Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Geoscience and Petroleum

Eirik Kaasbøll Andresen

Development of an automation method to conduct sensitivity analysis on weakness zones using Finite Element Modelling

with focus on total deformations

Master's thesis in Tekniske Geofag
Supervisor: Alexandre Lavrov
Co-supervisor: Are Håvard Høien
July 2022

NTNU
Norwegian University of
Science and Technology

Eirik Kaasbøll Andresen

Development of an automation method to conduct sensitivity analysis on weakness zones using Finite Element Modelling

with focus on total deformations

Master’s thesis in Tekniske Geofag
Supervisor: Alexandre Lavrov
Co-supervisor: Are Håvard Høien
July 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Geoscience and Petroleum

Abstract

In hard rock areas, such as the Norwegian geological landscape, are weakness zones one of the most prominent reasons for instability related problems met in the tunnelling industry. It is therefore imperative to gain reliable information regarding such zones early enough, to make the impact on the economy and the excavation time of the project as low as possible. Over the years, many contributions have been made to include the effect weakness zone has on the general stability of the tunnels. Most of these contributions are empirically based, assessing the stability based on field observations. However, due to being empirically based, these systems are not optimized. It has been proven that the usage of security measures often is unnecessary high or the wrong security method is chosen in Norwegian road tunnelling projects in areas affected by weakness zones.

In this thesis, a sensitivity study on weakness zones' impact on the overall stability of a tunnel has been done using computer based Finite Element Modelling-tools. The FEM method was mainly chosen due to its abilities in modelling the complex geometries and material behavior. The 2-dimensional FEM program RS2 distributed by RockScienceTM was chosen in this thesis due to its popularity in the industry and being well documented.

However, RS2 is not optimal when conducting a sensitivity study, since it does not support model creation using scripting. This would potentially limit the number of models to be created. To overcome this problem, it was developed a script to automize the model construction, calculation and data gathering process of RS2. The development of the script made it possible to run in the total of 3696 models in matter of days, which would not be possible without it.

The focus of the sensitivity study was on geometrical features regarding weakness zones resembling faults, which is one of the most usual type of weakness zone faced in hard rock tunnelling as there is Norway. The included parameters were: (a) the zone thickness, (b) the overburden, (c) the zone angle and (d) the shortest distance between tunnel center and the zone.

An important finding was that the zone angle Θ did not have any significance on the tunnel stability for any zone thicknesses T, overburdens H, and normalized shortest distances between tunnel center and weakness zones Γ_{norm}. Another important insight was that maximum strain $\epsilon_{max} = \epsilon_{tot}(\Gamma_{norm} = 1.1)\forall\mathrm{T}$. A third curiosity was that the weakness zone was out of the influence zone of the tunnel when $\Gamma_{\mathrm{norm}} >\approx 2$. The exponential increase of ϵ_{tot} with increasing H is severely primed when T also is increased.

It is believed that the method developed in the thesis could be used to define a dimensionless number, analogous to the Reynolds number used in Fluid mechanics. This number can be used to predict when the weakness zone no longer affects the stability of the tunnel. To do this the tunnel size must be varied in addition with a more thoroughly defined sensitivity study in where the material behavior also must be varied

Sammendrag

I områder med harde bergarter, tilsvarende det Norske geologiske landskapet, er svakhetssoner en av de viktigste årsakene til stabilitetsrelaterte problemstillinger møtt i tunnelindustrien. Det er derfor nødvendig å få tilgang på representative data tilknyttet slike soner tidlig nok, til å kunne redusere dets negative konsekvenser angående økonomiske forhold og eventuell stans under driving. Gjennom årene har det vært bange bidrag for å inkludere svakhetssonens negative innvirkning på tunnelstabiliteten. Fleste parten av disse er empiriske konstruksjoner, som beskriver det totale stabilitetsbildet basert på feltobservasjoner. Men, det faktum at de er tuftet på empiri, medfører at de ikke er optimalisert. Det har blitt vist at overforbruk av sikringstiltak og/eller feil valg av sikringstiltak forkommer ofte i tunneldriving i norske tunneler når svakhetssoner er innenfor influensavstanden visavis tunnelen.

Denne masteroppgaven tar for seg en sensitivitetsstudie angående svakhetssoners effekt på en tunnels totalstabilitet ved å benytte programvare basert på endelig-element metoden (EEM). Hovedgrunnen til å benytte EEM var på grunn av dets evne til å modellere komplekse geometrier og materialoppførsler. Den 2-dimensjonale EEM-baserte programvaren RS2, distribuert av RockScienceTM, var valgt grunnet dets popularitet i industrien, og grunnet at dets bruk er vel dokumentert.

Men, RS2 er ikke optimal når det skal utføres en sensitivitetsanalyse, siden det ikke er mulig å definere modellene ved å anvende scriptbaserte løsninger direkte, som igjen ville kunne begrense antallet modeller som kunne blitt laget. For å overkomme denne hindringen ble det utviklet et script for å automatisere modelleringsprosessen prosessen til Rs2 og den påfølgende dataprosesseringen. Det utviklede scriptet muliggjorde å konstruere totalt 3696 modeller i løpet av få dager, noe som ikke ville vært mulig uten det.

Sensitivitetsstudien ble fokusert til å i hovedsak angå svakhetssonens geometriske parametere, hvor det ble kun sett på forkastningssoner. Dette er en av de hyppigst møtte svakhetssoner i hardt berg tilsvarende norske forhold. Følgende parametere ble undersøkt: (a) mektigheten til sonen, (b) overdekningen, (c) sonevinkelen, og (d) korteste avstand fra tunnelsenter til sonen.

En viktig innsikt var at sonevinkelen Θ ikke hadde en signifikant innvirkning på tunnel stabiliteten for alle sonetykkelser T, overdekninger H og den normaliserte korteste avstand mellom tunnelsenter og svakhetssone Γ_{norm}. En annen kuriositet var at den maksimale tøyning $\epsilon_{max} = \epsilon_{tot}(\Gamma_{norm} = 1.1)\forall \mathrm{T}, \mathrm{H}$. En tredje innsikt var at svakhetssonen forekom utenfor influensavstanden til tunnelen når $\Gamma_{norm} \approx 2$. Den eksponentielle vekst av ϵ_{tot} med økende H skjyter ytterligere fart når T samtidig blir økt.

Det er grunner til å tro at det vil være mulig å kunne utvikle et enhetsløst tall analogt til Reynoldstallet i fluidmekanikken. Dette tallet ville kunne uttrykke punktet hvor sonen forlater influensavstanden relativt tunnelen. For å gjøre dette må en mer omfattende studie bli utført, der hvor tunnelradiusen må bli endret, og hvor elastitets- og styrke parametere for sone og omkringliggende berg blir inkludert.

Preface

This thesis has been performed as the final work of the master's degree program Geotechnology at the Department of Geoscience and Petroleum, Norwegian University of Science and Technology.

The thesis has been done in cooperation with Norwegian Public Roads Administration (NPRA).

First and foremost, I want to thank my supervisors. Professor Alexandre Lavrov, thank you for your help regarding technicalities of modelling and stability calculations. Thank you for your good advice and for always being available and open for discussions. I have learned a great deal of your guidance. Are Håvard Høien, thank you for giving me the opportunity. I am grateful for the internship I was given the summer 2021 and for the opportunity to develop my understanding of how rock mechanics is used in practice in the field of rock engineering, and the implication it has on practical modelling. Thank you for your support and for your endurance, for it have been a few questions over the last months. And thank you for letting me be creative in the solution of this thesis. Without your support, I would not have been able to steer this thesis towards scripting development and automation, which is two interests of mine.

Thank you so much, Jorge, for being available for discussions and for sharing your experiences with me, and for believing in my work. It means a great deal to me. I am looking forward to follow the work of your doctoral thesis in the years to come!

