UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica

Studio di effetti microstrutturali attraverso teoria elastica a gradienti superiori: implementazione nella piattaforma FEniCS

Investigation of microstructural effects through higher gradient elasticity: implementation in the FEniCS computing platform

Candidate:
Greta Gasparini

Advisor:
Prof. Pierpaolo Belardinelli

Academic Year 2022-2023

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica

Studio di effetti microstrutturali attraverso teoria elastica a gradienti superiori: implementazione nella piattaforma FEniCS

Investigation of microstructural effects through higher gradient elasticity: implementation in the FEniCS computing platform

Candidate:
Greta Gasparini

Advisor:
Prof. Pierpaolo Belardinelli

Academic Year 2022-2023

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica
Via Brecce Bianche – 60131 Ancona (AN), Italy

Ai miei genitori, per aver sempre creduto in me.
Ai miei nonni Ciro, Clara e Fiorella.
A mio nonno Dolfo, che mi guarda dall'alto,
spero sia orgoglioso di me.

Ringraziamenti

Dedico questo spazio a tutti coloro che mi vogliono bene e che mi sono stati accanto durante il mio percorso universitario.

Innanzitutto vorrei ringraziare il mio relatore e professore Pierpaolo Belardinelli, per il costante supporto e gentilezza mostrata in questi mesi.

Ringrazio tutta la mia famiglia, mamma, papà, Giulio, Bovary e Sally per avermi sempre incoraggiata.

A mia mamma Laura, da sempre grande sostenitrice e fonte di sostegno e conforto. Grazie per l'amore che mi dimostri ogni giorno e grazie per essere la mamma che sei.

A mio papà Giacomo, grazie al suo lavoro costante mi ha insegnato che solo attraverso grandi sacrifici si possono ottenere grandi risultati.

Mamma e papà spero che voi possiate essere orgogliosi di me. Un infinito grazie per esserci sempre.

A mio fratello Giulio, grazie per aver contribuito al mio percorso universitario pur non essendone affine.

Ai miei nonni Ciro, Clara e Fiorella per l'affetto che non mi hanno mai fatto mancare.

A mio nonno Dolfo, quanto avrei voluto che mi vedessi con la corona d'alloro in testa, ma sono sicura che mi starai guardando dall'alto con un gran sorriso vantandoti di avere una nipote laureata.

A mio zio Luca, fisicamente lontano ma emotivamente sempre vicino.

Al mio fidanzato Luca, grazie per avermi sempre sopportato e supportato in questi anni passati insieme.

Infine, vorrei ringraziare me stessa, per la dedizione e per aver creduto costantemente nel mio potenziale.

Ancona, Luglio 2023

Greta Gasparini

Abstract

This thesis investigates microstructural effects through the generalization of the classical elasticity theory, paying special attention to the theory of higher order deformation gradient elasticity. As a matter of fact, many theories have been developed over the years to untangle the role of microstructural effects in the mechanical behavior of structures. The theory presented in this thesis finds application in the field of metamaterials, engineered materials where low dimensional effects significantly influence the final mechanical characteristics. The birth and applications of the elastic theories at higher gradients are reported to provide proper background on the mathematical formulation. Then, a focus on the general strain gradient elasticity show how well-formulated hypotheses reduce the initial dependence from five parameters to three. This step allows to easily give physical interpretation of the extended theory. A further reduction is provided by the couple stress with the use of a single parameter, and in which only the symmetric part of the rotation gradient tensor plays a role. The use of a minimum number of parameters is an important simplification in the study of the behavior of materials. Furthermore, this thesis reports a practical implementation of non-classical theories within the FEniCS platform. FEniCS is a simple and effective software for solving partial differential equations using finite-element methods. The mathematical framework of non-classical elasticity theories is implemented in FEniCS via the variational formulation of the equations. Finally, a the case study is parametrically analyzed, and discussed by means of the Paraview software in order to highlight the effects of microstructures.

Sommario

L'obiettivo di questa tesi consiste nello studio degli effetti microstrutturali attraverso la generalizzazione della teoria classica dell'elasticità, in particolare si presta attenzione alla teoria dell'elasticità del gradiente di deformazione di ordine superiore. Nel corso degli anni sono state sviluppate numerose teorie atte a comprendere il ruolo degli effetti microstrutturali nel comportamento meccanico delle strutture. La teoria presentata in questa tesi trova applicazione nel campo dei metamateriali, materiali ingegnerizzati dove bassi effetti dimensionali influenzano significativamente le caratteristiche meccaniche finali. Nella trattazione sono riportate la nascita e le applicazioni delle teorie elastiche a gradienti superiori per fornire un adeguato background alla formulazione matematica. In seguito, un focus sull'elasticità del gradiente di deformazione generale mostra come ipotesi ben formulate riducano la dipendenza iniziale da cinque parametri a tre parametri. Questo passo permette di dare un'interpretazione fisica della teoria estesa. Un'ulteriore riduzione è fornita dalla teoria "couple stress" con l'utilizzo di un unico parametro, e in cui gioca un ruolo importante solo la parte simmetrica del tensore del gradiente di rotazione. L'utilizzo di un numero minimo di parametri rappresenta un'importante semplificazione nello studio del comportamento dei materiali. Inoltre, questa tesi riporta un'implementazione pratica di teorie non classiche all'interno della piattaforma FEniCS. FEniCS è un software semplice ed efficace per risolvere equazioni alle derivate parziali tramite l'utilizzo dei metodi agli elementi finiti. Il quadro matematico delle teorie dell'elasticità non classiche è implementato in FEniCS tramite la formulazione variazionale delle equazioni. Infine, il caso studio viene analizzato parametricamente e discusso mediante il software Paraview per evidenziare gli effetti delle microstrutture.

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

1.1. Motivation and state of the art

Microstructural effects play a very significant role in the mechanical behavior of materials [1]. A simple example of a microstructural effect can be seen in a metamaterial that is able to rotate on itself when subjected to a compression stress (Fig.1.1). Typ-

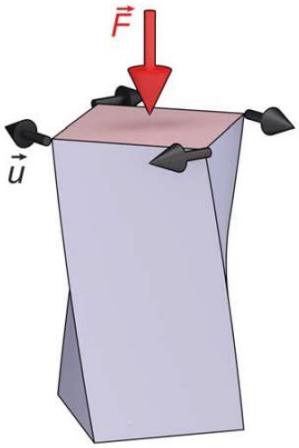

Figure 1.1.: A twist forbidden in ordinary linear continuum mechanics.

ically, metamaterials owns a microscopic chiral structure, which is then suppressed on a global scale. In the example shown, chirality is not only studied at the microscopic level, but also at the macroscopic level. It represents a very important step in the study of microstructural effects in the field of metamaterials. Chirality is a primary property for the study of metamaterials, it is exploited to make microscopic structures rotate on themselves when subjected to some types of stresses [2]. The figure 1.2 shows a unit cell design that respects the chirality property. The cell exhibits quadruple rotational symmetry about the three principal axes. The figure 1.3 shows the structure obtained if several unit cells are overlaped (Fig.1.2). In figure 1.4 it is shown what is obtained if the finite element analysis is performed. Usually, a "normal" material has a well-known behavior, for example if it is stretched it becomes thinner, or if it is compressed it expands laterally. In metamaterials the

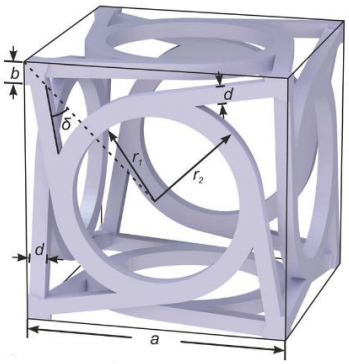

Figure 1.2.: One unit cell of the chiral tetragonal mechanical metamaterial considered.Are indicated the lattice constant a, the angle δ, the radii r_1 and r_2, and the widths b and d.

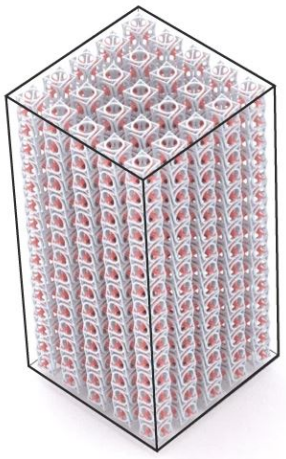

Figure 1.3.: Whole structure of the chiral tetragonal mechanical metamaterial considered.

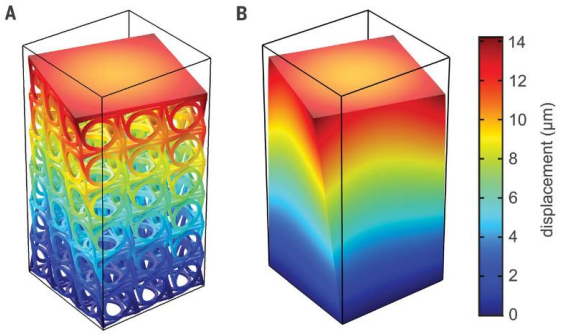

Figure 1.4.: FEM calculations. Structure and modulus of the displacement vector field on a false-color scale are overlaid. (A) Microstructure calculation for a 3×3 basis of unit cells. (B) Calculation following chiral micropolar continuum mechanics for $L = 3a$ with $a = 500\mu$m.

opposite behavior could occur, this leads to a violation of the fundamental laws of deformation [3].

1.2. Background on linear and advanced elasticity

1.2.1. History of gradient elasticity

In the first half of the 19th century, the classical theory of elasticity was almost completely formulated by Navier, Cauchy and Green. In the late 50s, it has been thought to improve the understanding of constitutive laws by introducing new material-related parameters, in turn bringing an extention of the classical theory.

Before diving into non-classical elastic theories, here it is provided a brief background of the classical theory pillars [4]. These are the following:

- Kinematic hypothesis,
- Dynamic hypothesis,
- Constitutive hypothesis.

The kinematic hypothesis does not take into account the atomic structure of matter. Consider a material body as a continuum set of structure less entities that behave like material points. It refers to the continuum hypothesis, that is, it considers the matter of a homogeneous elastic body and continuously distributed throughout its volume in such a way that the smallest element cut from the body has the same properties as the whole body. This hypothesis was formulated by Timoshenko and Goodier in 1970. The dynamic hypothesis deals with imposing constraints on the previously mentioned volume elements. The constraints impose that they are only contact interactions represented by force vectors. The latter are determined by the stress tensor, which is symmetrical. The constitutive hypothesis describes the stress tensor. From the dynamic hypothesis we know that it is symmetrical, this means that if we define the stress tensor at a point, through the constitutive hypothesis, we can determine it from the strain tensor at the same point. Now we can better understand how the classical theory of elasticity fails to better define reality, and for this reason advanced theories have been born.

1.2.2. Advanced theories

Of a particular interest, it is the theory of elasticity of the deformation gradient, formulated by Toupin and Mindlin in the 1960s. Inspired from Toupin's work developed in 1962, Mindlin began to find interest in considering solving linear elastic problems in materials with microstructures, using the earliest and highest strain gradients in potential energy expressions. In 1964 Mindlin formulated his linear elastodynamic theory. It considered the potential energy density as a function of the strains and its first gradient, and the kinetic energy density as a function of the

velocity and its gradient. Unfortunately, Mindlin's theory was very complex due to the large number of intrinsic parameters and materials involved. A few decades later, the theory of elasticity of deformation was reformulated considering only two parameters, one taking into account the microstructure and one the micro-inertia. Another advanced theory of great interest is the non-local elastic theory. It was first introduced by Kröner in 1967 and was later extensively analyzed by Eringen. The nonlocal elastic theory was introduced to deal with the fact that statistical models applied to random inhomogeneous materials can be replaced by nonlocal ones for a single solid.

