Master's thesis

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

Heidi Igland Jacobsen
Maren Eriksen Eia

2D Triangular Finite Elements based on Assumed Natural Coordinate Strains and the Seth-Hill Family of Finite Strains

Master's thesis in Mechanical Engineering
Supervisor: Bjørn Haugen
June 2021

NTNU
Norwegian University of Science and Technology

Heidi Igland Jacobsen
Maren Eriksen Eia

2D Triangular Finite Elements based on Assumed Natural Coordinate Strains and the Seth-Hill Family of Finite Strains

Master's thesis in Mechanical Engineering
Supervisor: Bjørn Haugen
June 2021

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

NTNU
Norwegian University of
Science and Technology

Abstract

The Poynting effect is a phenomenon investigated by John Henry Poynting that proved steel wires to lengthen when subjected to torsion. This thesis include the preparations to perform experiments and finite element analysis (FEA) for different materials in order to document the Poynting effect. The thesis consist of a study on the Seth-Hill Family (SH) of generalized strain tensors. With this as fundamental principle, together with finite strain theory, a constant strain triangle (CST) element is investigated using finite element method (FEM). The triangle element is build up by two formulations: traditional strain and Assumed Natural Strain (ANS). These formulations together with SH strain is still considered at an experimental phase, and further investigation and optimization is needed.

Based on a model presented in a previous master thesis, a method for solving the equations is implemented and extended for this thesis. The method includes usage of open-source software for mesh generation and data visualisation. A simple nonlinear model consisting of these theoretical aspects is computed by a Python FEM solver. The numerical results from the FEM solver is presented to verify and investigate the Poynting effect. The results presents that the Poynting effect can be proven for several materials.

To perform experimental work for investigation of the Poynting effect, calculations of maximum torque and rotational angle was conducted. A suggested setup to perform the experimental work is presented in the thesis. The experiment could not be completed due to Covid-19 restrictions in the laboratory, and is presented as a suggestion for continuation of this thesis.

Sammendrag

Poynting effekten er et fenomen utforsket av John Henry Poynting, som beviser at tynne stålrør forlenges når de blir utsatt for torsjon. Denne avhandlingen inneholder numeriske undersøkelser av deformasjon og tøyning ved hjelp av elementmetoden (FEM), samt forberende arbeid for å kunne gjennomføre eksperiment for å undersøke Poynting effekten. Det var ønsket å avdekke om fenomenet er gyldig for flere materialtyper. Avhandlingen innebærer studie av Seth-Hill familien, en samling av generaliserte tøyningsuttrykk. Sammen med endelig tøyning er et konstant trekantelement undersøkt ved hjelp av elementmetoden. Trekantelementet er bygd opp ved bruk av tradisjonell og antatt naturlig tøyningsformulering. Dette er anerkjente metoder, som sammen med tøyningsutrykkene i Seth-Hill familien utgjør det teoretiske aspektet i oppgaven. Uttrykkene er ikke veletablert for denne bruk og ytterligere optimalisering og verifisering burde gjennomføres.

Basert på en modell presentert i en tidligere utgitt masteravhandling, er det tatt i bruk en metode for å gjennomføre studien. Røret ble kun undersøkt som et to-dimensjonalt system. Metoden inkluderer åpen kildekode for generering av mesh og visualisering. En enkel ikke-lineær modell er utarbeidet gjennom en elementmetode løser. De numeriske resultatene er presentert for å undersøke og verifisere Poynting effekten. Resultatene viser at Poynting effekten er gyldig for flere materialtyper.

For å gjennomføre eksperimentelt arbeid i undersøkelsen av Poynting effekten ble det utført kalkulasjoner av maksimal torsjon for hvert materiale, samt maksimal vridningsvinkel. Forslag til oppsett og gjennomførelse av eksperimentet er presentert. Det ble ikke anledning til å gjennomføre eksperimentet i tidsforløpet grunnet Covid-19 begrensinger i laboratoriet for denne avhandlingen og det er dermed inkludert som videreføring av dette arbeidet.

Preface

This master thesis is completed for the Department of Mechanical and Industrial Engineering (MTP) at Norwegian University of Science And Technology (NTNU). The study is based on previous research of Bjørn Haugen and Carlos Felippa. The thesis examines the characteristics of finite element method (FEM) and has been both educational and engaging. The thesis was completed during the spring of 2021. During this period, a global pandemic affected the world. Adapting to new working methods and restrictions in consideration, the ambitions and original plans for the thesis were influenced. Due to critical increase in infection during the final phase of the course new restrictions were launched in Trondheim. This resulted in shutdown of the workshop at the University and the experiment could not be executed as planned.

Acknowledgements

We would like to emphasise our gratitude to our supervisor Bjørn Haugen. His guideline and help during the completion of this master thesis has been important for the completed work.

A gratitude should be given to the employees at the laboratory at NTNU. They shared great understanding and advice when we needed it. The experimental setup would not be as complete without their shared expertise.

We would like to thank our student partner Anders Østebø for cooperation while working towards the formulation of the strain equations. It was beneficial to have another student to discuss composition and solutions during the process.

Finally, we would also like to thank each other for a great teamwork during the completion of this thesis. It has been essential to have a partner for discussion and motivation during the writing process.

Contents

List of Figures

List of Tables

Acronyms

ANS Assumed Natural Strain. I, 1, 6, 15, 25, 29, 42, 52, 54–56

API Application Programming Interface. 26

CST constant strain triangle. I

DOF degree of freedom. 28, 29, 45

FEA finite element analysis. I, 2, 45, 55, 56

FEM finite element method. I, III, 2, 26, 28, 42, 45, 54–56

MTP Department of Mechanical and Industrial Engineering. III

NTNU Norwegian University of Science And Technology. III

SH Seth-Hill Family. I, IX, 1, 5, 6, 8–10, 25, 26, 28, 30, 42, 43, 45–52, 54–56

TL Total Lagrange. 2, 3

1 Introduction

When drilling in rough soil, different conflicts can be experienced. If the drill bit get stuck while the drilling continues, the bore will be exposed to large torsion. This may cause a lengthening or compression of the bore rod. For these kind of cases it will be beneficial to know what happens to the length of the rod. This is the main motivation for this thesis.

John Henry Poynting experimented with steel wires exposed to twisting and wanted to investigate the change in dimension for the wires while in the elastic region.[1, 2] The study of this phenomenon proved that the wires lengthen when twisted, as alternative c) in Figure 1.0.1 indicate. This effect is dependent on material properties and the strain measure used. By the use of this effect and strain calculations, the goal will be to establish a numerical model for calculating the strain in elements for different materials.

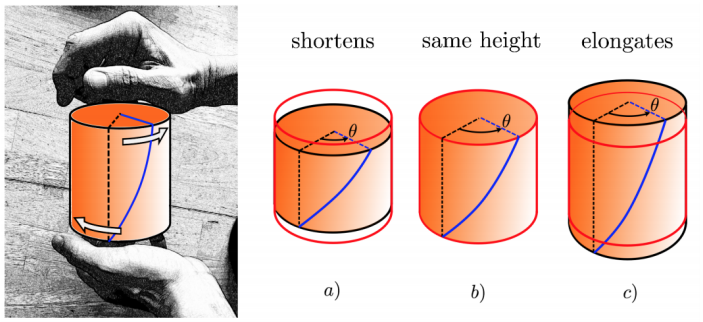


Figure 1.0.1: Cylinder exposed to torsion
[3]

This is a complex problem. To make it admissible and understandable a smaller element consisting of three sub-elements will be examined. The composition of this element will be formed as a triangle and the problem will only be considered in 2D.

In order to perform calculations for this fixed-end problem, some theoretical aspects need to be addressed. These include the main theory about strain and deformations, as well as the Seth-Hill Family (SH) with different expressions for strains.

There are two strategies to follow for investigation of the finite element strain. Traditional strain formulation uses a constant triangle formulation to obtain the local coordinates of the element. This formulation is based on the element strain displacement. The other strategy includes the Assumed Natural Strain (ANS) formulation. This formulation is based on a triangle set-up including three bars along each side of the element.

This thesis will include both traditional strain formulation and ANS-formulations in collaboration with Seth-Hill Family of finite strain measures. The theory and formulations in this thesis will be based on Carlos Felippa's methodology. [4]

1.1 Thesis outline

The aim of this thesis is to investigate the Poynting effect for pipes in a selection of materials subjected to free-end torsion, and implement a simple nonlinear model using finite strain measures.

2 Theoretical background

2.1 Finite element method

The finite element method (FEM) is a method widely used to solve complex problems. The method is based on giving a numerical solution to partial differential equations. With use of FEM the system viewed will be divided into subsystems consisting of smaller parts called *finite elements*. The finite elements are obtained by constructing a mesh for the object. [5, 6]

The method is commonly used to make a problem more admissible and understandable. The finite elements can be used to solve the problem numerically. The problem can be set up by a set of algebraic equations. When the differential equations are solved for the elements, they can be assembled into the total system again. The solution for the elements is also applicable for the entire system.

The use of FEM in a study or analysis is usually referred to as finite element analysis (FEA).

2.2 Continuum mechanics

The Total Lagrange (TL) description is a kinematic description of geometrically nonlinear finite element analysis (FEA) and is used for this fixed-end problem. In TL analysis the configuration is not changed, it is kept equal to the base configuration through the analysis. The stresses and strains can be measured with respect to this base configuration. [7]

2.2.1 Tensor notation

To determine the motion of a solid body a specific notation will be used in this thesis. This is called tensor notation and is an objective characteristic of a body. The tensors describe the motion of the body which is in a Cartesian frame. For the torsion problem in this thesis, the tensors are expressed in 2D. [8]

The main concept consider the tensor to be independent of the frame however the components depend on change in the frame. [8]

Different notations are used in continuum mechanics for tensor and matrix. For clearance, one notation will be followed in this thesis and is presented here:

$\mathbf{x}, \mathbf{E}$: Matrix notation

$\boldsymbol{x}, \boldsymbol{E}$: Tensor notation

2.2.2 Finite strain theory

Finite strain theory is a well known theory in continuum mechanics that considers large strains when a configuration is exposed to deformation, rotation or both. This will be explained more detailed in the upcoming sections.

2.2.3 Deformation gradient tensor

The configuration, that often is a body or an element, is during the analysis linked by a coordinate system - a Cartesian global frame. Each element is given a base frame and a reference frame with axes. When the elements are exposed to deformation, they can be expressed in terms of its coordinates $\boldsymbol{x}$, see Figure 2.2.1. The free-end torsion problem in this study is only considered in

two-dimensional space and coordinates referring to the third axis (x_3) is consequently not included in the equations.

The transformation will track the location of the base particle $K_0(B)$ to $K_t(B)$. The displacement vector of the particle can be expressed by the following equation [7]:

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} x_1 - X_1 \\ x_2 - X_2 \end{bmatrix} = \boldsymbol{x} - \boldsymbol{X} \tag{2.2.1}$$

The deformation gradient is expressed as $\boldsymbol{F}$ and represent the local deformation at each component coordinate $\mathbf{x}$. This is obtained by taking the partial derivative of $\mathbf{x}$ with respect to the undeformed configuration's coordinate $\mathbf{X}$, see Figure 2.2.1. [9, 10]

$$d\boldsymbol{x} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}} d\boldsymbol{X} = \boldsymbol{F}(\boldsymbol{X}, t) d\boldsymbol{X} \tag{2.2.2}$$

By use of tensor notation for the deformation gradient, it can be expressed as Equation (2.2.3). $\boldsymbol{F}$ can also be arranged as a matrix, as seen in Equation (2.2.4).

$$\boldsymbol{F} : F_{ij} = \frac{\partial x_i}{\partial X_j} \tag{2.2.3}$$

$$\boldsymbol{F} = \begin{bmatrix} \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_2} \\ \frac{\partial x_2}{\partial X_1} & \frac{\partial x_2}{\partial X_2} \end{bmatrix} \tag{2.2.4}$$

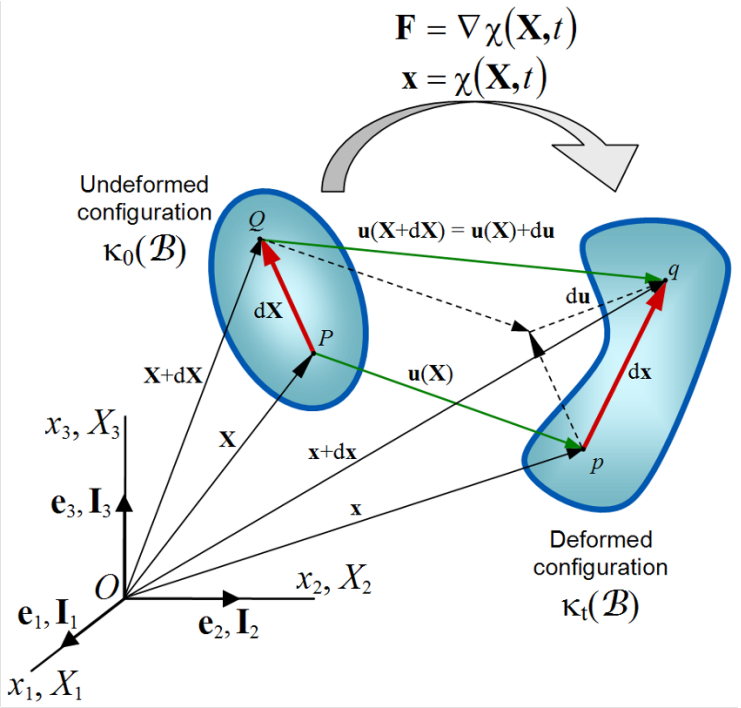


Figure 2.2.1: Deformation of a continuum body
[10]

The material gradient is a second order tensor that characterizes the local deformation at a material point at position $\boldsymbol{X}$. The material coordinates $\boldsymbol{x}_i$ are by the TL description defined as seen in Equation (2.2.5), where $\boldsymbol{u}(\boldsymbol{X}, t)$ represent the displacement field and $\boldsymbol{b}(t)$ the displacement vector

representing rigid-body translation. It is common to superimpose the coordinate system for the deformed and undeformed configurations such that $\boldsymbol{b} = \boldsymbol{0}$. [10]

$$\boldsymbol{u}(\boldsymbol{X}, t) = \boldsymbol{b}(t) + \boldsymbol{x}(\boldsymbol{X}, t) - \boldsymbol{X} \tag{2.2.5}$$

$$\boldsymbol{U}(\boldsymbol{x}, t) = \boldsymbol{x} - \boldsymbol{X}(\boldsymbol{x}, t) \tag{2.2.6}$$

$$\nabla_x \boldsymbol{u} = \nabla_x \boldsymbol{x} - I = \boldsymbol{F} - \boldsymbol{I} \tag{2.2.7}$$

The inverse of the deformation gradient $\boldsymbol{F}^{-1}$ can be expressed as:

$$\boldsymbol{F}^{-1} = \begin{bmatrix} \frac{\partial X_1}{\partial x_1} & \frac{\partial X_1}{\partial x_2} \\ \frac{\partial X_2}{\partial x_1} & \frac{\partial X_2}{\partial x_2} \end{bmatrix} \tag{2.2.8}$$

The purpose of these matrices is that they can be used to relate the coordinate differentials.

$$d\boldsymbol{X} = \boldsymbol{F}^{-1} d\boldsymbol{x} \tag{2.2.9}$$

The displacement gradient $\boldsymbol{G}$ with respect to the reference configuration can be expressed by:

$$\boldsymbol{G}_{ij} = \frac{\partial u_i}{\partial X_j} \tag{2.2.10}$$

$$\boldsymbol{G} = \boldsymbol{F} - \boldsymbol{I} = \begin{bmatrix} \frac{\partial u_1}{\partial X_1} & \frac{\partial u_1}{\partial X_2} \\ \frac{\partial u_2}{\partial X_1} & \frac{\partial u_2}{\partial X_2} \end{bmatrix} \tag{2.2.11}$$

2.2.4 Polar decomposition of the gradient tensor

Polar decomposition theorem claim that any deformation can be expressed as pure deformation followed by a rotation, or a rotation followed by deformation. This is illustrated in Figure 2.2.1. [7]

The tensors $\boldsymbol{F}$ and $\boldsymbol{G}$ describes the deformation measure in nonlinear continuum mechanics. The gradient tensor $\boldsymbol{F}$ can be decomposed into a product of two second order tensors.[11] Mathematically it can be described by the following equation

$$\boldsymbol{F} = \boldsymbol{RU} = \boldsymbol{VR} \qquad \begin{cases} \boldsymbol{U} = \text{right stretch tensor} \\ \boldsymbol{V} = \text{left stretch tensor} \\ \boldsymbol{F} = \text{deformation gradient tensor} \\ \boldsymbol{R} = \text{proper orthogonal tensor (rotation)} \end{cases} \tag{2.2.12}$$

The calculation of $\boldsymbol{F}$ is dependent on whether the configuration is first subject to rotation or deformation, as illustrated in Figure 2.2.2.

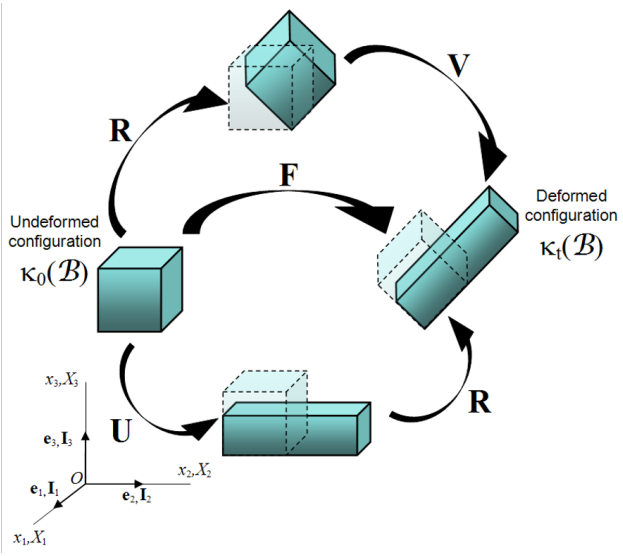


Figure 2.2.2: Left Right Stretch
[10]

2.3 Seth-Hill Family of strains

The Seth-Hill Family (SH) consists of generalized strain tensors. It was proposed in 1964 by B. Seth, who was the first to show that the Green and Almansi strain tensors are special cases of a more general strain measure. In 1968, Rodney Hill extended the idea. The Seth-Hill Family of strain measures can be expressed as Green-Lagrangian strain, Biot strain, Hencky strain, and Almansi strain. All the equations are based on the main SH equation in Equation (2.3.1). [4]

$$\varepsilon^{(m)} = \frac{1}{m}(\lambda^m - 1) \tag{2.3.1}$$

The different strain measure equations are connected to different values of m, known as the measure index.[4] The correspondence between value of m and equation is displayed in table 1.

Finite strain measure	**Values of m**
Green	m = 2
Biot	m = 1
Hencky	m = 0
Swainger	m = -1
Almansi	m = -2

Table 1: Finite strains

If $m > 0$: Strain will not go to infinite under compression, and therefore not converge
If $m < 0$: Strain will not go to infinite under tension, and therefore not converge

2.3.1 Seth-Hill strain using traditional formulation

The 1D finite strain measure using traditional formulation can be expressed as:

$$\varepsilon^{(m)} = \frac{1}{m}((1+g)^m - 1) \tag{2.3.2}$$

Taylor series expansion about $g = 0$ gives:

$$g - (m-1)\frac{g^2}{2} + (m-1)(m-2)\frac{g^3}{6} - H.O \tag{2.3.3}$$

The higher order parts of the Taylor series will not be included.

A generalized strain tensor can describe how a particular result may be extended for obtaining better agreement with experimental data. [12]

2.3.2 Seth-Hill strain using ANS-formulation

The 1D finite axial strain measure using ANS formulation can be expressed as:

$$\boldsymbol{\varepsilon}^{(m)} = \frac{1}{m}((\lambda)^m - 1) \tag{2.3.4}$$

The axial strain for ANS-element can be written in terms of the stretch λ where $\lambda = \frac{L}{L_0}$. Variation of strain is associated with variation in L. The first and second length derivative of the strain equation is given by Equation (2.3.5) and Equation (2.3.6) respectively.

$$\varepsilon^{(m)}_{L_d} = \frac{\partial \varepsilon}{\partial L_d} = \frac{L^{m-1}}{L_0^m} \tag{2.3.5}$$

$$\varepsilon^{(m)}_{L_d^2} = \frac{\partial^2 \varepsilon}{\partial^2 L_d} = \frac{(m-1)L^{m-2}}{L_0^m} \tag{2.3.6}$$

These equations will be used as base for each strain equations in the Seth-Hill Family. The separate SH equations for ANS-formulation are presented in Sections 3.4.2 to 3.4.6.

3 Triangle element

3.1 Strain gauge rosette

When investigating an element a common way to measure the strain is by use of strain gauges on the element. The strain gauge can only measure the strain in one direction. If several directions are desired to be measured the gauges must be placed at each bar in the element. When two or more strain gauges are placed at one location on the stressed element, it is referred to as *strain gauge rosette*. The most common types are presented in Figure 3.1.1. [13]

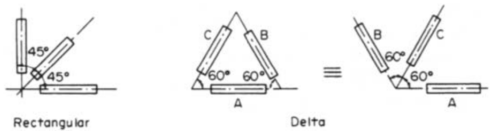


Figure 3.1.1: Typical strain gauge rosette configurations
[13]

To obtain the strain in the relevant pipe, only one element from the pipe is examined in this free-end torsion problem. This element is shaped as an equilateral triangle. The element is replaced by three side bars in order to obtain the strain, see Figure 3.1.2 for illustration. To obtain strain values in three nonaligned directions, strain gauge are placed on each bar. This result in a formation similar to strain gauge rosette, as demonstrated in Figure 3.1.3. The rosette gauge is a delta configuration with $\theta = 0$ and 120 degrees. [4, 13]

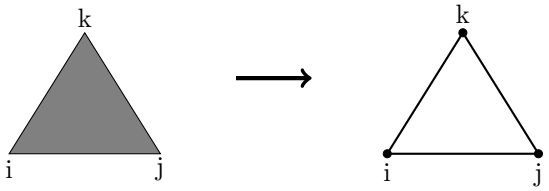


Figure 3.1.2: From element to three side-bars

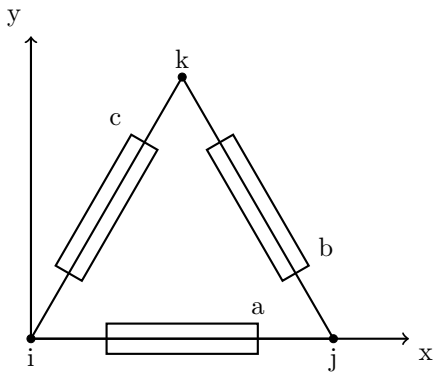


Figure 3.1.3: Equilateral triangle with strain gauge

3.2 Principal stretches

The deformations in the element are based on the change in length of each side bar of the element. These are the principle stretches in the element which can be determined by Equation (3.2.1).

$$\lambda = \frac{L}{L_0} \begin{cases} \lambda > 1 & extended \\ \lambda = 1 & unstretched \\ \lambda < 1 & compressed \end{cases} \tag{3.2.1}$$

To investigate the stretch in the element, a node can be moved and the stretch can be measured in the new element consisting of node **i,j,k'** (see Figure 3.2.1).

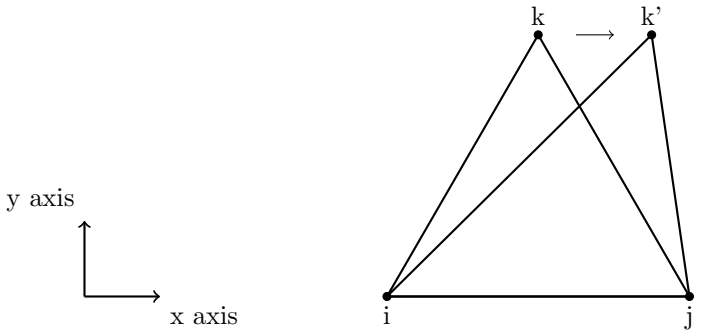


Figure 3.2.1: Equilateral triangle with node displacement

3.3 Traditional strain for triangle element

This section will include the mathematical aspects of the implementation of a simple nonlinear model in Python using traditional strain formulation. This is done to obtain the numerical results. The goal is to implement a model to calculate the generalized strain equations of the Seth-Hill Family and compare them in a plot.

