NTNU – Trondheim
Norwegian University of
Science and Technology

Accumulated Plastic Strain Program

Lars Magnus Valnes

Master of Science in Physics and Mathematics
Submission date: June 2014
Supervisor: Alex Hansen, IFY

Norwegian University of Science and Technology
Department of Physics

Abstract

In this thesis we will look at the implementation and results of the APS program. The program computes the accumulated plastic strain in a umbilical tube for the deformations axial tension, bending and internal pressure. The computation is based on solving incremental problems with the finite element method and the return-mapping algorithm. The return-mapping algorithm encountered convergence failure for large strain increments, this was solved by using the strain increment of $1.0e^{-4}\%$. The implementation of axial tension and bending was consistent with the results provided by Nexans Norway. While the plastic behaviour of the internal pressure proved to be more complex, so the implementation requires improvement.

Sammendrag

I denne avhandlingen skal vi se på implementasjon og resultater av APS programmet. Programmet kalkulere den akkumulerte plastiske forvrengningen i et umbilical rør for deforamsjonen aksial strekk, bøyning og internt trykk. Utregning er gjort ved løse inkrementelle problemer med element method og return-mapping algoritmen. Return-mapping algoritmen hadde convergens problemer for store inkrement, dette ble løst ved å bruke strain inkrement $1.0e^{-4}\%$. Implementasjonen av aksial strekk og bøyning var i samsvar med de resultatene som ble gitt av Nexans Norway. Mens den plastiske oppførselen av internt trykk var mer kompleks og krever en forbedring av implementasjonen.

Preface

The goal was to implement a fully functional program to estimate the accumulated plastic strain caused by a series of deformations. The implementation encountered technical issues, concerning the return-mapping algorithm, computational time and Neumann conditions for internal pressure. The program is not fully tested and errors and bugs may occur.
I would like to thank my supervisor at Nexans Norway, Ph.d Magnus Komperoed, for the assistance and introduction to continuum mechanics. I would also like to thank Prof. Alex Hansen for being my supervisor at NTNU.

Lars Magnus Valnes, June 2014

Contents

Nomenclature

APS	the accumulated plastic strain [1]
APS_a	the accumulated plastic strain estimated with analytical approximation [1]
APS_d	the accumulated plastic strain for a dynamic mesh [1]
APS_s	the accumulated plastic strain for a static mesh [1]
A^*	the computed cross-sectional area $[m^2]$
A_t	the theoretical cross-sectional area $[m^2]$
a	the inner radius of the tube [m]
$a(\cdot, \cdot)$	the bilinear form
b	the outer radius of the tube [m]
c	the ratio between $\dot{\beta}$ and $\dot{\epsilon}^p$ [Pa]
$\mathbf{D}$	the elastic relation between stress and strain [Pa]
$\mathbf{D}_{ep}$	the tangent stiffness [Pa]
$\mathrm{d}\gamma$	the boundary integrand for the domain Ω
$\mathrm{d}\Omega$	the integrand for the domain Ω
$\mathrm{d}A$	the integrand for the domain A
E	the Young modulus [Pa]
e_{ij}	deviatoric strain tensor [1]
$\dot{f}$	the rate of the yield function
f	the yield function
F	the stress dependent part of the yield function
$\dot{\mathbf{f}}$	the force rate on the domain Ω [N]

$\mathbf{f}$ the force on the domain Ω [N]

$\mathbf{f}_\sigma$ the stress derivative of the yield function

$\mathbf{f}_\beta$ the back stress derivative of the yield function

f_k the hardening term derivative of the yield function

$\dot{\mathbf{g}}$ the boundary force on the domain Ω [N]

$\mathbf{g}_\sigma$ the stress derivative of the flow potential

H the plastic modulus or work-hardening modulus [Pa]

$\mathbf{h}$ the vector function that determines the internal variable rate

h_n the function that determines the internal variable rate for component n

J_2 the second invariant of a deviatoric tensor $\left[\text{Pa}^2\right]$

$\mathbf{J}$ the Jacobian matrix for the residual vector $\mathbf{r}$

k the hardening term of the yield function

$k*$ the part of the hardening term independent of the plastic multiplier

L the length of the mesh [m]

l indicator for the linear approximation $\left[{}^{faces}/_{\text{mm}}\right]$

$l\left(\cdot\right)$ the linear form

$\hat{n}$ the surface normal of the domain Ω [1]

$\mathbf{P}$ projection matrix for deviatoric stress [1]

p the internal pressure [Pa]

p_y the internal pressure corresponding to the yield strength σ_y, [Pa]

$\mathbf{r}$ the different different residuals represented as a vector

r_A the ratio of the approximated and the theoretical cross-sectional area. [1]

$r_{\Delta\lambda}$ the scalar residual for the plastic multiplier, equivalent with the yield function

$\mathbf{r}_{\boldsymbol{\xi}}$ the residual for the internal variables

$r_{\bar{\varepsilon}^p}$ the scalar residual for the equivalent plastic strain [1]

$\mathbf{r}_\sigma$ the residual vector for stress [Pa]

s_{ij} the deviatoric stress tensor [Pa]

SMYS Nexans' technical term for yield strength [Pa]

T the axial force [N]

$\mathbf{u}$ the displacement vector [m]

u_r the radial displacement[m]

V a function space

V_h a projection of the function space V

$\mathbf{v}$ the test function, equal to the displacement vector[m]

α used to constructing the tubes end conditions [1]

β the back stress [Pa]

Δp the incremental internal pressure [Pa]

Δu_r the incremental radial displacement [m]

$\Delta\lambda$ increment of the plastic multiplier

$\Delta\epsilon$ the strain increment [1]

Δt the time increment [s]

ϵ_{ij} the infinitesimal strain tensor [1]

ϵ^e_{ij} the elastic strain tensor [1]

ϵ^p_{ij} the plastic strain tensor[1]

$\bar{\epsilon}$ the equivalent strain [1]

$\bar{\epsilon}^p$ the equivalent plastic strain [1]

$\dot{\bar{\epsilon}}$ the equivalent plastic strain rate [s^{-1}]

$\boldsymbol{\epsilon}$ the Voight notation of the strain tensor [1]

$\boldsymbol{\epsilon}^p$ the Voight notation of the plastic strain tensor [1]

ϵ_z the strain in z-direction i.e. the axial direction [1]

ϵ_0 the strain in z-direction caused by an axial force [1]

κ the hardening variable, dimensions are dependent on the definition.

κ_x the curvature projection onto the y-axis and the curvature axis is directed along the x-axis [m^{-1}]

κ_y the curvature projection onto the x-axis and the curvature axis is directed along the y-axis [m^{-1}]

λ the first Lamé coefficient [Pa]

$\dot{\lambda}$ the plastic multiplier or Lagrange multiplier

μ the second Lamé coefficient [Pa]

ν the Poisson's ratio [1]

Ω the domain which the partial differential equation is defined plastic multiplier

ϕ_i a set of basis functions

ρ nodal density $\left[{}^{nodes}/_{\mathrm{mm}^3}\right]$

σ_{ij} the Cauchy stress tensor [Pa]

$\boldsymbol{\sigma}$ the Voight notation of the stress tensor [Pa]

$\boldsymbol{\sigma}^t$ the trial stress in the return-mapping algorithm [Pa]

σ_y the yield strength [Pa]

σ_r the radial stress, known as Hoop stress [Pa]

σ_θ the circumferential stress. Units [Pa]

σ_z the axial stress [Pa]

σ_z^* the computed axial stress [Pa]

Ξ the consistent tangent moduli [Pa]

$\boldsymbol{\xi}$ the internal variable [1]

Chapter 1

Introduction

The construction and transportation of umbilical tubes require that the tube is deformed in several processes. These processes will each generate an amount of plastic and elastic strain.
The absolute value of the plastic strain will contribute to the hardening of the tube, and continued hardening will eventually cause fracture. Therefore the accumulation of plastic strain (APS) is used as an indicator for the hardening. Defined as

$$\text{APS} = \sum_{i=steps} |\epsilon^p|_i, \tag{1.1}$$

where ϵ_i^p is the plastic strain generated in process i. The use of the accumulated plastic strain is quite common and can be seen in [9].
In this thesis we will give a representation of the Accumulated Plastic Strain Program. The program is developed at the request of Nexans Norway, for analysing the accumulation of plastic strain.
We will use the finite element method with linear Lagrange elements to compute the total strain for each deformation. The finite element method is used in many different problems, involving all from fluid mechanics to electromagnetism. For this reason, there are many commercial and open-source programs using the finite element method. We will construct the program using the FEniCS project [5].
The amount of plastic strain will be determined by the return-mapping algorithm, which is used in problems involving plastic and elastic strain, [1], [2] and [9].
The program is constructed with a graphic user interface for easier handling of different input parameters. The graphic user interface has two main functions; to specify the material and dimensional data, and to specify the deformation processes, including the order.
The program is made to handle three separate deformations, which are axial tension, bending and internal pressure. For these, the input parameters are respectively the axial force, curvature, and the internal pressure. The curvature input requires that the curvature projections in both plane axis are specified.
We will in the implementation specify a piecewise linear stress-strain relation. This will make it possible to approximate the APS value analytically, so that we may

asses the computed results. The results will also be compared with Nexans' current APS program.

Chapter 2

Theory

2.1 General

The theory presented here is based on the theory presented in [10], which consists of elastic and finite element theory. We will in this thesis focus on the theory of plasticity found in [6]. The theory also extends to some articles and books, which will be referred to in the appropriate sections.
The theory will consist of some important parts of plasticity, which may not be included in the implementation. However, it will give an assessment to the limitations and future improvements of the program. We assume isothermal conditions, therefore the thermal dependence is not included.
We will continue with the tube orientation defined in [10], which places the tube in a Cartesian coordinate system where z-axis aligns with the axial direction, shown in Fig.2.1. So that a second-rank tensor

$$T_{ij} = \begin{bmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{bmatrix}, \tag{2.1}$$

will have the orientation, where T_{11} points in the x-direction, T_{22} points in the y-direction and T_{33} points in the z-direction.

2.1.1 Elastic Summary

The theory of plasticity in this thesis is a continuation of the elastic theory, given in [10], thus we will give a brief summary of definitions, equations and notations.
The notation of tensor is $T_{ij..kl}$, where the number of subscripts indicates the tensor's rank. While matrices are denoted as $\mathbf{M}$, and vectors are given with the notation $\mathbf{v}$.
The operator ∇ will be defined as

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x}, & \frac{\partial}{\partial y}, & \frac{\partial}{\partial z} \end{bmatrix} \quad . \tag{2.2}$$

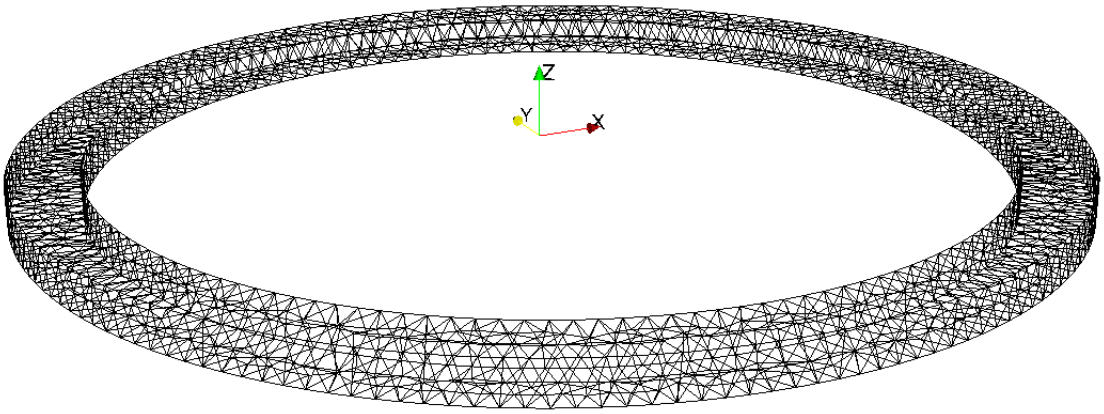


Figure 2.1: Shows the tube orientation in the Cartesian coordinate system.

We will give a short introduction to the definitions stress and strain, for more detailed introduction see [6]. The stress represents the pressure on the surfaces of a 3-dimensional element, which is subject to an external force. The stress has dimension [Pa] and produces a 3x3 second-rank tensor

$$\sigma_{ij} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix}. \tag{2.3}$$

The strain is the distortion of the element due to an external force. It is dimensionless and is constructed by the gradient of the displacement. The displacement is a vector field given as

$$\mathbf{u} = \begin{bmatrix} u_x, & u_y, & u_z \end{bmatrix}, \tag{2.4}$$

and the gradient is defined as

$$\nabla u = \begin{bmatrix} \frac{\partial}{\partial x}, & \frac{\partial}{\partial y}, & \frac{\partial}{\partial z} \end{bmatrix}^T \begin{bmatrix} u_x, & u_y, & u_z \end{bmatrix}. \tag{2.5}$$

The infinitesimal strain tensor is constructed with

$$\epsilon_{ij} = \frac{1}{2}\left((\nabla \mathbf{u})^T + \nabla \mathbf{u}\right), \tag{2.6}$$

which will produce a symmetric 3x3 second-rank tensor

$$\epsilon_{ij} = \begin{bmatrix} \epsilon_{11} & \epsilon_{12} & \epsilon_{13} \\ \epsilon_{21} & \epsilon_{22} & \epsilon_{23} \\ \epsilon_{31} & \epsilon_{32} & \epsilon_{33} \end{bmatrix}. \tag{2.7}$$

The stress and strain tensors are symmetric and can therefore be presented as a 6 dimensional vector, such as

$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ 2\epsilon_{23} \\ 2\epsilon_{13} \\ 2\epsilon_{12} \end{bmatrix} \qquad \sigma = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{23} \\ \sigma_{13} \\ \sigma_{12} \end{bmatrix}. \tag{2.8}$$

This is referred to as Voight notation and it is important to note the factor 2 for the shear strain.
With the Voight notation the linear elastic relation is given as

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\epsilon}, \tag{2.9}$$

with the matrix $\mathbf{D}$ defined as

$$\mathbf{D} = \begin{bmatrix} \lambda + 2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda + 2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda + 2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix}. \tag{2.10}$$

The λ and μ are known as the first and second Lamé coefficients, defined as

$$\lambda = \frac{E\nu}{(1-2\nu)(1+\nu)} \qquad , \qquad \mu = \frac{E}{2(1+\nu)}. \tag{2.11}$$

With E as the Young Modulus and ν as the Poission's ratio, definitions can be found in [6].
Plasticity often involves deviatoric stress and strain. The deviatoric strain is defined as

$$e_{ij} = \epsilon_{ij} - \delta_{ij}\frac{1}{3}\epsilon_{kk}, \tag{2.12}$$

and the deviatoric stress as

$$s_{ij} = \sigma_{ij} - \delta_{ij}\frac{1}{3}\sigma_{kk}. \tag{2.13}$$

The tensor can also be presented as scalar known as equivalent stress and strain. The equivalent stress is defined as

$$\bar{\sigma} = \sqrt{\frac{3}{2} s_{ij} s_{ij}}, \tag{2.14}$$

and the equivalent strain as

$$\bar{\epsilon} = \sqrt{\frac{2}{3} e_{ij} e_{ij}}. \tag{2.15}$$

Which is similar to the second invariant of a deviatoric tensor defined as

$$J_2 = \frac{1}{2} T_{ij} T_{ij}. \tag{2.16}$$

This will conclude the brief summary, and see [10] for a more detailed presentation.

2.1.2 Plastic Notations

We will also include some new notations, such as the step function, defined as

$$< \theta >= \begin{cases} \theta, & \text{if } \theta \geq 0 \\ 0, & \text{if } \theta < 0 \end{cases}. \tag{2.17}$$

We will continue to use the Voight notation in this thesis, therefore it is convenient to introduce the matrix $\mathbf{P}$. Defined as

$$\mathbf{P} = \frac{1}{3} \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{bmatrix}, \tag{2.18}$$

with the matrix $\mathbf{P}$, the transformation to deviatoric stress can be written as

$$\mathbf{s} = \mathbf{P}\boldsymbol{\sigma}. \tag{2.19}$$

The matrix $\mathbf{P}$ is a symmetric and unitary with the properties

$$\begin{aligned} \mathbf{P}^T &= \mathbf{P} \\ \mathbf{P} \cdot \mathbf{P} &= \mathbf{P} \end{aligned}. \tag{2.20}$$

So that the second invariant of the deviatoric stress tensor Eq.2.16 can be rewritten to

$$J_2 = \frac{1}{2} \boldsymbol{\sigma}^T \mathbf{P}^T \mathbf{P} \boldsymbol{\sigma} = \frac{1}{2} \boldsymbol{\sigma}^T \mathbf{P} \boldsymbol{\sigma}. \tag{2.21}$$

In the modelling of plasticity we will use derivatives, which will have the following notation

$$\begin{aligned} \frac{\partial F}{\partial t} &= \dot{F} \\ \frac{\partial F}{\partial \sigma_{ij}} &= \mathbf{F}_{\sigma} \end{aligned}, \tag{2.22}$$

where F is an arbitrary function. The stress and strain derivatives of a scalar function is a symmetric second-rank tensor, and will therefore be given in Voight notation. This requires some structuring of partial derivation, and will acquire the form of

$$\dot{F} = \mathbf{F}_{\sigma}^{T} \dot{\boldsymbol{\sigma}}. \tag{2.23}$$

2.2 Plasticity

In this section we will give a general introduction to plasticity, with the theory taken from [6]. Plasticity covers different types of inelastic behaviour, among them are viscoplasticity and elastic-plastic. The former is also known by other terms, but it covers the combination of elastic and plastic deformation. We will focus on elastic-plastic problems, which can be seen as the limiting case of viscoplasticity as the viscosity goes to zero.
Plasticity constitutes a non-linear relation between stress and strain. The non-linear relation indicates that there are more parameters effecting the strain, than the stress and temperature. This prompted the notion of material memory, meaning a dependency on past events. This material memory dependence can be seen in the rate sensitivity. For example, stress which is applied slowly produces different and mostly greater displacement than stress that is applied rapidly.

2.2.1 Plastic Effects

There are some plastic effects, which we will address in this section. The first effect is known as Bauschinger effect and is observed when the strain direction is reversed. For instance from extension to compression, and it will cause a lowering of the yield strength. The modelling of the Bauschinger effect requires the use of back stress, which we will introduce later in this thesis.
The material can also experience the effect of recovery, which is a spontaneous softening of the yield surface, i.e a decrease in yield strength. The effect is slow and decreases along with temperature. For room temperature (300 K), the recovery may be neglected for steel, copper and aluminium.
There are more plastic effects, but these effects are more related to temperature dependence, thus not included, see [6] for more details.

2.3 Tangent Stiffness

For infinitesimal strain theory, the total strain can be decomposed additively as

$$\epsilon_{ij} = \epsilon_{ij}^{p} + \epsilon_{ij}^{e}, \tag{2.24}$$

with superscript p and e, respectively denoting plastic and elastic strain. This assumption is essential in the implementation and the results of the program. The elastic rate relation can be constructed using Eq.2.9, which will give

$$\dot{\boldsymbol{\sigma}} = \mathbf{D}\dot{\boldsymbol{\epsilon}}^{e}. \tag{2.25}$$

By inserting Eq. 2.24 into Eq. 2.9 we obtain

$$\dot{\boldsymbol{\sigma}} = \mathbf{D}\left(\dot{\boldsymbol{\epsilon}} - \dot{\boldsymbol{\epsilon}}^{p}\right). \tag{2.26}$$

It is common to write the equation as

$$\dot{\boldsymbol{\sigma}} = \mathbf{D}_{ep}\dot{\boldsymbol{\epsilon}}, \tag{2.27}$$

where the matrix $\mathbf{D}_{ep}$ is called the tangent stiffness moduli.

2.4 Plasticity Modelling

The theory behind plasticity behaviour is not fully understood, therefore we will look at a generalized scheme of modelling. The generalized scheme is based on three functions, known as yield function, flow rule and hardening rule. These functions will be implemented into the program, thus we will look at each function individually.
To manage these functions properly, internal variables were introduced to govern the local plastic behaviour. The internal variables are denoted as a vector $\boldsymbol{\xi}$ with N components. We will treat each component in $\boldsymbol{\xi}$ as a scalar, but in some models they are second-rank tensors.
The internal variables are, like other parameters in the plastic model, determined by a rate equation. The rate equation for component n is

$$\dot{\xi}_n = h_n^{*}\left(\sigma_{ij}, \boldsymbol{\xi}\right). \tag{2.28}$$

For simplifications later on, we will use

$$\dot{\xi}_n = \phi h_n\left(\sigma_{ij}, \boldsymbol{\xi}\right), \tag{2.29}$$

with ϕ as a scalar function.
The internal variables can be considered auxiliary variables, which are replaced in the plastic model for known parameters.

2.4.1 Flow Rule

The flow rule determines the plastic strain rate, defined as

$$\epsilon_{ij}^{p} = g_{ij}\left(\sigma, \xi\right). \tag{2.30}$$

Here the function g_{ij} given as

$$g_{ij} = \phi \sum_{n}^{N} \frac{\partial \epsilon^{p}}{\partial \xi_{n}} h_{n}. \tag{2.31}$$

Based on the Eq.2.29, we assume that there exists a function g, so that

$$g_{ij}\left(\sigma, \xi\right) = \phi \frac{\partial g}{\partial \sigma_{ij}}. \tag{2.32}$$

This function g is known as the flow potential or plastic potential. In this way the plastic strain rate can be written as

$$\dot{\epsilon}_{ij}^{p} = \phi \frac{dg}{d\sigma_{ij}}. \tag{2.33}$$

The use of a potential is similar with that in a fluid flow, and in each model the potential is specified. We can also construct a general potential with Helmholtz free energy, but we will not include this in the thesis, see [6]. The plastic strain rate also determines the accumulation of plastic strain, thus it is a crucial part of the implementation.

2.4.2 Yield Function

The transition from elastic to plastic strain is marked by the yield surface, thus the yield function is constructed to give an indication of yield surface, and is denoted $f\left(\boldsymbol{\sigma}, \boldsymbol{\xi}\right)$. For $f < 0$ the stress is inside the elastic range, while $f > 0$ gives the stress in the plastic range. The case of $f = 0$, marks the yield surface.
The yield surface can be subjected to hardening, which is determined by the yield functions dependence on the internal variables. Under constant stress, the rate of the yield function becomes

$$\dot{f}\Big|_{\sigma = const} = \phi \sum_{n}^{N} \frac{\partial f}{\partial \xi_{n}} h_{n}. \tag{2.34}$$

From which the work-hardening modulus H can be defined as

$$H = -\sum_{n}^{N} \frac{\partial f}{\partial \xi_{n}} h_{n}. \tag{2.35}$$

The hardening properties of the material can be determined by the work-hardening modulus. The cases of $H < 0$ and $H = 0$, are respectively known as softening

and perfect plastic. These cases have little relevance in this thesis and will not be further explained. We will only consider the case of $H > 0$, which constitutes a work-hardening material.
The dynamic behaviour of the yield surface can be assessed by taking the time derivative, which can be presented as

$$\dot{f} = \frac{df}{d\sigma_{ij}}\dot{\sigma_{ij}} + \sum_k \frac{\partial f}{\partial \xi_n} h_n. \tag{2.36}$$

The last term is known from Eq.2.35 as ϕH, so we can write it as

$$\dot{f} = \mathbf{f}_\sigma \dot{\boldsymbol{\sigma}} - H\phi. \tag{2.37}$$

The generation of plastic strain occurs while the yield function is expanding for a stress state on the yield surface, i.e. $\dot{f} > 0$ and $f = 0$. For $H > 0$, we can see in Eq.2.37 that this can only happen for $\mathbf{f}_\sigma \dot{\boldsymbol{\sigma}} > 0$, which is known as loading. The case of $\mathbf{f}_\sigma \dot{\boldsymbol{\sigma}} < 0$ is called unloading, which generates no plastic strain. This prompts that the expression of the plastic strain includes a step function.

2.4.3 Hardening Rules

It is common for the yield function to be decomposed as

$$f(\boldsymbol{\sigma}, \boldsymbol{\xi}) = F(\boldsymbol{\sigma}) - k(\boldsymbol{\xi}). \tag{2.38}$$

The hardening behaviour of the material is determined by $k(\boldsymbol{\xi})$, where the internal variables determine the hardening variable.
The hardening variable κ can either be defined by equivalent plastic strain or inelastic work. The inelastic work definition yields

$$\kappa = \int \sigma_{ij} \dot{\epsilon}_{ij} \mathrm{d}t, \tag{2.39}$$

while the equivalent plastic strain definition is

$$\kappa = \int \sqrt{\frac{2}{3}\dot{\epsilon}^p_{ij}\dot{\epsilon}^p_{ij}}\mathrm{d}t. \tag{2.40}$$

The factor of $^2/_3$ was introduced to scale the expression. So that a strain tensor given as

$$\dot{\epsilon}_{ij} = \begin{bmatrix} \dot{\epsilon}^p & 0 & 0 \\ 0 & -\frac{1}{2}\dot{\epsilon}^p & 0 \\ 0 & 0 & -\frac{1}{2}\dot{\epsilon}^p \end{bmatrix}, \tag{2.41}$$

would give

$$\dot{\kappa} = \sqrt{\frac{2}{3}\dot{\epsilon}^p_{ij}\dot{\epsilon}^p_{ij}} = |\dot{\bar{\epsilon}}^p| \tag{2.42}$$

We will use the equivalent plastic strain definitions in the implementation, but the definitions are often equivalent.