A great thanks to my father, your support and guidance have been more than welcome. You always manage to motivate me somehow; you have to teach me your secret. I want to express my gratitude to my siblings, for always being bored when I was talking about my thesis. It helped me remember the life outside the realm of academic writing and that the thesis is not everything. We have had many interesting and amusing conversations about all and nothing over the last months. And thanks to you mom, for believing in me and for your daily support. I would not have been here I am today without it.

Last, my appreciation goes to my classmates, without your comradeship and daily lunches I do not know if I would ever finish the thesis. Erlend, together we have raised our skills in table tennis to new heights. It has been quite a journey. Thanks to you, John Isak, your sense of humor is a delight. Thank you, Anders and Brage, for your positive energy and your initiative to also have some fun at the department. The idea of one of us bringing "nogo godt attått kaffen" the last weeks of writing was one of those great initiatives. Thank you, Sander and Espen, for good and frequent discussions regarding our thesis work. It has been interesting to follow your journey. Thank you, Kristoffer, Vegard, Are Håvard and Alexandre for giving helpful feedbacks regarding the technicalities of writing.

P.S

I must confess; I greatly underestimated the amount of scripting needed to complete this thesis. In fact, if I knew what I do today regarding the knowledge and skills needed to create the script, this journey would have ended faster than you can say $(Mg,Fe^{2+})_2(Mg,Fe^{2+})_5Si_8O_{22}(OH)_2$. Indeed, it has truly been a journey, which I am glad I stumbled across.

> He is: frog: unworried by the self-consciousness with which the human animal is stuck; it is our blessing and our curse; not only do we know; we know that we know. And we are not often willing to face how little we know.
> ― **Madeleine L'Engle, The Summer of the Great-Grandmother**

Luckily, I was not willing to know!

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations:

FEM	Finite Element Modelling
GHB	Generalized Hoek-Brown failure criterion
MC	Mohr-Coulomb failure criterion
GSI	Geological Strength Index
CAD	Computer Aided Drawing
rm	Rock mass
i	Intact
r	Residual
t	Tunnel

Symbols:

n	Number of models
n_t	Number of points defining the tunnel in the model
R_t	Tunnel radius
E	Young's modulus/Elastic modulus (can be used with the lower-case acronyms except of t)
ν	Poisson's ratio (can be used with the lower-case acronyms except of t)
t_0	peak tensile strength
σ_c	peak strength, uniaxial compression test
t_r	residual tensile strength
ϕ	friction angle
c	cohesion
ψ	dilation angle
a	Hoek-Brown constant
m_b	Hoek-Brown constant
s	Hoek-Brown constant
k	vertical to horizontal stress ratio
H	overburden
Θ	angle of weakness zone
T	thickness of weakness zone
Γ	shortest distance from tunnel center to weakness zone
ε	Young's modulus/Elastic modulus (can be used with the lower-case acronyms except of t)
σ	Poisson's ratio (can be used with the lower-case acronyms except of t)
d_{tot}	Total deformation
ϵ_{tot}	Total strain
ϵ_{max}	Maximum total strain of all models of an experiment
Γ_{norm}	The normalized distance, unitless

1 Introduction

The Norwegian geology generally consists of rock mass of good quality. The reason for this is mainly due to the last ice age, initiated in mid-Mesozoic, where the ice movement eroded away the weaker layers of rock. The end of the ice age marks the transition from Pleistocene to Holocene. In Holocene, the ice melted away. This exposed a geological landscape dominated by folding and fault structures of the Caledonian mountain range mainly defined by bedrock from Precambrian and Caledonian. A map is presented in Figure 1. The Caledonian rocks are mainly metamorphosed rocks of both sedimentary and volcanic origin. The Precambrian rocks generally consists of granites, gneisses, and gabbro of all ranges of metamorphism. The Norwegian geology is therefore dominated by hard rock intersected by weakness zones originated from tectonic activity and in some cases, Mesozoic weathering. The weakness zones vary therefore in character and shows a great variation of quality and extent.

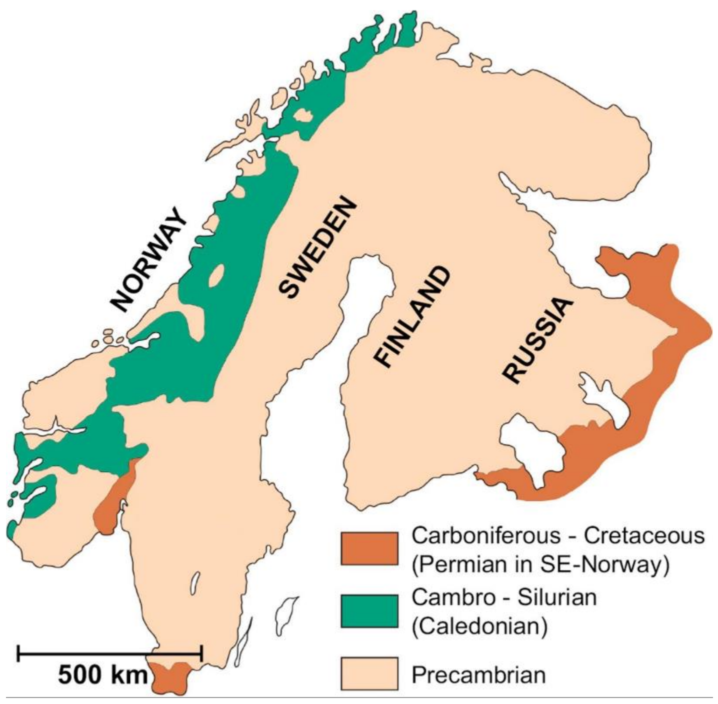

NORWAY
SWEDEN
FINLAND
RUSSIA
Carboniferous - Cretaceous (Permian in SE-Norway)
Cambro - Silurian (Caledonian)
Precambrian
500 km

Figure 1: A map over the main geological components of the Norwegian geological landscape.

Weakness zones is the main reason for stability challenges in Norwegian tunnelling. According to (Nilsen, 2016), typical challenges can include: (a) minor zones and jointed rock in urban areas usually with an overburden between 5 and 100 meters, (b) weakness zones with Mesozoic weathering which may contain swelling clay, (c) wide weakness zones in sub-sea fjord crossings between 100 to 300 m below sea level, and (d) overstressing of solid rock and weakness zones for tunnels under high mountains or along steep valley sides, ranging up to 1000 meters overburden or more.

Rock mass classification systems to approximate rock mass quality has been the industry standard for planning and excavation stages in Norwegian tunnelling for decades. Classification systems main strengths is to make fast and safe assumptions of the quality of the rock mass. The gathered stability assessment is then used to predict necessary security measures. Classification systems is fast in use since the parameters used is mostly based on field observations. However, classification systems are empirically based. Therefore, it does not necessarily capture the physical nature of the rock mass. For instance, it have been proven an overconsumption of ribs of reinforced sprayed concrete (RRS), a heavy method used to secure against weakness zones of low quality, in construction of Norwegian road tunnels over the two last decades (Høien, 2018; Høien & Nilsen, 2019). Of this it is argued that the classification system used do not depict the impact of weakness zone in a satisfactory manner. Overconsumption of security measures and the choice of more heavy security measures leads to unnecessary time- and cost consumption. An optimalization of the stability assessment is also important in an environmental point of view.

The physical nature of how weakness zones alter the stress state and the distribution of deformations is still in some extent a mystery and needs more thorough investigations (Edelbro, 2003). To be able to make better predictions of rock mass quality in future tunnelling projects with hard rock conditions, it is imperative to gather more knowledge on the physical nature of weakness zones' impact on tunnel stability.