The constant strain triangle formulation is used to find the local coordinates of the element. This method is further explained in appendix A. The nodes of the element are placed in a Cartesian coordinate system. To simplify the equations, node i and j are located on the x-axis. The location of the nodes are inserted in vector $\mathbf{e}_x$ and $\mathbf{e}_y$.

$$\begin{aligned} \mathbf{e}_x &= [x_i, x_j, x_k] \\ \mathbf{e}_y &= [y_i, y_j, y_k] \end{aligned} \tag{3.3.1}$$

The strain components are defined as ε_{xx}, ε_{yy} and γ_{xy} and is arranged as a 3-component strain vector $\boldsymbol{\varepsilon}$:

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \end{bmatrix} = \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} + \varepsilon_{yx} \end{bmatrix} = \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{bmatrix} \tag{3.3.2}$$

To obtain the strain equations, it is necessary to compute the strains over the element. This will be achieved through cyclic permutation of the nodes *i,j,k* and partial derivation of the area coordinates, explained in Appendix A. The partial derivatives of the area coordinates represents the strains over the element. These are gathered in the element strain displacement matrix, $\mathbf{B}_\varepsilon$. The displacement vector, $\mathbf{v}$, consist of the displacement coordinate of each node. Equation (3.3.3) and

Equation (3.3.5) represents the element strain displacement matrix and the displacement vector respectively.

$$\mathbf{B}_\varepsilon = \begin{bmatrix} \frac{\partial N_i}{\partial X} & 0 & \frac{\partial N_j}{\partial X} & 0 & \frac{\partial N_k}{\partial X} & 0 \\ 0 & \frac{\partial N_i}{\partial Y} & 0 & \frac{\partial N_j}{\partial Y} & 0 & \frac{\partial N_k}{\partial Y} \\ \frac{\partial N_i}{\partial Y} & \frac{\partial N_i}{\partial X} & \frac{\partial N_j}{\partial Y} & \frac{\partial N_j}{\partial X} & \frac{\partial N_k}{\partial Y} & \frac{\partial N_k}{\partial X} \end{bmatrix} \tag{3.3.3}$$

$$N = \frac{A_i}{A} \tag{3.3.4}$$

$$\mathbf{v}^T = \begin{bmatrix} u_i & v_i & u_j & v_j & u_k & v_k \end{bmatrix} \tag{3.3.5}$$

Through matrix multiplication of $\mathbf{B}_\varepsilon$ with the displacement vector $\mathbf{v}$, which consists of the displacement coordinate of each node, we could compute the linear strain, meaning Biot strain, described in section Section 3.3.3. Equation (3.3.6) and Equation (3.3.7) represents the calculations of the strains over the element.

$$\boldsymbol{\varepsilon} = \mathbf{B}_\varepsilon \mathbf{v} \tag{3.3.6}$$

$$\begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_i}{\partial X} & 0 & \frac{\partial N_j}{\partial X} & 0 & \frac{\partial N_k}{\partial X} & 0 \\ 0 & \frac{\partial N_i}{\partial Y} & 0 & \frac{\partial N_j}{\partial Y} & 0 & \frac{\partial N_k}{\partial Y} \\ \frac{\partial N_i}{\partial Y} & \frac{\partial N_i}{\partial X} & \frac{\partial N_j}{\partial Y} & \frac{\partial N_j}{\partial X} & \frac{\partial N_k}{\partial Y} & \frac{\partial N_k}{\partial X} \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_k \\ v_k \end{bmatrix} = \begin{bmatrix} u_{,x} \\ v_{,y} \\ u_{,y} + v_{,x} \end{bmatrix} \tag{3.3.7}$$

To obtain the nonlinear strain equations, it is necessary to modify the element strain displacement matrix, $\mathbf{B}_\varepsilon$. The partial derivatives of the area coordinates will be arranged by a 4x6 matrix. The modified displacement gradient, $\mathbf{B}_{gv}$, can be defined as:

$$\mathbf{B}_{gv} = \begin{bmatrix} \frac{\partial N_i}{\partial X} & 0 & \frac{\partial N_j}{\partial X} & 0 & \frac{\partial N_k}{\partial X} & 0 \\ 0 & \frac{\partial N_i}{\partial X} & 0 & \frac{\partial N_j}{\partial X} & 0 & \frac{\partial N_k}{\partial X} \\ \frac{\partial N_i}{\partial Y} & 0 & \frac{\partial N_j}{\partial Y} & 0 & \frac{\partial N_k}{\partial Y} & 0 \\ 0 & \frac{\partial N_i}{\partial Y} & 0 & \frac{\partial N_j}{\partial Y} & 0 & \frac{\partial N_k}{\partial Y} \end{bmatrix} \tag{3.3.8}$$

By inserting Equation (3.3.8) into Equation (3.3.7) we can compute the strain equation for solving the nonlinear strains. This is illustrated in Equation (3.3.9).

$$\begin{bmatrix} \frac{\partial u}{\partial X} \\ \frac{\partial v}{\partial X} \\ \frac{\partial u}{\partial Y} \\ \frac{\partial v}{\partial Y} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_i}{\partial X} & 0 & \frac{\partial N_j}{\partial X} & 0 & \frac{\partial N_k}{\partial X} & 0 \\ 0 & \frac{\partial N_i}{\partial X} & 0 & \frac{\partial N_j}{\partial X} & 0 & \frac{\partial N_k}{\partial X} \\ \frac{\partial N_i}{\partial Y} & 0 & \frac{\partial N_j}{\partial Y} & 0 & \frac{\partial N_k}{\partial Y} & 0 \\ 0 & \frac{\partial N_i}{\partial Y} & 0 & \frac{\partial N_j}{\partial Y} & 0 & \frac{\partial N_k}{\partial Y} \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_k \\ v_k \end{bmatrix} = \begin{bmatrix} u_{,x} \\ v_{,x} \\ u_{,y} \\ v_{,y} \end{bmatrix} \tag{3.3.9}$$

For solving the nonlinear analysis with the Seth-Hill Family strains, it is beneficial to arrange the displacement gradient, $\mathbf{B}_{gv}$, as a four-component vector, $\mathbf{g}$ [7]:

$$\mathbf{g}^T = \begin{bmatrix} g_1 & g_2 & g_3 & g_4 \end{bmatrix} = \begin{bmatrix} \frac{\partial u}{\partial X} & \frac{\partial v}{\partial X} & \frac{\partial u}{\partial Y} & \frac{\partial v}{\partial Y} \end{bmatrix} \tag{3.3.10}$$

By inserting the partial derivative of $\mathbf{g}$ into $\mathbf{B}_{gv}$, the displacement gradient matrix used for calculations of tangent stiffness is obtained, see Equation (3.3.11).

$$\mathbf{B}_{gv} = \begin{bmatrix} \frac{\partial g_1}{\partial u_i} & \frac{\partial g_1}{\partial v_i} & \frac{\partial g_1}{\partial u_j} & \frac{\partial g_1}{\partial v_j} & \frac{\partial g_1}{\partial u_k} & \frac{\partial g_1}{\partial v_k} \\ \frac{\partial g_2}{\partial u_i} & \frac{\partial g_2}{\partial v_i} & \frac{\partial g_2}{\partial u_j} & \frac{\partial g_2}{\partial v_j} & \frac{\partial g_2}{\partial u_k} & \frac{\partial g_2}{\partial v_k} \\ \frac{\partial g_3}{\partial u_i} & \frac{\partial g_3}{\partial v_i} & \frac{\partial g_3}{\partial u_j} & \frac{\partial g_3}{\partial v_j} & \frac{\partial g_3}{\partial u_k} & \frac{\partial g_3}{\partial v_k} \\ \frac{\partial g_4}{\partial u_i} & \frac{\partial g_4}{\partial v_i} & \frac{\partial g_4}{\partial u_j} & \frac{\partial g_4}{\partial v_j} & \frac{\partial g_4}{\partial u_k} & \frac{\partial g_4}{\partial v_k} \end{bmatrix} \tag{3.3.11}$$

3.3.1 Tangent stiffness

The tangent stiffness is established by the variation of the internal forces, $\mathbf{f}$, with respect to the displacement, $\mathbf{v}$. To build the tangent stiffness, it is convenient to start with the strain energy and variation of strains.

Strain energy is obtained by use of Equation (3.3.12). The m in $\varepsilon^{(m)}$ indicate the applicable SH strain.

$$\begin{aligned} u &= \frac{1}{2}\varepsilon^{(m)T}\mathbf{C}\varepsilon^{(m)} \\ \delta u &= \frac{1}{2}\delta\varepsilon^{(m)T}\mathbf{C}\varepsilon^{(m)} + \frac{1}{2}\varepsilon^{(m)T}\mathbf{C}\delta\varepsilon^{(m)} \\ \delta u &= \delta\varepsilon^{(m)T}\mathbf{C}\varepsilon^{(m)} \end{aligned} \tag{3.3.12}$$

Where the constitutive stiffness matrix, $\mathbf{C}$, for isotropic linear elastic material is:

$$\mathbf{C} = \frac{E}{1-\upsilon^2}\begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & \frac{1-\upsilon}{2} \end{bmatrix} \tag{3.3.13}$$

Variation of the strains can be described as:

$$\begin{aligned} \delta\varepsilon = \frac{\partial\varepsilon}{\partial\mathbf{g}}\frac{\partial\mathbf{g}}{\partial\mathbf{v}}\delta\mathbf{v} = \mathbf{B}_{\varepsilon g}\mathbf{B}_{gv}\delta\mathbf{v} \\ \text{where: } \mathbf{B}_{\varepsilon g_{ij}} = \frac{\partial\varepsilon_i}{\partial\mathbf{g}_j} \text{ and } \mathbf{B}_{gv_{ij}} = \frac{\partial\mathbf{g}_i}{\partial\mathbf{v}_j} \end{aligned} \tag{3.3.14}$$

Matrix $\mathbf{B}_{\varepsilon g}$ presented in Equation (3.3.14) depend on which strain formulation used from the Seth-Hill Family, while matrix $\mathbf{B}_{gv}$ depend on the element size (i.e. 3 or 4 nodes). The matrix contains the gradients of Cartesian strain with respect to displacement gradient.

$$\mathbf{B}_{\varepsilon g(m)} = \begin{bmatrix} \frac{\partial\varepsilon_1}{\partial g_1} & \frac{\partial\varepsilon_1}{\partial g_2} & \frac{\partial\varepsilon_1}{\partial g_3} & \frac{\partial\varepsilon_1}{\partial g_4} \\ \frac{\partial\varepsilon_2}{\partial g_1} & \frac{\partial\varepsilon_2}{\partial g_2} & \frac{\partial\varepsilon_2}{\partial g_3} & \frac{\partial\varepsilon_2}{\partial g_4} \\ \frac{\partial\varepsilon_3}{\partial g_1} & \frac{\partial\varepsilon_3}{\partial g_2} & \frac{\partial\varepsilon_3}{\partial g_3} & \frac{\partial\varepsilon_3}{\partial g_4} \end{bmatrix} \tag{3.3.15}$$

If we insert Equation (3.3.14) into Equation (3.3.12), the internal force expression can be obtained:

$$\delta u = \delta\varepsilon^T\mathbf{C}\varepsilon = \delta\mathbf{v}^T\mathbf{B}_{gv}^T\mathbf{B}_{\varepsilon g}^T\mathbf{C}\varepsilon = \delta\mathbf{v}^T\mathbf{B}_{gv}^T\mathbf{B}_{\varepsilon g}^T\mathbf{s} = \delta\mathbf{v}^T\mathbf{f} \tag{3.3.16}$$

where the stresses for linear material are defined as:

$$\mathbf{s} = \mathbf{C}\varepsilon \tag{3.3.17}$$

$$\mathbf{f} = \int_V \mathbf{B}_{gv}^T \mathbf{B}_{\varepsilon g}^T \mathbf{s}\, dV \tag{3.3.18}$$

Stiffness from gradients of the forces:

$$\Delta\mathbf{f} = \frac{\partial \mathbf{f}}{\partial \mathbf{v}} \Delta\mathbf{v} = \mathbf{K}\Delta\mathbf{v} \tag{3.3.19}$$

$$\begin{aligned} \mathbf{f} &= \mathbf{B}_{gv}^T \mathbf{B}_{\varepsilon g}^T \mathbf{s} \\ \delta\mathbf{f} &= \underbrace{\delta\mathbf{B}_{gv}^T \mathbf{B}_{\varepsilon g}^T \mathbf{s}}_{1} + \underbrace{\mathbf{B}_{gv}^T \delta\mathbf{B}_{\varepsilon g}^T \mathbf{s}}_{2} + \underbrace{\mathbf{B}_{gv}^T \mathbf{B}_{\varepsilon g}^T \delta\mathbf{s}}_{3} \end{aligned} \tag{3.3.20}$$

The material stiffness matrix and geometric stiffness matrix is obtained by the variation of the internal force vector. From the third therm in Equation (3.3.20) the material stiffness, $\mathbf{K}_3$, can be obtained.

$$\delta\mathbf{s} = \mathbf{C}\frac{\partial \varepsilon}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial \mathbf{v}} = \mathbf{C}\mathbf{B}_{\varepsilon g}\mathbf{B}_{gv}\delta\mathbf{v} \tag{3.3.21}$$

$$\mathbf{K}_3 = \mathbf{B}_{gv}^T \mathbf{B}_{\varepsilon g}^T \mathbf{C}\mathbf{B}_{\varepsilon g}\mathbf{B}_{gv} \tag{3.3.22}$$

Further, the geometric stiffness can be obtained from the second therm, in Equation (3.3.20).

$$\delta\mathbf{B}_{\varepsilon g}\mathbf{s} = \frac{\partial \mathbf{B}_{\varepsilon g}^T}{\partial \mathbf{g}}\mathbf{s}\frac{\partial \mathbf{g}}{\partial \mathbf{v}}\delta\mathbf{v} \tag{3.3.23}$$

$$\mathbf{K}_2 = \mathbf{B}_{gv}^T \frac{\partial \mathbf{B}_{\varepsilon g}^T}{\partial \mathbf{g}}\mathbf{s}\mathbf{B}_{gv}\delta\mathbf{v} = \mathbf{B}_{gv}^T \mathbf{S}\mathbf{B}_{gv} \tag{3.3.24}$$

The first therm in Equation (3.3.20) will be neglected, as the expression leads to a zero division.

$$\delta\mathbf{B}_{gv} = \frac{\partial \mathbf{B}_{gv}}{\partial v}\delta\mathbf{v} = 0 \tag{3.3.25}$$

The tangent stiffness for traditional strain formulation is generated by adding the geometric and material stiffness matrices.

$$\mathbf{K}_t = \mathbf{K}_2 + \mathbf{K}_3 \tag{3.3.26}$$

3.3.2 Green Lagrangian Strain

The strains can be developed by use of the Green equations for the three directions. The Green strain tensor is widely used due to its simplicity, thus easy to compute. This method is good for small to moderate strains and rotations. [4] For the Green Lagrangian element, Equation (3.3.27) is used to calculate the Green strain for the problem. [7]

By inserting Equation (3.3.10) in Equation (3.3.2) we get

$$\begin{aligned} \varepsilon_1^G &= g_1 + \frac{1}{2}(g_1^2 + g_2^2) \\ \varepsilon_2^G &= g_4 + \frac{1}{2}(g_3^2 + g_4^2) \\ \varepsilon_3^G &= g_2 + g_3 + g_1 g_3 + g_2 g_4 \end{aligned} \tag{3.3.27}$$

The first derivatives of the Green Lagrangian strain equations is given by Equation (3.3.28).

$$\begin{aligned} d\varepsilon_1^G &= (1+g_1)dg_1 + g_2 dg_2 \\ d\varepsilon_2^G &= (1+g_4)dg_4 + g_3 dg_3 \\ d\varepsilon_3^G &= g_3 dg_1 + (1+g_4)dg_2 + (1+g_1)dg_3 + g_2 dg_4 \end{aligned} \tag{3.3.28}$$

The second derivatives of the Green Lagrangian strain equations is given by Equation (3.3.29).

$$\begin{aligned} d^2\varepsilon_1^G &= d^2g_1 + d^2g_2 \\ d^2\varepsilon_2^G &= d^2g_4 + d^2g_3 \\ d^2\varepsilon_3^G &= d^2g_1 + d^2g_2 + d^2g_3 + d^2g_4 \end{aligned} \tag{3.3.29}$$

3.3.3 Biot strain

The Biot strain is a generalization of engineering strain and is the simplest generalization of linearized strains. It is adequate for initial stress problems and hyperelasticity. This method is quite popular, but will cause self straining for large rotations. [4] To compute the strain, the element strain displacement matrix, $\mathbf{B}_\varepsilon^B$, is multiplied with the displacement vector, $\mathbf{u}$. To calculate the Biot strain, Equation (3.3.30) is used.

$$\begin{aligned} \varepsilon_1^B &= g_1 \\ \varepsilon_2^B &= g_4 \\ \varepsilon_3^B &= g_2 + g_3 \end{aligned} \tag{3.3.30}$$

The first derivatives of the Biot strains is given by Equation (3.3.31).

$$\begin{aligned} d\varepsilon_1^B &= dg_1 \\ d\varepsilon_2^B &= dg_4 \\ d\varepsilon_3^B &= dg_2 + dg_3 \end{aligned} \tag{3.3.31}$$

The second derivatives of the Biot strain is given by Equation (3.3.32).

$$\begin{aligned} d^2\varepsilon_1^B &= 0 \\ d^2\varepsilon_2^B &= 0 \\ d^2\varepsilon_3^B &= 0 \end{aligned} \tag{3.3.32}$$

3.3.4 Hencky strain

Hencky strain will be a common choice to use for finite elastoplasticity. It is a logarithmic strain and can cause complications to compute in 3D. This method is prone to singularities, and therefore often replaced by logarithmic free equations. To compute the Hencky strain, Equation (3.3.33) is used. [4]

$$\begin{aligned} \varepsilon_1^H &= \log(1+g_1) \\ \varepsilon_2^H &= \log(1+g_4) \\ \varepsilon_3^H &= \log(1+g_2+g_3) \end{aligned} \tag{3.3.33}$$

Equation (3.3.34) represents the first derivatives of the Hencky strains.

$$\begin{aligned} d\varepsilon_1^H &= \frac{1}{(1+g_1)} dg_1 \\ d\varepsilon_2^H &= \frac{1}{(1+g_4)} dg_4 \\ d\varepsilon_3^H &= \frac{1}{(1+g_2+g_3)} dg_2 + \frac{1}{(1+g_2+g_3)} dg_3 \end{aligned} \tag{3.3.34}$$

The second derivatives of the Hencky strain equations is given by Equation (3.3.35).

$$\begin{aligned} d^2\varepsilon_1^H &= -\frac{1}{(1+g_1)^2} d^2 g_1 \\ d^2\varepsilon_2^H &= -\frac{1}{(1+g_4)^2} d^2 g_4 \\ d^2\varepsilon_3^H &= -\frac{1}{(1+g_2+g_3)^2} d^2 g_2 - \frac{1}{(1+g_2+g_3)^2} d^2 g_3 \end{aligned} \tag{3.3.35}$$

3.3.5 Swainger strain

To develop the strain equations for Swainger strain, Equation (3.3.36) is used. This method is rarely used, due to its difficulties to compute in 3D. It is the counterpart to the more popular method, Engineering strain. [4]

$$\begin{aligned} \varepsilon_1^S &= \frac{g_1}{(1+g_1)} \\ \varepsilon_2^S &= \frac{g_4}{(1+g_4)} \\ \varepsilon_3^S &= \frac{g_2+g_3}{(1+g_2+g_3)} \end{aligned} \tag{3.3.36}$$

The first derivatives of Swainger strain is given by Equation (3.3.37).

$$\begin{aligned} d\varepsilon_1^S &= \frac{1}{(1+g_1)^2} dg_1 \\ d\varepsilon_2^S &= \frac{1}{(1+g_4)^2} dg_4 \\ d\varepsilon_3^S &= \frac{1}{(1+g_2+g_3)^2} dg_2 + \frac{1}{(1+g_2+g_3)^2} dg_3 \end{aligned} \tag{3.3.37}$$

The second derivatives of the Swainger strain equations is given by Equation (3.3.38).

$$\begin{aligned} d^2\varepsilon_1^S &= -\frac{2}{(1+g_1)^3} d^2 g_1 \\ d^2\varepsilon_2^S &= -\frac{2}{(1+g_4)^3} d^2 g_4 \\ d^2\varepsilon_3^S &= -\frac{2}{(1+g_2+g_3)^3} d^2 g_2 - \frac{2}{(1+g_2+g_3)^3} d^2 g_3 \end{aligned} \tag{3.3.38}$$

3.3.6 Almansi strain

The Almansi strain is fitted for flow-like behaviour, but imprecise for elasticity. It is relatively easy to compute, but can cause complexity for matrix inversion. This method is the counterpart to the Green-strain, and not commonly used in Lagrangian form. The Almansi strain is calculated by Equation (3.3.39). [4]

$$\begin{aligned} \varepsilon_1^A &= -\frac{1}{2}((1+g_1)^2 - 1) \\ \varepsilon_2^A &= -\frac{1}{2}((1+g_4)^2 - 1) \\ \varepsilon_3^A &= -\frac{1}{2}((1+g_2+g_3)^2 - 1) \end{aligned} \tag{3.3.39}$$

Equation (3.3.40) represents the first derivatives of Almansi strain.

$$\begin{aligned} d\varepsilon_1^A &= \frac{1}{(1+g_1)^3} dg_1 \\ d\varepsilon_2^A &= \frac{1}{(1+g_4)^3} dg_4 \\ d\varepsilon_3^A &= \frac{1}{(1+g_2+g_3)^3} dg_2 + \frac{1}{(1+g_2+g_3)^3} dg_3 \end{aligned} \tag{3.3.40}$$

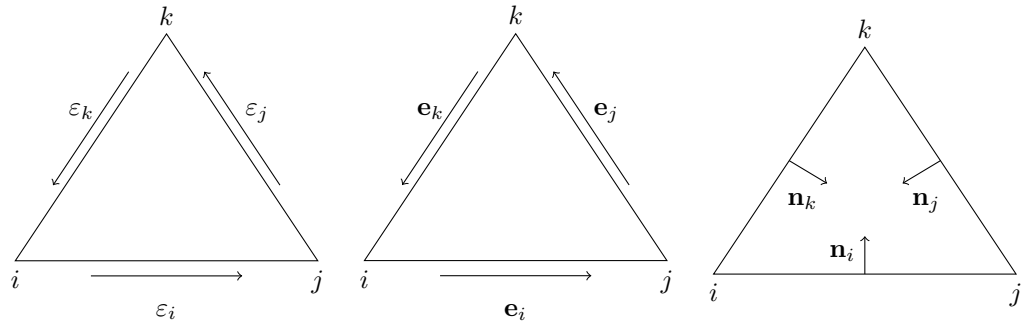
The second derivatives of the Almansi strain is given by Equation (3.3.41).

$$\begin{aligned} d^2\varepsilon_1^A &= -\frac{3}{(1+g_1)^4} d^2 g_1 \\ d^2\varepsilon_2^A &= -\frac{3}{(1+g_4)^4} d^2 g_4 \\ d^2\varepsilon_3^A &= -\frac{3}{(1+g_2+g_3)^4} d^2 g_2 - \frac{3}{(1+g_2+g_3)^4} d^2 g_3 \end{aligned} \tag{3.3.41}$$

3.4 Assumed Natural strain (ANS) for triangle element

The Assumed Natural Strain (ANS) formulation was first introduced by William in 1969.[14] The method was based on construction of four-node plane stress element where he assumed constant shear strain to be independent of the direct strains. During the following years, new approaches was made including developments of reduced and selective integration methods. The different methods concerns use of elements known as high-performance elements. High-performance elements is a general description for elements attaining special attributes which makes them desirable for problems concerning displacement, coarse mesh and can be extended to non-linear and dynamic problems. [15] This paper will follow the ANS-formulation from Militello and Felippa. [15, 16]

The strain, unit vector and inward normal which are used for the triangle element is defined as illustrated in Figure 3.4.1. The natural coordinate strains are selected to be strains along the three bars of the element. Three natural coordinate strains are obtain from this. The strains along a side *i*-*j* can be defined from the displacement $\mathbf{v}$ and the unit vector $\mathbf{e}_i$.



(a) Side edge strain (b) Unit vectors (c) Inward normals

Figure 3.4.1: Representation of side element

The unit vectors can be arranged in a vector as Equation (3.4.1) presents. The vector arrangement for inward normal's are presented in Equation (3.4.2). These vectors will be used to arrange the tangent stiffness matrix for the element.

$$\mathbf{e}_i = \begin{bmatrix} e_{ix} \\ e_{iy} \end{bmatrix} \tag{3.4.1}$$

$$\mathbf{n}_i = \begin{bmatrix} -e_{iy} \\ e_{ix} \end{bmatrix} \tag{3.4.2}$$

The definition of the natural strain is expressed as seen in Equation (3.4.3). Where $\mathbf{v}$ is the displacement vector as illustrated in Equation (3.3.5).

$$\boldsymbol{\varepsilon}_n = \frac{\partial \mathbf{v}_i}{\partial \mathbf{e}_i} \tag{3.4.3}$$

The subscript n and c represent Natural and Cartesian respectively.

For clearance, the notation used:

$\boldsymbol{\varepsilon}_c$: Cartesian strain
$\boldsymbol{\varepsilon}_n$: Natural strain
$\boldsymbol{e}_i$: Field vector

The natural coordinate strain ε_n can be expressed in therms of the global Cartesian components

as:

$$\begin{aligned}\varepsilon_n &= \frac{\partial u}{\partial x}e_{kx}^2 + \frac{\partial v}{\partial y}e_{ky}^2 + \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)e_{kx}e_{ky} \\ &= \varepsilon_{xx}e_{kx}^2 + \varepsilon_{yy}e_{ky}^2 + \gamma_{xy}e_{kx}e_{ky}\end{aligned} \tag{3.4.4}$$

Side edge length for each bar in the triangle is defined along each side, as indicated in Figure 3.4.2.

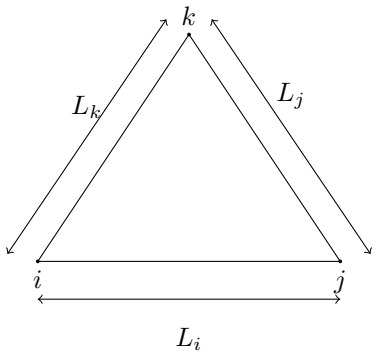


Figure 3.4.2: Side edge length

The variation of the natural strains is achieved by partial derivation of each side edge strain with respect to the length of the bar. This gives the expression in Equation (3.4.7), and arranged as a matrix in Equation (3.4.10)

$$\boldsymbol{\varepsilon}_n = \begin{bmatrix}\varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3\end{bmatrix} \tag{3.4.5}$$

$$\boldsymbol{\varepsilon}_c = \begin{bmatrix}\varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy}\end{bmatrix} \tag{3.4.6}$$

$$\delta\boldsymbol{\varepsilon}_n = \frac{\partial\boldsymbol{\varepsilon}_n}{\partial\mathbf{L}}\frac{\partial\mathbf{L}}{\partial\mathbf{v}}\delta\mathbf{v} = \mathbf{B}_{\varepsilon l}\mathbf{B}_{lv}\delta\mathbf{v} \tag{3.4.7}$$

Where the variation of strain with respect to the length is arranged as:

$$\frac{\partial\boldsymbol{\varepsilon}_n}{\partial\mathbf{L}} = \mathbf{B}_{\varepsilon l} = \begin{bmatrix}\frac{\partial\varepsilon_1}{\partial L_1} & 0 & 0 \\ 0 & \frac{\partial\varepsilon_2}{\partial L_2} & 0 \\ 0 & 0 & \frac{\partial\varepsilon_3}{\partial L_3}\end{bmatrix} = \begin{bmatrix}\frac{L_1}{\partial L_{01}^2} & 0 & 0 \\ 0 & \frac{L_2}{\partial L_{02}^2} & 0 \\ 0 & 0 & \frac{L_3}{\partial L_{03}^2}\end{bmatrix} \tag{3.4.8}$$

And the variation of the length with respect to the displacement is:

$$\frac{\partial\mathbf{L}}{\partial\mathbf{v}} = \mathbf{B}_{lv} = \begin{bmatrix}-\mathbf{e}_1^T & \mathbf{e}_1^T & 0 \\ 0 & -\mathbf{e}_2^T & \mathbf{e}_2^T \\ \mathbf{e}_3^T & 0 & -\mathbf{e}_3^T\end{bmatrix} \tag{3.4.9}$$

By combining these equations the variation of strain can be established as:

$$\delta\boldsymbol{\varepsilon}_n = \begin{bmatrix}\frac{L_1}{\partial L_{01}^2} & 0 & 0 \\ 0 & \frac{L_2}{\partial L_{02}^2} & 0 \\ 0 & 0 & \frac{L_3}{\partial L_{03}^2}\end{bmatrix}\begin{bmatrix}-\mathbf{e}_1^T & \mathbf{e}_1^T & 0 \\ 0 & -\mathbf{e}_2^T & \mathbf{e}_2^T \\ \mathbf{e}_3^T & 0 & -\mathbf{e}_3^T\end{bmatrix}\delta\mathbf{v} \tag{3.4.10}$$

3.4.1 Tangent stiffness

The tangent stiffness is a stiffness build up with respect to the internal forces. Thus, it it necessary to start looking at the internal energy before the tangent stiffness can be established.