Accumulated Plastic Strain

We have stated that the accumulated plastic strain can indicate the hardening of the material. Using implicit Euler on Eq.2.42, we obtain

$$\kappa_{t+1} = \kappa_t + |\bar{\epsilon}^p|_t. \tag{2.43}$$

Given that the initial condition $k_0 = 0$, we can write

$$\kappa_t = \sum_t |\bar{\epsilon}^p|_t. \tag{2.44}$$

Which is the same as the one dimensional expression of the accumulated plastic strain, given as

$$APS = \sum_i |\epsilon_i^p|. \tag{2.45}$$

Thus the hardening rule gives the relation between the APS and the hardening in the material. The term of accumulated plastic strain is often known as equivalent plastic strain, due to the definition of the hardening variable.

Isotropic Hardening

Isotropic hardening denotes that the boundary is symmetrically expanded. This occurs as the hardening variables increase in value, causing $k(\xi)$ and the yield strength to increase. The increase is determined by the hardening rule, which for linear hardening is given as

$$k(\kappa) = \sigma_y^0 + H\kappa, \tag{2.46}$$

with σ_y^0 as the initial yield strength.

Kinematic Hardening

Kinematic hardening was introduced for taking into account effects like Bauschinger, which can not be explained by isotropic hardening.
The kinematic hardening gives a local change in the yield surface, making the yield surface non-symmetric. This is done by using back stress, which can be seen as plastic strain memory. The back stress will be denoted $\boldsymbol{\beta}$, and it is inserted into Eq.2.38, yielding

$$f(\boldsymbol{\sigma}, \boldsymbol{\beta}, \xi) = F(\boldsymbol{\sigma} - \boldsymbol{\beta}) - k(\xi). \tag{2.47}$$

Following the Melan-Prager model the back stress is proportional to plastic strain

$$\dot{\boldsymbol{\beta}} = c\dot{\boldsymbol{\epsilon}}^p, \tag{2.48}$$

which is the simplest model. In more sophisticated models, the back stress is non-linear.

2.5 Drucker's Postulate

The Drucker's postulate is connected to plasticity, and it has a stricter definition of work-hardening than presented earlier.
The postulate defines a work-hardening plastic material as one where the work during incremental loading is positive and work done in loading -unloading cycle is non-negative. This can be shown as an inequality

$$\dot{\boldsymbol{\sigma}}\dot{\boldsymbol{\epsilon}}^p \geq 0, \tag{2.49}$$

known as Drucker's inequality, for details see [6].
The inequality also holds for any two different stress states, which we will denote σ and σ^*. We set that σ is on the yield surface and σ^* inside the elastic domain. So that the inequality can be written as

$$(\boldsymbol{\sigma} - \boldsymbol{\sigma}^*)\,\dot{\boldsymbol{\epsilon}}^p \geq 0. \tag{2.50}$$

The yield function is considered smooth, so every point has a tangential plane. Then it follows from Eq. 2.50, that σ^* is constrained by the tangential plane, thus making the yield function convex, In case of a nob convex surface, the inequality of Eq. 2.50 would be wrong, see Fig.2.3.

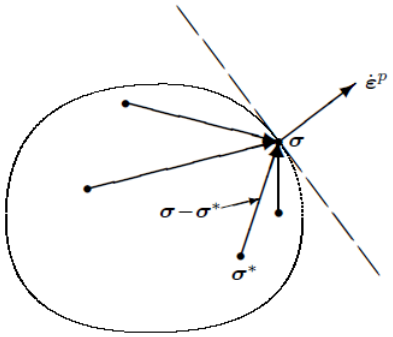


Figure 2.2: Shows that the plastic strain rate is normal to the tangential plane. Taken from [6].

The inequality in Eq.2.50 dictates that the plastic strain must be directed along the tangential plane normal. This is known as the normality rule and constitutes a relation between the flow rule and yield function, see Fig.2.2. In calculus the tangential plane normal can be shown to be proportional to the gradient of the function. Hence the normality rule indicates that the flow rule is associated with the yield criterion. This property makes the flow rule known as an associate flow rule, shown as

$$\mathbf{f}_\sigma \sim \mathbf{g}_\sigma. \tag{2.51}$$

It is often further simplified, by making them equal. Which we will do to simplify the implementation of the plastic model.

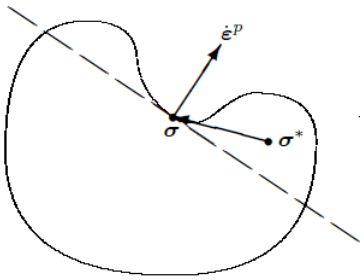


Figure 2.3: Shows a violation of the Eq.2.50, as the yield function must be convex. Taken from [6].

2.6 Rate Independent Plasticity

For sufficiently slow processes, compared to the material relaxation time, the transition to rate independent plasticity is possible. This means that the equations are no longer affected by changing the time scale. So we can multiply with a time increment Δt to obtain the incremental values. We will follow the procedure in [6]. The scalar function ϕ will from now on be denoted as $\dot{\lambda}$ and known as the Lagrange multiplier or plastic multiplier. We will insert the plastic multiplier into the rate equations, making them rate-independent. Thus the plastic strain rate becomes

$$\dot{\boldsymbol{\epsilon}}^p = \dot{\lambda}\mathbf{g}_\sigma, \tag{2.52}$$

and the internal variable rate becomes

$$\dot{\xi}_k = \dot{\lambda}h_k. \tag{2.53}$$

The hardening rate may be written in the general form of

$$\dot{k} = \frac{\partial k}{\partial \xi_n}\dot{\xi}_n = \dot{\lambda}k^*. \tag{2.54}$$

We will now obtain an expression for the plastic multiplier, which can be seen in [4]. The time derivative of the yield function can be presented with partial derivation as

$$\dot{f} = \mathbf{f}_\sigma^T\dot{\sigma} + \mathbf{f}_\beta^T\dot{\boldsymbol{\beta}} + f_k\dot{k}. \tag{2.55}$$

From Eq.2.47 we have that $\mathbf{f}_\beta = -\mathbf{f}_\sigma$. Inserting the rate-independent equitations into Eq.2.55 will give

$$\dot{f} = \mathbf{f}_\sigma^T\mathbf{D}\left(\dot{\boldsymbol{\epsilon}} - \dot{\lambda}\mathbf{g}_\sigma\right) - \dot{\lambda}\mathbf{f}_\sigma^T c\mathbf{g}_\sigma - \dot{\lambda}f_k k^*. \tag{2.56}$$

For a slow process the generation of plastic strain will occur close to the yield surface, i.e $f \to 0^+$. So that the plastic flow will keep the stress state on the yield surface, which means that the time derivative of the yield function is zero, thus we obtain

$$\dot{\lambda} = \frac{\mathbf{f}_\sigma^T\mathbf{D}\dot{\boldsymbol{\epsilon}}}{\mathbf{f}_\sigma^T\mathbf{D}\mathbf{g}_\sigma + \mathbf{f}_\sigma^T c\mathbf{g}_\sigma + f_k k^*}. \tag{2.57}$$

We see in Eq.2.52 that the generation of plastic strain is determined by the plastic multiplier. And we have seen that there are some requirements to the generation of the plastic strain, thus we must impose these requirements on the plastic multiplier. The first requirement is that the stress state is on the yield surface, which can be imposed by $f = 0$. While the second requirement is that we have loading, i.e. $\mathbf{f}_\sigma \dot{\sigma} > 0$. This expression is given in the nominator in Eq.2.57, and can therefore be enforced by a step function. This will generate an expression

$$\dot{\lambda} = \begin{cases} \frac{<\mathbf{f}_\sigma^T \mathbf{D}\dot{\boldsymbol{\epsilon}} - \dot{f}>}{\mathbf{f}_\sigma^T \mathbf{D}\mathbf{g}_\sigma + \mathbf{f}_\sigma^T c\mathbf{g}_\sigma + f_k k^*} & , f = 0 \\ 0 & , f < 0 \end{cases}. \tag{2.58}$$

The expression for the plastic multiplier is now known and can be used to determine the tangent stiffness moduli in Eq.2.27. We will use the expressions of Eq.2.52 and Eq.2.58 and insert them into Eq.2.26. This will yield

$$\dot{\sigma} = \left(\mathbf{D} - \frac{\mathbf{D}\mathbf{g}_\sigma \mathbf{f}_\sigma^T \mathbf{D}}{\mathbf{f}_\sigma^T \mathbf{D}\mathbf{g}_\sigma + c\mathbf{f}_\sigma^T \mathbf{g}_\sigma + f_k k^*} \right) \dot{\epsilon}, \tag{2.59}$$

for $\dot{\lambda} > 0$. Hence the tangent stiffness matrix form Eq.2.60 is determined as

$$\mathbf{D}_{ep} = \begin{cases} \mathbf{D}, & \dot{\lambda} = 0 \\ \mathbf{D} - \frac{\mathbf{D}\mathbf{g}_\sigma \mathbf{f}_\sigma^T \mathbf{D}}{\mathbf{f}_\sigma^T \mathbf{D}\mathbf{g}_\sigma + c\mathbf{f}_\sigma^T \mathbf{g}_\sigma + f_k k^*}, & \dot{\lambda} > 0 \end{cases}. \tag{2.60}$$

2.7 Return Mapping Algorithm

We will use the return mapping algorithm to map the stress onto the yield surface. The mapping occurs incrementally and is based on a predictor-corrector scheme. The incremental steps are generated by using implicit Euler on the rate equations, yielding

$$\boldsymbol{\epsilon}_{n+1}^p - \boldsymbol{\epsilon}_n^p = \Delta\lambda \mathbf{g}_\sigma, \tag{2.61}$$

for the plastic strain.

2.7.1 Prediction

The first step of the return mapping algorithm is the update of the strain increment

$$\boldsymbol{\epsilon}_{n+1} = \boldsymbol{\epsilon}_n + \boldsymbol{\Delta\epsilon}, \tag{2.62}$$

which is based on the incremental displacement. We use the updated strain increment to predict the stress, by using

$$\boldsymbol{\sigma}_{n+1}^t = \mathbf{D}\left(\boldsymbol{\epsilon}_{n+1} - \boldsymbol{\epsilon}_n^p\right), \tag{2.63}$$

where ϵ_n^p is plastic strain from the previous increment. The predicted stress is known as the trial stress, and will be marked by the superscript t. We can see that the prediction is based on Eq.2.26.
The next step is to insert the trial stress into the yield function Eq. 2.38, which has two possible outcomes either $f \leq 0$ or $f > 0$. The first case indicates that the trial stress is below the yield strength, so no corrections are required.
In case of present back stress, the back stress of the previous increment is also inserted into the yield function, see [9].

2.7.2 Correction

For $f > 0$, we have plasticity and therefore corrections are needed. The corrections are obtained by imposing that the yield function is zero and finding the corresponding plastic multiplier. We will discuss the procedure in more detail in the section 2.8 involving the closest-point projection. The plastic multiplier is used to correct the trial stress and update the other parameters, summarized as

$$\begin{aligned} \boldsymbol{\epsilon}_{n+1}^p &= \boldsymbol{\epsilon}_n^p + \Delta\lambda \mathbf{g}_\sigma \\ \boldsymbol{\xi}_{n+1} &= \boldsymbol{\xi}_n + \Delta\lambda \mathbf{h} \\ \boldsymbol{\sigma}_{n+1} &= \boldsymbol{\sigma}_{n+1}^t - \mathbf{D}\Delta\lambda \mathbf{g}_\sigma \\ \boldsymbol{\beta}_{n+1} &= \boldsymbol{\beta}_n + c\boldsymbol{\epsilon}_{n+1}^p. \end{aligned} \tag{2.64}$$

The corrections are also used in the construction of the tangent stiffness moduli Eq.2.27.

2.8 Closest-Point Projection

The calculation of $\Delta\lambda$ needs to take into account the differential changes of $\boldsymbol{\sigma}$ and ξ. This can be done with the closest-point projection. The differential problem to be solved can be summarized as

$$\frac{\partial \boldsymbol{\sigma}(\lambda)}{\partial \lambda} = -\mathbf{D}\mathbf{g}(\boldsymbol{\sigma}, \boldsymbol{\xi}), \qquad \frac{\partial \boldsymbol{\xi}(\lambda)}{\partial \lambda} = \mathbf{h}(\boldsymbol{\sigma}, \boldsymbol{\xi}). \tag{2.65}$$

With the constraint

$$f(\boldsymbol{\sigma}(\lambda), \boldsymbol{\xi}(\lambda)) = 0, \tag{2.66}$$

and initial conditions of $\boldsymbol{\sigma}^t, \boldsymbol{\xi}_0, 0$.
The problem is non-linear and can be solved with the Newton-Raphson method. We first construct the residuals, given as

$$\begin{aligned} \mathbf{r}_{\boldsymbol{\sigma}} &= \boldsymbol{\sigma}_{n+1} + \Delta\lambda \mathbf{D}\mathbf{g}_\sigma - \boldsymbol{\sigma}^t \\ \mathbf{r}_{\boldsymbol{\xi}} &= \boldsymbol{\xi}_{n+1} - \Delta\lambda \mathbf{h} - \boldsymbol{\xi}_n \\ r_{\Delta\lambda} &= f(\boldsymbol{\sigma}, \boldsymbol{\xi}) \end{aligned} \quad . \tag{2.67}$$

Here the residual subscripts refer to the variable, and must not be confused with derivatives. The solution to the problem is achieved when the equations in Eq.2.67 equal zero, and can be presented as a vector function

$$\mathbf{r}(\mathbf{x}) = 0. \tag{2.68}$$

Where $\mathbf{x} = \{\boldsymbol{\sigma}, \boldsymbol{\xi}, \Delta\lambda\}$. The first order expansion yields

$$\mathbf{r}(\mathbf{x}) = \mathbf{r}(\mathbf{x}_n) + \frac{\partial \mathbf{r}}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}_n}(\mathbf{x} - \mathbf{x}_n) \approx 0. \tag{2.69}$$

From which an iterative solution may be obtained with

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}. \tag{2.70}$$

And the increment is calculated based on

$$\Delta\mathbf{x} = -\mathbf{J}^{-1}(\mathbf{x}_n)\,\mathbf{r}(\mathbf{x}_n), \tag{2.71}$$

where $\mathbf{J}$ is the Jacobian matrix of $\mathbf{r}$.
There are known convergence difficulties regarding the Newton-Raphson method. For large enough increments, the method may not converge, and modifications have been purposed in [1] and [2].

2.9 Combined Stresses

The combination axial extension and bending, may occur simultaneously, which is known as combined stresses. The theory regarding bending can be found in [6]. In [10] we chose that the reference for the bending was the center of the tube. This gave that the axial strain for bending is

$$\epsilon_z = -\kappa_y x + \kappa_x y, \tag{2.72}$$

where κ_x is the curvature projection onto the y-axis and κ_y is the curvature projection onto the x-axis. The subscripts indicate the axis, that is directed in the same direction, as the curvature axis, which is normal to the projection.
Based on the theory in [6], the axial extension can be added to the equation, if the amount is sufficiently small. Which will yield

$$\epsilon_z = -\kappa_y x + \kappa_x y + \epsilon_0, \tag{2.73}$$

with ϵ_0 corresponding to the strain caused by axial extension.

2.10 Internal Pressure

The elastic stress in a tube with only internal pressure is derived in [6], and partial shown in [10]. The boundary conditions were

$$\sigma_r|_{r=a} = -p \qquad , \qquad \sigma_r|_{r=b} = 0, \tag{2.74}$$

with p denoting the inner pressure, and a and b respectively the tube's inner and outer radius. This yielded the radial and circumferential stress in cylindrical coordinates as

$$\begin{aligned} \sigma_r &= -\frac{p}{\left({}^{b}/_{a}\right)^2 - 1}\left(\frac{b^2}{r^2} - 1\right) \\ \sigma_\theta &= -\frac{p}{\left({}^{b}/_{a}\right)^2 - 1}\left(\frac{b^2}{r^2} + 1\right) \end{aligned}. \tag{2.75}$$

The axial stress component depends on the tube end conditions, and we considered closed and open-ended tubes. We obtain the axial stress in both cases, yielding

$$\sigma_z = \frac{(2\nu + \alpha)\, p}{\left({}^{b}/_{a}\right)^2 - 1}, \tag{2.76}$$

with α given as

$$\alpha = \begin{cases} 1 - 2\nu & \text{if closed-ended} \\ -2\nu & \text{if open-ended} \end{cases}. \tag{2.77}$$

Given that we know the stress components, we can find the corresponding strain components. Which in [6] were used to derive the elastic radial displacement

$$u\,(r) = \frac{(1-\nu)p}{E\left[(b/a)^2 - 1\right]}\left[(1-2\nu)\, r + \frac{b^2}{r}\right] - \nu\epsilon_z r, \tag{2.78}$$

and the axial strain

$$\epsilon_z = \frac{\alpha p}{E\left[(b/a)^2 - 1\right]}, \tag{2.79}$$

with E as the Young Modulus. We will now determine the pressure corresponding with the yield strength, which is done in [6]. The Mises yield criteria in cylindrical coordinates are given as

$$\sigma_r^2 + \sigma_\theta^2 + \sigma_z^2 - \sigma_r\sigma_\theta - \sigma_r\sigma_z - \sigma_\theta\sigma_z = 3\sigma_y^2. \tag{2.80}$$

In this equation the shear stresses are not included. Since we know the stress components, we can determine the internal pressure p_y corresponding with the yield criterion. This is done by inserting Eq.2.75 and Eq.2.76 into Eq.2.80, so that we obtain

$$p_y = \frac{\sigma_y\left(1 - \left(\frac{a}{b}\right)^2\right)}{\sqrt{1 + \frac{1}{3}\left(1 - 2\nu - \alpha\right)^2\left(\frac{a}{b}\right)^4}}. \tag{2.81}$$

We can see that the determination of the yield pressure did not require knowledge of the plastic behaviour in the tube. Thus we can asses Eq.2.81 by assuming elastic displacement.

2.11 The Finite Element Method

The finite element method consists of making a partial differential equation into a variational problem, which can be solved numerically.
The theory about finite element method is given in [8], while the theory in [10] covers what is necessary in constructing a variational problem by using linear Lagrange elements.
In this thesis, we will address the differences in the construction of the variational problem from that in [10]

2.11.1 Weak Formulation

In [10] the weak formulation was constructed by using the linear elastic relation Eq.2.25. We will now use the rate relation Eq.2.27 in the construction of the variational problem for the finite element method. We will assume quasi-static condition, so the partial differential equation is

$$\nabla \cdot \sigma_{ij} = \mathbf{f}. \tag{2.82}$$

The rate of Eq.2.82 is obtained by time derivation, yielding

$$\nabla \cdot \dot{\sigma}_{ij} = \dot{\mathbf{f}}. \tag{2.83}$$

The rate equation indicates a dependence on time, making the subspace temporal bounded. So to obtain the weak formulation we need to multiply a testfunction $\mathbf{v}$ for all $t > 0$. This is followed by the integration over the domain Ω, yielding

$$-\int_{\Omega} \nabla \cdot \dot{\sigma}_{ij}(\mathbf{u})\,\mathbf{v}\mathrm{d}\Omega = \int_{\Omega} \dot{\mathbf{f}}\mathbf{v}\mathrm{d}\Omega. \tag{2.84}$$

We will use Green's formula on the left-hand side of Eq.2.84 to obtain

$$-\int_{\Omega} \nabla \cdot \dot{\sigma}_{ij}(\mathbf{u})\,\mathbf{v}\mathrm{d}\Omega = \int_{\Omega} \dot{\sigma}_{ij}(\mathbf{u})\,\nabla\mathbf{v}\mathrm{d}\Omega - \int_{\partial\Omega} \dot{\sigma}_{ij}(\mathbf{u})\,\hat{\mathbf{n}}\mathbf{v}\mathrm{d}\gamma, \tag{2.85}$$

with $\partial\Omega$, $\mathrm{d}\gamma$ and $\hat{n}$ denoting the boundary surfaces, boundary integrand and surface normal. The symmetry of the stress tensor makes it possible to write

$$\sigma_{ij}(\nabla\mathbf{v})_{ij} = \sigma_{ij}\epsilon_{ij}(\mathbf{v}). \tag{2.86}$$

This is followed by the transformation to Voight notation, so that

$$\sigma_{ij}\epsilon_{ij} = \boldsymbol{\epsilon}^T\boldsymbol{\sigma}. \tag{2.87}$$

We can insert the relation in Eq.2.27 to obtain

$$\int_{\Omega} \boldsymbol{\epsilon}^T(\mathbf{v})\,\mathbf{D}_{ep}\dot{\boldsymbol{\epsilon}}(\dot{\mathbf{u}})\,\mathrm{d}\Omega = \int_{\Omega} \mathbf{v}^T\dot{\mathbf{f}}\cdot\mathrm{d}\Omega + \int_{\partial\Omega} \mathbf{v}^T\dot{\mathbf{g}}\mathrm{d}\gamma, \tag{2.88}$$

with $\dot{\mathbf{g}}$ as the stress at the boundary. The strain is determined by the displacement and using Eq.2.91, we have that

$$\dot{\epsilon}_{ij}(\mathbf{u}) = \epsilon_{ij}(\dot{\mathbf{u}}). \tag{2.89}$$

So we can write the weak formulation as

$$\text{Find } \dot{\mathbf{u}} \forall V : a(\dot{\mathbf{u}}, \mathbf{v}) = \dot{l}(\mathbf{v}) \quad \forall v \in V, \tag{2.90}$$

with

$$\begin{aligned} a(\dot{\mathbf{u}}, \mathbf{v}) &= \int_{\Omega} \boldsymbol{\epsilon}^T(\mathbf{v}) \mathbf{D}_{ep} \boldsymbol{\epsilon}(\dot{\mathbf{u}}) \, \mathrm{d}\Omega \\ \dot{l}(\mathbf{v}) &= \int_{\Omega} \mathbf{v}^T \dot{\mathbf{f}} \cdot \mathrm{d}\Omega + \int_{\partial\Omega} \mathbf{v}^T \dot{\mathbf{g}} \mathrm{d}\gamma \end{aligned}. \tag{2.91}$$

The term $a(.,.)$ is known as the bilinear form and $l(.)$ is known as the linear form.

2.11.2 Galerkin Method

The construction of the variational problem is similar to the procedure in [10]. We use the Galerkin method to construct a subspace of the function space V, which we denote as V_h. The subspace will be used to solve Eq.2.90, yielding

$$\text{Find } \dot{\mathbf{u}}_h \forall V_h : a(\dot{\mathbf{u}}_h, \mathbf{v}_h) = \dot{l}(\mathbf{v}_h) \quad \forall \mathbf{v}_h \in V_h. \tag{2.92}$$

The main concern is that we used a rate equation. This is solved by decoupling the position and time dependence in the basis, yielding

$$\dot{\mathbf{u}}_h = \sum_j \dot{\mathbf{u}}_j \phi_j(\mathbf{x}), \tag{2.93}$$

where $\mathbf{x}$ is the Cartesian coordinates. The basis of the testfunction is independent of time. So we can insert the basis into Eq.2.92, yielding

$$a\left(\sum_i \dot{\mathbf{u}}_i \phi_i, \sum_j \mathbf{v}_j \phi_j\right) = \dot{l}\left(\sum_j \mathbf{v}_j \phi_j\right). \tag{2.94}$$

We will use the properties of the linear and bilinear form, shown in [8], which will give

$$\sum_j \sum_i \mathbf{v}_j a(\phi_i, \phi_j) \dot{\mathbf{u}}_i = \sum_j \mathbf{v}_j \dot{l}(\phi_j). \tag{2.95}$$

Setting $a(\phi_i, \phi_j) = \mathbf{A}$, $\dot{l}(\phi_j) = \dot{\mathbf{b}}$ and $\dot{\mathbf{u}}_i = \dot{\mathbf{u}}$, we obtain the variational problem

$$\mathbf{A}\dot{\mathbf{u}} = \dot{\mathbf{b}}. \tag{2.96}$$

The subspace V_h will be constructed by using linear Lagrange elements, which can be seen in [8].