An important factor explaining the knowledge gap on how weakness zones affect tunnel stability; is the limited possibilities on data sampling faced in the field of engineering geology. This has paved the road for empirically based equations. It is especially difficult to gather information during tunnelling due to its relatively large scales. The basis for the decision-making is: (a) observations of free surfaces, (b) measurements of physical parameters using indirect methods such as geophysics, and (c) measurements of physical parameters given by direct methods such as the point load test or the UCS-test. Each of these general approaches comes with a variety of sources of error, pros, and cons. A common denominator is if the observations or measurements is representative of the neighboring rock mass. The answer is that it depends on the variability of the rock mass, which again depends on the location of the tunnel. Also, short time limits, defined by the contract of a project, has also driven the method of rock engineering in the direction of inductive decision making.

The general theoretical concepts, borrowed from the field of rock mechanics to describe rock mass behavior, are often based on assumptions simplifying the behavior of rock material to be isotropic and homogenous (Kaasbøll Andresen, 2021). There exists more complex theoretical framework. For example, there is developed failure criteria to describe the material behavior of breccias (Kalender et al., 2014), but they are seldom used in practice. One of the reasons is that the simpler equations such as Mohr-Coulomb and Hoek-Brown has been used for decades in many different applications such as squeezing, and their limitations is therefore readily understood. This makes it easier to manage the sources of error. The field of rock engineering is complicating the matter further by combining theoretical aspects borrowed from rock mechanics with empirical methods, often in where the parameters used in the equations are based on observations alone. To conclude, the sources of failure are many in this field and the methods used is not necessarily depicting the rock mass true physical nature, and is used because they work.

To be able to calculate on stress and strain distributions it was assumed that the behavior of rock mass can be calculated using linear elastic theory (Hooke's law) and that Newton's laws of motion also applies for rock mass. These two assumptions are vastly used in the field and enables the usage of the Finite Element Method (FEM) which have been used in stress analysis of rock mass in the field of rock engineering for decades. One reason for this is that FEM-tools enables analysis of rather complex geometries and variable material behavior. This method assumes that a mathematical continuum can be divided into discrete geometrical entities in which each entity behaves under the same mathematical rule. An example of such a rule is the failure criterion, which describes when and where a given material undergoes failure. Moreover, inner forces in this discretized continuum, is transferred from one entity to another by nodes defined on the rim of these entities. The source of the inner forces is often due to impact of outer forces such as gravity. A key concept of the force transfers is the superposition principle. This states that all forces working on a geometrical entity are independent and can be summed together making a resultant response called a resultant force. Thus, the principal of super position demands that the mathematical continuum to be a vector space, which implies that the continuum behaves linearly. Thus, the rock mass modelled is assumed to be a mathematical continuum and vector space abiding the laws of Newton and Hooke, which again makes it possible to use the Finite Element Method in the analysis of the thesis.

Due to the many sources of error, both in the theoretical abstraction of the rock mass medium and in the observations and measurements of parameters believed to describe rock mass behavior, a quantitatively designed experiment is a rather complex task. One of the reasons is that the necessary assumptions to be made must be applicable for the specimens of the experiment. Nevertheless, a qualitative study, in where several numerical models are conjunctively compared, may give results which can give new insights regarding the physical nature of weakness zones and how their presence alters the stress and strain distribution around excavations. The reason for this is that they all are defined by the same set of assumptions in where the accompanied errors are the same. Henceforth, if the models show satisfactory differences in stress and strain states, an analysis of these differences can be made.

By using FEM modelling it is possible to calculate the changes in stress and strain distributions induced by an excavation process. Furthermore, with this tool it is possible to change the input parameters and investigate changes in the output of the modelling process. More specifically, in this case it would be the parameters deemed important for describing the behavior of weakness zones that would be of interest. This kind of qualitative experiment is called a sensitivity analysis. Saltelli (2002) defines sensitivity analysis as the study of how the uncertainty in the output of a mathematical model or system can be divided and allocated to different sources of uncertainty in its inputs.

Consequently, it was conducted a sensitivity analysis on how the presence of a weakness zones in a host rock affects the distribution of deformation around a circular tunnel. The theoretical and practical basis of stability analysis in rock mechanics and rock engineering was researched in the specialization project (Kaasbøll Andresen, 2021). This project culminated with a discussion on weakness zone parameters effect on stability and was concluded with a suggestion of eight different parameters to be varied in a sensitivity analysis, ranging from material equations to its geometrical features. Regarding the sensitivity analysis, the eight parameters imply a need for a vast number of FEM models to be created and calculated. The reason for this is that the number of FEM models needed is given by the multiple of the number of values tested for each parameter. Thus, the number of models needed increases exponentially and more rapidly for each parameter included in the study.

In accordance with several sources (e.g.Mao & Nilsen, 2013; Trinh et al., 2010), the commercial software package RS2 made by RockScienceTM is the most used FEM-program for 2D-analysis in the industry. However, RS2 is not developed for experiments in where it is necessary to create many models. The only module created for batch-operations is RS2-compute in where the models to be calculated is prepared in advance. The most demanding task is the creation of the geometry and the allocation of the material behavior of the different geological entities which must be done using keyboard and mouse operations only. Thus, this is a bottle neck for how big the sensitivity study can be designed using RS2 in where the slow construction and material allocation processes limits how many models that can be produced.

A solution to this bottleneck was investigated and deemed possible to develop as part of the thesis. This was possible by combining the Open functionality of the python program language with the module PyAutoGUI distributed by Conda forge. The PyAutoGUI module is also written in python, and the implementation of this combination script was consequently written in python. This combination script's main purpose was to automate the entire modelling process; from the model and mesh construction to the first stages of the data processing.

The automation script was successfully implemented and worked as the backbone for the experimental method of this project. Consequently, this script enables the creation of thousands of numerical models in matter of days. Also, the relative short calculation time made it possible to use it as an integrated part in the development of the method.

A problem with datasets of this magnitude is that one single numerical model is by itself a big dataset. Thus, it was important to fetch parts of each dataset which was of importance for the sensitivity analysis of this thesis. The solution was to only analyze the deformations around the tunnel periphery, since the magnitudes of the deformations are greatest there depending on the intensity of rock failure. Furthermore, it was observed that the deformations near weakness zones often are skewed, implying a moment of force that must be included in calculations when security measures are planned. Therefore, it can be interesting to capture the skewness of the deformation distribution around the tunnel periphery in a way that a big dataset of models can be compared. This can be done by calculating deformation moments, an artificial parameter inspired by the concept of force moments borrowed from static mechanics.

To summarize, the general goal of the thesis was to map which parameters describing the geometry and the material behavior of the weakness zone that are leading to notable changes in the models, and how much each parameter contributes to the leading changes. In the thesis, the output monitored was the deformations on the tunnel periphery. A sensitivity analysis was conducted using FEM modelling as the base of the analysis. To be able to do an analysis with a big enough impact, a script was developed to compensate for RS2's lacking batch functionalities. The method of the experiment was developed by combining insights from the specialization project, and from test experiments using RS2 and the automation script. The key in this process was to map which parameters that was to be varied in the final experiment and which parameters suggested in the specialization project that was to be investigated in the full-scale experiment. A discussion upon: (a) the choices of the model construction in RS2 and of its input parameters, (b) experiences made during the development of the script, and (c) the implications of the results of the sensitivity analysis. It all ends with the conclusions of the sensitivity analysis and culminates with suggestions for further work.

2 Background

Weakness zones is the most prominent reason for instability problems in hard rock areas such as Norway (e.g. Høien et al., 2019).

In this chapter, the background of the development of the sensitivity experiment, and to underline the dilemmas that needs to be solved. The section is subdivided into the following subsections: (a) a summary of the insights gathered in the specialization project, to depict the foundation of the thesis, (b) a presentation of the relevant theory of rock mechanics and its application in rock engineering, (c) a brief representation of the Finite Element Modelling, and (d) a set of definitions of programming related terms and a presentation of the tools used in the development process.

It is important to note that section (a) and (b) is products of the specialization project (Kaasbøll Andresen, 2021). The reason they are included is given in the respective sections.