Internal energy expressed in terms of Cartesian strains is presented in Equation (3.4.11).

$$u = \frac{1}{2}\int \boldsymbol{\varepsilon}_c^T \mathbf{C}_c \boldsymbol{\varepsilon} dV = \frac{1}{2}\boldsymbol{\varepsilon}_c^T \mathbf{C}_c \boldsymbol{\varepsilon} A_0 t \tag{3.4.11}$$

Transformation between Cartesian and Natural strains can be accomplished by use of a transformation matrix. This matrix is defined by the unit vectors for each bar as seen in Equation (3.4.13).

$$\boldsymbol{\varepsilon}_n = \mathbf{T}_{\varepsilon nc}\boldsymbol{\varepsilon}_c \tag{3.4.12}$$

Where the transformation matrix $\mathbf{T}_{\varepsilon nc}$ is defined as:

$$\mathbf{T}_{\varepsilon nc} = \begin{bmatrix} \mathbf{e}_{1x}^2 & \mathbf{e}_{1y}^2 & \mathbf{e}_{1x}\mathbf{e}_{1y} \\ \mathbf{e}_{2x}^2 & \mathbf{e}_{2y}^2 & \mathbf{e}_{2x}\mathbf{e}_{2y} \\ \mathbf{e}_{3x}^2 & \mathbf{e}_{3y}^2 & \mathbf{e}_{3x}\mathbf{e}_{3y} \end{bmatrix} \tag{3.4.13}$$

The inverse relationship defines the Cartesian strain:

$$\boldsymbol{\varepsilon}_c = \mathbf{T}_{\varepsilon cn}\boldsymbol{\varepsilon}_n \tag{3.4.14}$$

where:

$$\mathbf{T}_{\varepsilon cn} = \mathbf{T}_{\varepsilon nc}^{-1} \tag{3.4.15}$$

Matrix $\mathbf{T}_{\varepsilon nc}$ is non-singular and the inverse relationship which gives $\mathbf{T}_{\varepsilon cn}$ can be established numerically:

$$\begin{aligned} \boldsymbol{\varepsilon}_1 &= \varepsilon_{xx}\cos^2\theta_i + \varepsilon_{yy}\sin^2\theta_i + \gamma_{xy}\sin\theta_i\cos\theta_i \\ \boldsymbol{\varepsilon}_2 &= \varepsilon_{xx}\cos^2\theta_j + \varepsilon_{yy}\sin^2\theta_j + \gamma_{xy}\sin\theta_j\cos\theta_j \\ \boldsymbol{\varepsilon}_3 &= \varepsilon_{xx}\cos^2\theta_k + \varepsilon_{yy}\sin^2\theta_k + \gamma_{xy}\sin\theta_k\cos\theta_k \end{aligned} \tag{3.4.16}$$

The strain transformation is utilised on the stress-strain constitutive matrix. The correct transformation could be found through evaluating the virtual work:

$$\delta\boldsymbol{\varepsilon}_c^T\sigma = \delta\boldsymbol{\varepsilon}_c^T\mathbf{C}\boldsymbol{\varepsilon}_c = \delta\boldsymbol{\varepsilon}_n^T\mathbf{T}_{\varepsilon cn}^T\mathbf{C}\mathbf{T}_{\varepsilon cn}\boldsymbol{\varepsilon}_n \tag{3.4.17}$$

$$\mathbf{C}_n = \mathbf{T}_{\varepsilon cn}^T\mathbf{C}\mathbf{T}_{\varepsilon cn} \tag{3.4.18}$$

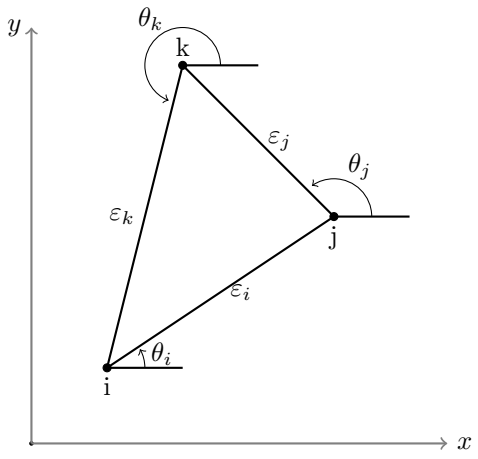


Figure 3.4.3: Triangle

Further the transformation matrix in Equation (3.4.15) is established as shown in Equation (3.4.19).

$$\begin{bmatrix}\varepsilon_1\\ \varepsilon_2\\ \varepsilon_3\end{bmatrix} = \begin{bmatrix}\cos^2\theta_i & \sin^2\theta_i & \cos\theta_i\sin\theta_i\\ \cos^2\theta_j & \sin^2\theta_j & \cos\theta_j\sin\theta_j\\ \cos^2\theta_k & \sin^2\theta_k & \cos\theta_k\sin\theta_k\end{bmatrix}\begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \gamma_{xy}\end{bmatrix} \tag{3.4.19}$$

where:

$$\cos\theta_i = \frac{e_{jx}-e_{ix}}{L_0}, \text{ and } \sin\theta_i = \frac{e_{jy}-e_{iy}}{L_0} \tag{3.4.20}$$

The initial length and deformed length is settled by the following formulas:

$$L_0 = \sqrt{X_i^2 + Y_i^2} \tag{3.4.21}$$

$$L_d = \sqrt{(X_i+U_i)^2 + (Y_i+V_i)^2} \tag{3.4.22}$$

Where $X_i = x_j - x_i$ and $Y_i = y_j - y_i$. This represents the length of the bar connecting between node i to node j. X and Y represents the coordinate of the node, while U and V is the displacement of the node. The same equation for initial and deformed length is used to establish the remaining bars of the triangle element.

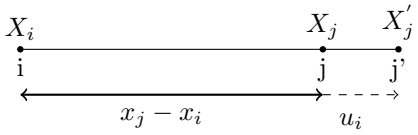


Figure 3.4.4: Definition of length and displacement

Internal energy can now be expressed as:

$$u = \frac{1}{2}\boldsymbol{\varepsilon}_n^T\mathbf{T}_{\varepsilon cn}^T\mathbf{C}_c\mathbf{T}_{\varepsilon cn}\boldsymbol{\varepsilon}_n A_0 t = \frac{1}{2}\boldsymbol{\varepsilon}_n^T\mathbf{C}_n\boldsymbol{\varepsilon}_n A_0 t \tag{3.4.23}$$

Internal forces from first variation of internal energy

$$\delta u = \frac{1}{2}\delta\boldsymbol{\varepsilon}_n^T\mathbf{C}_n\boldsymbol{\varepsilon}_n A_0 t + \frac{1}{2}\boldsymbol{\varepsilon}_n^T\mathbf{C}_n\delta\boldsymbol{\varepsilon}_n A_0 t = \delta\boldsymbol{\varepsilon}_n^T\mathbf{C}_n\boldsymbol{\varepsilon}_n A_0 t = \delta\boldsymbol{\varepsilon}_n^T\mathbf{s}_n A_0 t \tag{3.4.24}$$

Express the variation of the strains with respect to variations in nodal degrees of freedom.

$$\delta\boldsymbol{\varepsilon}_n = \frac{\partial\boldsymbol{\varepsilon}_n}{\partial\mathbf{L}}\frac{\partial\mathbf{L}}{\partial\mathbf{v}}\delta\mathbf{v} = \mathbf{B}_{\varepsilon l}\mathbf{B}_{lv}\delta\mathbf{v} \tag{3.4.25}$$

We insert this into the variation of the internal energy

$$\delta u = \delta\boldsymbol{\varepsilon}_n^T\mathbf{s}_n A_0 t = \delta\mathbf{v}^T\mathbf{B}_{lv}^T\mathbf{B}_{\varepsilon l}^T\mathbf{s}_n A_0 t = \delta\mathbf{v}^T\mathbf{f}_{int} \tag{3.4.26}$$

This defines the internal forces as

$$\mathbf{f}_{int} = \mathbf{B}_{lv}^T\mathbf{B}_{\varepsilon l}^T\mathbf{s}_n A_0 t \tag{3.4.27}$$

where the natural coordinate stresses are

$$\mathbf{s}_n = \mathbf{C}_n\boldsymbol{\varepsilon}_n \tag{3.4.28}$$

The stiffness matrix is derived from the variation of the internal forces

$$\delta\mathbf{f}_{int} = \frac{\partial\mathbf{f}_{int}}{\partial\mathbf{v}}\delta\mathbf{v} = \mathbf{K}\delta\mathbf{v} \tag{3.4.29}$$

This consist of three terms

$$\delta\mathbf{f}_{int} = \underbrace{\delta\mathbf{B}_{lv}^T\mathbf{B}_{\varepsilon l}^T\mathbf{s}_n A_0 t}_{1} + \underbrace{\mathbf{B}_{lv}^T\delta\mathbf{B}_{\varepsilon l}^T\mathbf{s}_n A_0 t}_{2} + \underbrace{\mathbf{B}_{lv}^T\mathbf{B}_{\varepsilon l}^T\delta\mathbf{s}_n A_0 t}_{3} \tag{3.4.30}$$

The third stiffness therm defines the material stiffness matrix:

$$\delta\mathbf{s}_n = \mathbf{C}_n\delta\boldsymbol{\varepsilon}_n = \mathbf{C}_n\mathbf{B}_{\varepsilon l}\mathbf{B}_{lv}\delta\mathbf{v} \tag{3.4.31}$$

$$\mathbf{K}_3 = \mathbf{B}_{lv}^T\mathbf{B}_{\varepsilon l}^T\mathbf{C}_n\mathbf{B}_{\varepsilon l}\mathbf{B}_{lv}A_0 t \tag{3.4.32}$$

The second stiffness term expression defines a part of the geometric stiffness:

$$\delta\mathbf{B}_{\varepsilon l} = \frac{\partial\mathbf{B}_{\varepsilon l}^T}{\partial L_i}\delta L_i \tag{3.4.33}$$

When the multiplication with $\mathbf{s}$ is included, this can be expressed as

$$\delta\mathbf{B}_{\varepsilon l}^T\mathbf{s} = \frac{\partial\mathbf{B}_{\varepsilon l}^T}{\partial L_i}\mathbf{s}\delta L_i = \begin{bmatrix} \frac{\partial\boldsymbol{\varepsilon}_1}{\partial L_1^2}s_1 & 0 & 0 \\ 0 & \frac{\partial\boldsymbol{\varepsilon}_2}{\partial L_2^2}s_2 & 0 \\ 0 & 0 & \frac{\partial\boldsymbol{\varepsilon}_3}{\partial L_3^2}s_3 \end{bmatrix}\begin{bmatrix}\delta L_1 \\ \delta L_2 \\ \delta L_3\end{bmatrix} = \mathbf{B}_{[\varepsilon_i, L_i L_i s_i]}\mathbf{B}_{lv}\delta\mathbf{v} \tag{3.4.34}$$

This gives us

$$\mathbf{K}_2 = \mathbf{B}_{lv}^T\mathbf{B}_{[\varepsilon_i, L_i L_i s_i]}\mathbf{B}_{lv} \tag{3.4.35}$$

The first therm of the variation of the internal forces, gives the remaining expression of the geometric stiffness matrix.

It is wanted to express the variation of $\mathbf{B}_{lv}^T$ with respect to nodal degrees of freedom. Before this can be accomplished, the equation must be expressed with respect to $\delta\alpha_i$.

$$\mathbf{B}_{lv}^T = \begin{bmatrix} -\mathbf{e}_1 & 0 & \mathbf{e}_3 \\ \mathbf{e}_1 & -\mathbf{e}_2 & 0 \\ 0 & \mathbf{e}_2 & -\mathbf{e}_3 \end{bmatrix} \tag{3.4.36}$$

This gives

$$\begin{aligned} \delta\mathbf{B}_{lv}^T = \frac{\partial \mathbf{B}_{lv}^T}{\partial \alpha_i}\delta\alpha_i &= \begin{bmatrix} -\mathbf{n}_1 & 0 & 0 \\ \mathbf{n}_1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\delta\alpha_1 + \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\mathbf{n}_2 & 0 \\ 0 & \mathbf{n}_2 & 0 \end{bmatrix}\delta\alpha_2 + \begin{bmatrix} 0 & 0 & \mathbf{n}_3 \\ 0 & 0 & 0 \\ 0 & 0 & -\mathbf{n}_3 \end{bmatrix}\delta\alpha_3 \\ &= \begin{bmatrix} -\mathbf{n}_1 & 0 & \mathbf{n}_3 \\ \mathbf{n}_1 & -\mathbf{n}_2 & 0 \\ 0 & \mathbf{n}_2 & -\mathbf{n}_3 \end{bmatrix}\begin{bmatrix} \delta\alpha_1 & 0 & 0 \\ 0 & \delta\alpha_2 & 0 \\ 0 & 0 & \delta\alpha_3 \end{bmatrix} \end{aligned} \tag{3.4.37}$$

The following term is defined:

$$\mathbf{S}_{HN} = \mathbf{B}_{\varepsilon l}^T \mathbf{s}_n \tag{3.4.38}$$

Combined with Equation (3.4.37) gives

$$\begin{aligned} \delta\mathbf{B}_{lv}^T\mathbf{B}_{\varepsilon l}^T\mathbf{S}_{HN} &= \begin{bmatrix} -\mathbf{n}_1 & 0 & \mathbf{n}_3 \\ \mathbf{n}_1 & -\mathbf{n}_2 & 0 \\ 0 & \mathbf{n}_2 & -\mathbf{n}_3 \end{bmatrix}\begin{bmatrix} \delta\alpha_1 & 0 & 0 \\ 0 & \delta\alpha_2 & 0 \\ 0 & 0 & \delta\alpha_3 \end{bmatrix}\begin{bmatrix} S_{HN1} \\ S_{HN2} \\ S_{HN3} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{n}_1 & 0 & \mathbf{n}_3 \\ \mathbf{n}_1 & \mathbf{n}_2 & 0 \\ 0 & \mathbf{n}_2 & \mathbf{n}_3 \end{bmatrix}\begin{bmatrix} S_{HN1} & 0 & 0 \\ 0 & S_{HN2} & 0 \\ 0 & 0 & S_{HN3} \end{bmatrix}\begin{bmatrix} \delta\alpha_1 \\ \delta\alpha_2 \\ \delta\alpha_3 \end{bmatrix} \end{aligned} \tag{3.4.39}$$

The rotation $\delta\alpha_i$ can be expressed with respect to the nodal degrees of freedom.

$$\delta\alpha_i = \frac{1}{L_i}\begin{bmatrix} -\mathbf{n}_i^T & \mathbf{n}_i^T \end{bmatrix}\begin{bmatrix} \delta u_1 \\ \delta v_1 \\ \delta u_2 \\ \delta v_2 \end{bmatrix} \tag{3.4.40}$$

When combining for all side edges:

$$\begin{bmatrix} \delta\alpha_1 \\ \delta\alpha_2 \\ \delta\alpha_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{L_1} & 0 & 0 \\ 0 & \frac{1}{L_2} & 0 \\ 0 & 0 & \frac{1}{L_3} \end{bmatrix}\begin{bmatrix} -\mathbf{n}_1^T & \mathbf{n}_1^T & 0 \\ 0 & -\mathbf{n}_2^T & \mathbf{n}_2^T \\ \mathbf{n}_3^T & 0 & -\mathbf{n}_3^T \end{bmatrix}\begin{bmatrix} \delta u_1 \\ \delta v_1 \\ \delta u_2 \\ \delta v_2 \\ \delta u_3 \\ \delta v_3 \end{bmatrix} \tag{3.4.41}$$

By combining Equation (3.4.39) and Equation (3.4.41), the stiffness expression term 1 can be written. (skriv om)

$$\begin{aligned} \delta\mathbf{B}_{lv}\mathbf{B}_{\varepsilon l}\mathbf{s} &= \begin{bmatrix} -\mathbf{n}_1 & 0 & \mathbf{n}_3 \\ \mathbf{n}_1 & -\mathbf{n}_2 & 0 \\ 0 & \mathbf{n}_2 & -\mathbf{n}_3 \end{bmatrix}\begin{bmatrix} \frac{S_{HN1}}{L_1} & 0 & 0 \\ 0 & \frac{S_{HN2}}{L_2} & 0 \\ 0 & 0 & \frac{S_{HN3}}{L_3} \end{bmatrix}\begin{bmatrix} -\mathbf{n}_1^T & \mathbf{n}_1^T & 0 \\ 0 & -\mathbf{n}_2^T & \mathbf{n}_2^T \\ \mathbf{n}_3^T & 0 & -\mathbf{n}_3^T \end{bmatrix}\delta\mathbf{v} \\ &= \mathbf{B}_k^T\mathbf{S}_{HN}\mathbf{B}_k\delta\mathbf{v}A_0 t \end{aligned} \tag{3.4.42}$$

This gives

$$\mathbf{K}_1 = \mathbf{B}_k^T \mathbf{S}_{HN} \mathbf{B}_k A_0 t \tag{3.4.43}$$

The tangent stiffness matrix is given by combining the expressions:

$$\mathbf{K}_T = \mathbf{K}_1 + \mathbf{K}_2 + \mathbf{K}_3 \tag{3.4.44}$$

3.4.2 Green Lagrangian strain

The bar strain equation for green Lagrangian strain is:

$$\varepsilon^G = \frac{1}{2}(\lambda^2 - 1) \tag{3.4.45}$$

Following the first length derivative is:

$$\varepsilon^G_{L_d} = \frac{L_d}{L_0^2} \tag{3.4.46}$$

and second length derivative:

$$\varepsilon^G_{L_d^2} = \frac{1}{L_0^2} \tag{3.4.47}$$

3.4.3 Biot strain

The strain equation for Biot strain using bar element theory is represented in Equation (3.4.48).

$$\varepsilon^B = \lambda - 1 \tag{3.4.48}$$

The first derivative of Biot strain is:

$$\varepsilon^B_{L_d} = \frac{1}{L_0} \tag{3.4.49}$$

And the second length derivative can be given by following expression:

$$\varepsilon^B_{L_d^2} = 0 \tag{3.4.50}$$

3.4.4 Hencky strain

Hencky strain is characterized by following logarithmic equation:

$$\varepsilon^H = \log(\lambda) \tag{3.4.51}$$

The first length derivative can be expressed by:

$$\varepsilon^H_{L_d} = \frac{1}{L_d} \tag{3.4.52}$$

Second length derivative:

$$\varepsilon^H_{L_d^2} = \frac{-1}{L_d^2} \tag{3.4.53}$$

3.4.5 Swainger strain

The bar strain equation for Swainger strain is:

$$\varepsilon^S = 1 - \frac{1}{\lambda} \tag{3.4.54}$$

That gives following equation for first length derivative:

$$\varepsilon^S_{L_d} = \frac{L_0}{L_d^2} \tag{3.4.55}$$

and for second length derivative

$$\varepsilon^S_{L_d^2} = \frac{-2L_0}{L_d^3} \tag{3.4.56}$$

3.4.6 Almansi strain

To calculate the Almansi strain, the transformation matrix needs to be defined by deformed geometry. The Almansi strain equation for bar element is:

$$\varepsilon^A = \frac{1}{2}(1 - \frac{1}{\lambda^2}) \tag{3.4.57}$$

Following the first length derivatives is:

$$\varepsilon^A_{L_d} = \frac{L_0^2}{L_d^3} \tag{3.4.58}$$

And the second length derivative:

$$\varepsilon^A_{L_d^2} = \frac{-3L_0^2}{L_d^4} \tag{3.4.59}$$

3.4.7 2D motion of ANS element

$$\varepsilon_{NG} = \frac{u_i - u_{i-1}}{L_0} + \frac{1}{2}(\frac{u_i - u_{i-1}}{L_0})^2 + \frac{1}{2}(\frac{v_i - v_{i-1}}{L_0})^2 \tag{3.4.60}$$

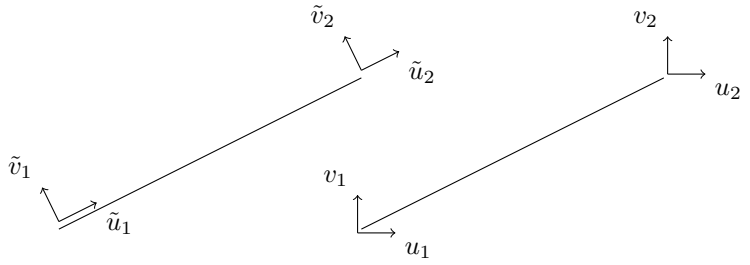


Figure 3.4.5: Local and global displacements

To go from global displacements to local, use following transformation matrix:

$$\mathbf{T}_{lg} = \begin{bmatrix} e_{xx} & e_{xy} \\ e_{yx} & e_{yy} \end{bmatrix} \tag{3.4.61}$$

$$\mathbf{e}_x = \begin{bmatrix} c \\ s \end{bmatrix} \tag{3.4.62}$$

$$\mathbf{e}_y = \begin{bmatrix} -s \\ c \end{bmatrix} \tag{3.4.63}$$

where $c = \cos\theta_i$ and $s = \sin\theta_i$.

The local displacements is obtained by following equation:

$$\mathbf{v}_b = \mathbf{T}_{lg}\mathbf{v}^T \tag{3.4.64}$$

$$\mathbf{v}_b^{'T} = \begin{bmatrix} u_1^{'} & v_1^{'} & u_2^{'} & v_2^{'} & u_3^{'} & v_3^{'} \end{bmatrix} \tag{3.4.65}$$

g with loacal displacement

$$\varepsilon_{NG} = g_{1loc} + \frac{1}{2}g_{1loc}^2 + \frac{1}{2}g_{2loc}^2 \tag{3.4.66}$$

A strain expression is developed for each bar of the triangle and gathered in the following strain vector

$$\varepsilon_{NG} = \begin{bmatrix} \varepsilon_{Ga} \\ \varepsilon_{Gb} \\ \varepsilon_{Gc} \end{bmatrix} \tag{3.4.67}$$

$$\varepsilon_C = \mathbf{T}_{\varepsilon NC}\varepsilon_N \tag{3.4.68}$$

First derivative of each element is gathered in a **B**-matrix, one matrix for each element:

$$\mathbf{B}_{\varepsilon g} = \begin{bmatrix} 1 + g_1 & g_2 \end{bmatrix} \tag{3.4.69}$$

A new **B**-matrix, for g in local system from displacements in local:

$$\mathbf{B}_{g\mathbf{v}_l} = \begin{bmatrix} -\frac{1}{L_0} & 0 & \frac{1}{L_0} & 0 \\ 0 & -\frac{1}{L_0} & 0 & \frac{1}{L_0} \end{bmatrix} \tag{3.4.70}$$

B-matrix, for g in local system from displacements in global G:

$$\mathbf{B}_{g\mathbf{v}_G} = \mathbf{B}_{g\mathbf{v}_l}\mathbf{T}_{lG} \tag{3.4.71}$$

$$\mathbf{T}_{lG} = \begin{bmatrix} C & S & 0 & 0 \\ -S & C & 0 & 0 \\ 0 & 0 & C & S \\ 0 & 0 & -S & C \end{bmatrix} \tag{3.4.72}$$

Compute the natural strain-displacement matrix for ANS element:

$$\mathbf{B}_{nv} = \mathbf{B}_{\varepsilon g}\mathbf{B}_{g\mathbf{v}_G} = \frac{\partial\varepsilon}{\partial g}\frac{\partial g}{\partial \mathbf{v}} \tag{3.4.73}$$

$$\mathbf{B}_{nv} = \begin{bmatrix} \frac{\partial \varepsilon_1}{\partial u_1} & \frac{\partial \varepsilon_1}{\partial v_1} & \frac{\partial \varepsilon_1}{\partial u_2} & \frac{\partial \varepsilon_1}{\partial v_2} & 0 & 0 \\ 0 & 0 & \frac{\partial \varepsilon_2}{\partial u_2} & \frac{\partial \varepsilon_2}{\partial v_2} & \frac{\partial \varepsilon_2}{\partial u_3} & \frac{\partial \varepsilon_2}{\partial v_3} \\ \frac{\partial \varepsilon_3}{\partial u_1} & \frac{\partial \varepsilon_3}{\partial v_1} & 0 & 0 & \frac{\partial \varepsilon_3}{\partial u_3} & \frac{\partial \varepsilon_3}{\partial v_3} \end{bmatrix} \tag{3.4.74}$$

To go from natural strain-displacement to Cartesian strain displacement matrix for the ANS element, need to perform the following equation:

$$\mathbf{B}_{cv} = \mathbf{T}_{cvnv}^{-1} \mathbf{B}_{nv} \tag{3.4.75}$$

4 Numerical generated stiffness matrix

A numerical test can be executed to generate the stiffness matrix. This method can be used to verify the results of the tangent stiffness matrix. The stiffness matrix is expressed by Equation (4.0.1).

$$\mathbf{K}_{ij} = \frac{\partial \mathbf{f}_i}{\partial \mathbf{v}_j} \tag{4.0.1}$$

In this method, a small variation Δ is added to the displacement vector, $\mathbf{v}_j$ at each node. An internal force, $\mathbf{f}_+$ where the variation is added, and an internal force $\mathbf{f}_-$ where the variation is subtracted needs to be established. The forces are recalculated for each displacement added to the nodes of the element.

$$\mathbf{f}_+ = \mathbf{f}_i(\mathbf{v} + \Delta \mathbf{v}_j) \tag{4.0.2}$$

$$\mathbf{f}_- = \mathbf{f}_i(\mathbf{v} - \Delta \mathbf{v}_j) \tag{4.0.3}$$

The stiffness matrix is generated by performing Equation (4.0.4) for each variation of the displacement. Where the force vector represents the rows of the matrix, and the displacements represents the columns.

$$\mathbf{K}_{ij} = \frac{\mathbf{f}_+ - \mathbf{f}_-}{2\Delta \mathbf{v}_j} \tag{4.0.4}$$

The matrix obtained from this test will be compared with the tangent stiffness matrix for each strain equation in SH for both traditional and ANS formulation.