Since the non-local elastic theory consisted of constitutive and integro-differential equations of motion, it found little application. Eringen, to overcome this obstacle, proposed a differential form of the non-local theory, known as the non-local differential form or stress gradient elastic theory. It presents some paradoxes, it indicates that an integral strain model is associated with a stress gradient model and similarly a strain gradient model can be associated with an integral stress model. This is accomplished by considering a nonlocality not on stresses but on the expressions of potential and kinetic energy densities.

1.2.3. Application of higher-order elastic theories

Higher-order elastic theories are widely used in many engineering applications, e.g. for the study of lattice columns made with three-dimensional chiral metamaterials, as it shown in Fig. 1.5.

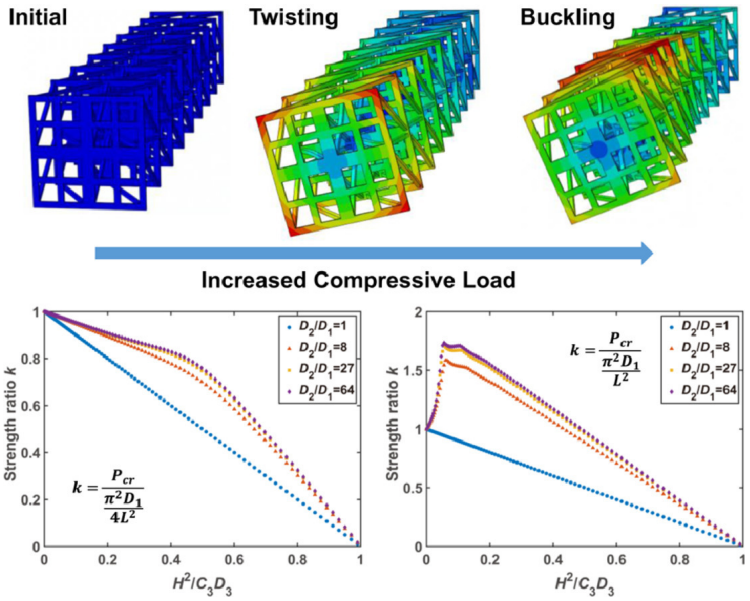

Figure 1.5.: The global buckling behavior of 3D chiral materials through FEM simulations.

Three-dimensional chiral mechanical metamaterials possess a unique torsion-compression

coupling effect [5]. The presence of chirality in the metamaterial may induce an unusual deformation behavior of the torsion-compression coupling. Although the metamaterials in question show good torsion-compression coupling performance in a single direction, most of the studies cited limit themselves to hypothesizing isotropic mechanical properties [6]. The modular unit cell design provides higher flexibility

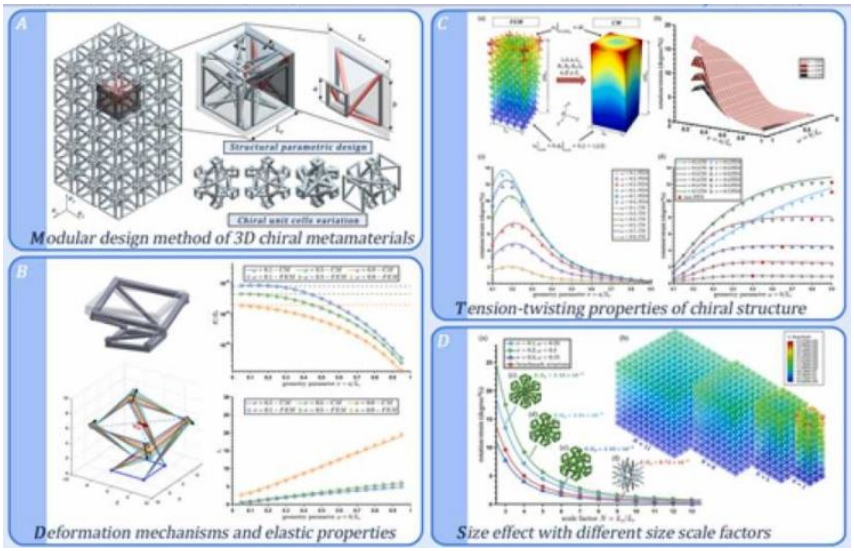

Figure 1.6.: 3D chiral metamaterial modular design with highly-tunable tension-twisting properties. (A) Modular design method of 3D chiral metamaterials. (B) Deformation mechanisms and elastic properties. (C) Tension-twisting properties of chiral structure. (D) Size effect with different size scale factors.

in tailoring chiral properties.

The extension to non-isotropic mechanical properties, always considering the non-violation of the geometry constraints, can be developed through the modular design method, as it shown in Fig. 1.6. If you divide the unit cell into several independent design cells, you greatly increase your design freedom.

1.2.4. Novel developments in the field of metamaterials

Metamaterials are a class of man-made materials that possess properties not available in nature. Metamaterials have enabled the development of new devices and their use in new applications. Metamaterials find many applications, from engineering to the medical, aerospace and automotive sectors [7]. Several examples of the application of metamaterials can be made, such as the influence of the microstructure on the dimensional effect applied in composite structures [8].

The microstructural deviation closely related to the elastic response is known as the “size effect”. Using the elasticity of the deformation gradient it is possible to study the size effect by adding different parameters that take into account the different characteristics. The size effect results in a stiffening behavior in the structure.

This effect is reported in Tab. 1.1. , where the maximum vertical displacement of a beam subjected to bending for a different thickness. In the example shown, a simple

Thickness in mm	$\bar{u}_{max}$ (round)	$\bar{u}_{max}$ (oval)
1000	7.37×10^{-3}	6.58×10^{-3}
100	7.37×10^{-3}	6.57×10^{-3}
50	7.35×10^{-3}	6.57×10^{-3}
25	7.28×10^{-3}	6.54×10^{-3}
12	7.02×10^{-3}	6.42×10^{-3}
6	6.35×10^{-3}	6.05×10^{-3}
3	4.78×10^{-3}	5.06×10^{-3}

Table 1.1.: Beam bending and size effect is normalized by the thickness.

beam subjected to bending with a length/thickness ratio of 10 is analysed. As a hypothesis, we keep the length/thickness ratio constant, lock the beam to the right and left, and use 10MPa traction in between. By varying the thickness and length two phenomena can be observed: the anisotropy leads to a stiffer beam along the horizontal axis and smaller the beam more dominant will be the effect of the second gradient.

Chapter 2.

Higher-order elasticity

2.1. Background

Microstructural effects are studied with different theories based on discrete or continuum models, according to field of application and underlying hypothesis.

This thesis focuses on the continuum description of the behavior of microstructural materials. In detail, it investigates the strain gradient elasticity theory. Strain gradient elasticity is an advanced elastic theory widely used in recent years for the description of the so-called dimensional effects. It can be classified as a non-classical theory because it considers the kinetic and potential energy densities, as well as functions of strain and velocity, also function of their respective gradients.

2.2. Strain gradient elasticity

The linear elastic behavior of materials is based on Hooke's Law

$$F = k_s \Delta u. \tag{2.1}$$

Equation (2.1) states that the force F is proportional to the change in axial displacement u according to a spring constant k_s [9]. For isotropic materials, considering the stress σ_{ij} and the strain ϵ_{ij}, the same relationship (Eq. (2.1)) can be obtained through the elastic modulus E

$$\sigma_{ij} = \frac{E}{1+\nu}\left[\epsilon_{ij} + \frac{\nu}{1-2\nu}\epsilon_{kk}\delta_{ij}\right], \tag{2.2}$$

ν denotes Poisson's ratio. The elastic constant k_s depends on the structure. The parameters E and ν are material constants independent of the geometry of the material. Obviously, the description of the deformation behavior of materials is not limited to the definition of elastic stresses and strains.

The main difference between the classical strain gradient theory and the conventional elastic theory lies in the definition of the strain energy density w, it depends on both the conventional strain and the second-order strain gradient

$$w = w(\epsilon_{ij}, \eta_{ijk}), \tag{2.3}$$

where

- $\epsilon_{ij} = \frac{1}{2}(\partial_i u_j + \partial_j u_i)$ is the strain tensor;
- $\eta_{ijk} = \partial_{ij} u_k$ is the second-order deformation gradient tensor.

∂_i is the forward gradient operator and u_i is the displacement vector. In the strain gradient elasticity the second-order strain gradient tensor is decomposed into two independent parts: the stretch gradient tensor and the rotation gradient tensor [10]. The strain tensor has six independent symmetric components and the second–order deformation gradient tensor has 18 independent components which are symmetric in the first two indexes. Considering the Mindlin's work, the Cauchy stress tensor and the double stress tensor are conjugated with the strain tensor and the second–order deformation gradient tensor

$$\sigma_{ij} = \frac{\partial w}{\partial \epsilon_{ij}}, \tag{2.4}$$

$$\tau_{ijk} = \frac{\partial w}{\partial \eta_{ijk}}. \tag{2.5}$$

The variation of total strain energy in the volume V of the body can be written as

$$\delta \int_V w dV = \int_V (\sigma_{ij} \delta \epsilon_{ij} + \tau_{ijk} \delta \eta_{ijk}) dV. \tag{2.6}$$

By applying Gauss's divergence theorem to the Eq. (2.6) the first variation of the total strain energy is obtained

$$\delta \int_V w dV = -\int_V (\partial_i \sigma_{ik} - \partial_{ij} \tau_{ijk}) \delta u_k dV + \int_{\partial V} [n_j (\sigma_{jk} - \partial_i \tau_{ijk}) \delta u_k + n_i \tau_{ijk} \partial_j \delta u_k] dS, \tag{2.7}$$

where n_i is a unit vector normal to the boundary surface ∂V of the volume V.

For linear elastic isotropic materials, the density of strain energy is

$$w = \frac{1}{2}\lambda \epsilon_{ii} \epsilon_{jj} + \mu \epsilon_{ij} \epsilon_{ij} + a_1 \eta_{ijj} \eta_{ikk} + a_2 \eta_{iik} \eta_{kjj} + a_3 \eta_{iik} \eta_{jjk} + a_4 \eta_{ijk} \eta_{ijk} + a_5 \eta_{ijk} \eta_{kji}, \tag{2.8}$$

where λ and μ are the Lamé constants, corresponding to the two invariants of deformation, and $a_n (n = 1, 2, ..., 5)$ are the five additional second–order elastic constants corresponding to the invariants of the second deformation gradient. From the Eq. (2.8), it can be seen that the strain energy density contains the cross terms for both strain and rotation gradients in the strain. Their presence prevents the direct application of the higher order equilibrium condition, for this reason the second-order deformation gradient is decomposed into symmetrical and antisymmetrical parts

$$\eta^s_{ijk} = \frac{1}{3}(\eta_{ijk} + \eta_{jki} + \eta_{kij}), \tag{2.9}$$

$$\eta^a_{ijk} = \frac{2}{3}(e_{ikl} \chi_{lj} + e_{jkl} \chi_{li}), \tag{2.10}$$

in which

- e_{ijk} is the alternating tensor;
- $\chi_{ij} = \frac{1}{2} e_{ipq} \eta_{jpq}$ is the curvature tensor.