2.1 Insights of the specialization project

The master thesis is an extension of the preceding specialization project "A literature review on important factors considering a numerical sensitivity study on weakness zones" written in the autumn term 2021 (Kaasbøll Andresen, 2021). Thus, in the specialization project it was performed a literature study on realistic representation of rock mass and weakness zones. The knowledge gathered in the specialization project was valuable in the development of the method of the sensitivity study and in the construction of the models used in the experiments. Therefore, this section works as an overview of the most important insights of the specialization project. This section is a summary of the discussion and conclusion. Together with the coming theory section outlines this section the foundation of the thesis.

The purpose of the literature study was three folded: (a) To clarify the theory on which the numerical models in rock mechanics is based, (b) to ensure that the simplified numerical model abstracts the geological setting in a realistic manner, and (c) to map which parameters describing weakness zones that needs to be investigated further in the planned sensitivity study. The project consisted of two main parts, namely a theory section and a discussion section. The theory section worked as the standalone basis of the discussion since the project was a pure literature review. Point (c) was the theme of the discussion of the specialization project in where point (a) and (b) functioned as the foundation of the argumentation.

Following points where discussed:

(a) The significance of different parameters describing both material and geometry attributions of both host rock and weakness zone, based on the gathered literature.
(b) The bounds of significant parameters describing both material and geometry attributions of both host rock and weakness zone, mostly based on review articles in where large-scale comparisons of test results is done.
(c) Limitations of the literature review in where further investigations were suggested.
(d) Limitations of the theory used in rock engineering.

An important notion, attained from the project, was the dilemma between simplicity and realistic representation. Since mining and civil engineer projects in rock extends over a relatively large volume, and since the rock mass is notoriously heterogenous and often show anisotropic behavior, it is almost an impossible task to gather enough information to represent the rock mass perfectly even though it is possible in theory. This is primarily due to shortcomings of the processes of information gathering in where many of the methods are point measurements. Thus, much effort must be given to ensure that these measurements represent the in-situ state of the rock. Additionally, practical- and economic considerations complicate this situation even further in engineering projects. Thorough investigations are both time consuming and expensive. Consequently, this has affected the practitioners' approach in the field of engineering geology in where empirically based equations describing the deformation and failure behavior of rock mass is vastly used. This approach is based on deductions whereby point-knowledge is used to describe the neighboring rock mass.

To ensure that the model abstracts a realistic geological setting, there was a focus on collected data describing deformation behavior of rock mass and weakness zones, and on the structure and geometry of weakness zones. In general, the variability of the gathered data is significant. However, since the model is meant to depict a weakness zone in a hard host rock from the atmosphere and down roughly 1000 meters into the crust, it is possible to define reasonable restrictions, especially for the host rock (Gercek, 2007; Kulhawy, 1975; SINTEF, 2016). However, several sources indicate that elastic parameters and strength of rock mass often show lower values compared to the intact counterpart due to attributes of the discontinuities (Gercek, 2007; Goodman, 1989; Hudson & Harrison, 1997; Li, 2018).

The geometry (Balsamo et al., 2010; Caine et al., 1996) and material behavior of faults (Riedmüller et al., 2001; Sæter, 2005) were deemed to be rather complex showing great variability in all directions of the extension of the fault. Hence, an assumption of faults showing geometry of a sheet were suggested which is a normal assumption of the fault geometry close to the atmosphere (Nilsen, 2016). Also, faults often appear in conjunction with other faults called fault zones, complicating the picture even further, as illustrated in Figure 2. The density of the damage zone of the fault reduces exponentially when moving away from the fault (Mitchell & Faulkner, 2009; Wilson et al., 2003). Thus, it was suggested to make several layers in the model of the weakness zone in where a gradual increase of strength is accounted for.

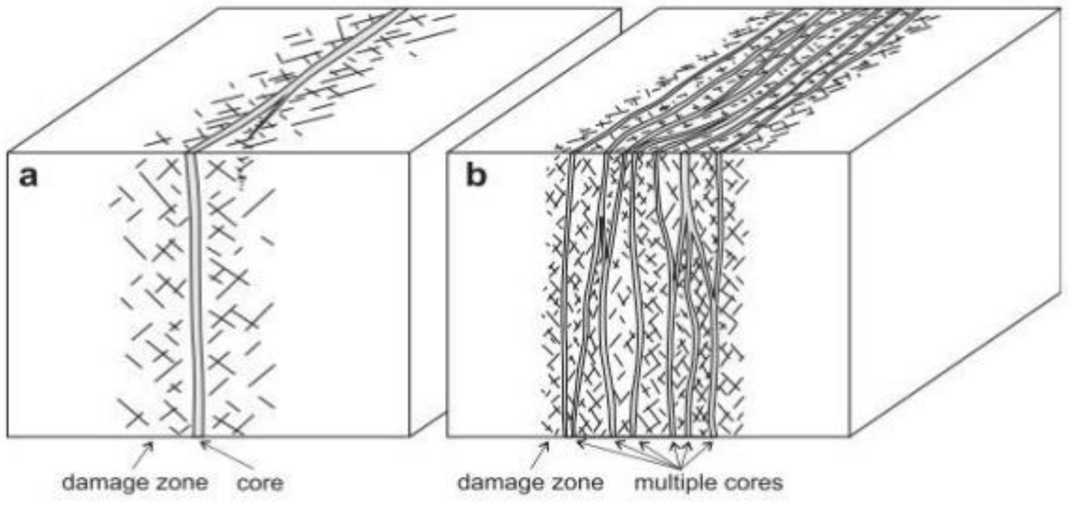


Figure 2 Typical structure of fault zones. (a) Replicates a singular high-strain core surrounded by fractured damage zone and (b) represents a model of multiple high-strain cores in which enclose lenses of fractured protolith. Taken from (Faulkner et al., 2010).

There was not found articles representing data on strength and stiffness of fault rock, breccia, or gouge for engineer geological applications which can be due to difficulties in sampling of data (Kalender et al., 2014). Still, according to Fasching and Vanek (2011) only tectonical breccias and mylonite can be attributed with hard rock behavior. Thus, regular breccias, fault gouge, and cataclasite show mostly soft rock- or soil like behavior. It is suggested to attain data on strength and stiffness of faults for engineer geological applications if possible. If this is not possible, assumptions must be made based on the qualitative descriptions given by Fasching and Vanek (2011).

Following parameters was suggested in the specialization project to be investigated further in the master thesis:

(a) The strength and stiffness of host rock.
(b) Failure criteria for host rock.
(c) The stress distribution.
(d) The strength and stiffness of weakness zone.
(e) Failure criteria for weakness zone.
(f) Number of weakness zones.
(g) Width of weakness zone.
(h) Orientation of weakness zone.
(i) Position of weakness zone relative to tunnel.

2.2 Rock mechanics theory

First of all, this section was written during the specialization project, which was the literature study conducted to prepare for the master thesis (Kaasbøll Andresen, 2021). It is included here since the gathered theory has been central in the development of the method of the thesis and sets the background of the discussion. This is the reason for it coming after the summary of the specialization project's discussion and conclusion. The section has been in some extent edited, but the structure is the same.

Second of all, this section is initiated with a set of definitions central to the study as a whole and is necessary to be defined before the coming subsections. The subsections are systemized in a bottom-up fashion, where the fundamentals of rock mechanics are presented first and where the application of this theory in the field of rock engineering comes in the later subsections.

finally, definitions which is relevant for a specific subsection is represented accordingly.

2.2.1 Important definitions

A **discontinuous** medium is a material containing separations or gaps. The antonym is continuous.

Discontinuities is defined by International Society of Rock Mechanics (ISRM, 1978) as:

> The general term for any mechanical discontinuity in a rock mass having zero or low tensile strength. It is the collective term for most types of joints, weak bedding planes, weak schistosity planes, weakness zones and faults.