5 Numerical Method

This section will include the mathematical aspects of the implementation of a simple nonlinear model to obtain the numerical results. To accomplish this, open-source software such as Gmsh and ParaView will be utilized. Gmsh is a software that generates mesh for three-dimensional finite element problems. [17] ParaView is a data analysis and visualization application. [18]

The method used to obtain the numerical results is based on the method Nygård used in his Master thesis [19] and will follow four steps by use of open-source software:

1. Gmsh
2. MeshIO
3. FEM solver
4. ParaView

Gmsh will be utilized to define the geometry and to generate the mesh for the model. MeshIO is a Python module for mesh Input/Output which is utilized for data formatting. The implementation of SH strain and a nonlinear load control solver will be introduced as a Python code, while the result obtained from the Python solver will be visualised in ParaView.

5.1 Gmsh

Gmsh is a three-dimensional finite element mesh generator. It is a meshing tool with parametric input and advanced visualization capabilities. The Gmsh Application Programming Interface (API) can be integrated to function with the preferred application. For the FEM problem, the Gmsh API will be following Python API and therefore be integrated into the numerical code using Python programming language. [20]

There are two default CAD kernels in Gmsh used to create geometry. One is called "OpenCASCADE" kernel and the other is "Built-in" kernel. The "OpenCASCADE" kernel will be used for this problem, to allow for scripting in Python. This is useful when defining the geometry of the model. Defining the *SetFactory* to "OpenCASCADE" is the first step when creating a model numerically or directly in Gmsh.

The initial step for solving a FEM problem is to define the geometry. The geometry of the cantilever will be defined numerically in a ".geo" file. A geometry created by Gmsh is defined by use of Boundary Representation. [20] A surface is created by lines, which are created by points. To create the geometry it is consequently necessary to start with the points, in this case representing the corners of the 2D-cantilever.

To create the geometry of the cantilever, the length in x-direction and y-direction is established as *dx, dy*. The corners framing the cantilever will be arranged as points. This is necessary because the lines framing the total cantilever need to be established by points. The points are identified with a "tag". This is their identification number displayed in the parenthesis which make them easier to obtain when creating the lines.

```
1    SetFactory("OpenCASCADE");
2    dx = 1000.0;
3    dy = 20.0;
4
5    Point(1) = {0.0,0.0,0.0};
6    Point(2) = { dx,0.0,0.0};
7    Point(3) = { dx, dy,0.0};
8    Point(4) = {0.0, dy,0.0};
```

After establishing the corners of the 2D cantilever, the lines framing the geometry can be created. These have a tag-number similar to the points. *Line(1)* represent a line connecting *Point(1)* and *Point(2)*.

```
    Line(1) = {1,2};
    Line(2) = {2,3};
    Line(3) = {3,4};
    Line(4) = {4,1};
```

When all lines are established, a *Curve Loop* connecting the four lines into a 2D entity can be specified. This is used to create the plane surface of the geometry, as the code below indicate.

```
    Curve Loop(1)    = {1,2,3,4};
    Plane Surface(1) = {1};
```

To establish where on the cantilever the load and boundary conditions shall be applied, *Physical Curves* can be created. They are not numbered like the points and lines were, but given strings as labels. This will later be referred to as *setname* when the boundary conditions will be applied in the Python code explained in Section 5.3. The physical curves are placed to lines, meaning *Physical Curve("fixedEnd")* is referring to the line where the fixed boundary condition will be applied, which for this case is line 4. The boundary conditions will be applied in the main Python code. The physical surface is defined from *Plane Surface(1)*, establishing a physical surface. This will represent the entire model.

```
    Physical Curve("fixedEnd") = {4};
    Physical Curve("LoadX")    = {2};
    Physical Curve("LoadY")    = {2};
    Physical Surface("domain") = {1};
```

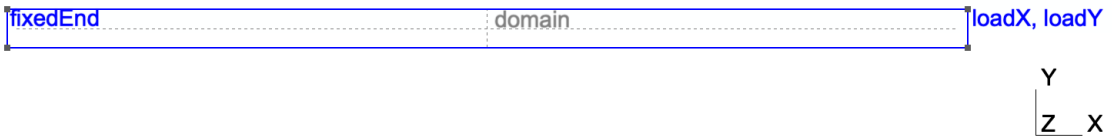


Figure 5.1.1: Gmsh cantilever model

After the entire model is fully created in the ".geo"-file, the file can be opened in the Gmsh program. In Figure 5.1.1, the model with the physical curves and surface is displayed. The mesh will be generated here. The *Global mesh size factor* is set to 0.3 for the model in Figure 5.1.2.

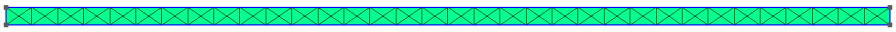


Figure 5.1.2: Meshed instance in Gmsh

After the mesh is created in Gmsh, the mesh data can be saved to a file. The saved file will be created as a ".msh"-file, and will be imported in the Python solver.

5.2 MeshIO

MeshIO is a Python module installed for direct use in coding. This module allows for input and output for mesh values, and will be valuable when performing data formatting of the mesh.

In the generated mesh file from Gmsh, MeshIO can be used to transform the mesh file into a data structure that a programming language, such as Python can read. An example of how MeshIO is used in the Python solver to read the mesh file:

```
import meshio
    # Reading mesh data
    msh = meshio.read(filename=meshfile)
```

The "msh" referred to here, is later used to connect the meshed geometry to the wanted output in ParaView, which for this problem is *Strain* and *Displacement*. This is further explained in Section 5.4. The ".msh"-file is often a text file consisting of data describing the formatting of the element type and size of mesh. This data is converted into a Python mesh object by MeshIO. This mesh object will contain all the information about the mesh. In MeshIO, elements are described as cells and the nodes as points. The MeshIO object is structured with the following data information:

- Cell data and sets
- Point data and sets
- Geometrical and psychical data

The use of cell and point data to extract displacement and strain results will be further explained in Section 5.4. To extract and use the information from the mesh object it is necessary to construct some particular functions in the Python solver. The *extract_set* function is created to extract and hold information about a given set of data, i.e. the collection of all nodes. This function returns the element connectivity array of a set. This set is critical when calculating the system stiffness matrix and for applying the loads and boundary conditions. From the data sets, the element coordinates and nodal coordinates can be extracted through the *extract_element_coords* and *extract_nodal_coords* functions. This information is essential when computing the system stiffness matrix and strain of the element. Further, the function *extract_dof_indx* is necessary to establish for converting the point data into correct degree of freedom indices. The reviewed functions are included in Appendix B.4.

5.3 FEM solver

The FEM solver, uses a nonlinear load control to perform a solution. The solution to the nonlinear geometrically analysis is based on the work of Haugen's PhD.[21] The method is solved by the following structure:

1. Define element-type and problem with the appropriate number of degree of freedom (DOF)
2. Read mesh data and select number of steps and iterations
3. Assemble the tangent stiffness matrix, $\mathbf{K}_t$, for the system
4. Establish the load factor ψ and add external load $\mathbf{P}(\psi)$ to the problem
5. Calculate the residual force, $\mathbf{r}(\psi, \mathbf{v})$
6. Solve matrix equation $\mathbf{Kv} = \mathbf{f}$
7. Update global displacement state $\hat{\mathbf{v}}$
8. Solve Seth-Hill Family strain $\boldsymbol{\varepsilon}$

A nonlinear load control algorithm is used to compute a solution to the geometrically structured problem. Step three to seven are performed in a loop for nonlinear problems. This algorithm allows for drift errors, causes the computed solutions to not drift away from the equilibrium path. Newton's method is used as the iterative solution, this method allows for large increments. A geometrically nonlinear structure often reaches a maximum load level, where the structure would not handle increases in load before it reaches a significant change in the geometry. These limits are often described as *critical points*, and can be characterized by a singular tangent stiffness matrix. [21]

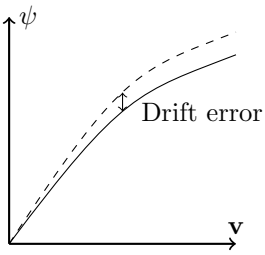


Figure 5.3.1: Equilibrium path

Figure 5.3.1 illustrates the numerically trace of the equilibrium path, by an incremental method using the force residual, $\mathbf{r}$, to reach the equilibrium state.

Before performing a solution, it is necessary to determine which element type the algorithm should use. The solver allows for two element types:

1. Traditional strain
2. Assumed Natural Strain (ANS)

The theory and implementation of the two element types are described in Section 3.3 and Section 3.4.

Considering our problem, a two-dimensional cantilever, with a fixed end and subjected to shear load in the other end, the degree of freedom is set to two. The selection of DOF allows for solving a numerous of different geometrically problems and makes the solver applicable for 3D-problems as well.

The ".msh"-file generated in Gmsh is imported to the solver, using MeshIO, and a maximum number of steps and iteration is established for solving the nonlinear load control.

The tangent stiffness matrix, $\mathbf{K}_t$, for the system is developed by the function *assemble_k*, included in Appendix B.4. Together with the functions reviewed in Section 5.2, the function returns the system tangent stiffness matrix. The function is applicable for different *setnames*, which is in our case set to "domain" representing the entire geometry.

The tangent stiffness is defined from the change in internal forces with respect to the displacements. It is said to be consistent if it is the gradient of the internal forces with respect to the degrees of freedom. The importance of the tangent stiffness is to obtain convergence through an iterative solution algorithm. It is wanted to get the equilibrium path for the solution, not only convergence, which is strongly dependent on the tangent stiffness. [21]

The external load $\mathbf{P}$ is applied to the structure by the function *add_load*, included in Appendix B.4. This function makes it possible to apply load to a given set. For our problem, a load is applied to "LoadX" and "LoadY", corresponding to a load in both x- and y-direction. The load is dependent on the load factor ψ. The load applied is given from the calculations made in Section 6.2. The load factor is determined by Equation (5.3.1), where i is the step number during the iterations and n represents the total number of steps.

$$\psi = \frac{(i+1)}{n} \tag{5.3.1}$$

A requirement for a nonlinear geometrically analysis is that the finite element and the global nodes are in equilibrium. For this solution, the force residual $\mathbf{r}$, is established to achieve the correct equilibrium path. The requirement for the solution is that the finite element internal force vector $\mathbf{f}$, is in a self-equilibrium state with respect to the deformed element geometry. [21]

$$\mathbf{r}(\psi, \mathbf{v}) = \mathbf{P}(\psi) - \mathbf{f}(\mathbf{v}) = \mathbf{0} \tag{5.3.2}$$

The force residual is determined by subtracting the internal force from the external force, $\mathbf{P}$, as in Equation (5.3.2). Where the internal force $\mathbf{f}$ is dependent on the current displacement of the nodes $\mathbf{v}$, and the external force $\mathbf{P}$ is a function of the loading parameter ψ. The global equilibrium is achieved when the force residual equals zero. [21]

$$\mathbf{K}\Delta\mathbf{v} = \mathbf{r}(\psi, \mathbf{v}) \tag{5.3.3}$$

After establishing the force residual, the matrix equation in Equation (5.3.3) could be solved for the incremental displacement $\mathbf{v}$. Then the global displacement state is updated for each iteration of the solution until the force residual reaches equilibrium. The updating of the global displacement state can be written as:

$$\hat{\mathbf{v}} = \hat{\mathbf{v}} + \Delta\mathbf{v} \tag{5.3.4}$$

Where $\hat{\mathbf{v}}$ represents the current total displacement state for both element and global level. And $\Delta\mathbf{v}$ the incremental displacement vector for an element.

After the equilibrium state or the maximum number of iterations is reached, the SH strain can be calculated for the triangle element model. The strain is calculated as point strain, meaning for each node of the total number of elements. To eliminate duplication of strain results for elements sharing nodes, a function in python is established to avoid such error of the result. The function used to calculate the point strain is *calculate_point_strain*, and is attached in Appendix B.4. The SH strain is defined by the total displacement state $\hat{\mathbf{v}}$, determined from the nonlinear load control solver.

5.4 ParaView

ParaView is an open-source data-analysis and visualisation application. [18] The results from the Python solver is written to a ".vtu"-file. The ".vtu"-file can be opened in ParaView where a visual representation of the results can be studied. The displacements, strain of the elements and deformed geometry are presented in ParaView. To get a visible representation is desirable, as it is convenient to view the results in relation to another and to check if the results meet the expectations. To present the results from the Python solver, explained in Section 5.3, it is necessary to format the results into the correct data format. The results are formatted as follows:

```
    # Format results
    tot_disp = tot_disp.reshape((msh.points.shape[0], 2))
    disp = tot_disp[:, 0:2]
    disp_3D = np.zeros((disp.shape[0], 3))
    disp_3D[:,0:2] = disp[:,0:2]
```

After the data is formatted to the correct shape, the element and node data is extracted from the mesh object created by the MeshIO module in Section 5.2. The *Displacement* and *Strain* is added as point data, meaning the result is accessible at each node. The *point_data* containers is represented as Python dictionaries, consisting of result data arrays. The result is added by the following commands:

```
    msh.point_data['Displacements'] = disp_3D
    msh.point_data['Strain'] = point_strain
```

MeshIO is utilized to write the results to the ".vtu"-file for post-processing in ParaView. The command *meshio.write()* writes the results to a new ".vtu"-file for each step of the iteration. ParaView allows for opening all the files in one group, making the results retrieved from the iterations accessible in one model. The ".vtu"-file can be accessed by:

```
    # Write results to file for post-processing
    num_dofs.append(tot_disp.size)
    tip_defs.append(disp.max())
    meshio.write(filename=(destination / (file +  str(step+1) + '.vtu')), mesh=msh)
```

Opening the ".vtu"-file in ParaView, the model created in the ".geo"-file is accessed, and the model in Figure 5.4.1 is accessed.

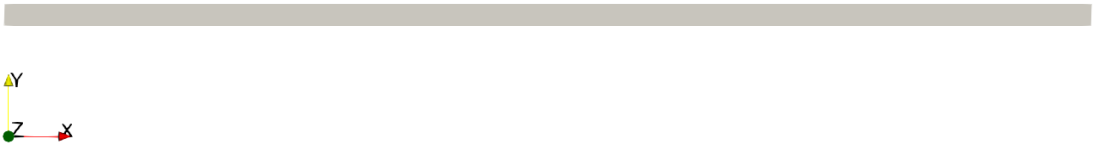
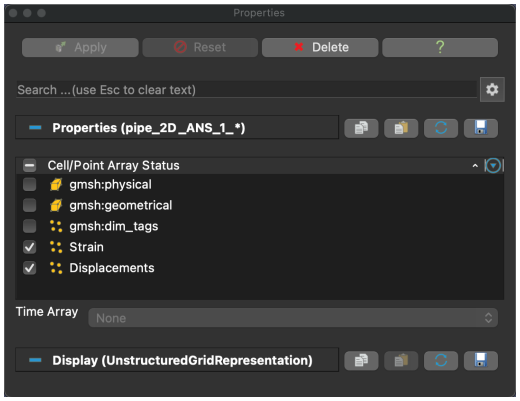
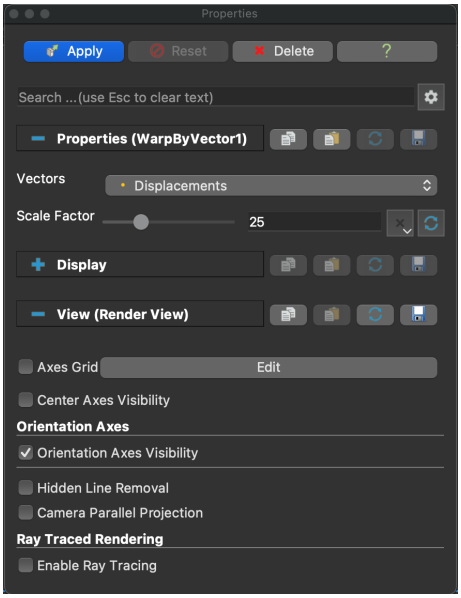


Figure 5.4.1: ParaView model

ParaView allows for different application of model properties to include in the model, such as cell and point data. The numerical results from the Python solver in Section 5.3, is available as model properties. In Figure 5.4.2a a selection of the different model properties that could be applied to the model is displayed. The wanted property is selected and applied to the model, and the result data is processed and visualised through colour-maps and filters added to the model.



(a) ParaView Model properties



(b) Warp Vector properties

Figure 5.4.2: ParaView Model properties

Through filters and model properties ParaView modifies the displayed cells and points. The filter *Warp by Vector* is favorable for displaying displacement and deformation in geometry. The filter is accessed by adding it to model properties, see Figure 5.4.2b. The filter creates a vector and applies it as warp to the nodes of the elements of the model. It also allows for different scale factors, to get a greater visualisation of deformation and displacement pattern. In Figure 5.4.3 model properties is added to the model with colour-maps and filters.

Figure 5.4.3: Meshed ParaView model with displayed strain

6 Experimental Method

Experimental work will be executed to investigate the fixed-end torsion problem. This will provide a better understanding of the Poynting effect as well as to give a better basis for the numerical model. To perform the experiment, it is necessary to establish a setup for a test rig. The concept of the test rig is further explained in Section 6.3.

Due to the size of the test rig the test specimens must be chopped to a size of maximum 200 cm. It is crucial that the pipe should not exceed yield, as it is wanted to perform the test in the elastic regime. In Section 6.2 the calculations of the maximum torque and twist angle are reviewed.

6.1 Materials used for the testing

The experimental investigation on Poynting effect will include pipes of different materials. In order to perform a more credible test, both isotropic and anisotropic materials should be included. An isotropic material has identical material properties and strength in all directions, while an anisotropic material has varying properties in the different orientations. This makes anisotropic materials a good choice of material in many cases as they often provide great strength in a favorable orientation. [22]

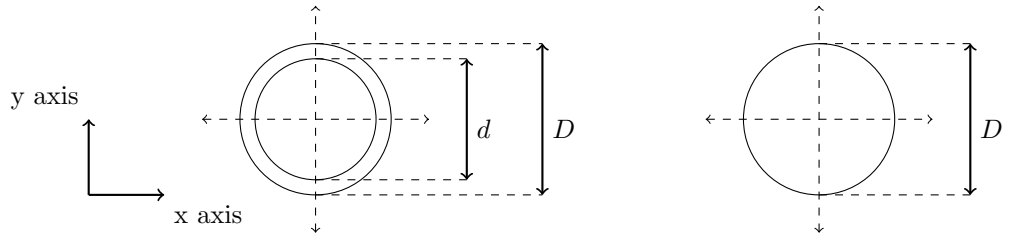
The selected materials for the pipe is represented in Table 2 and are materials that are commonly used for thin-walled pipes in the industry. Hardox400 is commonly used as solid cylinders. This material is included to provide more variation by use of a steel type with high yield strength. This type of configuration may be included in the experimental work as well.

Material	**Modulus of elasticity** E [GPa]	**Poisson's ratio** ν_b	**Yield strength** σ_y [MPa]
Stainless steel	200	0.29	215
Duplex steel	200	0.32	400
High-strength steel	200	0.29	700
Hardox400	210	0.30	1000
Aluminum 6063-T6	70	0.33	214
Glass Fiber	72	0.30	207
Carbon Fiber	238	0.28	207

Table 2: Material properties
[23–30]

6.2 Calculation of maximum torque moment

Before the test could be executed, it was important to calculate allowable moment of twist. This is to avoid twisting the pipe to yield during the experiment, considering that the material should stay in the elastic regime. The cross section of the pipe is illustrated in Figure 6.2.1.



(a) Hollow cross section (b) Solid cross section

Figure 6.2.1: Cross section

To calculate the maximum torque moment for the pipe, Equation (6.2.1) is used. [31]

$$T_{max} = \frac{\tau_{max} J}{R} = \frac{\tau_{max} \pi}{16D}[D^4 - d^4] \tag{6.2.1}$$

where J is the polar moment of inertia, τ_{max} is the maximum shear stress for the tube and R is the radius. In Equation (6.2.3)

As the Hardox400 comes as solid cylinder, the maximum torque will be calculated using the following equation:

$$T_{max} = \frac{\pi}{16}\tau_{max} D^3 \tag{6.2.2}$$

The polar moment of intertia for a pipe and for a solid sylinder is given by Equation (6.2.3) and Equation (6.2.4) respectively.

$$J_{pipe} = \frac{\pi(D^4 - d^4)}{32} \tag{6.2.3}$$

$$J_{solid} = \frac{\pi D^4}{32} \tag{6.2.4}$$

The angle of twist can be calculated by Equation (6.2.5). For our problem, a pipe under torsional loading, the angle of twist is the angle which fixed end of shaft rotates with respect to the free end. [32]

$$\theta = \frac{LT}{JG} = \frac{LT}{\frac{\pi(D^4 - d^4)}{32} G} \tag{6.2.5}$$

The modulus of rigidity can be calculated using Equation (6.2.6).

$$G = \frac{E}{2(1 + \nu)} \tag{6.2.6}$$

In our case, of pure shear stress, we can use the Von Mises criterion to find a relationship between the shear strength and tensile strength at yield. The von Mises criterion becomes:

$$\sigma = \frac{\sigma_y}{\sqrt{3}} \equiv \tau_{max} \tag{6.2.7}$$

Material	**Dimension** [mm]	**Torsion** [Nm]	**Angle** θ
304 Stainless steel	Ø19 x 1.20	69.768	0.337 rad — 19.30°
Duplex	Ø10.3 x 1.73	39.949	1.184 rad — 67.84°
High-strength steel	Ø42.4 x 2.0	1 978.37	0.492 rad — 28.19°
Hardox400	Ø40	7 255.197	0.715 rad — 40.97°
Aluminum	Ø6 x 1	2.713	3.13 rad — 179.34°
	Ø8 x 1	5.140	2.347 rad — 134.47°
	Ø10 x 1	8.343	1.878 rad — 107.60°
	Ø12 x 1	12.322	1.565 rad — 89.67°
Glass fiber	Ø11 x 4	26.111	1.246 rad — 71.39°
	Ø16 x 12	95.741	1.284 rad — 73.57°
Carbon fiber	Ø 4 x 3	1.496	0.643 rad — 36.84°
	Ø 5 x 3	2.858	0.514 rad — 29.45°

Table 3: Torque and angle of twist for the different material types of the pipe [24, 33–37]

6.3 Test rig

The setup of the test rig must be established to complete the experiment on Poynting effect. Numerous concepts and solutions of the test rig was considered during a brainstorming process. The final concept of the test rig was the composition that fulfilled the criteria of what the test rig should be dimensioned for:

- Test pipe shall be fixed in one end
- Test pipe shall be exposed for torque
- It shall allow for axial lengthening, or measure axial force
- The tube shall not be exposed to other forces than the torque - meaning no out-of-plane twisting or deflection

The laboratory at the university has a fastening structure available for use in this experiment. This structure consist of a load frame and a high-strength steel beam fastened to the frame. The beam allows for mounting of a coupling to fasten the test specimen, satisfying the criteria that the specimen should be fixed in one end. The setup of the load frame is illustrated in Figure 6.3.1. Further, it was necessary to establish and include components concerning the torque application and support of the test specimen. Section 6.3.4 will include the selection of components and the final concepts of the test rig is explained in Section 6.3.5.

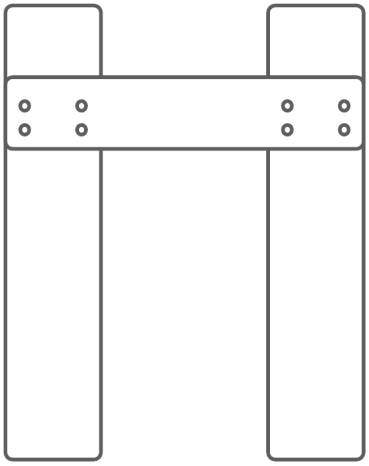


Figure 6.3.1: Load frame

6.3.1 Clamping devices

To fasten the test specimen, various clamping mechanism was considered. It is favorable to choose a fastening that would fit for cylindrical structures. The choice of clamping devices considered different types of chucks. These mechanisms are great for glass fiber and carbon fiber pipes, as it would not be necessary to glue or weld the test specimen to the clamping devices to get it fully clamped. The chucks also allows for easy fastening and change of test specimens.

(a) Lathe chuck [38]

(b) Drill chuck [39]

Figure 6.3.2: Chucks

A lathe chuck, illustrated in Figure 6.3.2a, could be used as the clamping devices for the fastening structure for the test specimen. The represented chuck consist of four separately adjustable jaws. It is commonly used for clamping of cylindrical and asymmetrical work pieces on turning lathes. The selected chuck can fasten test specimens with a maximum diameter of 45 mm. [38]

Another clamping method in consideration for this problem included to utilize a drill chuck, as seen in Figure 6.3.2b. It would be favorable to select a keyless drill chuck for easy fastening and removal of test specimen. These chucks comes in different design, making them applicable for various problems. [39]

Figure 6.3.3: Clamping device
[40]

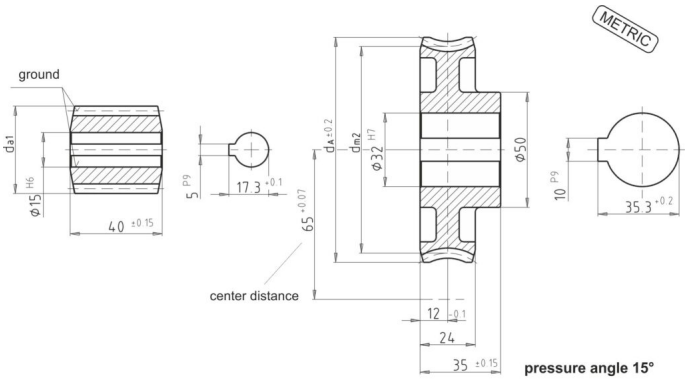
Figure 6.3.3 represents a possible set-up of the clamping device for a test machine. For this design, the test specimen would be clamped in both ends. It is applicable for cylindrical test specimens. This set up also allows for rotation and torque application at the free end of the test specimen.

