Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Katinka Hårdvik Engen

Machine-learning approach to design fatigue-resistant structure inspired by Pogonias cromis

Pogonias cromis has one of the highest biting forces per weight encountered in Nature. Recent study has reported the unusual porous structure of its lower jaw bone that can withstand high cyclic loads. However, the design principles of this porous structure are still unknown. In this investigation, a novel machine-learning approach will be exploited to understand the design principles and to design fatigue-resistant structures via numerical simulations and machine learning.

Master's thesis in Produktutvikling og produksjon
Supervisor: Chao Gao
Co-supervisor: Filippo Berto
June 2022

NTNU
Norwegian University of
Science and Technology

Katinka Hårdvik Engen

Machine-learning approach to design fatigue-resistant structure inspired by Pogonias cromis

Pogonias cromis has one of the highest biting forces per weight encountered in Nature. Recent study has reported the unusual porous structure of its lower jaw bone that can withstand high cyclic loads. However, the design principles of this porous structure are still unknown. In this investigation, a novel machine-learning approach will be exploited to understand the design principles and to design fatigue-resistant structures via numerical simulations and machine learning.

Master's thesis in Produktutvikling og produksjon
Supervisor: Chao Gao
Co-supervisor: Filippo Berto
June 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Sammendrag

Atlanterhavsfisken Pagonias cromis har den høyeste bitekraften per vekt [1], men det nedre svelgkjevebenet som er ansvarlig for å knuse bløtdyr og skalldyr er relativt porøst [2] sammenlignet med kortikalt bein funnet i pattedyr [3] . I denne oppgaven blir mikrostrukturen til dette beinet utforsket for om det inneholder egenskaper som gjør det spesielt egnet til å motstå utmattelsesskader. Det vil også bli undersøkt om dette problemet effektivt kan automatiseres og modelleres ved hjelp av maskinlæringsverktøy og optimaliseringsmetoden 'differential evoulution'.

Resultatene viser at bruk av nevrale nettverk for å undersøke variabel signifikans i strukturen er mulig med riktige kalibreringer, og modell, og en effektiv fremgangsmåte. Optimalisering ved bruk av differensiell evolusjon var svært effektiv kombinert med regresjonsmodell estimert med nevralt nettverk. På grunn av de begrensede tilgjengelige beregningsressursene var den ikke like robust kombinert med Abaqus, siden den ikke håndterte avbrudd uten å måtte starte på nytt. Studien viste også at faktorene som beskriver mikrostrukturen i betydelig grad påvirker strukturenes evne til å motstå tretthetsvrudd. For fast spenning må det kjøres ytterligere simuleringer for å gi et avgjørende svar på hva som er den optimale strukturen.

Abstract

The Black Drum fish has the highest biting force per weight [1], yet its lower pharyngeal jaw bone responsible for crushing the mollusks and shellfish is relatively porous [2] compared to cortical bone found in mammalian bone [3]. In this thesis, the microstructure of this bone will be investigated to find whether or not it contains properties making it especially suitable to withstand cyclic fatigue damage. It will also be investigated whether this problem can be effectively automized and modeled using machine learning tools and optimization differential evolution.

The results show that using neural networks to examine variable significance in the structure is an efficient method, possibly with the correct calibrations and inputs and outputs. Optimization using differential evolution was very efficient combined with a regression model obtained with a neural network. Due to the limited available computational resources, it was not as robust in combination with Abaqus, as it did not handle interruptions without having to restart. The study also showed that the micro structure's factors significantly affect the structures' ability to withstand fatigue. For fixed stress, further simulations must be run to give a conclusive answer to what is the optimal structure.

Preface

First, I would like to thank my supervisor Chao Gao and doctoral research fellow Marco Maurizi who has been exceedingly helpful, available for questions, reminded me of my goal when I have wandered off, and helped when stuck. The advice has been invaluable, and the criticism constructive.

Second, I would like to thank my boyfriend, Max, who's encouraged me when I've been unconstructively discouraged and always there when I needed someone to discuss my problems with. Your interest has been a massive contribution to this thesis, and your willingness to help me say my thoughts out load invaluable.

My mom, of course, deserves massive thanks. You are the best, I love you, and I would not have been here without you!

I would also like to thank my friends and classmates for all the information exchange on the writing, but most importantly, all the laughs, conversations, and parties. Lifting one's spirit has never seemed quite as important.

This has been a ride, and I've learned a lot of interesting things, both professionally and personally. The stress, however, is horrible. I'm never doing this again.

I hope you enjoy it!

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The subjects of this thesis are bio-inspired structures, fatigue, and the potential of machine learning and optimization in the design of structures. The bio-inspired structure is the jaw-bone of the Black Drum fish, which is subject to very high cyclic loading [2]. The microstructure of this bone is different from that seen in mammals, and the possibility of it containing some success factors for withstanding high cyclic loadings will be investigated. Machine learning and optimization will be tested as possible data-driven methods for better understanding the structure's behavior, along with some conventional Design of Experiments methods.

The Theory chapter presents the relevant theory needed to understand the methods and results. This includes the background of the jaw bone and how it differs from mammalian bone, an introduction to how to simulate fatigue using finite elements in Abaqus, how to simulate repetitive structures using finite element analysis and periodic boundary conditions, and how to interpret the numerical results. This chapter will also introduce what machine learning and deep learning using neural networks are and how this can be used to solve multivariate problems that are not linearly separable. Optimization using differential evolution will be introduced, and the mathematical tool, Design of Experiments (DoE), will be used to present results.

The methods used to create the numerical model, the simplified structure, the multi-layer perceptron (MLP) model, and the optimization script are introduced in the Methods chapter. The method used to conduct numerical experiments is also presented.

In the Results chapter, the results obtained regarding the MLP-model and the optimization of the structure using the different approaches presented in the methods chapter are introduced.

In the Discussion chapter, the results will be discussed. Here the focus will be on the relevance of the findings and experiences made during the thesis. The resulting optimal factors will be discussed. In this chapter, the use of machine learning and optimization and its potential will also be addressed, as will the sources for error and possible further works.

In the Appendix the code used to create the finite element model with PBC as described in the methods section; interpret the results; create an MLP-model and optimize factors using differential evolution is added, so the reader can review it, and perhaps find some use for it. Some additional plots and tables are also put in the Appendix. These were considered as not integral to the flow of the thesis but potentially interesting to the reader. The Appendix also contains the project thesis of the author, which includes some work used in the thesis.

Chapter 2

Theory

This section will introduce the theory relevant to the methods used and the results represented in this thesis. This includes background information on the lower pharyngeal jaw bone of the Black drum, which is the basis for this thesis; background information on bone in general; theory on fatigue and fatigue simulation using finite element method; damage models relevant for fatigue simulation using finite element; mathematical tools to interpret and understand multivariate problems; machine learning using neural networks to examine variable significance in multivariate models and optimization using differential evolution as a solution for optimization of multivariate problems.

2.1 Bone - an introduction to the material

2.1.1 The Black Drum Jaw Bone

Due to its diet consisting primarily of shellfish and ammonites, the lower pharyngeal jaw bone of Black Drum is subject to high cyclic loading. Further examination of the bone has shown that its microstructure is unlike that we are familiar with from the mammalian bone [2].

2.1.2 Comparison of different kinds of bone

Mammalian bone is the bone most extensively described in modern science [4]. Its structures share several common traits across species. Bones of mammals are built of two kinds of bone structures: cortical and trabecular bone. The two types have significantly different properties and purposes, described in the following sections.

Cortical Bone

The cortical bone consists of about 10% soft tissue and makes up 80 % of the skeletal mass. It forms the outer layer of the bone and is often more prominent in weight-bearing areas such as the femur [5]. An illustration of the difference between trabecular bone and cortical bone in humans is provided in figure 2.1. For comparison, the cortical bone of a bovine is depicted in figure 2.2.

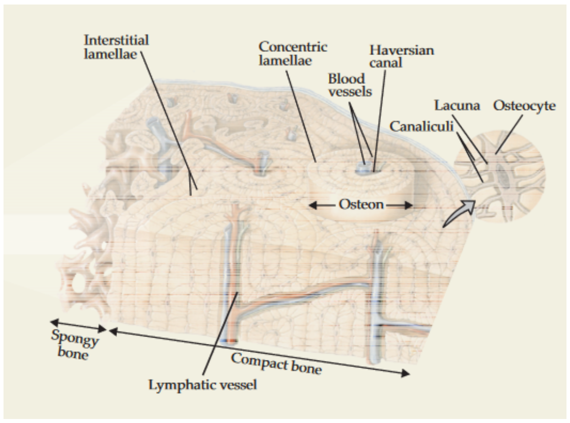


Figure 2.1: Macroscopic view of cortical and trabecular (here: spongy) bone. Figure is from G. J. Tortora [6]

As can be seen in both these figures, the cortical bone is quite dense and consists of unidirectional fibers, or lamellae. It does vary across species, but attributes like density and regularity are seen in all mammals [3].

Loading direction

The unidirectional fibers and the high density of the cortical bone makes it suitable for load-bearing in the fiber direction. Its placement along the edges in the macrostructure of the bone has the additional effect of bracing the bone against bending forces.

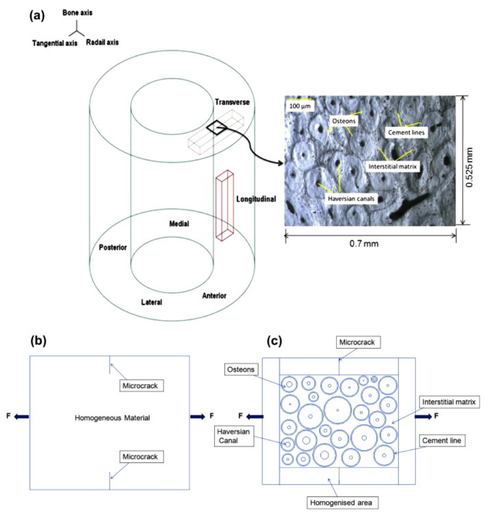


Figure 2.2: The bovine cortical bone. (a) Light-microscopy micrograph and its position in transverse-radial cross-section of osteonal bovine cortical bone tissue, (b) schematic illustration of homogeneous model, and (c) schematic illustration of microstructural model. Figure is from A. A. Abdel-Wahab, A. R. Maligno, and V. V. Silberschmid [7]

Trabecular Bone

The trabecular bone consists of tiny rods around 100 μm thick, placed without the regular order associated with the cortical bone. As can be seen in the figure 2.1 it has a sponge-like appearance, with voids as large as 1 mm wide [8]. These voids are filled with soft tissue (bone marrow), which makes up about 75 % of its volume [5]. Trabecular bone is also referred to as spongy or cancellous bone.

As with the cortical bone, the microstructure of trabecular bone varies between mammalian species. For example, the properties of the trabecular bone belonging to the ostrich are similar to that of humans, but both are quite different from the equine trabecular bone [9].

Loading direction

The relatively randomized microstructure of the trabecular bone makes it more suitable to withstand multidirectional loading than, e.g., the cortical bone. Boyle and Kim (2011) [10] note that trabecular bone has adapted to the dominant loading direction by directing more rods to align with it.

Trabecular bone is found in the center of long bones, where it supports the cortical exterior. It is also thought that this is the reason it is more dominant near the joints, as the loading direction here will shift. Because around 75 % of the volume in the trabecular bone is soft tissue, it cannot withstand as much pressure as cortical bone. This might cause the bone to increase in size near the joints.

Fishbone

As mentioned earlier, mammalian bone is the kind most investigated and described in science. The bone of fish is not as well studied. In 2015 A. Atkins et al. noted

> While the structure of mammalian bones is therefore reasonably well studied in three dimensions (...) similar data with regard to fish bone are lacking. In particular, the fibrillar arrangement in fish bone lamellae is unknown, as, indeed, is whether their layered structure consists of lamellar units at all [4].

Fishbone structures can be roughly divided into two types: cellular and acellular [4][3]. Acellular bone, also known as anosteocytic (osteocytes are the cells that make up most of the human cortical bone), was previously thought to be featureless. An investigation by Atkins et al. (2015) found it to be just slightly less ordered than cortical bone. It reportedly consists of thin fibers which are 1-2 μm thick, compared to 2-7 μm for cortical bone. It was also found to be tougher than mammalian bone and more like bone found in the antlers of deer [4].

The last section of the fish was provided to address the issue of specious variation. It seems the two-bone system found in mammals might not be directly comparable with fish. The skeletal of fish' is complicated, and the fish-bone described are one type from one fish.

2.2 The Black Drum Lower pharyngeal jaw bone

From the last section it is clear that the mammalian bone is relatively well understood by science, compared to other types of bone like the fish bone. The Black Drum has the highest biting force per weight [1], yet it's lower pharyngeal jaw bone is relatively porous compared to cortical bone [2]. In this section the know features of this structure is described.

2.2.1 Bone structure

The lower pharyngeal jaw (LPJ) consists of two halves of a dental plate, each half supported by cone-shaped struts. The two halves meet in the middle of the jaw at a suture. The dental plate is covered with molars, which become larger the closer to the plate's center. The two cones will be called struts. These struts are at their largest, where they meet the dental plates and thin out to where they are connected to the cleithrum. The cleithrum is the bone transferring motion, much like the jaw in humans, but the LPJ is equipped with an additional link in the chain - the struts. The macro-structure of the struts is depicted in figure 2.3a.

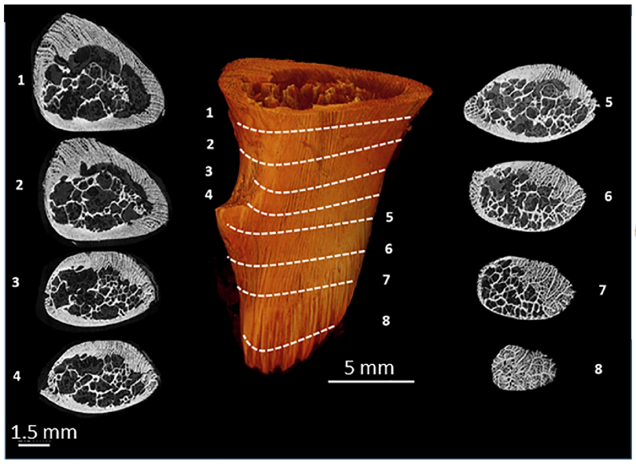
The load is transferred from the cleithrum through the struts and dental plate, where the mollusks and shellfish are crushed. These struts are thus subject to cyclic compressive loads. The macro-structure of these struts is similar to that seen in the mammalian femur. It has a relatively dense exterior that is relatively ordered in structure and a more porous interior that is more randomly organized. Here the similarities stop. While the cortical bone is approximately 97 % dense, the exterior of the LPJ has a porosity of about 50 %. It is, in other words, significantly less dense than the cortical bone but not as porous as the trabecular bone. These outer walls of the LPJ struts are also considerably more ordered than the trabecular bone [2].

The microstructure of the outer walls is depicted in figure 2.3b. It bears a reassembly to a lattice structure, an open-celled structure of connecting struts. A lattice structure is a load-bearing structure designed to carry as much load as possible, using as little material as possible [11]. The outer walls are built up by lamellae (plates) oriented to align with the load-bearing direction (z-axis). They also roughly align with the radial direction of the struts and are slightly curved around the z-axis. There are several thinners, transversely oriented beams connecting the lamellae between these lamellae.

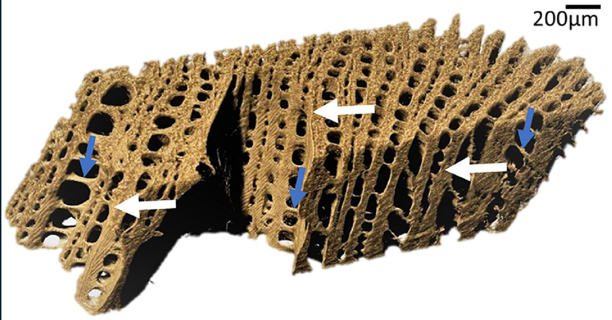
The central, more porous LPJ bone struts section consists of irregularly placed thin rods. It also varies in porosity, and the voids can vary in size by as much as a factor of 10, in contrast to the trabecular bone, which is commonly more uniformly distributed [12].

Loading direction

The lamella in the outer walls is the most significant mass in the load-bearing direction, capable of transferring the load from the cleithrum to the dental plate. They also make the struts able to withstand bending forces. The beams connecting the lamellas might work as



(a) The architecture of the strut material.



(b) Micro-tomography 3D images of the mid ventral zone of the strut. White arrows point to the lamellae, and blue arrows point to the transversal, supporting beams.

Figure 2.3: The architecture and microstructure of the lower pharyngeal jaw bone of the Black drum. Both figures are from E. Ziv et. al. [2].

support to absorb in-plane share forces and stabilize the plates [2].

Taking a closer look at the fibers of which the lamellae and beams are made supports this. The fibers in the lamellae are oriented in the load-bearing direction. The fibers in the beams merge into the plates, similar to what is observed in joints with trabecular bone.

2.3 Modeling fatigue using Finite Element Method

Direct cyclic is a method of simulating periodic loading that reduces computational time by extrapolating damage over a given number of steps.

When using Direct Cyclic to study fatigue, this can be done by the Extended Finite Element Method, also known as XFEM, or by using inelastic strain energy, or Hysteresis energy accumulated in the elements. Both will be introduced below.

2.3.1 Direct Cyclic

The Direct Cyclic step was introduced as a computationally effective method to predict the stabilized response of a structure subject to periodic, cyclic loading. This method is valid for the elastic-plastic structures subject to linear deformations[13] [14]. It ignores changes in contact, and frictional slipping is treated as non-slip contact. The Direct cyclic step will have difficulty converging when the structure is close to ratcheting [13].

2.3.2 Linear elastic fracture mechanics and XFEM

Using the extended finite element method (XFEM), a crack can propagate along a path that is not predefined. It uses enriched elements, eliminating the need to re-mesh the model.

When using XFEM, the user can choose whether they want to define a crack beforehand or not. If the user does not define a crack, the XFEM must be used in combination with the Maximum principal stress or strain damage criteria. Abaqus will then search for areas where the stress or strain exceeds maximum principal stress or strain, and initiate a crack/grow the existing crack [15].

If the XFEM-method is to be used in combination with fatigue analysis and the Direct cyclic step, a crack must be defined before the Direct-cyclic step starts. This can be either by e.g. a static step before the Direct Cyclic step or a user-defined crack [16].

For fatigue analysis, the XFEM follows Paris' law for crack propagation and crack growth onset [1]. Crack growth onset is defined to be when

$$f = \frac{N}{c_1 \Delta G^{c_2}} \geq 1. \tag{2.1}$$

Here c_1 and c_2 are material parameters, ΔG is the relative fracture energy release rate, and N is the cycle number. When the equation 2.1 is satisfied, the elements at the crack-tip will release only as long as G_{max}, the highest energy release rate that occurs during the cyclic loading, exceeds G_{thresh}. G_{thresh} is a material parameter.

When the criteria stated above are satisfied, the crack growth propagation is governed by

$$\frac{da}{dN} = c_3 \Delta G^{c_4}, \tag{2.2}$$

where a is the crack-length, $\frac{da}{dN}$ is the crack propagation rate and c_3 and c_4 are material parameters [17]. When ΔG exceeds $G_{plastic}$, the part fractures. In the simulation the crack will propagate by one element width for each cycle when this occurs.

2.3.3 Continuum damage approach and hysteresis energy

The continuum damage approach uses the accumulated inelastic energy, Δw, per cycle and

[1] Note that the crack growth onset refers to the cycle N where the pre-existing crack starts to grow.

material constants to predict damage initiation and prediction. After damage initiates, the material starts to degrade by a degradation factor, D. D signifies to what degree each element is degraded, ranging between values from 0 to 1. 0 means no damage, 1 means the element is no longer capable of carrying any load [18]. Damage initiation is set to be

$$N = c_1 \Delta w^{c_2}, \tag{2.3}$$

where c_1 and c_2 are material constants, and N signifies the cycle at which damage initiates [19]. After this the element degrades at the rate

$$\frac{dD}{dN} = \frac{c_3 \Delta w^{c_4}}{L}, \tag{2.4}$$

where c_3 and c_4 are material parameters, and L is the element width [20].

When the degradation factor is used, the element's load-carrying capacity is calculated as

$$\boldsymbol{\sigma} = (1 - D)\overline{\boldsymbol{\sigma}}, \tag{2.5}$$

where $\boldsymbol{\sigma}$ is the stress-tensor of the damaged element, and $\overline{\boldsymbol{\sigma}}$ is the undamaged stress tensor of the element [20], i.e what the load bearing capacity of the element would be at this point in the cycle had it not been damaged.

2.4 Cyclic hardening

A cyclic hardening model describes the response of a material subjected to cyclic loading. In this section, some models used to describe the behavior of some metals subject to cyclic loading are introduced.

2.4.1 Isotropic hardening

Implementing an isotropic hardening model, in effect, scale the yield surface of the material by a scalar value [21] as illustrated in figure 2.4. The yield surface, σ^0, of a material given a certain plastic strain, ε_p, is being described by the equation

$$\sigma^0 = \sigma_0 + Q(1 - e^{-b\bar{\varepsilon}^{pl}}), \tag{2.6}$$

where σ_0 is the yield surface before any plastic strain is accumulated, Q and b are material parameters [22]. Q is an asymptotic value and is calculated by finding the yield surface of the stabilized cycle for the material, and b indicates the speed at which the material stabilizes [21]. Thermodynamic effects are not considered in this thesis.

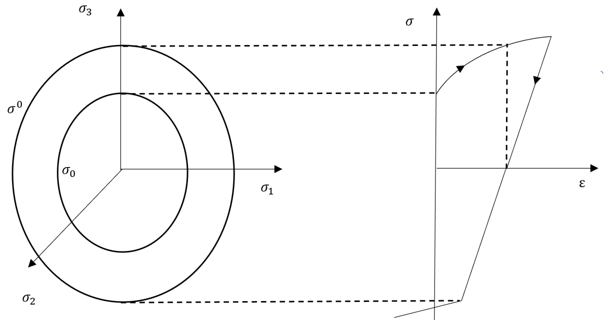


Figure 2.4: Isotropic hardening

2.4.2 Kinematic hardening

Implementing a linear, kinematic hardening model in effect translates the loading-surface of the stress-strain curve by a tensorial hardening variable **X** by

$$f = f_y(\boldsymbol{\sigma} - \boldsymbol{X}) - k. \tag{2.7}$$

Here f indicates the present loading function, f$_y$ the form of the yield criterion, and k is the yield stress (can be different from the usually known σ_y, which is the initial yield stress). This is illustrated in figure 2.5.

Several variations of linear kinematic hardening are formulated. In Abaqus, one is available for implementation, Ziegler's hardening rule, [22] and the basis for this, Prager's rule.

$$d\boldsymbol{X} = \frac{2}{3} C d\varepsilon^p, \tag{2.8}$$

where C is a material parameter, $d\varepsilon^p$ is equivalent plastic strain.

Ziegler's rule adds a term to Prager's rule

$$d\boldsymbol{X} = \frac{2}{3} C d\varepsilon^p + \frac{1}{C} \boldsymbol{X} dC, \tag{2.9}$$

where dC is the change of C with respect to time. As can be seen, if C is set as a constant value, it is equal to Prager's rule.

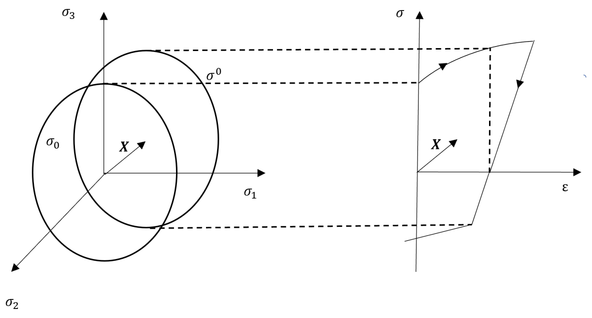


Figure 2.5: Linear kinematic hardening

Nonlinear kinematic hardening

In Prager's rule, equation 2.8, there is a proportionality between equivalent plastic strain, ε^p, and the hardening variable, **X**. This is removed in the nonlinear kinematic hardening rule from Lemaitre and Caboche (1990)[21], with the addition of a recall term

$$d\boldsymbol{X} = \frac{2}{3} C d\varepsilon^p - \gamma \boldsymbol{X} dp, \tag{2.10}$$

where dp is the change in accumulated plastic strain, and γ is a material parameter referred to as a decreasing function by Lemaitre and Caboche (1990) [21]. Both C and γ can vary with accumulated plastic strain, p, but that is not considered in this paper.

2.4.3 Combined isotropic and kinematic hardening

As simply scaling the yield surface, or simply translating the yield surface is not enough to describe the cyclic response for many materials, a combination of these two were introduced by Caboche and Lemaitre (1990) [21]. Combining the nonlinear kinematic hardening model with the isotropic hardening model allows the yield surface of the material model to both expand and translate.

Superpositioning kinematic models

The evolution of kinematic components, "backstresses" are described in the equation 2.11

$$d\boldsymbol{X}_k = \frac{2}{3} C_k d\varepsilon_p - \gamma_k \boldsymbol{X}_k \varepsilon_p \tag{2.11}$$

The overall back-stresses are summed together, as shown in equation 2.12

$$\boldsymbol{X} = \sum_{k=1}^{N} \boldsymbol{X_k} \tag{2.12}$$

Superpositioning several kinematic models, in addition to combining the nonlinear kinematic and isotropic hardening model, has the added effect that the accuracy of the model is improved for small strains and damping the excessive ratcheting effect that can occur if only one kinematic backstress is implemented [21].

2.5 Micromechanical modeling and Periodic boundary conditions

A representative volume element is used for modelling structures on a micromechanical level. The RVE represents the behaviour of the material on this scale. To ensure that the element deforms periodically, i.e. that the deformed RVE is periodic and spatially filling, periodic boundary conditions (PBC) are applied [23] [24]. The details of PBC implementation for RVEs is presented in this section.

2.5.1 Periodic Boundary conditions in 2 dimensions

As metioned above, there are certain requirements an RVE must fulfill when modelling it. It must be spatially filling, and periodic. This means there will be no cavities or overlaps. If we define an RVE with several pairs of points along the edges, each pair on the two vertical edges having the same y-coordinates, and each pair placed on the two horizontal edges having the same x-coordinated. An illustration of an undeformed RVE with PBC and node pairs A and B is provided in figure 2.6a. The displacement of each such pair of periodically placed points A and B is described as

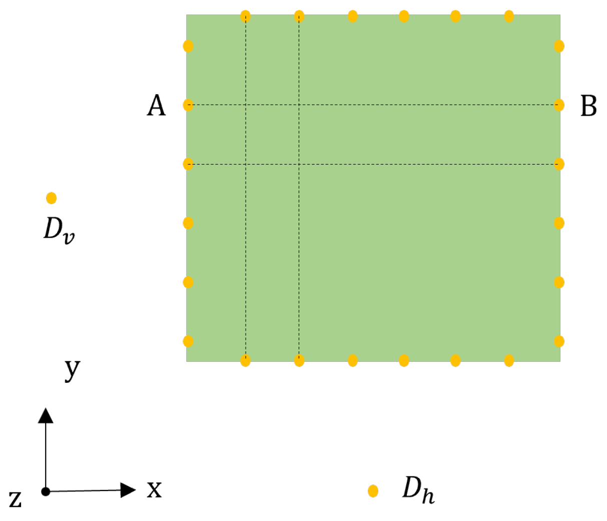
$$u(B) - u(A) = (\overline{F} - 1)(X(B) - X(A)) = \overline{H}(X(B) - X(A)). \tag{2.13}$$

Here u(A) and u(B) is the displacement at nodes A and B, respectively, ($\overline{F}$-1) is the macroscopic displacement, and X(A) and X(B) is the position in the reference configuration. For successfully modeling an RVE, this relation must be applied to each such pair of nodes along the edges of the RVE [25]. The macroscopic displacement, ($\overline{F}$-1), is applied to 'dummy' nodes, a node unconnected to the RVE itself. There is one dummy node for each degree of freedom, i.e. for a 2-dimensional RVEs there would be 2 'dummy' nodes, while for 3-dimensional RVEs there would be 3.