By dividing the symmetrical part of the second-order deformation gradient we obtain a trace part $\eta^{(0)}_{ijk}$ and a traceless part $\eta^{(1)}_{ijk}$ [11]

$$\eta^{s}_{ijk} = \eta^{(0)}_{ijk} + \eta^{(1)}_{ijk}, \tag{2.11}$$

where

$$\eta^{(0)}_{ijk} = \frac{1}{5}(\delta_{ij}\eta^{s}_{mmk} + \delta_{jk}\eta^{s}_{mmi} + \delta_{ki}\eta^{s}_{mmj}), \tag{2.12}$$

$$\eta^{(1)}_{ijk} = \eta^{s}_{ijk} - \eta^{(0)}_{ijk}, \tag{2.13}$$

$$\eta^{s}_{mmk} = \frac{1}{3}(\eta_{mmk} + 2\eta_{kmm}). \tag{2.14}$$

The curvature tensor is decomposed into symmetric and antisymmetric parts

$$\chi_{ij} = \chi^{s}_{ij} + \chi^{a}_{ij}, \tag{2.15}$$

where

$$\chi^{s}_{ij} = \frac{1}{2}(\chi_{ij} + \chi_{ji}), \tag{2.16}$$

$$\chi^{a}_{ij} = \frac{1}{2}(\chi_{ij} - \chi_{ji}). \tag{2.17}$$

The number of independent components in the trace part of the second-order symmetrical strain gradient are three; in the part without a trace there are seven. While, in the symmetrical part of the curvature tensor there are five and in the antisymmetrical part there are seven. The trace part of the second-order symmetric deformation gradient is a function of the gradient expansion and the antisymmetric part of the curvature

$$\eta^{s}_{ipp} = \epsilon_{,i} + \frac{2}{3} e_{imn}\chi^{a}_{mn} = \epsilon_{,i} + \frac{2}{3} e_{imn}\chi_{mn}, \tag{2.18}$$

where $\epsilon = \epsilon_{mm}$ is the expansion stress. The second-order virtual work density written as follows

$$\delta\hat{w} = \tau^{(0)}_{ijk}\delta\eta^{(0)}_{ijk} + \tau^{(1)}_{ijk}\delta\eta^{(1)}_{ijk} + \tau^{a}_{ijk}\delta\eta^{a}_{ijk}, \tag{2.19}$$

where

$$\tau^{(0)}_{ijk} = \frac{1}{5}(\delta_{ij}\tau^{s}_{mmk} + \delta_{jk}\tau^{s}_{mmi} + \delta_{ki}\tau^{s}_{mmj}), \tag{2.20}$$

$$\tau^{(1)}_{ijk} = \tau^{s}_{ijk} - \tau^{(0)}_{ijk}, \tag{2.21}$$

are the trace and traceless parts of the symmetric part of the double stress tensor. These tensors are orthogonal to each other. The force tensors are conjugate to the strain tensor. Indicating with $\epsilon_{,i}$ the dilatation gradient, $\eta^{(1)}_{ijk}$ the deviatoric stretch

and χ_{ij} the rotational gradient, we obtain

$$\delta\hat{w} = p_i\delta\epsilon_{,i} + \tau^{(1)}_{ijk}\delta\eta^{(1)}_{ijk} + m'_{ij}\delta\chi_{ij}, \tag{2.22}$$

where:

$$p_i = \frac{3}{5}\tau^s_{mmi}, \tag{2.23}$$

$$m'_{ij} = \frac{4}{3}\tau^a_{ipq}e_{jpq} - \frac{2}{5}e_{ijk}\tau^s_{mmk}. \tag{2.24}$$

Thus, $p_i, \tau^{(1)}_{ijk}, m'_{ij}$ are respectively conjugated to $\epsilon_{,i}\,, \eta^{(1)}_{ijk}, \chi_{ij}$. The inverse forms of Eqs. (2.23), (2.24) is

$$\tau^a_{ijk} = \frac{1}{4}(e_{ikp}m_{jp} + e_{jkp}m_{ip}) - \frac{1}{6}(2\delta_{ij}p_k - \delta_{ik}p_j - \delta_{jk}p_i), \tag{2.25}$$

$$\tau^{(0)}_{ijk} = \frac{1}{3}(\delta_{ij}p_k + \delta_{jk}p_i + \delta_{ki}p_j). \tag{2.26}$$

2.2.1. Modified Strain Gradient

The total strain energy density, applying the appropriate constraints, is a function of four elements, the symmetric strain tensor, the dilatation gradient vector, the deviatoric stretch tensor and the symmetric rotation gradient tensor. While, it remains independent of the symmetric rotation gradient tensor, i.e.

$$w = w(\epsilon_{ij}, \epsilon_{,i}, \eta^{(1)}_{ijk}, \chi^s_{ij}). \tag{2.27}$$

For linear elastic center-symmetric isotropic materials, the constitutive relation becomes

$$w = \frac{1}{2}k\epsilon_{ii}\epsilon_{jj} + \mu\epsilon'_{ij}\epsilon'_{ij} + a'_0\epsilon_{mm,i}\epsilon_{nn,i} + a'_1\eta^{(1)}_{ijk}\eta^{(1)}_{ijk} + a'_2\chi^s_{ij}\chi^s_{ij}, \tag{2.28}$$

where

- $\epsilon'_{ij} = \epsilon_{ij} - \frac{1}{3}\epsilon_{mm}\delta_{ij}$ is the deviatoric strain,
- k is the bulk modul,
- μ is the shear modul,
- a'_n $(n = 0, 1, 2)$ are the additional independent material parameters associated with dilatation gradients, deviatoric stretch gradients and rotation gradients.

Referring to Eq. (2.28), we rewrite the constants as

$$a'_0 = \mu l_0^2, \tag{2.29}$$

$$a'_1 = \mu l_1^2, \tag{2.30}$$

$$a_2' = \mu l_2^2, \tag{2.31}$$

where l_n $(n = 0, 1, 2)$ are the three material length scale parameters. The constitutive equations become

$$\sigma_{ij} = k\delta_{ij}\epsilon_{mm} + 2\mu\epsilon'_{ij}, \tag{2.32}$$

$$p_i = 2\mu l_0^2 \epsilon_{mm,i}, \tag{2.33}$$

$$\tau^{(1)}_{ijk} = 2\mu l_1^2 \eta^{(1)}_{ijk}, \tag{2.34}$$

$$m^s_{ij} = 2\mu l_2^2 \chi^s_{ij}. \tag{2.35}$$

In doing so, from five independent higher-order length scale parameters, in the modified theory we find only three parameters. This reduction is very important and represents a step forward for the definition of the behavior of the deformation gradient.

2.2.2. Coupled stress

The couple stress theory is a special case of higher-order stress theory in which the effects of the dilatation gradient and the deviatoric stretch gradient are assumed to be negligible. In this case, the total internal virtual work density will become

$$\delta w = \sigma_{ij}\delta\epsilon_{ij} + m_{ij}\delta\chi_{ij}. \tag{2.36}$$

As a result you will have

$$p_i = \frac{\partial w(\epsilon_{ij}, \epsilon_{,i}, \eta^{(1)}_{ijk}, \chi_{ij})}{\partial \epsilon_{,i}} = 0, \tag{2.37}$$

$$\tau^{(1)}_{ijk} = \frac{\partial w(\epsilon_{ij}, \epsilon_{,i}, \eta^{(1)}_{ijk}, \chi_{ij})}{\partial \eta^{(1)}_{ijk}} = 0. \tag{2.38}$$

Equilibrium is satisfied if the couple stress tensor is symmetric

$$m_{ij} = m_{ji}. \tag{2.39}$$

We have that the antisymmetric part of the rotation gradient tensor is independent of the strain energy, consequently we will have

$$m^a_{ij} = \frac{\partial w(\epsilon_{ij}, \epsilon_{,i}, \eta^{(1)}_{ijk}, \chi^s_{ij}, \chi^a_{ij})}{\partial \chi^a_{ij}} = 0. \tag{2.40}$$

The constraint in Eq. (2.40) cannot be obtained simply from the definition of the strain gradient (Eqs. (2.16)-(2.17)), because the symmetric part involves the antisymmetric part of the gradient.

Chapter 3.

Solving PDEs in Python

3.1. Mathematical problem formulation

FEniCS is a research and software project created for the creation of mathematical software and methods for computational mathematical modeling [12]. FEniCS was born in 2003, the idea was to create an easy, efficient and flexible software to solve partial differential equations (PDE) with the help of finite element methods. FEniCS is composed of a series of software:

- DOLFIN: is a FEniCS component from C++ library, with a Python interface. It implements data structures such as meshes, functions, algorithms;
- FFC: is the FEniCS code generation engine;
- FIAT: it is the finite element backend of FEniCS;
- UFL: it is a FEniCS component that implements the mathematical language to allow the solution of variational problems;
- Viper: it is used for the quick visualization of meshes and finite element solutions.

In order to explain the FEniCS implementation we employ the simple Poisson equation. It is present in many scientific contexts: in the torsion of elastic rods, in water waves and above all in the Navier–Stokes equations. It reads

$$-\nabla^2 u(x) = f(x), \quad x \ in \ \Omega, \tag{3.1}$$

$$u(x) = u_D(x), \quad x \ on \ \partial\Omega. \tag{3.2}$$

Here,

- $u = u(x)$ is the unknown function;
- $f = f(x)$ is the prescribed function;
- ∇^2 is the Laplace operator;
- Ω is the spatial domain;

- $\partial\Omega$ is the spatial domain boundary.

In a two-dimensional space, at coordinates x and y, is possible to write Poisson's equation as

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x, y). \tag{3.3}$$

In this case, the unknown function u is a two variables function, defined on a two-dimensional domain.

The FEniCS solution of the Poisson equation can be found by following the steps below:

1. Identify the computational domain, the PDE, its boundary conditions and source terms;
2. Reformulate the PDE as a variational finite element problem;
3. Write a Python program that defines the computational domain, variational problem, boundary conditions, and source terms;
4. Call FEniCS to solve the limit value problem and view the results.

3.1.1. Finite element variational formulation

To work in FEniCS the PDEs must be expressed in variational form. In this section the approach used to convert PDEs into a variational problem will be explained. The procedure for converting a PDE is as follows:

1. Multiplying it by a function v is called a test function;
2. Integrate the resulting equation on the domain considered Ω; the equation that is found, denoted by u, is called a trial function;
3. Integrate by parts of terms with second-order derivatives.

We apply the procedure to the Poisson equation

$$-\int_\Omega (\nabla^2 u) v dx = \int_\Omega f v dx. \tag{3.4}$$

dx denotes the differential element for integration over the domain Ω. While, later on, we will denote by ds the differential element for integration over the boundary $\partial\Omega$. Applying integration by parts, we get

$$-\int_\Omega (\nabla^2 u) v dx = \int_\Omega \nabla u \cdot \nabla v dx - \int_{\partial\Omega} \frac{\partial u}{\partial n} v ds, \tag{3.5}$$

Where $\frac{\partial u}{\partial n}$ is the derivative of u in the outward normal direction n on the boundary $\partial\Omega$.

An important consideration of the variational problem is that the test function v must vanish in the parts of the domain boundary where the test function u is known. Still referring to the Poisson equation, this means that $v = 0$ on the entire boundary $\partial\Omega$. From Eqs. (3.4)-(3.5) we get

$$\int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v dx. \tag{3.6}$$

Furthermore, if we extend the concept, then we require Eq. (3.6) to hold for all test functions v in some test space $\hat{V}$. In doing so, a test function u defined in a trial test space V is determined. If the Eq. (3.6) is defined as the weak form or variation of the original limit problem Eqs. (3.1)- (3.2), the correct statement (strong form of the variational problem) will be

$$\int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v dx \quad \forall v \in \hat{V}. \tag{3.7}$$

Where the test and trial space are defined as

$$V = v \in H^1(\Omega) : v = u_D \ \ on \ \ \partial\Omega, \tag{3.8}$$

$$\hat{V} = v \in H^1(\Omega) : v = 0 \ \ on \ \ \partial\Omega. \tag{3.9}$$

Here, $H^1(\Omega)$ is the Sobolev space containing the functions v such that v^2 and $|\nabla v|^2$ have finite integrals over Ω. At this point we have that the defined function must be in a function space with continuous derivatives, and the Sobolev space also allows functions with discontinuous derivatives. All this guarantees the use of piecewise function spaces, therefore the possibility of joining function spaces on simple domains, such as triangles, tetrahedrons, etc.