Weakness zones Palmström and Stille (2010) defines as follows:

> A part or a zone in the ground in which the mechanical properties are significantly lower than those of the surrounding rock mass. Weakness zones can be faults, shears/shear zones, thrust zones, weak mineral layers, etc.

In Figure 3, weakness zone is defined as the largest length range in the category of discontinuities, ranging from roughly 5 meters in length. Also, according to Palmstrom (1996) faults is one category of weakness zone. In the thesis, it is the behavior of faults that are to be modelled, mainly to narrow the scope. Thus, the term weakness zone is often used interchangeably with the term fault zone.

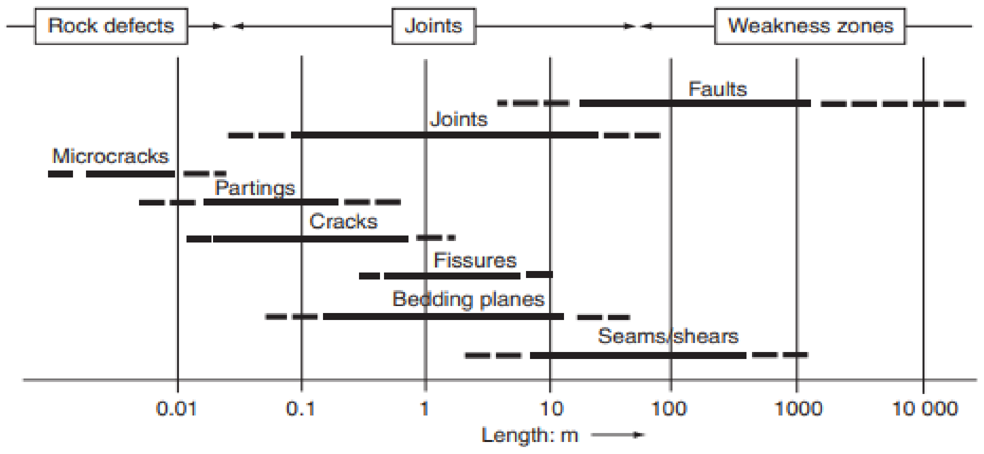


Figure 3: Types of discontinuities given by their lengths, made by Palmstrom (1996).

Intact rock is defined as the rock material within the framework of discontinuities (e.g. Hudson & Harrison, 1997). Nevertheless, micro-cracks are in this definition excluded from the discontinuity category and is regarded as an integrated part of the rock material itself. Furthermore, intact rock and rock material is here considered as synonyms in this project.

Rock mass is the term used when considering the intact rock in conjunction with the weathering of the rock and the rock's discontinuities.

A material behaves **isotropic** if a specific parameter is independent of the orientation of the measurement in a particular point. The antonym is **anisotropic**. For instance, pressure is an isotropic physical quantity and force is anisotropic. Also, these terms are used to describe a variety of physical behavior. Furthermore, in rock mechanics it is also used in a broader sense. If a rock mass is said to be isotropic it refers to the material as a whole and implies that several physical quantities such as acoustic impedance or thermal conductivity will be isotropic.

Homogeneity refers to the situation where the properties of the measurements that is taken is independent of position. The antonym is **inhomogeneity**. For instance, a homogenous rock mass behaves identically over its entire volume. Additionally, the terms isotropy/anisotropy and homogeneity/inhomogeneity are often used in conjunction. For example, in a homogeneous and isotropic rock mass the material properties are both independent of position and direction.

Elasticity is a body's ability to withstand distortion in response to applied forces and to regenerate to its original state when that force is removed or changed. A material is said to be perfectly elastic if it returns to its initial size and shape completely and is in practice not possible in the strictest sense. **Plasticity** is the ability of a body to deform permanently in response to applied forces. A material is said to be perfectly plastic if there is no regeneration to the original size and shape when the forces is removed or changed.

The **strength of rock** is defined by Chapple (1987):

> Strength is the stress level that is required to produce a certain type of permanent deformation, fracture or flow, under well-defined experimental conditions.

2.2.2 Stress

Stress can be defined as the internal forces in which adjacent particles in a continuous material exerts on each other (e.g. Huston & Josephs, 2009). Stress is defined as follows:

$$\sigma = F_{avg}/A \qquad \textbf{(2.1)}$$

where F_{avg} is the average force that a given particle of a body exerts on a neighbouring particle, across an imaginary surface A that separates them. σ is a resultant vector and can be considered a sum of the normal stress σ_n acting normal to the surface A and the shear stress τ acting parallel with surface A.

The **stress state** in three dimensions is defined as the complete description of the stresses in a particular point. So, a body at rest consists of three normal stresses and three independent shear stresses. This term refers to both magnitude and direction. Tensor notation is one way to express the stress state. A tensor is a mathematical object analogous to a vector. However, it is a more general description compared to a vector. In fact, a zero-order tensor is a scalar, and a second order tensor is a vector. The order is referring to the span of the variable. By multiplying a vector with a scalar this vector can span a line which is 1-dimensional. Thus, it is a first order tensor. A scalar multiplied by another scalar gives a third scalar and can therefore not span a space. Thus, it is a zero-order tensor. In general, a tensor is represented by an array of components that are functions of the coordinates of a given space of n dimensions. An important distinction between a vector and a tensor is that the tensor is dependent on the plane in which it is examined, which is not the case of a vector (Hudson & Harrison, 1997). The description of static stress is given by the following second order tensor:

$$\boldsymbol{\sigma} = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{pmatrix} ;\ \sigma_{12} = \sigma_{21} ;\ \sigma_{13} = \sigma_{31} ;\ \sigma_{32} = \sigma_{23} \tag{2.2}$$

This tensor is symmetric since it is defined on the base of static equilibrium, hence the equalities of the shear stresses to the right of the tensor. Subscript i refers to the plane the stress is affecting and subscript j refers to the orientation of the stress related to one of three element vectors of the system. A graphical representation of the components of the stress tensor in a cartesian coordinate system is given in Figure 4.

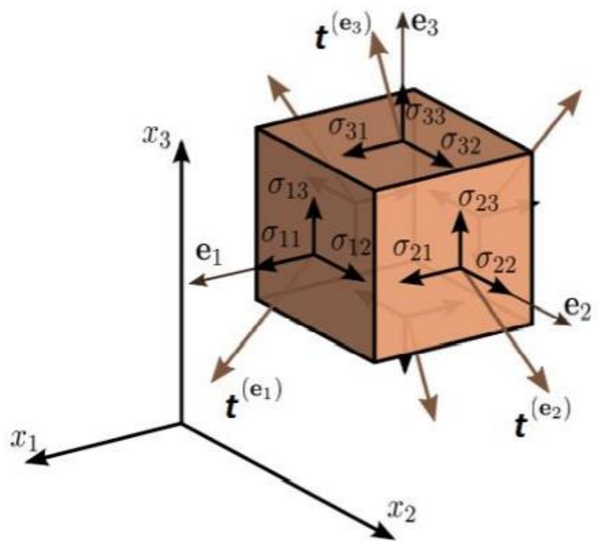


Figure 4: Stress state and vectors at a given point, from RocScience (2021).

The stress tensor is dependent of the coordinate system. A coordinate system oriented such that the shear stresses are zero defines the principal stress space. The principal stress tensor is given by:

$$\boldsymbol{\sigma} = \begin{pmatrix} \sigma_{11} & 0 & 0 \\ 0 & \sigma_{22} & 0 \\ 0 & 0 & \sigma_{33} \end{pmatrix} = \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{pmatrix} \tag{2.3}$$

The principal stresses are the eigen values of the stress tensor given by the diagonal of the principal stress tensor. σ_1 has the largest magnitude of stress out of all orientations of the coordinate system and σ_3 show the lowest magnitude of stress despite of orientation. The three principal stresses are important to map to be able to describe the strength of a material theoretically.