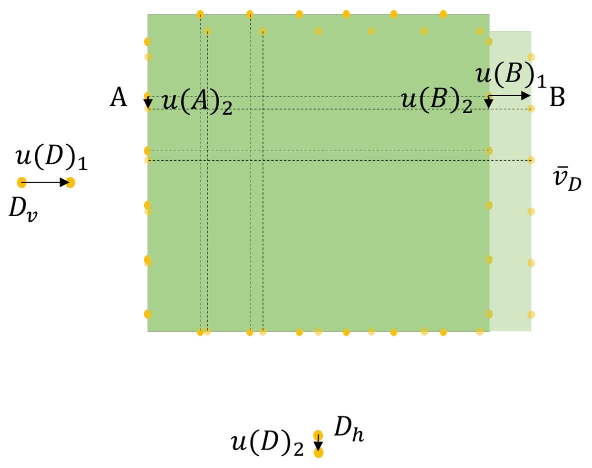
A constrain equation is used to implemenet the constraint described in equation 2.13 in finite element analysis. For node pair A and B on a 2-dimensional RVE, the equations would be

$$\begin{aligned} u(A)_1 - u(B)_1 - u(D)_1 &= 0 \\ u(A)_2 - u(B)_2 - u(D)_2 &= 0, \end{aligned} \tag{2.14}$$

where u(X)$_{\mathrm{i}}$ denotes the displacement in point X in direction i [24]. Point D refers to the 'dummy' node associated with the edge-pair the points A and D. An illustration of this deformation is provided in figure 2.6b.



(a) Undeformed RVE with PBC.



(b) Deformed RVE with PBC.

Figure 2.6: RVE with PBC. D$_v$ denotes the 'dummy' node associated with node pairs on vertical edges, and D$_h$ is the 'dummy' node associated with node pairs on horizontal edges.

2.6 Elastoplasticity

Total strain ε is composed of elastic strain, ε_e, and inelastic strain, ε_{ie}. Inelastic strain can be plastic strain, viscoplastic strain, anelastic strain, among others [21]. In the realm of elasto-plasticity, strain is defined as consisting only of elastic strain and plastic strain, ε_p,

$$\varepsilon = \varepsilon_e + \varepsilon_p. \tag{2.15}$$

2.7 Numerical integration

Numerics is used for problems in arithmetics that cannot be solved exactly. There are several methods for finding the integral of a function, f, using only some points on it. Two such methods is the Trapezoidal rule, and the Simpson rule.

2.7.1 Trapezoidal rule

The Trapezoidal rule uses linear interpolation between two points to approximate f, and sums up these trapezoids for the entire interval of the function as shown in equation 2.16.

$$\int_a^b f(x)dx = h(\frac{f(x_0) + f(x_1}{2} + ... + \frac{f(x_{n-1}) + f(x_n)}{2}) \tag{2.16}$$

where n is the number of points on the function f [26].

2.7.2 Simpsons Rule

The Simpson rule merges two intervals neighboring intervals, making one interval with three known values, and approximates f through these three points using the unique quadratic function.

$$\int_a^b f(x)dx = \frac{h}{3}(f(x_0) + 4f(x_1) + 2f(x_2) + ... + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)) \tag{2.17}$$

The Simpson rule requires an even number of intervals.

2.7.3 Accuracy

Given a set of nodes on a function, it is proved that the Simpsons method is more accurate than the Trapezoidal rule [27]. A simple proof is provided here.

$$\begin{gathered} f(x) = cos(\frac{\pi}{2}x) \\ \int_0^1 f(x)dx = 0.63662... \end{gathered} \tag{2.18}$$

Given the nodes (0, f(0)), (0.5, f(0.5)) and (1, f(1)), the accuracy of the Trapezoidal rule and the Simpson rule will be demonstrated.

Trapezoidal rule

$$\begin{gathered}\int_0^1 f(x)dx \approx \frac{1}{2}(\frac{f(0)+f(0.5)}{2}+\frac{f(0.5)+f(1)}{2}) \\ \int_0^1 f(x)dx \approx 0.60355... \\ e = 0.63662 - 0.60355 = 0.03307\end{gathered} \tag{2.19}$$

Simpson Rule

$$\begin{gathered}\int_0^1 f(x)dx \approx 16(f(0)+4f(0.5)+f(1)) \\ \int_0^1 f(x)dx \approx 0.63807... \\ e = 0.63662 - 0.63807 = -0.00145\end{gathered} \tag{2.20}$$

As can be seen from equation 2.19 and 2.20, given the same three nodes, the Simpson-rule is more accurate.

2.8 Machine learning using neural networks

The motivation for using machine learning in this thesis is to solve a multivariate problem that is not linearly separable. To do this, a multi-layer perceptron (MLP) will be used. A multi-layer perceptron is a deep-learning method. Deep learning is a form of machine learning, where a model learns by exposure to data. This learning can be supervised or unsupervised. In supervised learning, the model is exposed to a data-set containing input and outputs or questions and answers. The model then adapts to predict the correct output corresponding to a known input. A typical example where supervised learning is used is to predict house prices in an area or classify handwritten numbers from an image. This will be explained more in detail later. Unsupervised learning is not part of this thesis, but it means that a model is given a data set that is not labeled. Unsupervised learning would be used, e.g., for identifying groups with similarities in large data sets, like separating a forest into trees, birds, insects, etc. This is useful when the groups, or clusters, in a population are not already known, e.g., in a population of patients with unidentified diseases.

From the example given above alone, it can be seen that machine learning has a wide range of potential in many fields.

2.8.1 Neural networks

Neural networks 'learn' by training. A neural network consists of an input layer, one or more hidden layers, and an output layer, where the data flows from the input layer, through the hidden layers, and to the output layer. Each hidden layer consists of several nodes, each of which output, x, to another node is weighed. This is illustrated in figure 2.7. The back-propagation learning algorithm adjusts these weights as training data is added to the network so that the network predicts the correct output for a known set of inputs [28]. E.g. for a regression model, the output would be the predicted value of the variable(s), e.g., pricing of a house. For a classification model, the output would be the probability that each classification was the correct one, e.g., which number the image of the handwritten number depicts.

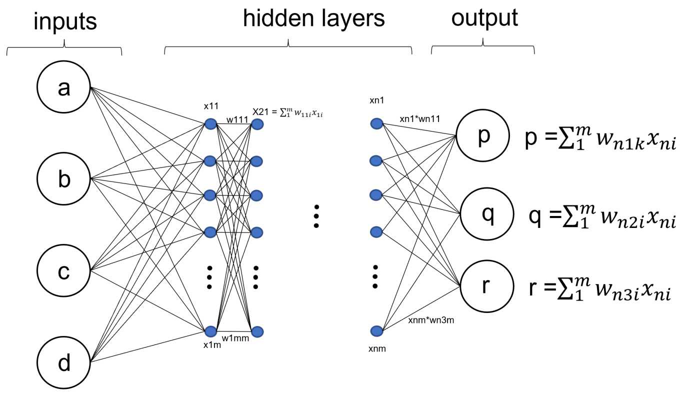

inputs
hidden layers
output
a
b
c
d
x11
w111
$X21 = \sum_1^m w_{11i}x_{1i}$
x1m
w1mm
xn1
xn1*wn11
xnm*wn3m
xnm
p
q
r
$p = \sum_1^m w_{n1k}x_{ni}$
$q = \sum_1^m w_{n2i}x_{ni}$
$r = \sum_1^m w_{n3i}x_{ni}$

Figure 2.7: Illustration of a fully connected, regression neural network. a, b, c and d are the inputs, xij is the value each node holds depending on the input values, wijk is the weight attributed to each node connection, and p, q, and r are the predicted output values.

Such a model is called a multi-layer perceptron (MLP). Several different optimizers are proposed for optimizing the weights, such as the Adam optimizer [29] and the LBFGS (Limited memory Broyden–Fletcher–Goldfarb–Shanno) optimizer [30][31].

This makes the neural network well suited to predict the correct output for the data it's been trained for, but the user also needs to know how well the algorithm performs for data it hasn't seen before. The model can be assessed by the coefficient of determination, R2. The coefficient of determination is defined as

$$R2 = (1 - \frac{SSE}{SST}), \tag{2.21}$$

where SSE is the error sum of squares, and SST is the true, or total, the correct sum of squares, defined as

$$\begin{aligned} SSE &= \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \\ SST &= \sum_{i=1}^{n}(y_i - \overline{y}_i)^2 \end{aligned} \tag{2.22}$$

where y_i is the true value of the output, $\hat{y}_i$ is the predicted value of the output, and $\overline{y}_i$ is the mean value of y [32] [33]. When all residuals, (y_1 - $\hat{y}_i$), is zero, the R2 = 1. The higher value the R2 has, the better fit it is. Despite this, relying entirely upon the R2 value when deciding between models is not recommended. E.g., adding an additional term could decrease the SSE, which could lead to an artificially high R2 value [32].

2.9 Optimizing using differential evolution

Differential evolution is one of the most popular global optimization methods for complex problems and was first proposed in 1996 by Storn [34][35]. Differential evolution is a population-based, stochastic Evolution Algorithm. I.e., the algorithm goes through the different combinations of the factors, limited by a set of boundaries (the population) in a stochastic manner to find the minimum solution. It was proposed as a minimization technique fulfilling the following five requirements proposed by Storn and Price (1997) [35]:

1) Ability to handle non-differentiable, nonlinear, and multi-modal cost functions.

2) Parallelizability to cope with computation-intensive cost functions.

3) Ease of use, i.e., few control variables to steer the minimization. These variables should also be robust and easy to choose.

4) Good convergence properties, i.e., consistent convergence to the global minimum in consecutive independent trials.

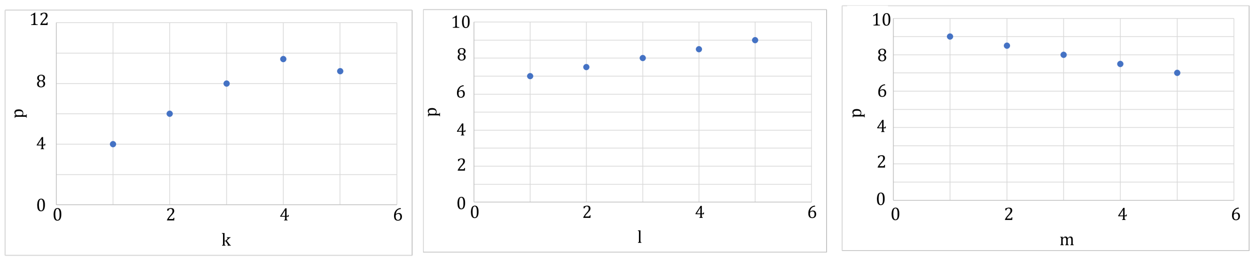
Fulfilling these requirements makes the differential evolution method suitable for time-consuming finite element experiments.

2.10 Design of Experiments

Design of Experiments (DoE) is a mathematical tool used to design the results of experiments where multiple variables control the result of the experiment. Knowing how to conduct experiments that are more complex than one input-one output can severely reduce the number of experiments required, and increase the chance of getting an accurate result [36]. When conducting experiments where several factors are considered, there are several different ways to conduct said experiments. Three different approaches will be presented in this section.

2.10.1 OFAT

One factor at a time (OFAT) is a traditional approach used by engineers and scientists to experiment with several factors. Using this approach, the experimenter changes only one factor at a time while the other remains constant. I.e., given three factors a, b and c and fixed factor levels k, l, m, and response p, the experiment could yield a response as seen in figure 2.8 [36].



(a) OFAT factor k (b) OFAT factor l (c) OFAT factor m

Figure 2.8: OFAT illustratory example

Given the response p(k, l, m) seen in figure 2.8, and assuming the goal is to maximize p, it would seem that the best factor levels are k = 4, maximize factor l, and minimize factor m. Here the downside of OFAT comes into play, as this method does not consider the effect one factor has on the other. For all the experimenters know, the response p with varying k, l=5, and m=1 could be as illustrated in figure 2.9.

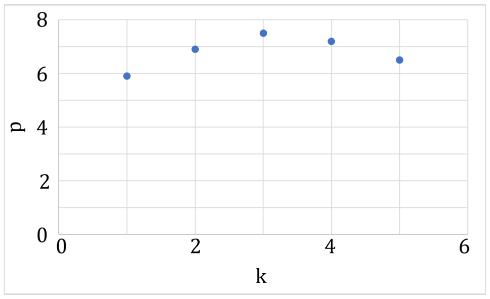


Figure 2.9: OFAT k, maximized l, and minimized m.

Here the overall response p is lower, and the maximum yield is found in k=3, not k=4. This example illustrates the flaw in using OFAT, based on the literature available on the subject [36], [37].

2.10.2 Factorial design

For factorial design, each factor is varied n levels. The most common is a 2-level factorial design. This makes it possible to examine the interaction between the parameters. This design requires n^k experiments, where n is the number of levels each factor k is to be studied [38][39]. If one can assume no interaction between the parameters, the method can be simplified with the Placket-Burman Design to 4k experiments [40].

2.10.3 Split Plot design

Split Plot design is derived from the factorial design. It is a DoE method where the experimenter can divide the experiment into bulks. I.e., it lets the experimenter differentiate between the different factors and single some out for closer examination[41] . For each bulk, one or more parameter have set factor levels, and within each bulk, the factors that the experimenter want to look at more closely are varied. An example is a two-factor experiment with parameters a and b, and response p. The value for b is more important than a, so the experimenter wants to try three values for b, but five values for a.

	a_1	a_2	a_3	a_4	a_5
b_2	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
b_2	p_{21}	p_{22}	p_{23}	p_{24}	p_{25}

Table 2.1: Split plot design example

Chapter 3

Methods

In this chapter, the methods used in this thesis will be described. This is the method relating to the structure: how the structure and the parameters are defined; the finite element model: material model, boundary conditions, and mesh; data collection and parametric study: how the data is collected and on what basis they are evaluated, and what method is used; machine learning aspect: what method is used to train a neural network model, and what steps are taken; optimization: methods used to optimize the parameters of the structure depending on the evaluation criteria.

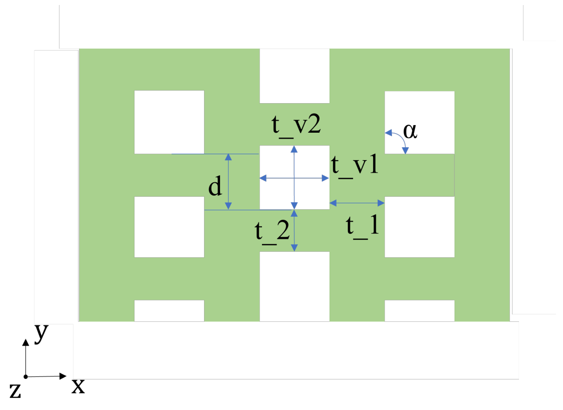
3.1 Lattice structure

The lattice structure of which the parameters are to be investigated is a simplified geometry based on image scans of the jaw-bone [2]. The image scan and the simplified lattice structure can be seen in figure 3.1. The resulting Representative Volume Element (RVE) is illustrated in figure 3.1c.

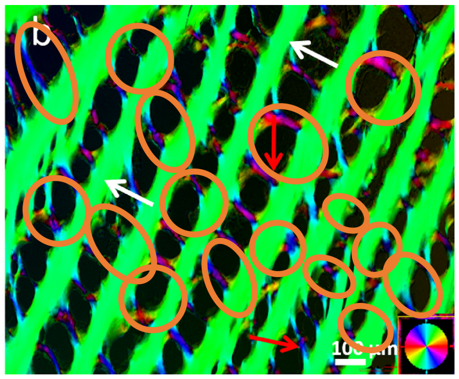
Based on the scan in figure 3.1, the mean values and Standard Deviation of the parameters in figure 3.1 a) were calculated [42]. The result of this can be seen in table 3.1.

3.1.1 Parametric study

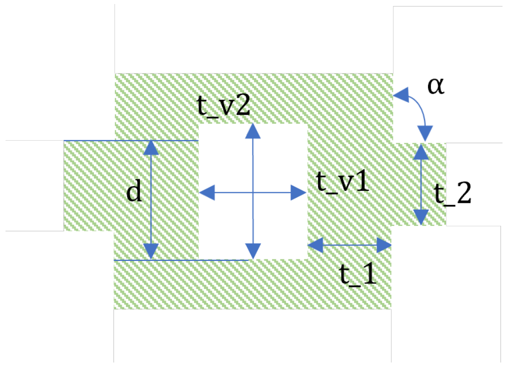
The goal of the parametric study is to understand how the different parameters influence the performance of the structure. The goal was to find the lowest number of factors necessary to



(a) A sketch of simplified lattice structure with key parameters based on the scan. The figure is from the authors' project thesis [42]



(b) Scan of the LBJ bone. 16 areas for measuring 5 key parameters t_{v1}, t_{v2}, t_1, t_2, and α are highlighted. Background figure in b) is from E. Ziv et al. [2].



(c) Resulting representative volume element (RVE).

Figure 3.1: Simplified lattice structure and original scan.

Table 3.1: Mean values and Standard Deviations for four of the key parameters. Table is from K. Engen [42]

Parameter	Mean value	Standard Deviation [μm]	Standard Deviation [%]
t_1	71.12 μm	15.94	22.4
t_{v1}	94.12 μm	28.03	29.8
t_2	27.37μm	10.48	38
t_{v2}	106.19 μm	30.29	28.5
f_v	0.547	0.331	60.57
α	77.69°	12.2°	15.7

describe the structure to make the parametric study as simple as possible. There are several different ways to define the geometry of the structure, by Volume fraction, by the angle α, the values t_{v1}, t_{v2}, t_1 and t_2, and the relationships between the last four. There a total of 6 unique relations (12 counting the inverse) between t_{v1}, t_{v2}, t_1 and t_2. Two of these will be investigated more closely, namely $\frac{t_{v2}}{t_{v_1}}$ and $\frac{t_1}{t_2}$, from here on will these be referred to as R_1 and R_2, respectively. The structure can be defined using these two parameters, R_1 and R_2, combined with the volume fraction and the angle, α. Making sure one is not influenced by changing another is key, and the method to do this is demonstrated in the following sub-sections.

R$_1$

Given a ratio $R_1 = R_1$, volume fraction $f_v = v$, $t_1 = t_1$ and $t_2 = t_2$, if was found that the new values for t_{v1} and t_{v2} was best found by the method in equation 3.5.

We start with the equation for the volume fraction,

$$\begin{aligned} v &= \frac{V_t - V_v}{V_t} = \frac{A_t - A_v}{A_t} = 1 - \frac{A_v}{A_t} \\ v &= 1 - \frac{t_{v1}t_{v2}}{(t_{v1}+t_1)(t_{v2}+t_2)}, \end{aligned} \tag{3.1}$$

where V_t is the volume of the RVE [1] per unit thickness, V_v is the volume of the void-area per unit thickness, A_t is the total area of the RVE in the xy-plane defined in figure 3.1, and A_v is the area of the void in the xy-plane. Inserting the relation

$$\begin{aligned} \frac{t_{v2}}{t_{v1}} &= R_1 \\ t_{v2} &= R_1 \cdot t_{v1} \end{aligned} \tag{3.2}$$

into equation 3.1 and solving for t_{v1} gives

[1]figure to explain RVE is needed

$$\begin{aligned}
v &= 1 - \frac{R_1 t_{v1}^2}{(t_{v1}+t_1)(R_1 t_{v1}+t_2)} \\
0 &= 1 - v - \frac{R_1 t_{v1}^2}{(t_{v1}+t_1)(R_1 t_{v1}+t_2)} \\
0 &= (1-v)(t_1 t_2 + R_1 t_1 t_{v1} + t_2 t_{v1} + R_1 t_{v1}^2) - R_1 t_{v1}^2 \\
0 &= (1-v)t_1 t_2 + t_{v1}(R_1 t_1 + t_2)(1-v) + t_{v1}^2 R_1 (1-v) - R_1 t_{v1}^2 \\
0 &= c + b t_{v1} + a t_{v1}^2
\end{aligned} \tag{3.3}$$

where

$$\begin{aligned}
a &= -R_1 \cdot v \\
b &= (R_1 t_{v1} + t_2)(1-v) \\
c &= (1-v)(t_1 t_2).
\end{aligned} \tag{3.4}$$

t_{v1}, t_{v2} is then obtained by solving

$$\begin{aligned}
t_{v1} &= \frac{-b-\sqrt{b^2-4ac}}{2a} \text{ and} \\
t_{v2} &= R_1 \cdot t_{v1}.
\end{aligned} \tag{3.5}$$

By following this method, only t_{v1}, t_{v2}, R_1, and the size of the RVE is changed. The volume fraction, t_1, t_2 and R_2 remains unchanged.

R_2

Calculating R_2 follows the same procedure as calculating R_1. Inserting the relation

$$\begin{aligned}
\frac{t_1}{t_2} &= R_2 \\
t_1 &= R_2 \cdot t_2
\end{aligned} \tag{3.6}$$

into equation 3.1 and solving for t$_2$ gives

$$\begin{aligned}
v &= 1 - \frac{t_{v1}t_{v2}}{(t_{v1}+t_1)(t_{v2}+t_2)} \\
0 &= 1 - v - \frac{t_{v1}t_{v2}}{(t_{v1}+R_2t_2)(t_{v2}+t_2)} \\
0 &= (1-v)(t_{v1}t_{v2} + t_2t_{v1} + R_2t_2t_{v2} + R_2t_2^2) - t_{v1}t_{v2} \\
0 &= -v(t_{v1}t_{v2}) + (1-v)(t_{v1} + R_2t_{v2})t_2 + (1-v)R_2t_2^2 \\
0 &= c + bt_2 + at_2^2
\end{aligned} \tag{3.7}$$

where

$$\begin{aligned}
a &= (1-v)R_2 \cdot v \\
b &= (1-v)(t_{v1} + R_2t_2) \\
c &= -v(t_{v1}t_{v2}).
\end{aligned} \tag{3.8}$$

t$_1$ and t$_2$ are then obtained by solving

$$\begin{aligned}
t_2 &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ and} \\
t_1 &= R_2 \cdot t_2
\end{aligned} \tag{3.9}$$

By implementing this method, only t$_1$, t$_2$ and R$_2$, while the volume fraction, the size of the RVE, t$_{v1}$, t$_{v2}$ and R$_1$ remains unchanged.

Volume fraction

Changing the volume fraction, while maintaining the other parameters also requires careful calculations. Here we maintain the size of the RVE, A$_t$, R$_1$ and R$_2$, while the numerical values of t$_1$, t$_2$, t$_{v1}$, t$_{v2}$ is changed.

We start with the definitions of the area of the RVE

$$A_t = (t_{v1} + t_1)(t_{v2} + t_2) \tag{3.10}$$

A_t, R_1 and R_2 remaining unchanged is used as the basis for calculating A_v, t_{v1} and t_{v2} in equation set 3.11.

$$\begin{aligned} A_v &= A_t(1 - v) \\ A_v &= t_{v1}t_{v2} = R_1 t_{v1}^2 \\ t_{v1} &= \sqrt{\frac{A_v}{R_1}} \end{aligned} \tag{3.11}$$

From here, the method described in sub-section 3.1 is used to find t_1 and t_2, using the already known R_2, v, t_{v1} and t_{v2}.

3.2 FE model

3.2.1 Material model

In this section, the mechanical properties of the material implemented in the Finite element model are described. This model describes an elastic and plastic behavior and the damage model, i.e. under which criteria and how damage initiates and evolves in the model. The material described is not being investigated in this thesis.

Cyclic hardening model

The plastic deformation of the finite element model is described using a cyclic hardening model, as the response to a periodic, cyclic loading and obtaining a stabilized stress-strain cycle is key when investigating fatigue.

The cyclic hardening model implemented is nonlinear combined isotropic and kinematic cyclic hardening, as described in section 2.4. The kinematic model consists of a superposition of several back-stresses, which leads to a less pronounced ratcheting effect [21], as described in section 2.4. This is important to help with the convergence of the direct cyclic step, as described in section 2.3. The combination of kinematic and isotropic hardening has the advantage of describing both a scalar deformation and translation of the stress-strain curve [22] [21]. The material parameters for the kinematic hardening model and the isotropic hardening model are collected from a material model developed and tested with experimental data by Song et al. (2021) [43], and are presented in table 3.2 and 3.3, respectively.

Table 3.2: Kinematic hardening parameters

i	1	2	3	4	5
C_i [MPa]	84844	60486	18041	4935	2426
γ_i	5085	881.1	163	100.6	9

Table 3.3: isotropic hardening parameters

σ_0 [MPa]	Q [MPa]	b
450	-70	2

Damage model - XFEM and MaxPS

The XFEM method, in combination with the Max Principal Stress damage model, was investigated as a potentially suitable damage model to investigate the performance of the different RVE's. Because a crack cannot be initiated in a Direct cyclic step, it would have to follow a static step in which the crack was initiated. This method would require confirmation that a crack was initiated. As this process was supposed to be as automatic as possible, it was decided that the continuum damage model using Hysteresis energy was a better fit.

Damage model - Hysteresis energy

The damage initiation and damage evolution based on accumulated hysteresis energy (plastic strain energy) were introduced in Abaqus using keyword editor. The code for this can be seen in Appendix A.4.3, from line 589. Calculation of the parameter c_3 has to be done for each RVE because it is dependent on the characteristic element length [20].

Because of some discrepancy between sources [43] [20] on how to calculate c_3 based on the element size for use in Abaqus, some simulations using different mesh sizes were run to verify the computational method. The paper from which the material data is collected indicates that the c3 value should be calculated in the following manner:

$$c_3 = a/L, \tag{3.12}$$

where L is the characteristic length of the element, and a is a material constant from the relation between degradation values in the damage-evolution state, and the plastic strain energy Δw

$$\Delta D/\Delta N = a\Delta w^{c_4} = 1.39e - 3\Delta w^{0.095} \tag{3.13}$$

Song et al. (2021) [43] conclude that the c_3 value should be 7.94e-4 because their characteristic element length L=1.75 mm. The investigation of this author, however, supports that the c_3 value should be calculated as

$$c_3 = a \cdot L, \tag{3.14}$$

to return the same behavior when changing the element size. Three finite-element jobs were used to conduct the test that led to this conclusion: 1) replication of the results of Song et al. (2021) to ensure that the model was correct; 2) repeating 1) with half the element mesh-size calculating c_3 as in equation 3.12 using L=0.875; 3) repeat 2) but calculating c_3 as in equation 3.14. As the results in Song et al. (2021) using c_3 = 7.94e-4 yielded the correct results [43], a was calculated based on this c_3 value and equation 3.14 for step 3) to be

$$a = \frac{c_3}{L} = 0.0004537 \tag{3.15}$$

yielding

$$c_3 = a \cdot L = 0.0004537 \cdot 0.875 = 0.0003971 \tag{3.16}$$

for case 3).

The results of the trials can be seen in table 3.4. From these results, it is clear that the method in equation 3.14 is more accurate; thus, the c_3 value for the Fe-models will be calculated according to equation 3.17.

Table 3.4: Confirming calculation of c_3

	c_3	SDEG at N=1000
1)	7.94e-4	0.2299
2)	1.589e-3	0.6858
3)	3.971e-4	0.2496

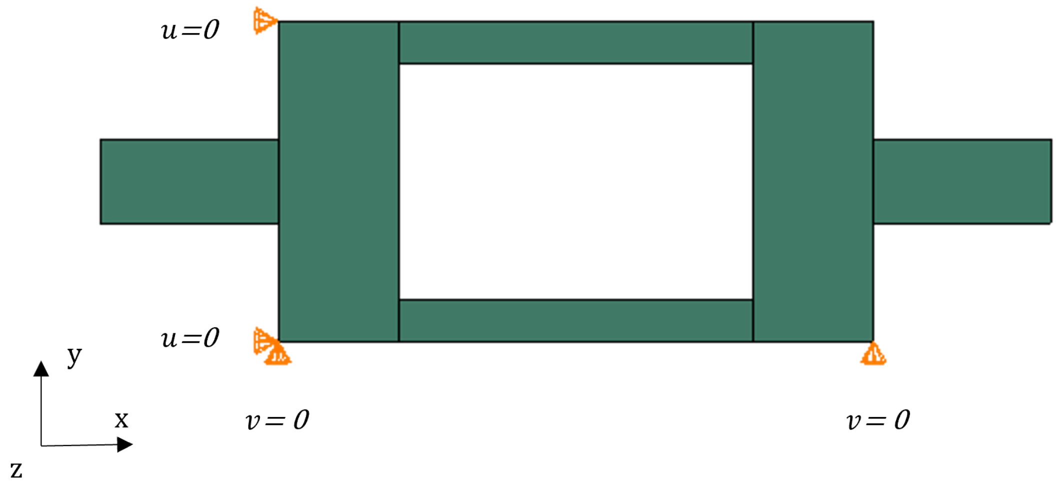


Figure 3.2: Rigid motion Boundary condition on the RVE

$$c_3 = 0.000453878 \cdot L \tag{3.17}$$

3.2.2 Boundary conditions and loads

Periodic boundary conditions (PBC) were used on the RVE. Code written by the author was used for the implementation of the PBC. This code is available in section A.4.3 lines 188 through 433. In addition to this, boundary conditions were implemented to prevent rigid motion. The Boundary conditions were tested on the simple square. In addition, 5 RVE geometries were checked for wrongful Reaction forces and out-of-character stresses and strains, i.e., out-of-place Reaction forces on the boundary conditions and dummy nodes.

