R_{yy}R_{zz} + R_{yz}R_{zy} & R_{yx}R_{zz} + R_{yz}R_{zx} & R_{yx}R_{zy} + R_{yy}R_{zx} \\ R_{xx}R_{zx} & R_{xy}R_{zy} & R_{xz}R_{zz} & R_{xy}R_{zz} + R_{xz}R_{zy} & R_{xx}R_{zz} + R_{xz}R_{zx} & R_{xx}R_{zy} + R_{xy}R_{zx} \\ R_{xx}R_{yx} & R_{xy}R_{yy} & R_{xz}R_{yz} & R_{xy}R_{yz} + R_{xz}R_{yy} & R_{xx}R_{yz} + R_{xz}R_{yx} & R_{xx}R_{yy} + R_{xy}R_{yx} \end{bmatrix}.$$

In Figure 38, vertical axis $\overline{||E_{r,i}||}$ in different norms denotes the average relative interpolation errors of stiffness tensors $C_{i,k}$ with prismatic and tetrahedral finite element in 12 points $P_{i,k} : (\lambda_{1,i}, \lambda_{2,i}, z'_k)$ with the same coordinate z'_k, for $z'_k \in z'$, $k = 1:9$. In Table 7, we compare the executing time t^{PFE} and t^{TFE}.