The variational problem is a continuous problem, it defines the solution u in the space of infinite-dimensional functions V. The finite element method for Poisson's equation finds an approximate solution to the variational problem by replacing the spaces of infinite-dimensional functions V and $\hat{V}$ with discrete trial and test spaces $V_h \subset V$ and $\hat{V}_h \subset \hat{V}$. The discrete variational problem will be

$$\int_\Omega \nabla u_h \cdot \nabla v dx = \int_\Omega f v dx \quad \forall v \in \hat{V}_h \subset \hat{V}. \tag{3.10}$$

This gives us an approximate solution of Poisson's equation. In FEniCS it is possible to automatically solve discrete variational problems as in the Eq. (3.10).

Chapter 4.

Results

4.1. Implementation

The aim of the study is to implement in FEniCS advanced theories, discussed in the paragraph 1.2.2. We started from a code, in particular from the Abali code, reported in the appendix A.1.3. In the first lines of the code the geometry to be analyzed is described. The various functions are described below, described in the paragraph 3.1.1. Then the material parameters are described, in this case we consider an aluminum. The various tensors and gradients are defined and finally the force applied to the object. From line 100 onwards, the definition of the cycle and the resolution of the searched parameters begins. The last few lines are dedicated to saving the code.

4.2. Case study

The code of Abali [13] is modified in order to to make it more "autonomous" and to insert the desired geometry. Respectively, you insert a loop and define the geometry via .xml file. First of all let's insert a for loop to make the saving of data for an unknown of choice more automatic, in this case c_6. Furthermore, wanting to use a geometry of one's choice, we have to insert the following strings in the code for the definition of the geometry itself. In the code it will be necessary to insert three .xml files, the procedure is reported in the appendix A.1.2. The resulting code is as follows:

```
    """Supply code of
2
Theory and computation of higher gradient elasticity theories based on
    action principles
4
by B.E.Abali, W.H.Mueller, F. dell'Isola
6 """
__author__ = "B. Emek Abali"
8 __license__  = "GNU LGPL Version 3.0 or later"
from TTT import *
10 from ufl import *
from fenics import *
```

```
#set_log_level(ERROR)
# the chosen metric units: N, mm, s, ton
scale = 30. #beam length in mm
xlength=scale #in mm
ylength=scale/30. #in mm
zlength=scale/30. #in mm
non = 3
xml_file = "/home/bifthe/Documents/AbaliFenics/Mesh_Strange_Beam.xml"
mesh = Mesh(xml_file)
cells = MeshFunction('size_t', mesh, "/home/bifthe/Documents/
    AbaliFenics/Mesh_Strange_Beam_physical_region.xml");
facets = MeshFunction('size_t', mesh, "/home/bifthe/Documents/
    AbaliFenics/Mesh_Strange_Beam_facet_region.xml");
V = VectorFunctionSpace(mesh, 'P', 2)

minimal_cell_size = mesh.hmin()
number_of_cells = mesh.num_cells()
number_of_vertices = mesh.num_vertices()
print("Total number of cells %.0f, Minimal cell size %.2f, %.0f dof "%(
    number_of_cells, minimal_cell_size,  number_of_vertices))

VV = FunctionSpace(mesh, "CG", 1)
VV2 = FunctionSpace(mesh, "CG", 2)
print(VV.dim(), VV2.dim())


cells = MeshFunction("size_t", mesh, mesh.topology().dim(), 0)
facets = MeshFunction("size_t", mesh, mesh.topology().dim()-1, 0)
dA = Measure('ds', domain=mesh, subdomain_data=facets)
dV = Measure('dx', domain=mesh, subdomain_data=cells)
left = CompiledSubDomain('near(x[0],0) && on_boundary')
right = CompiledSubDomain('near(x[0], length) && on_boundary',length=
    xlength)
facets.set_all(0)
right.mark(facets, 1)
bc=[]
bc.append( DirichletBC(V, (0.0,0.0,0.0), left) )
shearing_force = 10./(100./scale)**2 #in N
tHat = Expression(('0.0','0.0','amp*time'),amp=shearing_force/ylength/
    zlength,time=0,degree=1)

du = TrialFunction(V)
del_u = TestFunction(V)
u  = Function(V)
u0  = Function(V)
u00  = Function(V)

for c6 in [0,1,10,100,1000,10000]:
  # material parameters of an aluminum
  rho_0 = 2.7E-9 #in ton/mm3
  nu = 0.33
```

```
E = 72E3 #in MPa
lambada = E*nu/(1.0+nu)/(1.0-2.0*nu)
mu = E/(2.0*(1.0+nu))
c1, c2 = lambada, mu
c3, c4, c5, c7 = 0., 0., 0., 0. #in N
#c3, c4, c5, c6, c7 = 0., 0., 0., 0., 0. #in N
i, j, k, l, m, n = indices(6)
delta = Identity(3)

t, dt, t_end = 0., 0.05, 0.9

C = as_tensor(c1*delta[i,j]*delta[k,l] + c2*(delta[i,k]*delta[j,l]+
  delta[i,l]*delta[j,k]) , [i,j,k,l])
D = as_tensor(c3*(delta[i,j]*delta[k,l]*delta[m,n] + delta[i,n]*delta
  [j,k]*delta[l,m] + delta[i,j]*delta[k,m]*delta[l,n] + delta[i,k]*
  delta[j,n]*delta[l,m]) + c4*delta[i,j]*delta[k,n]*delta[m,l] + c5*(
  delta[i,k]*delta[j,l]*delta[m,n] + delta[i,m]*delta[j,k]*delta[l,n]
 + delta[i,k]*delta[j,m]*delta[l,n] + delta[i,l]*delta[j,k]*delta[m,n
  ]) + c6*(delta[i,l]*delta[j,m]*delta[k,n] + delta[i,m]*delta[j,l]*
  delta[k,n]) + c7*(delta[i,l]*delta[j,n]*delta[m,k] + delta[i,m]*
  delta[j,n]*delta[l,k] + delta[i,n]*delta[j,l]*delta[k,m] + delta[i,n
  ]*delta[j,m]*delta[k,l]) , [i,j,k,l,m,n])

def stored_energy(g_u,gg_u):
  E = as_tensor(1./2.*g_u[k,i]*g_u[k,j] + 1./2.*(g_u[i,j]+g_u[j,i]) ,
   [i,j])
  g_E = as_tensor(1./2.*(gg_u[l,i,k]*g_u[l,j]+g_u[l,i]*gg_u[l,j,k])
  +1./2.*(gg_u[i,j,k]+gg_u[j,i,k]) , [i,j,k])
  return as_tensor(E[i,j]*C[i,j,k,l]*E[k,l] + g_E[i,j,k]*D[i,j,k,l,m,
  n]*g_E[l,m,n], [])

grad_u = variable(grad(u))
gradgrad_u = variable(grad(grad(u)))
w=stored_energy(grad_u,gradgrad_u)

grad_u0 = variable(grad(u0))
gradgrad_u0 = variable(grad(grad(u0)))
w0=stored_energy(grad_u0,gradgrad_u0)

grad_u00 = variable(grad(u00))
gradgrad_u00 = variable(grad(grad(u00)))
w00=stored_energy(grad_u00,gradgrad_u00)

# body force
f = Constant((0.,0.,0.)) #in N/ton
ll = as_tensor([[0.,0.,0.],[0.,0.,0.],[0.,0.,0.]])

Form = (rho_0*f[i]*del_u[i] \
- rho_0*(u-2.*u0+u00)[i]/dt/dt*del_u[i] \
- diff(w,grad_u)[i,j]*grad(del_u)[i,j] \
+ rho_0*ll[j,i]*grad(del_u)[i,j] \
```

```
  - 1./dt/dt*(diff(w,grad_u)-2.*diff(w0,grad_u0)+diff(w00,grad_u00))[i,
    j]*grad(del_u)[i,j] \
- diff(w,gradgrad_u)[i,j,k]*grad(grad(del_u))[i,j,k]  )*dV \
+ tHat[i]*del_u[i]*dA(1)

Gain = derivative(Form,u,du)

tic()
while t<t_end:
  t += dt
  tHat.time = t
  tic()

  # solve(Form == 0, u, bc, J=Gain, \
  # solver_parameters={"newton_solver":{"linear_solver": "mumps", "
  relative_tolerance": 1e-3} }, \
  # form_compiler_parameters={"cpp_optimize": True, "representation":
   "quadrature", "quadrature_degree": 2}  )
  solve(Form == 0, u, bc, J=Gain, \
  solver_parameters={"newton_solver":{"linear_solver": "mumps", "
  relative_tolerance": 1e-5} }, \
  form_compiler_parameters={"cpp_optimize": True, "representation": "
  uflacs" }  )

  print('time: ', t, 'calculated in ',toc(),' seconds. Maximum
  deflection ',max(abs(u.vector().get_local())))
  u0.assign(u)

  pwd = '/home/bifthe/Documents/Fenics/Geometry_Diff_Abali/'
  file = File(pwd+str(scale)+str(c6)+'strangebeam.pvd')
  file << (u,t)
```

4.2.1. c_6 effect

Let us consider a beam with a fixed length and vary the parameter c_6. Although the applied shear force acts on the whole section, the parameter imposes, in addition to a shear bending, also a torsion. In the Fig. 4.1 and 4.2 it is possible to see how the parameter c_6 influences both the torsion and the deflection.

In the table 4.1 shows indicative data regarding the maximum deflection as a function of the variation of c_6.

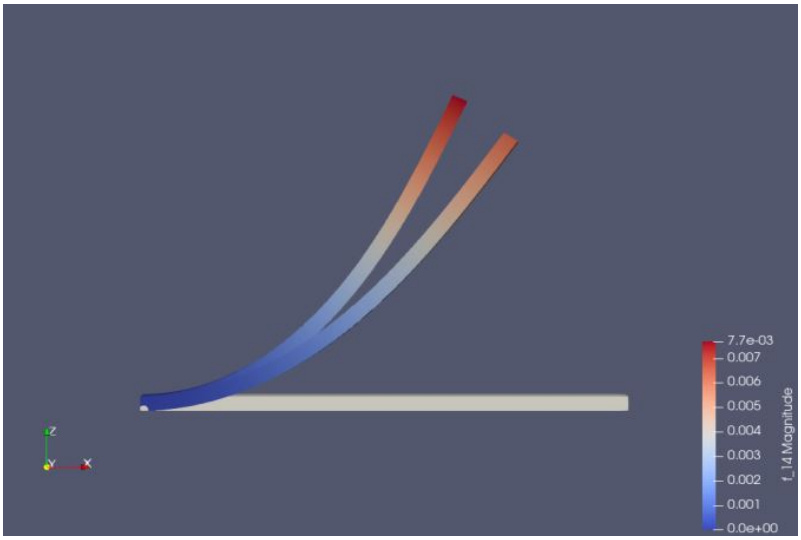

Figure 4.1.: Deflection strain of three-dimensional beam with $c_6 = 0$ and $c_6 = 10000N$, without using any scaling.

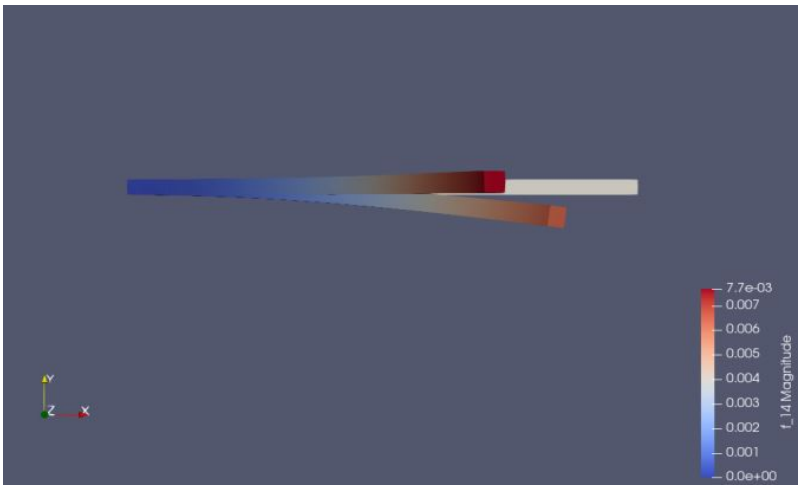

Figure 4.2.: Deflection strain from above of three-dimensional beam with $c_6 = 0N$ and $c_6 = 10000N$, without using any scaling.

c_6 in N	u_i^{max} in mm
0	4, 21286
1	4, 21285
10	4, 21274
100	4, 21164
1000	4, 20077
10000	4, 10762

Table 4.1.: Maximum deflection.