2.12 Solving Methods

The variational formulation in Eq.2.96 can be solved with a scheme as

$$\begin{aligned} \mathbf{A}\Delta\mathbf{u} &= \Delta\mathbf{b} \\ \mathbf{u}_{n+1} &= \mathbf{u}_n + \Delta\mathbf{u}' \end{aligned} \tag{2.97}$$

with the initial condition $\mathbf{u}_0 = 0$. However the matrix $\mathbf{A}$ becomes non-linear with the tangent stiffness $\mathbf{D}_{ep}$, given in Eq.2.60.
There are different methods to solve the problem posed in Eq. 2.97 and we will now look at the methods mentioned in [6].
The most extensive method is the tangent stiffness method, which recomputes the stiffness matrix $\mathbf{A}$ for each iteration in obtaining the incremental displacement, making it equivalent to the Newton-Raphson method. A simplification is the modified tangent stiffness method, which only recomputes the matrix once for each incremental step.
The simplest method is the initial stress method, which simplifies by using the elastic tangent only. This requires less computation related to matrix inversion, but will require more incremental steps.
The combination of the tangent stiffness method and the tangent stiffness in Eq. 2.27 will cause the method to lose the quadratic convergence rate. This can be solved by using the consistent tangent moduli, defined as

$$\boldsymbol{\Xi} = \left.\frac{\Delta\sigma}{\Delta\epsilon}\right|_{n+1}. \tag{2.98}$$

The moduli is obtained by a linearisation of the return-mapping algorithm, see [9].

Chapter 3

Implementation

The implementation is given in two parts and organized to give minimal interactions. This makes it easy to adjust the implementation, without unwanted effects.
We will start with the implementation of graphic user interface, then we will proceed with the implementation of the calculation. The calculations are performed with the FEniCS-project [5] to create and solve partial differential equations. We constructed a flowchart to indicate the use of FEniCS-class and functions, see Fig.A.1.

3.1 Graphic User Interface

We based graphic user interface on input handling from the user and the code was written in python with the wx-module, see D.A. The graphic user interface (GUI) is not essential to the calculations, therefore we will give a brief implementation overview.
The different input parameters are listed in Tab.C.1, combined with their functionality and possible options.

3.1.1 CustomApp

The implementation of the calculations may be integrated into the GUI. But the graphic user interface tends to freeze, while the computation is ongoing, causing the program to fail. This can be avoided by extracting the data and starting the computation after the GUI is closed, see D.B.
The input values are extracted from the GUI with the wx.GetApp function. This required that the CustomApp had additional attributes to hold the input values.

3.1.2 Advanced Frame

The AdvancedFrame is for handling specific input parameters, which should remain on the default setting. We can see the advanced frames appearance in Fig.3.1. The construction of the frame was done by declaring an array of the class CustomBox. We can see there are two buttons present in the advanced frame. The buttons are

known as, OK and Reset, the reset button restores the default parameters, while the OK button stores the input and closes the frame.

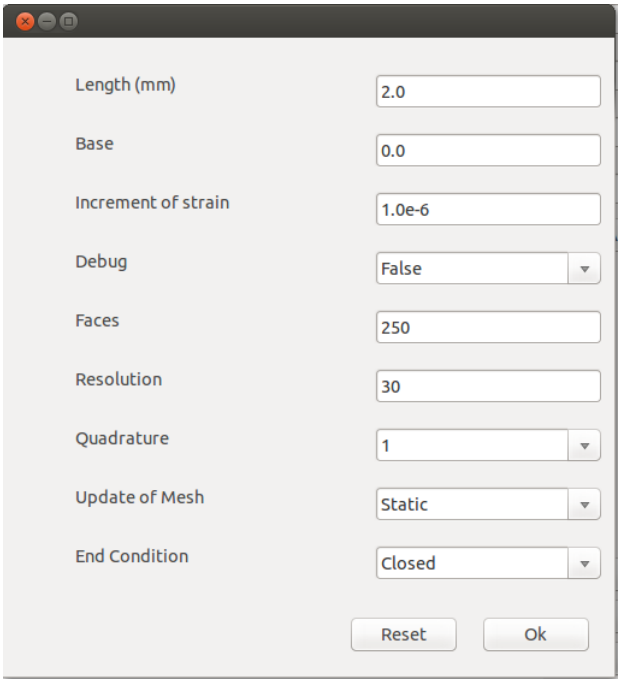


Figure 3.1: Shows the advanced frame for the graphic user interface.

3.1.3 CustomBox

The construction of the class CustomBox was done due to repeated implementation of input boxes. The class makes it easy to add more parameters to the interface, since the appearance is predetermined. We can also generate a drop down menu, but this requires that the possible choices are determined in the initialization of the class.
The inputs are obtained by unicode, which requires a conversion to the appropriate form. This conversion will cause failure if for instance a float input contains letters or symbols . So to avoid any errors, the return form is determined by the class-function Rform. The function uses a try-except sequence on the default input parameter, which undetermined will be float. In case that there is an error, an error message will appear.
The extraction of the input values and the corresponding label is done with the class-functions GetValue and GetLabel. These variables are then combined to create a dictionary.

3.1.4 Main Frame

The main frame contains four panels, each with their own specific task, see Fig.3.2.

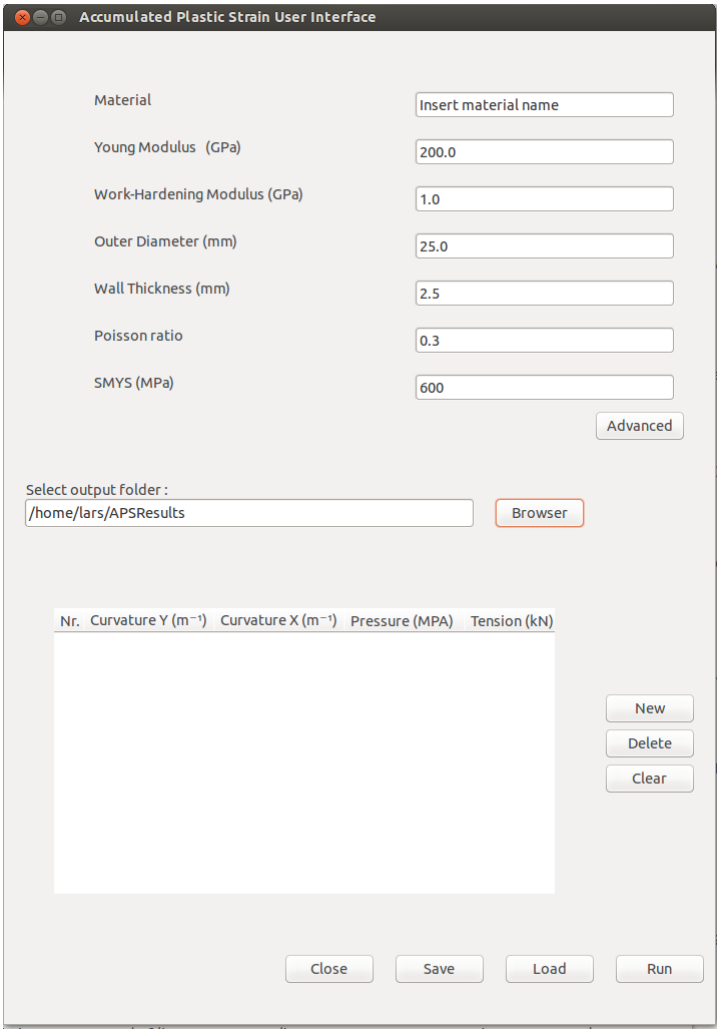


Figure 3.2: Shows the main frame for the graphic user interface. The input name of SMYS in the main frame is Nexans' technical term for yield strength.

First Panel

The first panel holds the general parameters, which are fixed during the computation. This is done by constructing an array of CustomBox. We can see that the first panel also includes the advanced button, which opens the advanced frame.

Second Panel

The second panel is created for the user to select the output folder, in which the results will be stored. The default setting of the folder is determined by the use of config-module, which loads the previous selected folder.

```
−<Data>
 −<General>
    <Value Material="Insert material name"/>
    <Value Young="200.0"/>
    <Value Work-Hardening="1.0"/>
    <Value Outer="25.0"/>
    <Value Wall="2.5"/>
    <Value Poisson="0.3"/>
    <Value SMYS="600.0"/>
  </General>
  <Steps Step="1.0 0.0 0.0 0.0 0.0"/>
  <Steps Step="2.0 0.0 0.0 0.0 100.0"/>
  <Steps Step="3.0 0.0 0.0 0.0 110.0"/>
  <Steps Step="4.0 0.0 0.0 0.0 120.0"/>
  <Steps Step="5.0 0.0 0.0 0.0 130.0"/>
  <Steps Step="6.0 0.0 0.0 0.0 140.0"/>
  <Steps Step="7.0 0.0 0.0 0.0 150.0"/>
  <Steps Step="8.0 0.0 0.0 0.0 160.0"/>
  <Steps Step="9.0 0.0 0.0 0.0 170.0"/>
  <Steps Step="10.0 0.0 0.0 0.0 180.0"/>
</Data>
```

Figure 3.3: Shows the structure of an xml-file, SMYS is Nexans' technical term for yield strength.

Third Panel

The third panel is the user input for defining each load step, there are four different possible inputs for each step. Which can be edited by clicking with the cursor and then inserting the specified value.
Three buttons are connected to the list, which are known as new, clear and delete. The new button initiate a new step to be added to the list. The delete button removes the previous added step, and the clear button removes all previous added steps.

Fourth Panel

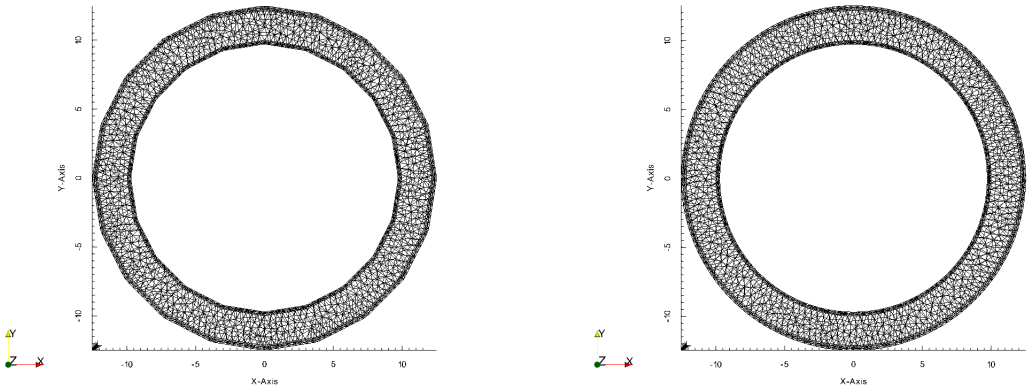
The fourth panel contains four buttons; "close", "run", "save" and "load". The buttons "close" and "run", will both close the program, but "run" will also initiate the calculations.
The options of "load" and "save" the input were implemented by storing the data in xml-file. All user inputs may be stored, with the exception of the output folder and advanced inputs. To avoid error, only the first word is used to identify the general parameters. While each load step is stored as an array, and identified by the word "Step", see Fig.3.3.

3.2 Calculations

In this section we will review the implementation regarding the calculations. The calculations are based on the return mapping algorithm and the finite element method. Most of the code uses FEniCS functions and classes, of which a general introduction is given in [5].

3.2.1 Generating Mesh

The mesh generation is divided into two parts, the first one is to define the initial vertex structure. This is done with the FEniCS-function Cylinder, which requires the geometrical dimensions and the number of faces. The number of faces corresponds to linear surfaces on the curved sides. Similar to the linear approximation of a circle, thus higher number, leads to better approximation, see Fig 3.4.



(a) Shows the mesh with number of faces equal 20.

(b) Shows the mesh with number of faces equal 200.

Figure 3.4: Shows the mesh difference with faces set to 20 and 200, with geometrical data in Tab.4.1.

The next step is to connect the vertices and create the cells, which is done with the FEniCS-function Mesh. We obtain the mesh by inserting the vertex structure and the wanted resolution. The resolution controls the maximum volume of the cells and splits the cells which exceed the maximum volume. The splitting will generate more vertices, but only interior. This means that the linear approximation of the curved side will remain even for higher resolution.

3.2.2 Marking Boundaries

After constructing the mesh, comes the defining and marking of the exterior boundaries. This involves defining the appropriate boundary values with the FEniCS class

SubDomain, see D.H.
The marking of a curved surface proved difficult, since some boundaries can remain unmarked. The solution was to repeatedly mark the curved boundaries, until none was left unmarked. Then the top and bottom boundaries were marked using lower tolerance.

3.3 Function Spaces

The determination of number of variables is done with defining the FunctionSpace, which requires that we specify both the polynomial and quadrature.
The displacement is vectorial, so we will use FEniCS-function VectorFunctionSpace, which will generate three degrees of freedom for each node. The number of degrees of freedom can also be specified, which is done for the tangent stiffness and the Voight tensors.
In the implementation, both the terms of nodes and points are used. The terms are similar, with a slight difference. While nodes refer to the number of displacement nodes, the points refer to the number of Voight points. Only for linear elements are these terms equal.

3.4 Plastic Model

In the theory section we have seen that a plastic model is determined by three functions, yield function, flow rule and hardening rule. We will now define the plastic model and implement the needed functions.
We chose to implement the plastic model given in [9], where the yield function is dependent on the second invariant of the deviatoric stress tensor Eq.2.16. This is known as the Mises yield criteria and will give

$$F\left(\sigma_{ij}\right) = \frac{1}{2} s_{ij} s_{ij}, \tag{3.1}$$

where s_{ij} is defined in Eq.2.13. The model includes an associative flow rule, so that

$$\mathbf{g}_{\sigma_{ij}} = \mathbf{f}_{\sigma} = s_{ij}. \tag{3.2}$$

To obtain a piecewise linear stress-strain relation, a linear hardening is needed. We chose that the hardening parameter is based on the equivalent plastic strain, so the internal variables will not be used.
We chose the hardening to be purely isotropic, excluding the back stress, thus we will use Eq.2.46. The use of Eq.3.1 requires that the hardening is scaled and inserted quadratically into the yield function, see [9].
The specified plastic model will give the following functions in Voight notation

$$\begin{aligned} f\left(\sigma, \bar{\epsilon^p}\right) &= \frac{1}{2} \boldsymbol{\sigma}^T \mathbf{P} \boldsymbol{\sigma} - \frac{1}{3} k^2\left(\bar{\epsilon^p}\right) \\ k\left(\bar{\epsilon^p}\right) &= \sigma_y + H \bar{\epsilon^p} \\ \epsilon^p &= \mathbf{P} \boldsymbol{\sigma} \end{aligned} \quad . \tag{3.3}$$

The hardening parameter is scaled with $^1/_3$, which is known as the characterising yield strength. For uni-axial case, the characterising yield strength is 1.0.
The equations in Eq.3.3 were coded into the class PlasticModel, see D.G. The linear hardening in the plastic model will make the stress-strain relation piecewise linear. This can be seen in Fig.3.5, it will also give a piecewise linear relation between plastic strain and strain, see Fig.3.6. Thus, by knowing the yield stress and the slops, we can approximate the generated plastic strain.

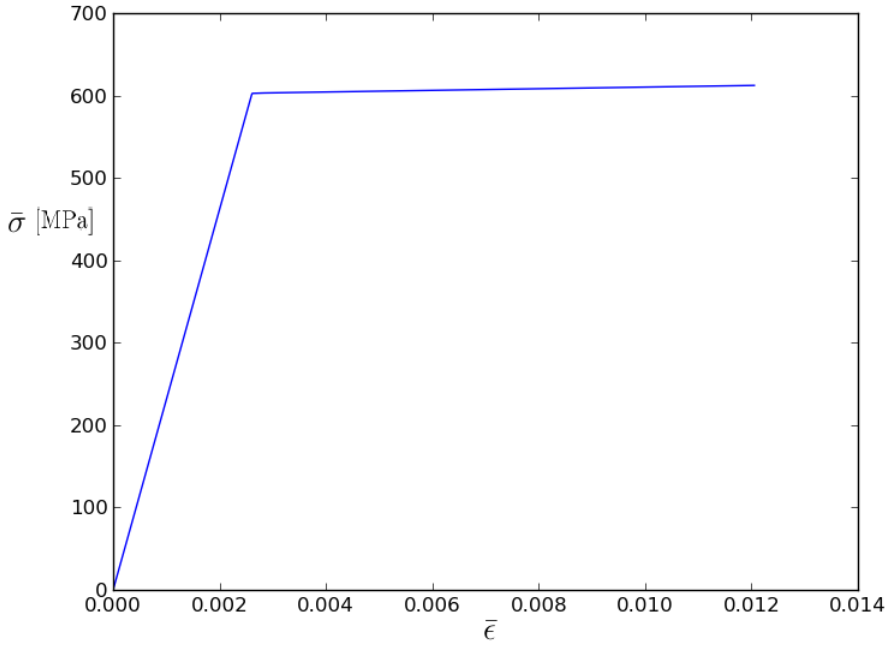


Figure 3.5: Shows the piecewise linear stress-strain relation.

3.5 Modified Tangent Stiffness

We implemented the modified tangent stiffness method to solve the problem posed by Eq.2.97. The method can be seen as two processes; solving the incremental problem and updating the matrix **A**.

3.5.1 Solving Linear Problem

The increment displacement is obtained by solving the variational problem Eq.2.97, which can be done with a direct or an iterative method. The direct method of solving a variational problem requires a high precision to avoid the accumulation of round-off errors. Hence we selected that the problem was to be solved with an iterative method, which is implemented in FEniCS. There are several options for both iterative solvers and precondtioners in FEniCS, see [5]. Most of the iterative

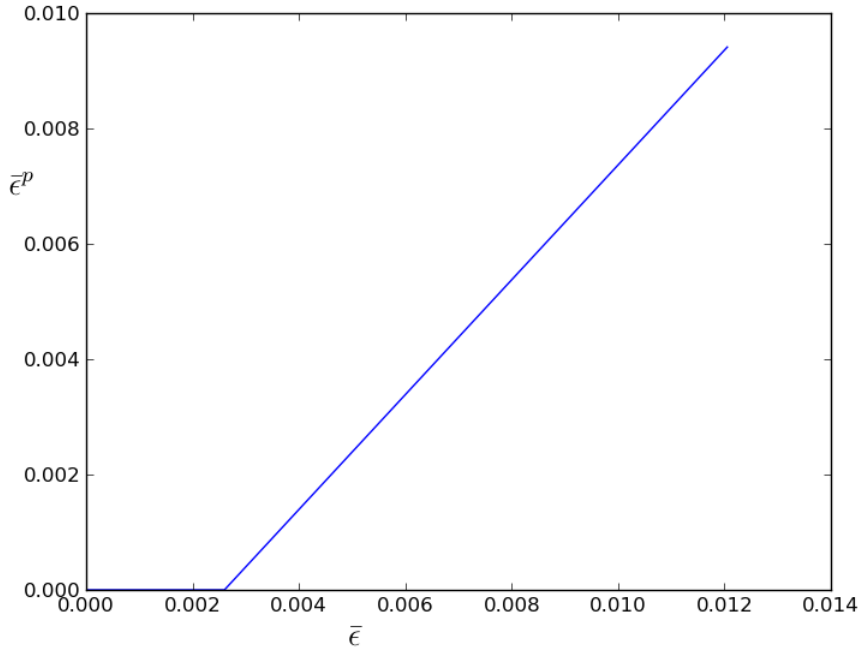


Figure 3.6: Shows the piecewise linear plastic strain and strain relation.

methods experienced convergence problem after the matrix **A** was updated with plastic strain. The option Richardson and successive over relaxation(SOR) proved to be most reliable. The details for the iteration method and preconditioner can be found in [7].
After the incremental displacement is obtained, we need to calculate the tangent stiffness for the next increment. The procedure starts by calculating the total strain. This can easily be done by using the FEniCS-function project, but there are faster methods available.

3.5.2 Faster Projection

The function project in FEniCS is a relatively slow function, and the computation time can be decreased at the cost of memory. The scheme is to assemble the projection matrix, so only matrix multiplication is needed to obtain the projection.
The assembled matrix is stored in the computers memory throughout the computation. Since there are numerous projections in the implementation, the lower computation time outweighs the extra memory consumption.

3.5.3 Return Mapping

The return mapping algorithm is performed on all the points connected to the mesh, thus we implemented a for-loop.

We obtain the strain with matrix multiplication and the results are stored in FEniCS-class Function. The implementation of the return-mapping algorithm uses numpy.matrix objects, which makes matrix operations easier to handle. This requires that we extract the strain corresponding to a point from FEniCS-class Function and assign them to a numpy.matrix object. This is also done for the plastic strain.
We use the total strain and the plastic strain to calculate the trial stress $\boldsymbol{\sigma}^t$ in Eq.2.63. Which will be inserted into the yield function to determine if the stress must be corrected.The corrections require that we know the plastic multiplier, which is obtained with closest-point-projection.

Closest-Point Projection

We need to construct the residual from Eq.2.67, but in the chosen plastic model we have replaced the internal variables with the equivalent plastic strain Eq.3.3. This will produce a set of residuals as

$$\begin{aligned} \mathbf{r}_\sigma &= \boldsymbol{\sigma}_{n+1} + \Delta\lambda \mathbf{D}\mathbf{P}\boldsymbol{\sigma}_{n+1} - \boldsymbol{\sigma}^t \\ r_{\bar{e}^p} &= \bar{e}^p_{n+1} - \Delta\lambda\sqrt{\frac{2}{3}\boldsymbol{\sigma}^T_{n+1}\mathbf{P}\boldsymbol{\sigma}_{n+1}} - \bar{e}^p_n \quad . \\ r_{\Delta\lambda} &= \frac{1}{2}\boldsymbol{\sigma}^T_{n+1}\mathbf{P}\boldsymbol{\sigma}_{n+1} - \frac{1}{3}\left(\sigma_y + H\bar{e}^p_{n+1}\right)^2 \end{aligned} \tag{3.4}$$

Since the residuals should equal zero, we can from the first residual obtain the expression

$$\boldsymbol{\sigma}_{n+1} = \frac{1}{\mathbf{I} + \Delta\lambda\mathbf{D}\mathbf{P}}\boldsymbol{\sigma}^t, \tag{3.5}$$

with $\mathbf{I}$ as the identity matrix. We can see that the only variable is $\Delta\lambda$, so by inserting Eq.3.5 in the other residuals, Eq.3.4, we obtain

$$\frac{1}{2}\boldsymbol{\sigma}^T_{n+1}\mathbf{P}\boldsymbol{\sigma}_{n+1} - \frac{1}{3}\left(\sigma_y + H\bar{e}^p_{n+1}\right)^2 = 0. \tag{3.6}$$

Hence the three coupled equations is reduced to only one. The back stress can be included with minor changes, see [9].
The convergence difficulties explained in the theory section, made the implementation difficult. Both an iterative method and a method using python-functions were tried.
The high computational time with iterative methods made smaller increments with python-functions more preferable. We selected the python-function Brentq, but other functions worked just as well.

Corrections

The plastic multiplier is used to correct the stress and generate plastic strain. We can use Eq.3.5 to obtain the corrected stress $\boldsymbol{\sigma}_{n+1}$. Which is used to obtain

$$\begin{aligned} \boldsymbol{\epsilon}^p_{n+1} &= \boldsymbol{\epsilon}^p_n + \Delta\lambda\mathbf{P}\boldsymbol{\sigma}_{n+1} \\ \bar{\epsilon}^p_{n+1} &= \bar{\epsilon}^p_{n+1} + \Delta\lambda\sqrt{{}^2/_3\boldsymbol{\sigma}^T_{n+1}\mathbf{P}\boldsymbol{\sigma}_{n+1}}. \end{aligned} \tag{3.7}$$

The return mapping algorithm is also used to determine the tangent stiffness, Eq.2.27, for the next increment. Which is

$$\mathbf{D}_{ep} = \begin{cases} \mathbf{D}, & \Delta\lambda \leq 0 \\ \mathbf{D} - \frac{\mathbf{D}\mathbf{P}\boldsymbol{\sigma}^T\boldsymbol{\sigma}\mathbf{P}\mathbf{D}}{\boldsymbol{\sigma}^T\mathbf{P}\mathbf{D}\mathbf{P}\boldsymbol{\sigma}+\frac{2}{3}Hk(\bar{\epsilon}^p)\sqrt{\frac{2}{3}\boldsymbol{\sigma}^T\mathbf{P}\boldsymbol{\sigma}}}, & \Delta\lambda > 0. \end{cases}, \tag{3.8}$$

with $k\left(\bar{\epsilon}^p\right)$ given in Eq.3.3. The plastic strain, stress and tangent stiffness will at the end be reassigned to their corresponding FEniCS-class Function.

3.6 Deformation Steps

We will handle three different deformation steps; axial tension, curvature and internal pressure. The implementation uses only Dirichlet conditions, where the definition is given in [10]. We will focus on applying minimal boundary conditions, so that we preserve enough free variables in the variational problem.
The different deformation steps use different Dirichlet conditions and if these conditions are applied combined, we will have defined a completely different problem with a different solution. Therefore the deformations in each step are ordered in the following sequence, axial tension, curvature and internal pressure.
The inputs are in reference to the tubes initial state, considering that the deformation of bending requires the input of curvature κ_y and κ_x. These inputs will give the curvature state of the tube, and the tube will be bent to that curvature state. So the input of $\kappa_y = 0.0$ and $\kappa_x = 0.0$ will unbend the tube, and likewise for axial tension and internal pressure.