6.3.2 Torque applicator

There are several methods to apply the torque on the specimen. The setup could include a rod to be connected to the test specimen as an extension of the test setup. The torque would be applied on the rod, which by a connection propagates the torsion to the test specimen. It would be beneficial to be able to apply the torsion gradually. Three suggested torque applications are included here, a worm drive, a socket wrench and a crank wheel.

The first suggested torque applicator is by use of a worm drive as illustrated in Figure 6.3.4a. The worm drive consist of a shaft with threads called a worm, and a worm wheel which will be rotated by rotation of the worm. [41] The rod could be fastened in the hole on the worm wheel. The torque would be applied by rotating the worm on top of the worm wheel. The size of the worm drive applicable for this problem is dependent on the rod size, and not the size of the test specimen. Though, the rod would rather be larger than smaller than the test specimen. A Worm Gear Sets A65 from Morat with inner diameter 32 mm would be a good choice.

(a) Worm drive



(b) Dimensions

Figure 6.3.4: Worm drive
[42]

Another application that could be used for torque application is a socket wrench, often called

ratchet as illustrated in Figure 6.3.5a. This could be placed at the end of the rod and allows to twist and release the rod easily. The benefits of a ratchet is the ability for it to be connected to the rod continuously. There is no need to remove the wrench from the rod and reposition it to be able to apply the torsion. A ratchet has a various selection of belonging sockets which come in different sizes. This makes the ratchet a good choice for torsion application as the socket can be chosen depending on the rod size, and a socket with size of approximately 32 mm could be a good choice for the current rod. The ratchet would be managed manually. The ratchet allows easy locking at a given rotation angle.

(a) Ratchet [43] (b) Crank wheel

Figure 6.3.5: Torque applicators

A crank wheel could be a good solution to apply the torque. The gears would allow for gradually application of the torque, as is common for the two previously discussed suggestions. The rod could be placed in the inner hole of the crank wheel, and it would be crucial to keep them fully fastened to avoid slippage when the torque is applied. A chain could be placed around the crank wheel and be connected to another smaller wheel where the rotation would be conducted. By use of a smaller wheel, less power is necessary to apply the torsion. It would be beneficial to fasten the additional wheel on another rod beside the structure, to make sure the chain is horizontal during the test. The crank wheel solution is illustrated in Figure 6.3.5b.

6.3.3 Measurement

The lengthening and stretch of the test specimen must be measurable. A measure of the change in length, would lead to a complex setup of test specimen and torque applicator. Considering that the setup must allow for change in length. To avoid such complexity, the stretch would be considered as the measured value, and two measuring methods was considered.

A load cell attached to the test specimen could be utilized as a measure equipment. For this fixed-end torsion problem, a S-type load cell would be the best fit. These types of load cells are mainly used for measuring tensile forces. The load cell consist of a spring element, that is elastically deformed under loading and recovers when the load is removed. The deformations or strain is then picked up by strain gauges that is installed on the spring elements and converts the data into an electrical output. For this case, the load cell would work as a force transducer, and the results is displayed in Newton. [44]

Figure 6.3.6: S-type load cell SN20
[44]

Another method to measure the strain and deformation is to use natural frequency as a measure method. A eigenfrequency analysis can determine the shape of the current mode. When a structure is vibrating at a certain frequency, the structure will deform into a corresponding shape, referred to as eigenmode. [45] The measurement could be conducted by measuring the eigenfrequencies before and after the applied torque. The change in frequency can determine how much tension the test specimen is exposed to, as well as the current shape. To measure the frequency a cell phone or other frequency measure tools could be used when the test specimen is "pluched" or hit to create a audible sound.

6.3.4 Components

A thrust ball bearing, single direction, is chosen as the connecting component between the test specimen and torque applicator. It is a critical component, considering it would not allow radial loading and out-of-plane twisting of test specimen. A thrust ball bearing with single direction is designed to handle axial load only. A 51106 thrust ball bearing from the SKF catalogue is chosen. [46] The dimensions and technical specifications of the bearing is displayed in Figure 6.3.7 and Table 4.

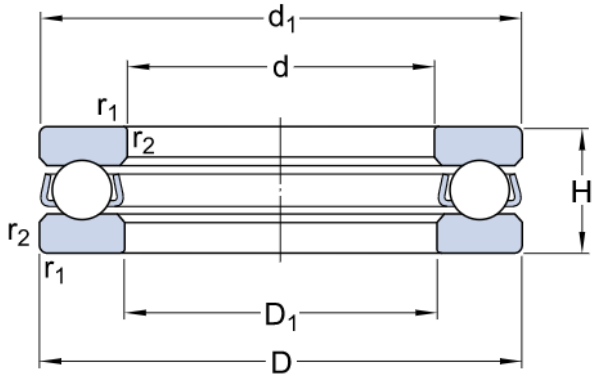


Figure 6.3.7: Thrust ball bearing
[46]

Dimensions	
d	$30mm$
D	$47mm$
H	$11mm$
d_1	$\approx 47mm$
D_1	$\approx 32mm$
$r_{1,2}$	$min.0.6mm$

Table 4: Dimensions of thrust bearing
[46]

The bearing is designed to accommodate the following loading parameters: [46]

- Dynamic load rating $C = 19kN$
- Static load rating $C_0 = 43kN$

Considering the dimensions and technical specification of the selected bearing and that the torque would be applied manually, one can assume that the bearing would be sufficiently dimensioned for this experiment. Further it was necessary to design and manufacture a bearing case and structure to couple the bearing to the test specimen and make it applicable to the torque problem.

To ensure the bearing case to stay rigid, a beam of high-steel is attached between the case and load frame. This would provide fastening of the test specimen and assure that it is not exposed to other forces than the applied torque. The beam would be fastened to the the load frame with bolts. It is critical that the test specimen is fastened enough to keep the material in stretch during testing.

The most ideal torque applicator for this problem is to use the worm drive, described in Section 6.3.2. This would allow to stop the rotation during testing, and ensure that it would stay in the set position. The set-up of the bearing casing and torque applicator is illustrated in Figure 6.3.8.

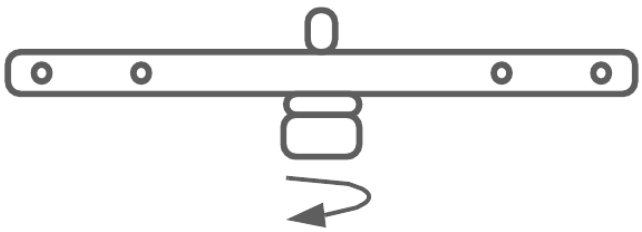


Figure 6.3.8: Bearing case and torque applicator connected

6.3.5 Concepts

The setup of the test rig could be separated into two main concepts. The difference between the to concepts is based on which measurement method is used. Common for the concepts is the bottom part consisting of the torque applicator and the fastening structure as explained in Section 6.3.4. The test specimen would be clamped in lathe chucks as this allows for easy change and use of different outer diameters of test specimen, and is consequently the better choice. The upper part of the test rig would be modified relative to the selected measurement.

The concept concerning eigenfrequency as measure method would consist of few components. A steel plate would be fixed to the upper beam and connected to the upper chuck on the clamping device. This stops the rotation of the specimen. The concept is illustrated in Figure 6.3.9.

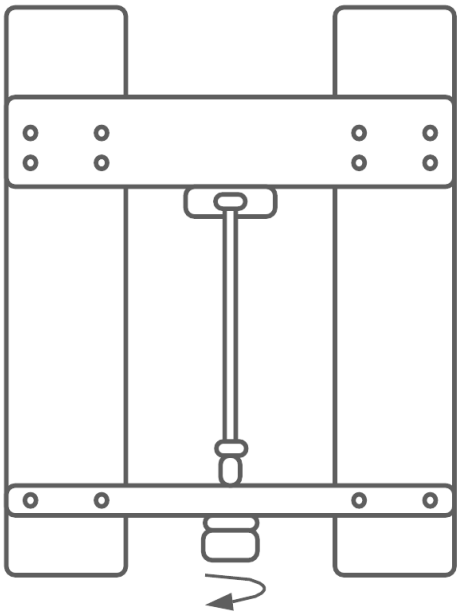


Figure 6.3.9: Total set up of test rig using eigenfrequency measurement method

The concept using a load cell to measure the stretch is illustrated in Figure 6.3.10. This concepts concern a more complex upper part of the test rig. The upper chuck would be connected to a rod for this concept. It is crucial that the rod should be threaded to be able to connect the load cell to the rod. An additional beam placed between the upper chuck and the load cell would stop the rotation of the rod. This is crucial as the load cell do not tolerate rotation or twisting. The middle beam consist of an opening allowing for the rod to move in axial direction, and two arms connected to the load frame on each side. This beam does not need to be in a high-strength material because it is not part of the support structure of the test rig.

The load cell would be connected to the upper beam by an additional rod. This could be fastened by a cap nut at the top of the beam. This beam represent the support of the test rig in combination with the lower beam and should be in a high strength material. The cap nut is fastened to apply stretch in the material of the specimen. Before the experiment can be executed the material is preloaded to a certain tension, measured by the load cell.

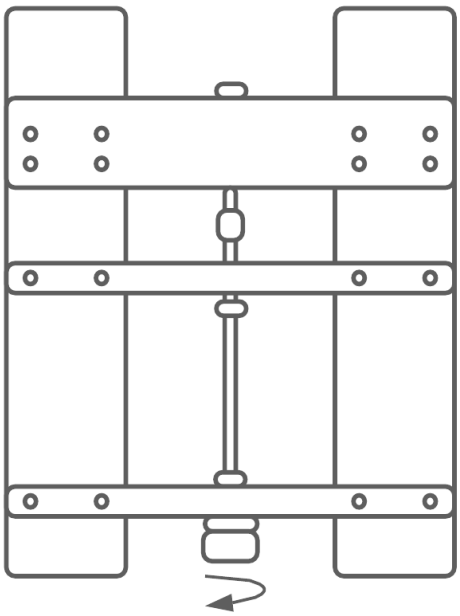


Figure 6.3.10: Total set up of test rig using load cell measurement method

7 Numerical Results

This section will include the numerical results obtained from a study on the variation of the SH strains for triangle element. In addition, the numerical results from the Python FEM solver using a nonlinear load control. The results from the FEM solver is based on a two-dimensional cantilever model subjected to shear load, illustrated in Figure 5.4.1. The results will include solutions of the two element types introduced in Section 3:

1. Traditional strain
2. Assumed Natural Strain (ANS)

7.1 Seth-Hill Family

From the theory in Section 3 for a triangle element using SH, a comparison of the SH strains were conducted in Python for one element as illustrated in Figure 3.2.1. A relation between the generalized strains is represented in Figure 7.1.1. The Python code for plotting the strain results is further explained and represented in Appendix B. The SH over the triangle element were plotted in a displacement range from -400 to 1000, where Equation (7.1.1) was used as the displacement of the element.

$$\mathbf{v}^T = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 & 0.5 & 1.0 \end{bmatrix} \tag{7.1.1}$$

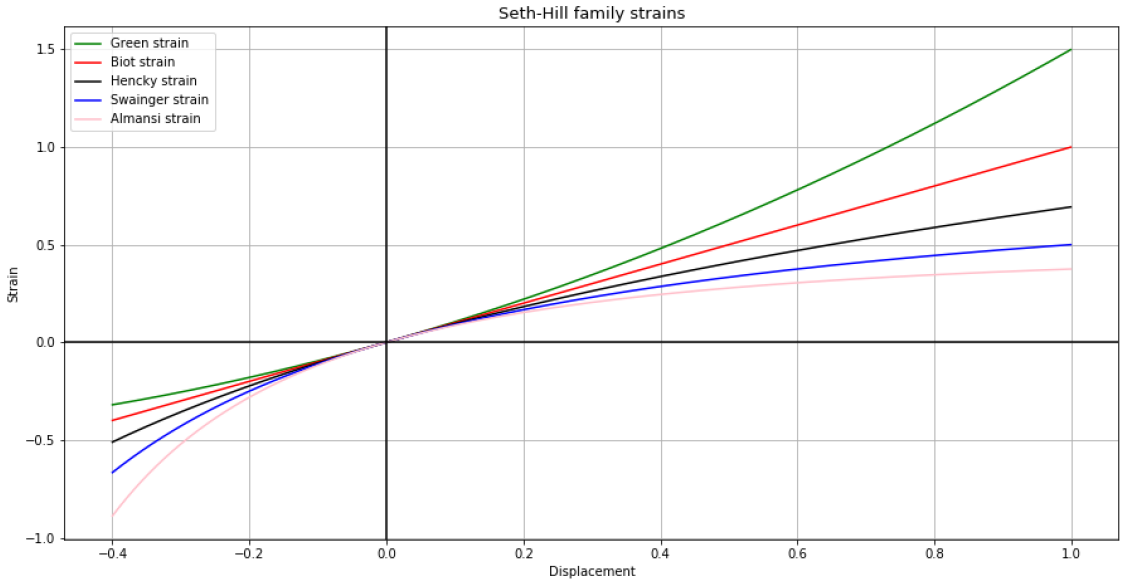


Figure 7.1.1: Plot with comparison of Seth-Hill Family strain

Reading from the plot in Figure 7.1.1 an observation is that the Engineering strain, with a measure index $m = 1$ represents a linearized strain. The Green strain computes the highest values for strain. This was predicted considering its measure index $m = 2$, which is the highest value of the measure index in the SH. Following the Hencky, Swainger and Almansi strain computes lower values of strain than the engineering strain respectively, with a measure index of $m = 0$, $m = -1$, and $m = -2$.

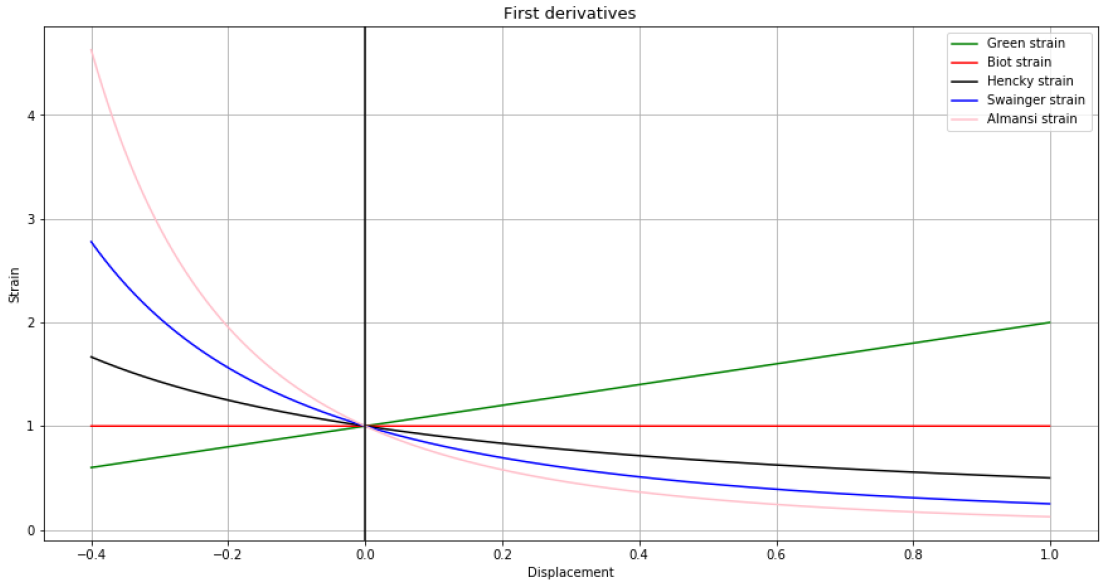


Figure 7.1.2: Plot with comparison of Seth-Hill family as first derivatives

In Figure 7.1.2 the SH strains are plotted as first derivatives as a function of strain. The first derivatives of the strain is computed in order to obtain an expression for the tangential stiffness of the element, by Equation (4.0.1). For this case, the engineering strain is defined by a constant value of 1. The Green strain represents a linearlized strain. While Hencky, Swainger and Almansi is visualized by a quadratic equation.

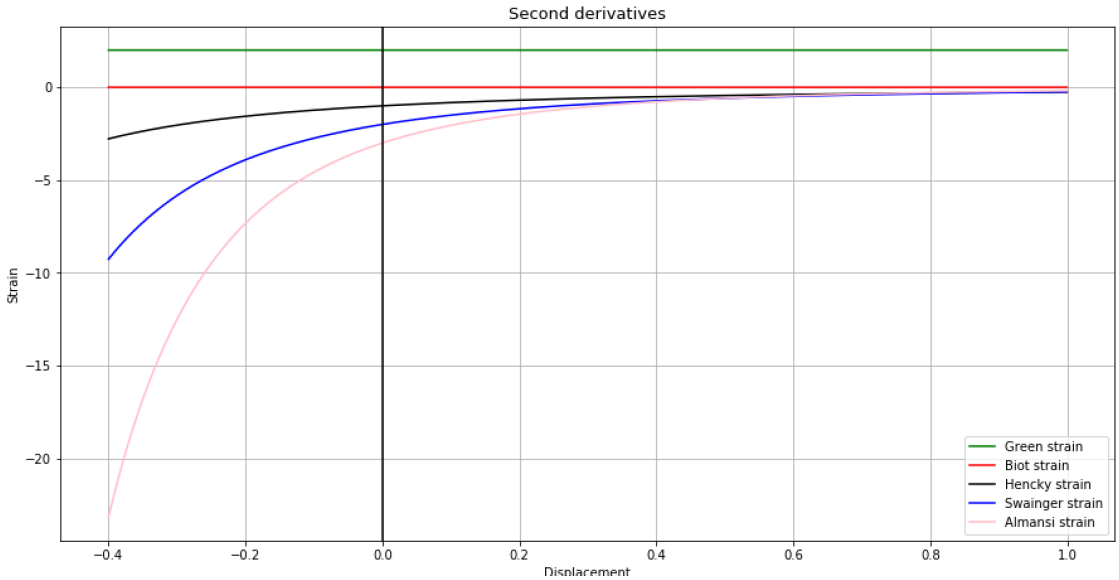


Figure 7.1.3: Plot with comparison of Seth-Hill Family as second derivatives

The relation between the second derivatives of SH strain is represented in Figure 7.1.3. The second derivatives is necessary to compute as it is a part of the second therm of Equation (3.4.30) and Equation (3.3.20), building the tangent stiffness matrix, $\mathbf{K}_t$. The second derivative of Biot and Green corresponds to a constant value of 0 and 1 respectively. While Hencky, Swainger and Almansi are represented by a quatratic expression.

7.2 Triangle element

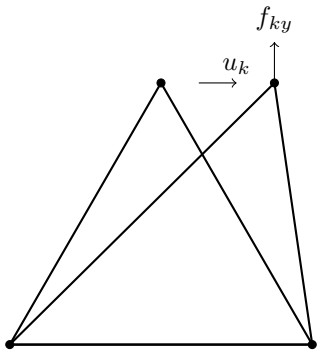


Figure 7.2.1: Triangle element with node displacement

The Poynting effect could be proven through study of a single triangle element. The test was performed using traditional formulation for Green strain. This was conducted by applying a displacement at one node, while the others are fixed, illustrated in Figure 7.2.1. The displacement was added and gradually increased through a given number of iterations. The applied displacement and topology used for this study is listed below:

```
ex = np.array([0.,1.,0.5])
ey = np.array([0.,0.,1.])

E  = 2.1e11
nu = 0.3

C = np.array([
        [ 1.0,  nu,  0.],
        [  nu, 1.0,  0.],
        [  0.,  0., (1.0-nu)/2.0]]) * E/(1.0-nu**2)

dispVecEpsX  = np.array([[0.0],[0.0],[0.0],[0.0],[0.1], [0.0]])
```

The results of the test is represented in Figure 7.2.2.

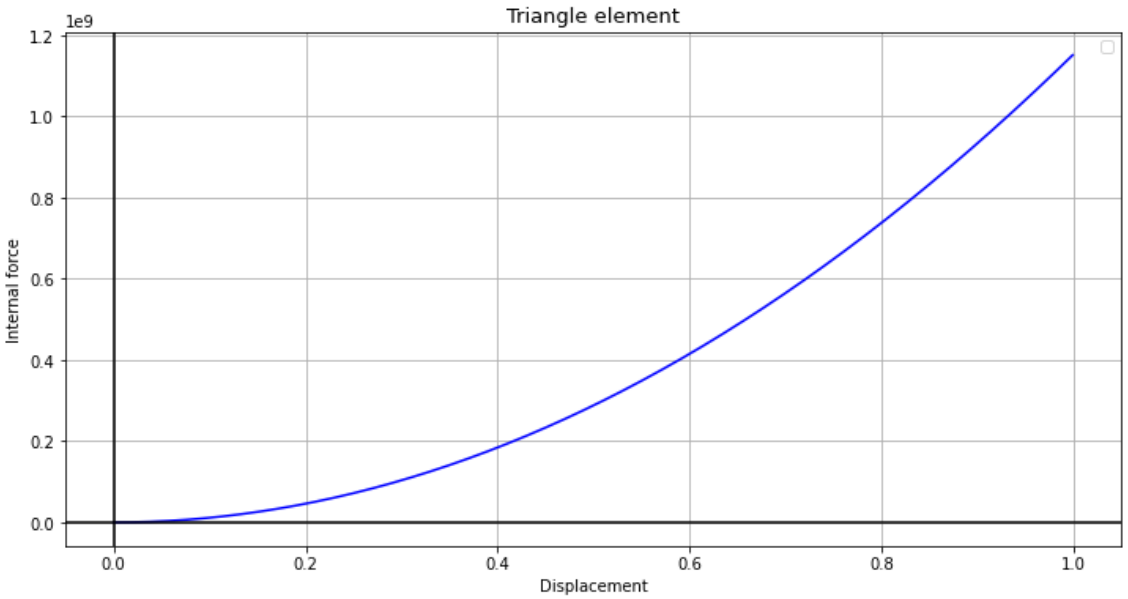


Figure 7.2.2: Plot of tension in the element

7.3 Python FEM solver

This section will include the numerical results obtained by the Python FEM solver in Section 5. The external force, **P**, is applied to the nodes of the free end. The force is applied iterative, with a load factor. The magnitude of the force depends on the choice of material and the maximum moment of twist calculated in Section 6.2. The material properties for the different materials used in the test is presented in Table 2. This method allows for a more accurate result and available results and behaviour of model through each iteration. The results from the FEA will included displacement and strain data. The tests are completed with different mesh sizes and the following parameters are used:

- Mesh size factor: $0.3, 0.2, 0, 1$
- Material properties of 304 Stainless Steel and Aluminum
- DOF= 2
- Number of steps, $n_{steps} = 10$
- Number of iterations, $n_{iter} = 5$

7.3.1 Traditional strain

To obtain the numerical solution using traditional formulation, the tangent stiffness for the actual SH-strain is used in the main Python FEM solver. A mesh is generated for the geometry using Gmsh. The mesh-file is selected in the solver. The material properties are inserted for 304 Stainless steel, Aluminum and Carbon Fiber in separate runs of the code, collected from Table 2 in Section 6.1. The applied force for the cantilever is 500N applied in x- and y-direction.

The solution is obtained for different mesh sizes using the same geometry and force. The results are obtained for the different SH-strain equations and are displayed in Table 5, Table 6 and Table 7 for mesh size 0.3, 0.2 and 0.1 respectively. An illustration of how the displacement and strain result affect the cantilever is displayed in Figure 7.3.1.

(a) Displacement (b) Strain

Figure 7.3.1: Results for Carbon Fiber using Green strain equation

Material	**Seth-Hill Family strain**				
	Green	Biot	Hencky	Swainger	Almansi
304 Stainless steel, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	2.4	0.12	0.14	0.15	0.16
Strain ε_y	2.4	0.071	0.053	0.0049	0.0040
Strain ε	3.4	0.19	0.24	0.013	0.013
Displacement X	2.9	120	130	140	150
Displacement Y	1700	7800	7800	7800	7900
Displacement	1900	7800	7800	7800	7900
Aluminum, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	22	0.34	0.34	0.04	0.37
Strain ε_y	22	0.22	0.22	0.067	0.091
Strain ε	31	0.54	0.54	0.58	0.67
Displacement X	3.4	340	340	510	530
Displacement Y	4500	23000	23000	23000	23000
Displacement	4600	23000	23000	23000	23000
Carbon Fiber, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	1.6	0.097	0.11	0.12	0.13
Strain ε_y	1.6	0.058	0.054	0.047	0.038
Strain ε	2.3	0.16	0.18	0.2	0.21
Displacement X	2.6	99	110	110	120
Displacement Y	1500	6500	6500	6500	6600
Displacement	1700	6500	6500	6500	6600

Table 5: Numerical results for traditional strain using mesh size 0.3

Material	**Seth-Hill Family strain**				
	Green	Biot	Hencky	Swainger	Almansi
304 Stainless steel, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	1.7	0.13	0.14	0.32	0.16
Strain ε_y	1.7	0.062	0.058	0.089	0.057
Strain ε	2.5	0.14	0.15	0.41	0.18
Displacement X	2.7	100	110	140	140
Displacement Y	1500	6700	6700	8300	6900
Displacement	1700	6700	6700	8300	6900
Aluminum, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	1.6	0.37	0.5	0.65	0.43
Strain ε_y	1.6	0.2	0.16	0.44	0.39
Strain ε_x	2.2	0.42	0.6	1.5	70
Displacement X	3.9	290	360	540	640
Displacement Y	3900	1900	2000	2100	1800
Displacement	4000	1900	2000	2100	1800
Carbon Fiber, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	1.2	0.11	0.12	0.24	0.13
Strain ε_y	1.2	0.05	0.048	0.1	0.06
Strain ε_x	1.7	0.12	0.13	0.26	0.26
Displacement X	2.6	85	91	150	120
Displacement Y	1400	5600	5600	6300	6200
Displacement	1500	5600	5600	6300	6200

Table 6: Numerical results for traditional strain using mesh size 0.2

Material	**Seth-Hill Family strain**				
	Green	Biot	Hencky	Swainger	Almansi
304 Stainless steel, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	0.42	0.075	0.079	0.072	0.092
Strain ε_y	0.42	0.031	0.029	0.02	0.027
Strain ε	0.6	0.083	0.085	0.077	0.1
Displacement X	2.3	57	60	63	73
Displacement Y	1100	3700	3700	3700	4200
Displacement	1200	3700	3700	3700	4200
Aluminum, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	4.3	0.21	0.24	2.3	0.45
Strain ε_y	4.3	0.1	0.081	6.8	0.42
Strain ε	6.1	0.24	0.26	9.6	9700
Displacement X	3.1	150	180	220	2900
Displacement Y	220	1000	1000	10000	11000
Displacement	230	1000	1000	10000	11000
Carbon Fiber, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	0.26	0.063	0.066	0.051	0.46
Strain ε_y	0.26	0.025	0.024	0.018	0.34
Strain ε	0.37	0.07	0.071	0.055	11000
Displacement X	2.1	48	51	56	260
Displacement Y	970	3200	3200	3000	3800
Displacement	1100	3200	3200	3000	3900

Table 7: Numerical results for traditional strain using mesh size 0.1

7.3.2 Assumed Natural strain

The test results are obtained by different mesh sizes and materials. The SH strain is solved using the theory from Section 3.4. The results from the test using different mesh sizes is represented in Table 8, Table 9 and Table 10.