Verification

Checking the RF and stress distribution of the model using different boundary conditions. Referring mainly to the angle RVE and the simple square.

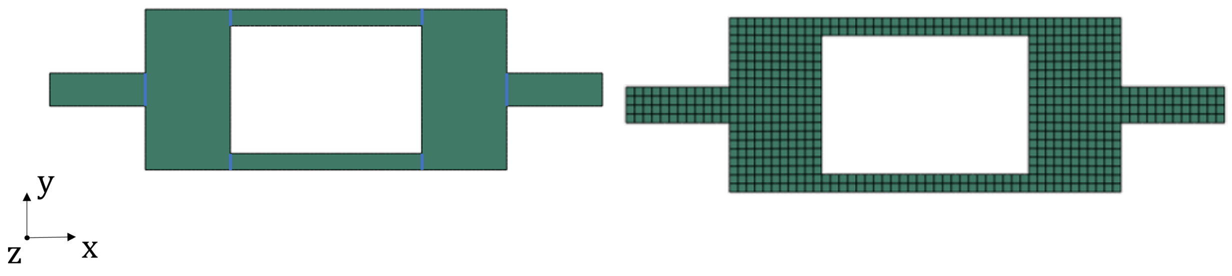
3.2.3 Mesh

The Element type used in the analysis is CPE8R. This is a plane strain, 8 node element with reduced integration. The validity of this element was checked against the material model fatigue evolution from Song et al. (2021) [43], checking that the analysis yielded the correct result using this element and the boundary conditions chosen.

The 8 node element was used instead of the more cost-effective 4-node element because the deformation of the structure demands a second-degree equation to describe it.

Partition

As the value c_3 has to be calculated by the user concerning the element width [20], care was taken that the elements throughout the mesh had the same width. This was done by partitioning each RVE in such a manner that the mesh would always be regular, and the element width could be known without having to measure it manually. The partition and the meshed part are illustrated in figure 3.3.



(a) Illustration of partition

(b) Resulting Mesh

Figure 3.3: Illustration of how mesh-regularity is achieved

Convergence analysis

A convergence analysis was conducted to ensure as low run-time per job as possible. As the RVEs could vary significantly in absolute sizes, The convergence analysis was performed for several different RVEs, with regards to the calculated inelastic strain, Δ w. This was seen as being a universal approach. As the number of elements along the width of the beams of the RVE would also be affected by this value, as well as relation $\frac{t_1}{t_2}$, the convergence analysis was conducted for several different RVEs. The result of the convergence study can be seen in table 3.5, table 3.6, figure 3.4 and figure 3.5[2]. These are the two most extreme cases. Other results from the convergence study can be seen in the appendix, A.1.1.

Table 3.5: Convergence study: R_1, R_2 = 0.5, $\varepsilon = 0.006$, N=1000.

n	Elements	Time [s]	Δ w (N=1000)	d Δ w
5	34	42.17	0.595	
10	136	68.48	0.597	-0.3 %
20	544	212.95	0.584	- 2.2 %
40	2160	775.67	0.581	0.5 %

Table 3.6: Convergence study: R_1, R_2 = 3, $\varepsilon = 0.006$, N=1000.

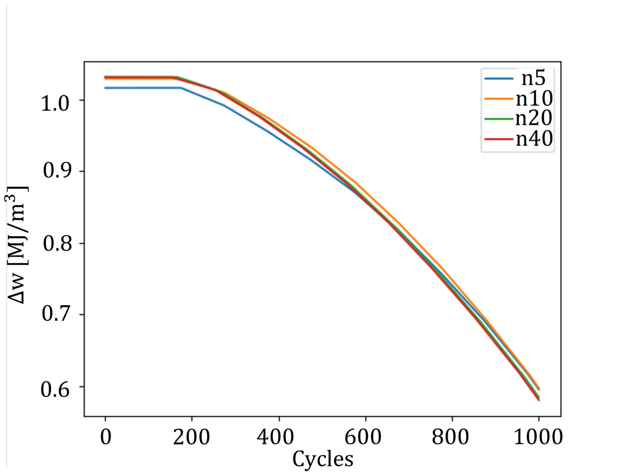
n	Elements	Time [s]	Δ w (N=1000)	d Δ w
5	20	32.13	1.161	
10	76	38.48	1.813	+ 56.2 %
20	224	60.41	1.727	- 4.7 %
40	904	155.22 s	1.701	1.5 %

3.3 Evaluation of the RVEs

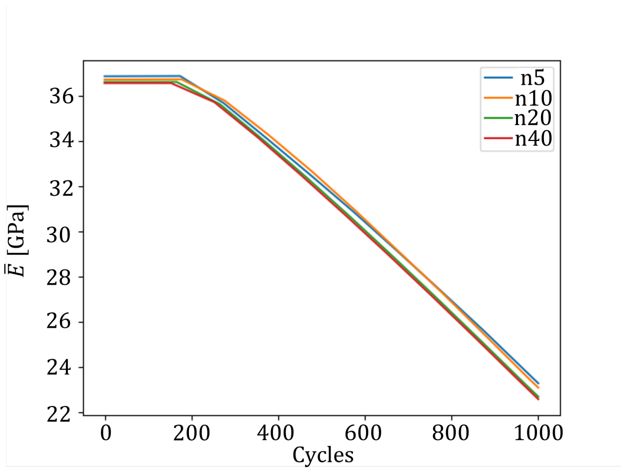
The RVEs will be evaluated based on the number of cycles N it takes for the effective stiffness, $\overline{E}$, of a unique RVE to degrade by a set factor D.

The code written for collecting the different result can be found in the Appendix A.4. The results were evaluated by the accumulated inelastic strain energy, Δ w, and the effective stiffness, $\overline{E}$.

[2]Note the convergence-behaviour in R=3. Instead of converging towards one value from one side, it seems to oscillate around a value. This could have implications for the stability of the mesh (presented in section ??).

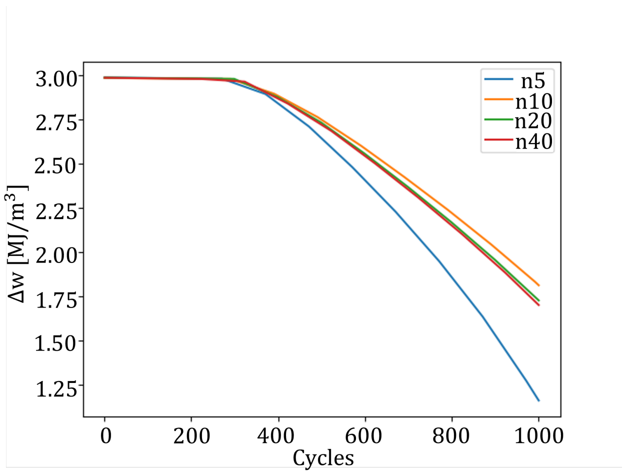


(a) Plastic strain energy, Δw

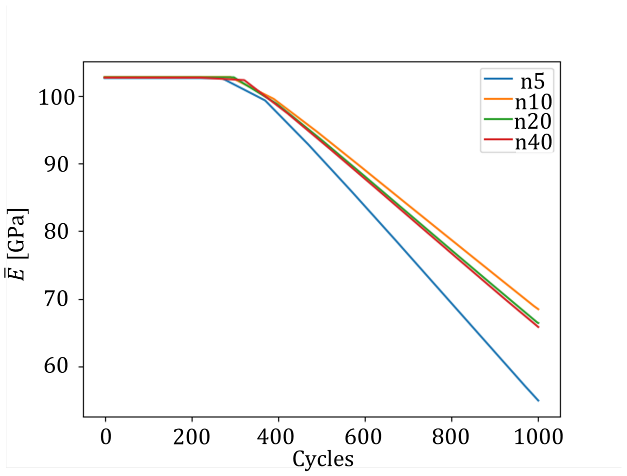


(b) Effective Stiffness, E'

Figure 3.4: Convergence study: R_1, R_2 = 0.5, ε = 0.006, N=1000.



(a) Plastic strain energy, Δw



(b) Effective Stiffness, E'

Figure 3.5: Convergence study: R_1, R_2 = 3, ε = 0.006, N=1000

Effective stress and strain in the y-direction were collected for each cycle N for the different RVEs. The effective stiffness was calculated based on this data. In this case, the effective stiffness is set to be the slope of the stress-strain curve where the strain is released from its maximum value in compression, i.e when the strain is decreased from 0.8%. As the Direct cyclic step is set to a fixed step-increment at 0.1, and the load is applied as a sinus-curve [3], this means the effective stiffness is calculated as the slope of the stress-strain curve between

[3]Introduce this earlier

the strains -0.8% (x1i) and -0.76 % (x2i), the two points highlighted in figure 3.6. As the Direct cyclic step extrapolates damage and does not iterate through every cycle, the exact step increment where the stiffness decreases by 10 % is calculated using linear interpolation between the two points around which this transition occurs.

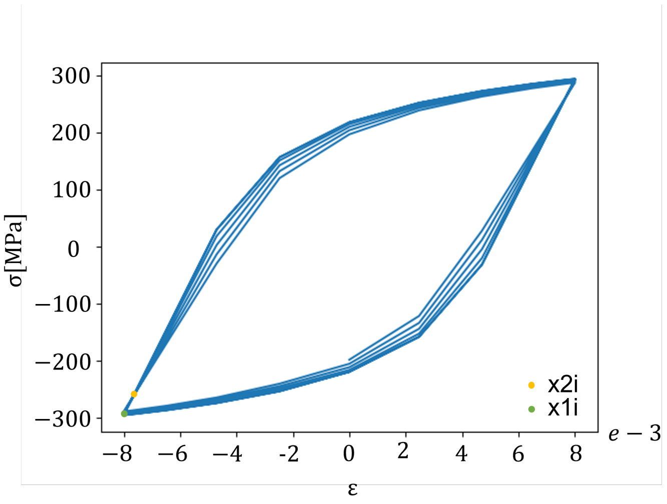


Figure 3.6: Exemplary stress-strain curve with x1 and x2 points highlighted.

The hysteresis energy, or accumulated inelastic strain energy, Δ w, is calculated using numerical integration and the Simpson rule. As there are 20 points on the stress-strain curve for each cycle, the error is expected to be small [27].

3.3.1 Data Collection for parametric study

The data collection was done in several steps. After verifying the validity in the model, the first step was running it for an extensive range of parameters presented in Table 3.7. As can be seen from the table this results in 450 variations of the RVE and 450 jobs. From the convergence study and tables 3.6, 3.5 and A.1 through A.6 we can see that the time it takes to run one job with n=10 ranges from 0.5-2 minutes. Assuming an average of 68 seconds, this results in 8 hours and 30 minutes of run-time. The computer used was equipped with a Ryzen AMD Ryzen 7 4700U processor with Radeon Graphics and a clock rate of 2.00 GHz.

The next round of data collection was conducted for a range of parameters based on the work done on the structure's geometry in the authors' project thesis [42]. The parameters focus

Table 3.7: Range of parameters for first round of data-collection

Parameter	Values	No.
R_1	0.33, 0.5, 1, 2, 3	5
R_2	0.33, 0.5, 1, 2, 3	5
V_f	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9	9
ε	0.6e-02, 0.8e-02	2

Table 3.8: Mean values and standard deviation. Table is from K. Engen (2021) [42].

Parameter	Mean value	Standard Deviation	Standard Deviation [%]
R_1	1.128	0.465	41.2
R_2	2.598	1.153	44.4
f_v	0.547	0.331	60.57
α	77.69°	12.2°	15.7

on the mean value and the standard deviation of the structure presented in Table 3.8.

For the second round of data collection, the parameter range of [Mean-1SD, Mean - 0.5 SD, Mean, Mean + 0.5SD, Mean + 1 SD] was chosen to look closer at what happens around the values for the standard deviation. The numerical values are presented in table 3.9.

Table 3.9: Range of parameters for second round of data-collection

Parameter	Values	No.
R_1	0.663, 0.895, 1.128, 1.360, 1.593	5
R_2	1.445, 2.022, 2.598, 3.1745, 3.751	5
V_f	0.216, 0.382, 0.547, 0.713, 0.878	5
ε	0.6e-02, 0.8e-02	2

3.4 DoE

The factors R_1, R_2, and f_v, was chosen to describe the RVEs, reducing the number of factors to three. To understand the effect these factors have on the RVE and its performance to withstand fatigue wear, the OFAT approach and full factorial design was tested. The full factorial design will be implemented to investigate and understand the interaction between the three factors. The structure has 3 factors and will be set to three levels each, resulting in $3^3 = 27$ experiments.

3.5 Predicting and understanding the structure using Machine learning

Part of the goal of this thesis was to investigate if machine learning could be a valuable tool to understand the structure better, predict the performance of a given RVE and possibly find an optimal RVE for fatigue life.

The method implemented was a neural-network regression model from sk learn library [33]. Here two optmizers will be tested, the Adam optimizer [29], and as the data-set might be small, the LBFGS optimizer [30] will also be tested.

The MLPRegressor model takes in a data-matrix X and a set of Y targets. The X-data will in this case contain three values R_1, R_2, f_v, and the strain in parts per thousand. The Y-data will be one of the following three: N before $\overline{E} = 0.85\overline{E}_0$, i.e the effective stiffness of the RVE has decreased 15 %; N_{onset} and relative slope $a/\overline{E}_0$; or N_{onset}, slope a and $\overline{E}_0$.

The MLP model was verified using the coefficient of determination R2, introduced in section 2.8. 20% of the data-set is reserved for testing, and only 80% of the data-set was used for training the model. The selection of which data was used for training and for testing is done randomly using the sk learn function shuffle [44] to prevent experimenters' bias from impacting the resulting model. This also means the resulting coefficient of determination R2 can differ each time a new model is created. The R2 value will also be reported for each MLP model used to generate results through performance prediction.

3.5.1 Optimizing using MLP and differential evolution

After creating an MLP model that can successfully predict the performance of a given RVE, this model could be used to find a global best performing RVE given a set of requirements. In this case, the function f($\mathbf{X}$) that was optimized was

$$f(\boldsymbol{X}) = \frac{D\overline{E}_0(\boldsymbol{X})}{a(\boldsymbol{X})} + N_{onset}(\boldsymbol{X}), \tag{3.18}$$

where $\mathbf{X} = (R_1, R_2, f_v, \varepsilon)$; $\overline{E}$, a and N_{onset} is predicted from the MLP model; and D is the degradation of $\overline{E}$.

To optimize the function f($\mathbf{X}$), the differential evolution technique as introduced in section

2.8 from the scipy library [45] was used.

The code used for the optimization of the function f($\mathbf{X}$) is made available in the appendix section A.2.

3.5.2 Optimizing using Abaqus and differential evolution

Optimizing using the differential evolution method, combined with the Abaqus solver. I.e., instead of the MLP model predicting the result for each combination of factors the differential evolution algorithm asks to evaluate, the result is calculated by the Abaqus solver. This takes more time (days instead of seconds), and is risky as the differential evolution optimizer does not handle interruptions. Because of this inability to handle interruptions, the structures where each given default material parameters in case of error in the simulation. In this case, the fault would be reported. In the end result, no such faults occurred. Script is available in Appendix, section A.3. This was done for both fixed strain and stress. Different outcomes were expected as effective stiffness varies greatly.

3.6 Simplifications

Simplifications made in this thesis are: a is representative for the slope of the E-degradation curve for the area of interest: 15% - 30% degradation.

The fact that the FE-model cannot produce a reliable result in the are before 10% degradation is not detrimental to the experiments conducted.

The convergence-study results are representative for the entire scope of RVE-parameters covered in this thesis.

Material contribution is not considered.

The effects these simplifications may have on the result will be discussed at the end of this thesis.

Chapter 4

Results

In this section, the results from the various methods will be presented.

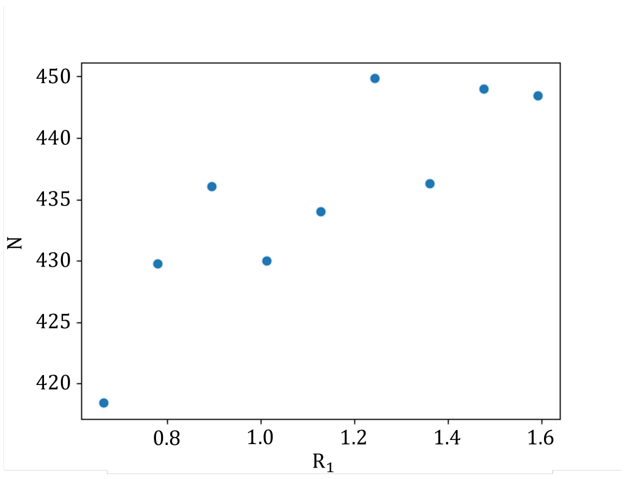
4.1 Parametric study

4.1.1 OFAT

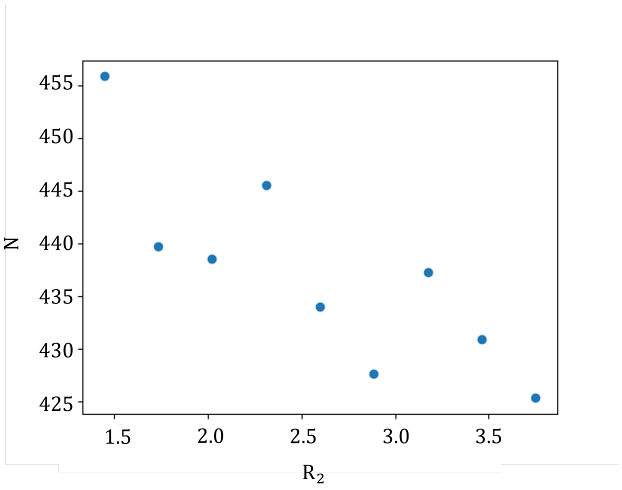
Results: Graphical display shows: minimum fv, minimum R2, and particular R1

The results from the OFAT analysis of the RVE structure are presented graphically in figures 4.1a, 4.1b and 4.1c.

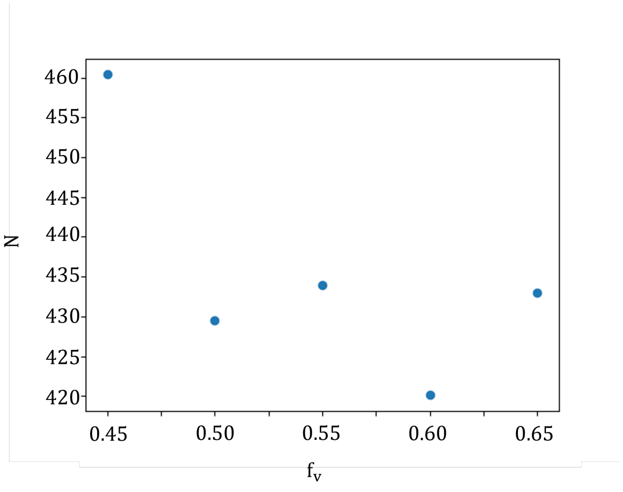
The response N before a 15 % degradation of $\overline{E}$ was reached, using the OFAT approach is presented in figure 4.1a, 4.1b and 4.1c. As can be seen from the graphical displays, the predicted optimal yield is granted by minimizing R2 and f_v, and setting $R_1 \approx 1.24$. This would predictably lead to a lifetime $N > 460$, as is the highest recorded using the OFAT.



(a) OFAT R_1



(b) OFAT R_2



(c) OFAT f_v

Figure 4.1: OFAT plots for R_1, R_2 and f_v.

4.1.2 Full factorial

The results from the full factorial design (described in sections 2.10 and 3.4) are presented in table 4.2. The factor level corresponding to each sign in table 4.2 and each factor is described in table 4.1. Take e.g row number 4 (excluding the header) in table 4.2: R_1=0, R_2=0, f_v=+, N15%=420. This means that an RVE with factor levels R_1=1.128, R_2=2.598 and f_v=0.6, reaches 15% degradation of the effective stiffness, $\overline{E}$, at N cycles = 420.

Table 4.1: Factorial design symbols

symbol	Factor level		
	R_1	R_2	f_v
0	1.128	2.598	0.55
+	1.36	3.174	0.60
-	0.896	2.022	0.50

The values in table 4.2 can be used to better understand the interaction between the factors R_1, R_2, and f_v [38]. The interaction between R_1 and R_2 is examined in figure 4.2. In this plot the effect of altering R_1 changes with different levels of R_2. I.e. the lines in the plots aren't parallel, indicating an interaction between the two factors. The values in the plot are the average of all three f_vs for each point.

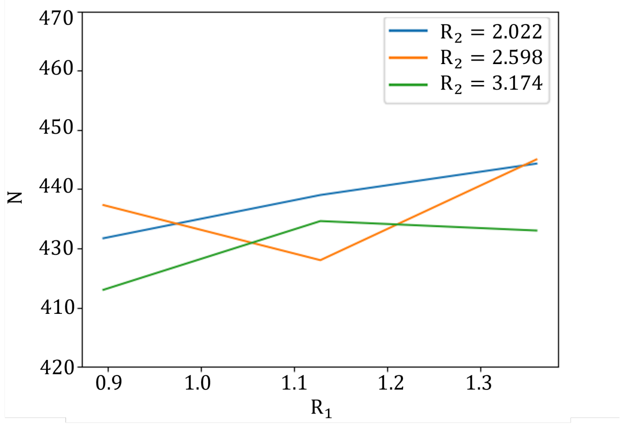


Figure 4.2: Plot for R_1-R_2 interaction. Values are the mean for all f_v.

Next, the interaction between R_1, R_2, and f_v are examined. In figure 4.3 the behavior of R_1 for different R_2 and f_v are shown. It can be seen that changing f_v has a noticeable effect on

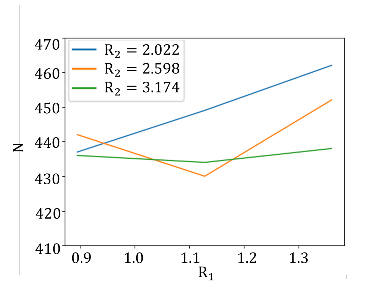
Table 4.2: Factorial design

R1	R2	f_v	N 15%
0	0	0	434
+	0	0	436 (+)
0	+	0	437 (+)
0	0	+	420 (-)
0	+	+	433 (-)
+	+	0	428 (-)
+	0	+	447 (+)
+	+	+	433 (-)
-	0	0	436 (+)
0	-	0	439 (+)
0	0	-	430 (-)
0	-	-	449 (+)
-	-	0	428 (-)
-	0	-	442 (+)
-	-	-	437 (+)
-	+	+	414 (-)
+	-	+	431 (-)
+	+	-	438 (+)
+	-	-	462 (+)
-	-	+	430 (-)
-	+	-	436 (+)
-	0	+	434
0	-	+	429 (-)
0	+	-	434
+	-	0	440 (+)
+	0	-	452 (+)
-	+	0	420 (-)

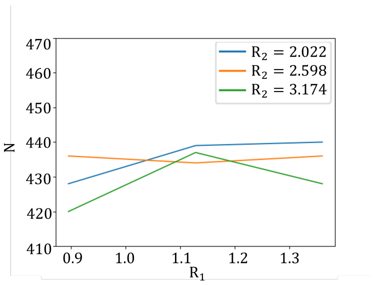
the behavior of R_1 and R_2. It is, therefore, reasonable to assume interaction between the three factors.

In figure 4.4, R_2 and f_v are switched. Let's first look at figure 4.4a. The lines for each individual volume fraction are not parallel, as seen in figure 4.3. As the f_v affects the behavior of R_1 significantly, it would be reasonable to assume interaction between the two. The change in the R1-f_v trends between the different R2 levels, in combination with the trends seen in figure 4.4a, makes it clear that both f_v and R_2 have a non-negligible interaction with R_1.

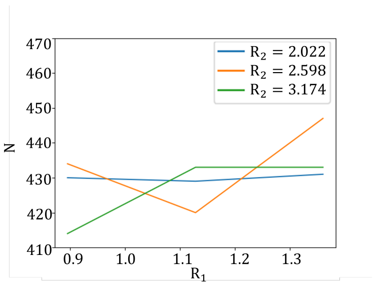
In figure 4.5 the data from the full factorial design in table 4.2 is reorganized, so the interaction



(a) R_1-R_2 interaction for f_v = 0.5.

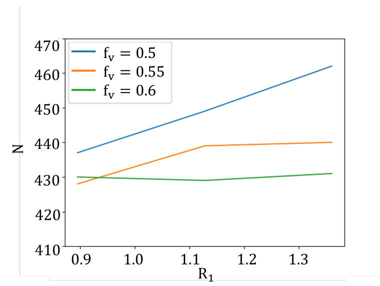


(b) R_1-R_2 interaction for f_v = 0.55.

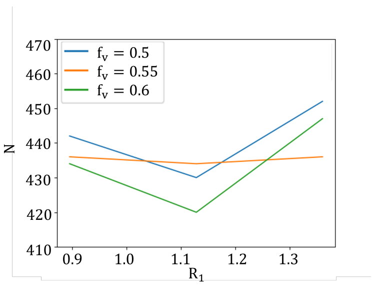


(c) R_1-R_2 interaction for f_v = 0.60.

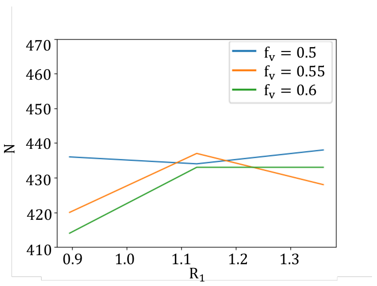
Figure 4.3: R_1-R_2 interaction plots for different f_v.



(a) R_1-f_v interaction for R_2 = 2.02.



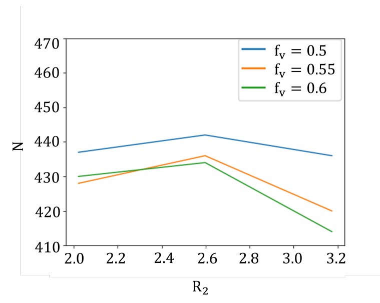
(b) R_1-f_v interaction for R_2 = 2.598.



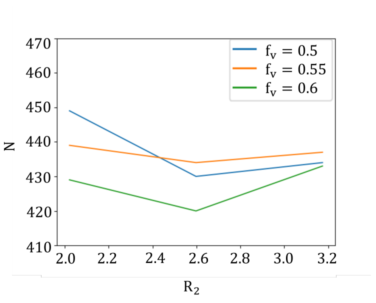
(c) R_1-f_v interaction for R_2 = 3.174.

Figure 4.4: R_1-f_v interaction plots for different R_2.

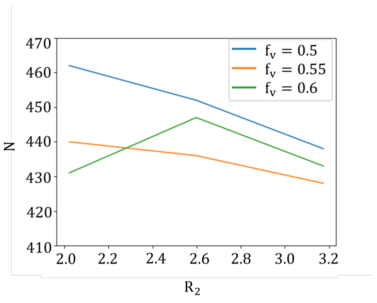
between factors R_2 and f_v can be investigated. The trend lines for the different f_v levels in, e.g. figure 4.5a are different. The distinction is arguably more pronounced with the relations R_1-R_2 and R_1-Vf, but still not negligible, especially when considering figure 4.5c, where the difference is clear.



(a) R_2-f_v interaction for R_1 = 0.896.



(b) R_2-f_v interaction for R_1 = 1.128.



(c) R_2-f_v interaction for R_1 = 1.36.

Figure 4.5: R_2-R_1 interaction plots for different f_v.

4.1.3 Split plot

The result of the Split plot using two values can be seen in table 4.4. The meaning of the symbols in table 4.4 is explained in table 4.3. For comparison the values for the mean values, the middle values of the values in table 4.3 are $R_1 = 1.13$, $R_2 = 2.598$, $f_v = 0.55$. They yield N=434.