3.6.1 Axial Tension

The axial tension is defined as

$$T = \int \sigma_z \mathrm{d}A, \tag{3.9}$$

where σ_z is the axial stress for every point on the cross-section A.
The implementation uses a while-loop, which continues until the desired axial force is acquired. We see in Eq.3.9, that for a constant area A, this can only be achieved by increasing σ_z. Therefore we will increase the axial stress by increasing the axial strain ϵ_0, which is constant on the surface A.
The axial strain is incrementally increased by using Dirichlet conditions, which requires the expression of the corresponding incremental axial displacement. We can obtain the expression for axial displacement by integrating the strain increment $\Delta\epsilon$ in the axial direction, yielding

$$u_z = -z\Delta\epsilon + \phi\left(x, y\right). \tag{3.10}$$

The axial displacement for $z = 0$ is assumed to be zero, so that $\phi(x, y) = 0$.
The integration given in Eq.3.9 is done by assembling a vector similar to the construction of a linear form in FEniCS. This vector contains the integration over each element on the boundary. Thus we can obtain the axial force by adding the contribution for each vector element together. In the implementation, we chose that the top cross-section corresponded to A, while the Dirichlet conditions were applied on the top and bottom cross-sections of the tube.
The input of axial force is given in the uni-axial case, therefore we need to scale the integration of the axial stress with factor of $^3/_2$ to take multiple dimensions into account.

3.6.2 Bending

We will give a brief repetition of the bending implementation, taken from [10]. We used the infinitesimal strain definition Eq.2.6 to obtain the axial displacement

$$u_z = -z(\kappa_y x + \kappa_x y) + \phi(x, y). \tag{3.11}$$

The axial displacement was set to zero for $z = 0$, which meant that $g = 0$. This will produce the shear strain of

$$\begin{aligned} \epsilon_{23} &= -z\kappa_y \\ \epsilon_{13} &= -z\kappa_x \end{aligned}. \tag{3.12}$$

We see in Eq.3.12 that strain is dependent on the axial position z, thus if the length of the mesh remains small, we can neglect the shear strain. Based on the results of [10], we implemented the Dirichlet conditions on the cross-sections of the tube, i.e top and bottom surface.

3.6.3 Combined Stresses

The combination of axial tension and bending can occur simultaneously. However, bending and axial tension have different while-statements, so it is simpler to asses them separately.

3.6.4 Internal Pressure

The plastic behaviour for a tube under internal pressure is more complex than for bending and axial tension. This requires additional conditions and can be difficult to implement. Therefore we will focus on checking the yield pressure, given in Eq.2.81 against the computed results.
This makes it possible to implement based Eq.2.78, which is the radial displacement in the elastic region. We can see in Eq.2.78 that the displacement is constant for $r = a$, thus we can use Dirichlet conditions.
In Eq.2.78, we can see that the displacement is determined by the pressure. Therefore we need a relation between the incremental strain and incremental pressure,

which we set to

$$\Delta\epsilon = \frac{\Delta p}{E\left(\left({}^{b}/_{a}\right)^2 - 1\right)}, \tag{3.13}$$

with Δp as the incremental pressure.
The insertion of the Eq.3.13 into Eq.2.78 and Eq.2.79 will provide the expression of the incremental radial displacement as

$$\Delta u_r\left(a\right) = \left(\left(1+\nu\right)\left[\left(1-2\nu\right)a + \frac{b^2}{a}\right] - \left(1-2\nu\right)\nu a\right)\Delta\epsilon. \tag{3.14}$$

The incremental displacement is normal to the surface, so we multiply the negative surface normal with the increment radial displacement to obtain the Dirichlet conditions.
We have not taken into account the presence of an axial strain defined in Eq.2.79. Since the axial and radial displacements are orthogonal, we can apply Dirichlet conditions separately. This is no longer valid for curved tubes, which require additional implementations. The axial strain was also used applied with Dirichlet conditions on the cross sections of the tube. This was to ensure that the axial strain was constant in the tube.
The program will continue to solve the incremental problems, until the internal pressure exceeds the input parameter. This requires a good indicator of the internal pressure. In Eq.2.75 we see that for $r = a$ the radial stress will be equal to the internal pressure. So the average radial pressure on the inner surface can be a good indicator. The average radial stress can be obtained with

$$\text{Average}\left(\sigma_r\right) = \frac{\int \sigma_r \mathrm{d}A}{\int \mathrm{d}A}, \tag{3.15}$$

where A is the inner curved surface.
The integration of the radial stress required some extra implementation. This is because for each element integration, there will exist an opposite contribution for an element on the opposite side of the curved surface. Therefore we take the absolute value of each element integration and add them together.
The implementation is based on an elastic equation and constant displacements. Thus the implementation does not account for plastic strain and changes in the mesh.

3.7 Scaling

The geometrical dimensions are scaled to m from mm, which require a scaling of other parameters. The scaling is shown i Tab.3.1, which is taken from [10].

3.8 Output

The data produced by the program is saved in the selected output-folder. This is either done by the class DataHandler or the function File in FEniCS.

Table 3.1: Shows the scaling factors and units for the appropriate parameters taken from [10].

-	u	E	σ	κ	Area
Units	mm	$^{\mathrm{N}}/_{\mathrm{m}^2}$	$^{\mathrm{N}}/_{\mathrm{m}^2}$	m^{-1}	m^2
scaling	10^3	10^{-6}	10^{-6}	10^{-3}	10^6

3.8.1 DataHandler

The class Data was created to extract the necessary data, it is constructed to reinitialize accordingly to iteration and step. We extract the maximum values of the equivalent stress, and the corresponding strain and plastic strain in each iteration. These values are then saved at the end of each step.
The class also stores the maximum APS-value for each step in an attribute array, which is saved at the end of the program.

3.8.2 File

The FEniCS-function File creates a pvd-file to save the data, which links the values to the mesh, which can be visualised in Paraview, see [3]. This generation of data is done at the end of each step, for stress, strain and APS.

Chapter 4

Results

4.1 General Parameters

The APS-program requires that we specify multiple input parameters, which means many possible combinations. Thus we will focus on one set of general parameters, which is given in Tab.4.1. We set the length of the mesh to 2.0 mm, which will decrease the computational time. The length will also make it possible to neglect the shear strain in Eq.3.12, which is produced by the Dirichlet conditions for bending. We will focus on linear elements, and only use quadratic elements if specified.
We will first look at mesh convergence and then the strain increment dependence, for each deformation step.

Table 4.1: Shows the general input parameters for the tube. The term SMYS is Nexans' technical term for yield strength.

Label	Value
Young Modulus	200 GPa
Poission's ratio	0.3
Outer Diameter	25.0 mm
Wall thickness	2.5 mm
Work-Hardening Modulus	1 GPa
SMYS	600 MPa

4.1.1 Analytical Approximations

The chosen plastic model is characterised by having a piecewise linear stress-strain relation, This makes it possible to approximate the results analytically. We will denote the APS generate by the analytical approximations as APS_a
The analytical approximations can be obtained by

$$\epsilon^e = \frac{1}{E}\min(\sigma, \sigma_y) \qquad , \qquad \epsilon^p = \frac{<\sigma - \sigma_y>}{H}. \tag{4.1}$$

In the case of bending, we can estimate the total strain with Eq.2.72. So by calculating the elastic strain ϵ^e in Eq.4.1, we can obtain the plastic strain by Eq.2.72.
For axial tension, we must assume uniform stress distribution, so that the axial stress is obtained by dividing the axial force with the cross-sectional area.
The analytical approximation of internal pressure is more complicated, thus we will not try to approximate the results. But we can use Eq.2.81 to asses the computed yield pressure.
The analytical approximations do not take into account the differential changes of the tube, which means that we must consider a static mesh.

4.2 Consistency

We need to evaluate the consistency of results, because the construction of the mesh uses Delauney triangulation to generate the cells. The Delauney triangulation uses randomization to generate the cells, which means that each mesh generate is unique, see [7]. The difference between two meshes is small, but can produce different incremental solutions that will accumulate for each increment.
So to test the consistency, we will run the program in two sequences. A sequence consists of running the program 10 times with the same input parameters. We will consider the case of bending. In the first sequence, we set$\kappa_y = 1.0\text{m}^{-1}$ and for the second sequence $\kappa_y = 0.3\text{m}^{-1}$. The strain increment was set to $1.0e^{-4}\%$, which produced the results listed in Tab.4.2.

Table 4.2: Shows the APS values for the two sequences with strain increment $1.0e^{-4}\%$.

(a) Shows computed APS values for 10 runs with the input parameters $\kappa_y = 1.0\text{m}^{-1}$ and the number of increments was 1000.

Run	APS [%]
1	0.941157
2	0.943280
3	0.941303
4	0.939854
5	0.938016
6	0.938935
7	0.941519
8	0.944565
9	0.943483
10	0.944692

(b) Shows computed APS values for 10 runs with the input parameters $\kappa_y = 0.3\text{m}^{-1}$ and the number of increments was 300.

Run	APS [%]
1	0.072247
2	0.071677
3	0.071583
4	0.072707
5	0.073247
6	0.073222
7	0.070891
8	0.072060
9	0.072221
10	0.072181

The results in Tab.4.2a gave a mean value of 0.942% and standard deviation of 0.002 %. And the results in Tab.4.2b gave a mean value of 0.0722% and a standard deviation of 0.0007 %. This indicates a dependence between the number of increments and the standard deviation. This can be related to the different incremental solutions.
The coefficient of variation is obtained by dividing the standard deviation with the mean values. Based on the results in Tab.4.2, the coefficient of variation was less than 1.0% in both cases. Thus we can conclude that the results are consistent. We assume that the results are also consistent for axial tension and pressure.
We see in Fig.4.2, that the standard deviation marks the uncertainty in the third decimal, thus we will only present results up to the third decimal.

4.3 Mesh Convergence

The finite element method is characterised by the fact that the results converge with finer mesh. However, the increased quality of the mesh will require longer computational time. Thus finding a balance between mesh quality and computational time will be beneficial.
We will generate a finer mesh by increasing the number of nodes within the mesh, but to generalize the quality, we need to account for different mesh volumes. Thus we will assert the quality of the mesh by the density of nodes. This density will be denoted ρ and will have the dimension $^{\text{nodes}}/_{\text{mm}^3}$.
The linear approximation of the curved surfaces are connected to the outer diameter. So to correctly indicate the quality of the linear approximation, we will use the number of faces divided by the outer diameter, denoted as l, with dimensions $^{\text{faces}}/_{\text{mm}}$.

4.3.1 Bending

We will first asses the convergence in the case of bending. The curvature was set $\kappa_y = 1.0\text{m}^{-1}$, while the other step parameters remained zero. The computed results are shown in Fig. 4.1. In the computation the strain increment was set to $1.0e^{-3}\%$ to decrease the computation time.
From Fig.4.1 we see that the APS-values converge with higher values of ρ. The results seem to have the shape of a damped oscillation, indicating a more unstable convergence than asymptotic. We see that higher l converge quicker, so there is a relation between l and the damping. This makes a high number of faces preferable, so that we can obtain convergence with lower node density.
We will look closer on the shape of convergence, which can be caused by linear elements. The same procedure was used in the convergence of quadratic elements, giving the results shown in Fig.4.2.

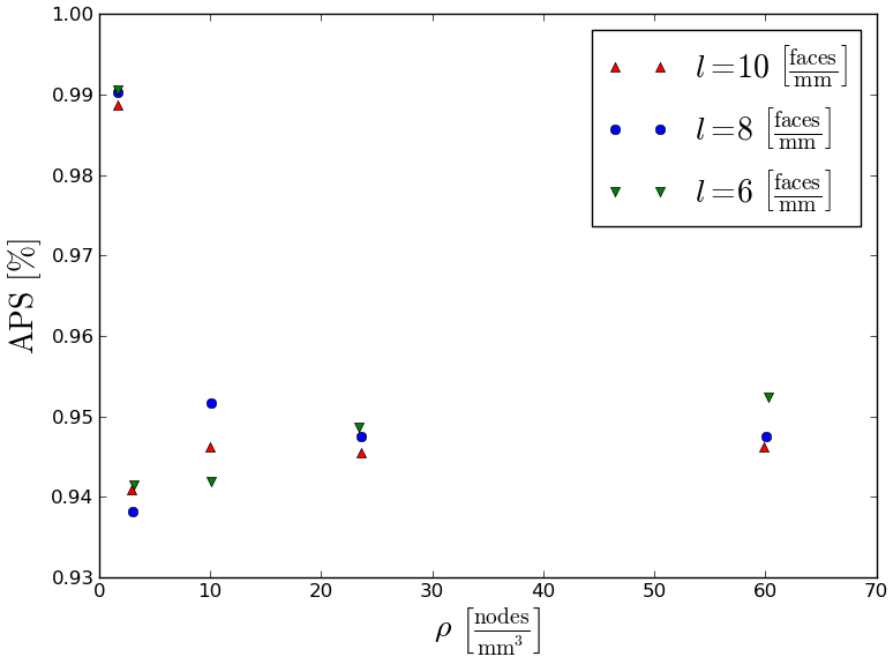


Figure 4.1: Shows the convergence of the APS versus the density ρ for different number of faces, with the strain increment set to $10^{-3}\%$.

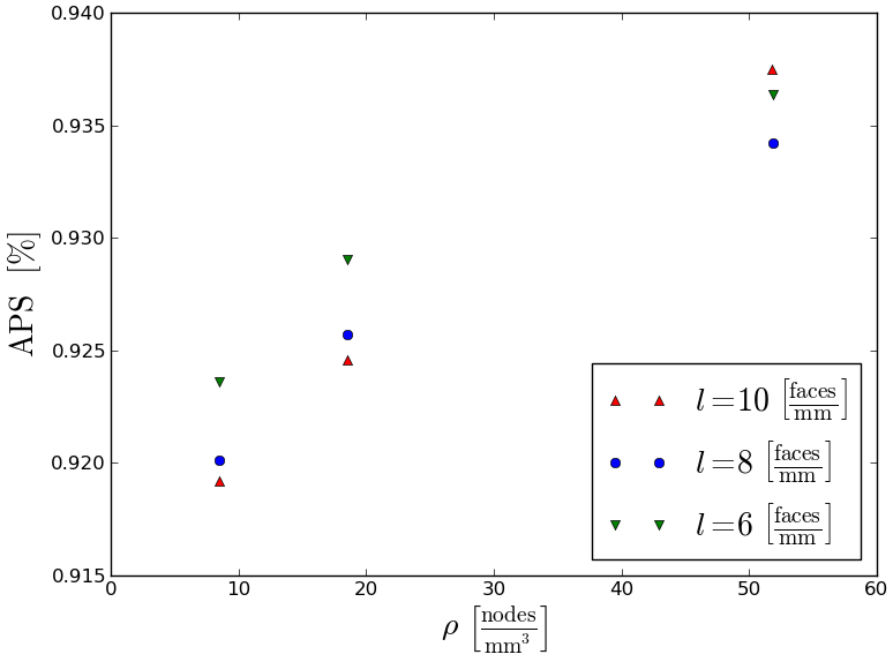


Figure 4.2: Shows the convergence of the APS versus the density ρ an with different numbers of l marked in the legend, with $\kappa_y = 1.0\text{m}^{-1}$.

We observe that in Fig.4.2 the results converge asymptotic. So the convergence of a damped oscillation occurs by using linear elements. The relation between l and the damping indicates that the oscillation is connected to the linear approximation of the curved surface. However, the maximum APS values are taken from one point on the outer rim of the tube. So the oscillation which occurs is caused by linear under- and over-approximation for one point in the mesh.
The analytical approximation was set by using Eq.4.1, and yields 0.95%. This corresponds to the converged results in both Fig.4.1 and Fig.4.2.

4.3.2 Axial Tension

We will now consider the case of axial tension, and input for the axial force was set to 106.2kN. The analytical approximations in Eq.4.1 gave that yield force was 106.028kN, thus the axial force of 106.2kN should produce a low amount of plastic strain. We set the input parameter close to the yield surface to avoid long computation time, due to high nodal density. The strain increment was set to $10^{-4}\%$, which computed the results found in Fig.4.3.

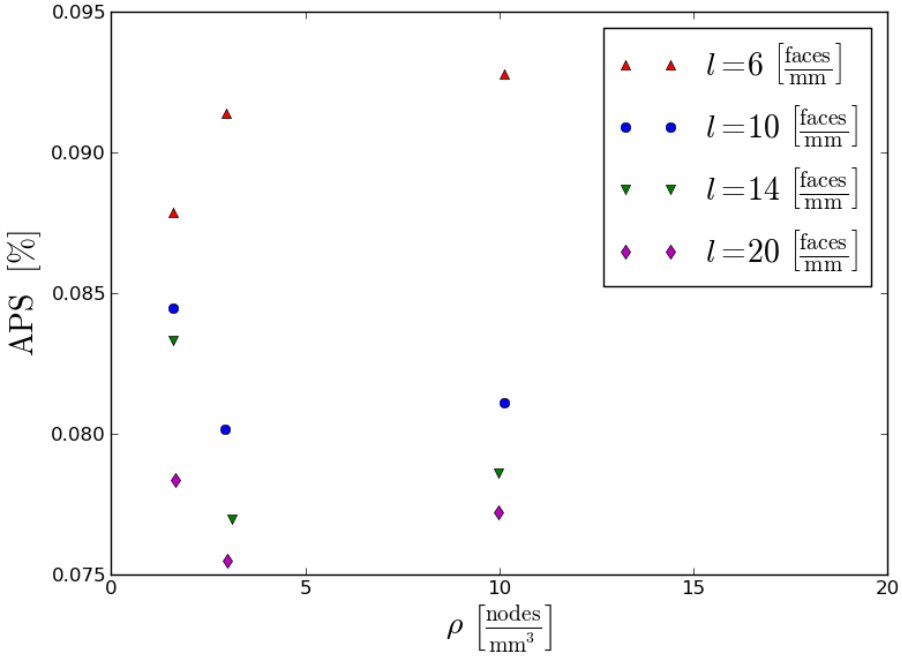


Figure 4.3: Shows the convergence of the APS versus the density ρ for different number of faces, given an axial force of 106.2kN.

In Fig.4.3, we see that the results converge. However, we see that lower values of l generate more APS, so there is a discrepancy related to l. This discrepancy is due to the fact that the parameter l also presents the quality of the cross-sectional area, over which we integrate. We will now look at the effects of the APS due to the approximation of the cross sectional area.

The while-loop will end for a specific value of axial forces, which was set to 106.2kN. We can see in Fig.4.4 that the stress is almost uniform, which allows Eq.3.9 to be written as

$$T = \sigma_z A. \tag{4.2}$$

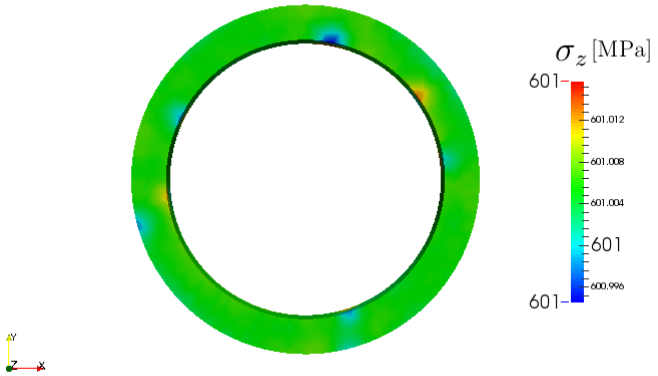


Figure 4.4: Shows axial stress distribution for axial force of 106.2kN.

Since the tension value is constant, we can obtain the corrected axial stress with

$$\sigma_z = \frac{A^*}{A_t}\sigma_z^*. \tag{4.3}$$

The mark * denotes computed values and A_t is the theoretical cross-section. We will denote the ratio of A^* and A_t as r_A. We made the corrections on the results from Fig.4.3, which are listed in Tab.4.3.
We can see in Tab.4.3 that the corrected APS values are similar for all the different combinations of ρ and l. This indicates that there is almost an immediate convergence of the APS value. However, the analytical approximation gives an APS value of 0.0969%, which do not correspond to the corrected values in Tab.4.3. This discrepancy is quite large and will require looking at other parameters.

4.3.3 Internal Pressure

The radial displacement is applied on the inner curved surface, which can indicate a large dependence on l. Therefore we will consider a larger variation of l. We will consider the case of close-ended tube, so that Eq.2.77 will give $\alpha = 0.4$. This is used with Eq.2.81 and the values in Tab.4.1, to estimate the yield pressure as 214MPa. We set the input for internal pressure to 220MPa, so that we could asses the yield pressure and the generation of APS. The results were produced with a strain increment of $1.0e^{-5}$%, and seen in Fig.4.5.
We observe in Fig.4.5 that the results converge for higher values of nodal density ρ. The APS is low indicating a correspondence with the yield pressure.

Table 4.3: Shows the results from Tab.4.3 and the corresponding area ratio and corrected APS. The axial input set 106.2kN.

ρ	l	APS* [%]	r_A	APS [%]
1.5	6	0.0879	0.99979	0.0753
1.5	10	0.0845	0.99984	0.0749
1.5	14	0.0833	0.99986	0.0751
1.5	20	0.0783	0.99994	0.0748
3	6	0.0914	0.99972	0.0744
3	10	0.0802	0.99990	0.0744
3	14	0.0770	0.99995	0.0742
3	20	0.0755	0.99998	0.0744
10	6	0.0928	0.99970	0.0749
10	10	0.0811	0.99989	0.0745
10	14	0.0786	0.99994	0.0751
10	20	0.0772	0.99997	0.0753

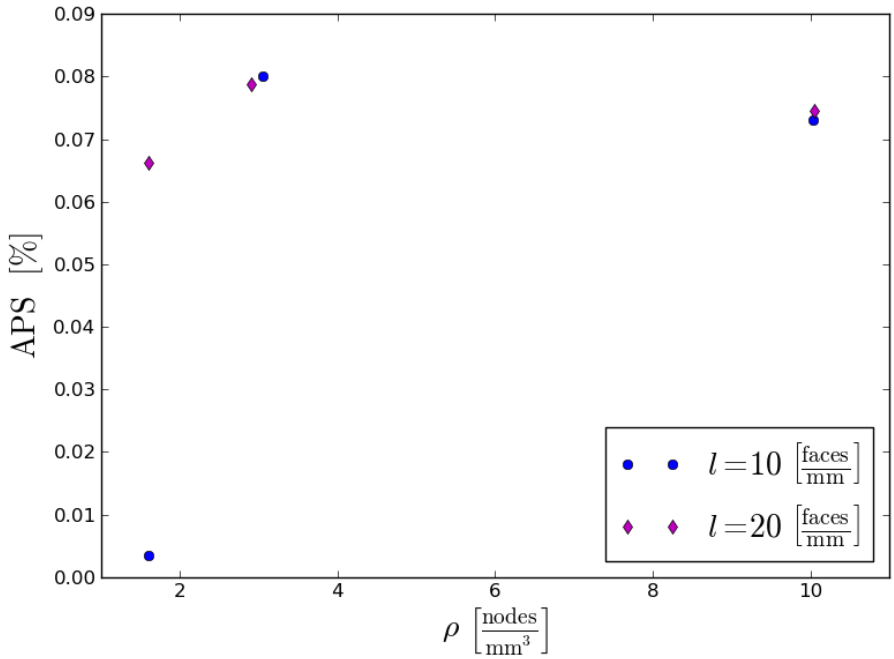


Figure 4.5: Shows the APS values for different values of l and ρ, which are marked in the legend and along the x-axis. The results were computed with the internal pressure of 220MPa and strain increment of $1.0e^{-5}$%.

4.4 Strain Increment

The results are obtained by adding strain increments to the total strain, we will therefore investigate the effects of different strain increments for each of the deformations.
To better investigate the dependence, the resolution and number of faces where respectively set to 30 and 250. This corresponds to a nodal density of $\rho = 3\ ^{\text{nodes}}/_{\text{mm}^3}$ and $l = 10\ ^{\text{faces}}/_{\text{mm}}$.

4.4.1 Bending

The computation of bending is done by adding strain increments, until the total strain is reached. This means that the total strain value is not affected by different strain increments.
We set parameter $\kappa_x = 1.0\text{m}^{-1}$, while the other parameters remained zero. The APS values with corresponding strain increments are listed in Tab. 4.4.

Table 4.4: Shows the APS and the incremental strain contribution for an axial force of $110kN$.

APS [%]	$\Delta\epsilon$ [10^{-4}%]
0.941	200
0.942	100
0.943	50
0.944	25
0.945	12.5
0.945	1

We observe a slight difference for the APS values in Tab.4.4. Since the total strain value remains constant, the difference must be caused by the generation of plastic strain.
Thus we will look at the stress-strain diagram for the procedure. For simplicity, we will consider the equivalent stress and strain, which require no scaling of the yield strength. The results can be seen in Fig.4.6, and we observe a discrepancy in the yield strength. Therefore we will look closer on the yield surface, shown in Fig. 4.7. The discrepancy seen in Fig.4.7, is caused by using an incorrect value of the plastic multiplier in the return mapping algorithm. This incorrect value is acquired due to the convergence problems of the closest-point projection. The incorrect plastic multiplier is either too small or too large, each with a different effect on the APS values.
The case that the value of the plastic multiplier is too small, will cause the stress not to be fully mapped onto the yield surface. This is seen in Fig.4.7, and will generate a a lower amount of APS. If the value is too large, the effects are opposite, causing an increase in the APS value.