4.2.2. Loop

If you want to automate the code, thus having the ability to assign specific values to a parameter, you can modify the code by placing a loop. You may have two possibilities:

- For loop:

```
for c_6 in [1, 100, 1000]:
    print("We're on time %d" %(c_6))
    print("We're on time %d" %(c_6))
    pwd + str(c_6)
```

- While loop:

```
vector = [1,100,1000]
x=0
dx=1
y=3
while x<y:
    x+=dx
    print("current index %d" %(x)
    c_6 vector [x-1]
    print("We're on time %d" %(c_6))
```

In the case study of this thesis a for loop was used.

4.2.3. Other effects

Another noteworthy effect is the change in length. Keeping the parameter $c_6 = 100N$ fixed and making the length vary, in particular we consider a beam $1mm$, $1m$ and $100m$ long respectively shown in the Fig. 4.3, 4.4 and 4.5, it can be seen that the qualitative deformation varies with the length. In particular, shorter beams show more stiffening, this effect is called size effect.

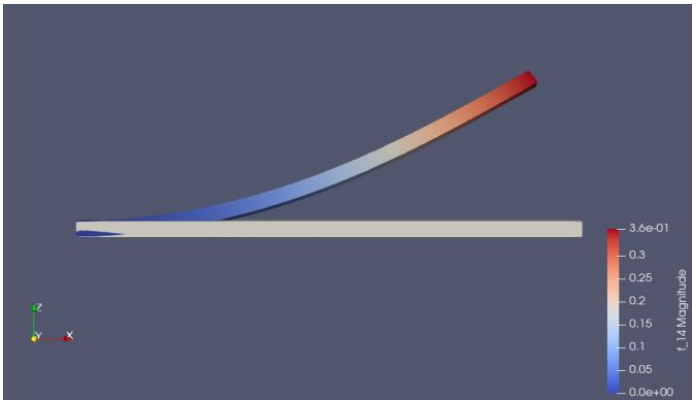

Figure 4.3.: Bending strain of the three-dimensional beam with $c_6 = 100N$ and $length = 1mm$, without any scaling.

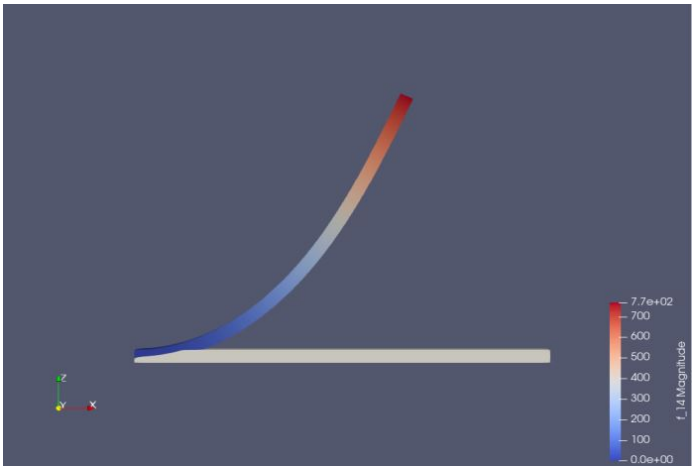

Figure 4.4.: Bending strain of the three-dimensional beam with $c_6 = 100N$ and $length = 1m$, without any scaling.

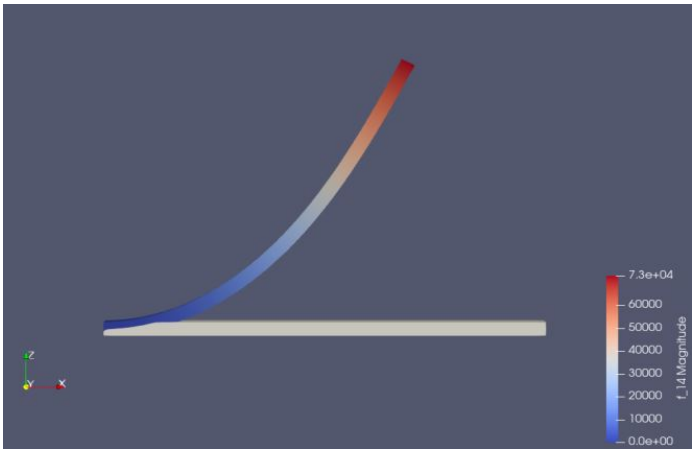

Figure 4.5.: Bending strain of the three-dimensional beam with $c_6 = 100N$ and $length = 100m$, without any scaling.

4.2.4. Different geometry

The procedure for modifying the code to vary the geometry has already been seen in the paragraph 4.2. In this paragraph we will focus on understanding the effect on the buttonholes when the various parameters are changed one at a time. The code used is the following:

```
    """Supply code of

Theory and computation of higher gradient elasticity theories based on
    action principles

by B.E.Abali, W.H.Mueller, F. dell'Isola
"""
__author__ = "B. Emek Abali"
__license__  = "GNU LGPL Version 3.0 or later"
from TTT import *
from ufl import *
from fenics import *
#set_log_level(ERROR)
# the chosen metric units: N, mm, s, ton
scale = 30 #beam length in mm   0.01
xlength=scale #in mm
ylength=scale/30. #in mm
zlength=scale/30. #in mm

xml_file = "Mesh_G.xml"
mesh = Mesh(xml_file)
cells = MeshFunction('size_t', mesh, "Mesh_G_physical_region.xml");
facets = MeshFunction('size_t', mesh, "Mesh_G_facet_region.xml");


#import basix
#element = basix.create_element("Nedelec 1st kind H(curl)", "
    tetrahedron", 2)
#print(element.base_transformations)

V = VectorFunctionSpace(mesh, 'P', 2)

minimal_cell_size = mesh.hmin()
number_of_cells = mesh.num_cells()
number_of_vertices = mesh.num_vertices()
print("Total number of cells %.0f, Minimal cell size %.2f, %.0f dof "%(
    number_of_cells, minimal_cell_size,  number_of_vertices))


VV = VectorFunctionSpace(mesh, "CG", 1)
VV2 = VectorFunctionSpace(mesh, "CG", 2)
print(VV.dim(), VV2.dim())

dA = Measure('ds', domain=mesh, subdomain_data=facets, metadata={'
```

```
    quadrature_degree':3})
dV = Measure('dx', domain=mesh, subdomain_data=cells, metadata={'
    quadrature_degree':3})
left = CompiledSubDomain('near(x[0],0) && on_boundary')
right = CompiledSubDomain('near(x[0], length) && on_boundary',length=
    xlength)


facets.set_all(0)
right.mark(facets, 1)
bc=[]
bc.append( DirichletBC(V, (0.0, 0.0, 0.0), left) )


shearing_force =  10./(100./scale)**2 #in N
tHat = Expression(('0.0','0.0','amp*time'),amp=shearing_force/ylength/
    zlength,time=0,degree=1)

du = TrialFunction(V)
del_u = TestFunction(V)
u  = Function(V)
u0  = Function(V)
u00  = Function(V)


# material parameters of an aluminum
rho_0 = 2.7E-9 #in ton/mm3
nu = 0.33
E = 72E3 #in MPa
lambada = E*nu/(1.0+nu)/(1.0-2.0*nu)
mu = E/(2.0*(1.0+nu))
c1, c2 = lambada, mu
c3, c4, c5, c6, c7 = 0., 0., 0., 0.0, 0. #in N
#c3, c4, c5, c6, c7 = 0., 0., 0., 0., 0. #in N
i, j, k, l, m, n = indices(6)
delta = Identity(3)

t, dt, t_end = 0., 0.025, 0.9

C = as_tensor(c1*delta[i,j]*delta[k,l] + c2*(delta[i,k]*delta[j,l]+
    delta[i,l]*delta[j,k]) , [i,j,k,l])
D = as_tensor(c3*(delta[i,j]*delta[k,l]*delta[m,n] + delta[i,n]*delta[j
    ,k]*delta[l,m] + delta[i,j]*delta[k,m]*delta[l,n] + delta[i,k]*delta
    [j,n]*delta[l,m]) + c4*delta[i,j]*delta[k,n]*delta[m,l] + c5*(delta[
    i,k]*delta[j,l]*delta[m,n] + delta[i,m]*delta[j,k]*delta[l,n] +
    delta[i,k]*delta[j,m]*delta[l,n] + delta[i,l]*delta[j,k]*delta[m,n])
     + c6*(delta[i,l]*delta[j,m]*delta[k,n] + delta[i,m]*delta[j,l]*
    delta[k,n]) + c7*(delta[i,l]*delta[j,n]*delta[m,k] + delta[i,m]*
    delta[j,n]*delta[l,k] + delta[i,n]*delta[j,l]*delta[k,m] + delta[i,n
    ]*delta[j,m]*delta[k,l]) , [i,j,k,l,m,n])
```

```
80 def stored_energy(g_u,gg_u):
       E = as_tensor(1./2.*g_u[k,i]*g_u[k,j] + 1./2.*(g_u[i,j]+g_u[j,i]) ,
        [i,j])
82     g_E = as_tensor(1./2.*(gg_u[l,i,k]*g_u[l,j]+g_u[l,i]*gg_u[l,j,k])
       +1./2.*(gg_u[i,j,k]+gg_u[j,i,k]) , [i,j,k])
       return as_tensor(E[i,j]*C[i,j,k,l]*E[k,l] + g_E[i,j,k]*D[i,j,k,l,m,
       n]*g_E[l,m,n], [])
84
   grad_u = variable(grad(u))
86 gradgrad_u = variable(grad(grad(u)))
   w=stored_energy(grad_u,gradgrad_u)
88
   grad_u0 = variable(grad(u0))
90 gradgrad_u0 = variable(grad(grad(u0)))
   w0=stored_energy(grad_u0,gradgrad_u0)
92
   grad_u00 = variable(grad(u00))
94 gradgrad_u00 = variable(grad(grad(u00)))
   w00=stored_energy(grad_u00,gradgrad_u00)
96
   # body force
98 f = Constant((0.,0.,0.)) #in N/ton
   ll = as_tensor([[0.,0.,0.],[0.,0.,0.],[0.,0.,0.]])
100
   Form = (rho_0*f[i]*del_u[i] \
102 - rho_0*(u-2.*u0+u00)[i]/dt/dt*del_u[i] \
   - diff(w,grad_u)[i,j]*grad(del_u)[i,j] \
104 + rho_0*ll[j,i]*grad(del_u)[i,j] \
   - 1./dt/dt*(diff(w,grad_u)-2.*diff(w0,grad_u0)+diff(w00,grad_u00))[i,j
       ]*grad(del_u)[i,j] \
106 - diff(w,gradgrad_u)[i,j,k]*grad(grad(del_u))[i,j,k]  )*dV \
   + tHat[i]*del_u[i]*dA(1)
108
   Gain = derivative(Form,u,du)
110
   #Setting solver parameters
112 parameters["form_compiler"]["representation"] = "uflacs"
   parameters["form_compiler"]["cpp_optimize"] = True
114 parameters["form_compiler"]["optimize"] = True
   #parameters["allow_extrapolation"] = True
116 parameters["mesh_partitioner"] = "SCOTCH"
   parameters["ghost_mode"] = 'shared_facet' #'shared_vertex'  |  '
       shared_facet'
118
   krylov_params = parameters["krylov_solver"]
120 krylov_params["relative_tolerance"] = 1E-7
   krylov_params["absolute_tolerance"] = 1E-8
122 #krylov_params["divergence_limit"] = 1E-8
   krylov_params["maximum_iterations"] = 150000
124 krylov_params["nonzero_initial_guess"] = True
   krylov_params["monitor_convergence"] = False
```

```
krylov_params["error_on_nonconvergence"] = True
krylov_params["report"] = True

#convergence: incremental, residual
solver_parameters = {"nonlinear_solver": "newton", "symmetric": False,
    "newton_solver":{"linear_solver": "superlu",
                            "convergence_criterion": "incremental",
                            "relative_tolerance": 1E-7,
                            "absolute_tolerance": 1E-7,
                            "krylov_solver": krylov_params,
                            "relaxation_parameter":1.0,
                            "maximum_iterations":100,
                            "error_on_nonconvergence": True
                        }
    }


'''


tic()
while t<t_end:
    t += dt
    tHat.time = t
    tic()
    #solve(Form == 0, u, bc, J=Gain, \
    #solver_parameters={"newton_solver":{"linear_solver": "mumps", "
    relative_tolerance": 1e-3} }, \
    #form_compiler_parameters={"cpp_optimize": True, "representation":
    "quadrature", "quadrature_degree": 2}  )
    #solve(Form == 0, u, bc, J=Gain, \
    #solver_parameters={"newton_solver":{"linear_solver": "mumps", "
    relative_tolerance": 1e-3} }, \
    #form_compiler_parameters={"cpp_optimize": True, "representation":
    "uflacs" }  )
    problem = NonlinearVariationalProblem(Form, u , bc , Gain)
    solver  = NonlinearVariationalSolver(problem)
    prm = solver.parameters
    prm['newton_solver']['absolute_tolerance'] = 1E-8
    prm['newton_solver']['relative_tolerance'] = 1E-5
    prm['newton_solver']['maximum_iterations'] = 100
prm['newton_solver']['relaxation_parameter'] = 1.0


    solver.solve()
 '''
tic()


problem = NonlinearVariationalProblem(Form, u, bcs=bc, J=Gain)
solver  = NonlinearVariationalSolver(problem)
```

```
solver.parameters.update(solver_parameters)
174
176
start_time = time.time()
178 while t<t_end:
    t += dt
180     tHat.time = t
    solver.solve()
182     print('time: ', t, 'calculated in ',toc(),' seconds. Maximum
    deflection ',max(abs(u.vector().get_local())))
    u0.assign(u)
184     #u00.assign(u0)

186     pwd = '/home/bifthe/Documents/Fenics/AbaliResults/'
    file = File(pwd+str(scale)+'_mm_c6_1000_displacementM_GGG)scale.pvd
    ')
188     file << (u,t)
```