Table 4.3: Split Plot symbols

symbol	Factor level		
	R_1	R_2	f_v
+	1.593	3.751	0.65
-	0.663	1.445	0.45

Table 4.4: Split Plot

		R2	
R1	f_v	+	-
-	-	435 (+)	444 (+)
-	+	417 (-)	435 (+)
+	-	432 (-)	429 (-)
+	+	424 (-)	461 (+)

4.2 Brute Force

Of the 770 data points collected for the MLP-model, the best result was yielded for the factors presented in table 4.5. The resulting figure is presented in figure 4.6. it took 21 hours to complete.

Table 4.5: Optimal factors as predicted by differential evolution for ε=0.8 %.

R_1	R_2	f_v	N
1.477	2.31	0.5	472

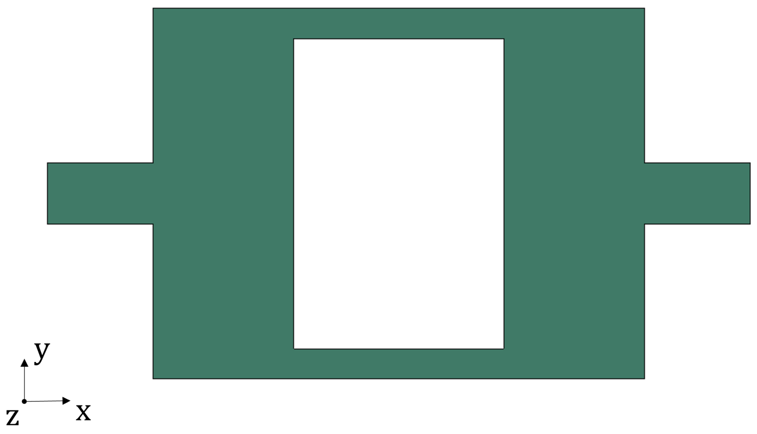


Figure 4.6: Illustration of structure with optimal factors as predicted by brute force for ε=0.8%.

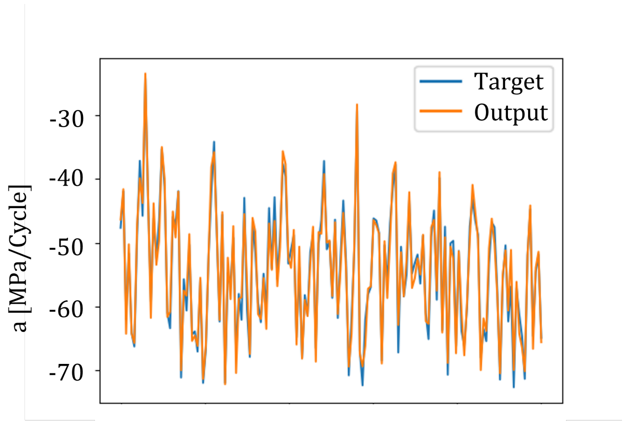
4.2.1 MLP and optimization

MLP

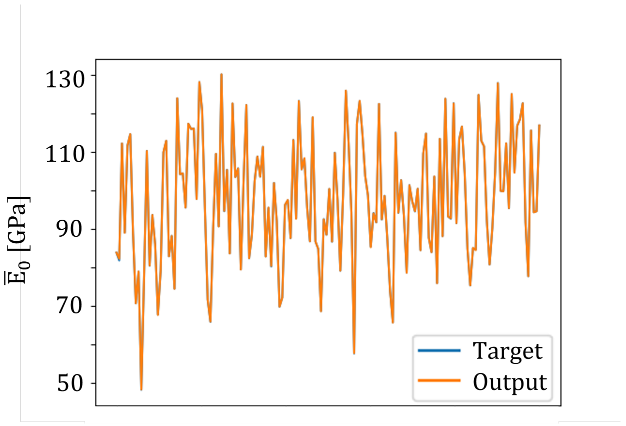
In this section, the result of the MLP will be presented. The best R2 value (not to be confused with the factor R_2) 2.8 reached for the regression model using the Adam optimizes (see section 2.8) prediction all three factors a, N, and $\overline{E}_0$ was 0.96.

The data set collected contained only 770 data points. This is a relatively small sample, indicating that the lbfgs optimizer could be a better fit [33][31]. Changing the optimizer from adam to lbfgs yielded an R2 value of 0.97. The R2 value for the three individual outputs is 0.95, 0.96 and 1 for N, a, and E, respectively.

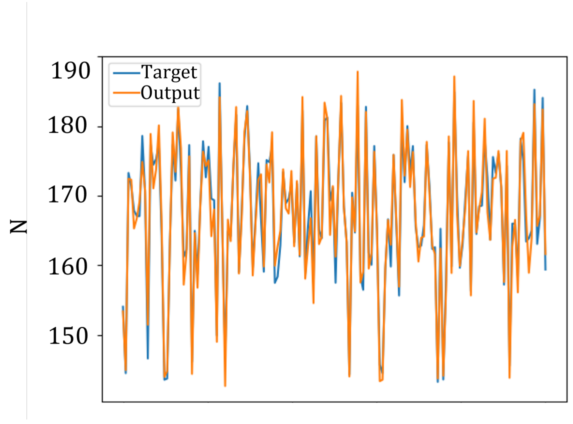
A graphical display of the target-output plots using the lbfgs-optimizer is presented in figure 4.7. The subplots show that the prediction for $\overline{E}_0$ is better than for a and N. This level of accuracy was accomplished by scaling the data using a standard scaler [46], setting the tolerance of the trainer to 1e-12, and setting the Y = [N_{onset}, a, $\overline{E}_0$], as discussed in section 3.5. For 300 data points, the R2 for the regression model predicting all three values was 0.93 using the adam-optimizer. Using Y = [N_{onset}, a/$\overline{E}_0$] or Y = [N ($\overline{E}$=0.85$\overline{E}_0$)] resulted in a much worse prediction, worst-case 0.65 and 0.53 respectively for 770 data points. When using original data for X and Y instead of scaling, the R2 value was worst-case 0.83 for 770 data points.



(a) Target-output plot for factor a. R2=0.9654.



(b) Target-output plot for factor $\overline{E}_0$. R2=1.0



(c) Target-output plot for factor N. R2 = 0.958.

Figure 4.7: Target-output plots for a, N, and E from the MLP regression model using the lbfgs optimizer. Data is shuffled, and the value on the x-axis only denotes the relative placement in the shuffled vector.

Optimizing

The optimal factors, as predicted by the differential evolution in combination with the MLP-model, are presented in table 4.6. The resulting structure with these factors is illustrated in figure 4.8. The number of evaluations performed by the optimizer was 1548, and the number of iterations was 21.

Table 4.6: Optimal factors as predicted by differential evolution for ε=0.8 %.

R_1	R_2	f_v	N
2.46136337	1.02242361	0.5718927	506

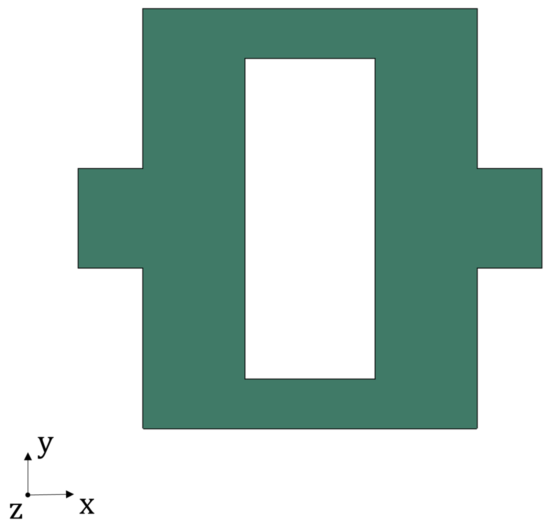


Figure 4.8: Illustration of structure with optimal factors as predicted by differential evolution for ε=0.8%.

4.3 Result from Differential Evolution using Abaqus directly

In this section, the resulting optimal factors and time and resources spent from using the differential evolution with the Abaqus solver directly are presented.

4.4 Fixed Strain

For the fixed strain of 0.8 %, the optimal factors are presented in table 4.7. The resulting structure is illustrated in figure 4.9. The number of evaluations performed by the optimizer was 904, and the number of iterations was 14. It took 17 hours to complete.

Table 4.7: Optimal factors as predicted by differential evolution for ε=0.8 %.

R_1	R_2	f_v	N
2.43	1.036	0.56	508

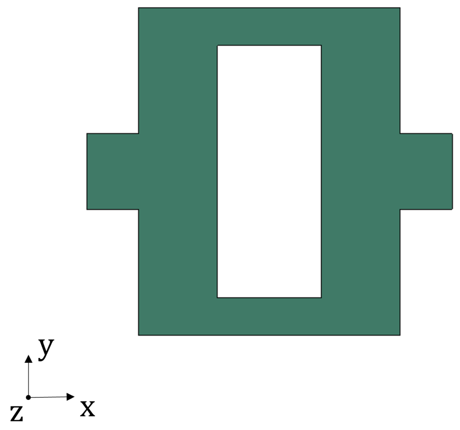


Figure 4.9: Illustration of structure with optimal factors as predicted by differential evolution for ε=0.8%.

4.5 Fixed Stress

Preliminary results for the fixed, effective stress of 300 MPa, the optimal factors are presented in table 4.8. The resulting structure is illustrated in 4.10. The number of evaluations performed by the optimizer was 891, and the number of iterations was 12. It took 38 hours to complete. Due to the time-demand on the analysis, a final result was not reached. The error was not enough cycles was included in the analysis, and so higher-performing structures were disregarded. Analyzing the data points to higher R_2 values being better performers, but further investigations are required.

Table 4.8: Optimal factors as predicted by differential evolution for ε=0.8 %.

R_1	R_2	f_v	N
3.32	4.14	0.55	2682

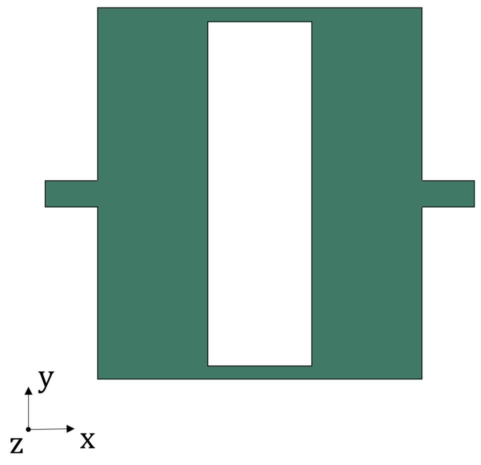


Figure 4.10: Illustration of structure with optimal factors as predicted by differential evolution for σ=300 MPa.

Chapter 5

Discussion

In this chapter, the various results from the last chapter will be discussed. The focus will be on the sources of error and what can be read from these results, if there is anything of interest, and why that is or is not the case.

5.1 What can be learned from the results?

For understanding the interaction between the different parameters, the full factorial method was the most efficient, and good for visualization and presentation. Using it to find an optimum structure, was however difficult, and not labor efficient with regards to man-hours. The OFAT was not appropriate for this structure, as there was a high degree of interaction between the different factors.

Using the machine-learning algorithm and the Differential Evolution equation to predict the optimum set of parameters, was more efficient, but required more computational power as several hundreds of jobs had to be created beforehand. It was, however, robust, in the sense that the process could stop and start without having lost several hours of work. The optimum result N=506 was also significantly higher than the value found from the data set it was based on N=472, proving its efficacy.

Using the differential evolution directly on the Abaqus solver, was the least robust alternative. With the computational power accessed by the author, the algorithm had to run uninterrupted for 2 days, and if interrupted would have to start from scratch. It also did not require fewer jobs than the MLP combined with DE, but was perhaps more reliable as it did not have authors' bias when it came to data selection.

5.1.1 The structure and the method of study

The structure is predicted to have an optimum set of parameters around $R_1 = 2.46$, $R_2=1.02$ and $f_v=0.57$ by the MLP combined with DE, and $R_1 = 2.43$, $R_2=1.04$ and $f_v=0.56$ for DE combined with Abaqus solver for fixed strain.

The differential evolution combined with the MLP-model provided the same result as the DE with Abaqus. As the MLP was more robust with regard to interruptions, this could be a good alternative as long as the predictions of the MLP are good enough. That being said, when the DE optimization algorithm combined with Abaqus was not interrupted, it was more time economical than the MLP, requiring 17 hours instead of 21. This is probably because the DE solver converges towards the RVE with the highest N. The number of cycles each RVE is analyzed for is fixed, so the longer it takes before damage initiates, the fewer resources are required to complete the job. As the DE focus on the RVEs with higher N, it selects the Abaqus jobs that require less time, while for the MLP the jobs are distributed in the population regardless of N.

The preliminary, optimal structure for fixed stress points to different factors and lifetime from the fixed strain. This is probably largely caused by the fact that the effective stiffness depends on the factors. This is arguably a more useful result for load-bearing structures, but further investigations are needed.

5.2 Further investigations

If there were to be any further investigations on this, or if there was more time, the next step could be to continue the investigation of optimal structures for a fixed effective stiffness, or for fixed stress. This is more interesting for load-bearing structures. It would also be advisable to investigate the validity of the finite element model using physical experiments with FDM-printed structures and a material model created for the FDM-material. This was, unfortunately, not something I managed to accomplish in the allotted time.

If further investigations were to be made, however, it would be strongly advised to allocate better computational resources to save time.

A suggestion for further investigation would also be to investigate finite element analysis for PBC combined with the cyclic hardening material model and hysteresis energy damage initiation and evolution. I suggest this, as my investigations show the expected results using the PBC implemented alone, as well as the material and damage model alone, but combined they do not function as they should as seen in the convergence study conducted.

5.3 Sources of error

The factors. It could have been better if one of the factors were t_1 and t_{v1}, as that would have made it possible to directly control the amount of material carrying load, and thus the effective stiffness.

Not using randomized data for the construction of the machine learning. The data collected for the MLP-model should have been randomized, not chosen to ensure a broad distribution. This could result in experimenters' bias.

From the convergence study, it is clear that the results are too varied for the onset of degradation, and should therefore only be used after a certain point. This is approximately where $\overline{E} = 0.9\overline{E}_0$.

No physical experiments were conducted, so there has not been any validation of the model created, other than the ones described in this thesis.

The range of the R_2 value was to limited and should have been expanded to ensure a global result.

Bibliography

[1] J. R. Grubich. “Disparity between Feeding Performance and Predicted Muscle Strength in the Pharyngeal Musculature of Black Drum, Pogonias cromis(Sciaenidae)”. In: *Environmental Biology of Fishes* 74.3 (2005), pp. 261–272. ISSN: 1573-5133. DOI: `10.1007/s10641-005-3218-0`. URL: `https://doi.org/10.1007/s10641-005-3218-0`.

[2] E. Ziv et al. “Neither cortical nor trabecular: An unusual type of bone in the heavy-load-bearing lower pharyngeal jaw of the black drum (Pogonias cromis)”. In: *Acta Biomaterialia* 104 (2020), pp. 28–38. ISSN: 1742-7061. DOI: `https://doi.org/10.1016/j.actbio.2020.01.001`. URL: `https://www.sciencedirect.com/science/article/pii/S1742706120300027`.

[3] B. K. Hall. “Chapter 2 - Bone”. In: *Bones and Cartilage (Second Edition)*. Ed. by B. K. Hall. Second Edition. San Diego: Academic Press, 2015, pp. 17–42. ISBN: 978-0-12-416678-3. DOI: `https://doi.org/10.1016/B978-0-12-416678-3.00002-1`. URL: `https://www.sciencedirect.com/science/article/pii/B9780124166783000021`.

[4] A. Atkins et al. “The three-dimensional structure of anosteocytic lamellated bone of fish”. In: *Acta Biomaterialia* 13 (2015), pp. 311–323. ISSN: 1742-7061. DOI: `https://doi.org/10.1016/j.actbio.2014.10.025`. URL: `https://www.sciencedirect.com/science/article/pii/S174270611400467X`.

[5] M. Monier-Faugere, M. Chris Langub, and H. H. Malluche. “Chapter 8 - Bone Biopsies: A Modern Approach”. In: *Metabolic Bone Disease and Clinically Related Disorders (Third Edition)*. Ed. by Louis V. Avioli and Stephen M. Krane. Third Edition. San Diego: Academic Press, 1998, 237–280e. ISBN: 978-0-12-068700-8. DOI: `https://doi.org/10.1016/B978-012068700-8/50009-8`. URL: `https://www.sciencedirect.com/science/article/pii/B9780120687008500098`.

[6] G. J. Tortora. *Principles of Human Anatomy*. Sixth edition. New York: John Wiley Son, 2002.

[7] A. A. Abdel-Wahab, A. R. Maligno, and V. V. Silberschmidt. "Micro-scale modelling of bovine cortical bone fracture: Analysis of crack propagation and microstructure using X-FEM". In: *Computational Materials Science* 52.1 (2012). Proceedings of the 20th International Workshop on Computational Mechanics of Materials - IWCMM 20, pp. 128–135. ISSN: 0927-0256. DOI: `https://doi.org/10.1016/j.commatsci.2011.01.021`. URL: `https://www.sciencedirect.com/science/article/pii/S0927025611000450`.

[8] M. Buehler R. Ritchie and P. Hansma. "Plasticity and toughness in bone". In: *Physics Today - PHYS TODAY* 62 (June 2009). DOI: `10.1063/1.3156332`.

[9] C. E. Ramírez A et al. "Assessing mechanical behavior of ostrich and equine trabecular and cortical bone based on depth sensing indentation measurements". In: *Journal of the Mechanical Behavior of Biomedical Materials* 117 (2021), p. 104404. ISSN: 1751-6161. DOI: `https://doi.org/10.1016/j.jmbbm.2021.104404`. URL: `https://www.sciencedirect.com/science/article/pii/S175161612100093X`.

[10] Christopher Boyle and Il Yong Kim. "Three-dimensional micro-level computational study of Wolff's law via trabecular bone remodeling in the human proximal femur using design space topology optimization". In: *Journal of Biomechanics* 44.5 (2011), pp. 935–942. ISSN: 0021-9290. DOI: `https://doi.org/10.1016/j.jbiomech.2010.11.029`. URL: `https://www.sciencedirect.com/science/article/pii/S002192901000655X`.

[11] M.F Ashby. "The properties of foams and lattices". eng. In: *Philosophical transactions of the Royal Society of London. Series A: Mathematical, physical, and engineering sciences* 364.1838 (2006), pp. 15–30. ISSN: 1364-503X.

[12] P. Xiao et al. "Can DXA image-based deep learning model predict the anisotropic elastic behavior of trabecular bone?" In: *Journal of the Mechanical Behavior of Biomedical Materials* 124 (2021), p. 104834. ISSN: 1751-6161. DOI: `https://doi.org/10.1016/j.jmbbm.2021.104834`. URL: `https://www.sciencedirect.com/science/article/pii/S1751616121004756`.

[13] *2.2.3 Direct Cyclic Algorithm*. URL: `http://130.149.89.49:2080/v6.13/books/stm/default.htm`.

[14] *Direct cyclic analysis*. URL: `http://130.149.89.49:2080/v6.13/books/usb/default.htm?startat=pt03ch06s02at05.html#usb-anl-adirectcyclic`.

[15] *The extended finite element method (XFEM)*. URL: `https://abaqus-docs.mit.edu/2017/English/SIMACAECAERefMap/simacae-c-engconcxfemoverview.htm`.

[16] *Modeling discontinuities as an enriched feature using the extended finite element method*. URL: `https://abaqus-docs.mit.edu/2017/English/SIMACAEANLRefMap/simaanl-c-enrichment.htm#simaanl-c-enrichment-t-ApplyingCohesiveMaterialConceptsToXFEMbased sma-topic13`.

[17] *Low-cycle fatigue criterion*. URL: `http://130.149.89.49:2080/v6.13/books/usb/default.htm?startat=pt04ch11s04aus69.html#usb-anl-acrackpropagation-fatigue`.

[18] *Low-cycle fatigue analysis using the direct cyclic approach*. URL: `https://abaqus-docs.mit.edu/2017/English/SIMACAEANLRefMap/simaanl-c-directcyclicfatigue.htm#simaanl-c-directcyclicfatigue-t-ProgressiveDamageAndDamageExtrapolationInBulkDu sma-topic4`.

[19] *Damage initiation for ductile materials in low-cycle fatigue*. URL: `https://abaqus-docs.mit.edu/2017/English/SIMACAEMATRefMap/simamat-c-damageinitfatigue.htm`.

[20] *Damage evolution for ductile materials in low-cycle fatigue*. URL: `https://abaqus-docs.mit.edu/2017/English/SIMACAEMATRefMap/simamat-c-damageevolfatigue.htm`.

[21] J. Lemaitre and J.-L- Caboche. *Mechanics of solid materials*. Cambridge, United Kingdom: Cambridge University Press, 1990. Chap. 5.

[22] *4.3.5 Models for metals subjected to cyclic loading*. URL: `http://130.149.89.49:2080/v6.13/books/stm/default.htm`.

[23] M. Danielsson, D.M. Parks, and M.C. Boyce. "Three-dimensional micromechanical modeling of voided polymeric materials". In: *Journal of the Mechanics and Physics of Solids* 50.2 (2002), pp. 351–379. ISSN: 0022-5096. DOI: `https://doi.org/10.1016/S0022-5096(01)00060-6`. URL: `https://www.sciencedirect.com/science/article/pii/S0022509601000606`.

[24] M. Okereke and S. Keates. *Finite Element Applications, A Practical Guide to the FEM Process*. Cham, Switzerland: Springer International Publishing AG, 2018. Chap. 8.

[25] M. Danielsson. “Micromechanics, macromechanics and constitutive modeling of the elasto-viscoplastic deformation of rubber-toughened glassy polymers”. PhD thesis. Massachusetts Institute of Technology, 2003.

[26] *Numerikk*. URL: `https://wiki.math.ntnu.no/tma4100/tema/numerics?&#numerisk_integrasjon`.

[27] *Numerical integration: Introduction*. URL: `https://www.math.ntnu.no/emner/TMA4130/2021h/lectures/SimpleQuadrature.pdf`.

[28] S. Abirami and P. Chitra. “Chapter Fourteen - Energy-efficient edge based real-time healthcare support system”. In: *The Digital Twin Paradigm for Smarter Systems and Environments: The Industry Use Cases*. Ed. by Pethuru Raj and Preetha Evangeline. Vol. 117. Advances in Computers 1. Elsevier, 2020, pp. 339–368. DOI: `https://doi.org/10.1016/bs.adcom.2019.09.007`. URL: `https://www.sciencedirect.com/science/article/pii/S0065245819300506`.

[29] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. eng. In: (2014).

[30] J. Nocedal. “Updating quasi-Newton matrices with limited storage”. eng. In: 35.151 (1980), pp. 773–782. ISSN: 0025-5718.

[31] D. C. Liu and J. Nocedal. “On the limited memory BFGS method for large scale optimization”. eng. In: *Mathematical programming* 45.3 (1989), pp. 503–528. ISSN: 0025-5610.

[32] R. E. Walpole et al. *Probability statistics for engineers and scientists*. eng. 9th ed. Harlow: Pearson Education, 2016. ISBN: 978-1-292-16136-5.

[33] *sklearn.neural_network.MLPRegressor*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor.score`. accessed: 24.05.2022.

[34] M. F. Ahmad et al. “Differential evolution: A recent review based on state-of-the-art works”. In: *Alexandria Engineering Journal* 61.5 (2022), pp. 3831–3872. ISSN: 1110-0168. DOI: `https://doi.org/10.1016/j.aej.2021.09.013`. URL: `https://www.sciencedirect.com/science/article/pii/S111001682100613X`.

[35] R. Storn and K. Price. “Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”. eng. In: *Journal of global optimization* 11.4 (1997), pp. 341–359. ISSN: 0925-5001.

[36] C. Yuangyai and H.B. Nembhard. “Chapter 8 - Design of Experiments: A Key to Innovation in Nanotechnology”. In: *Emerging Nanotechnologies for Manufacturing*. Ed. by Waqar Ahmed and Mark J. Jackson. Micro and Nano Technologies. Boston: William Andrew Publishing, 2010, pp. 207–234. ISBN: 978-0-8155-1583-8. DOI: https://doi.org/10.1016/B978-0-8155-1583-8.00008-9. URL: https://www.sciencedirect.com/science/article/pii/B9780815515838000089.

[37] T. P. Ryan and J. P. Morgan. “Modern Experimental Design”. In: *Journal of statistical theory and practice.* 1.3-4 (2007), pp. 501–506. ISSN: 1559-8608.

[38] A. Dean, D. Voss, and D. Draguljić. *Design and Analysis of Experiments*. eng. Springer texts in statistics. Cham: Springer International Publishing AG, 2017. ISBN: 3319522485.

[39] J. Antony. “6 - Full Factorial Designs”. In: *Design of Experiments for Engineers and Scientists (Second Edition)*. Ed. by Jiju Antony. Second Edition. Oxford: Elsevier, 2014, pp. 63–85. ISBN: 978-0-08-099417-8. DOI: https://doi.org/10.1016/B978-0-08-099417-8.00006-7. URL: https://www.sciencedirect.com/science/article/pii/B9780080994178000067.

[40] A. K. Das and S. Dewanjee. “Chapter 3 - Optimization of Extraction Using Mathematical Models and Computation”. In: *Computational Phytochemistry*. Ed. by Satyajit D. Sarker and Lutfun Nahar. Elsevier, 2018, pp. 75–106. ISBN: 978-0-12-812364-5. DOI: https://doi.org/10.1016/B978-0-12-812364-5.00003-1. URL: https://www.sciencedirect.com/science/article/pii/B9780128123645000031.

[41] G. Box and S. Jones. “SPLIT PLOTS FOR ROBUST PRODUCT AND PROCESS EXPERIMENTATION”. eng. In: *Quality engineering* 13.1 (2001), pp. 127–134. ISSN: 0898-2112.

[42] K. H. Engen. “Data driven approach to bio-inspired structures”. In: (2021).

[43] W. Song et al. “Low-Cycle Fatigue Life Prediction of 10CrNi3MoV Steel and Undermatched Welds by Damage Mechanics Approach”. In: *Frontiers in Materials* 8 (2021). ISSN: 2296-8016. DOI: 10.3389/fmats.2021.641145. URL: https://www.frontiersin.org/article/10.3389/fmats.2021.641145.

[44] *sklearn.neural_network.MLPRegressor*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.utils.shuffle.html. (accessed: 24.05.2022).

[45] *scipy.optimize.differential$_e$volution*. URL: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html`. (accessed: 24.05.2022).

[46] *sklearn.preprocessing.StandardScaler*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`. (accessed: 22.05.2022).

Appendix

Table of Contents

Appendix A

A.1 Methods

A.1.1 Convergence study

Table A.1: Convergence study: R_1, $R_2 = 0.5$, $\varepsilon = 0.008$, N=1000.

n	Elements	Time [s]	Δ w (N=1000)	d Δ w
5	34	50.35 s	0.936	
10	136	2 m 0.83 s	0.958	2.3 %
20	544	5 m 55.75 s	0.934	2.5 %
40	2160	874.43	0.928	0.6 %

Table A.2: Convergence study: R_1, $R_2 = 1$, $\varepsilon = 0.006$, N=1000.

n	Elements	Time [s]	Δ w (N=1000)	d Δ w
5	24	38.11 s	0.842	
10	118	0 m 58.22 s	1.128	34 %
20	408	1 m 58.41 s	1.176	4.3 %
40	1632	450.22 s	1.163	1.1 %

Table A.3: Convergence study: R_1, $R_2 = 1$, $\varepsilon = 0.008$, N=1000.

n	Elements	Time [s]	Δ w (N=1000)	d Δ w
5	24	0 m 38.09 a	0.945	
10	118	1 m 28.42 s	1.727	82.8 %
20	408	3 m 51.33 s	1.823	5.6 %
40	1632	676.47	1.82	0.2%

Table A.4: Convergence study: R_1, R_2 = 2, ε = 0.006, N=1000.

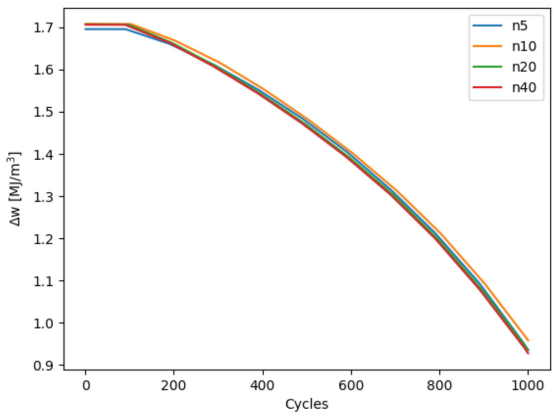
n	Elements	Time [s]	Δ w (N=1000)	d Δ w
5	18	32.05 s	1.151	
10	78	40.28 s	1.525	32.5 %
20	312	1 m 18.29	1.573	3.1 %
40	1170	307.87 s	1.579	0.4 %

Table A.5: Convergence study: R_1, R_2 = 2, ε = 0.008, N=1000.