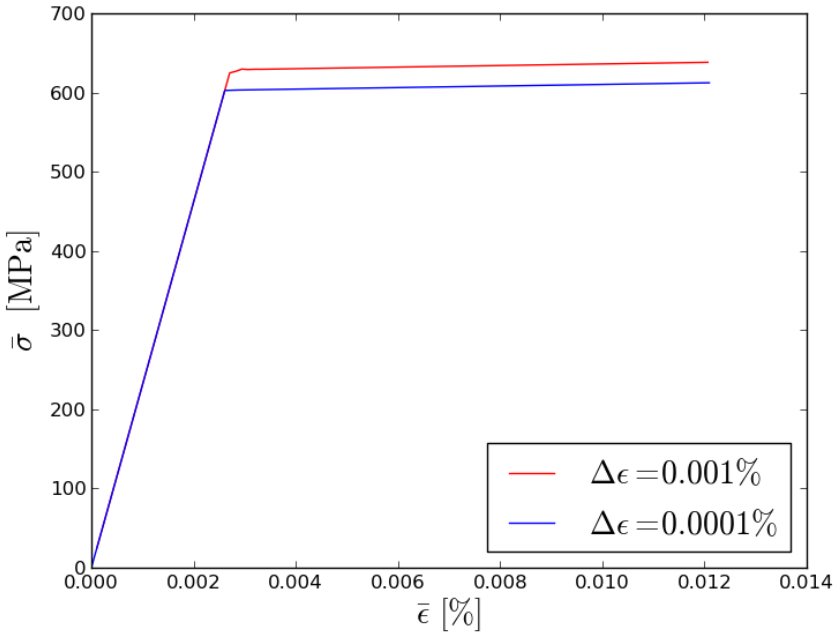


Figure 4.6: Shows the deviatoric stress-strain diagram for bending to $\kappa_x = 1.0\text{m}^{-1}$, with the number of increments marked in the legend.

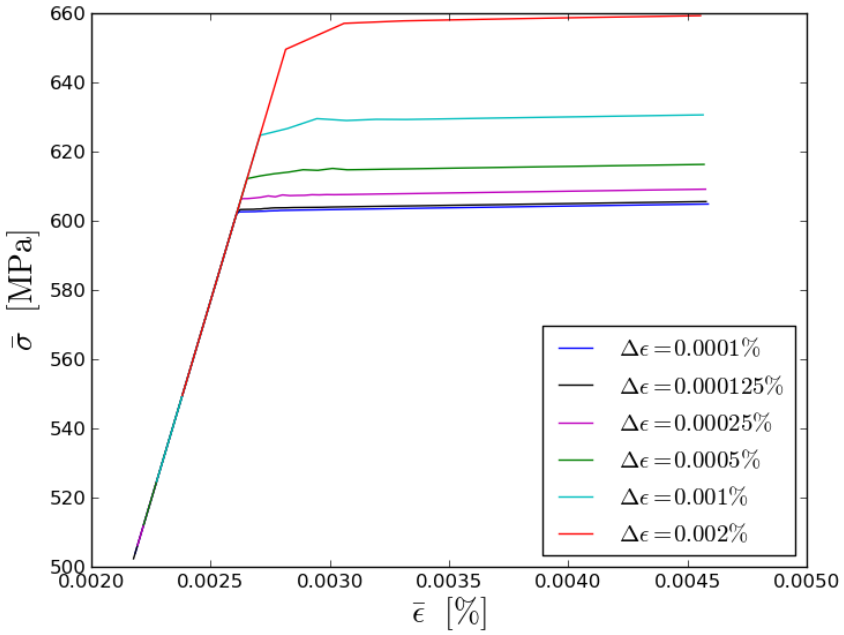


Figure 4.7: Shows the deviatoric stress-strain diagram for bending to $\kappa_y = 1.0\text{m}^{-1}$, with the number of increments marked in the legend.

4.4.2 Axial Tension

We will now consider the deformation of axial tension, which, like bending, applies the Dirichlet conditions on the top and bottom surfaces. We set the axial force to be 110kN and we varied the strain increments, which produced the results given in Tab.4.5.

Table 4.5: Shows the APS and the incremental strain contribution for an axial force of $110kN$.

APS [%]	$\Delta\epsilon$ [10^{-4}%]
0.023	100
0.458	80
1.090	60
1.340	40
1.893	20
2.027	10
2.072	8
2.165	4
2.231	1

We can see large discrepancies in the APS values given in Tab.4.5. The analytical approximation for an axial force of 110kN is 2.247%. The approximation was used to compute the relative error seen in Fig.4.8. In Fig.4.8, we observe a linear relation between the relative error and the increment size.

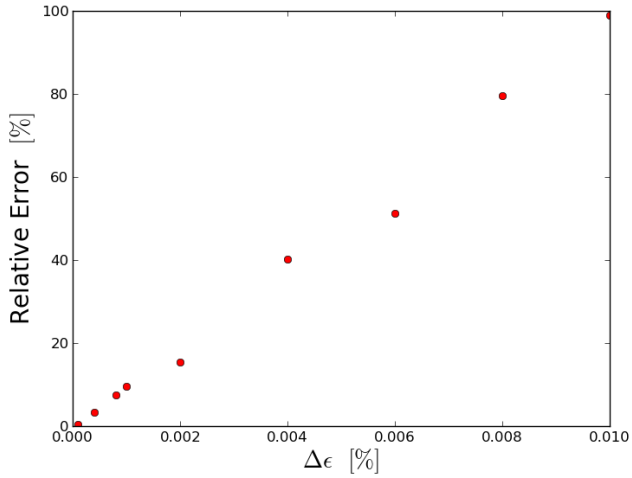


Figure 4.8: Shows the relative error for the APS-value compared to the strain increments. The relative error is constructed with the analytical approximation of 2.247%.

This dependency on the strain increment is not present in the case of bending. Thus

we will compare the transition from elastic to plastic in the two cases. The transition for bending is seen in Fig.4.7, while the transition for axial tension is seen in Fig.4.9.

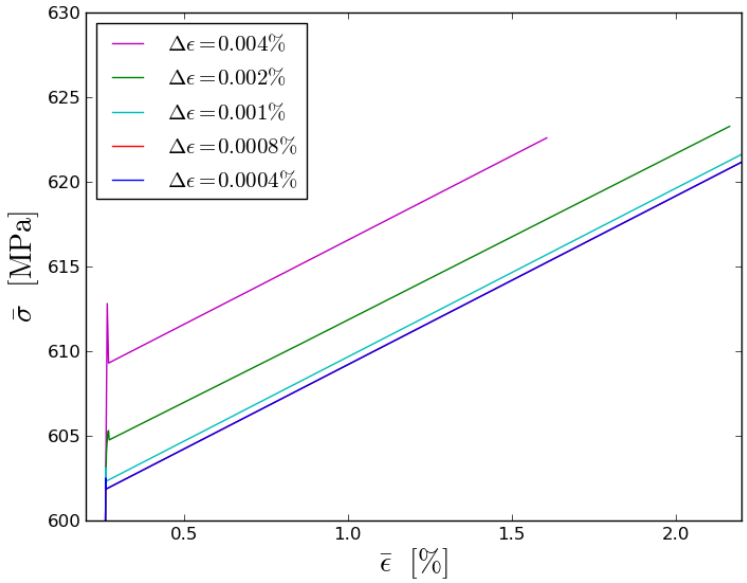


Figure 4.9: Shows the deviatoric stress-strain diagram close to the yield surface for an axial force of $T = 110.0$kN, with the strain increments marked in the legend.

We can see that Fig.4.9 has similar discrepancies as Fig. 4.7. But in Fig.4.9 we can see that the stress is cut-off, generating less APS. This is due to the while-statement, which ends when the axial force exceeds the input value. If the stress is not fully mapped onto the yield surface, the integration will give an axial force of greater value, thus ending the while-loop prematurely.
This can also explain the linear relation shown in Fig.4.8. We see that in Fig.4.9 the different stress-strain curves remain parallel in the plastic region, thus the APS error is proportional to the discrepancy of the yield strength.
This would mean that the difference in APS is constant for a given strain increment. This can be seen in Tab.4.6, which shows the APS value for different values of axial force and the difference from the corresponding analytical approximations.
We can see from Tab.4.6 that the difference is constant for a given strain increment. The linear dependence in the relative error, indicates that infinitesimal strain increments will give zero error. Though this would mean an infinite computation time.

4.4.3 Internal Pressure

The incremental Dirichlet conditions are calculated with Eq.3.14, which multiplies the strain increment with the radial dependency. Therefore the incremental displacement is larger than for bending and axial tension, so we will use smaller strain increments. We set the internal pressure to 220MPa, which produced the results

Table 4.6: Shows the APS for different values of axial strain, and the corresponding approximations, for strain increment of $1.0e^{-4}\%$.

Axial force [kN]	APS [%]	APS_a [%]	ΔAPS [%]
106.2	0.080	0.097	0.017
106.4	0.193	0.210	0.017
106.6	0.306	0.323	0.017
106.8	0.419	0.436	0.017
107.0	0.533	0.550	0.017
107.2	0.646	0.663	0.017
107.4	0.759	0.776	0.017

seen in Tab.4.7 for different strain increments. We see in Tab.4.7 that the APS

Table 4.7: Shows the APS and the incremental strain contribution for an axial force of $110kN$.

APS [%]	$\Delta\epsilon$ [10^{-5}%]
0.079	10
0.088	8
0.088	6
0.076	4
0.073	2
0.087	1

values are small, which means that the computed yield pressure corresponds with Eq.2.81. The implementation was based on the elastic radial displacement, thus the behaviour in the plastic region is incorrect. Therefore we will look at the equivalent stress-strain diagram close to the yield surface, see Fig.4.10.
In Fig.4.10, we can observe that the different increments have different slops in the elastic area. This indicates a difference in the incremental solution, which accumulates for each increment. The more curious behaviour occurs in the transition to the plastic region. We can see an instantaneous lowering of the stress. The behaviour in the plastic region can best be seen in Fig.4.11, which shows a strain and plastic strain diagram.
We see in Fig.4.11 that the plastic strain is produced instantaneously. This can explain the lower of the stress in Fig.4.10, since the plastic strain is included in the determination of the trial stress. If the plastic strain strain increment is larger than next strain increment, then we will have a decrease in the trial stress. The decrease will move the trial stress into the elastic region, thus no corrections.
This clearly indicates that the elastic displacement can not be used in the plastic region. The lowering of the stress indicates that the plastic displacement is larger than the elastic.
We will also asses the use of the average radial stress Eq.3.15 in the while-statement. For this, we need to look at the distribution of the radial stress, seen in Fig.4.12.

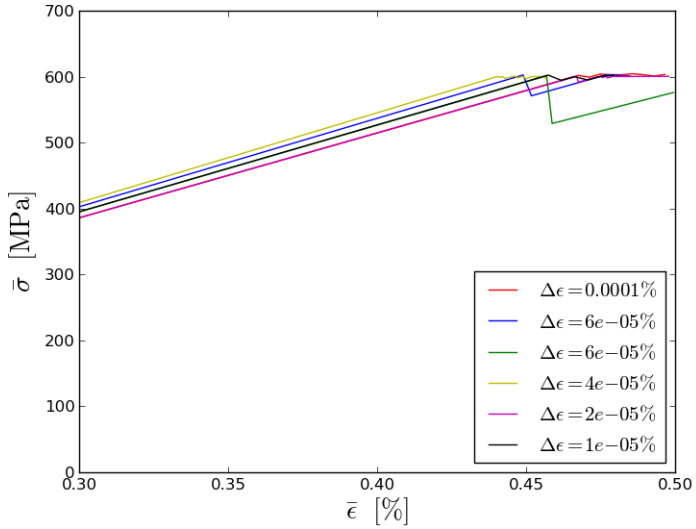


Figure 4.10: Shows stress-strain diagram close to the yield surface with the internal pressure set to 220MPa. The different strain increments are marked in the legend.

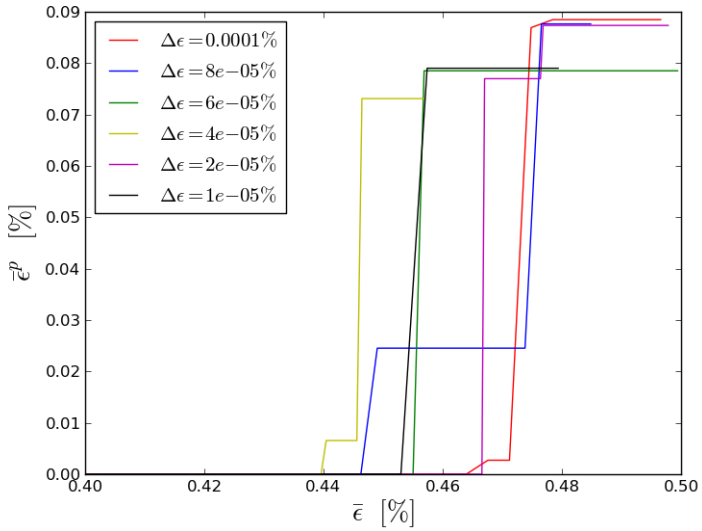


Figure 4.11: Shows the generation plastic strain close to the yield surface in Fig.4.10. The input for internal pressure was 220MPa and the different strain increments are marked in the legend.

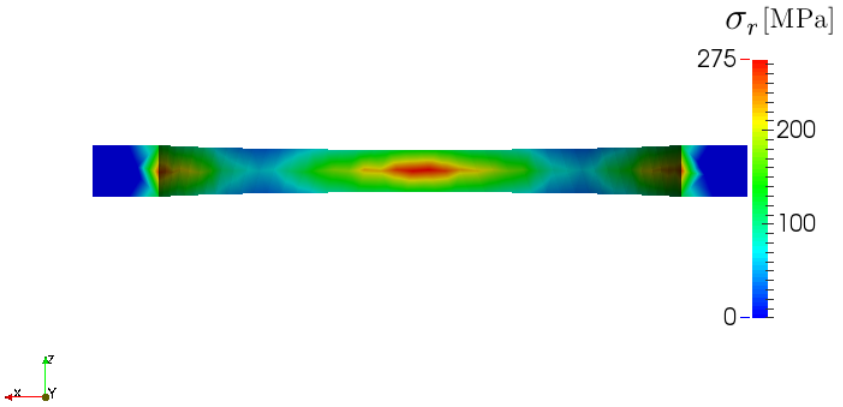


Figure 4.12: Shows the radial stress distribution on one half of the inner surface. The results were computed with the internal pressure of 220MPa and strain increment of $1.0e^{-5}\%$

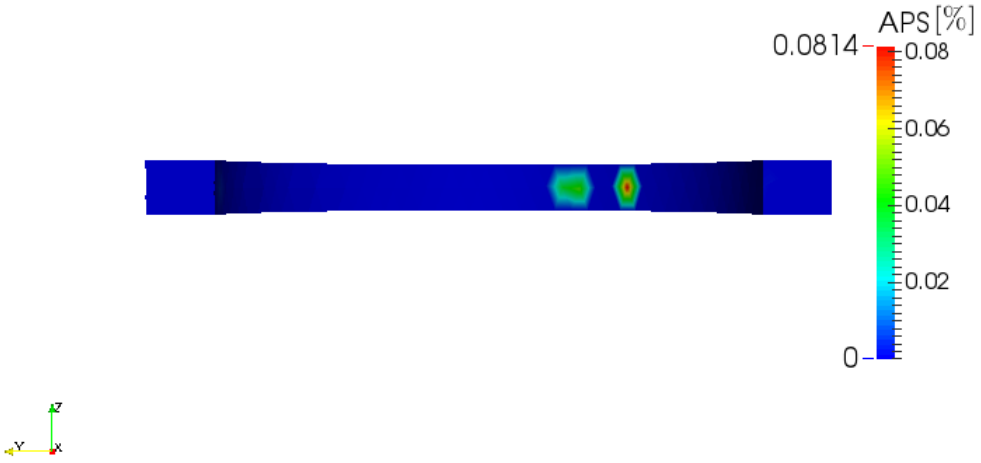


Figure 4.13: Shows the APS distribution on one half of the inner surface. The results were computed with the internal pressure of 220MPa and strain increment of $1.0e^{-5}\%$.

In Fig.4.12 we can see that the stress distribution is not uniform. This does not coincide with the theory, which should produce a uniform radial stress distribution. Thus the use of the Eq.3.15 do not present an accurate relation to the APS with the current implementation. This can be further shown by looking at the APS distribution in Fig.4.13. We can see in Fig.4.13 that the generation of APS occurs locally and not corresponding with the maximum radial stress. This indicates that most of the radial stress distribution is in the elastic region. Thus not caused by using elastic displacement in the plastic region.
The Dirichlet conditions enforce a uniform displacement field, but we see a non-uniform radial stress distribution. This indicates that linear Lagrange elements are not to be preferred for this problem.

4.5 Length Dependence

The results have been computed, based on mesh with length 2.0 mm. So we will consider the results that are computed with different lengths of the mesh. The length of the tube will be denoted as L and dimension of [mm].

4.5.1 Bending

We will look at the case of bending with the input parameter $\kappa_x = 1.0\text{m}^{-1}$. The strain increment was set to $1.0e^{-4}\%$, which gave the results shown in Tab.4.8

Table 4.8: Shows the APS value for different lengths L. The step input was $\kappa_x = 1.0\text{m}^{-1}$ and strain increment $1.0e^{-4}\%$.

L [mm]	APS [%]
2.0	0.941
10.0	0.946
30.0	1.101
50.0	3.761

In Tab.4.8, we can see that the length has a significant effect on APS value. We have previously calculated that the analytical approximation for $\kappa_y = 1.0\text{m}^{-1}$ was 0.95 %. This was used to generate the relative error seen in Fig.4.14.
We can see in Fig.4.14 that the relative error has the shape of a higher order equation. This indicates that there is difference in the generation of APS. The APS distribution for $L = 10$ and $L = 50$ can be seen in Fig.4.15.
We can see that maximum value of APS in Fig.4.15a is generated on the top surface. While in Fig.4.15b the APS generation is constant along the x-direction. The top surface is a Dirichlet boundary, so the difference can be in the Dirichlet condition. We mentioned in the implementation that the Dirichlet conditions would produce shear strain. In Eq.3.12 we see a dependence on the z-coordinates.

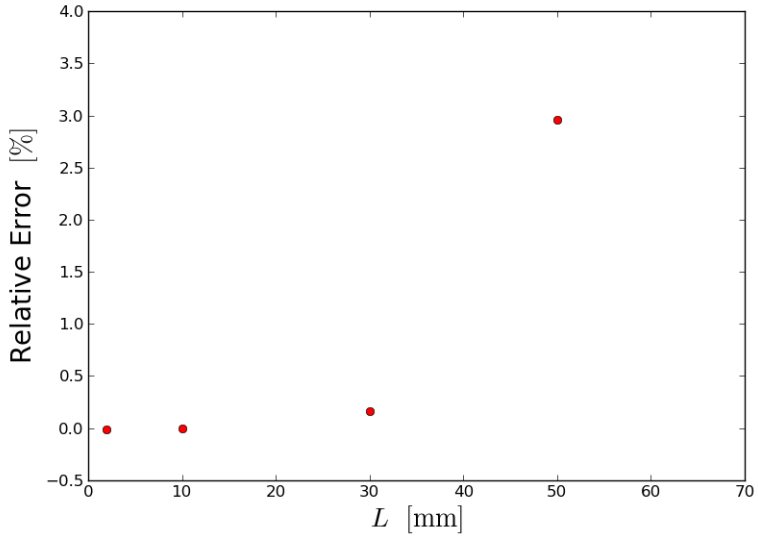


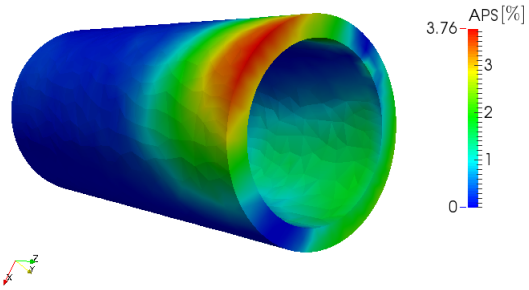
Figure 4.14: Shows relative error of the APS value 0.95% for different lengths of the mesh. The input parameter was $\kappa_x = 1.0\text{m}^{-1}$ and strain increment of $1.0e^{-4}\%$.

However, we listed the absolute values of the components in the final maximum stress tensor in Tab.4.9. We can see that the shear stress increases, but compared to the discrepancy in σ_{22} and σ_{33} it is still small.

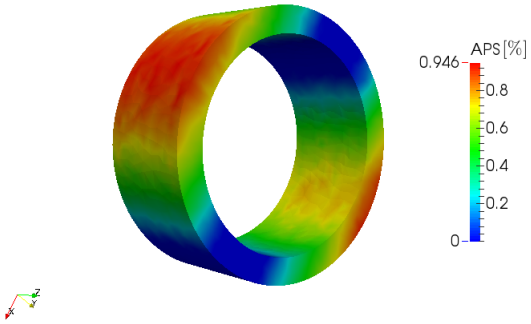
Table 4.9: Shows the absolute value for each component in the final incremental maximum stress tensor for different lengths L. With $\kappa_x = 1.0\text{m}^{-1}$ and strain increment $1.0e^{-4}\%$.

$\lvert\sigma_{ij}\rvert$	$L = 10$mm	$L = 50$mm
$\lvert\sigma_{11}\rvert$	198.090 MPa	190.820 MPa
$\lvert\sigma_{22}\rvert$	210.054 MPa	242.000 MPa
$\lvert\sigma_{33}\rvert$	408.146 MPa	432.819 MPa
$\lvert\sigma_{12}\rvert$	1.590 MPa	3.262 MPa
$\lvert\sigma_{13}\rvert$	1.498 MPa	1.861 MPa
$\lvert\sigma_{23}\rvert$	2.289 MPa	9.342 MPa

So the discrepancy is not caused by the Dirichlet conditions. In Fig.4.15a, we can see that the generation of APS is focused on the top surface. The decrease along the axial direction implies that the Dirichlet condition does not affect the middle of the tube. This can be investigated by applying the Dirichlet conditions also on the curved surfaces, this produced the APS distribution seen in Fig.4.16. The distribution was created with $\kappa_y = 1.0\text{m}^{-1}$ and strain increment set to $1.0e^{-4}\%$
We can see that the APS distribution in Fig.4.16 and Fig.4.15b coincide. So for long tubes, the bending displacement requires additional Dirichlet conditions on the curved sides.



(a) Shows the APS distribution for $L = 50$mm.



(b) Shows the APS distribution for $L = 10$mm.

Figure 4.15: Shows the APS distribution for two different lengths with $\kappa_x = 1.0\text{m}^{-1}$. The strain increment was set to $1.0e^{-4}\%$ and the tube shape will remain unbent, since we consider a static mesh.

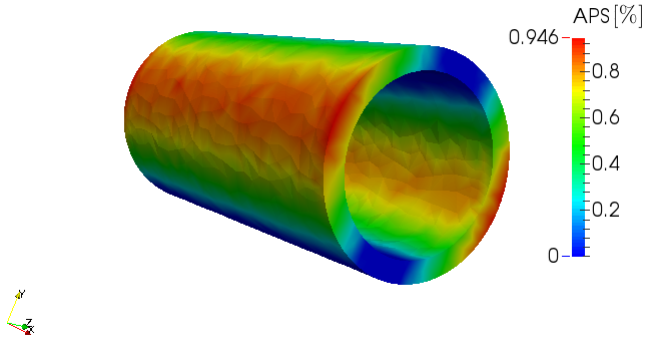


Figure 4.16: Shows the APS distribution for $L = 50$mm, and $\kappa_y = 1.0\text{m}^{-1}$. With the Dirichlet conditions on all the surfaces.

4.5.2 Axial Tension

We will now investigate the generation of APS in the case of axial tension. The step input for axial force was set to 106.2kN and the strain increment was set to $1.0e^{-4}\%$. These inputs produced the results seen in Tab.4.10, for different lengths L of the mesh.

Table 4.10: Shows the APS value for different lengths L. The step input was $\kappa_y = 1.0\text{m}^{-1}$ and strain increment $1.0e^{-4}\%$.

L [mm]	APS [%]
2.0	0.0802
10.0	0.0809
30.0	0.0601
50.0	0.0472

In Tab.4.10, we see that by increasing the length of the mesh, we will decrease the APS value. The analytical approximation has been calculated as 0.097%, which was used to create Fig.4.17. The figure shows the relative error in the APS for different lengths of the mesh.