By varying the parameter c_6, respectively with $c_6 = 10N$, $c_6 = 100N$ and $c_6 = 1000N$ as shown in the Fig. 4.6, 4.7 and 4.8 we obtain that the more parameter c_6 increases, the more the buttonhole closest to the joint will deflect and twist on itself, up to a certain limit. From fig. 4.8 it can be seen that for increasingly higher values of c_6 the beam will have less deflection and torsion, until it flattens out completely for very high values. The comparision in shown in Fig. 4.9 and 4.10.

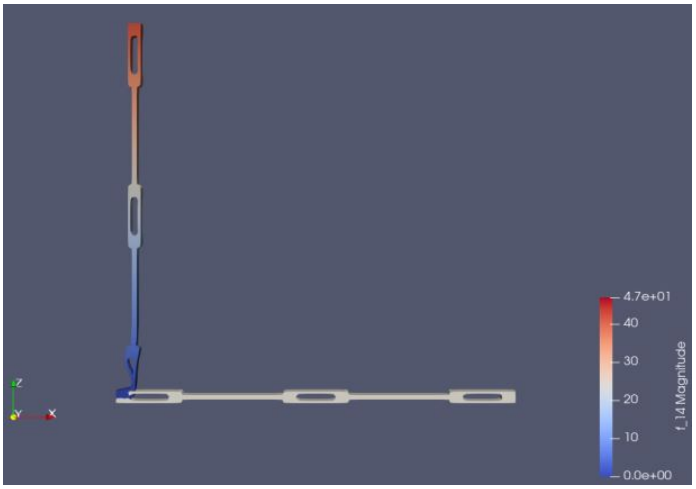

Figure 4.6.: Bending strain of the three-dimensional beam with $c_6 = 10N$.

By varying the parameter c_7, respectively with $c_7 = 10N$, $c_7 = 50N$ and $c_7 = 100N$ as shown in the Fig. 4.11, 4.12 and 4.13 we obtain that the following parameter influences the deflection of the beam, the more it increases and the lower the deflection will be, therefore there will be greater stiffening. The comparision in shown in Fig. 4.14. In the figure 4.15 we have the comparison of the effects of

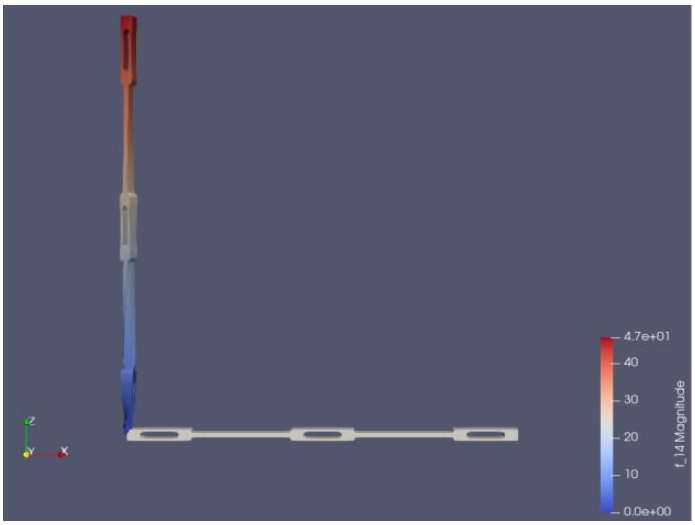

Figure 4.7.: Bending strain of the three-dimensional beam with $c_6 = 100N$.

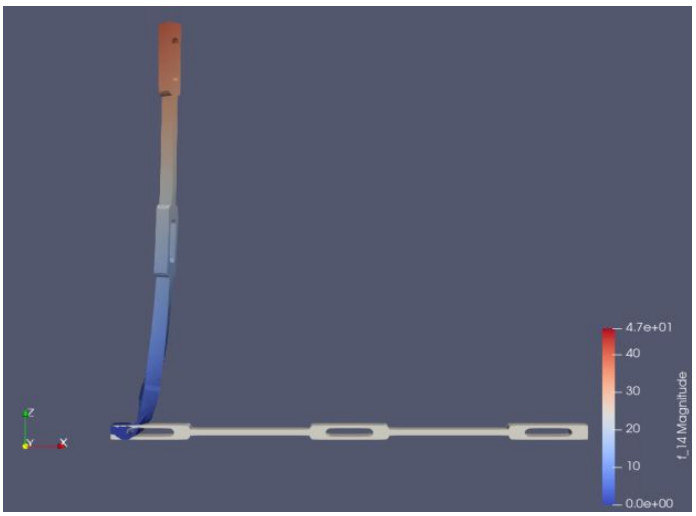

Figure 4.8.: Bending strain of the three-dimensional beam with $c_6 = 1000N$.

the parameters with a single value, therefore $c_3 = c_4 = c_5 = c_6 = c_7 = 50N$ to see the difference between them. The parameter c_3 influences the deflection very little, the parameter c_4 influences both the deflection and the elongation, the parameter c_5 causes an almost imperceptible effect, it acts both on the elongation, deflection and both on the torsion, the parameter c_6 acts on the torsion and deflection and finally the parameter c_7 acts on the deflection and shortening.

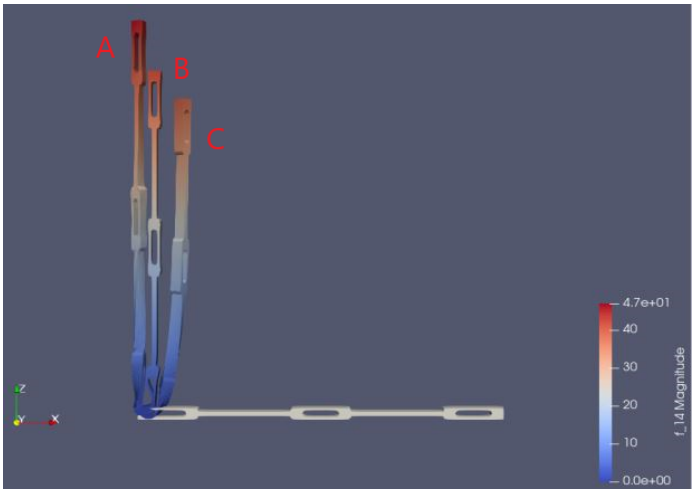

Figure 4.9.: Comparison of three-dimensional beam with (A) $c_6 = 100N$, (B) $c_6 = 10N$ and (C) $c_6 = 1000N$.

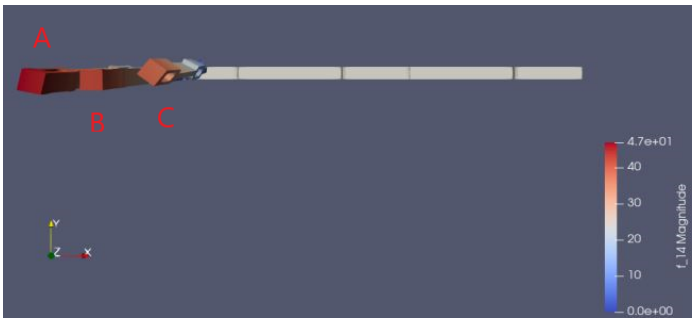

Figure 4.10.: Comparison from above of three-dimensional beam with (A) $c_6 = 100N$, (B) $c_6 = 10N$ and (C) $c_6 = 1000N$.

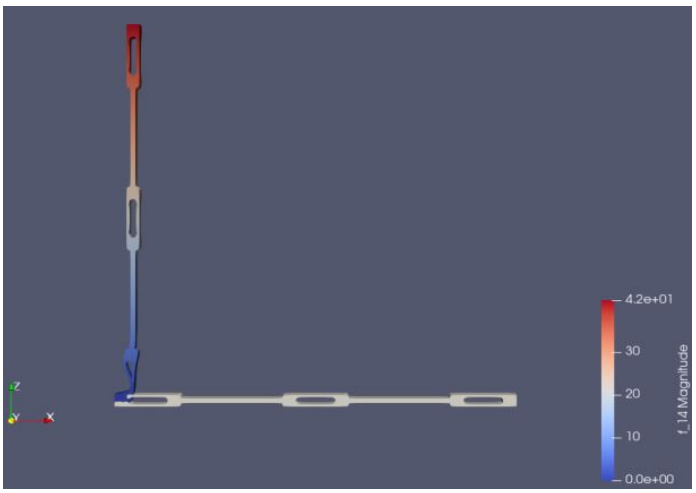

Figure 4.11.: Bending strain of the three-dimensional beam with $c_7 = 10N$.

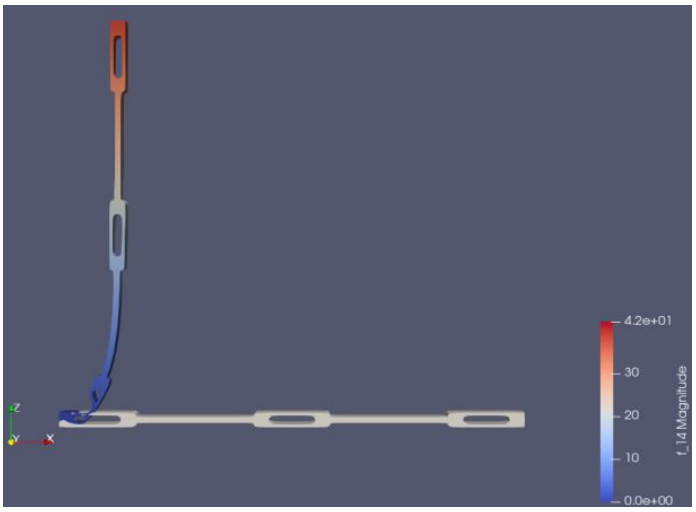

Figure 4.12.: Bending strain of the three-dimensional beam with $c_7 = 50N$.