Euler angles (e.g. Roylance, 2001) can be used to transform the orientation of the stress tensor. The following transformation matrix is defined:

$$a = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \textbf{(2.4)}$$

The principle of the transformation is depicted in Figure 5. The rotation matrix rotates a given tensor by one axis at a time. The angles ψ, θ and ϕ refers to the magnitude of rotation about the Z, Y and X axes respectively. The rotation matrix rotates a given tensor by one axis at a time. It starts the rotation about the Z-axis. Then it rotates about the new orientation of the Y-axis denoted y'. Lastly, it rotates about the double-transformed axis of

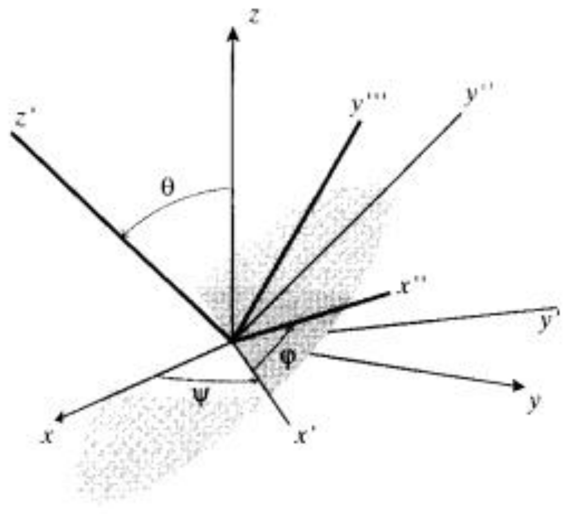


Figure 5: Transformation in terms of Euler angles, from Roylance (2001).

X denoted x''. The new rotation of the coordinate system is then defined by the x'', y'' and z''. A prime refers to an action of rotation in which the given axis was translated. Thus, each axis is translated two times.

The concept of **effective stress** is also relevant. In porous media with hydraulic connectivity, the hydraulic pressure must be accounted for in the following manner:

$$\boldsymbol{\sigma}' = \begin{pmatrix} \sigma_{11} - p & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} - p & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} - p \end{pmatrix}, \qquad \textbf{(2.5)}$$

whereby $\boldsymbol{\sigma}'$ is the effective stress and p is the fluid pressure.

In situ stress is in rock mechanics defined as the stress distribution within the rock mass before excavation has been conducted.

Volumetric stress is related to the volume change of a body and is defined as the average of the three principal stresses:

$$\boldsymbol{\sigma}_{\mathbf{vol}} = \begin{pmatrix} \sigma_{\mathrm{vol}} & 0 & 0 \\ 0 & \sigma_{\mathrm{vol}} & 0 \\ 0 & 0 & \sigma_{\mathrm{vol}} \end{pmatrix}, \tag{2.6}$$

in where $\sigma_{\mathrm{vol}} = (\sigma_{11} + \sigma_{22} + \sigma_{33})/3$ despite the orientation of the basis of the stress state.

Deviatoric stress is related to the shape change of a body and is defined as defined as follows:

$$\boldsymbol{\sigma}_{\mathbf{dev}} = \boldsymbol{\sigma} - \boldsymbol{\sigma}_{\mathbf{vol}}, \tag{2.7}$$

which is a matrix subtraction using Equation (2.5) and Equation (2.6).

Some combinations of the stresses in the stress tensor are invariant of the orientation of the coordinate system. Such combinations are called **invariants**. There are three invariants when regarding the general stress state tensor: $\mathrm{I}_1(\sigma_{\mathrm{ij}})$, $\mathrm{I}_2(\sigma_{\mathrm{ij}})$, and $\mathrm{I}_3(\sigma_{\mathrm{ij}})$. I_1 is the trace of σ_{ij}, I_2 is the sum of the minors of σ_{ij}, and I_3 is the value of the determinant of σ_{ij}. There are also several other invariants, for instance there are two invariants related to the deviatoric stress tensor. Invariants are important in the formulation of several failure criteria.

A **free surface** is defined as the interface between a medium in where share forces can propagate and a medium that lacks this attribute. For instance, the interface between air and rock mass.

A **surface force** is a force exerted on the surface of a body. For instance, the force wind exerts on a windmill is a surface force. On the other hand, a **body force** is a force acting throughout the entire body. For instance, the force gravity exerts on all bodies is a body force.

2.2.3 Strain

Deformations resulted from the work of forces is usually related in terms of relative displacement of particles that excludes rigid-body motion (e.g. Huston & Josephs, 2009). These relative displacements are called strain. Strictly speaking, strain is defined as the displacement between particles related to a reference length and is given as follows:

$$\epsilon = \partial/\partial \mathrm{X}\,(\mathrm{x}-\mathrm{X}), \tag{2.8}$$

by which $\mathbf{X}$ is the reference position of material points of the body, and $\mathbf{x} = \mathbf{F}(\mathbf{X})$ is the deformation of the body in which both are vectors.

In continuum mechanics, the **Cauchy strain** is the expression of the ratio between total deformation and the initial dimension of the material body on which the forces are exerted. The infinitesimal strain tensor are derivatives of displacement components and is as follows:

$$\epsilon = \begin{pmatrix} \epsilon_{11} & \epsilon_{12} & \epsilon_{13} \\ \epsilon_{21} & \epsilon_{22} & \epsilon_{23} \\ \epsilon_{31} & \epsilon_{32} & \epsilon_{33} \end{pmatrix}; \epsilon_{\mathrm{kl}} = \frac{1}{2}(\mathrm{u}_{\mathrm{kl}} + \mathrm{u}_{\mathrm{lk}}) \tag{2.9}$$

This is known as the Cauchy strain tensor or **the small strain tensor** and implies that the material described behaves linearly elastic. Thus, the small strain tensor is by definition symmetric. Transformation and method to find principal strains is conducted similar as the case of stress.

A material shows **strain-hardening** behavior if the resistance towards deformation increases with deformation. The antonym is **strain-softening** behavior.

A **constitutive law** is a formulation describing the relation between stress and strain. The stress is in most often regarded as the cause and strain as the effect.

A system is in **static equilibrium** when there are no changes in the strain distribution.

Rheology is the study of flow and deformation of a material.

The **rheologic mode** of a material is the mathematical rule under which the deformation in a material behaves.

The **peak strength** of a material is the load in which the material fails. This load is used to mark the transition point where the rheological mode of the rock material changes drastically. For instance, for rock salt it goes from behave elastically to behave plastically.

Dilatancy is the volume change observed in granular materials when subjected to shear deformations.

2.2.4 Deformation properties of rock

It is crucial to capture the rock mass' response to a given distribution of stress to get a reliable description of its distribution of deformations. Li (2018) divides this problem into three main components: (a) The **rock mass quality** which is used interchangeably with the strength of the rock mass, (b) the in-situ stresses, and (c) the geometry and size of the **excavation**. These three constituents are mutually dependent. Consequently, by changing attributes belonging to one of the components will lead to changes of the attributes in the other components. This relationship is presented graphically in Figure 6.

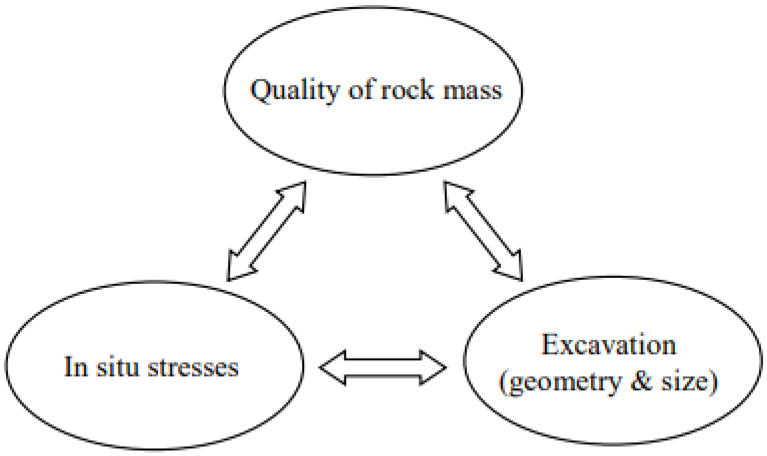


Figure 6: The three main constituents of rock mechanics and their mutual dependency (Li, 2018).