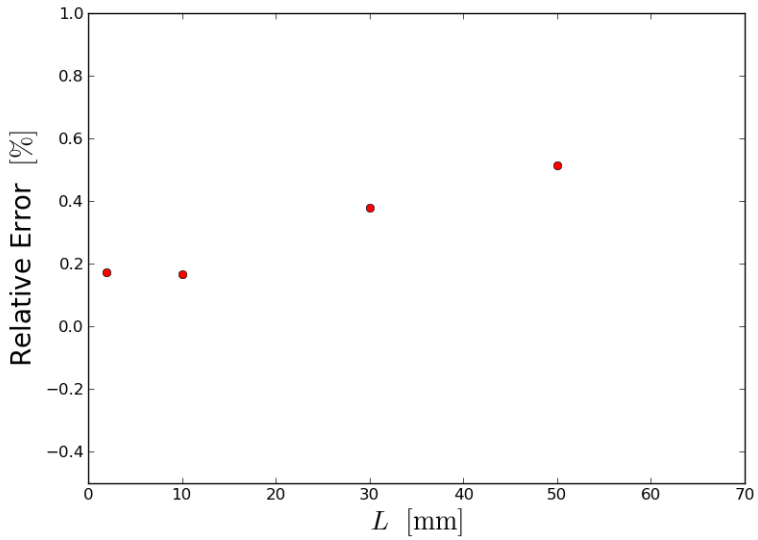
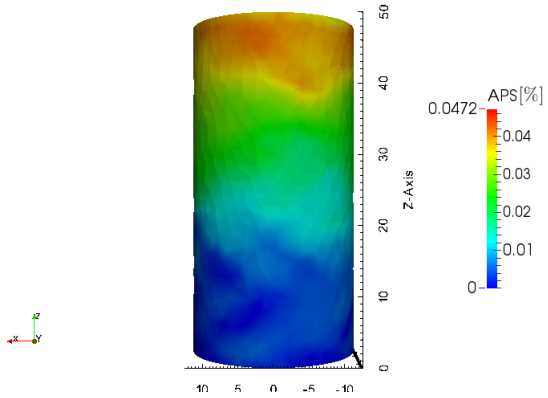
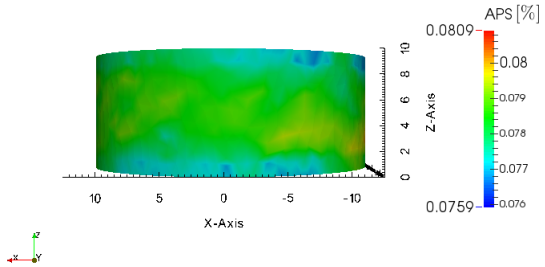


Figure 4.17: Shows relative error between the values in Tab.4.10 and the analytical approximation of APS value 0.097%. The input parameter for axial force was 106.2.0kN and strain increment of $1.0e^{-4}\%$.

We see that in Fig.4.17, the relative error increases close to linearly with the length of the mesh. The APS distribution for the cases of $L = 10$mm and $L = 50$mm
In Fig.4.18b, we can see that the generation of APS is close to constant. While in Fig.4.18a the maximum APS is generated at the top surface and decreases along the axial direction of the mesh. This is similar to the case of bending, so we added



(a) Shows the APS distribution for $L = 50$mm.



(b) Shows the APS distribution for $L = 10$mm.

Figure 4.18: Shows the APS distribution for two different lengths for an axial force of 106.2kN. The strain increment was set to $1.0e^{-4}$%.

Dirichlet conditions on the curved sides, which generated the APS distribution in Fig.4.19.

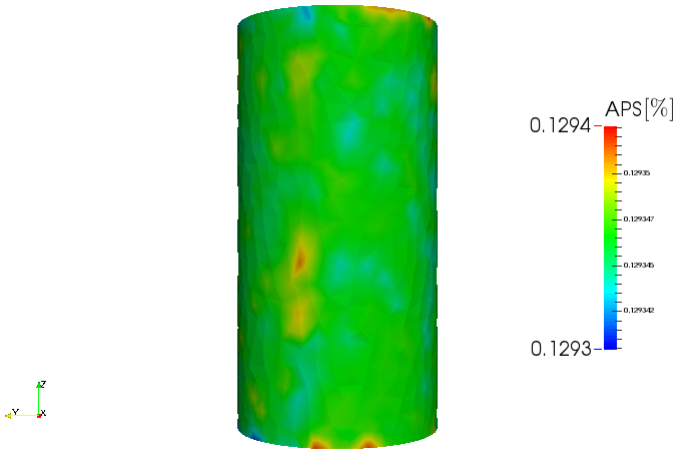


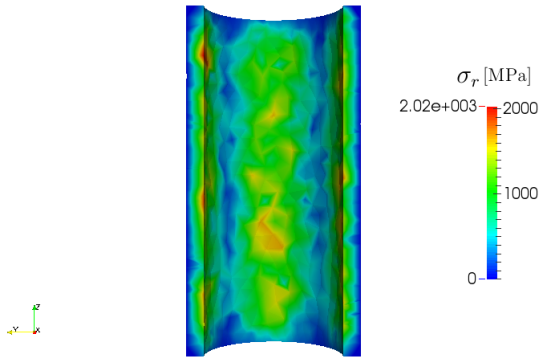
Figure 4.19: Shows the APS distribution with Dirichlet conditions on all the surfaces. The input parameter were axial force 106.2.0kN, strain increment of $1.0e^{-4}$% and mesh length of 50mm.

We can see that the APS distribution in Fig.4.19 is constant in the axial direction. However, the maximum APS value is larger than for smaller lengths. This was due to approximation of the cross sectional, which was 176.55mm^2. So by performing the corrections, we obtain that the maximum APS value is 0.0740%, similar to the corrected values in Tab.4.3.
The reason for the decrease of APS along the axial direction is of the length between the Dirichlet boundaries. The increased length will generate more nodes in the axial

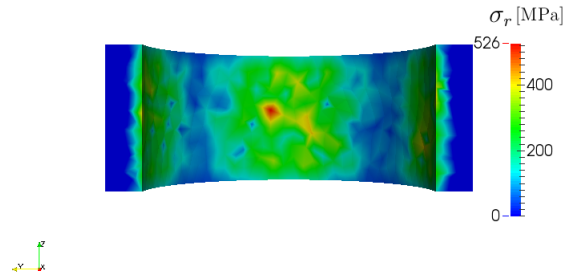
direction, so that the Dirichlet boundaries do not enforce the axial displacement throughout the mesh.

4.5.3 Internal Pressure

The elastic radial displacement causes for unwanted behaviour in the plastic region. So looking at the length dependence of the APS value will not be useful. However, it can be beneficial to look at the difference in radial stress distribution.



(a) Shows the radial stress distribution for $L = 50$mm.



(b) Shows the radial stress distribution for $L = 10$mm.

Figure 4.20: Shows radial stress distribution for two different lengths with an internal pressure of 220MPa. The strain increment was set to $1.0e^{-5}$%.

In both Fig.4.20b and Fig.4.20a, we can see that stress values are highest close to the x and y axis. We also see a significant increase in the maximum radial stress, which means that using the Eq.3.15 is not preferred indicator of the internal pressure for any lengths.

4.6 Deformation Invariance

The implementation is based on decomposing the combined stress into two separate deformations of bending and axial tension. Since there are no restrictions to the order of the decomposition, we would expect an invariance of the deformation order. So to determine if the decomposition is valid, we will investigate if the order is invariant. We will do this by comparing the results from different combinations of deformation order.
We will look at a sequence of three deformation steps, which include two bending deformations and one axial tension deformation. The bending deformations are bending and unbending with input $\kappa_x = 1.0\text{m}^{-1}$. We set the input for axial tension to 10kN. The explicit deformation inputs are given in Tab.B.1, and the corresponding results are given in Tab.4.11.

Table 4.11: The APS produced for the deformation steps seen in Tab.B.1, with $\Delta\epsilon = 1.0e^{-1}\%$,$\rho = 3$ $^{nodes}/_{\text{mm}^3}$ and $l = 10$ $^{\text{faces}}/_{\text{mm}}$.

(a) Shows the produced APS for the deformation steps given in Tab.B.1a.

Step	APS [%]
1	0.941
2	1.573
3	1.614

(b) Shows the produced APS for the deformation steps given in Tab.B.1b.

Step	APS [%]
1	0.000
2	0.945
3	1.581

(c) Shows the produced APS for the deformation steps given in Tab.B.1c.

Step	APS [%]
1	0.942
2	0.991
3	1.651

The final APS value results in Tab.4.11a, Tab.4.11b and Tab.4.11c do not coincide. This indicates that the order is not invariant.
The different APS values are caused by applying an axial force on different stress states. In Tab.4.11b, the axial force on the initial tube. The initial stress state is zero, thus the axial force will not generate any APS. While in Tab.4.11a the stress state on the on the sides tube is on the yield surface and therefore will generate APS.
The fact that the deformation is variant to the order coincides with large deformation theory. Which states that the plastic deformation is dependent on the order, see [6].

4.7 Dynamic Mesh

So far we have considered the results by using a static mesh, making it possible to approximate the results analytically. However, a static mesh do not reassemble real case of deformation, where the mesh changes infinitesimal.
Therefore we will consider a dynamic mesh, where the mesh is moved incrementally. We will compare the difference in the results for bending and axial tension. However, the implementation of internal pressure is based on static Dirichlet conditions and on a static mesh.
The static solutions will be marked with a subscript s and the dynamic solutions with subscript d. We set the strain increment to $1.0e^{-4}\%$, and selected $\rho = 3$ $^{nodes}/_{\text{mm}^3}$ and $l = 10$ $^{\text{faces}}/_{\text{mm}}$.

4.7.1 Bending

First, we will look at the difference between static and dynamic APS values for bending. The results were produced with different values of κ_y and are shown in Tab.4.12.
We can see that for increased curvature, the difference between static and dynamic APS values increases. However, the relative difference will remain small, so the difference can be neglected.

Table 4.12: Shows the APS value for dynamic and static mesh, for different values of axial force and strain increment $1.0e^{-4}$%.The analytical approximations are marked with subscript a and listed for reference.

κ_y m^{-1}	APS$_s$ [%]	APS$_d$ [%]	ΔAPS [%]	APS$_a$ [%]
0.3	0.072	0.073	0.001	0.075
0.5	0.322	0.321	0.001	0.325
1.0	0.943	0.958	0.015	0.950
2.0	2.182	2.201	0.019	2.200

4.7.2 Axial Tension

We will now investigate the difference for axial tension, which produced the results seen in Tab.4.13.

Table 4.13: Shows the APS value for dynamic and static mesh, for different values of axial force. The input of strain increment was set to $1.0e^{-4}$% and $\rho = 3\ ^{nodes}/_{\text{mm}^3}$ and $l = 10\ ^{\text{faces}}/_{\text{mm}}$. The analytical approximations are marked with subscript a and listed for reference.

Axial force [kN]	APS$_s$ [%]	APS$_d$ [%]	ΔAPS [%]	A$_a$ [%]
106.2	0.080	0.476	0.396	0.096
106.4	0.193	0.766	0.573	0.210
106.6	0.306	1.058	0.752	0.323

We observe that the difference between static and dynamic in Tab.4.13 is significant. And the reason is due to incremental movement of the mesh.
The axial force will generate an axial displacement, but the continuity condition require that the volume remain constant. Therefore we will have displacements in other directions, and as the height increases, the cross-sectional area will decrease. We have seen the effects due to small changes of the cross-sectional area, which cause the APS value to increase.
The Poisson's ratio of 0.3 will give the material some compressibility but only in the elastic region. Thus the decrease in the area of the cross section is mostly an effect that occurs when the tube is fully plastic. So the difference is small for partial plastic tubes.
This can be seen in Fig.4.21, which shows the final APS distribution after the three deformation steps defined in Tab.B.1a. We set the mesh is to be updated incrementally and set the strain increment to $1.0e^{-4}$%. We see that the maximum value in the APS distribution in Fig.4.21 is 1.60 %. So the maximum APS produced with dynamic mesh is lower than for a static mesh, but the relative error is only 0.8%.

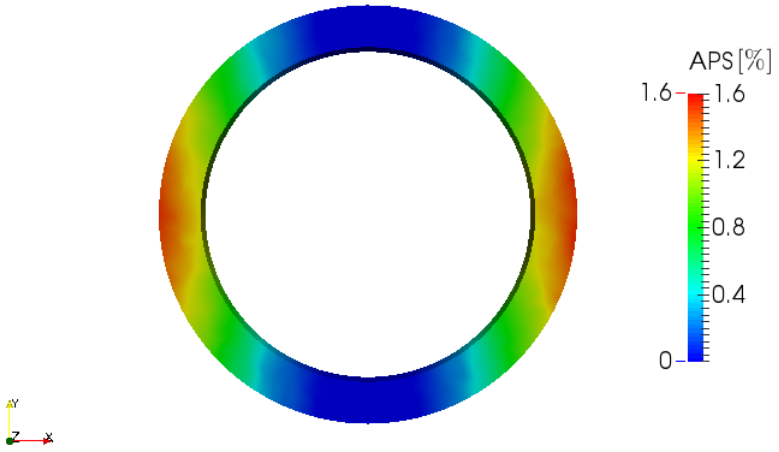


Figure 4.21: Shows relative error between the values in Tab.4.10 and the analytical approximation of APS value 0.097%. The input parameters are given in Tab.B.1a and strain increment was set to $1.0e^{-4}$%.

4.8 Computational Time

The computational time will vary for different computers, and we will therefore investigate the computational time dependence on the number of nodes and the number of increments. This will be done for linear elements, so that the number of nodes equals the number of points. The computation was done on a commercial computer with a Linux operating system, and hardware specifies listed in Tab.4.14.

Table 4.14: Shows the hardware specifies in the computer that computed the results.

Component	Value
Memory	7.8 GiB
Processor	AMD FX(tm)-8350 Eight-Core Processors x8
OS-type	64-bit

We see that the curves in both Fig.4.22 and Fig.4.23 seem to be linear. However, we see a small deviation in Fig.4.23. So we will assume that the curve has the shape of $y = ax^n$, and use logarithmic linear regression to determine n. The procedure was done using python and produced $n = 1.21$.
The assumption of $y = ax^n$ is not valid in Fig.4.22, since the curve does not intersect with origo. This is due to computational time required to construct the mesh. We used linear regression on Fig.4.22, which had very low error, so we assume that the relation is linear, i.e. $n = 1$.

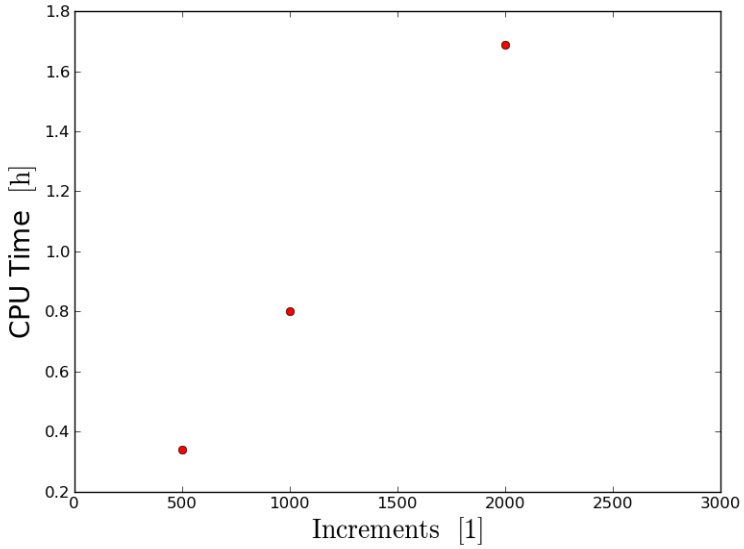


Figure 4.22: Shows the CPU time for various numbers of increments, with the number of nodes at 1000.

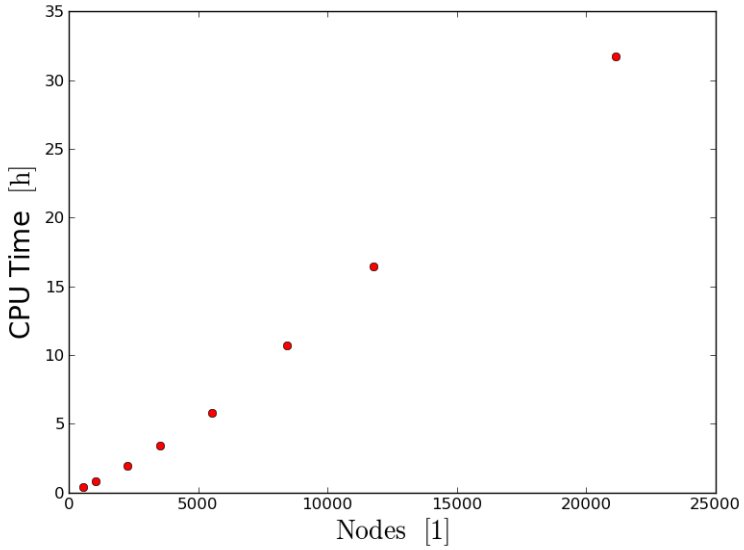


Figure 4.23: Shows the CPU time for various numbers of nodes, with the number of increments at 1000.

4.9 Default Parameters

We have seen how the results are affected by different parameters and we will now recommend some default parameters.
The variation of length caused large deviations in the APS values, but this was solved by adding additional Dirichlet conditions on the curved surfaces. Thus we will recommend a length of 2.0mm, since it will give less computation time.
In case of bending we have seen that the generation of APS is dependent on the nodal density. The relation between damping and l makes it possible to decrease the resolution and still obtain a converged result. The mesh also affected the deformation of axial tension through the approximation of the cross-sectional area, which resulted in greater generation of APS. So we set the number of faces to 250 and the resolution to 30. This will correspond to $\rho = 3\ {}^{\text{nodes}}/_{\text{mm}^3}$ and $l = 10\ {}^{\text{faces}}/_{\text{mm}}$ with the geometrical dimensions given in Tab.4.1.
The variation of the strain increment caused the closest-point projection convergence failure, so that we obtained an incorrect value of the plastic multiplier. This had minor effects concerning bending, but for axial tension it gave a large discrepancy in the APS values. It was further shown that the relative error had a linear relation to the strain increment. Therefore we will recommend the strain increment $1.0e^{-4}\%$, which would cause minor deviation and reasonable computation time.
We have seen that dynamic mesh had a large effect on the generation of plastic strain for axial tension. This is due to the decrease in the cross-sectional area caused by the continuity condition. However, this effect is mostly present when the tube is fully plastic, so the effects are minor for partially elastic tubes. Thus the option gives minor changes , but it is more realistic with a dynamic mesh.
We can not conclude that these parameters will give the optimal results for other tubes, since we have only considered one tube.

4.10 Comparison with Nexans' APS Program

We will in this section compare the results with Nexans' current APS program. The program is not based on the finite element method, but uses a piecewise stress-strain relation. We will only compare the results, since the implementation is confidential. The comparison is seen in Tab.4.15 and is done for a static mesh and with the recommended parameters.
In the table we see that the relative error is less than 1.0% for all the different sequences. We can therefore conclude that the results coincide.

Table 4.15: Shows the input parameters with the available option and the functionality.

Deformation Steps	Deformations in Sequence	APS computed by Nexans' APS program [%]	APS computed by the program of this thesis [%]	Relative Error [%]
Tab.B.2	Bending	4.069	4.066	0.07
Tab.B.3	Bending	6.545	6.557	0.2
Tab.B.4	Bending,axial tension	5.582	5.560	0.4
Tab.B.1a	Bending, axial tension	1.624	1.614	0.6
Tab.B.1b	Bending, axial tension	1.583	1.581	0.1
Tab.B.1c	Bending, axial tension	1.666	1.651	0.9

Chapter 5

Conclusion

We have looked at the results produced by the APS program with the deformations of axial tension, bending and internal pressure. The results were computed on a umbilical tube with the general parameters given in Tab.4.1, while the construction of the mesh and strain increments varied. We used linear Lagrange as elements in the finite element method.
The construction of the mesh involves randomization, which required that we investigated the consistency of the computed results. We observed small differences in the results, but the coefficient of variation was less than 1.0%, thus we concluded that the results were consistent.
The results converged with higher nodal density for the different deformations, but for bending, the convergence had a shape of a damped oscillation. Therefore we looked at the convergence of quadratic elements, which indicated that the oscillations were due to the use of linear Lagrange elements.
In the case of axial tension, we observed that lower values of l generate more APS. This was caused by the approximation of the cross-sectional area, and the corrected results gave equal values.
We have seen how the variation of the strain increment effects the generation of plastic strain for the different deformations. The effect on the axial tension proved to be most significant, due to the convergence failure in the closest-point projection. This failure causes the stress not to be mapped onto the yield surface, which subsequently causes the while-loop to end prematurely. We observed that the strain increment of $1.0e^{-4}\%$ did not cause convergence failure.
In the case of internal pressure we have seen that the radial stress distribution was not uniform, which is not in accordance with the presented theory. Therefore we can conclude that the implementation for internal pressure needs to be improved.
We have seen that an incremental moving mesh affected the generation of APS due to a decrease in the cross-sectional area. This effect becomes more apparent for a large axial force and a tube that is fully plastic.
In the implementation we decomposed the combination of axial tension and bending into two separate processes. There was no restriction to the order of the decomposition, so we assumed that the order was invariant. This assumption was tested by

looking at the results for different order combinations of axial tension and bending. The computed results did not coincide, thus proved that the assumption was invalid. We used the observed results to recommend some default parameters to the program. In the recommendations the computation time became a factor, and therefore we looked at the computation time dependence on the number of nodes and increments. The dependence on the number of increments was linear,while the dependence on the number of nodes had the form of $y = ax^n$ with $n = 1.21$. This resulted in the recommendation given in section 4.9.
The results for different deformation sequences, containing axial tension and bending, was compared with Nexans' current APS program. There were some small deviations in the results, but the relative error was less than 1.0 %.

Chapter 6

Future work

We concluded that the implementation of the internal pressure needs improvement. The implementation should use Neumann conditions, see [8], but the focus should be on implementing a uniform radial stress distribution. This may require the use of other element than linear Lagrange, so a further understanding of the finite element method may be required, see [8].
The APS-program generates a significant amount of output, and assessing each deformation step individually can be time consuming. So the APS program should generate a report based on the most significant output.
The implementation of the return mapping algorithm is based on a closest-point projection, which has problems converging for large strain increments. This results in longer computation due to increased number of increments. Therefore the implementation of a globally convergent closest-point projection algorithm can decrease the number of increments and subsequently the computation time. Since the convergence issue is a well documented problem, there are algorithms for global convergence, see [2].
We implemented a simple plastic model, so that the APS values could be analytically approximated. This meant that we could asses the computed results. However, the model may not correspond to experimental results. Therefore a validation based on experimental results is preferred. This may require the implementation of another plastic model, if the results do not correspond with the experimental results.
The implementation did not include the presence of back-stress and this can be implemented following [9].
The input of resolution and faces can be transformed into the input of nodal density and linear approximation quality. This requires that we look at the convergence of tubes with different sizes.

References

[1] F. Armero and A. Perez-Foguet. On the formulation of closest-point projection algorithms in elastoplasticity. part i: The variational structure. *International Journal for Numerical Methods in Engineering*, 53(2), 2002.

[2] F. Armero and A. Perez-Foguet. On the formulation of closest-point projection algorithms in elastoplasticity. part ii: Globally convergent schemes. *International Journal for Numerical Methods in Engineering*, 53(2), 2002.

[3] A. Henderson. *A Parallel Visualization Application*. Kitware Inc, 2007.

[4] M. Jirasek and Z. P. Bazant. *Inelastic Analysis of Structures*, chapter 20. John Wiley & Sons, 2002.

[5] A. Logg, K.-A. Mardal, G. N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[6] J. Lubliner. *Plasticity Theory*. Dover Publicatins Inc, 2008.

[7] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[8] A. Quarteroni. *Numerical models for Differential Problems*. Springer-verlag Italia, 2008.

[9] J. C. Simo and R. L. Taylor. A return mapping algorithm for plane stress elastoplasticity. *International Journal for Numerical Methods in Engineering*, 22(3):649–670, March 1986.

[10] L. M. Valnes. *Stress and Strain in Elastic Tubes with the Finite Element Method*. Project Thesis, NTNU, 2013.