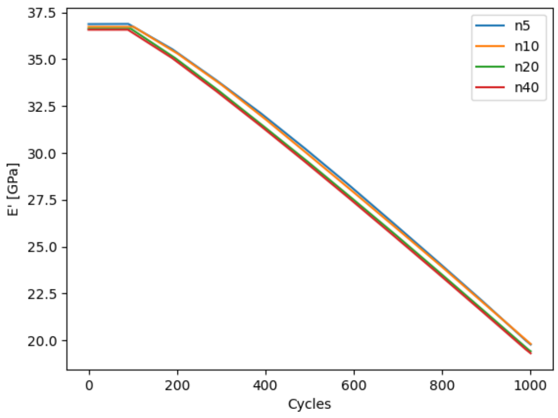
n	Elements	Time [s]	Δ w (N=1000)	d Δ w
5	18	34.47 s	1.305	
10	78	48.58 s	2.287	75.2 %
20	312	1 m 38.83 s	2.373	3.8 %
40	1170	306.66	2.418	1.90%

Table A.6: Convergence study: R_1, R_2 = 3, ε = 0.008, N=1000.

n	Elements	Time [s]	Δ w (N=1000)	d Δ w
5	20	36.44	1.218	
10	76	42.35	2.807	30.5 %
20	224	60.41	2.666	5 %
40	904	234.75	2.483	6.80%

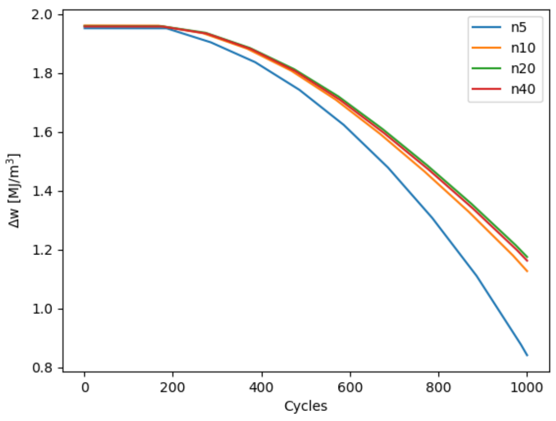


(a) Plastic strain energy, Δw

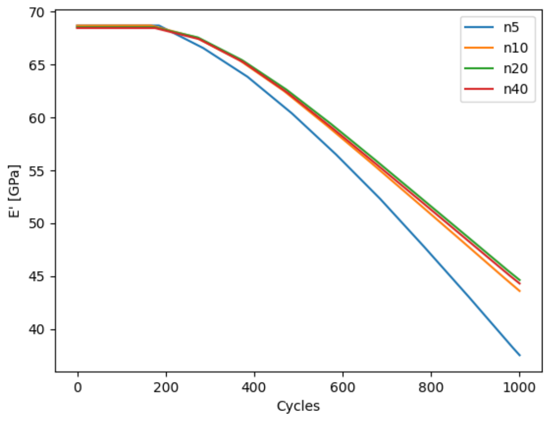


(b) Effective Stiffness, E'

Figure A.1: Convergence study: R_1, R_2 = 0.5, ε = 0.008, N=1000.

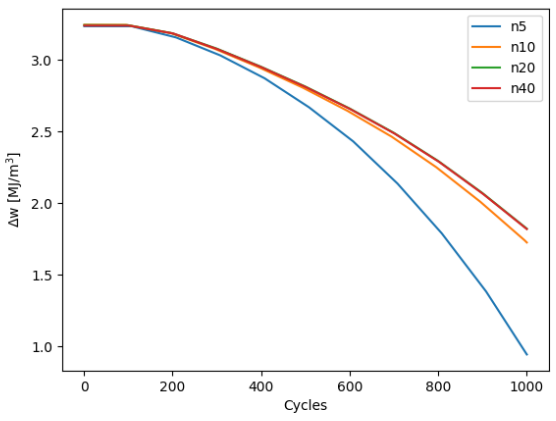


(a) Plastic strain energy, Δw

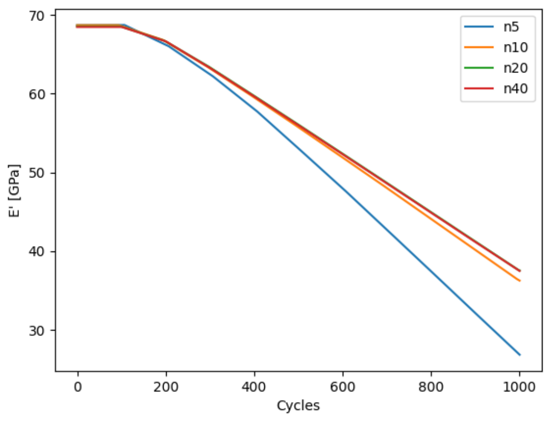


(b) Effective Stiffness, E'

Figure A.2: Convergence study: R_1, R_2 = 1, ε = 0.006, N=1000.

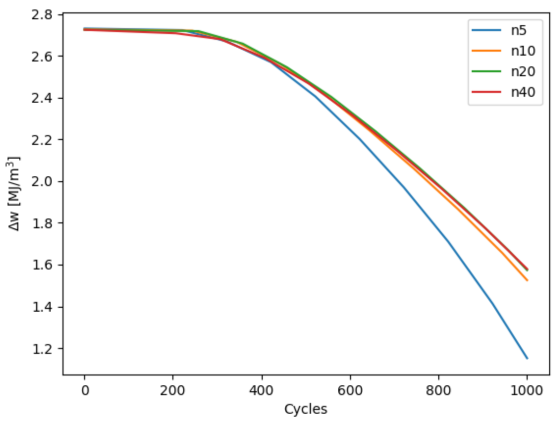


(a) Plastic strain energy, Δw

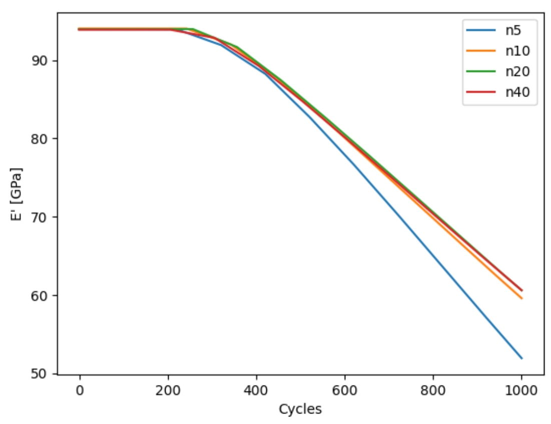


(b) Effective Stiffness, E'

Figure A.3: Convergence study: R_1, R_2 = 1, ε = 0.008, N=1000.

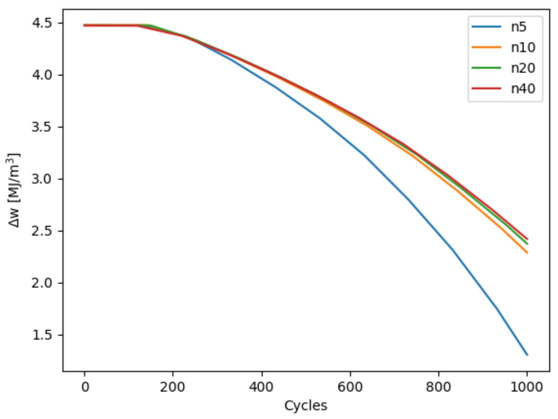


(a) Plastic strain energy, Δw

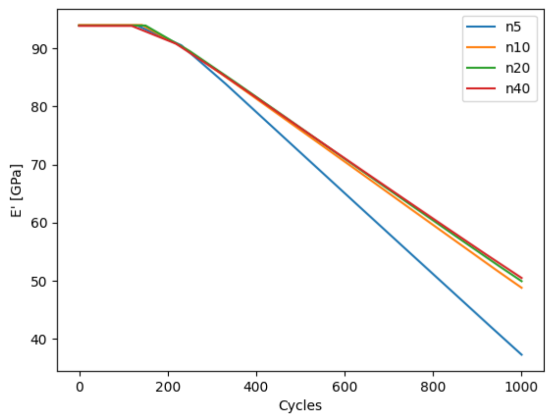


(b) Effective Stiffness, E'

Figure A.4: Convergence study: R_1, $R_2 = 2$, $\varepsilon = 0.006$, N=1000.

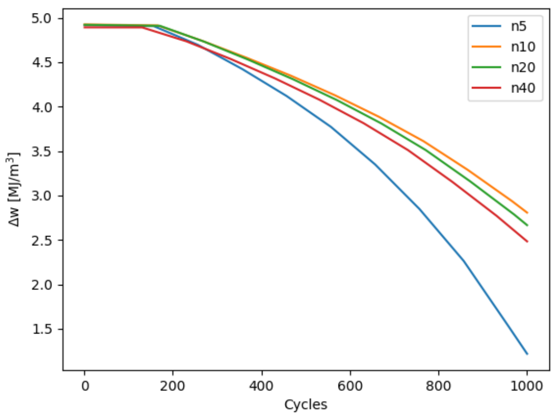


(a) Plastic strain energy, Δw

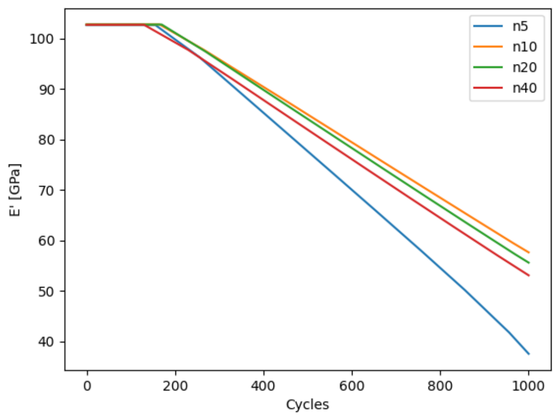


(b) Effective Stiffness, E'

Figure A.5: Convergence study: R_1, $R_2 = 2$, $\varepsilon = 0.008$, N=1000.



(a) Plastic strain energy, Δw



(b) Effective Stiffness, E'

Figure A.6: Convergence study: R_1, R_2 = 3, ε = 0.008, N=1000.

A.2 Code for Machine learning and Optimization

```python
import scipy.optimize as optimize
from MachineLearning import doTheThing_0
from sklearn.preprocessing import StandardScaler
import numpy as np

run = 0
D = 0.15
regr, RScore, ScalerX, ScalerY = doTheThing_0()


def f(T):
    global run
    global D
    T = ScalerX.transform(T.reshape(1,-1))
    run += 1
    output = ScalerY.inverse_transform(regr.predict(T))
    N, a, E = output[0,0], output[0,1], output[0,2]
    n_diff = -(D*E) / a
    N_goal = N + n_diff
    #output[0,0] = N_goal
    #output = ScalerY.transform(output.reshape(1,-1))
    return -N_goal

def findOptimum():
    Upper= np.array([3.5, 3.5,0.65, 8]) #np.array([100, 37.85,122.15,
     ↪ 136.48])#
    Lower = np.array([ 0.5, 0.5,0.45,8 ])#np.array([ 50, 16.89, 66.09,
     ↪ 75.9])#

    bnds = [(Lower[0], Upper[0]), (Lower[1], Upper[1]), (Lower[2],
     ↪ Upper[2]), (Lower[3], Upper[3])]

    result = optimize.differential_evolution(f, bounds=bnds)

    T = (result["x"].reshape(1, -1))
    scaledT = ScalerX.transform(T)
    nonscaledN = regr.predict(scaledT)
    N = ScalerY.inverse_transform(nonscaledN.reshape(1,-1))

    print('Result: \n', result)

    return T,N,result

if __name__ == '__main__':
    T, N, result = findOptimum()

    g = open('00_Result.txt', 'a')
    g.write('T: ' + str(T) + ' N: ' + str(N) + '\n')
    g.close()
```

A.3 Code for optimisation using Abaqus

```
import scipy.optimize as optimize
from MachineLearning import doTheThing_0
from sklearn.preprocessing import StandardScaler
import numpy as np

run = 0
D = 0.15
regr, RScore, ScalerX, ScalerY = doTheThing_0()


def f(T):
    global run
    global D
```

```python
    T = ScalerX.transform(T.reshape(1,-1))
    run += 1
    output = ScalerY.inverse_transform(regr.predict(T))
    N, a, E = output[0,0], output[0,1], output[0,2]
    n_diff = -(D*E) / a
    N_goal = N + n_diff
    #output[0,0] = N_goal
    #output = ScalerY.transform(output.reshape(1,-1))
    return -N_goal

def findOptimum():
    Upper= np.array([3.5, 3.5,0.65, 8]) #np.array([100, 37.85,122.15,
    ↪ 136.48])#
    Lower = np.array([ 0.5, 0.5,0.45,8 ])#np.array([ 50, 16.89, 66.09,
    ↪ 75.9])#

    bnds = [(Lower[0], Upper[0]), (Lower[1], Upper[1]), (Lower[2],
    ↪ Upper[2]), (Lower[3], Upper[3])]

    result = optimize.differential_evolution(f, bounds=bnds)

    T = (result["x"].reshape(1, -1))
    scaledT = ScalerX.transform(T)
    nonscaledN = regr.predict(scaledT)
    N = ScalerY.inverse_transform(nonscaledN.reshape(1,-1))

    print('Result: \n', result)

    return T,N,result

if __name__ == '__main__':
    T, N, result = findOptimum()

    g = open('00_Result.txt', 'a')
    g.write('T: ' + str(T) + ' N: ' + str(N) + '\n')
    g.close()
```

A.4 Code for data collection and Calculation

A.4.1 Creation file

```
#Draws, meshes and creates PBC for Buckle and PB analysis
#NOTE: Keywords must be edited manually, so these do not submit the jobs, or
↪  do PostProcessing analysis.
from abaqus import *
from abaqusConstants import *
from math import *
import sketch
import part
import mesh
import assembly
import regionToolset
import job
import visualization

from DrawAndMeshLib import getT, getNewT, Model
#import math
#session.Viewport(name='Viewport: 1', origin=(0.0, 0.0),
↪  width=307.999969482422,
#    height=170.116683959961)
#session.viewports['Viewport: 1'].setValues(displayedObject = None)
from datetime import datetime
import time
import numpy as np
#import PySimpleGUI as sg


```

```
#BOTH MODELS:
t1=  48.3129513737351
tv1=  108.32641187258622
t2= 18.6536491790483
tv2=  122.23123660344206

T_0 = [t1, t2, tv1, tv2]
#t1/t2
sigma = 1.128
SD = 0.465
R1 = [sigma - SD, sigma - 0.75*SD, sigma - 0.5*SD, sigma - 0.25*SD, sigma,
↪  sigma + 0.25*SD, sigma + 0.5*SD, sigma + 0.75*SD, sigma + SD] #0.9, 1.2,
↪  1.5
sigma = 2.598
SD = 1.153
R2 = [sigma - SD, sigma - 0.75*SD, sigma - 0.5*SD, sigma - 0.25*SD, sigma,
↪  sigma + 0.25*SD, sigma + 0.5*SD, sigma + 0.75*SD, sigma + SD]
_hx = 20

alpha = pi/2#1.361
r = 0.5
Vf_array = [0.45, 0.5, 0.55, 0.6, 0.65]

#Mdb('Testmdb_1') #only if the database doesnt exist. If it does, insert the
↪  string in the "mdb.save()"
openMdb(pathName = 'Testmdb_1.cae')

from os.path import exists

for i, n in enumerate(R1):
    for j, m in enumerate(R2):
        for k, Vf in enumerate(Vf_array):

            T = getNewT(Vf, T_0)
            T = getT(T, n=n, case=1)
            T = getT(T, n=m, case=2)
            t1, t2, tv1, tv2 = T[0], T[1], T[2], T[3]

            dependent = OFF #boolean to decide whether assembly instance is
             ↪  indcependent or dependet. OFF -> independent

            #Beware the naming convention
            strN = str(int(n*100))
            strM = str(int(m*100))
            strVf = '0' + str(int(Vf*100))

            model_name = "Test_R1" + strN + 'R2' + strM + "_Vf" + strVf +
             ↪  '_20_'  # NB: XYplotname has max. num of charachters

            stepName = "DC"
            materialName = "Steel"
            jobName = model_name + '_08e'

            path =
             ↪  "C:/Users/katin/Documents/Studie/0_V2022/Thesis/FEM/Data/Data/"
             ↪  + jobName + "_SDEG.txt"
            if exists(path):
                continue

            geometry = [r, alpha, T]

            model = Model(model_name=model_name, geometry=geometry,
             ↪  exists=False, hx = _hx, flexElmS=True)

            #m.createPartition()
            try:
                model.doItAll()
            except:
                print("Issue getting nodes for PBC in job " + jobName)
                mdb.save()
                continue


            ↪   mdb.models[model_name].steps['DC'].setValues(maxNumCycles=1000,maxCycleI
            ↪   =100)


            ↪   #mdb.models[model_name].boundaryConditions['Displacement'].setValues(u2=
            #m.createPBC()

            aJob = model.createJob(jobName = jobName)

            #getjob
            #aJob = mdb.jobs[jobName]
            mdb.save()

            #Record job-start
            path =
            ↪   "C:/Users/katin/Documents/Studie/0_V2022/Thesis/FEM/Data/Data/"
            ↪   + "00_timekeeper.txt"
            file_object = open(path, 'a')
            now = datetime.now()
            start_time = time.time()
            current_time = now.strftime("%H:%M:%S")
            string = '\n' + jobName + ' started at: ' + current_time + '\n'
            file_object.write(string)
            file_object.close()

            aJob.submit()
            aJob.waitForCompletion()

            # Record job-finish time
            file_object = open(path, 'a')
            now = datetime.now()
            current_time = now.strftime("%H:%M:%S")
            total_time = time.time() - start_time
            min = int(total_time / 60)
            sek = total_time % 60
            string = jobName + ' finished at: ' + current_time + '. Total
            ↪  run-time: ' + str(min) + ' min and ' + str(np.round(sek, 2))
            ↪  + ' sek (' + str(total_time) +' sek)\n'
            file_object.write(string)
            file_object.close()

            mdb.save()

            model.postProcessStressStrain(aJob, XYplotname='XYplot' +
            ↪  jobName + "000")

            model.postProcessSDEG(job = aJob)
```

A.4.2 Post-processing file

```python
import numpy as np
import matplotlib.pyplot as plt

def plotStressStrain(r, elm):

    for i, n in enumerate(r):
        for j, m in enumerate(elm):
            modelName = 'Test_' + n + '_' + m + '_06e_1000N'
            f = open(modelName + '_wh_stressStrain.txt', 'r')
            content = f.readlines()
            x = np.zeros(len(content))
            y = np.zeros(len(content))
            for l, line in enumerate(content):
                firstNumber = True
                x_i = ''
                y_i = ''
                for c in line:
                    if c==',':
                        firstNumber = False
                    elif firstNumber:
                        x_i += c
                    else:
                        y_i += c
                x[l] = float(x_i)
                y[l] = float(y_i)
            plt.plot(x[:], y[:], label = 'Ratio_' + n + '_' + m)
            f.close()


    plt.xlabel('Strain')
    plt.ylabel('Stress')
    plt.title('Stress-Strain curve')
    #plt.ylim(top=0.5)
    plt.legend()
    plt.show()

def plotSDEG(r, elm):
    for i, n in enumerate(r):
        for j, m in enumerate(elm):
            map = {'0.33': '03', '0.5': '05', '1': '1', '2': '2', '3': '3'}
            k = map[str(n)]
            l = map[str(m)]
            modelName = 'Test_R' + k + l + '_Vf01_06e_1000N'
            f = open(modelName + '_SDEG.txt', 'r')
            content = f.readlines()
            x = np.zeros(int(len(content)/20))
            y = np.zeros(int(len(content)/20))
            count = 0
            for l in range(0, len(content)-1, 20):
                firstNumber = True
                x_i = ''
                y_i = ''
                for c in content[l]:
                    if c==',':
                        firstNumber = False
                    elif firstNumber:
                        x_i += c
                    else:
                        y_i += c
                x[count] = float(x_i)
                y[count] = float(y_i)
                dN = x[count] - x[count-1]
                if (l == len(content) - 1) and (dN < 100):
                    x[count] += 100 - (dN)
                    y_last = y[count-1] + dN * (y[count] - y[count-1]) /
                     ↪  (100)
                    print(y_last)
                    y[count] = y_last
                    x[count] = 1000
                count += 1
            plt.plot(np.log(x[:]), y[:], label = 'Ratio_' + str(n) + '_' +
            ↪  str(m))
            f.close()


    plt.xlabel('Cycles')
    plt.ylabel('Degradation factor')
    plt.title('Degradation')
    #plt.ylim(top=0.5)
    plt.legend()
    plt.show()

def getSdeg(X, Y):
    SDEG = np.zeros((len(X), len(Y)))
    SDEG_temp = 0
    for i,x in enumerate(X):
        for j,y in enumerate(Y):

            map = {'0.33': '03', '0.5': '05', '1': '1', '2': '2', '3': '3'}
            k = map[str(x)]
            l = map[str(y)]
            modelname = "Test_R" + k + l + '_Vf09_06e_1000N'
            f = open(modelname + '_SDEG.txt', 'r')
            content = f.readlines()
            #x = np.zeros(len(content))
            #y = np.zeros(len(content))
            x_1000 = 0
            y_1000 = 0

            maxIter = 250
            steps = np.zeros(2)
            sDegs = np.zeros(2)
            for l in range(1,len(content)-1, 20):
                firstNumber = True
                x_i = ''
                y_i = ''
                line = content[l]
                for c in line:
                    if c == ',':
                        firstNumber = False
                    elif firstNumber:
                        x_i += c
                    else:
                        y_i += c
                x_i = float(x_i)
                y_i = float(y_i)

                step = x_i
                steps[0] = steps[1]
                steps[1] = step

                sDegs[0] = sDegs[1]
                sDegs[1] = y_i
                dN = steps[1] - steps[0]
            if (dN < 100):
                steps[1] += 100 - (dN)
                w_1000 = sDegs[0] + dN * (sDegs[1] - sDegs[0]) / (100)
                sDegs[1] = w_1000
                steps[1] = 1000

            f.close()
            SDEG[i,j] = sDegs[1]

    return SDEG
```

```python

def getRelativeStiffness(X,Y, Vf):
    E_N = np.zeros((len(X), len(Y)))
    for i, n in enumerate (X):
        for j, m in enumerate (Y):
            map = {'0.33': '03', '0.5': '05', '1': '1', '2': '2', '3': '3'}
            k = map[str(n)]
            l = map[str(m)]
            modelname = "Test_R" + k + l + '_Vf0'+ str(int(Vf*10))
             ↪  +'_06e_1000N'
            f = open(modelname + '_stressStrain.txt', 'r')
            content = f.readlines()
            E_eff = np.zeros(int(len(content)/20))
            step = np.zeros(int(len(content) / 20))
            g = open(modelname + '_SDEG.txt', 'r')
            StepNumber = g.readlines()
            count = 0
            for k in range(5, len(content), 20):
                string1 = content[k]
                string2 = content[k+1]
                firstNumber=True
                x_1 = ''
                y_1 = ''
                for c in string1:
                    if c==',':
                        firstNumber = False
                    elif firstNumber:
                        x_1 += c
                    else:
                        y_1 += c
                firstNumber = True
                x_2 = ''
                y_2 = ''
                for c in string2:
                    if c == ',':
                        firstNumber = False
                    elif firstNumber:
                        x_2 += c
                    else:
                        y_2 += c
                step_string = ''
                firstNumber=True
                for c in StepNumber[k]:
                    if c == ',':
                        firstNumber = False
                    elif firstNumber:
                        step_string += c
                x1 = float(x_1)
                y1 = float(y_1)
                x2 = float(x_2)
                y2 = float(y_2)
                E_eff[count] = (y2-y1)/(x2-x1)
                step[count] = int(float(step_string))
                dN = step[count] - step[count - 1]
                if (k == len(content)-16) and (dN < 100):
                    step[count] += 100 - (dN)
                    w_1000 = E_eff[count - 1] + dN * (E_eff[count] -
                    ↪   E_eff[count - 1]) / (100)
                    E_eff[count] = w_1000
                    step[count] = 1000
                    if E_N[i,j] ==0:
                        E_N[i,j] = 1100
                if (E_eff[count]/E_eff[0] == 0.9):
                    E_N[i,j] = step[count]
                    continue
                elif E_eff[count]/E_eff[0] < 0.9:
                    E_N[i,j] = step[count-1] + ((step[count] -
                     ↪  step[count-2])/(E_eff[count]-E_eff[count-1]))*(E_eff[0]*0.9
                     ↪  - E_eff[count-1])
                    continue
                count += 1
    return E_N


def ThreeDplot(X, Y, Vf):
    from mpl_toolkits.mplot3d import Axes3D
    import matplotlib.pyplot as plt
    from matplotlib import cm
    from matplotlib.ticker import LinearLocator, FormatStrFormatter
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    X = [0.33, 0.5, 1, 2, 3]
    Y = [0.33, 0.5, 1, 2, 3]
    E_N = getRelativeStiffness(X,Y, Vf)
    X,Y = np.meshgrid(X,Y)
    print(E_N)
    surf1 = ax.plot_surface(X, Y, E_N, cmap=cm.coolwarm,
                            linewidth=0, antialiased=False)
    plt.title("Degradation at N=1000 for Vf: " + str(Vf))
    plt.xlabel("R1")
    plt.ylabel("R2")
    plt.show()

def plotEffectiveStiffness(r, elm, Vf):
    for i, n in enumerate (r):
        for j, m in enumerate (elm):
            map = {'0.33': '03', '0.5': '05', '1': '1', '2': '2', '3': '3'}
            k = map[str(n)]
            l = map[str(m)]
            modelname = "Test_R" + k + l +
             ↪  '_Vf0'+str(int(Vf*10))+'_06e_1000N'
            f = open(modelname + '_stressStrain.txt', 'r')
            content = f.readlines()
            E_eff = np.zeros(int(len(content)/20))
            step = np.zeros(int(len(content) / 20))
            g = open(modelname + '_SDEG.txt', 'r')
            StepNumber = g.readlines()
            count = 0
            for i in range(5, len(content), 20):
                string1 = content[i]
                string2 = content[i+1]
                firstNumber=True
                x_1 = ''
                y_1 = ''
                for c in string1:
                    if c==',':
                        firstNumber = False
                    elif firstNumber:
                        x_1 += c
                    else:
                        y_1 += c
                firstNumber = True
                x_2 = ''
                y_2 = ''
                for c in string2:
                    if c == ',':
                        firstNumber = False
                    elif firstNumber:
                        x_2 += c
                    else:
                        y_2 += c
                step_string = ''
                firstNumber=True
                for c in StepNumber[i]:
                    if c == ',':
                        firstNumber = False
                    elif firstNumber:
                        step_string += c
                x1 = float(x_1)
                y1 = float(y_1)
                x2 = float(x_2)
                y2 = float(y_2)
                E_eff[count] = (y2-y1)/(x2-x1)
                step[count] = int(float(step_string))
                dN = step[count] - step[count - 1]
                if (i == len(content)-16) and (dN < 100):
                    step[count] += 100 - (dN)
                    w_1000 = E_eff[count - 1] + dN * (E_eff[count] -
                     ↪  E_eff[count - 1]) / (100)
                    E_eff[count] = w_1000
                    step[count] = 1000
                count += 1
            f.close()
            g.close()
            plt.plot(step, E_eff, label = modelname)
    plt.legend()
    plt.xlabel('Cycles')
    plt.ylabel('E\'')
    plt.title('Effective stiffness Vf: ' + str(Vf))
    plt.show()

def plotPlasticStrainEngergy(r, elm):
    for i, n in enumerate(r):
        for j, m in enumerate(elm):
            modelname = 'Test_' + n + '_' + m + '_06e_1000N'
            f = open(modelname + '_wh_stressStrain.txt', 'r')
            content = f.readlines()
            step = np.zeros(int(len(content) / 20))
            print(modelname, '\nLength: ', len(content))
            g = open(modelname + '_wh_SDEG.txt', 'r')
            StepNumber = g.readlines()
            count = 0
            w = np.zeros(int(len(content)/ 20) )
            for i in range(0, len(content)-1, 20):
                string_i = content[i]
                x_i = ''
                y_i = ''
                firstNumber=True
                for c in string_i:
                    if c == ',':
                        firstNumber = False
                    elif firstNumber:
                        x_i += c
                    else:
                        y_i += c
                x0 = float(x_i)
                y0 = float(y_i)
                dw = 0
                for j in range(1, 21):
                    string_i = content[i+j]
                    x_i = ''
                    y_i = ''
                    firstNumber=True
                    for c in string_i:
                        if c == ',':
                            firstNumber = False
                        elif firstNumber:
                            x_i += c
                        else:
                            y_i += c
                    xi = float(x_i)
                    yi = float(y_i)
                    dw += ((yi + y0)/2)*(xi - x0)
                    x0 = xi
                    y0 = yi
                #dw += ((y1 + y0) / 2) * (x1 - x0)
                w[count] = dw
                step_string = ''
                firstNumber = True
                for c in StepNumber[i +j]:
                    if c == ',':
                        firstNumber = False
                    elif firstNumber:
                        step_string += c
                step[count] = int(float(step_string))
                dN = step[count] - step[count-1]
                if (i + j == len(content) -1) and (dN < 100):
                    step[count] += 100 - (dN)
                    w_1000 = w[count-1] + dN*(w[count] - w[count-1])/(100)
                    w[count] = w_1000
                    step[count] = 1000
                count +=1


            f.close()
            g.close()
            plt.plot(step, w, label=modelname)
            print(modelname, ': \n', w)
    plt.legend()
    plt.xlabel('Cycles')
    plt.ylabel('\u0394w')
    plt.title('Plastic strain energy')
    plt.show()

def sumTime():
    f=open("00_timekeeper.txt")

    content = f.readlines()

    totalTime = 0

    for l in content:
        if l[len(l)-2] != ')':
            continue
        count = len(l)-2
        c = l[count]
        tempNumber = ''
        while c != '(' and iter:
            if c.isdigit() or c =='.':
                tempNumber += c
            count -= 1
            c = l[count]
        number = ''
        for n in tempNumber[::-1]:
            number += n

        totalTime += float(number)

    print('Total time: ', totalTime, ' s. ')

    h = int(totalTime / 3600)

    min = int((totalTime - h * 3600) / 60)

    s = (totalTime - h * 3600) % 60

    print('Time: ', str(h), ' h ', str(min), ' min ', str(s), ' sek')

    return totalTime, [h, min, s]

_Vf = [ 0.1]
```

```python
r = [0.33, 0.5, 1, 2, 3]

for Vf in _Vf:
    ThreeDplot(r, r, Vf)

    plotEffectiveStiffness(r,r, Vf)
```