The excavation process induces changes to a massive system which is assumed was in a static equilibrium to begin with. The removal of rock mass leads to alterations of the proximate in situ stress distribution whereby a concentration of stresses at the tunnel periphery develops. If the material shows little cohesion and/or friction in a minor stress field, or the strength of the rock mass is exceeded due to a major stress field deformations will occur. The altered stress field in combination with the absence of restrictions from the removed rock mass leads to convergence of the tunnel perimeter due to the corresponding deformations. So, the material behavior of the rock mass is an important factor controlling the magnitude of the strains, the failure mode, and the upper limit before failure initiates.

In Figure 8, characteristic axial stress-axial strain curves for some rocks are indicating that the behavior of rock materials varies significantly. Constitutive laws are used to describe the relationship between stress and strain and is linked physically to the material behavior. Brady and Brown (1993) states that the idealized constitutive laws depend on time-dependent and time-independent responses of the material in relation to applied load. Constitutive laws describe responses in terms of elasticity, plasticity, **viscosity** and **creep**, or combinations of these idealized cases. A flow chart demonstrating time-independent and time-dependent deformations relevant for tunnelling are presented in Figure 7.

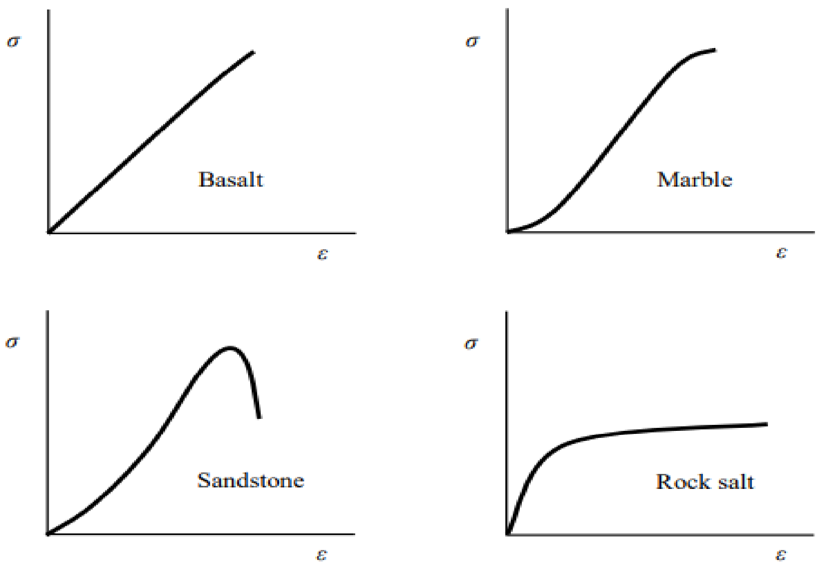


Figure 8: Typical stress-strain curves for rock, taken from Jaeger et al. (2009).

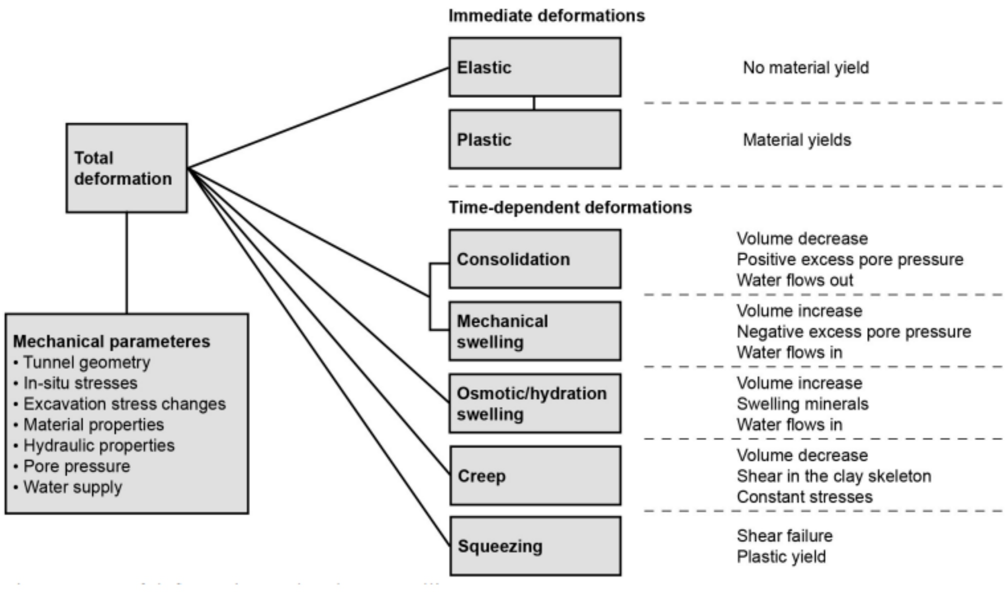


Figure 7: Tunnelling related deformations, made by Høien (2018).

Time-independent deformations

Consider a tunnel under excavation. The face advances through the rock mass along with the removal of rock mass and a three-dimensional free surface is growing accordingly. This leads to alterations of the stress distribution in the vicinity of the face leading to deformations. Hoek et al. (1995) states that the deformation initiates around one-half of the tunnel diameter in front of the face, as portrayed in Figure 9. About one-third of the deformations of the rock mass have taken place at the face. The time-independent deformations reach terminal magnitude approximately one to one and a half tunnel diameters behind the face. Time-independent deformations results from the change of the stress distribution induced by the excavation process. The elastic and plastic attributes of the rock mass are the underlying cause of the occurring deformations.

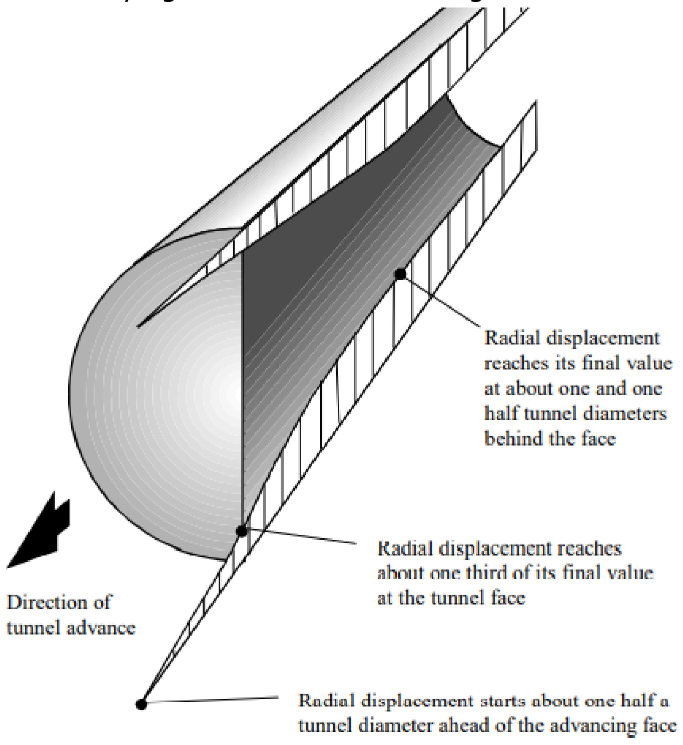

Radial displacement reaches its final value at about one and one half tunnel diameters behind the face
Radial displacement reaches about one third of its final value at the tunnel face
Direction of tunnel advance
Radial displacement starts about one half a tunnel diameter ahead of the advancing face

Figure 9: Pattern of radial displacement in the roof and floor of an advancing tunnel. Illustration made by Hoek et al. (1995).