Appendices

Appendix A

Flow Chart

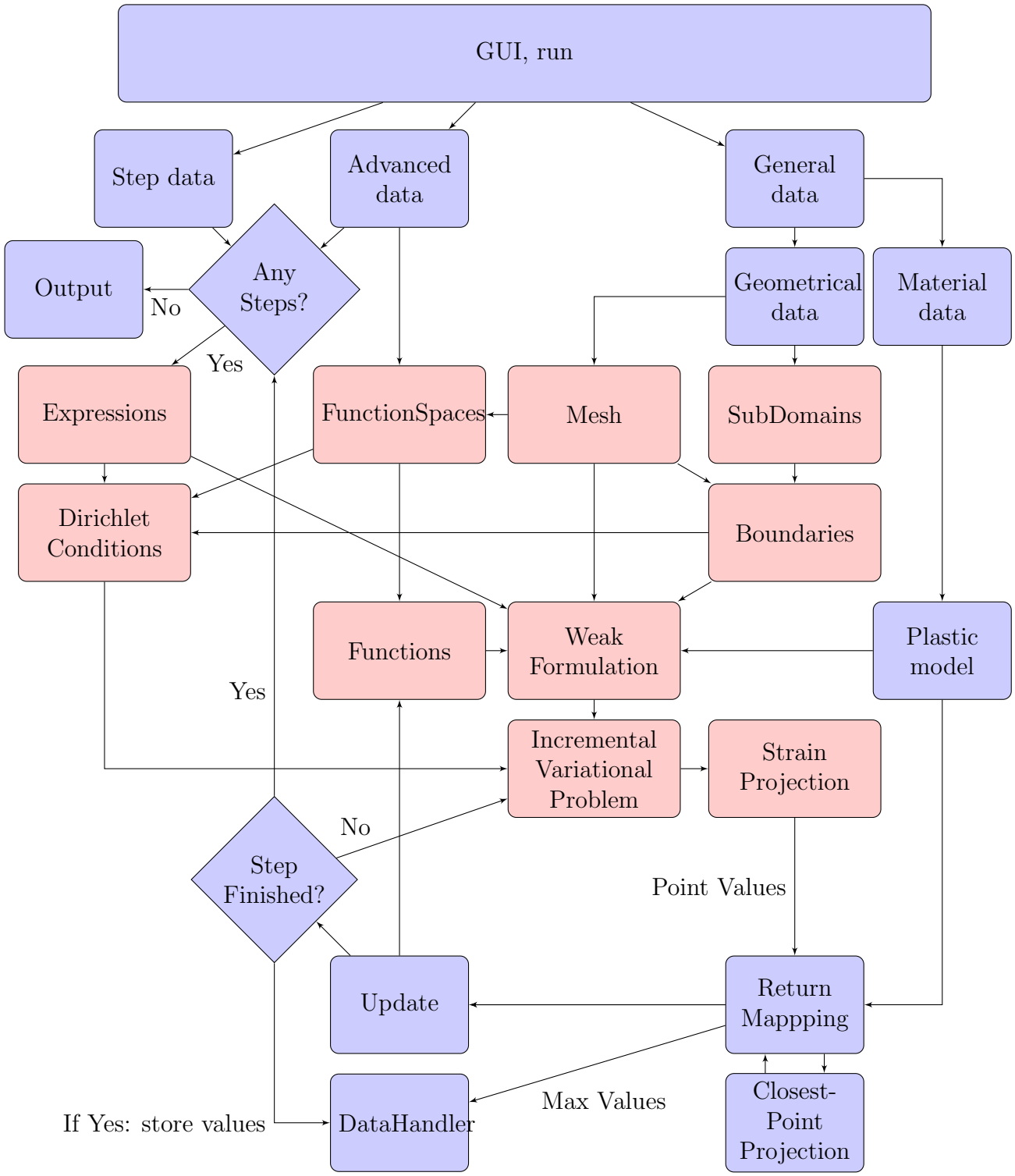


Figure A.1: Shows a flowchart of the APS program, red blocks represent FEniCS classes and functions. The interaction between FEniCS functions and classes are more complicated than what is shown.

Appendix B

Deformation Steps

Table B.1: Shows the input for deformation steps related to the section Order Invariance

(a) Shows the inputs for the first order of deformation steps.

κ_y [m^{-1}]	κ_x [m^{-1}]	Internal Pressure [MPa]	Axial Force [kN]
1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	10.0

(b) Shows the inputs for the second order of deformation steps.

κ_y [m^{-1}]	κ_x [m^{-1}]	Internal Pressure [MPa]	Axial Force [kN]
0.0	0.0	0.0	10.0
1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

(c) Shows the inputs for the third order of deformation steps.

κ_y [m^{-1}]	κ_x [m^{-1}]	Internal Pressure [MPa]	Axial Force [kN]
1.0	0.0	0.0	0.0
1.0	0.0	0.0	10.0
0.0	0.0	0.0	0.0

Table B.2: Shows the specific inputs for a deformation sequence.

κ_y [m^{-1}]	κ_x [m^{-1}]	Internal Pressure [MPa]	Axial Force [kN]
0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Table B.3: Shows the specific inputs for a deformation sequence.

κ_y [m^{-1}]	κ_x [m^{-1}]	Internal Pressure [MPa]	Axial Force [kN]
0.0	0.0	0.0	0.0
0.5	0.0	0.0	0.0
1.0	0.0	0.0	0.0
1.5	0.0	0.0	0.0
2.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
-0.5	0.0	0.0	0.0
-1.0	0.0	0.0	0.0
-1.5	0.0	0.0	0.0
-2.0	0.0	0.0	0.0

Table B.4: Shows the specific inputs for a deformation sequence.

κ_y [m^{-1}]	κ_x [m^{-1}]	Internal Pressure [MPa]	Axial Force [kN]
1.0	0.0	0.0	0.0
1.0	1.0	0.0	0.0
0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	-1.0	0.0	0.0
-1.0	-1.0	0.0	0.0
-1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	20.0

Appendix C

Input Overview

Table C.1: Shows the input parameters with the available option and the functionality.

Name	Options	Location	Function
Material Name	String	Main Frame	Stores the material name for later use.
Young Modulus	Float	Main Frame	Sets the Young Modulus in the computation.
Work-Hardening Modulus	Float	Main Frame	Sets the work-hardening modulus in the computation
SMYS (Yield Strength)	Float	Main Frame	Sets the yield strength in the computation
Poission's ratio	0.0 -0.5	Main Frame	Sets the Poisson's ratios in the computation.
Outer Diameter	Float	Main Frame	Sets the outer geometrical values of the mesh
Wall Thickness	Float	Main Frame	Sets the wall thickness in the computation.
Base	Float	Adv. Frame	Set the lower z-position of the tube in a Cartesian coordinate system.
Length	Float	Adv. Frame	Sets the length of the tube slice.
Facets	Integer	Adv. Frame	Used in the construction of the mesh
Resolution	Integer	Adv. Frame	Sets the Young Modulus in the computation.
Strain Increment	float	Adv. Frame	Sets the strain increment in the computation.
Quadrature	1 or 2	Adv. Frame	The use of linear of quadratic elements.
Debug	True/False	Adv. Frame	Allows for additional data during the computation.
Update of Mesh	Static or Increment	Adv. Frame	Specifies if the nodes are move based on the incremental displacements.
End Conditions	Closed or Open	Adv. Frame	Specific if the tube is closed-ended or open-ended for the internal pressure.

Appendix D

Code

Appendix D.A Graphic User Interface

```
import wx as wx
from wx.lib.mixins.listctrl import  TextEditMixin
import xml.etree.cElementTree as ET


class CustomApp(wx.App):
    def __init__(self):
        wx.App.__init__(self)
        self.steps=None
    def SetInput(self, Steps,Values):
        self.settings = dict(self.settings.items() +Values.
            items())
        self.steps  = Steps
    def SetAdvanced(self,array):
        self.settings=array


class AdvancedFrame(wx.Frame):
      def __init__(self,parent):
            wx.Frame.__init__(self, parent)
            vbox=wx.BoxSizer(wx.VERTICAL)
            panel = wx.Panel(self)
            self.boxes=[ CustomBox(panel,default="2.0", label
                ="Length (mm)"),
                         CustomBox(panel,default="0.0", label
                             ="Base"),
                         CustomBox(panel,default="1.0e-6",
                             label="Increment of strain"),
```

```
                CustomBox(panel,default="False",
                    label="Debug",choices=["True","
                    False"]),
                CustomBox(panel,default="250", label
                    ="Faces"),
                CustomBox(panel,default="30", label=
                    "Resolution"),
                CustomBox(panel,default="1", label="
                    Quadrature",choices=["1","2"]),
                CustomBox(panel,default="Static",
                    label="Update of Mesh",choices=["
                    Increment","Static"]),
                CustomBox(panel,default="Closed",
                    label="End Condition",choices=["
                    Open","Closed"])]


        btn=wx.Button(panel,100,"Ok",size=(90,30))
        self.Bind(wx.EVT_BUTTON, self.OnQuit,id=100)
        dfl=wx.Button(panel,99,"Reset",size=(90,30))
        self.Bind(wx.EVT_BUTTON, self.OnDefault,id=99)
        vbox.AddSpacer((-1,20))

        for i in self.boxes:
            vbox.Add(i,1,wx.ALL|wx.EXPAND|wx.CENTER,5)

        hbox=wx.BoxSizer(wx.HORIZONTAL)
        hbox.Add(dfl,0,wx.ALIGN_RIGHT|wx.ALL,10)
        hbox.Add(btn,0,wx.ALIGN_RIGHT|wx.ALL,10)


        vbox.Add(hbox,0,wx.ALIGN_RIGHT|wx.ALL,10)
        vbox.Fit(self)
        panel.SetSizer(vbox)

        self.Center()
        self.parent=parent
    def OnQuit(self,event):
        if self.SetValues()==True:
            self.parent.adv_frame=None
            self.Close()
    def OnDefault(self,event):
        [i.SetDefault() for i in self.boxes]
```

```
    def GetValues(self):
        return { i.GetLabel().split()[0]:i.GetValue() for
            i in self.boxes}

    def SetValues(self):
            try:
                wx.GetApp().SetAdvanced(self.GetValues())
                return True
            except:
                wx.MessageBox("Please insert correct input
                    values", "Incorrect Values",wx.OK)
                return False
    def SetDefaults(self):
        wx.GetApp().settings=self.GetValues()
        self.Close()



class CustomBox(wx.BoxSizer):
    def __init__(self, panel, default="0",choices=[],*args
        ,**kwargs):
        super(CustomBox,self).__init__(wx.HORIZONTAL)
        if not choices :
            self.value=wx.TextCtrl(panel,value=default)
        else:
            self.value=wx.ComboBox(panel,value=default,
                choices=choices)
        self.default=default
        self.Rform()

        self.AddSpacer((50,30))
        self.Add(wx.StaticText(panel,*args,**kwargs) , 1, wx
            .ALL, 5)
        self.AddSpacer((50,30))
        self.Add(self.value, 1, wx.ALL|wx.EXPAND, 5)
        self.label=kwargs.get("label")
    def Rform(self):
        try:
            float(self.default)
            self.flag=True
        except:
            self.flag=False
```

```
    def SetValue(self,value):
        self.value.ChangeValue(value)
    def SetDefault(self):
        try:
            self.value.ChangeValue(self.default)
        except:
            self.value.SetValue(self.default)
    def GetValue(self):
        if self.flag==True:
                return float(self.value.GetValue())
        else :
                return str(self.value.GetValue())

    def GetLabel(self):
        return self.label


class EditableListCtrl(wx.ListCtrl, TextEditMixin):
    def __init__(self, parent):
        wx.ListCtrl.__init__(self, parent, -1, style=wx.
            LC_REPORT,size=(500,-1))
        TextEditMixin.__init__(self)


class User_interface(wx.Frame):
    def __init__(self, *args, **kwargs):
        super(User_interface, self).__init__(*args, **kwargs
            )
        self.config=wx.Config("APS")
        self.dirname=self.config.Read("Path")
        self.output_folder=str(self.config.Read("
            Output_folder"))

        self.InitUI()

    def InitUI(self):

        Main_pnl = wx.Panel(self, -1)
        AdvancedFrame(self).SetDefaults()

##############################################################
#                          PANEL 1
##############################################################
        pnl1=wx.Panel(Main_pnl,-1)
```

```
        vbox1=wx.BoxSizer(wx.VERTICAL)
        self.boxes=[CustomBox(pnl1, default="Insert material
            name", label='Material'),
                    CustomBox(pnl1, default="200.0" ,label='
                        Young Modulus    (GPa)'),
                    CustomBox(pnl1, default="1.0" ,label='
                        Work-Hardening Modulus (GPa)'),
                    CustomBox(pnl1, default="25.0",label='
                        Outer Diameter (mm)'),
                    CustomBox(pnl1, default="2.5",label='
                        Wall Thickness (mm)'),
                    CustomBox(pnl1, default="0.3", label='
                        Poisson ratio '),
                    CustomBox(pnl1, default="600",label='
                        SMYS (MPa)')]

        vbox1.AddSpacer((-1,20))
        [vbox1.Add(i,1,wx.ALL|wx.EXPAND|wx.CENTER,5) for i
            in self.boxes]
        adv=wx.Button(pnl1,27,'Advanced',size=(90,30))
        vbox1.Add(adv,0,flag=wx.ALIGN_RIGHT, border=10)
        pnl1.SetSizer(vbox1)
##############################################################
#                          Panel 2
##############################################################

        pnl2= wx.Panel(Main_pnl,-1)
        hbox2=wx.BoxSizer(wx.HORIZONTAL)

        vbox2=wx.BoxSizer(wx.VERTICAL)

        text=wx.StaticText(pnl2,label="    Select output
            folder :")
        brw=wx.Button(pnl2,19,'Browser',size=(90,30))

        self.brw_tc=wx.TextCtrl(pnl2,size=(450,-1),value=
            self.output_folder)



        hbox2.Add(self.brw_tc,0.1, wx.LEFT|wx.EXPAND,
            border=10)
        hbox2.Add((20, -1),1,wx.EXPAND)
```

```
        hbox2.Add(brw,0, wx.ALIGN_RIGHT|wx.EXPAND ,border
            =10)
        vbox2.Add(text,1,wx.LEFT|wx.ALL|wx.EXPAND)
        vbox2.Add(hbox2)
        vbox2.Add((-1,20))


        pnl2.SetSizer(vbox2)
##############################################################
#                          PANEL 3
##############################################################
        hbox = wx.BoxSizer(wx.HORIZONTAL)
        pnl3=wx.Panel(Main_pnl,-1)
        self.list = EditableListCtrl(pnl3)
        self.list.InsertColumn(0, 'Nr.', width=30)
        self.list.InsertColumn(1, 'Curvature Y (m'+u"\u207B"
            +u"\u00B9"+')', width=130)
        self.list.InsertColumn(2, 'Curvature X (m'+u"\u207B"
            +u"\u00B9"+')', width=130)
        self.list.InsertColumn(3, 'Pressure (MPA)', width
            =120)
        self.list.InsertColumn(4, 'Tension (kN)', width= 90)

        hbox.Add(self.list, 1,flag=wx.EXPAND|wx.ALL, border
            =50)

        btnPanel = wx.Panel(pnl3, -1)
        new = wx.Button(btnPanel, 20, 'New', size=(90, 30))
        dlt = wx.Button(btnPanel, 21, 'Delete', size=(90,
            30))
        clr = wx.Button(btnPanel, 22, 'Clear', size=(90, 30)
            )



        vbox = wx.BoxSizer(wx.VERTICAL)
        vbox.Add((-1, 130))
        vbox.Add(new,0,wx.TOP,5)
        vbox.Add(dlt, 0, wx.TOP, 5)
        vbox.Add(clr, 0, wx.TOP, 5)

        vbox.Add((-1,150))
```

```
        btnPanel.SetSizer(vbox)

        hbox.Add(btnPanel, 0, flag=wx.ALIGN_RIGHT|wx.RIGHT|
            wx.EXPAND ,border= 20)

        pnl3.SetSizer(hbox)

##############################################################
#                          PANEL 4
##############################################################

        pnl4= wx.Panel(Main_pnl,-1)
        hbox4=wx.BoxSizer(wx.HORIZONTAL)
        save=wx.Button(pnl4,23,'Save',size=(90,30))
        load=wx.Button(pnl4,24,'Load',size=(90,30))
        run = wx.Button(pnl4, 25, 'Run', size=(90, 30))
        close = wx.Button(pnl4, 26, 'Close', size=(90, 30))


        hbox4.Add(close,0,flag=wx.BOTTOM|wx.ALL,border=10)

        hbox4.Add(save, 0 ,flag=wx.BOTTOM|wx.ALL, border=10)
        hbox4.Add(load, 0 ,flag=wx.BOTTOM|wx.ALL, border=10)
        hbox4.Add(run,0,flag=wx.BOTTOM|wx.ALL, border=10)

        pnl4.SetSizer(hbox4)


##############################################################
#                          APPERANCE
##############################################################
        main_vbox=wx.BoxSizer(wx.VERTICAL)

        main_vbox.Add(pnl1,flag=wx.ALL|wx.EXPAND ,border=30)
        main_vbox.Add(pnl2,flag=wx.ALIGN_RIGHT|wx.RIGHT |wx.
            EXPAND|wx.ALL,border=10)
        main_vbox.Add(pnl3,flag= wx.EXPAND,border=30)
        main_vbox.Add(pnl4,flag=wx.ALIGN_RIGHT|wx.BOTTOM,
            border=30)

        main_vbox.Fit(self)
        Main_pnl.SetSizer(main_vbox)

        self.Bind(wx.EVT_BUTTON,self.OnBrowser,id=19)
```

```
        self.Bind(wx.EVT_BUTTON, self.NewItem, id=20)
        self.Bind(wx.EVT_BUTTON, self.OnDelete, id=21)
        self.Bind(wx.EVT_BUTTON, self.OnClear, id=22)
        self.Bind(wx.EVT_BUTTON, self.OnSave, id=23)
        self.Bind(wx.EVT_BUTTON, self.OnLoad, id=24)

        self.Bind(wx.EVT_BUTTON, self.OnRun, id=25)
        self.Bind(wx.EVT_BUTTON, self.OnClose, id=26)
        self.Bind(wx.EVT_BUTTON, self.OnAdvanced, id=27)



        self.adv_frame=None
        self.count=0
        self.SetTitle('Accumulated Plastic Strain User
            Interface')
        self.Centre()
        self.Show(True)
#####################################################################
#                              EVENT FUNCTIONS
#####################################################################


    def OnAdvanced(self,event):
        self.adv_frame=AdvancedFrame(self)
        self.adv_frame.Show()
    def NewItem(self, event):
        self.count+=1
        self.list.Append([int(self.count),0.0,0.0,0.0,0.0])

    def OnDelete(self, event):
        if self.count>0:
            self.count-=1
            self.list.DeleteItem(self.count)
    def OnClear(self, event):
        self.count=0
        self.list.DeleteAllItems()

    def OnClose(self,e):
        if self.adv_frame!=None:
            self.adv_frame.Close()
        self.Destroy()
```

```
    def OnRun(self,e):
        if self.SetData():
            if self.adv_frame!=None:
                self.adv_frame.Close()
            self.Destroy()

    def OnBrowser(self,event):
        dialog = wx.DirDialog(None, "Choose a directory:",
           style=wx.DD_DEFAULT_STYLE | wx.DD_NEW_DIR_BUTTON)
        if dialog.ShowModal() == wx.ID_OK:
            self.output_folder=str(dialog.GetPath())
            self.config.DeleteGroup("Output_folder")
            self.config.Write("Output_folder",dialog.GetPath
               ())
            self.brw_tc.ChangeValue(dialog.GetPath())

        dialog.Destroy()


    def OnLoad(self,event):
         self.LoadXML()


    def OnSave(self,event):
        self.WriteXML()

##############################################################
#                  Other Functions
##############################################################
    def SetSteps(self,Steps):
        self.count=0
        self.list.DeleteAllItems()
        for i in Steps:
            self.list.Append([(i[j],int(i[j]))[int(j==0)]
               for j in range(len(i))])
        self.count=len(Steps)

    def SetBoxes(self,array):
        [self.boxes[i].SetValue(array[i]) for i in range(len
           (self.boxes))]

    def SetData(self):

        try:
```

```
            wx.GetApp().SetInput(self.GetSteps(),self.
                GetValues())
            return True

        except:
            wx.MessageBox("Please insert correct input
                values", "Incorrect Values",wx.OK)
            return False

    def GetValues(self):
        temp={ i.GetLabel().split()[0]:i.GetValue() for i in
            self.boxes}
        temp["output_folder"]=self.output_folder
        return temp

    def GetSteps(self):
        return [[float(self.list.GetItem(itemId=row, col=i).
            GetText()) for i in range( self.list.ColumnCount)
            ] for row in range(self.count)]

    def WriteXML(self):
        dlg=wx.FileDialog(self, "Save", self.dirname, "", "
            *.xml*", wx.SAVE)
        if dlg.ShowModal() == wx.ID_OK:
            self.dirname=dlg.GetDirectory()


            Data = ET.Element("Data")
            General = ET.SubElement(Data,"General")
            for i in self.boxes:
                info = ET.SubElement(General,"Value")
                info.set(i.GetLabel().split()[0],str(i.
                    GetValue()))


            for i in self.GetSteps() :
                    Step=ET.SubElement(Data,"Steps")
                    Step.set("Step",str(i).replace(",","").
                        strip("[]"))

            tree = ET.ElementTree(Data)
            tree.write(dlg.GetDirectory()+"/"+dlg.
                GetFilename())
```

```
        dlg.Destroy()

    def LoadXML(self):
        dlg=wx.FileDialog(self, "Load", self.dirname, "", "
            *.xml*", wx.OPEN)
        if dlg.ShowModal() == wx.ID_OK:
            doc=ET.parse(dlg.GetDirectory()+"/"+dlg.
                GetFilename())
            self.config.DeleteGroup("Path")
            self.config.Write("Path",dlg.GetDirectory())

            self.SetSteps([[float(j) for j in i.attrib["Step
                "].split()]for i in doc.findall("Steps")])

            self.SetBoxes([i.attrib.values()[0].strip("[]")
                for i in doc.findall(".//Value")])

        dlg.Destroy()
```

Appendix D.B Run Script

```
#!/usr/bin/env python

import APSUI as GUI
from dolfin import *
import Calculations as CAL

def main():

    app= GUI.CustomApp()
    GUI.User_interface(None)
    app.MainLoop()

    if app.steps!=None:

        CAL.Calculations(app.settings,app.steps)

    else:
        pass

if __name__ == '__main__':

    main()
```