(a) Displacement (b) Strain

Figure 7.3.2: Results for Stainless Steel using Green strain equation

Material	**Seth-Hill Family strain**				
	Green	Biot	Hencky	Swainger	Almansi
304 Stainless steel, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	2.4	0.022	7.8	1	0.64
Strain ε_y	2.4	0.012	7.2	1.2	0.61
Strain ε	3.4	0.036	9.7	1.5	0.75
Displacement X	2.9	1.8	73000	13000	3.1×10^{30}
Displacement Y	170	600	280000	38000	3.0×10^{38}
Displacement	190	630	28000	43000	3.1×10^{38}
Aluminum, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	22	0.043	8.9	1.0	0.6
Strain ε_y	22	0.02	7.7	1.8	0.5
Strain ε	31	0.063	10	1.8	0.71
Displacement X	3.1	2.6	2.3×10^5	9.2×10^{15}	4.7×10^{148}
Displacement Y	4500	650	4.6×10^5	1×10^{18}	3.7×10^{183}
Displacement	4600	690	5.1×10^5	5.6×10^{19}	1.9×10^{183}
Carbon Fiber, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	1.6	0.02	8.0	1.0	0.61
Strain ε_y	1.6	0.011	7.7	1.0	0.51
Strain ε	2.3	0.033	10	1.4	0.71
Displacement X	2.6	1.7	6400	1.7×10^5	3.4×10^{19}
Displacement Y	1500	590	23000	5.1×10^4	8.3×10^{15}
Displacement	1700	620	24000	3.4×10^5	3.4×10^{19}

Table 8: Numerical results for assumed natural strain using mesh size 0.3

Material	**Seth-Hill Family strain**				
	Green	Biot	Hencky	Swainger	Almansi
304 Stainless steel, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	1.7	0.024	6.0	1.4	0.51
Strain ε_y	1.7	0.011	6.1	1.0	0.52
Strain ε	2.5	0.027	8.5	1.5	0.71
Displacement X	2.7	1.7	4.2×10^4	1.3×10^4	5.8×10^{18}
Displacement Y	1500	590	2.6×10^5	4.2×10^4	2.6×10^{16}
Displacement	1700	620	2.7×10^5	4.4×10^4	7.9×10^{18}
Aluminum, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	1.6	0.038	7.9	1.0	0.51
Strain ε_y	1.6	0.019	6.9	1.0	0.5
Strain ε	2.2	0.047	9.4	1.4	0.71
Displacement X	3.9	2.2	3.6×10^4	6.1×10^{10}	0
Displacement Y	3900	640	5.8×10^5	2.7×10^{10}	5.6×10^{153}
Displacement	4000	680	5.8×10^5	6.1×10^{11}	6.2×10^{153}
Carbon Fiber, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	1.2	0.022	6.5	6.5	0.75
Strain ε_y	1.2	0.0096	6.7	1.0	0.5
Strain ε	1.7	0.025	9.3	1.4	0.79
Displacement X	2.6	1.7	4.6×10^4	1×10^4	9.3×10^6
Displacement Y	1400	580	2.5×10^5	3.2×10^4	1×10^9
Displacement	1500	610	2.6×10^5	3.3×10^4	1.1×10^9

Table 9: Numerical results for assumed natural strain using mesh size 0.2

Material	**Seth-Hill Family strain**				
	Green	Biot	Hencky	Swainger	Almansi
304 Stainless steel, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	0.42	0.02	10	1	0.51
Strain ε_y	0.42	0.0077	9.2	1	0.59
Strain ε	0.6	0.022	14	27	0.97
Displacement X	2.3	1.5	3.1×10^5	1.2×10^4	1.2×10^{118}
Displacement Y	1100	550	5.2×10^5	2.9×10^4	1.2×10^{117}
Displacement	1200	580	5.9×10^5	3.2×10^4	1.1×10^{122}
Aluminum, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	4.3	0.034	8.3	1.0	0.68
Strain ε_y	4.3	0.015	9.4	1.0	0.59
Strain ε	6.1	0.038	12	2.3	0.73
Displacement X	3.1	1.9	1.6×10^5	1.2×10^{18}	3.9×10^{101}
Displacement Y	2200	620	1.7×10^5	1.4×10^{18}	5.2×10^{113}
Displacement	2300	650	2.0×10^5	1.9×10^{18}	5.3×10^{113}
Carbon Fiber, $\mathbf{F}_x = \mathbf{F}_y = 500$N					
Strain ε_x	0.25	0.092	9.3	1.0	0.6
Strain ε_y	0.25	0.0071	9.0	1.1	0.89
Strain ε	0.36	0.0021	13	2.8	1.2
Displacement X	2.1	1.4	4.9×10^4	2.8×10^9	9.9×10^{17}
Displacement Y	970	530	1.9×10^5	1.3×10^9	4.0×10^{17}
Displacement	1100	560	1.9×10^5	3.1×10^9	5.7×10^{21}

Table 10: Numerical results for assumed natural strain using mesh size 0.1

7.3.3 Cantilever

To further study the displacement and strain, a cantilever with different geometry was considered. This cantilever has same length as the one described in Section 5.1. The height is increased, resulting in a more robust cantilever. The mesh size 0.3 was used for this model and the meshed geometry is displayed in Figure 7.3.3.

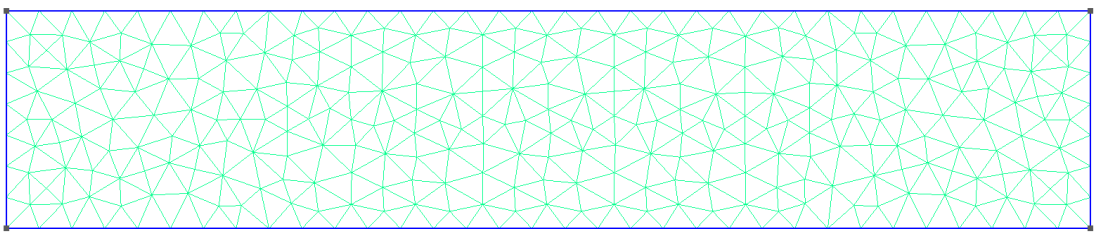


Figure 7.3.3: Meshed cantilever

The method described in Section 5 is followed for this cantilever. The load is increased to 1000N

and applied in y-direction. Number of steps and iterations are increased to be able to run the solver longer to obtain convergence. The strain result for the cantilever using Green strain equation is displayed in Figure 7.3.4 and plotted in Figure 7.3.5. The numerical results are presented in Table 11 and Table 12 for traditional and ANS formulation respectively.

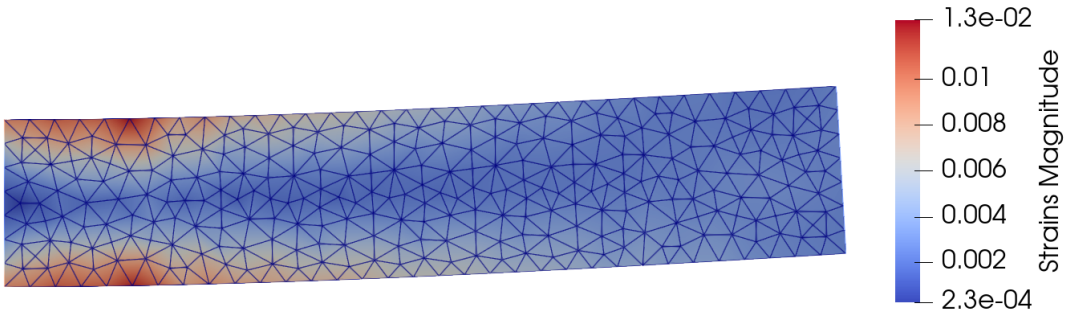


Figure 7.3.4: Strain result for cantilever using Green strain

Material	**Seth-Hill Family strain**				
	Green	Biot	Hencky	Swainger	Almansi
304 Stainless steel, **F** = 1000N					
Strain ε_x	0.012	0.011	0.011	0.011	0.011
Strain ε_y	0.0048	0.0046	0.0046	0.0046	0.0046
Strain ε	0.013	0.013	0.013	0.013	0.013
Displacement X	5.8	5.8	5.8	5.8	5.8
Displacement Y	40	40	40	40	40
Displacement	40	40	40	40	40

Table 11: Numerical results for traditional strain using for updated cantilever geometry

Material	**Seth-Hill Family strain**				
	Green	Biot	Hencky	Swainger	Almansi
304 Stainless steel, **F** = 1000N					
Strain ε_x	0.0045	0.0044	0.0044	0.0044	0.0044
Strain ε_y	0.0055	0.0055	0.0054	0.0054	0.0054
Strain ε	0.0085	0.0086	0.0086	0.0087	0.0087
Displacement X	5.8	5.8	5.8	5.8	5.8
Displacement Y	40	40	40	40	40
Displacement	40	40	40	40	40

Table 12: Numerical results for assumed natural strain using for updated cantilever geometry

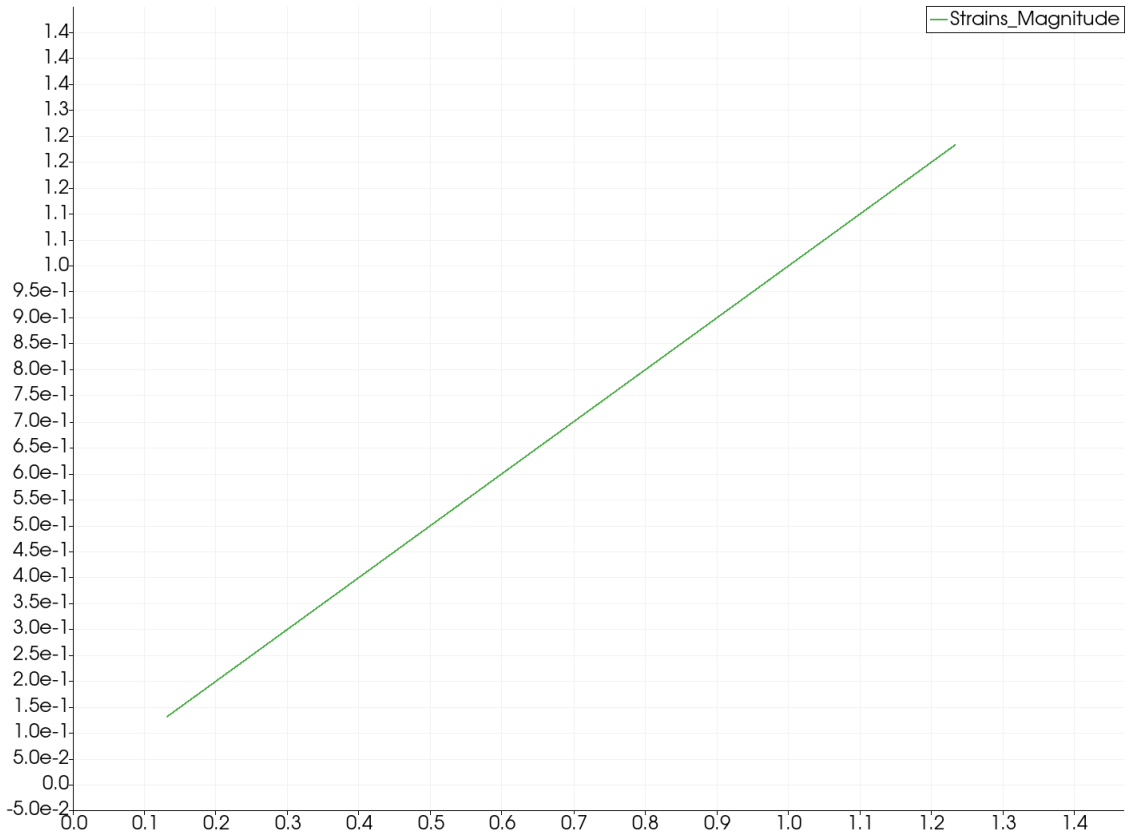


Figure 7.3.5: Plot of strain result for cantilever with load 1000 N

8 Discussion

The study of the triangle element involved forcing displacement in one node while the remaining were fixed. By forcing the displacement in a node the element would be deformed into a larger or shorter element depending on the applied displacement. The change in geometry anticipates the behavior of the element and result in a measurable tension. The tension is plotted in Figure 7.2.2 and indicate increasing value of the internal force with respect to the displacement of the node. The measured force is the vertical force in the free node. The result display increase in tension and proves the Poynting effect to be real.

Prior solving of the numerical method, testing and verification concerning the stiffness matrix of the SH strains were conducted. The wanted outcome was to obtain identical tangent stiffness matrix for the element formulations. The verification of stiffness matrix for Green strain was successful, meaning both traditional and ANS are correctly implemented. The verification concerning the other SH strains were not achieved and requires further research and implementation. Possible factors that may have interrupted the results will be discussed additionally. Verification of stiffness matrix could also be conducted through a closer inspection of iterations concerning the remaining SH strains.

The displacement result was obtained through the FEM solver and vary for each SH strain. The number of iterations before convergence is accomplished varies according to the difficulty of the SH equation. This result in running the solver longer before the wanted strain value is obtained, subjecting the cantilever to higher displacement. Numerous tests were performed with different materials and mesh sizes to optimize and get a greater understanding of the results. The result present a distinctive difference for Aluminum where the values of displacement are extremely high. This could be explained due to the low Young's modulus compared to the other materials. The value difference of the displacement for the SH strains is distinctive in ANS formulation. Almansi indicate extreme values of displacement in this formulation and is struggling to obtain decent strain values. In traditional formulation solving Hencky strain cause difficulties, due to the mathematical aspect of the equation. Logarithm development of matrices should be further investigated.

The strain obtained from the FEM solver resulted in more dissimilarities concerning the traditional and ANS formulation. Green strain for traditional and ANS formulation achieved identical results, which demonstrate why it is widely used for geometrically nonlinear FEM. Green appeared to give the most accurate solution as predicted, and the results signify that this strain equation was fully developed and successfully implemented for each formulation. The strain depend on material properties and the results show this variation. They vary according to mesh size, though not drastically. The strain was generally low for each material with any mesh size. The strain obtained from the numerical FEM solver show the influence in material properties and how they affect the results.

The remaining strain equations in the SH family gave more variations between traditional and ANS formulation. The traditional formulation resulted in similar values for each SH strain, though with a distinctive difference to the results for Green strain. Similar results would initially be the wanted outcome of the solver, though the difference to Green strain indicate that the results are not correct. Accordingly, the structure of these strain equations are deficient. This result was anticipated as these equations are less used in FEM for nonlinear problems, and is the reason the ANS formulation was included for this fixed-end problem. Though, the numerical results for strain using ANS formulation indicate faults in this formulation as well. Hencky strain computes strain results deviating from the other strain equations, while the remaining SH strain equations achieves comparable results. These result indicate that an optimization would be necessary for this formulation.

A study concerning an updated geometry of the cantilever was performed in the FEM solver. This was conducted to examine how the geometry influenced the strain due to the result for the initial geometry. The study was completed using Stainless Steel with a bigger width of the cantilever making the model more resistant. In addition, the applied force was increased and only applied in one direction, resulting in a shear load. The results show identical displacement for both element formulations, thus the applied load was not sufficiently high. The load should be additionally

increased as the updated geometry result in a drastically more resistant model. The strain results for the cantilever are similar for all equations of SH strain, though deviate between the element formulations. As the implementation of SH in traditional formulation seem to have weak links, the results obtained from ANS formulation are more reliable. The strain value increase for the iterative solution as the plot in Figure 7.3.5 illustrate. The result indicate that the applied load was too low for this model, thus similar strain values within the SH equations are obtained.

Considering the SH strains could not be implemented correctly, a few disturbances and factors should be further investigated. For Almansi strain using ANS formulation the constitutive matrix $\mathbf{C}$ must be updated for each iteration. The matrix is dependent on the coordinates and should be implemented for the deformed geometry. These factors were taken into account and implemented, though the Almansi strain could still not provide a correct solution. For Green strain, these factors were not considered as it is dependent on a consistent constitutive matrix. A suggestion for the remaining SH strains would be a combination of a consistent and updated $\mathbf{C}$-matrix. Further research should include optimization of the transformations between the natural and Cartesian coordinates for ANS element.

Concerning the traditional formulation, several factors should be researched. Green strain is assumed to be implemented successfully, though the remaining SH strains need to be further investigated. The formulated equations require additional mathematical development and several aspects of matrix operation should be considered. A known deficiency for Hencky strain involve diagonalization and principal logarithms of matrices.

Due to unpredicted circumstances the experiment could not be conducted, consequently the results could not be obtained. The main motive for the experiment was to obtain results to compare with the numerical results, in order to investigate the Poynting effect. In addition, it was desired to study the strain behaviour for different materials with contrasting material properties while in the elastic region. Two concepts were presented and the ideal concept would be using the load cell as measurement tool. The load cell would likely give the most accurate measurements, and be the most accessible measure device.

Use of the numerical method was valuable for this thesis. By creating a model and generating a mesh through Gmsh an adequate foundation of the problem was accomplished. Some obstacles occurred regarding the ".msh"-file, as it generated duplicate sets of elements. This may have interrupted the solved results. The MeshIO function was essential for solving the Python problem. It operates by collecting the mesh information constructed in Gmsh, and formatting it into accessible Python objects. This is a critical function for solving the FEA. The nonlinear load control allows for more accurate results and greater understanding of the behavior. Implementation of this iterative method proved to be educational and applicable for problems concerning FEM. Post-processing the results using ParaView appeared to be convenient for the thesis. It allows for a greater understanding and visualisation of the results. Some complications were met when formatting the results to the ".vtu"-file. The wanted outcome was to apply the strain results to the element as *cell data*. This data format did not allow for post-processing strain results relative to coordinates. As a result, the strain needed to be applied as *point data* and additional functions were established to prevent duplication of node data. The use of MeshIO and ParaView provided a strong platform for formatting and visualisation of data, and proved to be relevant for the thesis using FEA.

9 Conclusion

The study of behavior in the Seth-Hill Family of generalized strain regarding a triangular element was successfully completed. The implementation included the original SH strains as well as the first and second derivative of the equations. The results indicate that the SH strains are cases of a more general strain measure which are optimal for use in strain measurement. Through investigation of the triangle element, the Poynting effect was achieved.

The results achieved during the numerical approach of the fixed-end problem by use of Python FEM solver indicated faulty result for the strain. The implementation and testing of Green strain was successfully carried out. This was an anticipated outcome, as Green is widely used for FEA. Concerning the remaining equations of SH strain, proceeding examination is necessary to obtain reliable results.

Traditional and ANS formulation gave varying values for displacement of the cantilever. The results show that the cantilever would lengthen when subjected to shear load, and imply that the Poynting effect is achieved by use of SH strain. The study indicate that the Poynting effect is applicable for several materials.

The experiment could not be executed due to Covid-19 restrictions. Consequently, fundamental results for strain measure to prove the Poynting effect was not obtained.

9.1 Further work

Further work must be carried out to accomplish more accurate results for the SH strains. This work consider both FEM solver and experimental work. The current FEM solver allows for elastic deformation, and expanding it to include plastic deformation would be practical to get a more comprehensive understanding of the SH strains. A closer inspection of iterations of SH strain could indicate which generalized strain produces the most accurate result. It would be necessary to compute a more efficient and advanced testing method. Reconstruction and optimization of the code is essential to be prioritised.

Additional study on SH strain using traditional and ANS formulation should be accomplished. These are still considered to be in a experimental phase, and additional research and documentation is essential. Further work should include expanding the notation and make it applicable for 3D-problems.

A total arrangement of the test rig must be established and assembled to execute the experiment. Further research should include development of more advanced testing methods of Poynting effect.

Bibliography

[1] John Henry Poynting. On pressure perpendicular to the shear planes in finite pure shears, and on the lengthening of loaded wires when twisted. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 82(557):546–559, 06 1909.

[2] John Henry Poynting. On the changes in the dimensions of a steel wire when twisted, and on the pressure of distortional waves in steel. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 86(590):534–561, 1912.

[3] Giuseppe Zurloa, James Blackwellb, Niall Colganb, and Michel Destradea. The poynting effect. *American Journal of Physics 88*, 1036, 04 2020. doi: https://doi.org/10.1119/10.0001997.

[4] Carlos A. Felippa and Carlos Melo. Unified kinematic description of geometrically nonlinear finite elements, 2018.

[5] Olek C Zienkiewicz, Robert L Taylor, and J.Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 2005.

[6] Daryl L. Logan. *A first course in the finite element method*, volume ukjent. Cengage Learning, 2011.

[7] Carlos Felippa. Chapter 7: Review of continuum mechanics. In *Nonlinear Finite Element Methods– ASEN 6107*. University of Colorado, 2017. URL `https://www.coursicle.com/colorado/courses/ASEN/6107/`. ASEN at CU.

[8] Lebedev, P. Leonid, Victor A. Eremeyev, and Michael J. Cloud. *Tensor Analysis with Application in Mechanics*. World Scientific Publishing Company, 2003.

[9] Deformation gradient. URL `http://www.continuummechanics.org/deformationgradient.html`.

[10] Wikipedia. Finite strain theory, 2020. URL `https://en.wikipedia.org/wiki/Finite_strain_theory`.

[11] Zhi-Qiao Wang and Yu Wang. A natural generalization of linear isotropic relations with seth-hill strain tensors to transversely isotropic materials at finite strains. *Mathematical Problems in Engineering*, 2016:1–9, 01 2016. doi: 10.1155/2016/7473046.

[12] B. R. Seth. Generalised strain measure with application to physical problems. Technical report, Indian Institute of Technology and Mathematics Research Center, 1961.

[13] E. J. Hearn. *Mechanics of materials 1: an introduction to the mechanics of elastic and plastic deformation of solids and structural materials*. Butterworth-Heinemann, 1997.

[14] Kaspar J. William. *Finite element analysis of cellular structures*. PhD thesis, University of California, Berkeley, 1969.

[15] Carmelo Militello and Carlos A. Felippa. A variational justification of the assumed natural strain formulation of finite elements—i. variational principles. *Computers & Structures*, 34(3): 431–438, 1990. ISSN 0045-7949. doi: https://doi.org/10.1016/0045-7949(90)90267-6. URL `https://www.sciencedirect.com/science/article/pii/0045794990902676`.

[16] Carmelo Militello and Carlos A. Felippa. A variational justification of the assumed natural strain formulation of finite elements—ii. the c0 four-node plate element. *Computers & Structures*, 34(3):439–444, 1990. ISSN 0045-7949. doi: https://doi.org/10.1016/0045-7949(90)90268-7. URL `https://www.sciencedirect.com/science/article/pii/0045794990902687`.

[17] C. Geuzaine and J.-F. Remacle. Gmsh 4.8.1. URL `https://gmsh.info/doc/texinfo/gmsh.html`.

[18] ParaView. Overview. URL `https://www.paraview.org/`.

[19] Karsten Hennøen Nygård. Implementation and testing of 4-node andes tetrahedral solid element using python. Master's thesis, Norwegian University of Science and Technology, 2020.

[20] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79:1309–1331, 2009. URL https://gmsh.info.

[21] Bjorn Haugen. Buckling and stability problems for thin shell structures using high performance finite elements. 1994.

[22] Jr. William D. Callister and David G. Rethwisch. *Material Science and Engineering*, volume 9th. John Wiley & Sons Inc, 2014.

[23] Norsk Staal. Asm material data sheet, 2020. URL http://asm.matweb.com/search/SpecificMaterial.asp?bassnum=MQ304A.

[24] Norsk Staal. Hardox 400 bars, 2020. URL https://www.norskstaal.no/Files/Files/Produktkatalog/Stangst%C3%A5l/Hardox%20400%20Bars%20-%20Datablad.pdf.

[25] Aalco Metals Limited. Type 1.4003 stainless steel, 2020. URL https://www.aalco.co.uk/literature/files/aalco-stainless-steel-4003.pdf.

[26] Asm material data sheet, 2020. URL http://asm.matweb.com/search/SpecificMaterial.asp?bassnum=MA6063T6.

[27] SSAB. Strenx tube 700mh. URL https://www.ssab.com/products/steel-categories/hollow-section/products/strenx-tube-700-mh.

[28] Mechanical characterization of fiber reinforced polymer concrete. URL https://www.scielo.br/scielo.php?script=sci_arttext&pid=S1516-14392005000300023.

[29] Fjodor Sergejev and Mihhail Petrov. Assessment of residual stresses in steels and carbide composites by load and depth sensing indentation with spherical indenter. *Estonian Journal of Engineering*, 18, 01 2012. doi: 10.3176/eng.2012.3.11.

[30] Nils Petter Vedvik. Tmm4175 polymer composites, 2020. URL http://folk.ntnu.no/nilspv/TMM4175/material-properties.html.

[31] The Engineering toolbox. Torsion of shafts, 2005. URL https://www.engineeringtoolbox.com/torsion-shafts-d_947.html.

[32] Mechanics of materials: Torsion ≫ mechanics of slender structures — boston university. URL https://www.bu.edu/moss/mechanics-of-materials-torsion/.

[33] Rustfrie slipte dekorasjonsrør. URL https://www.norskstaal.no/produkter/alle-produkter/rustfrie-slipte-dek-roer-1-4301.

[34] Aluminiums rør - eidolon a/s. URL https://www.eidolon.no/products/aluminiumsror.

[35] Glassfiberrør - eidolon a/s. URL https://www.eidolon.no/products/glassfiberror.

[36] The Engineering toolbox. Asme/ansi b36.10/19 - carbon, alloy and stainless steel pipes - dimensions, 2003. URL https://www.engineeringtoolbox.com/steel-pipes-dimensions-d_43.html.

[37] Karbonfiber - nor-trading as. URL https://www.nortrading.no/categories/karbonfiber.