A.4.3 Library

```python
from abaqus import *
from abaqusConstants import *
from math import *
import sketch
import part
import mesh
import assembly
import regionToolset
import job
import interaction
import step
import os

def getNewT(Vf, T):
    #keeps the ralations tv2/tv1 and t1/t2 - but changes the volume fracion
    ↪  to Vf

    t1, t2, tv1, tv2 = T[0], T[1], T[2], T[3]

    w = tv1 + t1
    h = t2 + tv2
    A_t = w * h
    Av = A_t*(1-Vf)

    n = tv2/tv1
    tv1 = +sqrt(Av/n)
    tv2 = n*tv1

    r = t2/t1
    a = tv2 / 2
    b = tv1 / 2
    l = r
    m = 2 * (r * a + b)
    n = 4 * a * b - A_t

    t2 = (-m + sqrt(m ** 2 - 4 * l * n)) / (2 * l)
    t1 = t2 * r
    T = [t1, t2, tv1, tv2]


    return T

def GetKeywordPosition(m, blockPrefix, occurrence=1):
    #if blockPrefix == '':
        #return len(m.keywordBlock.sieBlocks)+1
    pos = 0
    foundCount = 0
    for block in m.keywordBlock.sieBlocks:
        if block[0:len(blockPrefix)]==\
           blockPrefix:
            foundCount = foundCount + 1
            if foundCount >= occurrence:
                return pos
        pos=pos+1
    return +1

def getT(T, n, case = 0):
    # case: 1 - tv2/tv1; 2 - t1/t2;
    t1, t2, tv1, tv2 = T[0], T[1], T[2], T[3]
    w = tv1 + t1
    h = t2 + tv2
    A_t = w * h
    A_v = tv2 * tv1
    Vf = (A_t - A_v) / A_t
    print(Vf)
    if case == 1:
        # Vf, t2, t1 stays the same
        a = -n*Vf
        b = (n*t1 + t2)*(1-Vf)
        c = (1-Vf)*(t1*t2)
        tv1 = (-b - sqrt(b**2 - 4*a*c))/(2*a)
        tv2 = n*tv1
        print("tv1: ", tv1)
        print("tv2: ", tv2)

        T = [t1, t2, tv1, tv2]
    elif case == 2:
        r = n
        a = tv2/2
        b = tv1/2
        l = r
        m = 2*(r*a + b)
        n = 4*a*b - A_t

        t2 = (-m + sqrt(m**2 - 4*l*n))/(2*l)
        t1 = t2*r
        T = [t1, t2, tv1, tv2]
    elif case ==3:
        T = T  # TODO

    w = tv1 + t1
    h = t2 + tv2
    A_t = w * h
    A_v = tv2 * tv1
    Vf = (A_t - A_v) / A_t
    print(Vf)
    return T

def get_abwh(t1,t2,tv1,tv2):
    #gets a, b, w, and h for elliptical RVE
    w = tv1 + t1
    h = t2 + tv2
    A_t = w * h
    A_v = tv2 * tv1
    Vf = (A_t - A_v) / A_t

    A_sq = w * h
    c_1 = t1 / t2
    c_0 = w / 2 - c_1 * (h / 2)

    # solve the 2nd degree eq
    a = (-pi * c_0 + sqrt((pi * c_0) ** 2 + 4 * pi * c_1 * A_sq * (1 - Vf)))
    ↪  / (2 * pi * c_1)
    b = c_1 * a + c_0

    return [a, b, w, h]

def draw_SV(T, model_name = "RVE_square_void", r=0.5, alpha=pi/2):
    #Draws the RVE with ellipse void. Not sloped transverse lamella
    #Sizes is vector containing the sizes: [a, b, w, h]
    t1, t2, tv1, tv2 = T[0], T[1], T[2], T[3]
    w = (tv1 + t1)
    h = t2 + tv2
    beta = pi/2-alpha
    myModel = mdb.Model(name=model_name)
    mySketch = myModel.ConstrainedSketch(name="RVE_square_void", sheetSize =
    ↪  200)
    #Create center square
    #mySketch.rectangle((-tv1/2, tv2/2), (tv1/2,-tv2/2))


    #Create translation-vectors for the other ellipses:

    lcx = -w
    rcx = w
    d = h*(0.5-r)
    tlcy = h/2 - w*tan(beta)
    trcy = h/2 + w*tan(beta)
    blcy = -h/2 - w*tan(beta)
    brcy = -h/2 + w*tan(beta)
    b = tv1/2
    a = tv2/2
    t = tan(beta)

    #Drax center void
    mySketch.Line((-b, a - b*t), (b, a + b*t))
    mySketch.Line((b, a + b * t), (b, -a + b*t))
    mySketch.Line((b, -a + b*t), (-b, -a - b*t))
    mySketch.Line((-b, -a - b*t), (-b, a - b*t))


    if (t2/2 + h*(0.5-r)) < h/2:
        mySketch.Line((lcx, tlcy - a + d), (lcx+b, tlcy - a + d + b*t))
        mySketch.Line((lcx+b, tlcy - a + d + b*t), (lcx+b, tlcy + b*t))
        mySketch.Line((lcx+b, tlcy + b*t), (rcx-b, trcy - b*t))
        mySketch.Line((rcx-b, trcy - b*t), (rcx - b, trcy - b*t - a + d))
        mySketch.Line((rcx - b, trcy - b*t - a + d), (rcx, trcy - a + d))
        mySketch.Line((rcx, trcy - a + d), (rcx, brcy + a + d))
        mySketch.Line((rcx, brcy + a + d), (rcx - b, brcy + a + d - b*t))
        mySketch.Line((rcx - b, brcy + a + d - b*t), (rcx - b, brcy - b*t))
        mySketch.Line((rcx - b, brcy - b*t), (lcx + b, blcy + b*t))
        mySketch.Line((lcx + b, blcy + b*t), (lcx + b, blcy + b*t + a + d))
        mySketch.Line((lcx + b, blcy + b*t + a + d), (lcx, blcy + a + d))
        mySketch.Line((lcx, blcy + a + d), (lcx, tlcy - a + d))
    elif (t2/2 + h*(0.5-r)) == h/2 :
        mySketch.Line((-w, tlcy - t2), (-w, tlcy))
        mySketch.Line((-w, tlcy), (w, trcy))
        mySketch.Line((w, trcy), (w, trcy - t2))
        mySketch.Line((w, trcy - t2), (w - b, trcy - t2 - b*t))
        mySketch.Line((w - b, trcy - t2 - b*t), (w - b, brcy - b*t))
        mySketch.Line((w - b, brcy - b*t), (-w + b, -blcy + b*t))
        mySketch.Line((-w + b, -blcy + b*t), (-w + b, tlcy - t2 + b*t))
        mySketch.Line((-w + b, tlcy - t2 + b*t), (-w, tlcy - t2))
    elif (t2/2 + h*(0.5-r)) > h/2:
        mySketch.Line((-w, tlcy - t2 - a + d), (-w, tlcy))
        mySketch.Line((-w, tlcy), (w, trcy))
        mySketch.Line((w, trcy), (w, trcy - t2 - a + d))
        mySketch.Line((w, trcy - t2 - a + d), (w - b, trcy - t2 - a + d -
        ↪  b*t))
        mySketch.Line((w - b, trcy - t2 - a + d - b*t), (w - b, brcy - a + d
        ↪  - b*t))
        mySketch.Line((w - b, brcy - a + d - b*t), (w, brcy - a + d))
        mySketch.Line((w, brcy - a + d), (w, brcy))
        mySketch.Line((w, brcy), (-w, blcy))
        mySketch.Line((-w, blcy), (-w, blcy - a + d))
        mySketch.Line((-w, blcy - a + d), (-w + b, blcy - a + d + b*t))
        mySketch.Line((-w + b, blcy - a + d + b*t), (-w + b, blcy + a + d +
        ↪  b*t))
        mySketch.Line((-w + b, blcy + a + d + b*t), (-w , blcy + a + d))



    return myModel, mySketch

def create_nodes_and_PBC(T, model_name, instance_name, strainDirectionX =
↪  False, strainDirectionY=True, alpha = pi/2, dependent = False):
    #creates the nodes and equationconstraints on the specifies model and
    ↪  instance
    t1, t2, tv1, tv2 = T[0], T[1], T[2], T[3]
    w = (tv1 + t1)
    h = t2 + tv2
    beta = pi/2 - alpha
    t = tan(beta)
    myModel = mdb.models[model_name]
    myAssm = myModel.rootAssembly
    myAssmInst = myAssm.instances[instance_name]
    part = myModel.parts[instance_name]
    if dependent == False:
        allNodes = myAssmInst.nodes
    else:
        allNodes = part.nodes
        myAssm = part


    # finds the outer bounds
    node0 = allNodes[0]
    x_min = node0.coordinates[0]
    x_max = node0.coordinates[0]
    y_min = node0.coordinates[1]
    y_max = node0.coordinates[1]

    left_nodes_mesh = []
    right_nodes_mesh = []
    top_nodes_mesh = []
    bottom_nodes_mesh = []

    TRC_mesh = []
    TLC_mesh = []
    BRC_mesh = []
    BLC_mesh = []

    corner = False

    if beta ==0:
        for node in allNodes:
            x = node.coordinates[0]
            y = node.coordinates[1]
            if x < x_min:
                x_min = x
            elif x > x_max:
                x_max = x

            if y < y_min:
                y_min = y
            elif y > y_max:
                y_max = y



        for node in allNodes:
            x = node.coordinates[0]
            y = node.coordinates[1]
            if x == x_min:
                if y == y_max:
                    TLC_mesh.append(node)
                    corner = True
                elif y == y_min:
                    BLC_mesh.append(node)
                    corner = True
                else:
                    left_nodes_mesh.append(node)
            elif x == x_max:
                if y == y_max:
                    TRC_mesh.append(node)
                    corner = True
                elif y == y_min:
                    BRC_mesh.append(node)
                else:
                    right_nodes_mesh.append(node)
            elif y == y_max and x != x_max and x != x_min:
                top_nodes_mesh.append(node)
            elif y == y_min and x != x_min and x != x_max:
                bottom_nodes_mesh.append(node)
    else:
        for node in allNodes:
            x = node.coordinates[0]
            if x < x_min:
                x_min = x
            elif x > x_max:
                x_max = x
        for node in allNodes:
            x = node.coordinates[0]
            y = node.coordinates[1]
            if y >= h/2 - w*t and y<= h/2 + w*t + 0.01:
                y_d = y - h/2
                r = (y_d-x*t)**2
                if r<0.0000001:
                    if x==x_min:
                        TLC_mesh.append(node)
                        corner = True
                    elif x==x_max:
                        TRC_mesh.append(node)
                        corner = True
                    else:top_nodes_mesh.append(node)
                elif x == x_min:
                    left_nodes_mesh.append(node)
                elif x == x_max:
                    right_nodes_mesh.append(node)
            elif y >= -h/2 - w*t - 0.01 and y<= - h/2 + w*t:
                y_d = y + h / 2
                r = (y_d - x * t) ** 2
                if r < 0.0000001:
                    if x==x_min:
                        BLC_mesh.append(node)
                        corner = True
                    elif x==x_max:
                        BRC_mesh.append(node)
                        corner = True
                    else:bottom_nodes_mesh.append(node)
                elif x == x_min:
                    left_nodes_mesh.append(node)
                elif x == x_max:
                    right_nodes_mesh.append(node)
            elif x == x_min:
                left_nodes_mesh.append(node)
            elif x == x_max:
                right_nodes_mesh.append(node)


    leftNodes = mesh.MeshNodeArray(left_nodes_mesh)
    rightNodes = mesh.MeshNodeArray(right_nodes_mesh)
    topNodes = mesh.MeshNodeArray(top_nodes_mesh)
    bottomNodes = mesh.MeshNodeArray(bottom_nodes_mesh)

    myAssm.Set(nodes=leftNodes, name="Left_nodeSet")
    myAssm.Set(nodes=rightNodes, name="Right_nodeSet")
    myAssm.Set(nodes=topNodes, name="Top_nodeSet")
    myAssm.Set(nodes=bottomNodes, name="Bottom_nodeSet")



    if len(leftNodes) != len(rightNodes):
        exit("Length of right and left nodeset not equal")
    i = 0
    for node in leftNodes:
        temp = [node]
        string = "L" + str('{:03}'.format(i))
        node_t = mesh.MeshNodeArray(temp)
```

```
            myAssm.Set(nodes=node_t, name=string)
            i += 1

    for node in rightNodes:
        temp = [node]
        node_t = mesh.MeshNodeArray(temp)
        i = 0
        for node_check in leftNodes:
            r = (node.coordinates[1] - node_check.coordinates[1] - 2*w*t) **
             ↪  2
            if r < 0.000001:
                string = "R" + str('{:03}'.format(i))
                myAssm.Set(nodes=node_t, name=string)
            i += 1


    if len(topNodes) != len(bottomNodes):
        exit("Length of top an bottom nodes not equal")
    i = 0


    for node in topNodes:
        temp = [node]
        node_t = mesh.MeshNodeArray(temp)
        string = "T" + str('{:03}'.format(i))
        myAssm.Set(nodes=node_t, name=string)
        i += 1



    for node in bottomNodes:
        temp = [node]
        node_t = mesh.MeshNodeArray(temp)
        i = 0
        for node_check in topNodes:
            r = (node.coordinates[0] - node_check.coordinates[0]) ** 2
            if r < 0.000001:
                string = "B" + str('{:03}'.format(i))
                myAssm.Set(nodes=node_t, name=string)
            i += 1



    for i in range(len(leftNodes)):
        for j in range(2):
            CE_name = "LR_" + str('{:03}'.format(i)) + "_" + str(j + 1)
            LN_name = "L" + str('{:03}'.format(i))
            RN_name = "R" + str('{:03}'.format(i))
            if dependent==True:
                LN_name = instance_name + ".L" + str('{:03}'.format(i))
                RN_name = instance_name + ".R" + str('{:03}'.format(i))
            myModel.Equation(name=CE_name,
                             terms=((1.0, RN_name, j + 1), (-1.0, LN_name, j
                             ↪  + 1), (-1.0, "RP_LR", j + 1)))

    for i in range(len(topNodes)):
        for j in range(2):
            CE_name = "TB_" + str('{:03}'.format(i)) + "_" + str(j + 1)
            TN_name = "T" + str('{:03}'.format(i))
            BN_name = "B" + str('{:03}'.format(i))
            if dependent==True:
                TN_name = instance_name + ".T" + str('{:03}'.format(i))
                BN_name = instance_name + ".B" + str('{:03}'.format(i))
            myModel.Equation(name=CE_name,
                             terms=((1.0, TN_name, j + 1), (-1.0, BN_name, j
                             ↪  + 1), (-1.0, "RP_TB", j + 1)))

    # If there are corners, this should be executed:
```

```


    if corner == True:
        TRC = mesh.MeshNodeArray(TRC_mesh)
        TLC = mesh.MeshNodeArray(TLC_mesh)
        BRC = mesh.MeshNodeArray(BRC_mesh)
        BLC = mesh.MeshNodeArray(BLC_mesh)
        myAssm.Set(nodes=BLC, name="BLC")
        myAssm.Set(nodes=BRC, name="BRC")
        myAssm.Set(nodes=TLC, name="TLC")
        myAssm.Set(nodes=TRC, name="TRC")


        if (strainDirectionX == True and dependent == False): # for strain
         ↪  in x-direction.
            myModel.Equation(name="TC_1", terms=((1.0, "TRC", 1), (-1.0,
             ↪  "TLC", 1), (-1.0, "RP_LR", 1)))
            myModel.Equation(name="TC_2", terms=((1.0, "TRC", 2), (-1.0,
             ↪  "TLC", 2), (-1.0, "RP_LR", 2)))

            myModel.Equation(name="BC_1", terms=((1.0, "BRC", 1), (-1.0,
             ↪  "BLC", 1), (-1.0, "RP_LR", 1)))
            myModel.Equation(name="BC_2", terms=((1.0, "BRC", 2), (-1.0,
             ↪  "BLC", 2), (-1.0, "RP_LR", 2)))
        elif (strainDirectionX==True and dependent ==True ):
            myModel.Equation(name=instance_name + ".TC_1", terms=((1.0,
             ↪  instance_name + ".TRC", 1), (-1.0, instance_name + ".TLC",
             ↪  1), (-1.0, "RP_LR", 1)))
            myModel.Equation(name=instance_name + ".TC_2", terms=((1.0,
             ↪  instance_name + ".TRC", 2), (-1.0, instance_name + ".TLC",
             ↪  2), (-1.0, "RP_LR", 2)))

            myModel.Equation(name=instance_name + ".BC_1", terms=((1.0, instance_name + ".BRC", 1), (-1.0, instance_name + ".BLC", 1), (-1.0, "RP_LR", 1)))
            myModel.Equation(name=instance_name + ".BC_2", terms=((1.0, instance_name + ".BRC", 2), (-1.0, instance_name + ".BLC", 2), (-1.0, "RP_LR", 2)))
        elif (strainDirectionY == True and dependent == False): #for strain i y-direction
            myModel.Equation(name="RC_1", terms=((1.0, "TRC", 1), (-1.0, "BRC", 1),(-1.0, "RP_TB", 1)))
            myModel.Equation(name="RC_2", terms=((1.0, "TRC", 2), (-1.0, "BRC", 2),(-1.0, "RP_TB", 2)))

            myModel.Equation(name="LC_1", terms=((1.0, "TLC", 1), (-1.0, "BLC", 1),(-1.0, "RP_TB", 1)))
            myModel.Equation(name="LC_2", terms=((1.0, "TLC", 2), (-1.0, "BLC", 2),(-1.0, "RP_TB", 2)))
        elif (strainDirectionY == True and dependent == True): #for strain i y-direction
            myModel.Equation(name=instance_name + ".RC_1", terms=((1.0, instance_name + ".TRC", 1), (-1.0, instance_name + ".BRC", 1),(-1.0, "RP_TB", 1)))
            myModel.Equation(name=instance_name + ".RC_2", terms=((1.0, instance_name + ".TRC", 2), (-1.0, instance_name + ".BRC", 2),(-1.0, "RP_TB", 2)))

            myModel.Equation(name=instance_name + ".LC_1", terms=((1.0, instance_name + ".TLC", 1), (-1.0, instance_name + ".BLC", 1),(-1.0, "RP_TB", 1)))
            myModel.Equation(name=instance_name + ".LC_2", terms=((1.0, instance_name + ".TLC", 2), (-1.0, instance_name + ".BLC", 2),(-1.0, "RP_TB", 2)))