Figure 4.13.: Bending strain of the three-dimensional beam with $c_7 = 100N$.

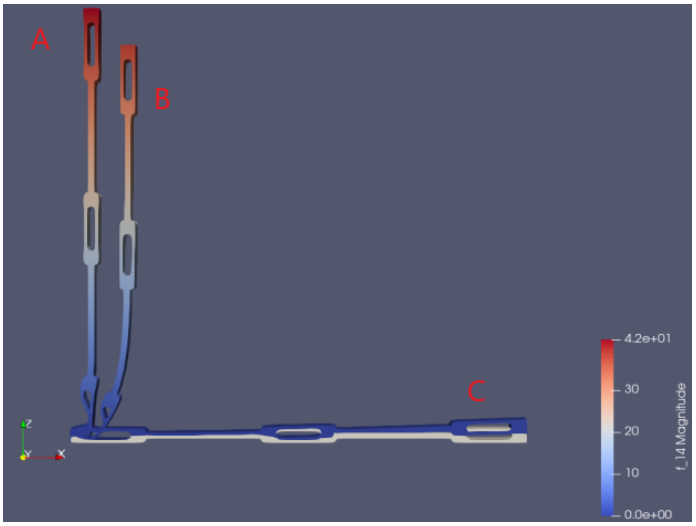

Figure 4.14.: Comparison of three-dimensional beam with (A) $c_7 = 10N$, (B) $c_7 = 50N$ and (C) $c_7 = 100N$.

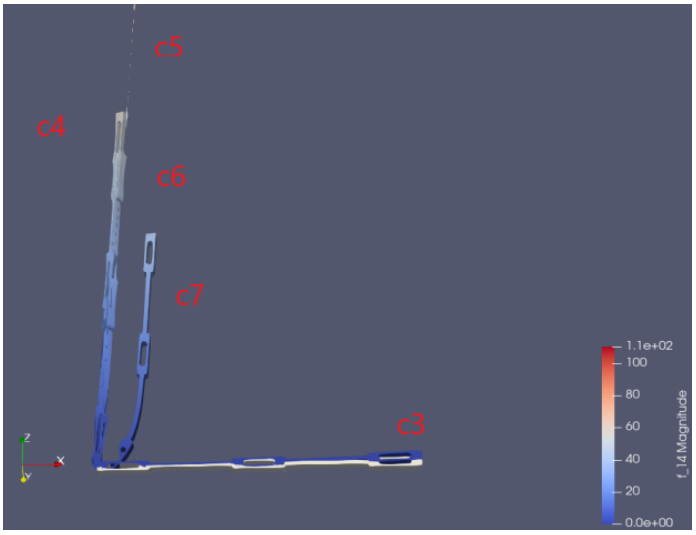

Figure 4.15.: Comparison of three-dimensional beam with a single value of $50N$.

Chapter 5.

Conclusions

5.1. Conclusions

In the simulations carried out it was noted how the variation of a parameter can influence the behavior of the entire structure. Furthermore, the variation of the length also influences the deflection. Particular attention should be paid to small beams, because during deflection they have greater stiffeners. With the studied geometry it is possible to notice how the variation of each parameter influences the structure. The c_3 parameter gives the structure a lot of stiffening. The c_4 parameter gives the structure deflection and elongation. The c_5 parameter confers torsion, deflection but above all elongation. The parameter c_6 influences the rotation of the structure, the more the parameter increases the more the maximum deflection of the beam will decrease. The parameter c_7 influences the deflection and shortening of the structure.

Appendix A.

Appendix

A.1. Methods

A.1.1. SALOME

The steps for creating a mesh are as follows:

1. First, select the "Geometry" module in the command bar, shown in Fig. A.1;

Figure A.1.: Command bar SALOME

2. Through appropriate commands you can start drawing the geometry or decide to import the geometry using BREP, STEP, IGES, STL, XAO files;

3. Once the geometry has been created/imported, select the "Mesh" module in the command bar, shown in Fig. A.1;

4. From the toolbar select “Create Mesh”, shown in Fig. A.2;

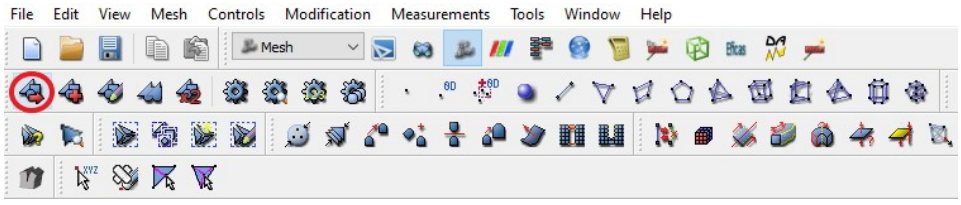

Figure A.2.: Create Mesh button SALOME

5. Select the geometry;

6. Then in "Algorithm" select "Free volumes - NETGEN 1D-2D-3D", shown in Fig. A.3;

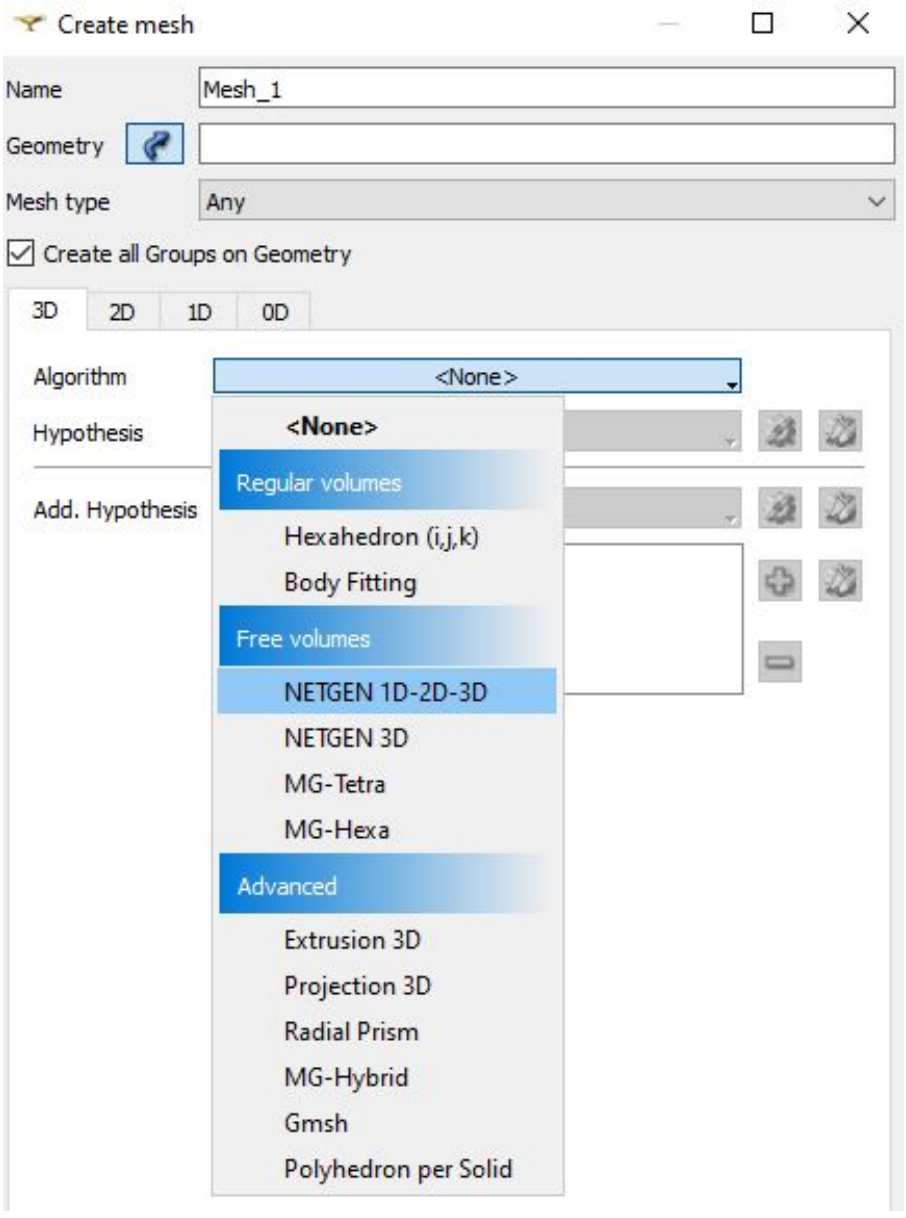

Figure A.3.: Settings mesh SALOME

7. In Hypothesis, select "NETGEN 3D Parameters" from the right icon, shown in Fig. A.4

8. Now all that remains is to right-click on Mesh – Compute Mesh.

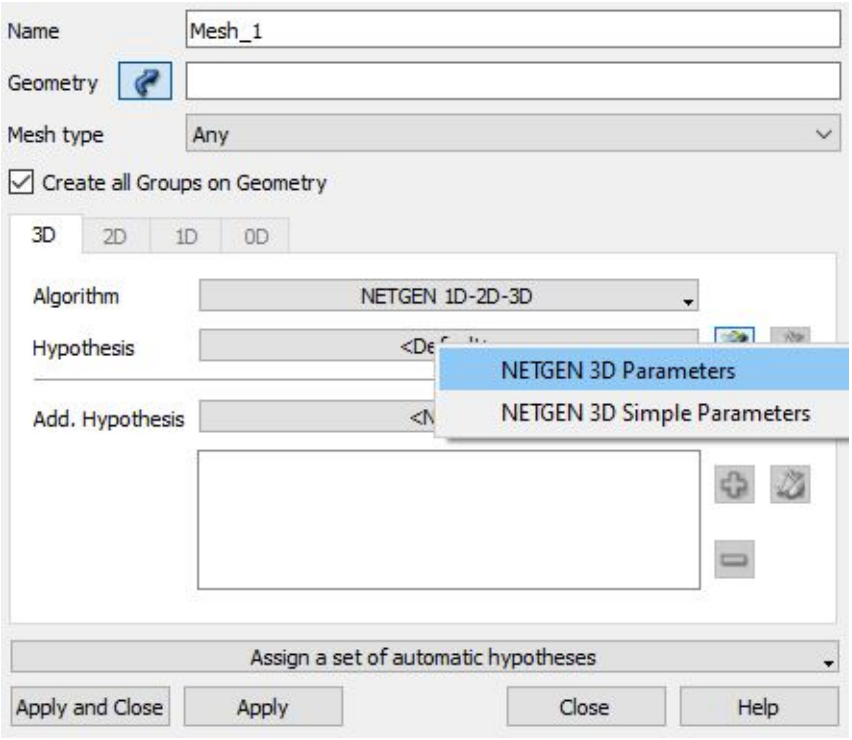

Figure A.4.: Settings parameters mesh SALOME

A.1.2. GMESH

From the Qterminal, call "./med/2xml NameFile.med" to convert the file from med to xml.

```
#!/bin/bash
fil=$1
strlen=${#filename}
filename=${fil:0:strlen-4}
gmsh $1 -3 -v 0 $fil -format msh2
dolfin-convert "$filename.msh" "$filename.xml"
```