[38] Bernardo. Independent chuck k72 - bernardo. URL https://www.bernardo.at/en/planscheibe-k72-160.html.

[39] MSC. Drill chucks technical information — msc industrial supply co. URL https://www.mscdirect.com/basicsof/drill-chucks.

[40] TesT. Clamping devices: Test gmbh. URL https://www.test-gmbh.com/en/products/testing-machines/accessory-for-testing-machines/clamping-devices/.

[41] Noria Corporation. Worm gears explained, 2021. URL https://www.machinerylubrication.com/Read/1080/worm-gears.

[42] Framo Morat. Worm gear sets a65, 2016. URL https://framo-morat.com/products/worm-gear-sets/worm-gear-sets-a65/.

[43] Amazon. Stanley 0-94-606 ratchet rotator (9 piece), 2021. URL https://www.amazon.com.au/Stanley-0-94-606-Ratchet-Rotator-Silver/dp/B0024LEASQ.

[44] BOSCHE. S-type load cell — load cells — scale components — bosche. URL https://www.bosche.eu/en/scale-components/load-cells/s-type-load-cell.

[45] COMSOL. Eigenfrequency analysis. https://www.comsol.com/multiphysics/eigenfrequency-analysis. (Accessed on 06/05/2021).

[46] SKF. 51106 - thrust ball bearings, single direction — skf. URL https://www.skf.com/us/products/rolling-bearings/ball-bearings/thrust-ball-bearings/productid-51106.

[47] M. S. Floater. Generalized barycentric coordinates and applications. *Acta Mechanica*, 24: 161–214, 2015.

A Constant Strain Triangle (CST)

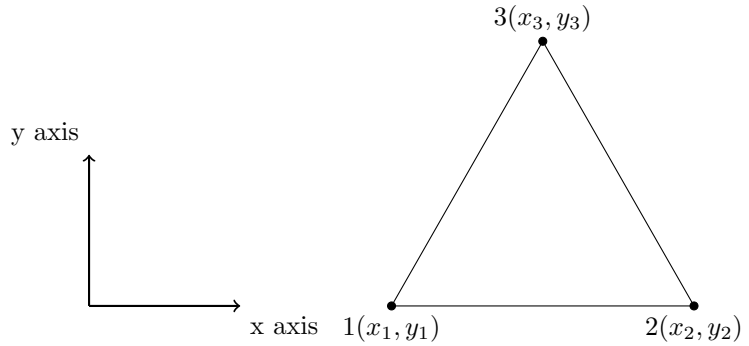


Figure A.0.1: Constant strain triangle

The local coordinates of a triangle element can be associated as area coordinates, triangular coordinates or by barycentric coordinates. The mathematical aspect is quite favorable and extensively used in computer algorithms dealing with polygonal geometry elements, such as the finite element method. [47]

The area at the triangle is given by the cross product of the side edge vector:

$$2A = e_{12} \times e_{13} = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix} \times \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \end{bmatrix} = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1) \tag{A.0.1}$$

$$= \begin{cases} 1\,(x_2 y_3 - x_3 y_2) \\ 1\,(x_1 y_3 - x_3 y_1) \\ 1\,(x_1 y_2 - x_2 y_1) \end{cases} = det \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \tag{A.0.2}$$

If we let the top point be an internal point (x, y), we have:

$$2A_1 = det \begin{bmatrix} 1 & x & y \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} = \underbrace{(x_2 y_3 - x_3 y_2)}_{a_1} + x \underbrace{(y_2 - y_3)}_{b_1} + y \underbrace{(x_3 - x_2)}_{c_1} \tag{A.0.3}$$

Through cyclic permutation of the nodes we have:

$$a_i = x_j y_k - x_k y_j \tag{A.0.4}$$

$$b_i = y_j - y_k \tag{A.0.5}$$

$$c_i = x_k - x_j \tag{A.0.6}$$

$$2A_i = a_i + b_i x + c_i y \tag{A.0.7}$$

The area coordinates is given by partial derivatives, $\zeta_i = \frac{A_i}{A}$. The area coordinate ζ_i have unit value at node i and are zero at the line between $j - k$. The partial derivatives at the area coordinates, is computed in Equation (A.0.8) and Equation (A.0.9)

$$\frac{\partial \zeta_i}{\partial x} = \frac{b_i}{2A} = \frac{y_j - y_k}{2A} \tag{A.0.8}$$

$$\frac{\partial \zeta_i}{\partial y} = \frac{c_i}{2A} = \frac{x_k - x_j}{2A} \tag{A.0.9}$$

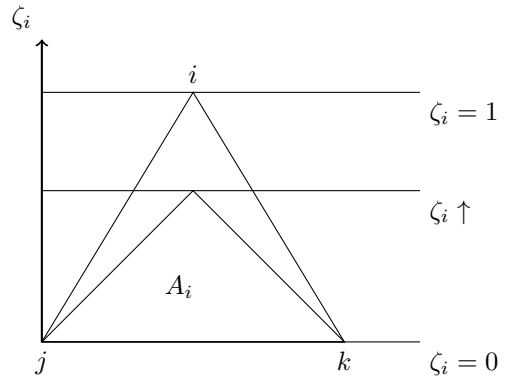


Figure A.0.2: Area coordinates of a triangle

The strains can be computed from the area coordinates. The relation is explained in Equation (A.0.10)

$$\begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} u_{,x} \\ v_{,y} \\ u_{,y} + v_{,x} \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} b_1 & 0 & b_2 & 0 & b_3 & 0 \\ 0 & c_1 & 0 & c_2 & 0 & c_3 \\ c_1 & b_1 & c_2 & b_2 & c_3 & b_3 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix} \tag{A.0.10}$$

Since the strains over the element will be constant, the computation to get the stiffness matrix of the element is simple:

$$\mathbf{k}_e = \int_v \mathbf{B}^T \mathbf{C} \mathbf{B} \, dV = Ah \mathbf{B}^T \mathbf{C} \mathbf{B} \tag{A.0.11}$$

In Equation (A.0.11), h is thickness at the element and the constitutive matrix for plane stress is given by Equation (A.0.12).

$$\mathbf{C} = \frac{E}{(1 - \upsilon^2)} \begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & \frac{1}{2}(1 - \upsilon) \end{bmatrix} \tag{A.0.12}$$

B Code

B.1 Traditional Strain

B.1.1 Function zeta_partials_x_and_y

```python
def zeta_partials_x_and_y(ex,ey):
    """
    Compute partials of area coordinates with respect to x and y.
    :param list ex: element x coordinates [x1, x2, x3]
    :param list ey: element y coordinates [y1, y2, y3]
    """

    tmp = np.array([[1,ex[0],ey[0]],
                    [1,ex[1],ey[1]],
                    [1,ex[2],ey[2]]])

    A2 = np.linalg.det(tmp)  # Double of triangle area

    cyclic_ijk = [0,1,2,0,1]      # Cyclic permutation of the nodes i,j,k

    zeta_px = np.zeros(3)           # Partial derivative with respect to x
    zeta_py = np.zeros(3)           # Partial derivative with respect to y

    for i in range(3):
        j = cyclic_ijk[i+1]
        k = cyclic_ijk[i+2]
        zeta_px[i] = (ey[j] - ey[k]) / A2
        zeta_py[i] = (ex[k] - ex[j]) / A2

    return zeta_px, zeta_py
```

Listing 1: Partial derivatives ζ_i

B.1.2 Function get_B_matrix_linear

```python
def get_B_matrix_linear(ex,ey):
    """
    Compute the element strain displacement matrix for a triangle element
    :param list ex: element x coordinates [x1, x2, x3]
    :param list ey: element y coordinates [y1, y2, y3]
    :return mat Be: element strain displacement matrix [3 x 6]
    """
    zi_px, zi_py =  zeta_partials_x_and_y(ex ,ey)

    Be = np.array([
            [zi_px[0],         0, zi_px[1],         0, zi_px[2],         0],
            [       0, zi_py[0],         0, zi_py[1],         0,zi_py[2]],
            [zi_py[0], zi_px[0], zi_py[1], zi_px[1], zi_py[2], zi_px[2]]])

    return Be
```

Listing 2: Element strain displacement matrix $\mathbf{B}_\varepsilon$

B.1.3 Function get_Disp_grad

```python
def get_Disp_grad(ex,ey):
    """
    Compute the Displacement gradient matrix for a triangle element

    :param list ex: element x coordinates [x1, x2, x3]
    :param list ey: element y coordinates [y1, y2, y3]
    :return mat Disp_grad: element strain displacement matrix [4 x 6]
    """

    zi_px, zi_py =  zeta_partials_x_and_y(ex ,ey)

    Disp_grad = np.array([
            [zi_px[0],         0, zi_px[1],         0, zi_px[2],         0],
```

```
            [       0, zi_px[0],         0, zi_px[1],         0, zi_px[2]],
            [zi_py[0],         0, zi_py[1],         0, zi_py[2],         0],
            [       0, zi_py[0],         0, zi_py[1],         0, zi_py[2]]])

    return Disp_grad
```

Listing 3: Displacement gradient matrix B_{gv}

B.1.4 Function get Disp_grad_Vec

```
def getDisp_grad_Vec(ex,ey,dispVec):
    """
    Compute the displacement gradient vector for a triangle element.
    :param list ex: element x coordinates [x1, x2, x3]
    :param list ey: element y coordinates [y1, y2, y3]
    :return gVec: element strains [epsXX,epsYY,gammaXY]
    """
    Dispmat = get_Disp_grad(ex,ey)
    gVec = Dispmat @ dispVec
    return gVec
```

Listing 4: Displacement gradient **g**

B.1.5 Function get_StrainVec_trad

```
def get_strainVec_trad(ex,ey,dispVec, m):
    """Compute the strain vector for Seth-hill family strain"""

    for i in range(3):
        gVec = get_Disp_grad_Vec(ex,ey,dispVec)
        eVec = np.zeros(3)
        Bmat = get_B_matrix_linear(ex,ey)

        if m ==2: #Green
            eVec[0] = gVec[0]+ (gVec[0]**2 + gVec[1]**2)/2.0
            eVec[1] = gVec[3]+ (gVec[3]**2 + gVec[2]**2)/2.0
            eVec[2] = gVec[1]+ gVec[2] + gVec[0]*gVec[2] + gVec[1]*gVec[3]

        elif m == 1: #Biot
            eVec = Bmat @ dispVec

        elif m == 0: #Hencky
            eVec[0] = math.log(1 + gVec[0])
            eVec[1] = math.log(1 + gVec[3])
            eVec[2] = math.log(1 + gVec[1] + gVec[2])

        elif m == -1: #Swainger
            eVec[0] = gVec[0] / (1 + gVec[0])
            eVec[1] = gVec[3] / (1 + gVec[3])
            eVec[2] =  (gVec[1]+gVec[2]) / (1 + gVec[1]+gVec[2])


        elif m == -2: #Almansi
            eVec[0] = ((1 + gVec[0])**-2 - 1) * -1/2
            eVec[1] = ((1 + gVec[3])**-2 - 1) * -1/2
            eVec[2] = ((1 + gVec[1]+gVec[2])**-2 - 1) * -1/2

    return eVec
```

Listing 5: Seth-Hill strain ε

B.1.6 Function get_StrainVec_trad_FD

```
def get_strainVec_trad_FD(ex,ey,dispVec, m):
    """Compute the strain vector for the first derivative of Seth-hill family
    strain"""

    for i in range(3):
        gVec = get_Disp_grad_Vec(ex,ey,dispVec)
        eVec = np.zeros(3)
```

```
7
8           if m ==2: #Green
9               eVec[0] = (gVec[0]+ 1) + gVec[1]
10              eVec[1] = (gVec[3]+ 1) + gVec[2]
11              eVec[2] =  gVec[2] + (1 + gVec[0]) + (1 + gVec[3]) + gVec[1]
12
13          elif m == 1: #Biot
14              eVec[0] = 1
15              eVec[1] = 1
16              eVec[2] = 1
17
18          elif m == 0: #Hencky
19              eVec[0] = 1 / (1 + gVec[0])
20              eVec[1] = 1 / (1 + gVec[3])
21              eVec[2] =  1 / (1 + gVec[1] + gVec[2])
22
23          elif m == -1: #Swainger
24              eVec[0] = 1 / (1 + gVec[0])**2
25              eVec[1] = 1 / (1 + gVec[3])**2
26              eVec[2] = 1 / (1 + gVec[1] + gVec[2])**2
27
28
29          elif m == -2: #Almansi
30              eVec[0] = ( -2 * (1 + gVec[0])**-3 ) * - 1/2
31              eVec[1] = ( -2 * (1 + gVec[3])**-3 ) * - 1/2
32              eVec[2] = ( -2 * (1 + gVec[1] + gVec[2])**-3 ) * - 1/2
33      return eVec
```

Listing 6: First derivative of Seth-Hill strain $d\boldsymbol{\varepsilon}$

B.1.7 **Function** get_StrainVec_trad_SD

```
1  def get_strainVec_trad_SD(ex,ey,dispVec, m):
2      """Compute the strain vector for the second derivative of Seth-hill family
       strain"""
3
4      for i in range(3):
5          gVec = get_Disp_grad_Vec(ex,ey,dispVec)
6          eVec = np.zeros(3)
7
8          if m ==2: #Green
9              eVec[0] = 2
10             eVec[1] = 2
11             eVec[2] = 4
12
13         elif m == 1: #Biot
14             eVec[0] = 0
15             eVec[1] = 0
16             eVec[2] = 0
17
18         elif m == 0: #Hencky
19             eVec[0] = - 1 / (1 + gVec[0])**2
20             eVec[1] = - 1 / (1 + gVec[3])**2
21             #eVec[2] = 4
22             eVec[2] =- 1 / (1 + gVec[1] + gVec[2])**2
23
24         elif m == -1: #Swainger
25             eVec[0] = - 2 / (1 + gVec[0])**3
26             eVec[1] = - 2 / (1 + gVec[3])**3
27             eVec[2] = - 2 / (1 + gVec[1] + gVec[2])**3
28
29
30         elif m == -2: #Almansi
31             eVec[0] = ( 6 * (1 + gVec[0])**-4 ) * - 1/2
32             eVec[1] = ( 6 * (1 + gVec[3])**-4 ) * - 1/2
33             eVec[2] = ( 6 * (1 + gVec[1] + gVec[2])**-4 ) * - 1/2
34
35     return eVec
```

Listing 7: Second derivative of Seth-Hill strain $d^2\boldsymbol{\varepsilon}$

B.1.8 Function get_B_eg_matrix_trad

```
def get_B_eg_matrix_trad(ex, ey, dispVec, m):
    """
    Compute the B_eg matrix for traditional strain.
    Consist of partial derivative of strain with respect to g
    """
    g = np.zeros(4)
    g = tri3.get_Disp_grad_Vec(ex, ey, dispVec)

    if m == 2:
        Beg_mat = np.array([[1 + g[0], g[1], 0, 0],
                            [0, 0, g[2], 1 + g[3]],
                            [g[2], 1 + g[3], 1 + g[0], g[1]]])

    if m == 1:
        Beg_mat = np.array([[1, 0, 0, 0],
                            [0, 0, 0, 1],
                            [0, 1, 1, 0]])

    if m == 0:
        Beg_mat = np.array([[1/(1+g[0]), 0, 0, 0],
                            [0, 0, 0, 1 / (1+g[3])],
                            [0, 1/(1+g[1]+g[2]),1/(1+g[1]+g[2]), 0]])


    if m == -1:
        Beg_mat = np.array([[ (1 + g[0])**-2,                    0,
      0,              0],
                            [              0,                    0,
      0, (1 + g[3])**-2],
                            [              0, 1*(1+g[1]+g[2])**-2, 1*(1+g[1]+g[2])
    **-2,              0]])

    if m == -2:
        Beg_mat = np.array([[(1 + g[0]) ** -3, 0, 0, 0],
                            [0, 0, 0, (1 + g[3]) ** -3],
                            [0, (1 + g[1] + g[2]) ** -3, (1 + g[1] + g[2]) ** -3,
    0]])

    Beg = Beg_mat.astype(float)

    return Beg
```

Listing 8: $\mathbf{B}_{\varepsilon g}$ matrix

B.1.9 Function get_internal_force_trad

```
def get_internal_force_trad(ex, ey, dispVec, C, m):
    """Compute the internal forces for green strain"""

    B_gv = tri3.get_Disp_grad(ex,ey)
    B_eg = get_B_eg_matrix_trad(ex,ey,dispVec, m)

    eps = tri3.get_strainVec_trad(ex,ey,dispVec, m)
    s = C @ eps

    fe =  B_gv.T @ B_eg.T @ s
    return fe
```

Listing 9: Internal force $\mathbf{f}_{int}$

B.1.10 Function get_material_stiffness_matrix_trad

```
def get_material_stiffness_matrix_trad(ex, ey, dispVec, C, m):
    """
    Compute the material stiffness matrix for traditional seth-hill strain, K3
    """
    B_gv = tri3.get_Disp_grad(ex,ey)
    B_eg = get_B_eg_matrix_trad(ex,ey,dispVec, m)

```

```
    K3 =  B_gv.T @ B_eg.T @ C @ B_eg @ B_gv
    return K3
```

Listing 10: Material stiffness matrix $\mathbf{K}_3$

B.1.11 Function get_S_matrix

```
def get_S_matrix(ex, ey, dispVec, C, m):
    """
    Compute the S-matrix for green strain.
    Hver kolonne i s matrisen er et produkt av B_eg.T*s / g
    """

    eps = tri3.get_strainVec_trad(ex, ey, dispVec, m)
    g = np.zeros(4)
    g = tri3.get_Disp_grad_Vec(ex, ey, dispVec)

    s = C @ eps
    if m == 2:
        B_eg1 = np.array([
            [1, 0, 0],
            [0, 0, 0],
            [0, 0, 1],
            [0, 0, 0]])

        B_eg2 = np.array([
            [0, 0, 0],
            [1, 0, 0],
            [0, 0, 0],
            [0, 0, 1]])

        B_eg3 = np.array([
            [0, 0, 1],
            [0, 0, 0],
            [0, 1, 0],
            [0, 0, 0]])

        B_eg4 = np.array([
            [0, 0, 0],
            [0, 0, 1],
            [0, 0, 0],
            [0, 1, 0]])

    if m == 1:
        B_eg1 = np.array([
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0]])

        B_eg2 = np.array([
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0]])

        B_eg3 = np.array([
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0]])

        B_eg4 = np.array([
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0]])

    if m == 0:
        B_eg1 = np.array([
            [-1/(1+g[0])**2, 0, 0],
            [0, 0, 0],
```

```
            [0, 0, 0],
            [0, 0, 0]])

        B_eg2 = np.array([
            [0, 0, 0],
            [0, 0, 1/(1+g[1]+g[2])],
            [0, 0, 0],
            [0, 0, 0]])

        B_eg3 = np.array([
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 1/(1+g[1]+g[2])],
            [0, 0, 0]])

        B_eg4 = np.array([
            [0, 0, 0],
            [0, 0, 1],
            [0, 0, 0],
            [0,-1/(1+g[3])**2, 0]])

    if m == -1:
        B_eg1 = np.array([
            [-2*(1+g[0])**-3, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0]])

        B_eg2 = np.array([
            [0, 0, 0],
            [0, 0, -2*(1+g[1]+g[2])**-3],
            [0, 0, 0],
            [0, 0, 0]])

        B_eg3 = np.array([
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, -2*(1+g[1]+g[2])**-3],
            [0, 0, 0]])

        B_eg4 = np.array([
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, -2*(1+g[3])**-3, 0]])

    if m == -2:
        B_eg1 = np.array([
            [-3 * (1+g[0])**-4, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0]])

        B_eg2 = np.array([
            [0, 0, 0],
            [0, 0, -3*(1+g[1]+g[2])**-4],
            [0, 0, 0],
            [0, 0, 0]])

        B_eg3 = np.array([
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, -3*(1+g[1]+g[2])**-4],
            [0, 0, 0]])

        B_eg4 = np.array([
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0],
            [0, -3*(1+g[3])**-4, 0]])

    S1 = B_eg1 @ s
    S2 = B_eg2 @ s
```

```
    S3 = B_eg3 @ s
    S4 = B_eg4 @ s

    S = np.array([[S1[0], S2[0], S3[0], S4[0]],
                  [S1[1], S2[1], S3[1], S4[1]],
                  [S1[2], S2[2], S3[2], S4[2]],
                  [S1[3], S2[3], S3[3], S4[3]]])

    Smat = S.astype(float)
    return Smat
```

Listing 11: **S** matrix

B.1.12 Function get_geometric_stiffness_matrix_trad

```
def get_geometric_stiffness_matrix_trad(ex, ey, dispVec, C, m):
    """
    Geometric stiffness matrix for traditional strain
    """
    B_gv = tri3.get_Disp_grad(ex,ey)
    S = get_S_matrix(ex,ey,dispVec, C, m)

    K2 =  B_gv.T @ S @ B_gv
    return K2
```

Listing 12: Geometric stiffness matrix $\mathbf{K}_2$

B.2 Assumed Natural strain

B.2.1 Function get_length_initial

```
def get_length_initial(ex, ey, dispVec):
    """Compute the intitial length of the ANS element"""
    L0 = np.zeros(3)

    L0i = math.sqrt((ex[1]-ex[0])**2 + (ey[1]-ey[0])**2)
    L0j = math.sqrt((ex[2]-ex[1])**2 + (ey[2]-ey[1])**2)
    L0k = math.sqrt((ex[0]-ex[2])**2 + (ey[0]-ey[2])**2)

    L0[0] = L0i
    L0[1] = L0j
    L0[2] = L0k

    return L0
```

Listing 13: Initial length for bar $\mathbf{L}_0$

B.2.2 Function get_length_deformed

```
def get_length_deformed(ex, ey, dispVec):
    """Compute the deformed length of the ANS element"""
    L = np.zeros(3)

    Li = math.sqrt( ((ex[1]-ex[0]) + (dispVec[2]-dispVec[0]))**2 + ((ey[1]-ey[0])
    + (dispVec[3]-dispVec[1]))**2)
    Lj = math.sqrt( ((ex[2]-ex[1]) + (dispVec[4]-dispVec[2]))**2 + ((ey[2]-ey[1])
    + (dispVec[5]-dispVec[3]))**2)
    Lk = math.sqrt( ((ex[0]-ex[2]) + (dispVec[0]-dispVec[4]))**2 + ((ey[0]-ey[2])
    + (dispVec[1]-dispVec[5]))**2)

    L[0] = Li
    L[1] = Lj
    L[2] = Lk

    return L
```

Listing 14: Deformed length for bar **L**

B.2.3 Function get_Triangle_element_transformation_matrix

```python
def get_Triangle_element_transformation_matrix(ex, ey, dispVec):
    """Compute the Transformation matrix for triangle element"""
    L0 = get_length_initial(ex, ey, dispVec)

    Ci = (ex[1] - ex[0])/L0[0]
    Si = (ey[1] - ey[0])/L0[0]
    Cj = (ex[2] - ex[1])/L0[1]
    Sj = (ey[2] - ey[1])/L0[1]
    Ck = (ex[0] - ex[2])/L0[2]
    Sk = (ey[0] - ey[2])/L0[2]

    T_nc = np.array([  [ Ci**2, Si**2, Ci*Si],
                       [ Cj**2, Sj**2, Cj*Sj],
                       [ Ck**2, Sk**2, Ck*Sk]  ])
    return T_nc
```

Listing 15: Transformation matrix between Cartesian and Natural strain $\mathbf{T}_{nc}$

B.2.4 Function get_Triangle_element_transformation_matrix_deformed

```python
def get_Triangle_element_transformation_matrix_deformed(ex, ey, dispVec):
    """Compute the Transformation matrix for triangle element, with deformed length
    """
    L = get_length_deformed(ex, ey, dispVec)

    Ci =  float(( ex[1] + dispVec[2] - ex[0] - dispVec[0] ) / L[0])
    Si =  float(( ey[1] + dispVec[3] - ey[0] - dispVec[1] ) / L[0])
    Cj =  float(( ex[2] + dispVec[4] - ex[1] - dispVec[2] ) / L[1])
    Sj =  float(( ey[2] + dispVec[5] - ey[1] - dispVec[3] ) / L[1])
    Ck =  float(( ex[0] + dispVec[0] - ex[2] - dispVec[4] ) / L[2])
    Sk =  float(( ey[0] + dispVec[1] - ey[2] - dispVec[5] ) / L[2])

    T_nc = np.array([  [ Ci**2, Si**2, Ci*Si],
                       [ Cj**2, Sj**2, Cj*Sj],
                       [ Ck**2, Sk**2, Ck*Sk]  ])
    return T_nc
```

Listing 16: Transformation matrix between Cartesian and Natural strain with deformed length $\mathbf{T}_{nc}$

B.2.5 Function get_Triangle_element_lambda_vector

```python
def get_Triangle_element_lambda_vector(ex, ey, dispVec):
    """Compute the lambda vector for ANS element"""
    Lambda = np.zeros(3)
    L0     = get_length_initial(ex, ey, dispVec)
    L      = get_length_deformed(ex, ey, dispVec)

    for i in range(3):
        Lambda[i] = L[i] / L0[i]
        i += 1
    return Lambda
```

Listing 17: Principal stretch for bar $\boldsymbol{\lambda}_i$

B.2.6 Function get_NaturalCoordinates_strain

```python
def get_NaturalCoordinates_strain(ex, ey, dispVec, m):
    """Calculate the Seth-Hill Natural Coordinate strains for ANS element"""
    e_n = np.zeros(3)

    for i in range(3):
        Lambda = get_Triangle_element_lambda_vector(ex, ey, dispVec)

        if m == 2: #Green
            e_n[i] = 1/2 * (Lambda[i]**2 - 1)
            i += 1

        elif m == 1: #Biot
            e_n[i] = Lambda[i] - 1
```

```
            i += 1

        elif m == 0: #Hencky
            e_n[i] = math.log(Lambda[i])
            i += 1

        elif m == -1: #Swainger
            e_n[i] = 1 - 1/Lambda[i]
            i += 1

        elif m == -2: #Almansi
            e_n[i] = (1 - (1/Lambda[i]**2))*1/2
            i += 1
    return e_n
```