```

```
def postProcess(path, stepName, ovbU, Uname, ovbRF, RFname, setName,
↪ XYplotname):
    session.Viewport(name='Viewport: 1', origin=(0.0, 0.0),
    ↪ width=77.2499923706055,
                     height=31.0)
    session.viewports['Viewport: 1'].makeCurrent()
    session.viewports['Viewport: 1'].maximize()
    from caeModules import *
    from driverUtils import executeOnCaeStartup
    executeOnCaeStartup()
    session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
        referenceRepresentation=ON)

    o3 = session.openOdb(
        name=path)

    session.viewports['Viewport: 1'].setValues(displayedObject=o3)
    session.viewports['Viewport: 1'].makeCurrent()
    odb = session.odbs[path]

    xy_result = session.XYDataFromHistory(name=RFname, odb=odb,
                                          outputVariableName=ovbRF,
                                          steps=(stepName,),
                                          ↪ __linkedVpName__='Viewport:
                                          ↪ 1')
    c1 = session.Curve(xyData=xy_result)
    xyp = session.XYPlot(XYplotname)
    chartName = xyp.charts.keys()[0]
    chart = xyp.charts[chartName]
    chart.setValues(curvesToPlot=(c1,), )

    odb = session.odbs[path]
    xy_result = session.XYDataFromHistory(
        name=Uname, odb=odb,
        outputVariableName=ovbU,
        steps=(stepName,), __linkedVpName__='Viewport: 1')
    c1 = session.Curve(xyData=xy_result)
    xyp = session.xyPlots[XYplotname]
    chartName = xyp.charts.keys()[0]
    chart = xyp.charts[chartName]
    chart.setValues(curvesToPlot=(c1,), )
    xy1 = session.xyDataObjects[Uname]
    xy2 = session.xyDataObjects[RFname]
    xy3 = combine(-xy1, -xy2)
    xyp = session.xyPlots[XYplotname]
    chartName = xyp.charts.keys()[0]
    chart = xyp.charts[chartName]
    c1 = session.Curve(xyData=xy3)
    chart.setValues(curvesToPlot=(c1,), )
    xy1 = session.xyDataObjects[Uname]
    xy2 = session.xyDataObjects[RFname]
    xy3 = combine(-xy1, -xy2)

    sDC = 'combine (-"' + Uname + '",-"' + RFname + '" )'
    xy3.setValues(
        sourceDescription=sDC)
    tmpName = xy3.name
    session.xyDataObjects.changeKey(tmpName, setName)

    return session.xyDataObjects[setName]

class Model:

    shape = "Rectangle"

    def __init__(self, model_name, geometry, exists = False, elmsize = 5, hx
     ↪  = 10, flexElmS = False):
        import numpy as np
        # geometry: [r, alpha, T]
        # T: [t1, t2, tv1, tv2]
        self.name = model_name
        self.geomtry = geometry
        self.r, self.alpha, self.T = geometry[0], geometry[1], geometry[2]

        self.beta = pi/2 - self.alpha
        self.t = tan(self.beta)

        self.h = self.T[1] + self.T[3]
        self.w = self.T[0] + self.T[2]

        self.elmSize = elmsize
        if flexElmS:
            self.elmSize=self.h/hx

        #if self.elmSize > self.T[1]/4:
           # self.elmSize = self.T[1]/4

        tempInt = np.round(self.T[0] / self.elmSize)
        if tempInt ==0:
            tempInt = 1

        self.elementWidth = 0.5*self.T[0]/tempInt


        self.stepNames = ["Initial"]
        self.stepTypes = ["Initial"]
        self.steps = []
        self.jobs = []

        if exists:
            self.model=mdb.models[self.name]
            self.Assembly = self.model.rootAssembly
            self.AssemblyInstance = self.Assembly.instances[self.name]

    def getModel(self):
        return self.model

    def createPart(self):
        self.model, self.sketch = draw_SV(self.T, model_name=self.name,
        ↪   r=self.r, alpha=self.alpha)
        self.part = self.model.Part(dimensionality=TWO_D_PLANAR,
        ↪   name=self.name, type=DEFORMABLE_BODY)
        self.part.BaseShell(sketch=self.sketch)

        h = self.h
        w = self.w
        r = self.r

        t = self.t

        self.part.Set(edges=self.part.edges.findAt(((0, h / 2, 0),)),
        ↪   name="Top_edge")
        self.part.Set(edges=self.part.edges.findAt(((w, h * (0.5 - r) + w *
        ↪   t, 0),)), name="Right_edge")
        self.part.Set(edges=self.part.edges.findAt(((0, -h / 2, 0),)),
        ↪   name="Bottom_edge")
        self.part.Set(edges=self.part.edges.findAt(((-w, h * (0.5 - r) - w *
        ↪   t, 0),)), name="Left_edge")
        self.part.Set(faces=self.part.faces.findAt(
            ((-w + 0.1, h * (0.5 - r) - (w - 0.1) * t, 0), (w - 0.1, h *
            ↪   (0.5 - r) + (w - 0.1) * t, 0)), ), name="Face")

    def createMaterial(self, materialName="Steel"):
        self.material = self.model.Material(name=materialName)

    def editMaterialElasticPlasticHardening(self, tableElastic = ((205000, 0.3),), tablePlastic = ((450, 84844, 5085, 60486, 881.1, 18041, 163, 4935, 100.6, 2426, 9),), tableHardening = ((450, -70,2),), hardening = COMBINED, dataType=PARAMETERS, numBackstresses = 5, hardeningParameters = ON):
        # tableElastic: ((E, vy),)
        # tablePlastic: ((sigma0, C1, gamma1, ..., Cn, gamman),)
        # tablehardening: ((sigma0, Qinf, b),)
        self.material.Elastic(table=tableElastic)

        self.material.Plastic(table=tablePlastic, hardening = hardening, dataType=dataType, numBackstresses=numBackstresses)

        self.material.plastic.CyclicHardening(table = tableHardening, parameters = hardeningParameters)

    def getMaterial(self):
        return self.material

    def GetKeywordPosition(self, m, blockPrefix, occurrence=1):
        # if blockPrefix == '':
        # return len(m.keywordBlock.sieBlocks) - 1
        pos = 0
        foundCount = 0
        for block in m.keywordBlock.sieBlocks:
            if block[0:len(blockPrefix)] == \
                    blockPrefix:
                foundCount = foundCount + 1
                if foundCount >= occurrence:
                    return pos
            pos = pos + 1

        return +1

    def editKeywords(self, findArgument="*Material", insertString = 'ns'):
        if insertString=='ns':
            insertString = "*Damage Initiation, criterion=HYSTERESIS
             ↪  ENERGY\n3162.3,-1.126\n*Damage Evolution, type=HYSTERESIS
             ↪  ENERGY\n" + str(0.000453878*self.elementWidth) + ",0.095"

        self.model.keywordBlock.synchVersions(storeNodesAndElements=
                                             False)
        position = GetKeywordPosition(self.model, findArgument)
        #self.model.keywordBlock.replace(position, '\n')
        self.model.keywordBlock.insert(position,
                                      insertString)

    def assignSection(self):
        self.model.HomogeneousSolidSection(material=self.material.name,
         ↪  name=self.name, thickness=None)
        self.wholeModelRegion = ((self.part.faces.findAt(((0, -self.h / 2,
         ↪  0), (self.w, self.h * (0.5 - self.r) + self.w * self.t, 0)),
         ↪  )),)
        self.part.SectionAssignment(offset=(0.0), offsetField=" ",
         ↪  offsetType=MIDDLE_SURFACE, region=self.wholeModelRegion,
                                    sectionName=self.name)

    def createAssembly(self,instanceName, dependency = OFF):
        self.InstanceName = instanceName
        self.Assembly = self.model.rootAssembly
        self.AssemblyInstance = self.Assembly.Instance(name=instanceName,
         ↪  part=self.part, dependent=dependency)

    def createDirectCyclicStep(self, stepName="DC", timePeriod = 2,
     ↪  timeIncrementationMethod = FIXED, initialInc = 0.1, fatigue = ON,
     ↪  minCycleInc = 1, maxCycleInc = 1000, maxNumCycles=1000):
        previousStep = self.stepNames[-1]
        self.stepNames.append(stepName)
        self.stepTypes.append("DirectCyclic")

        self.steps.append(self.model.DirectCyclicStep(name=stepName,
         ↪  previous = previousStep, timePeriod = timePeriod,
         ↪  timeIncrementationMethod = timeIncrementationMethod
                                              , initialInc = initialInc, fatigue =
                                               ↪  fatigue, minCycleInc = minCycleInc,
                                               ↪  maxCycleInc = maxCycleInc,
                                               ↪  maxNumCycles=maxNumCycles))

    def createPartition(self):
        # T: [t1, t2, tv1, tv2]
        t1, t2, tv1, tv2 = self.T[0], self.T[1], self.T[2], self.T[3]
        b = tv1/2 + t1
        s = self.model.ConstrainedSketch(name='partitionSketch',
            sheetSize=2000)
        s.Line(point1=(-b, -self.t*b + t2/2), point2=(-b,-self.t*b - t2/2))
        s.Line(point1=(b, self.t * b + t2 / 2), point2=(b, self.t * b - t2 /
         ↪  2))

        s.Line(point1=(-tv1/2, -self.t * (tv1/2) + tv2/2), point2=(-tv1/2,
         ↪  -self.t * (tv1/2) + tv2/2 + t2/2))
        s.Line(point1=(-tv1 / 2, -self.t * (tv1 / 2) - tv2 / 2),point2=(-tv1
         ↪  / 2, -self.t * (tv1 / 2) - tv2 / 2 - t2 / 2))
        s.Line(point1=(tv1 / 2, self.t * (tv1 / 2) + tv2 / 2),
               point2=(tv1 / 2, self.t * (tv1 / 2) + tv2 / 2 + t2 / 2))
        s.Line(point1=(tv1 / 2, self.t * (tv1 / 2) - tv2 / 2),
               point2=(tv1 / 2, self.t * (tv1 / 2) - tv2 / 2 - t2 / 2))

        f1 = self.AssemblyInstance.faces


         ↪  self.Assembly.PartitionFaceBySketch(faces=f1.getSequenceFromMask(mask=('[#1
         ↪  ]', ), ), sketch=s)
```

```python

    def createPBC(self):
        rp1 = self.Assembly.ReferencePoint(point=(self.w * 1.5, 0, 0))
        id_1 = rp1.id
        rp2 = self.Assembly.ReferencePoint(point=(0, self.h * 0.75, 0))
        id_2 = rp2.id
        self.Assembly.Set(referencePoints=(self.Assembly.referencePoints[id_1],), name="RP_LR")
        self.RP_LR = regionToolset.Region(referencePoints=(self.Assembly.referencePoints[id_1],))
        self.Assembly.Set(referencePoints=(self.Assembly.referencePoints[id_2],), name="RP_TB")
        self.RP_TB = regionToolset.Region(referencePoints=(self.Assembly.referencePoints[id_2],))

        self.id_2 = id_2

        create_nodes_and_PBC(T = self.T, model_name=self.name, instance_name= self.name, alpha = self.alpha)

    def createLoads(self, stepName, u2 = 0.6/100, cf2=60, loadType = 'disp'):
        leftNodes, bottomNodes = [0,0], [0,0]
        i_min = 0
        i_max = 0
        nodeset = self.Assembly.allSets['Top_nodeSet'].nodes
        for i in range(len(nodeset)):
            tempNode = nodeset[i]
            if i ==0:
                x_max = tempNode.coordinates[0]
                x_min = tempNode.coordinates[0]
            if tempNode.coordinates[0] < x_min:
                x_min = tempNode.coordinates[0]
                i_min = i
            elif tempNode.coordinates[0] > x_max:
                x_max = tempNode.coordinates[0]
                i_max = i

        leftNodes[0], leftNodes[1] = "B" + str('{:03}'.format(i_min)), "T" +
        ↪  str('{:03}'.format(i_min))
        bottomNodes[1], bottomNodes[0] = "B" + str('{:03}'.format(i_max)),
        ↪  "B" + str('{:03}'.format(i_min))

        Region = regionToolset.Region(nodes =
        ↪  (self.Assembly.allSets[leftNodes[0]].nodes +
        ↪  self.Assembly.allSets[leftNodes[1]].nodes))

        self.constraint_X = self.model.DisplacementBC(
            name='xBC', createStepName='Initial',
            region=Region, u1=0)

        Region = regionToolset.Region(nodes =
        ↪  (self.Assembly.allSets[bottomNodes[0]].nodes +
        ↪  self.Assembly.allSets[bottomNodes[1]].nodes))
        self.constraint_Y = self.model.DisplacementBC(
            name='yBC', createStepName='Initial',
            region=Region, u2=0)

        amplitude = self.model.PeriodicAmplitude(name = "PeriodicAmplitude",
        ↪  frequency = pi, start = 0, a_0 = 0, data = ((0,1),))

        if loadType == 'disp':
            Region =
            ↪  regionToolset.Region(referencePoints=(self.Assembly.referencePoints[self
            self.displacement = self.model.DisplacementBC(
                name='Displacement', createStepName=stepName,
                region=Region, u2=u2, amplitude = "PeriodicAmplitude")
        else:
            Region =
             ↪  regionToolset.Region(referencePoints=(self.Assembly.referencePoints[self
            self.displacement = self.model.ConcentratedForce(
                name='Force', createStepName=stepName,
                region=Region, cf2=cf2, amplitude="PeriodicAmplitude")

    def createMesh(self, elm = mesh.ElemType(elemCode=CPE8R) ):
        Top_edge = self.Assembly.allSets[self.name + ".Top_edge"].edges
        Bottom_edge = self.Assembly.allSets[self.name +
         ↪  ".Bottom_edge"].edges
        Left_edge = self.Assembly.allSets[self.name + ".Left_edge"].edges
        Right_edge = self.Assembly.allSets[self.name + ".Right_edge"].edges
        Face = self.Assembly.allSets[self.name + ".Face"].faces


        seedTopBottom = self.elmSize
        seedLeftRight = self.elmSize

        self.Assembly.seedPartInstance((self.AssemblyInstance,), size =
         ↪  self.elmSize)
        self.Assembly.seedEdgeBySize(edges=Top_edge, size = seedTopBottom,
         ↪  constraint=FIXED)
        self.Assembly.seedEdgeBySize(edges=Bottom_edge, size=seedTopBottom,
         ↪  constraint=FIXED)
        self.Assembly.seedEdgeBySize(edges=Left_edge, size = seedLeftRight,
         ↪  constraint=FIXED)
        self.Assembly.seedEdgeBySize(edges=Right_edge, size=seedLeftRight,
         ↪  constraint=FIXED)

        self.Assembly.setMeshControls(regions=Face, elemShape=QUAD,
         ↪  technique=FREE)
        self.Assembly.setElementType(regions=self.Assembly.allSets[self.name + ".Face"], elemTypes=(elm,))

    def meshPart(self):
        self.Assembly.generateMesh((self.AssemblyInstance,))

    def createHistoryOuput(self,stepName, region, name = "Hist-2", variables = ["RF2", "U2", "CF2"]):
        self.model.HistoryOutputRequest(name=name, createStepName=stepName, region=region, variables=variables)

    def editFieldOutput(self, name = "F-Output-1", values = ["CF", "RF", "U", "S", "SDEG", "STATUS", "CYCLEINI", "E"]):
        self.model.fieldOutputRequests[name].setValues(variables = values)

    def createJob(self, jobName):
        self.jobs.append(mdb.Job(name = jobName, model = self.model))
        return self.jobs[-1]

    def submitJobAndWait(self, job):
        job.submit()
        job.waitForCompletion()

    def postProcessStressStrain(self, job, stepname = 'DC', XYplotname = 'nS'):
        import numpy as np
        cwd = os.getcwd() #path for this directory
        jobname = job.name
        path = cwd + '/' + jobname + '.odb'

        RFkey = 'Reaction force: RF2 PI: rootAssembly Node 2 in NSET RP_TB'
        CFkey = 'Concentrated force: CF2 PI: rootAssembly Node 2 in NSET RP_TB'
        U2key = 'Spatial displacement: U2 PI: rootAssembly Node 2 in NSET
        ↪  RP_TB'

        if XYplotname == 'nS':
            XYplotname = 'XYplot' + jobname

        self.xydataObj = postProcess(path = path, ovbU=U2key, Uname='U2' +
        ↪  jobname, ovbRF=RFkey, RFname='RF2' + jobname, setName = 'Temp' +
        ↪  jobname, XYplotname=XYplotname, stepName=stepname)
        temparray = np.zeros((len(self.xydataObj.data),2))
        self.data = ''
        for i,d in enumerate(temparray):
            d[0] = self.xydataObj[i][0]/self.h
            d[1] = self.xydataObj[i][1]/(2*self.w)
            self.data += str(d[0]) + ',' + str(d[1]) + '\n'

        pathToDataStorage =
        ↪  "C:/Users/katin/Documents/Studie/0_V2022/Thesis/FEM/Data/Data/"
        ↪  + jobname + "_stressStrain.txt"
        f = open(pathToDataStorage, 'w')
        f.write(self.data)
        f.close()

    def postProcessSDEG(self, job, stepname = 'DC'):
        cwd = os.getcwd() #path for this directory
        jobname = job.name
        setName = 'XYdata_' + jobname
        path = cwd + '/' + jobname + '.odb'

        #---------------------------------------------------------------
        from abaqus import *
        from abaqusConstants import *
        import numpy as np
        session.Viewport(name='Viewport: 1', origin=(0.0, 0.0),
        ↪  width=307.999969482422,
                         height=170.116683959961)
        session.viewports['Viewport: 1'].makeCurrent()
        session.viewports['Viewport: 1'].maximize()
        from viewerModules import *
        from driverUtils import executeOnCaeStartup
        executeOnCaeStartup()
        o2 = session.openOdb(name=jobname + '.odb')

        session.viewports['Viewport: 1'].setValues(displayedObject=o2)
        session.viewports['Viewport: 1'].makeCurrent()
        odb = session.odbs[
            path]
        #----------------------------------------------------------------
        mytuple = ()

        for i in range (len(self.AssemblyInstance.elements)):
            mytuple = mytuple + (xyPlot.XYDataFromHistory(odb=odb,
                outputVariableName='Scalar stiffness degradation: SDEG PI:
                ↪  '+ self.name.upper()+' Element '+str(i+1)+' Int Point 1
                ↪  in ELSET FACE',
                steps=(stepname, ), suppressQuery=True,
                ↪  __linkedVpName__='Viewport: 1'),)
        x_final = maxEnvelope(mytuple)
        xy_result = session.XYData(name=setName, objectToCopy=x_final)

        del mytuple
        self.xySDEGdataObj = session.xyDataObjects[setName]
        temparray = np.zeros((len(self.xySDEGdataObj.data), 2))
        self.SDEGdata = ''
        for i, d in enumerate(temparray):
            d[0] = self.xySDEGdataObj.data[i][0]/2
            d[1] = self.xySDEGdataObj.data[i][1]
```

```
                self.SDEGdata += str(d[0]) + ',' + str(d[1]) + '\n'

        pathToDataStorage =
         ↪  "C:/Users/katin/Documents/Studie/0_V2022/Thesis/FEM/Data/Data/"
         ↪  + jobname + "_SDEG.txt"
        f = open(pathToDataStorage, 'w')

        f.write(str(self.SDEGdata))
        f.close()

    def getDataFromFile(self, jobname):
        pathToDataStorage =
         ↪  "C:/Users/katin/Documents/Studie/0_V2022/Thesis/FEM/Data/Data/"
         ↪  + jobname + "_StressStrain.txt"
        f = open(pathToDataStorage, 'r')
        self.data = f.read()
        f.close()

    def readAndSortData(self, jobname):
        pathToDataStorage =
         ↪  "C:/Users/katin/Documents/Studie/0_V2022/Thesis/FEM/Data/Data/"
         ↪  + jobname + "_StressStrain.txt"
        f = open(pathToDataStorage, 'w')
        self.sortedData = self.data.replace('),', '\n')
        self.sortedData = self.sortedData.replace('(', '')
        self.sortedData = self.sortedData.replace(')', '')

        f.write(str(self.sortedData))
        f.close()

    def doItAll(self):
        #uses only default values

        #1. Draws sketch and makes part. Also defines edges and RPs to be used later
        self.createPart()
        #2. Defines material
        self.createMaterial()
        self.editMaterialElasticPlasticHardening()
        #3. Assigns section
        self.assignSection()
        #4. Creates Assembly
        self.createAssembly(instanceName = self.name)
        #5. Creates Step
        self.createDirectCyclicStep(maxNumCycles=2000)
        #5.5. Create partition
        self.createPartition()
        #6. Creates mesh
        self.createMesh()
        self.meshPart()
        #7. Creates PBC
        self.createPBC()
        #8. Assign Loads and BC
        self.createLoads(stepName = self.stepNames[-1], u2 = 0.8*(self.h/100), cf2=60*self.w*2)
        #9. Ask for History output
        self.createHistoryOuput(stepName = self.stepNames[-1], region = self.Assembly.allSets["RP_TB"])
        self.createHistoryOuput(stepName = self.stepNames[-1], region = self.Assembly.allSets[self.name + ".Face"], name = "Hist-3", variables=["SDEG"])
        #10. Edit Field Ouput
        self.editFieldOutput()
        #11. Edit keywords
        self.editKeywords()
```

Appendix B

B.1 Project Thesis Katinka Engen 2021

Data driven approach to bio-inspired structures

Specialisation Project

K. Engen

NTNU

MTP
NTNU
Norway
December 20, 2021

K. Engen December 20, 2021

Abstract

The Black drum is a saltwater fish with a diet consisting of crushing oysters and shellfish. It has one of the highest biting forces per weight [1], and for this it needs a powerful set of jaws that is resistant to cyclic loading, can handle high uni axial forces as well as being light weight. This preliminary examination of the micro structure of the lower pharyngeal jaw bone of the Black Drum fish aims to uncover structural properties fit for making other robust material that can handle uni-axial cyclic loading., e.g. for damping effects.

The lower pharyngeal jaw consists of a dental-plate and two supporting struts. These struts have a porous core, and denser walls. These walls have a volume-fraction of 40-60 percent [2]. This is a low volume fraction if compared to e.g. the density of mammalian bone [3][4][5]. Looking at a cross-section of the denser walls, we see thin plates oriented in the load-bearing direction, and supporting beams connecting these thin plates. These supportive beams can be there to stabilize, and to prevent shearing between the load-bearing plates [2]. This paper analyses the stabilizing effect the parameters of the micro structure has.

The thickness of the load-bearing plates, supporting beams and size and shape of the voids have implications for the stability. Despite this, it seems other mechanisms, e.g. shearing or fatigue are more likely to be the determining factors, as the structure's stability at the most unfavourable found in this examination is still able to withstand roughly twice the stress it is subject to during operation (150 MPa vs 80 MPa) [2].

Contents

Chapter 1

Theory

1.0.1 The Black Drum Jaw Bone

The Black Drum's jaw bone is subject to very high cyclic loading. The diet of the Black Drum consists mostly of shellfish and amonites. Yes, the structure of the bone is unlike that of wich we are familiar: Cortical and Trabecular bone from mammals, and to some degree bones from fish.

1.0.2 Comparison on different kinds of bone

Cortical Bone

The cortical bone is compact. In humans it consists of about 10% soft tissue, and makes up 80 % of the skeletal mass. It makes up all the outer layer of the bones, and is particularly found in weight-bearing areas such as the femur [3]. In Figure 1.1 a view of the human bone can be seen. In figure 1.2 a schematic drawing of the bovine cortical bone can be seen. As seen in the figure, it is made up of systematically placed lamellae, and is quite dense.

As far as I can understand from my limited review, bones in mammals are roughly similar. Though the bones and bone structure vary, the cortical and trabecular bone is found in most mammals, and consist of roughly the same features.

An interesting variety on the bone structure is found in fish, and it seems to be a field not as well studied as the mammalian bone. As of 2015 A. Atkins et al. wrote

While the structure of mammalian bones is therefore reasonably well studied

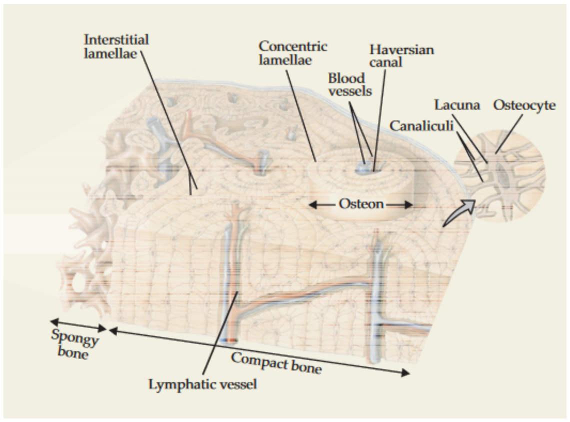


Figure 1.1: Macroscopic view of cortical and trabecular (here: spongy) bone. Figure is from G. J. Tortora [6]

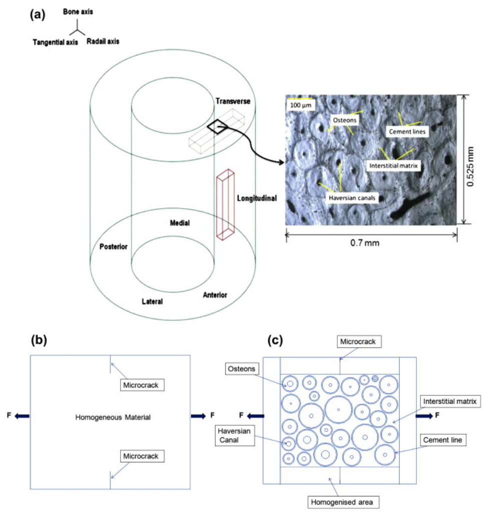


Figure 1.2: The bovine cortical bone. (a) Light-microscopy micrograph and its position in transverse-radial cross-section of osteonal bovine cortical bone tissue, (b) schematic illustration of homogeneous model, and (c) schematic illustration of micro-structural model. Figure is from A. A. Abdel-Wahab, A. R. Maligno, and V. V. Silberschmid [5]

> in three dimensions (...) similar data with regard to fish bone are lacking. In particular, the fibrillar arrangement in fish bone lamellae is unknown, as, indeed, is whether their layered structure consists of lamellar units at all [7].

Fish bone structures can be divided into two types: cellular and acellular [7][8]. The acellular, or anostecytic, bone of a fish was previously understood to be relatively featureless, but is now thought to be layered much like the mammalian lamellar bone, as well as consisting of a dense array of small-diameter collagen bundles. As the mammalian bone, the anostecytic bone consists of ordered material, but is less ordered. It also has much thinner individual lamellae, 1-2 vs. 2-7 μm. The bone studied in this paper also proves to be much tougher than mammalian bone, like the antlers of deer, without sharing the antler's micro structure or mechanisms [7].

The last section of the fish was provided to address the issue of specious variation. It seems the two-bone system found in mammals might not be directly comparable with fish. The skeletal of fish' are complicated, and the fish-bone described are one type from one fish.

Loading direction

Suitable to withstand unidirectional forces, along the length of the fibers. The macro structure of the bone, with this compact material along the periphery, enables it to withstand bending-forces effectively.

Trabecular Bone

The trabecular bone is a spongy form of bone, without any systematic placement of fiber or lamellae, but with rods 100 μm thick, and holes 1 mm thick [9]. Bone marrow makes up about 75 % of its volume [3]. The histography of the trabecular bone also varies across species. One example is the ostrich, equine (horse) and human. The morphology of the ostrich trabecular bone is utterly different from the equine trabecular metatarsal, while the nanomechanical properties ostrich trabecular bone is similar to the human bone [10].

Loading direction

Due to its randomly-oriented fibers, the trabecular bone is able to withstand multidirectional loading. This could be the reason it's found near the joints, where loading direction can vsry. It is not able to withstand as much pressure as cortical bone, because of it's porosity, which could be why bones often increase in size near the joints. Trabecular bone is also found in the central regions of long bones, e.g. the femur, where it can support the cortical bone. I also noted in the article about the LPJ bone a theory about the trabecular bone adapting to the direction of the largest forces [11].

1.1 The Black Drum Lower pharyngeal jaw bone

1.1.1 Bone structure

The geometry and topology of the lower pharyngeal jaw (LPJ) bone seem to differ from cortical and trabecular bone [2]. LBJ seem to be less dense than the cortical bone of mammals, and not as porous as the trabecular bone. The porosity of the outer wall of the struts of the bone have a porosity level of about 50 %. This is far more porous than the cortical bones found at the walls of bones i mammals, in which no more than 3% are voids.

The lower pharyngeal jaw is divided in two. It's dental plate consists of two halves, with a suture in the middle. On the plates there are molars, and the molars are larger near the suture (middle). Each half of the dental plate rests on the thick end of a cone shaped strut.

A cross-section of the struts show a dense, outer wall, and a porous middle-section, as with cortical bone and trabecular bone in mammalian bone. However, the structure of the outer wall and the middle-section in the LPJ-bone are not like the cortical and trabecular bone discussed previously.

The outer walls consist of plates (lamellar sheets) orienten in the load bearing direction, i.e. along the length of the strut (from now on referred to as the z-axis). They are slightly curved around the z-axis. In the void between each such plate there are several thinner, transversely oriented beams, dividing and supporting the plates. These supportive beams are oriented orthogonal to the load-bearing direction.

The central region of the bone consists of thin rods, and is irregular. Like the trabecular bone it is non-systematic in the structure, however it also varies in porosity. While the mammalian bone consist of a uniform distribution of bone matter in the trabecular bone [11], here the bone can have pores more than 10 times the size of other pores, seemingly without a predictable pattern.

Loading direction

The thick plates in the the exterior wall of the struts seem to carry the load. On the macro level, the denser outer wall prevents bending of the strut. The thinner, curved beams, supporting the load-bearing plates, could protect against in-plane shear-forces and stabilize the structure on a macro-level [2].

The fibers of the thick plates in the outer wall, are oriented along the z-axis, i.e. in the loading direction. This is similar to the fibers in cortical bone. For the supporting beams, the fibers merge into the thicker plates, much like we see for trabecular bone in joints.

1.2 Buckling

Structures subject to compressed load sometimes encounter a stability-problem if the structure is slender enough, such that for a critical load, P$_{cr}$, the structure looses its stability while the material still behaves linear-elastic [12]. The critical load for a beam is defined as

$$P_{crit} = \frac{\pi^2 EI}{L_k^2}, \tag{1.1}$$

where E is the elastic modulus of the material of the beam, I is the second moment of area, and L$_k$ is the buckling length. The buckling length is the effective length of the structure, and will be a function of the total length of beam, and how the structure is constrained. An illustration of this can be seen i figure 1.3.

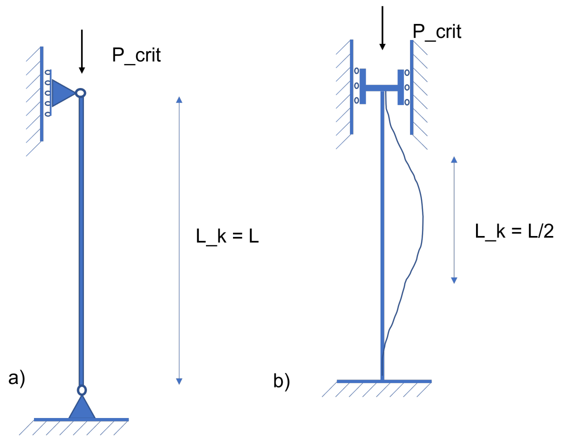


Figure 1.3: Illustration of two known buckling modes.

The critical stress, σ_{crit}, and the critical strain, ϵ_{crit} for when a structure becomes unstable is defined as

$$\begin{aligned}\sigma_{crit} &= \frac{P_{crit}}{A}, \\ \epsilon_{crit} &= \frac{\Delta L}{L}\end{aligned} \tag{1.2}$$

where A is the area of the cross-section, and ΔL is the deflection of the structure in the compressed direction as it becomes unstable, and L is the original length of the structure.

1.2.1 Bracing

From equation 1.1, we see that decreasing L_k with a factor 2, increase the critical load, P_{crit} with a factor 4. For long, slender beams, introducing some support along the length of the beam that stabilizes against the buckling-mode can be effective. An example is seen in figure 1.4.

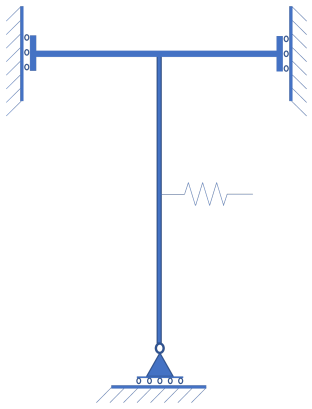


Figure 1.4: Example of bracing: a spring stabilizing against buckling.

In the figure, a spring is placed on the middle of the length of the beam. Depending on the stiffness of the spring, the critical load of the beam could be as much as four times as large as it was without the spring [12].

1.3 Micromechanical modelling and Periodic boundary conditions

Representative volume elements (RVEs) can be used when studying the deformation mechanisms of a porous material on the microscopic level. For a spatially periodic and space-filling RVE, the use of periodic boundary conditions (PBCs) on the surface (3D) or edges (2D) of the RVE ensures that it deforms in a periodic manner [13] [14].

1.3.1 Periodic Boundary conditions in 2 dimensions

When simulation mechanical deformation of an RVE, the modell must fulfill certain requirements. The RVE must deform in a periodic manner. This entail that deformation of the RVE happens in such a manner that it never ceases to be periodic and spatially filling. I.e no cavities or overlaps forms. The displacement of each pair of periodically placed points A and B is described as

$$u(B) - u(A) = (\overline{F} - 1)(X(B) - X(A)) = \overline{H}(X(B) - X(A)). \tag{1.3}$$

Here u(A) and u(B) is displacement at point A and B, respectively, ($\overline{F}$-1) is the macroscopic displacement, and X(A) and X(B) is position in reference configuration. Each periodic pair of nodes along the edges of the RVE must be constrained with this relation. [15]. The macroscopic displacement is applied to 'dummy' nodes. This is a node unconnected to the RVE itself. For 2-dimensional RVEs there would be 2 'dummy' nodes, while for 3-dimensional RVEs there would be 3.

Using the numerical simulation tool Abaqus, the constraint described in equation 1.3 is implemented using a constraint-equation. For a periodic pair C and D on a 2-dimensional RVE, the equations imposed on the pair would be

$$\begin{aligned} u(C)_1 - u(D)_1 - u(P)_1 &= 0 \\ u(C)_2 - u(D)_2 - u(P)_2 &= 0, \end{aligned} \tag{1.4}$$

where $\mathrm{u(X)_i}$ denotes the displacement in point X in direction i [14]. Point P refers to the 'dummy' node associated with the edge-pair the points C and D belongs to.

1.4 Mean values and Standard deviation

Some known formulas for calculating standard deviation of a nonlinear function with multiple variables, where these variables themselves possess an error in the form of a standard deviation is found in table 1.1.

Table 1.1: Formulas for calculating error propagation [16]

Function	Standard Deviation
$\mathrm{f} = \frac{\mathrm{A}}{\mathrm{B}}$	$\sigma_{\mathrm{f}} = \lvert\mathrm{f}\rvert\sqrt{(\frac{\sigma_{\mathrm{A}}}{\mathrm{A}})^2 + (\frac{\sigma_{\mathrm{B}}}{\mathrm{B}})^2 - 2\frac{\sigma_{\mathrm{AB}}}{\mathrm{AB}}}$
$\mathrm{f} = \frac{\mathrm{A}}{\mathrm{B}}$	$\sigma_{\mathrm{f}} = \lvert\mathrm{f}\rvert\sqrt{(\frac{\sigma_{\mathrm{A}}}{\mathrm{A}})^2 + (\frac{\sigma_{\mathrm{B}}}{\mathrm{B}})^2 + 2\frac{\sigma_{\mathrm{AB}}}{\mathrm{AB}}}$
$\mathrm{f} = \mathrm{aA^b}$	$\sigma_{\mathrm{f}} = \lvert\frac{\mathrm{fb}\sigma_{\mathrm{A}}}{\mathrm{A}}\rvert$

Chapter 2

Method

In order to analyse the structural properties of the structure through Finite Element Analysis, a simplified lattice structure had to be defined based on the available data from the real bone.