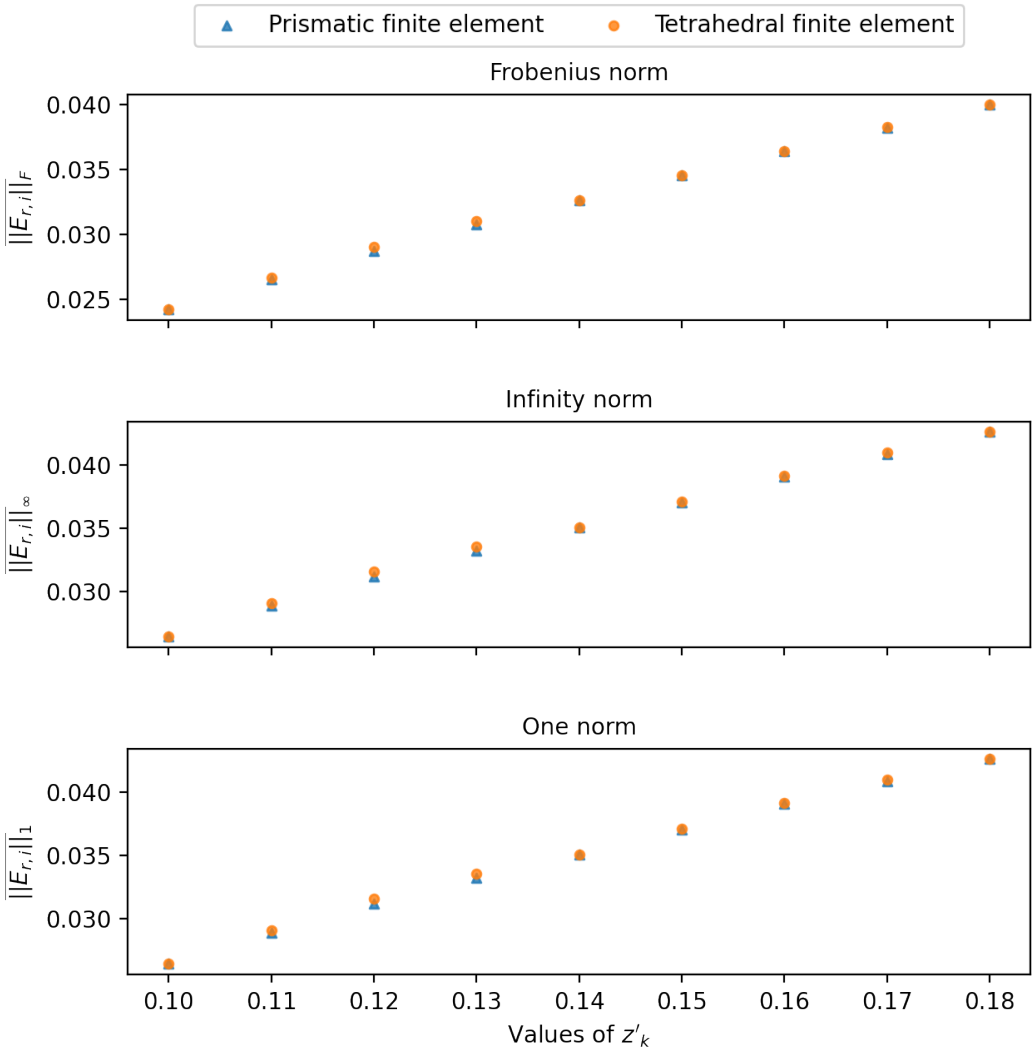

Figure 38: Compare average relative interpolation errors of stiffness tensors $C_{i,k}$ for a fixed z'_k in coordinates $P_{i,k} : (\lambda_{1,i}, \lambda_{1,i}, z'_k)_{i=1:12,k=1:9}$ using prismatic and tetrahedral finite element, for $z'_k \in z'$, $i = 1 : 12$, $k = 1 : 9$.

Table 7: Compare executing time (sec) that computing relative interpolation errors of stiffness tensors $C_{i,k}$ in coordinates $P_{i,k} : (\lambda_{1,i}, \lambda_{1,i}, z'_k)_{i=1:12,k=1:9}$ using prismatic and tetrahedral finite element in Python, for $z'_k \in z'$, $i = 1 : 12$, $k = 1 : 9$.

Executing time (sec)	
t^{PFE}	27.2329
t^{TFE}	19.4574

4.2 Conclusions

The research in this thesis extends the existing results from **[KSO$^+$18]** to also take the dependence of fiber volume fraction into account. For the case of linear elasticity, we developed an interpolation procedure with prismatic and tetrahedral finite elements for computing the numerical approximation of a stiffness tensor corresponding to a given fiber orientation tensor A and fiber volume fraction $z' \in [0.10, 0.18]$ of composite PBT (cf. Figure 33). Using interpolation procedure and three-dimensional finite elements created in this thesis, we could compute flexibly the approximation of a stiffness tensor with only a few seconds run time in Python.

Recalling all the numerical results that are presented in Chapter 4, it seems that an obvious conclusion is, interpolation with tetrahedral finite element has an evident advantage of run time in both numerical computation of analytic function $f(\lambda_1, \lambda_2, z') = sin(\alpha\lambda_1) \cdot sin(\alpha\lambda_2) \cdot sin(\alpha z') + 2$ for $\alpha = 1, 10, 100$ and given stiffness tensor $C_{i,k}$ in Voigt notation because a tetrahedral finite element has less vertices and simpler shape functions.

From Table 3-5, we see that with an increasing coefficient α of analytical function $f(\lambda_1, \lambda_2, z')$, for example $\alpha = 10, 100$, the accuracy performance of interpolation in 100 random points with both finite elements are not so satisfactory, especially for $\alpha = 100$. But, for a small coefficient $\alpha = 1$, the interpolation of analytic function $f(\lambda_1, \lambda_2, z')$ with prismatic finite element has a very good accuracy performance for most of the points from 100 random points.

Recall Figure 36 and Figure 38, there is no difference of accuracy when we have the third coordinate $z'_k = 10\%, 14\%, 18\%$. With those three z'_k values, the interpolation of analytic function $f(\lambda_1, \lambda_2, z')$ or interpolation of stiffness tensor $C_{i,k}$ inside the 3D space will be simplified to the interpolation in the 2D plane. Hence, for the third coordinate $z'_k = 10\%, 14\%, 18\%$, the faster interpolation with tetrahedral finite element will be the optimal choice.