A.1.3. FEniCS

```
"""Supply code of

Theory and computation of higher gradient elasticity theories based on
    action principles

by B.E. Abali, W.H. Mueller, F. dell'Isola
"""
__author__ = "B. Emek Abali"
__license__ = "GNU LGPL Version 3.0 or later"
from TTT import *
```

```
from ufl import *
from fenics import *
#set_log_level(ERROR)
# the chosen metric units: N, mm, s, ton
scale = 0.01 #beam length in mm
xlength=scale #in mm
ylength=scale/30. #in mm
zlength=scale/30. #in mm
non = 3
mesh = BoxMesh(Point(0, -ylength/2., -zlength/2.), Point(xlength,
    ylength/2.,zlength/2.), 30*non,1*non,1*non)
V = VectorFunctionSpace(mesh, 'P', 2)

minimal_cell_size = mesh.hmin()
number_of_cells = mesh.num_cells()
number_of_vertices = mesh.num_vertices()
print("Total number of cells %.0f, Minimal cell size %.2f, %.0f dof "%(
    number_of_cells, minimal_cell_size,  number_of_vertices))

VV = FunctionSpace(mesh, "CG", 1)
VV2 = FunctionSpace(mesh, "CG", 2)
print(VV.dim(), VV2.dim())


cells = MeshFunction("size_t", mesh, mesh.topology().dim(), 0)
facets = MeshFunction("size_t", mesh, mesh.topology().dim()-1, 0)
dA = Measure('ds', domain=mesh, subdomain_data=facets)
dV = Measure('dx', domain=mesh, subdomain_data=cells)
left = CompiledSubDomain('near(x[0],0) && on_boundary')
right = CompiledSubDomain('near(x[0], length) && on_boundary',length=
    xlength)
facets.set_all(0)
right.mark(facets, 1)
bc=[]
bc.append( DirichletBC(V, (0.0,0.0,0.0), left) )
shearing_force = 10./(100./scale)**2 #in N
tHat = Expression(('0.0','0.0','amp*time'),amp=shearing_force/ylength/
    zlength,time=0,degree=1)

du = TrialFunction(V)
del_u = TestFunction(V)
u  = Function(V)
u0  = Function(V)
u00  = Function(V)


# material parameters of an aluminum
rho_0 = 2.7E-9 #in ton/mm3
nu = 0.33
E = 72E3 #in MPa
lambada = E*nu/(1.0+nu)/(1.0-2.0*nu)
```

```
mu = E/(2.0*(1.0+nu))
c1, c2 = lambada, mu
c3, c4, c5, c6, c7 = 0., 0., 0., 0.1, 0. #in N
#c3, c4, c5, c6, c7 = 0., 0., 0., 0., 0. #in N
i, j, k, l, m, n = indices(6)
delta = Identity(3)

t, dt, t_end = 0., 0.05, 0.9

C = as_tensor(c1*delta[i,j]*delta[k,l] + c2*(delta[i,k]*delta[j,l]+
    delta[i,l]*delta[j,k]) , [i,j,k,l])
D = as_tensor(c3*(delta[i,j]*delta[k,l]*delta[m,n] + delta[i,n]*delta[j
    ,k]*delta[l,m] + delta[i,j]*delta[k,m]*delta[l,n] + delta[i,k]*delta
    [j,n]*delta[l,m]) + c4*delta[i,j]*delta[k,n]*delta[m,l] + c5*(delta[
    i,k]*delta[j,l]*delta[m,n] + delta[i,m]*delta[j,k]*delta[l,n] +
    delta[i,k]*delta[j,m]*delta[l,n] + delta[i,l]*delta[j,k]*delta[m,n])
     + c6*(delta[i,l]*delta[j,m]*delta[k,n] + delta[i,m]*delta[j,l]*
    delta[k,n]) + c7*(delta[i,l]*delta[j,n]*delta[m,k] + delta[i,m]*
    delta[j,n]*delta[l,k] + delta[i,n]*delta[j,l]*delta[k,m] + delta[i,n
    ]*delta[j,m]*delta[k,l]) , [i,j,k,l,m,n])

def stored_energy(g_u,gg_u):
  E = as_tensor(1./2.*g_u[k,i]*g_u[k,j] + 1./2.*(g_u[i,j]+g_u[j,i]) , [
    i,j])
  g_E = as_tensor(1./2.*(gg_u[l,i,k]*g_u[l,j]+g_u[l,i]*gg_u[l,j,k])
    +1./2.*(gg_u[i,j,k]+gg_u[j,i,k]) , [i,j,k])
  return as_tensor(E[i,j]*C[i,j,k,l]*E[k,l] + g_E[i,j,k]*D[i,j,k,l,m,n
    ]*g_E[l,m,n], [])

grad_u = variable(grad(u))
gradgrad_u = variable(grad(grad(u)))
w=stored_energy(grad_u,gradgrad_u)

grad_u0 = variable(grad(u0))
gradgrad_u0 = variable(grad(grad(u0)))
w0=stored_energy(grad_u0,gradgrad_u0)

grad_u00 = variable(grad(u00))
gradgrad_u00 = variable(grad(grad(u00)))
w00=stored_energy(grad_u00,gradgrad_u00)

# body force
f = Constant((0.,0.,0.)) #in N/ton
ll = as_tensor([[0.,0.,0.],[0.,0.,0.],[0.,0.,0.]])

Form = (rho_0*f[i]*del_u[i] \
- rho_0*(u-2.*u0+u00)[i]/dt/dt*del_u[i] \
- diff(w,grad_u)[i,j]*grad(del_u)[i,j] \
+ rho_0*ll[j,i]*grad(del_u)[i,j] \
- 1./dt/dt*(diff(w,grad_u)-2.*diff(w0,grad_u0)+diff(w00,grad_u00))[i,j
    ]*grad(del_u)[i,j] \
```

```
  - diff(w,gradgrad_u)[i,j,k]*grad(grad(del_u))[i,j,k]   )*dV \
96 + tHat[i]*del_u[i]*dA(1)

98 Gain = derivative(Form,u,du)

100 tic()
    while t<t_end:
102   t += dt
      tHat.time = t
104   tic()

106   # solve(Form == 0, u, bc, J=Gain, \
      # solver_parameters={"newton_solver":{"linear_solver": "mumps", "
        relative_tolerance": 1e-3} }, \
108   # form_compiler_parameters={"cpp_optimize": True, "representation": "
        quadrature", "quadrature_degree": 2}  )
      solve(Form == 0, u, bc, J=Gain, \
110   solver_parameters={"newton_solver":{"linear_solver": "mumps", "
        relative_tolerance": 1e-5} }, \
      form_compiler_parameters={"cpp_optimize": True, "representation": "
        uflacs" }  )
112
      print('time: ', t, 'calculated in ',toc(),' seconds. Maximum
        deflection ',max(abs(u.vector().get_local())))
114   u0.assign(u)

116   pwd = '/home/bifthe/Documents/AbaliFenics/results/'
      file = File(pwd+str(scale)+'_mm_c6_1000_displacementREFelementT.pvd')
118   file << (u,t)
```

A.1.4. Mathematica

```
      s={u[x,y,z],v[x,y,z],z[x,y,z]};
2     index[in_]:=Which[
      in===x,1,
4     in===y,2,
      in===z,3]
6     kd[i_,j_]:=If[
      i===j,1,0]
8     lc[i_,j_,k_]:=Which[
      {i,j,k}==={x,y,z},+1,
10    {i,j,k}==={y,z,x},+1,
      {i,j,k}==={z,x,y},+1,
12    {i,j,k}==={z,y,x},-1,
      {i,j,k}==={y,x,z},-1,
14    {i,j,k}==={x,z,y},-1,True,0]
      \[Epsilon][i_,j_]:=1/2 (D[s[[index[i]]],j]+D[s[[index[j]]],i])
16    fun1=Table[\[Epsilon][a,b],{a,{x,y,z}},{b,{x,y,z}}]
      fun1//MatrixForm
```

```
18  \[Eta][i_,j_,k_]:=D[s[[index[k]]],i,j]
    fun2=Table[\[Eta][c,d,e],{c,{x,y,z}},{d,{x,y,z}},{e,{x,y,z}}]
20  fun2//MatrixForm
    \[Gamma][i_]:=D[Tr[fun1],i]
22  fun3=Table[\[Gamma][f],{f,{x,y,z}}]
    fun3//MatrixForm
24  Overscript[\[Eta], (1)][i_,j_,k_]:=1/3 (D[\[Epsilon][j,k],i]+D[\[
    Epsilon][k,i],j]+D[\[Epsilon][i,j],k])-1/15 (kd[i,j](D[Tr[fun1],k
    ]+2(D[Sum[\[Epsilon][m,k],{m,{x,y,z}}],m])))-1/15 ((kd[j,k](D[Tr[
    fun1],i]+2(D[Sum[\[Epsilon][m,i],{m,{x,y,z}}],m])))+(kd[k,i](D[Tr[
    fun1],j]+2(D[Sum[\[Epsilon][m,j],{m,{x,y,z}}],m]))))
    fun4=Table[Overscript[\[Eta], (1)][g,h,l],{g,{x,y,z}},{h,{x,y,z}},{
    l,{x,y,z}}]
26  fun4//MatrixForm
    Overscript[\[Chi], s][i_,j_]:=1/2 ((Sum[lc[i,p,q]D[\[Epsilon][j,p],
    q],{p,{x,y,z}},{q,{x,y,z}}])+(Sum[lc[j,p,q]D[\[Epsilon][i,p],q],{p,{
    x,y,z}},{q,{x,y,z}}]))
28  fun5=Table[Overscript[\[Chi], s][n,o],{n,{x,y,z}},{o,{x,y,z}}]
    fun5//MatrixForm
```

Bibliography

[1] H. Askes and A. V. Metrikine. One-dimensional dynamically consistent gradient elasticity models derived from a discrete microstructure: Part 2: Static and dynamic response. *European Journal of Mechanics - A/Solids*, 21(4):573–588, 2002.

[2] T. Frenzel. Large characteristic lengths in 3d chiral elastic metamaterials, 2020.

[3] T. Frenzel, M. Kadic, and M. Wegener. Three-dimensional mechanical metamaterials with a twist. *Science*, 358(6366):1072–1074, 2017.

[4] T. Gortsas, D. Aggelis, and D. Polyzos. The strain gradient elasticity via nonlocal considerations. *International Journal of Solids and Structures*, 269:112177, 2023.

[5] G. Lin, J. Li, P. Chen, W. Sun, S. A. Chizhik, A. A. Makhaniok, G. B. Melnikova, and T. A. Kuznetsova. Buckling of lattice columns made from three-dimensional chiral mechanical metamaterials. *International Journal of Mechanical Sciences*, 194:106208, 2021.

[6] W. Xu, Z. Liu, L. Wang, and P. Zhu. 3d chiral metamaterial modular design with highly-tunable tension-twisting properties. *Materials Today Communications*, 30:103006, 2022.

[7] R. Kumar, M. Kumar, J. S. Chohan, and S. Kumar. Overview on metamaterial: History, types and applications. *Materials Today: Proceedings*, 56:3016–3024, 2022. 3rd International Conference on Contemporary Advances in Mechanical Engineering.

[8] B. E. Abali, B. Vazic, and P. Newell. Influence of microstructure on size effect for metamaterials applied in composite structures. *Mechanics Research Communications*, 122:103877, 2022.

[9] R. Hooke. *Lectures "de Potentia Restitutiva", Or of Spring, Explaining the Power of Springing Bodies, to which are Added Some Collections ... by Robert Hooke ...* J. Martyn, 1678.

[10] D. Lam, F. Yang, A. Chong, J. Wang, and P. Tong. Experiments and theory in strain gradient elasticity. *Journal of the Mechanics and Physics of Solids*, 51(8):1477–1508, 2003.

[11] C. Liebold and W. Mueller. Applications of strain gradient theories to the size effect in submicrostructures incl. experimental analysis of elastic material parameters. *Bull. TICMI*, 19:45–55, 01 2015.

[12] H. Langtangen and A. Logg. *Solving PDEs in Python*. 01 2016.

[13] B. Abali, W. Mueller, and F. dell'isola. Theory and computation of higher gradient elasticity theories based on action principles. *Archive of Applied Mechanics*, 87:1–16, 09 2017.