Appendix D.C Calculation

```
# -*- coding: utf-8 -*-


from dolfin import *
from SubDomains import *
from PlasticModel import *
from ReturnMapping import *
import DataHandler as Handler
import utils as utils



def Calculations(Settings, Steps):
    set_log_level(20)
    debug=bool(Settings["Debug"]=="True")
    directory=Settings["output_folder"]

    update=Settings["Update"]
    """
    1. Assign General parameters
    """
    """
    scaling:
        young               1.0e3
        Poisson             1
        SMYS                1  MPa auto scaled
        Work-hadrening      1.0e3
    """

    outer_radi=Settings["Outer"]/2.0
    inner_radi=outer_radi-Settings["Wall"]
    length=Settings["Length"]
    base=Settings["Base"]
    ratio= outer_radi/inner_radi
    Area= np.pi*(outer_radi**2 - inner_radi**2)


    nu =Settings["Poisson"]



    """
```

```
Setting alpha
"""
if Settings["End"]=="Closed":
    alpha=1-2*nu
elif Settings["End"]=="Open":
    alpha=-2*nu
r_dep = (1+nu)*((1 -2*nu)*inner_radi + outer_radi**2/
    inner_radi) - alpha*nu*inner_radi

plastic_model=PlasticModel(1.0e3*Settings["Young"],
    Settings["Poisson"],Settings["SMYS"],1.0e3*Settings["
    Work-Hardening"])

"""
2. Create Mesh
"""


faces=int(Settings["Faces"])
resolution=int(Settings["Resolution"])

Inner_geo = Cylinder(Point(0,0,length- base),Point(0,0,
    base),inner_radi,faces)
Outer_geo = Cylinder(Point(0,0,length -base),Point(0,0,
    base),outer_radi,faces)

mesh3d= Mesh(Outer_geo-Inner_geo, resolution)

if debug==True:
    print "Resolution   :", resolution
    print "Num. cells   :", mesh3d.num_cells()
    print "Num_vertices:", mesh3d.num_vertices()
    plot(mesh3d,interactive=True)


"""
3.Define boundaries and subdomains
"""
boundaries=FacetFunction("size_t",mesh3d)
boundaries.set_all(0)
tol=Settings["Wall"]/2
subdomains=[Top(tol,height=length-base),Bottom(tol,base=
    base),Outer(tol ,outer_radi=outer_radi),Inner(tol ,
    inner_radi=inner_radi)]
```

```
"""
4. Marking boundaries with subdomains
"""
while any(boundaries.array()==0) :
    subdomains[3].mark(boundaries ,4)
    subdomains[2].mark(boundaries ,3)
    tol+=0.01
    subdomains[2]=Outer(tol ,outer_radi=outer_radi)
    subdomains[3]=Inner(tol ,inner_radi=inner_radi)

tol=0.01
subdomains[0]=Top(tol,height=length-base)
subdomains[1]=Bottom(tol,base=base)
subdomains[0].mark(boundaries,1)
subdomains[1].mark(boundaries,2)

"""
5. Specify the boundary integration in the weak
   formulation
"""
ds = Measure("ds")[boundaries]

if debug==True:
    plot(boundaries,interactive=True)
    print 2


"""
6. Create the FunctionSpaces
"""
quad= int(Settings["Quadrature"])
V=VectorFunctionSpace(mesh3d,"Lagrange",quad)
Vt=VectorFunctionSpace(mesh3d,"Lagrange",1,36)
Vs=VectorFunctionSpace(mesh3d,"Lagrange",1,6)
W= FunctionSpace(mesh3d, "Lagrange", 1)
U= TensorFunctionSpace(mesh3d ,"Lagrange",1)
Vx,Vy,Vz = V.split()



"""
7. Store some auxillary values for easy access
"""
```

```
num_points = Vs.dim()/Vs.element().value_dimension(0)
num_nodes= V.dim()/V.element().value_dimension(0)
print V.element().value_dimension(0)
print Vs.element().value_dimension(0)
"""
8. Intalize Functions
"""
Cons_tangent=Function(Vt)
Cons_tangent.vector().set_local(np.array(plastic_model.
    return_list()*num_points))

Stress=Function(Vs)
Stress.vector()[:]=0.0

Strain_p=Function(Vs)
Strain_p.vector()[:]=0.0

Residual=Function(Vs)
Residual.vector()[:]=0

APS = Function(W)
APS.vector()[:]=0.0

Strain=Function(Vs)
Strain.vector()[:]=0.0



"""
9. Declear surfaces normals
"""
normal=FacetNormal(mesh3d)


"""
10. The weak formulation
"""
u=TrialFunction(V)
v=TestFunction(V)

a=inner(utils.eps(v), dot(utils.tangent(Cons_tangent),
    utils.eps(u)) )*dx
```

```
L = inner(v, Constant((0.0,0.0,0.0)))*dx


"""
11. Pre-assemble of projection matrix, for faster
    projections
"""

M = assemble(inner( TestFunction(Vs), utils.eps(
    TrialFunction(V)))*dx)
M.compress()
P= assemble(inner(TrialFunction(Vs), TestFunction(Vs))*
    dx)
ones = Function(Vs)
ones.vector()[:] = 1
P_diag = P * ones.vector()
P_diag.set_local(1.0/P_diag.array())

"""
12. Iniate Return-mapping and DataHandler
"""
RM=ReturnMapping(10000)
data=Handler.Data(num_points, directory)

"""
Saving initial parameters
"""
data.StoreGenereal(Settings, num_points, Area, num_nodes)
data.StoreSteps(Steps)
data.area_approx.append(sum(assemble(inner(TestFunction(
    V), Constant((1,0,0)))*ds(1))))
data.StoreApproxArea()
"""
13. Strain_increments
"""
strain_increment=Settings["Increment"]

"""
14. Problem function
"""
du = Function(V)
u  = Function(V)
dp=Function(V)
p=Function(V)
```

```
"""
15. Expression to be used with Dirichlet conditions
"""

Curvature= Expression("-x[2]*(CurvY*x[0]+CurvX*x[1])",
    CurvY=0.0,  CurvX=0.0)

Tension = Expression("x[2]*value",value=0.0 )

"""
16. Initialize previous state variables
"""

prv_curv_x=0.0
prv_curv_y=0.0
prv_tension=0.0
prv_pressure=0.0
for step in Steps:
    """
        All float :
        Step number          = i[0]
        Curvature y          = i[1]
        Curvature x          = i[2]
        Internal Pressure    = i[3]
        Axial Tension        = i[4]
    """
    num_step=int(step[0])


    data.New_step()


    print (step[4]-prv_tension)
    if (step[4]-prv_tension)!=0:

        """
        Initilize based on loading or unloading
        """
        if (step[4]-prv_tension) >0:
            Tension.value=strain_increment
            statement=utils.Loading
        elif (step[4]-prv_tension) < 0:
            Tension.value=-strain_increment
```

```
    statement=utils.Unloading


"""
Dirichlet conditions on boundary
"""
BCS=[]
BCS.append(DirichletBC(Vz,Tension,boundaries,2))
BCS.append(DirichletBC(Vz,Tension,boundaries,1))

force=sum(assemble(inner(TestFunction(V),dot(
   normal,utils.sigma_3(Stress)))*ds(1)))*1.5

while statement(1.00e3*step[4],force)   :
    """
    new iteration
    """
    data.New_iteration()
    """
    Assembling the problem
    """
    A=assemble(a)
    b=assemble(L)

    """
    Apply Dirichlet conditions
    """
    for i in BCS:
        i.apply(A,b)
    """
    Solving problem
    """
    solve(A,du.vector(),b,"richardson","sor")

    u.vector()[:] += du.vector()
    grad_eps=M*u.vector()
    Strain.vector()[:] = grad_eps* P_diag

    if update=="Increment":
        mesh3d.move(du)
    for point in range(num_points):
        dofs_s=[point*6 + j for j in range(6)]
        dofs_t=[point*36+ j for j in range(36)]
```

```
        epsilon          =np.matrix(Strain.vector
            ()[dofs_s]).T
        epsilon_p        =np.matrix(Strain_p.
            vector()[dofs_s]).T
        trial_sigma      =plastic_model.
            elastic_tangent*(epsilon-epsilon_p)

        tangent,data.e_p[point]=RM.automatic(
            plastic_model,trial_sigma,epsilon_p,
            data.e_p[point])

        data.MaxValues(trial_sigma,epsilon,
            epsilon_p)

        """
        Updating functions
        """

        Cons_tangent.vector()[dofs_t]=np.asarray
            (tangent).reshape(-1)
        Stress.vector()[dofs_s]=np.asarray(
            trial_sigma).reshape(-1)
        Strain_p.vector()[dofs_s]=np.asarray(
            epsilon_p).reshape(-1)


data.AddValues()

print max(data.e_p)

f_x =sum(assemble(inner(TestFunction(V),dot(
    normal,utils.sigma_1(Stress)))*ds(1)))
    *1.5
f_y= sum(assemble(inner(TestFunction(V),dot(
    normal,utils.sigma_2(Stress)))*ds(1)))
    *1.5
f_z= sum(assemble(inner(TestFunction(V),dot(
    normal,utils.sigma_3(Stress)))*ds(1)))
    *1.5

f_tot = (f_x**2+f_y**2+f_z**2)**0.5
```

```
            print max(data.e_p),f_tot,f_z
            force=f_z


        prv_tension=1.0e3*step[4]




"""


Setting curvature
"""

total_curv_y=step[1]-prv_curv_y
total_curv_x=step[2]-prv_curv_x
prv_curv_x=step[2]
prv_curv_y=step[1]

if not (total_curv_x==0 and total_curv_y==0):

    """
    Scaling and setting increment
    """
    max_iteration=(1.0e-3)*(total_curv_x**2 +
        total_curv_y**2)**0.5/strain_increment
    increment_curv_x = total_curv_x/max_iteration
    increment_curv_y = total_curv_y/max_iteration
    Curvature.CurvY=(1.0e-3)*increment_curv_y
    Curvature.CurvX=(1.0e-3)*increment_curv_x

    """
    Dirichlet on top and bottom markes 1 and 2, in
        subdomain 0 and 1
    """
    BCS = []

    BCS.append(DirichletBC(Vz,Curvature,boundaries
        ,1))
    BCS.append(DirichletBC(Vz,Curvature,boundaries
        ,2))
```

```
iteration=0

while iteration < max_iteration :
    """
    new iteration
    """
    data.New_iteration()
    print iteration
    iteration+=1

    """
    Assembling the problem
    """
    A=assemble(a)
    b=assemble(L)

    """
    Apply Dirichlet conditions
    """
    for bcs in BCS:
        bcs.apply(A,b)


    """
    Solving problem
    """
    solve(A,du.vector(),b,"richardson","sor")

    u.vector()[:] += du.vector()


    grad_eps=M*u.vector()
    Strain.vector()[:] = grad_eps* P_diag
    if update=="Increment":
        mesh3d.move(du)

    for point in range(num_points):

        dofs_s=[point*6 + j for j in range(6)]
        dofs_t=[point*36+ j for j in range(36)]
```

```
                epsilon          =np.matrix(Strain.vector
                    ()[dofs_s]).T
                epsilon_p        =np.matrix(Strain_p.
                    vector()[dofs_s]).T
                trial_sigma      =plastic_model.
                    elastic_tangent*(epsilon-epsilon_p)

                tangent, data.e_p[point]=RM.automatic(
                    plastic_model,trial_sigma,epsilon_p,
                    data.e_p[point])
                """
                Store max values
                """
                data.MaxValues(trial_sigma,epsilon,
                    epsilon_p)


                """
                Updating functions
                """
                Cons_tangent.vector()[dofs_t]=np.asarray
                    (tangent).reshape(-1)
                Stress.vector()[dofs_s]=np.asarray(
                    trial_sigma).reshape(-1)
                Strain_p.vector()[dofs_s]=np.asarray(
                    epsilon_p).reshape(-1)

            data.AddValues()

            print "Max APS :",max(data.e_p)




    if (step[3]-prv_pressure)!= 0.00:


        """
        Setting Boundary conditions,
        """
```

```
dp.vector()[:]=assemble(inner(TestFunction(V),-
    normal)*ds(4))
dp=utils.MakeUnitvector(dp)


BCS=[]


if    (step[3]-prv_pressure) > 0.00:
      Tension.value=alpha*strain_increment
      BCS.append(DirichletBC(V,Constant(r_dep*
          strain_increment)*dp,boundaries,4))
      statement=utils.Loading
elif    (step[3]-prv_pressure) < 0.00:
      Tension.value=alpha*(-strain_increment)
      BCS.append(DirichletBC(V,Constant(r_dep*(-
          strain_increment))*dp,boundaries,4))
      statement=utils.Unloading


BCS.append(DirichletBC(Vz,Tension,boundaries,1))
BCS.append(DirichletBC(Vz,Tension,boundaries,2))
BCS.append(DirichletBC(Vz,Tension,boundaries,4))


p.vector()[:]=assemble(inner(TestFunction(V),dot
    (normal,utils.sigma_axis(Stress)))*ds(4))
Area_4=sum(assemble(inner(TestFunction(V),
    Constant((1,0,0)))*ds(4)))
pressure  = utils.SumVector(p)/Area_4


while statement(step[3],pressure)  :
      """
      new iteration
      """
      data.New_iteration()
      """
      Assembling the problem
      """


      A=assemble(a)
      b=assemble(L)
```

```
    """
    Apply Dirichlet conditions
    """
    for bcs in BCS:
        bcs.apply(A,b)
    """
    Solving problem
    """
    solve(A,du.vector(),b,"richardson","sor")

    u.vector()[:] += du.vector()
    grad_eps=M*u.vector()
    Strain.vector()[:] = grad_eps* P_diag

    if update=="Increment":
        mesh3d.move(du)
    for point in range(num_points):

        dofs_s=[point*6 + j for j in range(6)]
        dofs_t=[point*36+ j for j in range(36)]


        epsilon      = np.matrix(Strain.vector()[
            dofs_s]).T
        epsilon_p    = np.matrix(Strain_p.vector
            ()[dofs_s]).T
        trial_sigma = plastic_model.
            elastic_tangent*(epsilon-epsilon_p)

        tangent, data.e_p[point]=RM.automatic(
            plastic_model,trial_sigma,epsilon_p,
            data.e_p[point])
        """
        Store max values
        """
        data.MaxValues(trial_sigma,epsilon,
            epsilon_p)

        """
        Updating functions
        """
        Cons_tangent.vector()[dofs_t]=np.asarray
            (tangent).reshape(-1)
```

```
                    Stress.vector()[dofs_s]=np.asarray(
                        trial_sigma).reshape(-1)
                    Strain_p.vector()[dofs_s]=np.asarray(
                        epsilon_p).reshape(-1)

                p.vector()[:]=assemble(inner(TestFunction(V)
                    ,dot(normal,utils.sigma_axis(Stress)))*ds
                    (4))
                Area_4=sum(assemble(inner(TestFunction(V),
                    Constant((1,0,0)))*ds(4)))
                pressure  = utils.SumVector(p)/Area_4

                data.AddValues()

                print "Pressure :",pressure
                print "Max APS :",max(data.e_p)
                print "Max Stress:",data.max_stress

            prv_pressure=step[3]

    data.AddAPS()
    APS.vector()[:]=np.asarray(data.e_p).reshape(-1)

    """
    Saving output
    """
    data.Plot(num_step)
    data.StoreValues(num_step)

    file = File(directory+"/Strain"+str(num_step)+".pvd"
        )
    file << project(sqrt((2.0/3.0)*dot(Strain,Strain)),W
        )
    file = File(directory+"/Stress"+str(num_step)+".pvd"
        )
    file << project(sqrt((3.0/2.0)*dot(Stress,Stress)),W
        )
    file = File(directory+"/APS"+str(num_step)+".pvd")
    file << APS
    file = File(directory+"/StrainTensor"+str(num_step)+
        ".pvd")
    file << project(utils.sigma(Strain),U)
    file = File(directory+"/StressTensor"+str(num_step)+
        ".pvd")
```

```
        file << project(utils.sigma(Stress),U)
        file = File(directory+"/radial_Stress"+str(num_step)
           +".pvd")
        file << p

    data.StoreAPS()
    return True
```

Appendix D.D Data Handling

```
import numpy as np
import utils as utils
import matplotlib.pyplot as pyplot

class Data(object):
    def __init__(self, num_cells,directory=""):
        self.directory=directory
        """
        Data handling
        """
        self.e_p =np.array([0.00 for i in range(num_cells)])
        """
        Re-init every iteration
        """
        self.New_iteration()

        """
        Re-init every step
        """
        self.New_step()
        self.APS_values=[]
        self.area_approx=[]

    def New_iteration(self):
        self.max_stress  =0.0
        self.max_strain  =0.0
        self.max_strain_p=0.0

        self.max_sigma=np.matrix([[0.0] for i in range(6)])
        self.max_epsilon=np.matrix([[0.0] for i in range(6)
           ])
        self.max_epsilon_p=np.matrix([[0.0] for i in range
           (6)])
```

```
        pass

    def New_step(self):
        self.max_array_strain=[]
        self.max_array_stress=[]
        self.max_array_strain_p=[]
        self.max_array_sigma=[]
        self.max_array_epsilon=[]
        self.max_array_epsilon_p=[]

        pass



    def MaxValues(self, stress , strain ,strain_p):
        temp=utils.equivalentStress(stress)
        if temp > self.max_stress:
            self.max_stress=temp
            self.max_sigma=stress
            self.max_strain=utils.equivalentStrain(strain)
            self.max_epsilon=strain
            self.max_strain_p=utils.equivalentStrain(
               strain_p)
            self.max_epsilon_p=strain_p

        pass

    def AddAPS(self):
        self.APS_values.append(max(self.e_p))

    def StoreAPS(self):
        np.savetxt(self.directory+"/APS.txt",self.APS_values
            )

    def AddValues(self):
        self.max_array_strain.append(self.max_strain)
        self.max_array_strain_p.append(self.max_strain_p)
        self.max_array_stress.append(self.max_stress)
        self.max_array_sigma.append(self.max_sigma)
        self.max_array_epsilon.append(self.max_epsilon)
        self.max_array_epsilon_p.append(self.max_epsilon_p)
        pass

    def Plot(self,num_step):
```

```
        pyplot.plot(self.max_array_strain,self.
            max_array_stress)
        pyplot.savefig(self.directory+"/StressStrain"+str(
            num_step)+".png")
        pyplot.clf()
        pyplot.plot(self.max_array_strain,self.
            max_array_strain_p)
        pyplot.savefig(self.directory+"/StrainpStrain"+str(
            num_step)+".png")
        pyplot.clf()
        pass

    def StoreValues(self,num_step):
        np.savetxt(self.directory+"/MaxStress"+str(num_step)
            +".txt",self.max_array_stress)
        np.savetxt(self.directory+"/MaxplasticStrain"+str(
            num_step)+".txt",self.max_array_strain_p)
        np.savetxt(self.directory+"/MaxStrain"+str(num_step)
            +".txt",self.max_array_strain)
        np.savetxt(self.directory+"/MaxStressTensor"+str(
            num_step)+".txt",self.max_array_sigma)
        np.savetxt(self.directory+"/MaxplasticStrainTensor"+
            str(num_step)+".txt",self.max_array_epsilon_p)
        np.savetxt(self.directory+"/MaxStrainTensor"+str(
            num_step)+".txt",self.max_array_epsilon)
        pass

    def StoreApproxArea(self):
        np.savetxt(self.directory+"/ApproxArea"+".txt",self.
            area_approx)
        pass

    def StoreGenereal(self,Settings,num_points,Area,Nodes):
        f=open(self.directory+"/General.txt ","w')
        for key, value in Settings.items():
            f.writelines(str(key)+":"+str(value)+"\n")
        f.writelines("Num points :" +str(num_points)+"\n")
        f.writelines("Area:"+str(Area)+"\n")
        f.writelines("Num nodes :" +str(Nodes)+"\n")
        f.close()
        pass
    def StoreSteps(self,steps):
        f=open(self.directory+"/Steps.txt ","w')
        for step in steps:
```

```
            f.writelines(str(step)+"\n")
        f.close()
        pass
```

Appendix D.E Collection of Utility Functions

```
from dolfin import*
import numpy as np
def deviatoric(tensor_array):
    temp=tensor_array
    trace= tensor_array[0]+tensor_array[1]+tensor_array[2]
    temp[0]=tensor_array[0]-trace/3.0
    temp[1]=tensor_array[1]-trace/3.0
    temp[2]=tensor_array[2]-trace/3.0
    return temp

def equivalentStress(tensor_array):
    T=deviatoric(tensor_array)
    return 1.2247448713915892*(float(T[0])**2 + float(T[1])
       **2 + float(T[2])**2+2*float(T[3])**2+2*float(T[4])
       **2+2*float(T[5])**2)**0.5

def equivalentStrain(tensor_array):
    T=deviatoric(tensor_array)
    return 0.816496580927726*(float(T[0])**2 + float(T[1])
       **2 + float(T[2])**2+float(T[3])**2+float(T[4])**2+
       float(T[5])**2)**0.5

def eps(u):
        return as_vector([u[i].dx(i) for i in range(3)] + [u
           [i].dx(j) + u[j].dx(i) for i, j in [(0, 1), (0,
           2), (1, 2)]])

def sigma(s):
    return as_matrix([[s[0], s[3], s[4]], [s[3], s[1], s
       [5]], [s[4], s[5], s[2]]])

def tangent(t):
  return as_matrix([[t[i*6 + j] for j in range(6)] for i in
     range(6)])


def sigma_axis(s):
```

```
    return as_matrix([[s[0], 0.0, 0.0], [0.0, s[1], 0.0],
       [0.0, 0.0, s[2]]])

def sigma_1(s):
    return as_matrix([[s[0], 0.0, 0.0], [0.0, 0.0, 0.0],
       [0.0, 0.0, 0.0]])
def sigma_2(s):
    return as_matrix([[0.0, 0.0, 0.0], [0.0, s[1], 0.0],
       [0.0, 0.0, 0.0]])
def sigma_3(s):
    return as_matrix([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0],
       [0.0, 0.0, s[2]]])

def MakeUnitvector(Function):
    V = Function.function_space()
    num_points = V.dim()/V.element().value_dimension(0)
    for point in range(num_points):
        dofs_u =[point*3 + j for j in range(3)]
        v=Function.vector()[dofs_u]
        if norm(v)!=0:
            Function.vector()[dofs_u] /= norm(v)

    return Function


def SumVector(Function):
    V = Function.function_space()
    num_points = V.dim()/V.element().value_dimension(0)

    SumVector=0.0
    for point in range(num_points):
        dofs_u =[point*3 + j for j in range(3)]
        v=Function.vector()[dofs_u]
        SumVector+=norm(v)

    return SumVector



def DeviatoricFunction(Function):
    V = Function.function_space()
    num_points = V.dim()/V.element().value_dimension(0)

    for point in range(num_points):
```

```
            dofs_u =[point*3 + j for j in range(3)]
            v=Function.vector()[dofs_u]
            temp=(v[0]+v[1]+v[2])/3.0
            Function.vector()[dofs_u[0]]-=temp
            Function.vector()[dofs_u[1]]-=temp
            Function.vector()[dofs_u[2]]-=temp

    return Function

def Loading(step_input,variable):
    return (variable -step_input) < 0


def Unloading(step_input,variable):
    return  (step_input-variable) < 0
```

Appendix D.F Return Mapping

```
import numpy as np
from PlasticModel import *
from scipy.optimize import *

def function(delta_lambda , plastic_model,trial_stress,e_p)
    :
    D       =plastic_model.elastic_tangent
    P       =plastic_model.P
    H       =plastic_model.H
    sigma_y=plastic_model.yield_stress
    Q= 1.0*np.identity(6) + float(delta_lambda)*D*P
    Qinv = Q.I
    sigma= Qinv*trial_stress
    phi    =float(plastic_model.dg(sigma).T*plastic_model.dg(
        sigma))
    kappa =  e_p + float(delta_lambda)*(2.0/3.0*phi)**0.5
    return float( 0.5*phi - (sigma_y+kappa*H )**2/3.0 )


class ReturnMapping():
    def __init__(self, maxit=1000):
        self.maxit=maxit

    def automatic(self,plastic_model ,trial_stress,
        plastic_strain,e_p):
        delta_lambda=0.0
```

```
De= plastic_model.elastic_tangent
P=plastic_model.P

if plastic_model.f(trial_stress,e_p)> 1.0e-12 :

    delta_lambda=float(brentq(function,0.0,1.0, args
        =(plastic_model,trial_stress,e_p, ),maxiter=
        self.maxit, full_output=False, disp=True ))

    Q= 1.0*np.identity(6) + float(delta_lambda)*De*P
    Qinv = Q.I

    sigma_current=Qinv*trial_stress



    trial_stress= sigma_current

    df_dsigma=plastic_model.df(trial_stress)
    dg_dsigma=plastic_model.dg(trial_stress)

    hardening=plastic_model.hardening(trial_stress,
        e_p)

    Rn = (De*df_dsigma).T
    D = De - De*(dg_dsigma*Rn)/(Rn*dg_dsigma +
        hardening)

    e_p=plastic_model.APS(trial_stress,delta_lambda,
        e_p)

    plastic_strain += delta_lambda*dg_dsigma

else:
    D=De


return D,e_p
```

Appendix D.G Plastic Model

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Mon Apr  7 20:02:24 2014

@author: lars
"""

import numpy as np

class PlasticModel(object):#P i.e  ddg_ddsigma
    def __init__(self,E,nu,yield_stress,work_hardening):
        """
            Must parameters
        """
        mu=E/(2*(1 + nu))
        lmbda=E*nu/((1 + nu)*(1 - 2*nu))

        self.elastic_tangent= np.matrix([[ (lmbda+mu*int(i==
            j))*int(j<3 and i<3)+mu*int(i==j)    for i in
            range(6)] for j in range(6) ])
        self.yield_stress=yield_stress
        self.H=work_hardening

        """
            Optinal parameters
        """
        self.P=np.matrix([[ float(i==j) -(1/3.0)*int(j<3 and
             i<3) + float(i==j and j>2 and i>2) for i in
            range(6)] for j in range(6)])


    def f (self,sigma,e_p):
        return 0.5*sigma.T*self.P*sigma - (( self.
            yield_stress + self.H*e_p)**2)/3.0

    def df(self,sigma):
        return self.P*sigma

    def dg(self,sigma):
        return self.P*sigma

    def ddg(self,sigma):
        return self.P

    def hardening(self,stress,e_p):
```

```
        return 2.0/3.0*(self.yield_stress+self.H*e_p)*self.H
            *np.sqrt(2.0/3.0*self.dg(stress).T*self.dg(stress
            ))

    def APS(self, sigma,delta_lambda,e_p):
        return e_p + delta_lambda*np.sqrt(2.0/3.0*self.dg(
            sigma).T*self.dg(sigma))


    def return_list(self):
        return list(np.asarray(self.elastic_tangent).reshape
            (-1))
```

Appendix D.H Subdomains

```
from dolfin import *
import math

class Top(SubDomain):
     def __init__(self,tol,**kwargs):
         SubDomain.__init__(self)
         self.tol=tol
         self.height=kwargs.get("height")
     def inside(self,x,on_boundary):
                 return near(self.height,x[2],self.tol)

class Bottom(SubDomain):
     def __init__(self,tol,**kwargs):
         SubDomain.__init__(self)
         self.tol=tol
         self.base=kwargs.get("base")
     def inside(self,x,on_boundary):
                 return near(x[2],self.base,self.tol)


class Outer(SubDomain):
     def __init__(self,tol,**kwargs):
         SubDomain.__init__(self)
         self.tol=tol
         self.outer_radi=kwargs.get("outer_radi")
     def inside(self,x,on_boundary):
                 r = math.sqrt( x[0]*x[0]+x[1]*x[1])
                 return near(r,self.outer_radi,self.tol)
```

```
class Inner(SubDomain):
      def __init__(self,tol,**kwargs):
          SubDomain.__init__(self)
          self.tol=tol
          self.inner_radi=kwargs.get("inner_radi")
      def inside(self,x,on_boundary):
                  r = math.sqrt( x[0]*x[0]+x[1]*x[1])
                  return near(r,self.inner_radi,self.tol)
```