When we have a look at Figure 36 and only take accuracy into consideration, for a

smaller coefficient $\alpha = 1$ of analytic function $f(\lambda_1, \lambda_2, z')$, it seems that the interpolation in centroids with both finite elements are satisfactory. However, with an increasing α, the average relative interpolation error of analytic function $f(\lambda_1, \lambda_2, z')$ also increases dramatically. If we set the limit of accuracy to a value of 5% (or 0.05), the interpolation of analytic function $f(\lambda_1, \lambda_2, z')$ in centroids with both finite elements are dissatisfactory.

In Section 4.1.2, we interpolate and compare stiffness matrices obtained by numerical homogenization with the tool FeelMath. In contrast to Figure 36, we see a regular increment of average relative interpolation error of given stiffness tensor $C_{i,k}$ with both finite elements in Figure 38 with the increasing fiber volume fraction z'_k. In this case, with the accuracy limit of 5% (or 0.05), we could claim that interpolation of given stiffness tensors $C_{i,k}$ with both finite elements has very good performance.

The link to the executable Python projects which created for this thesis will be given in Appendix. We encourage our readers to use our codes to find out the best numerical solution for their interpolation procedures.

Appendix

As we see the Chapter 3 and Chapter 4 in this thesis, we depend on Python very much to implement the numerical computation and analysis. I created all the Python scripts that are used in this master thesis by myself. Of course, some amazing functions in my scripts are quoted from public online resources, and I have listed already in the README file.

The link to access my Python scripts is
`https://drive.google.com/file/d/1PLOgwRYWlRGvWWErK-zjN-dz3aW3AiA5/view?usp=sharing`

The Python scripts can be only used for personal study and research, any commercial usage is not allowed.

Have fun with coding!

Bibliography

[ABB+99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *Lapack users' guide*, third ed., SIAM, Philadelphia, Pennsylvania, USA, 1999 (en).

[ADH+14] James Adler, Luis Dorfmann, D. Han, S. MacLachlan, and Christopher Paetsch, *Mathematical and computational models of incompressible materials subject to shear*, IMA Journal of Applied Mathematics **79** (2014), 889–914 (en).

[ADS+19] Heiko Andrä, Dascha Dobrovolskij, Katja Schladitz, Sarah Staub, and Ralf Müller, *Multi-scale simulation of composite materials: Results from the project musiko*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2019 (en).

[Aul73] B.A. Auld, *Acoustic fields and waves in solids*, A Wiley-Interscience publication, Wiley, 1973 (en).

[BBG+21] J. Becker, F. Biebl, E. Glatt, L. Cheng, A. Grießer, M. Groß, S. Linden, D. Mosbach, C. Wagner, A. Weber, and R. Westerteiger, *Geodict (release 2022) [simulation software]*, 2021, Available at `https://www.math2market.de/`.

[Bra13] Dietrich Braess, *Finite Elemente*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013 (German).

[Cam10] F.C. Campbell, *Structural composite materials*, ASM International, 2010 (en).

[Cha13] Eduardo W. V. Chaves, *Notes on continuum mechanics*, Springer Netherlands, Dordrecht, 2013 (en).

[CMS17] Helmut Clemens, Svea Mayer, and Christina Scheu, *Neutrons and synchrotron radiation in engineering materials science*, John Wiley & Sons, Ltd, 2017 (en).

[COO09] MARIANA COOK, *Mathematicians: An outer view of the inner world*, Princeton University Press, 2009 (en).

[DLVC99] Julien Dompierre, Paul Labbé, Marie-Gabrielle Vallet, and Ricardo Camarero, *How to subdivide pyramids, prisms, and hexahedra into tetrahedra.*, IMR, January 1999 (en).

[DO10] Roberto Lacerda De Orio, *Electromigration modeling and simulation*, na, 2010 (en).