Listing 18: Natural Coordinate Seth-Hill strain $\boldsymbol{\varepsilon}_n$

B.2.7 Function get_NaturalCoordinates_strain_FD

```
def get_NaturalCoordinates_strain_FD(ex, ey, dispVec, m):
    """Calculate the first derivative of Natural Coordinate strains for ANS element
    """
    e_n = np.zeros(3)

    for i in range(3):
        L0 = get_length_initial(ex, ey, dispVec)
        L  = get_length_deformed(ex, ey, dispVec)

        if m == 2: #Green
            e_n[i] = L[i] / L0[i]**2
            i += 1

        elif m == 1: #Biot
            e_n[i] = 1 / L0[i]
            i += 1

        elif m == 0: #Hencky
            e_n[i] = 1 / L[i]
            i += 1

        elif m == -1: #Swainger
            e_n[i] = L0[i] / L[i]**2
            i += 1

        elif m == -2: #Almansi
            e_n[i] = L0[i]**2 / L[i]**3
            i += 1
    return e_n
```

Listing 19: First derivative of Natural Coordinate Seth-Hill strain $d\boldsymbol{\varepsilon}_n$

B.2.8 Function get_NaturalCoordinates_strain_SD

```
def get_NaturalCoordinates_strain_SD(ex, ey, dispVec, m):
    """Calculate the second derivative of Natural Coordinate strains for ANS
    element"""
    e_n = np.zeros(3)

    for i in range(3):
        L0 = get_length_initial(ex, ey, dispVec)
        L  = get_length_deformed(ex, ey, dispVec)

        if m == 2: #Green
            e_n[i] = 1 / L0[i]**2
            i += 1

        elif m == 1: #Biot
            e_n[i] = 0
            i += 1

        elif m == 0: #Hencky
```

```
            e_n[i] = - 1 / L[i]**2
            i += 1

        elif m == -1: #Swainger
            e_n[i] = -2 * L0[i] / L[i]**3
            i += 1

        elif m == -2: #Almansi
            e_n[i] = -3 * L0[i]**2 / L[i]**4
            i += 1
    return e_n
```

Listing 20: Second derivative of Natural Coordinate Seth-Hill strain $d^2\boldsymbol{\varepsilon}_n$

B.2.9 Function get_CartesianCoordinates_strain

```
def get_CartesianCoordinates_strain(ex, ey, dispVec, m, n):
    """Calculate the Seth-Hill Cartesian Coordinate strains for ANS element"""
    e_c = np.zeros(3)

    for i in range(3):
        if n == 0:   #Natural coordinate strain
            e_n  = get_NaturalCoordinates_strain(ex, ey, dispVec, m)
        if n == 1:   #First derivative of natural coordinate strain
            e_n  = get_NaturalCoordinates_strain_FD(ex, ey, dispVec, m)
        if n == 2:   #Second derivative of natural coordinate strain
            e_n  = get_NaturalCoordinates_strain_SD(ex, ey, dispVec, m)

        T_nc = get_Triangle_element_transformation_matrix(ex, ey, dispVec)
        T_cn = np.linalg.inv(T_nc)
        e_c  =  T_cn @ e_n
    return e_c
```

Listing 21: Cartesian Coordinate Seth-Hill strain $\boldsymbol{\varepsilon}_c$

B.2.10 Function get_B_lv_matrix

```
def get_B_lv_matrix(ex,ey, dispVec):
    """Compute the B_lv matrix, consisting of unit vectors along side edge lengths
    """
    L = bar.get_length_deformed(ex, ey, dispVec)

    e1x =  float(( ex[1] + dispVec[2] - ex[0] - dispVec[0] ) / L[0])
    e1y =  float(( ey[1] + dispVec[3] - ey[0] - dispVec[1] ) / L[0])
    e2x =  float(( ex[2] + dispVec[4] - ex[1] - dispVec[2] ) / L[1])
    e2y =  float(( ey[2] + dispVec[5] - ey[1] - dispVec[3] ) / L[1])
    e3x =  float(( ex[0] + dispVec[0] - ex[2] - dispVec[4] ) / L[2])
    e3y =  float(( ey[0] + dispVec[1] - ey[2] - dispVec[5] ) / L[2])

    B_lv   = np.array([[ -e1x, -e1y,  e1x,  e1y,    0,     0 ],
                       [    0,    0, -e2x, -e2y,  e2x,   e2y ],
                       [  e3x,  e3y,    0,    0, -e3x,  -e3y ] ])
    return B_lv
```

Listing 22: $\mathbf{B}_{lv}$ matrix

B.2.11 Function get_B_el_matrix

```
def get_B_el_matrix(ex,ey, dispVec, m):
    """ B_el matrix consisting of partial derivative of natural strain with respect
     to length"""
    e_n    = bar.get_NaturalCoordinates_strain_FD(ex, ey, dispVec, m)

    B_el   = np.array([ [e_n[0],      0,    0  ],
                        [     0, e_n[1],    0  ],
                        [     0,      0, e_n[2]] ])
    return B_el
```

Listing 23: $\mathbf{B}_{\varepsilon l}$ matrix

B.2.12 Function get_interal_force_ANS

```
def get_internal_force_ANS(ex,ey, dispVec, C, m):
    """Compute the Internal Force vector for ANS element """
    B_lv  = get_B_lv_matrix(ex,ey, dispVec)
    B_el  = get_B_el_matrix(ex,ey, dispVec, m)

    eps_n = bar.get_NaturalCoordinates_strain(ex, ey, dispVec, m)
    #Transformation between Cartesian and Natural strain
    if m == -2:
        T_nc = bar.get_Triangle_element_transformation_matrix_deformed(ex, ey,
    dispVec)
        T_cn = np.linalg.inv(T_nc)
    else:
        T_nc = bar.get_Triangle_element_transformation_matrix(ex, ey, dispVec)
        T_cn = np.linalg.inv(T_nc)
    C_n   = T_cn.T @ C @ T_cn
    s_n   = C_n @ eps_n

    f_int = B_lv.T @ B_el.T @ s_n
    return f_int
```

Listing 24: Internal force vector $\mathbf{f}_{int}$

B.2.13 Function get_material_stiffness_ANS_K3

```
def get_material_stiffness_ANS_K3(ex,ey, dispVec, C, m):
    """ Compute the Material Stiffness Matrix, K3 for ANS element """

    B_lv = get_B_lv_matrix(ex,ey, dispVec)
    B_el = get_B_el_matrix(ex,ey, dispVec, m)

    #Transformation between cartesian and natural strain
    if m ==-2:
        T_nc = bar.get_Triangle_element_transformation_matrix_deformed(ex, ey,
    dispVec)
        T_cn = np.linalg.inv(T_nc)

    else:
        T_nc = bar.get_Triangle_element_transformation_matrix(ex, ey, dispVec)
        T_cn = np.linalg.inv(T_nc)

    C_n = T_cn.T @ C @ T_cn

    Km3 = B_lv.T @ B_el.T @ C_n @ B_el @ B_lv
    return Km3
```

Listing 25: Material stiffness matrix $\mathbf{K}_3$

B.2.14 Function get_B_eLL_s_matrix

```
def get_B_eLL_s_matrix(ex,ey, C, dispVec, m):
    """Compute the B_[ell * s] matrix for ANS"""
    eps_n = bar.get_NaturalCoordinates_strain(ex, ey, dispVec, m)

    #Transformation between Cartesian and Natural strain
    if m ==-2:
        T_nc = bar.get_Triangle_element_transformation_matrix_deformed(ex, ey,
    dispVec)
        T_cn = np.linalg.inv(T_nc)
    else:
        T_nc = bar.get_Triangle_element_transformation_matrix(ex, ey, dispVec)
        T_cn = np.linalg.inv(T_nc)

    C_n = T_cn.T @ C @ T_cn
    s_n = C_n @ eps_n

    #second derivatives of Natural green strain
    e_n     = bar.get_NaturalCoordinates_strain_SD(ex, ey, dispVec, m)

    B_ells  = np.array([[e_n[0]*s_n[0],             0,             0 ],
```

```
                        [            0, e_n[1]*s_n[1],             0 ],
                        [            0,            0, e_n[2]*s_n[2] ] ])
    return B_ells
```

Listing 26: $\mathbf{B}_{[\varepsilon lls]}$ matrix

B.2.15 Function get_geometric_stiffness_ANS_K2

```
def get_geometric_stiffness_ANS_K2(ex,ey, dispVec, C, m):
    """Compute the Geometric Dtiffness matrix, K2 for ANS """
    B_lv   = get_B_lv_matrix(ex,ey, dispVec)
    B_eLLs = get_B_eLL_s_matrix(ex,ey, C, dispVec, m)

    Kg2    =  B_lv.T @ B_eLLs @ B_lv
    return Kg2
```

Listing 27: Geometric stiffness matrix $\mathbf{K}_2$

B.2.16 Function get_B_k_matrix

```
def get_B_k_matrix(ex,ey,dispVec):
    """Get the B_k matrix, consisting of inward normal vectors"""

    L = bar.get_length_deformed(ex, ey, dispVec)
    nx1 = float(( ex[1] + dispVec[2] - ex[0] - dispVec[0] ) / L[0])
    ny1 = float(( ey[1] + dispVec[3] - ey[0] - dispVec[1] ) / L[0])
    nx2 = float(( ex[2] + dispVec[4] - ex[1] - dispVec[2] ) / L[1])
    ny2 = float(( ey[2] + dispVec[5] - ey[1] - dispVec[3] ) / L[1])
    nx3 = float(( ex[0] + dispVec[0] - ex[2] - dispVec[4] ) / L[2])
    ny3 = float(( ey[0] + dispVec[1] - ey[2] - dispVec[5] ) / L[2])

    B_k   = np.array([ [    ny1, -nx1,  -ny1,  nx1,     0,     0 ],
                       [      0,    0,   ny2, -nx2,  -ny2,   nx2 ],
                       [   -ny3,  nx3,     0,    0,   ny3,  -nx3 ] ])

    return B_k
```

Listing 28: $\mathbf{B}_k$ matrix

B.2.17 Function get_geometric_stiffness_ANS_K1

```
def get_geometric_stiffness_ANS_K1(ex,ey, dispVec, C, m):
    """Compute the geometric stiffness matrix K1 for ANS-element"""

    L   = bar.get_length_deformed(ex, ey, dispVec)
    B_el = get_B_el_matrix(ex,ey, dispVec, m)
    B_k  = get_B_k_matrix(ex,ey,dispVec)
    eps_n = bar.get_NaturalCoordinates_strain(ex, ey, dispVec, m)

    #Transformation between cartesian and natural strain
    if m ==-2:
        T_cn = bar.get_Triangle_element_transformation_matrix_almansi(ex, ey,
    dispVec)
        T_nc = np.linalg.inv(T_cn)
    else:
        T_cn  = bar.get_Triangle_element_transformation_matrix(ex, ey, dispVec)
        T_nc = np.linalg.inv(T_cn)

    C_n = T_nc.T @ C @ T_nc
    s_n = C_n @ eps_n

    SHN  = B_el.T @ s_n
    S_hn = np.array  ([ [SHN[0]/L[0],           0,           0  ],
                        [          0, SHN[1]/L[1],           0  ],
                        [          0,           0, SHN[2]/L[2]  ] ])

    Kg1 = B_k.T @ S_hn @ B_k
    return Kg1
```

Listing 29: Geometric stiffness matrix $\mathbf{K}_1$

B.2.18 Function get_tangent_stiffness_ANS

```
def get_tangent_stiffness_ANS(ex,ey, dispVec, C, m):
    """Compute the tanget stiffness matrix for ANS element"""
    Kg1 = get_geometric_stiffness_ANS_K1(ex,ey, dispVec, C, m)
    Kg2 = get_geometric_stiffness_ANS_K2(ex,ey, dispVec, C, m)
    Km3 = get_material_stiffness_ANS_K3(ex,ey, dispVec, C, m)

    Kt = Kg1 + Kg2 + Km3
    return Kt
```

Listing 30: Tangent stiffness matrix $\mathbf{K}_t$

B.3 Numerical generated stiffness matrix

B.3.1 Function test_internal_force_trad

```
def test_internal_force_trad(ex,ey, dispVec, C, m):
    delta = 1.0e-6
    K = np.zeros((6,6))
    disp_add = np.zeros(6)
    disp_sub = np.zeros(6)

    for i in range(len(dispVec)):
        disp_add = copy.deepcopy(dispVec)
        disp_add[i] = disp_add[i] + delta

        disp_sub = copy.deepcopy(dispVec)
        disp_sub[i] = disp_sub[i] - delta

        f_add = tan3.get_internal_force_trad(ex, ey, disp_add, C, m)
        f_sub = tan3.get_internal_force_trad(ex, ey, disp_sub, C, m)

        for j in range(len(f_add)):
            K[j,i] = ( f_add[j] - f_sub[j] ) / ( 2 * delta )
    return K
```

Listing 31: Test of internal force vector for traditional strain

B.3.2 Function test_internal_force_ANS

```
def test_internal_force_ANS(ex,ey, dispVec, C, m):
    delta = 1.0e-6
    K = np.zeros((6,6))
    disp_add = np.zeros(6)
    disp_sub = np.zeros(6)

    for i in range(len(dispVec)):
        disp_add = copy.deepcopy(dispVec)
        disp_add[i] = disp_add[i] + delta

        disp_sub = copy.deepcopy(dispVec)
        disp_sub[i] = disp_sub[i] - delta

        f_add = tanBar.get_internal_force_ANS(ex, ey, disp_add, C, m)
        f_sub = tanBar.get_internal_force_ANS(ex, ey, disp_sub, C, m)

        for j in range(len(f_add)):
            K[j,i] = ( f_add[j] - f_sub[j] ) / ( 2 * delta )
    return K
```

Listing 32: Test of internal force vector for assumed natural strain

B.4 Python solver

B.4.1 Function assemble_k

```python
def assemble_k(mesh, setname, elemtype, emat, dispVec):
    '''
    Assembles system stiffness matrix K_sys in dense form for given mesh with
    defined domain-set

    :param mesh:    MeshIO object generated from .msh file
    :param setname: string, setname of the domain as defined in Gmsh e.g. 'domain'
    :param elemtype:integer, 0 for CSTh element, 1 for ANDES
    :param emat:    tuple, (E-modulus, Poisson)
    :return:        np.array (dofs, dofs), dense-matrix of the system stiffness
    '''
    ndofs_per_node = 2
    num_dof  = mesh.points.shape[0] * ndofs_per_node
    elements = util.extract_set(mesh=mesh, setname=setname, connectivity=True)

    k_sys = np.zeros((num_dof, num_dof))

    for indx in range(elements.shape[0]):
        ex, ey = extract_element_coords(mesh=mesh, elements=elements, indx=indx)
        dof_indx = extract_dof_indx(elements=elements, indx=indx)

        disp_element = dispvec[np.ix_(dof_indx)]
        if elemtype == 0:
            kt = tan.get_tangent_Stiffness_matrix_green(ex, ey, disp_element, C =
    const.C_glob(emat[0], emat[1]))
        elif elemtype == 1:
            kt = bar.get_tangent_stiffness_ANS_green(ex,ey, disp_element, C = const
    .C_glob(emat[0], emat[1]))
        k_sys[np.ix_(dof_indx, dof_indx)] += kt

    return k_sys

    return k_sys
```

Listing 33: Assembling of stiffness matrix

B.4.2 Function add_load

```python
def add_load(mesh, setname, load_vec, load_magnitude, dof):
    '''
    Add load to an existing load_vector by lumping uniformly over given set of
    nodes

    :param mesh:            MeshIO object of mesh generated by Gmsh
    :param setname:         string, name of the load set as defined in Gmsh e.g. '
    freeEnd'
    :param load_vec:        np.array (dofs,), existing load vector
    :param load_magnitude:  float, total load to be distributed among nodes in set
    :param dof:             integer, load direction
    :return:                np.array (dofs,), new load vector
    '''
    ndofs_per_node = 2
    nodes = extract_set(setname=setname, mesh=mesh, connectivity=False)
    nodal_load = load_magnitude / nodes.__len__()
    for node in nodes:
        indx = int((node * ndofs_per_node) + dof)
        load_vec[indx] += nodal_load

    return load_vec
```

Listing 34: Load added to an existing load vector

B.4.3 Function extract_set

```python
def extract_set(mesh, setname, connectivity=True):
    '''
    Returns element connectivity array of given set
    Array data = node IDs
    If connectivity = False, returns list of node IDs stripped of duplicates

    :param mesh:            MeshIO object generated from .msh file
```

```
    :param setname:         string, setname as defined in Gmsh e.g. 'domain'
    :param connectivity:    boolean, if True gives connectivity array
    :return:                np.array or list, nodal IDs in given set
    '''

    loc = None

    for i in range(len(mesh.cell_sets[setname])):
        if mesh.cell_sets[setname][i].size:
            loc = i
            print('Found cells (elements) with setname =', setname, 'at location',
    loc)

    if loc is not None:
        if connectivity:
            return mesh.cells[loc].data
        else:
            return list(set(mesh.cells[loc].data.flatten()))
    else:
        print('Found no cells (elements) with setname =', setname)
        raise LookupError
```

Listing 35: Extract sets from the mesh file

B.4.4 Function set_fixed_dofs

```
def set_fixed_dofs(mesh, setname, k_sys, rhs, dofs=(0, 1, 2, 3, 4, 5)):
    '''
    Sets constrains in dense matrix of K_sys

    :param mesh:    MeshIO object of mesh generated by Gmsh
    :param setname: string, name of the constraint set as defined in Gmsh e.g. '
    fixedEnd'
    :param k_sys:   np.array (dofs, dofs)
    :param dofs:    tuple, components to be contrained for all nodes in set
    :return:        np.array (dofs, dofs), constrained system striffness matrix
    K_sys
    '''
    ndofs_per_node = 2
    nodes = extract_set(setname=setname, mesh=mesh, connectivity=False)
    for node in nodes:
        for dof in dofs:
            indx = int((node * ndofs_per_node) + dof)
            k_sys[indx, :] = 0.0
            k_sys[:, indx] = 0.0
            k_sys[indx, indx] = 1.0
            rhs[indx] = 0.0

    return k_sys
```

Listing 36: Set fixed dofs for the model

B.4.5 Function extract_element_coords

```
def extract_element_coords(mesh, elements, indx):
    '''
    Retrives the element coordinates in form np.array(3,) as used by other
    functions

    :param mesh:        MeshIO object of mesh generated by Gmsh
    :param elements:    np.array(n, 3), connectivity array of elements
    :param indx:        integer, which element in the connectivity array to return
    the coordinates of
    :return:            tuple (np.array (3,), np.array (3,), element global
    coordinates
    '''
    elem = elements[indx]
    i, j, k = elem[0], elem[1], elem[2]
    ecoords = np.array([mesh.points[i], mesh.points[j], mesh.points[k]])
    ex = ecoords[0:, 0]
    ey = ecoords[0:, 1]
```

```
15
16      return ex, ey
```

Listing 37: Extracts element coordinates from mesh

B.4.6 Function extract_nodal_coords

```
def extract_nodal_coords(mesh, nodeID):
    '''
    Helper function to extract nodal coordinates given a mesh and a nodeID

    :param mesh:    MeshIO object of mesh generated by Gmsh
    :param nodeID:  integer, node to retrieve
    :return:        tuple (x,y,z), global coordinates
    '''
    ncoords = mesh.points[nodeID]
    x = ncoords[0]
    y = ncoords[1]
    z = ncoords[2]

    return x, y, z
```

Listing 38: Extracts nodal coordinates from mesh

B.4.7 Function extract_dof_indx

```
def extract_dof_indx(elements, indx):
    '''
    Helper function to extract the global dof indexes given an element connectivity
     array

    :param elements:    np.array(n, 4), connectivity array of elements
    :param indx:        integer, which element in the connectivity array to return
    the global dof indexes of
    :return:            integer (6,) [ix,iy,jx,jy,kx,ky]
    '''
    elem = elements[indx]
    i, j, k = elem[0], elem[1], elem[2]

    dof_indx = np.array([
        i * 2, i * 2 + 1,
        j * 2, j * 2 + 1,
        k * 2, k * 2 + 1 ])

    return dof_indx
```

Listing 39: Extracts index of the degree of freedom

B.5 Plot for Seth-Hill strain

B.5.1 Topology

```
ex = np.array([0.,1.,0.5])
ey = np.array([0.,0.,1.])

th = 0.1
ep = [1,th]

E  = 2.1e11
nu = 0.3

D = np.array([
        [ 1.0,  nu,  0.],
        [  nu, 1.0,  0.],
        [  0.,  0., (1.0-nu)/2.0]]) * E/(1.0-nu**2)


dispVecEpsX    = np.array([[0.0],[0.0],[0.1],[0.0],[0.5],[0.0]])
dispVecEpsY    = np.array([[0.0],[0.0],[0.0],[0.0],[0.0],[0.1]])
```

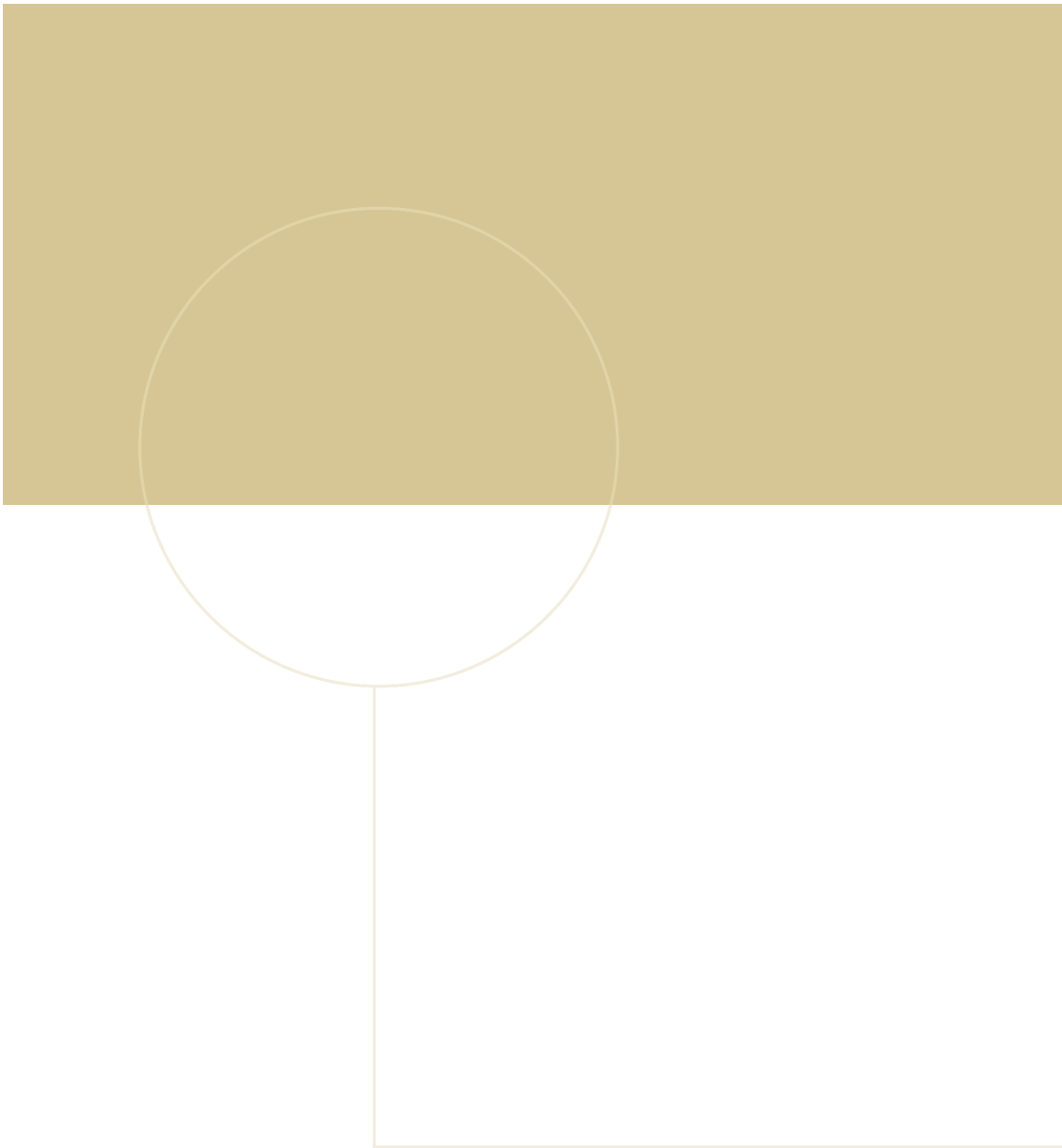
```
dispVecGammaXY = np.array([[0.0],[0.0],[0.0],[0.0],[0.1],[0.0]])
```

Listing 40: node and displacement coordinates

B.5.2 Strain comparison plot

```
dispList            = []
eps_greenList       = []
eps_engList         = []
eps_henckyList      = []
eps_swaingerList    = []
eps_almansiList     = []

for i in range(-400,1000):

    dispX        = i * dispVecEpsX * 0.001

    green_eps    = tri3.get_GreenStrainVec(ex,ey,dispX)
    eps          = tri3.getStrainVec_linear(ex,ey,dispX)
    hencky_eps   = tri3.get_HenckyStrain(ex,ey,dispX)
    swainger_eps = tri3.get_SwaingerStrain(ex,ey,dispX)
    almansi_eps  = tri3.get_AlmansiStrain(ex,ey,dispX)

    eps_greenList.append(green_eps[0])
    eps_engList.append(eps[0])
    eps_henckyList.append(hencky_eps[0])
    eps_swaingerList.append(swainger_eps[0])
    eps_almansiList.append(almansi_eps[0])

    dispList.append(i*0.001)

plt.figure(figsize=(20,10))
plt.plot(dispList, eps_engList,color= 'orange',linestyle= '-',label=r'Natural
    strain')
plt.plot(dispList, eps_greenList, color= 'green',linestyle= '-',label=r'Green
    strain')
plt.plot(dispList, eps_biotList,color= 'red',linestyle= '-',label=r'Biot strain')
plt.plot(dispList, eps_henckyList,color= 'black',linestyle= '-',label=r'Hencky
    strain')
plt.plot(dispList, eps_swaingerList,color= 'blue',linestyle= '-',label=r'Swainger
    strain')
plt.plot(dispList, eps_almansiList, color= 'pink',linestyle= '-',label=r'Almansi
    strain')

plt.title('Seth-Hill strain')
plt.xlabel('\lambda = L/L0', fontsize=10)
plt.grid(True)
plt.axhline(y=0, color='black')
plt.axvline(x=0, color='black')
plt.ylabel('Strain', fontsize=10)
plt.legend()
plt.show()
```

Listing 41: Set up plot for strain



NTNU

Norwegian University of
Science and Technology