2.1 Defining the lattice structure

The lattice structure's parameters where defined looking at the scans an 3D rendering of the jaw bone of the Black Drum [2], shown in figure 2.1.

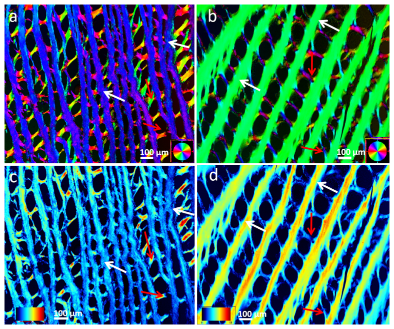
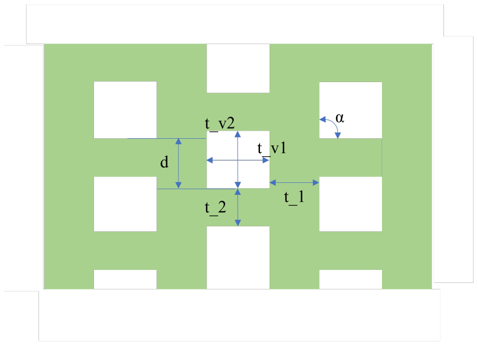


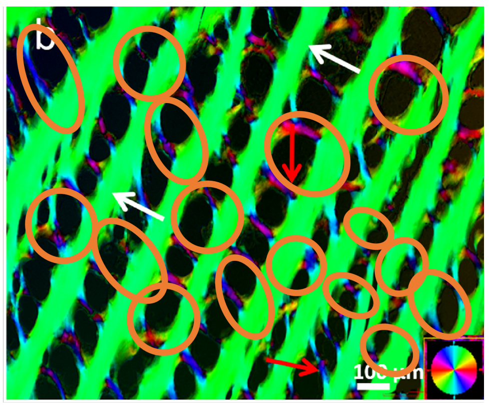
Figure 2.1: Scan of the jaw-bone of the Black Drum. Figure is from E. Ziv et al. [2].

As can be seen from figure 2.1, the bone is disordered and have significant variations in the thickness of the lamellar plates, both the vertical and the horizontal. A simplified model of the structure was made (figure 2.2), and the key parameters where defined. To define a mean value for each of the key parameters, 16 points where randomly chosen on the scans and a mean value and standard deviation was calculated. These points can be seen in Figure

2.2.



a)



b)

Figure 2.2: a) Sketch of simplified lattice structure with key parameters based on the scan. b) Scan of the LBJ bone [2]. 16 areas for measuring 5 key parameters tv1, t_{v2}, t$_1$, t2, $and\alpha$ are highlighted. Background figure in b) is from E. Ziv et al. [2].

The resulting mean values and standard deviations is reported in table 2.1 These values were used as basis in the modelling.

Table 2.1: Mean values and Standard Deviations for four of the key parameters.

Parameter	Mean value	Standard Deviation [μm]	Standard Deviation [%]
t_1	71.12 μm	15.94	22.4
t_{v1}	94.12 μm	28.03	29.8
t_2	27.37μm	10.48	38
t_{v2}	106.19 μm	30.29	28.5
α	77.69°	12.2°	15.7

Based on the mean values for the key parameters, the volume fraction, V_f, is

$$\begin{aligned} V_f &= \frac{A_{tot} - A_{void}}{A_{tot}} = 1 - \frac{A_{void}}{A_{tot}} \\ A_{tot} &= (t_1 + t_{v1}) * (t_2 + t_{v2}) \\ A_{void} &= t_{v1} * t_{v2} \end{aligned} \tag{2.1}$$

$$V_f = 0.547,$$

where A$_{tot}$ denotes the total area, and A$_{void}$ denotes the void area. Note here that the volume fraction based on the values in table 2.1, calculated in equation 2.1 is 0.547.

Unitless ratios describing the relationship between the parameters is found in table 2.2. The standard deviations were calculated based on the equations in section 1.4.

Table 2.2: Mean values and Standard Deviations for four of the key parameters.

Parameter	Mean value	Standard Deviation [μm]	Standard Deviation [%]
$\frac{t_{v2}}{t_{v1}}$	1.128	0.465	41.2
$\frac{t_1}{t_2}$	2.598	1.153	44.4
Vf	0.547	0.331	60.57
α	77.69°	12.2°	15.7

For simplicity, the distance d was defined as

$$d = h, \tag{2.2}$$

where

$$h = t_{v2} + t_2. \tag{2.3}$$

As can be seen from figure 2.1, the voids in the bone is not completely rectangular as in the simplified lattice structure in figure 2.2 a). An alternative simplification with elliptical voids was created for comparison and is shown in figure 2.3.

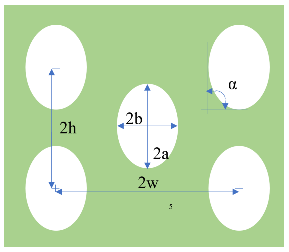


Figure 2.3: Structure imitating the elliptical shapes of the void.

In order to maintain a valid comparison of the simplifications, the parameters introduced in figure 2.3 were defined by the values from the structure in figure 2.2 with the following relation:

$$\begin{aligned} a &= \frac{t_{v2}}{2} \\ b &= \frac{t_{v1}}{2} \end{aligned} \tag{2.4}$$

Doing only this, however, changes the volume fraction, as the hollow area now is smaller, but the overall size is the same, as can be seen in equation 2.5.

$$\begin{aligned} V_f &= \frac{A_{tot} - A_{void}}{A_t} \\ A_{tot} &= (t_1 + t_{v1}) * (t_2 + t_{v2}) \\ A_{void} &= \pi ab \end{aligned} \tag{2.5}$$

$$V_f = 0.644$$

In order to maintain the original volume fraction, the ratio t_1/t_2, and the ratio a/b, a and b was scaled (equation 2.6).

$$\begin{aligned} c &= \frac{a}{b} = \frac{t_{v2}}{t_{v1}} \\ a &= cb \end{aligned}$$

$$V_f = 1 - \frac{\pi ab}{A_{tot}} = 1 - \frac{\pi cb^2}{A_{tot}} \tag{2.6}$$

$$\begin{aligned} b &= \sqrt{\frac{A_{tot}(1 - V_f)}{\pi c}} \\ a &= cb \end{aligned}$$

2.2 Understanding the structure

2.2.1 Bracing

Looking at the scans of the bone [2] the micro-structure of the bone seems to consist of several slender walls running in the direction the force applied, with several short supports running near-orthogonal to these slender walls 2.1. These short, orthogonal supports could contribute to stability of the structure, as well as preventing inter-lamellar shearing.

Considering the stability of the structure, we can view the orthogonal supporting plates as stabilizing the longitudinal lamellae, illustrated in figure 2.4, according to the theory of bracing in section 1.2. Note: as the longitudinal lamella, as well as they're orthogonal supports are very thick out of plane it is reasonable to assume stability would first and foremost be an issue in plane.

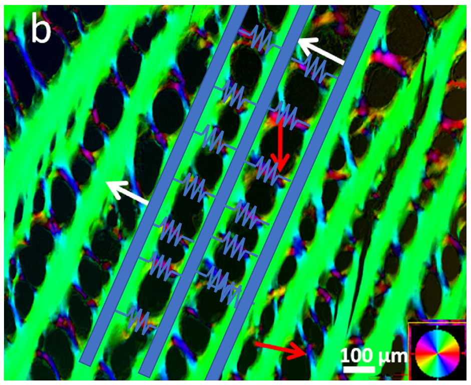


Figure 2.4: Orthogonal supports viewed as springs stabilizing the thick lamellae. Figure is from E. Ziv et al. [2].

Here the springs would have stiffness k, and the longitudinal beam would have the critical load P$_{crit}$. They're relationship with the parameters of the structure (figure 2.5) is defined in equation 2.7.

$$\begin{aligned} k &= \frac{EA}{L} \\ A &= t_2 \cdot t_3 \\ L &= t_{v1} \\ k &= A\frac{t_2}{t_{v1}} \end{aligned} \tag{2.7}$$

$$\begin{aligned} P_{crit} &= \frac{\pi^2 EI}{L_k^2}, \\ I &= \frac{t_1^3 \cdot t_3}{12} \\ L_k &= L_k(t_{v2}, k) \end{aligned}$$

Here E is the young modulus, A the area of the cross section and L the length of the supporting beam. t_3 is the out-of-plane thickness of the structure. I is the second area of moment of the longitudinal beam about the out-of plane axis, and L_k is the effective length of the beam (section 1.2).

This neglects part of the resistance against bending that the orthogonal supports provide, i.e. if the longitudinal lamellae where to bend, this would cause bending in the supporting beams as well. This bending of the supporting beams also provide a resistance against bending of the longitudinal lamellae, and would therefore affect the buckling mode of the longitudinal beam. The supports can break or buckle before the longitudinal lamellae would, due to compressive forces between two longitudinal lamellae, depending on the stability of the supports relative to the longitudinal lamellae.

Based on these assumptions, it would seem reasonable to assume that some of the parameters defining the structure play a more significant role regarding the stability than others.

In order to understand the structure of the bone, and evaluate its success factors completely, several aspects of the structure can be evaluated. A severely simplified lattice structure can be described using rectangular-shaped voids, or elliptical voids. The lattice-structure has an angle, α, and the ratios t_{v2}/t_{v1}, t_{v1}/t_1, t_{v2}/t_2 and t_1/t_2.

Evaluation of the two proposed structures (figure 2.2 and 2.3) and the ratios t_{v2}/t_{v1} and

t_1/t_2, was carried out for $\alpha = 90°$ in the way shown in table 2.3. The ratios for t_{v2}/t_{v1} and t_1/t_2 was chosen based on their respective mean values and standard deviations in table 1.1.

First the structures in figure 2.2 and figure 2.3 will be evaluated for different ratios of t_{v2}/t_{v1} and t_1/t_2. This will be done for the angle, $\alpha = 90°$ for both structures (figure 2.2 and 2.3). An overview of the first tests to be done is in table 2.3.

Table 2.3: Tests to evaluate the ratios t_{v2}/t_{v1}, t_1/t_2, and elliptical vs. rectangular shaped voids.

Structure	Rectangular void							Elliptical void						
Angle	90°							90°						
t_{v2}/t_{v1}	0.6	0.8	1	1.2	1.4	1.6	1.8	0.6	0.8	1	1.2	1.4	1.6	1.8
t_1/t_2	0.5	1	1.5	2	2.5	3	3.5	0.5	1	1.5	2	2.5	3	3.5

The tests in table 2.3 will be run on for a low volume fraction, 0.4, based on the mean value and standard deviations in table 1.1.

2.3 Finite element model

The finite element model is made in 2D, planar using shell elements. As the focus is the simplified lattice structure, and what can be learned from the structure of the bone using that as a tool, PBC boundary conditions where used (1.3.1). The purpose of these tests will be to evaluate the geometric contribution of the structure to its stiffness, and the material will therefor be elastic with parameters E and ν in table 2.4.

Table 2.4: Material data of the LPJ bone [2]

E	6010 MPa
ν	0.3

The representative volume elements (RVE) used to represent the different structures presented in the last sub-chapter is presented in figure 2.5.

The RVE will be subjected to one linear perturbation with displacement, and one post Buckling analysis - Static Step, non-linear geometry. The different test-samples will then be compared on the critical strain, ϵ_{crit}, and critical stress σ_{crit}. During operation the bone of the fish is subject to forces longitudinal direction. This is simulated by a displacement applied in y-direction to the dummy-node connecting the top- and bottom edges in the PBC.

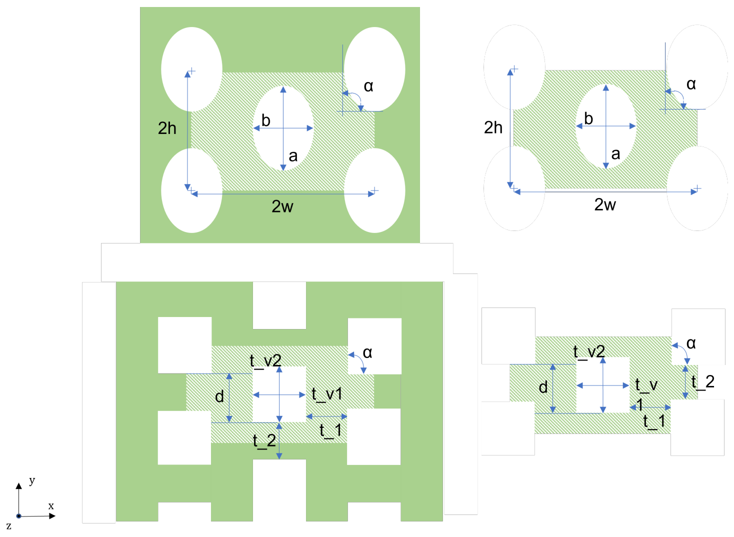


Figure 2.5: RVE's from the structures.

The critical stress and strain is defined as

$$\begin{aligned} \epsilon_{crit} &= \frac{u_y}{H} \\ \sigma_{crit} &= \frac{RF_y}{A}, \end{aligned} \tag{2.8}$$

where H is the total underformed height of the RVE, A is the total undeformed area of the RVE, u$_y$ is displacement in y-direction, and RF$_y$ is reaction force in y-direction recorded in the dummy nodes.

The constraints of a structure is significant for the buckling mode of it [12]. As PBC enforces constraints on the single RVE that prevents it from behaving differently from it's neighbour [15], it could be that there exists global buckling modes with a lower eigenvalue than the local ones that is not detected by the single RVE. To check for global buckling modes eigenvalue tests will also be run on "supercells" consisting of mxm RVEs.

Chapter 3

Results

3.1 The lattice structure

3.2 Buckling modes

To check for global buckling, simulations where run on a supercell, or a 2x2 RVE. The analysis on the 2x2 RVE showed there are global buckling modes for this structure that the 1 RVE cannot capture. To check for lower globl buckling modes, a convergence study was done on the elliptical RVE (figure 3.1), where the first eigenvalue was found for a configuration of m x m RVEs. 3.1.

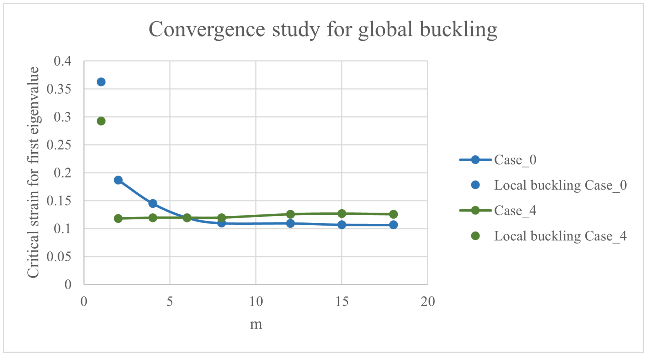


Figure 3.1: Convergence study of first global Buckle mode. m is the square root of the number of RVE in the supercell, $case_0$ and $case_4$ has a ratio $\frac{t_{v2}}{t_{v1}}$ of 0.8 and 1.6 respectively. The critical strain is given in absolute value.

3.2.1 Local buckling

The eigenvalue analysis on the single cell RVE showed one buckling mode was consistent as the first, and critical buckle mode. The critical strain for first local buckling mode is in the range 6-35 % (figure 3.2).

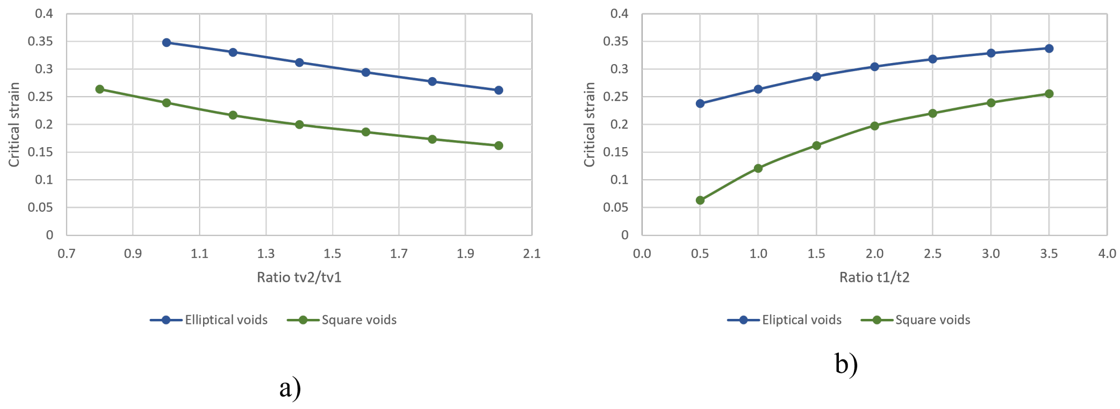


Figure 3.2: a) Critical strain for first local buckle mode for different values of t$_{v2}$/t$_{v1}$. Two values for the elliptical voids is missing due to a convergence issue with the Post Buckling analysis. b) Critical strain for first local buckle mode for different values of t_1/t_2.

3.2.2 Global buckling

For the 2x2 RVE "supercell", the critical stress lies in the range 6-22 % (figure 3.3, figure 3.4). The critical nominal stress lies in the range 150-875 MPa.

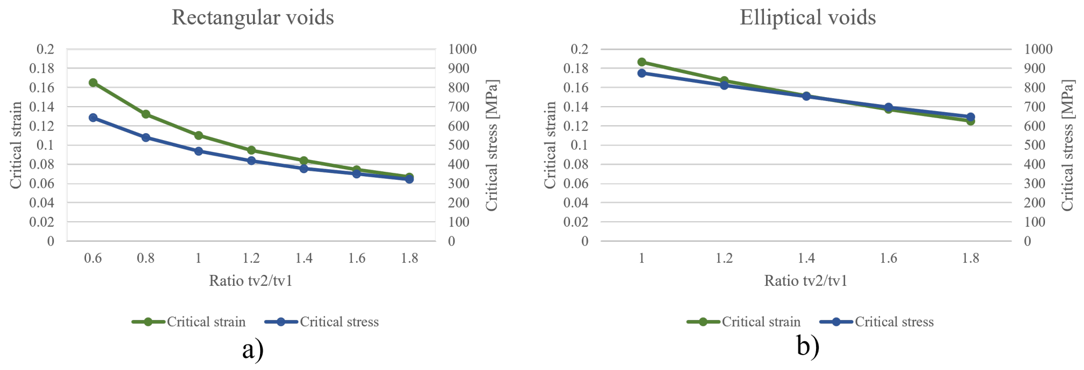
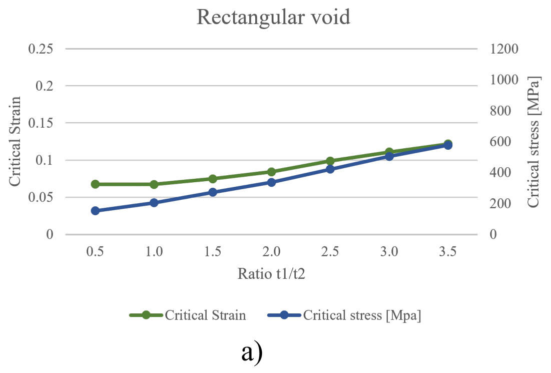


Figure 3.3: Critical stress and strain for different values of the ratio $\frac{t_{v2}}{t_{v1}}$ for structure with elliptical voids (a), and rectangular voids (b).



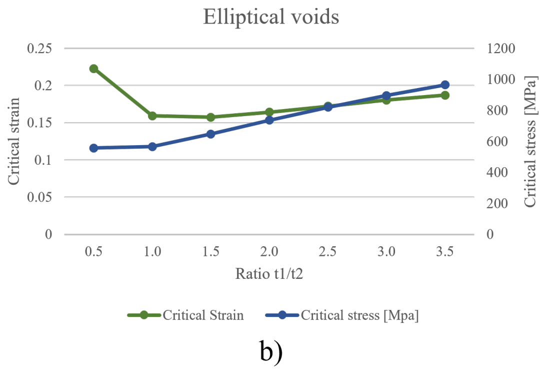


Figure 3.4: Critical stress and strain for different values of the ratio $\frac{t_1}{t_2 2}$ for structure with elliptical voids (a), and rectangular voids (b).

For the rectangular structure (figure 2.2), the numerical simulation predicts the same buckle mode for $\frac{t_1}{t_2}$=0.5, as for local buckling. For all other, the prediction is global buckling, with lower critical strain for the first buckling mode.

Chapter 4

Discussion

4.1 Convergence analysis

From the convergence analysis it seems global buckling will occur before local buckling, though this to ascertain this test should be run on the structure with rectangular voids. The result also varies between the two ratios that where run, so a convergence analysis should also be done on at least one more ratio to determine a pattern for the convergence.

4.2 Rectangular vs. elliptical voids

The structure with elliptical voids is significantly more stable than the structure with rectangular voids for the same parameter values. There seems to be no exception for this based on the plots in figure 3.3 and 3.4.

Sources for error

The

4.3 $\mathbf{t}_{v2}$ vs. $\mathbf{t}_{v1}$

From the stability analysis of the 2x2 RVE in figure 3.3, we see that the larger t_{v2} is relative to t_{v1} within the standard deviation of the measurements, the lower the critical strain and stress.

This supports the fact that the critical length of the load-bearing plate increase with t$_{v2}$, witch affects the critical load to the power of two (equation 1.1), while the stiffness of the supporting beam is increased with decreased t$_{v1}$ to the power of one (equation 2.7).

It also seems that buckling of the supporting beams is not a determining factor for the critical strain.

There is no obvious optimal ratio within the standard deviation range of the ratio, as the trendline suggest increasing the ratio further would be better for the stability. This suggests something other than stability is the determining factor for this ratio, e.g. fatigue or shear-stabilization.

4.3.1 Sources of error

One source of error here is the missing results of the structure with elliptical voids for the t_{v2}/t_{v1} ratios 0.8 and 0.6 due to convergence issues with the numerical analysis. For this structure we are left to interpret the trendline of the five remaining results.

4.4 t$_1$ vs t$_2$

From the stability analysis of the 2x2 RVE in figure 3.4, we see that the larger t1 is compared to t2, the higher the critical strain and stress. This conforms with the theory of bracing and buckling. t$_1$ increases the beams' second area of moment for in-plane buckling to the power of three, while t2 only increase the stiffness of the support with to the power of one (section 1.2.

As with the ratio t_{v1}/t_{v2}, there is no obvious optimal ratio within the standard deviation range of the ratio, as the trendline suggest decreasing the ratio further would be better for the stability. This suggests something other than stability is the determining factor for this ratio, e.g. fatigue or shear-stabilization.

4.5 Significance

Overall, it seems that instability occurs at a strain and stress higher that what it is subject to [2]. The lowest critical strain found for a low volume fraction of 0.4, is 150 MPa.

4.6 Missing pieces

Some tests where not completed. To understand the structure further, the tests done one the 2x2 RVE (figure 3.3 and 3.4) should have been one on a 6x6 RVE following the results of the convergence analysis of global buckling modes. A convergence analysis should also have been done on the structure with rectangular voids, and for more ratios to determine a pattern.

The impact the angle, α, and d has on the stability of the structure is also not known.

Bibliography

[1] J. R. Grubich. "Disparity between Feeding Performance and Predicted Muscle Strength in the Pharyngeal Musculature of Black Drum, Pogonias cromis(Sciaenidae)". In: *Environmental Biology of Fishes* 74.3 (2005), pp. 261–272. ISSN: 1573-5133. DOI: 10.1007/s10641-005-3218-0. URL: https://doi.org/10.1007/s10641-005-3218-0.

[2] E. Ziv et al. "Neither cortical nor trabecular: An unusual type of bone in the heavy-load-bearing lower pharyngeal jaw of the black drum (Pogonias cromis)". In: *Acta Biomaterialia* 104 (2020), pp. 28–38. ISSN: 1742-7061. DOI: https://doi.org/10.1016/j.actbio.2020.01.001. URL: https://www.sciencedirect.com/science/article/pii/S1742706120300027.

[3] M. Monier-Faugere, M. Chris Langub, and H. H. Malluche. "Chapter 8 - Bone Biopsies: A Modern Approach". In: *Metabolic Bone Disease and Clinically Related Disorders (Third Edition)*. Ed. by Louis V. Avioli and Stephen M. Krane. Third Edition. San Diego: Academic Press, 1998, 237–280e. ISBN: 978-0-12-068700-8. DOI: https://doi.org/10.1016/B978-012068700-8/50009-8. URL: https://www.sciencedirect.com/science/article/pii/B9780120687008500098.

[4] S. M. Ott. "Cortical or Trabecular Bone: What's the Difference?" In: 47 (2018), pp. 373–375. DOI: https://doi.org/10.1159/000489672.

[5] A. A. Abdel-Wahab, A. R. Maligno, and V. V. Silberschmidt. "Micro-scale modelling of bovine cortical bone fracture: Analysis of crack propagation and microstructure using X-FEM". In: *Computational Materials Science* 52.1 (2012). Proceedings of the 20th International Workshop on Computational Mechanics of Materials - IWCMM 20, pp. 128–135. ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2011.01.021. URL: https://www.sciencedirect.com/science/article/pii/S0927025611000450.

[6] G. J. Tortora. *Principles of Human Anatomy*. Sixth edition. New York: John Wiley Son, 2002.

[7] A. Atkins et al. “The three-dimensional structure of anosteocytic lamellated bone of fish”. In: *Acta Biomaterialia* 13 (2015), pp. 311–323. ISSN: 1742-7061. DOI: https://doi.org/10.1016/j.actbio.2014.10.025. URL: https://www.sciencedirect.com/science/article/pii/S174270611400467X.

[8] B. K. Hall. “Chapter 2 - Bone”. In: *Bones and Cartilage (Second Edition)*. Ed. by B. K. Hall. Second Edition. San Diego: Academic Press, 2015, pp. 17–42. ISBN: 978-0-12-416678-3. DOI: https://doi.org/10.1016/B978-0-12-416678-3.00002-1. URL: https://www.sciencedirect.com/science/article/pii/B9780124166783000021.

[9] M. Buehler R. Ritchie and P. Hansma. “Plasticity and toughness in bone”. In: *Physics Today - PHYS TODAY* 62 (June 2009). DOI: 10.1063/1.3156332.

[10] C. E. Ramírez A et al. “Assessing mechanical behavior of ostrich and equine trabecular and cortical bone based on depth sensing indentation measurements”. In: *Journal of the Mechanical Behavior of Biomedical Materials* 117 (2021), p. 104404. ISSN: 1751-6161. DOI: https://doi.org/10.1016/j.jmbbm.2021.104404. URL: https://www.sciencedirect.com/science/article/pii/S175161612100093X.

[11] P. Xiao et al. “Can DXA image-based deep learning model predict the anisotropic elastic behavior of trabecular bone?” In: *Journal of the Mechanical Behavior of Biomedical Materials* 124 (2021), p. 104834. ISSN: 1751-6161. DOI: https://doi.org/10.1016/j.jmbbm.2021.104834. URL: https://www.sciencedirect.com/science/article/pii/S1751616121004756.

[12] K. Bell. *Konstruksjonsmekanikk, Del II Fashetslære*. First edition. Fagbokforlaget Vigmostad Bjørke AS, 2015. Chap. 15.

[13] M. Danielsson, D.M. Parks, and M.C. Boyce. “Three-dimensional micromechanical modeling of voided polymeric materials”. In: *Journal of the Mechanics and Physics of Solids* 50.2 (2002), pp. 351–379. ISSN: 0022-5096. DOI: https://doi.org/10.1016/S0022-5096(01)00060-6. URL: https://www.sciencedirect.com/science/article/pii/S0022509601000606.

[14] M. Okereke and S. Keates. *Finite Element Applications, A Practical Guide to the FEM Process*. Springer International Publishing AG, 2018. Chap. 8.

[15] M. Danielsson. "Micromechanics, macromechanics and constitutive modeling of the elasto-viscoplastic deformation of rubber-toughened glassy polymers". PhD thesis. Massachusetts Institute of Technology, 2003.

[16] A.A. Clifford. *Multivariate error analysis : a handbook of error propagation and calculation in many-parameter systems*. eng. London, 1973.

Engen, Katinka Hårdvik

Machine-learning approach to design fatigue-resistant structure inspired by Pogonias cromis

NTNU

Norwegian University of
Science and Technology