[EG04] Alexandre Ern and Jean-Luc Guermond, *Theory and practice of finite elements*, Springer New York, New York, NY, 2004 (en).

[Esp21] D. Nicolas Espinoza, *Introduction to energy geomechanics*, na, February 2021 (en).

[fTuWI13] Fraunhofer-Institut für Techno-und Wirtschaftsmathematik ITWM, *Feelmath*, 2013, Available at `https://www.itwm.fraunhofer.de/en/departments/sms/products-services/feelmath.html`.

[Giu16] Victor Giurgiutiu, *Structural health monitoring of aerospace composites*, Academic Press, Oxford, 2016 (en).

[GSK19] Hannes Grimm-Strele and Matthias Kabel, *Runtime optimization of a memory efficient cg solver for fft-based homogenization: implementation details and scaling results for linear elasticity*, Computational Mechanics **64** (2019), 1339–1345 (en).

[GSSA20] P. Gumbsch, S. Sommer, Anita Schöbel, and Heiko Andrä, *Effiziente charakterisierung und modellierung des anisotropen versagensverhaltens von lft für crashsimulation*, FAT, 2020 (German).

[Irg19] Fridtjov Irgens, *Tensor analysis*, Springer International Publishing, Cham, 2019 (en).

[KFS16] Matthias Kabel, Sascha Fliegener, and Matti Schneider, *Mixed boundary conditions for fft-based homogenization at finite strains*, Computational Mechanics **57** (2016), 193–210 (en).

[KSO+18] Jonathan Köbler, Matti Schneider, Felix Ospald, Heiko Andrae, and Ralf Müller, *Fiber orientation interpolation for the multiscale analysis of short fiber reinforced composite parts*, Computational Mechanics **61** (2018), 729–750 (en).

[LQT17] Zhilin Li, Zhonghua Qiao, and Tao Tang, *Numerical solution of differential equations: Introduction to finite difference and finite element methods*, Cambridge University Press, 2017 (en).

[PH08] Federico Ponchio and Kai Hormann, *Interactive rendering of dynamic geometry*, IEEE transactions on visualization and computer graphics **14** (2008), 914–25 (en).

[RBD03] Michel Rappaz, Michel Bellet, and Michel Deville, *Numerical modeling in materials science and engineering*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003 (en).

[Sch17] Matti Schneider, *The sequential addition and migration method to generate representative volume elements for the homogenization of short fiber reinforced plastics*, Computational Mechanics **59** (2017), 247–263 (en).

[SCT+20] Tristan Seidlhofer, Caterina Czibula, Christian Teichert, Ulrich Hirn, and Manfred Ulz, *A compressible plasticity model for pulp fibers under transverse load*, Mechanics of Materials **153** (2020) (en).

[Sim13] Bernd Simeon, *Computational flexible multibody dynamics*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013 (en).

[SOK16] Matti Schneider, Felix Ospald, and Matthias Kabel, *Computational homogenization of elasticity on a staggered grid*, International Journal for Numerical Methods in Engineering **105** (2016), no. 9, 693–720 (en).

[TAMC11] Vuong-Dieu Trinh, Farid Abed-Meraim, and Alain Combescure, *A new assumed strain solid-shell formulation "SHB6" for the six-node prismatic finite element*, Journal of Mechanical Science and Technology **25** (2011), no. 9, 2345–2364 (en).

[TH21] Ngoc-Thien Tran and Nga Hong, *Investigation of the effect of polycarbonate rate on mechanical properties of polybutylene terephthalate/polycarbonate blends*, International Journal of Polymer Science **2021** (2021), 1–7 (en).

Declaration

I hereby affirm that I have independently written the master's thesis with the title "Computing the Stiffness Tensor of Composite Materials by Interpolation Procedures" and have not used any sources and aids other than those indicated, and that I marked quotations accordingly.

Kaiserslautern, July 28, 2022 Name: Lin, ChuHe