For small deformations Hooke's elasticity theory for continuous materials is applicable for rock mass. However, an assumption of continuous rock mass is herewith implied. If the load equalizes the materials peak strength, fracturing of the rock will occur and permanent deformations will dominate. The behavior of the material is said to be reaching its plastic deformation state. In general, from observations of stress-strain behavior of rocks in compression tests four stages of deformation behavior is notable (Walton et al., 2019): (a) pre-yield, showing approximately elastic behavior, (b) post-yield but pre-peak, where short term frictional strengthening effects due to the compaction of the material lead to strain-hardening behavior, (c) post-peak weakening, whereby significant cohesion loss under fracture growth and shearing leads to weakening of the rock material, and (d) the residual state, where further strain leads to an approximate constant strength and deformation behavior of the material.

Li (2018) states, due to difficulties measuring the post-peak stage for hard rocks, the post failure behavior of the material is often not known. However, according to Walton et al. (2019) the post-peak residual strength of rock is a relevant parameter to assess in engineer geological problem solving. For instance, in stability assessments of tunnels. Descriptions of rocks' deformation state requires a proper rheology model for the material. This enables a capture of the realistic elastic and plastic response to stress. This rheological model is included in the constitutive model (Li, 2018). In Figure 10, viable idealized rheological models for rock are compared with a realistic depiction of a stress-strain curve of a typical rock.

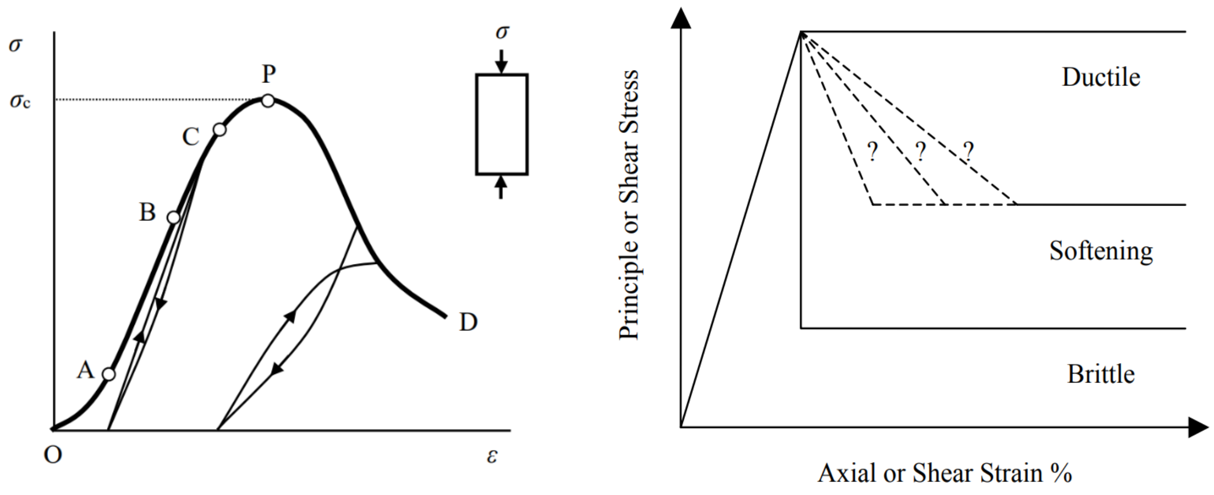


Figure 10: A realistic stress-strain curve (Li, 2018) compared to idealised rheological models of rock masses (Crowder & Bawden, 2004).

Crowder and Bawden (2004) states that rock in general constitutes material behavior between the two ideals of ductile and brittle post-peak deformation modes. A rock that is perfectly ductile in the post-peak stage is said to behave **perfectly elastic/perfectly plastic** and in the brittle case it is said to behave **perfectly elastic/perfectly brittle**. In other words, most rocks show some degree of post-peak strain-softening behavior.

The elastic constitutive model

Hooke's law is the general expression of the relation between stress and strain for an elastic material. The linear equation is:

$$\sigma_{ij} = C_{ijkl}\epsilon_{kl}, \quad \textbf{(2.10)}$$

whereby C_{ijkl} is components of the fourth order constitutive stiffness tensor, σ_{ij} and ϵ_{kl} is the stress and strain components respectively. Equation (2.10) is expressed by means of Voigt notation (Voight, 1928) to simplify the calculation. This notation is applicable for symmetric matrices (e.g. Lamuta, 2019) which means that:

$$\sigma_{ij} = \sigma_{ji},$$
$$\epsilon_{kl} = \epsilon_{lk}, \quad \textbf{(2.11)}$$
$$C_{ijkl} = C_{jikl};\ C_{ijkl} = C_{ijlk};\ C_{ijkl} = C_{klij.}$$

Thus, $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ can be expressed as 6×1 matrices containing the unique stress elements and the fourth order tensor can be reduced to a 6×6 matrix. This results in the following simplified matrix equation:

$$\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{23}\\ \sigma_{13}\\ \sigma_{12}\end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16}\\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26}\\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36}\\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46}\\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56}\\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66}\end{bmatrix} \cdot \begin{bmatrix}\epsilon_{11}\\ \epsilon_{22}\\ \epsilon_{33}\\ 2\epsilon_{23}\\ 2\epsilon_{13}\\ 2\epsilon_{12}\end{bmatrix}. \qquad \textbf{(2.12)}$$

The generic stiffness constant C_{mn} of matrix in Equation (2.12) is equal to the elastic modulus C_{ijkl}, where $m = i$ if $i = j$ or $m = 9 - i - j$ if $i \neq j$, and $n = k$ if $k = l$ or $n = 9 - k - l$ if $k \neq l$.

The stiffness constants $C_{mn} = C_{nm}$ according to the symmetry properties presented in Equation (2.11), and the matrix in Equation (2.12) is therefore symmetric. The stiffness constants C_{mn} are functions of Young's modulus' E_{mn}, Poisson's ratios ν_{mn}, and shear modulus' G_{mn}. The complexity of these functions depends on the material behavior of the medium. A general anisotropic material is characterized by 21 independent elastic constants in static equilibrium. The above mentioned symmetry and the existence of a strain energy function reason behind this constitution(e.g. Lamuta, 2019).

By doing assumptions of the medium's material behavior, it is possible to further decrease the number of independent variables. For example, if the material behaves orthotropic, only 9 elastic variables of Equation (2.12) are independent. The 9 variables consist of three Young's moduli, three Poisson's ratios, and three shear moduli respectively. For instance, a rock mass with three sets of orthogonal discontinuities (three planes of elastic symmetry) dominating the deformation mode would behave orthotropic. According to Ismael et al. (2017) the deformability of the intact rock of sandstones, granites, coals, and schists shows the orthogonal anisotropic nature. If the material behaves transversely isotropic, the number of independent variables describing the constitutive law is reduced to 5. These 5 variables consist of two Young's moduli, two Poisson's ratios, and one shear modulus. Transverse isotropy is often used to describe the elastic symmetry of rocks with one dominant layer system such as foliation, bedding planes or layering. According to Ismael et al. (2017) this assumption is relevant for most metamorphic rocks (e.g. gneisses, phyllites, schists, and slates).

Isotropic materials only have 2 independent parameters. The two independent parameters are the Young's modulus E and the Poisson's ratio ν. In this case the shear modulus G can be expressed as:

$$G = E/2(1 + \nu). \qquad \textbf{(2.13)}$$

Thus, Equation (2.12) simplifies to the equation below:

$$\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{23}\\ \sigma_{13}\\ \sigma_{12}\end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix}1-\nu & \nu & \nu & 0 & 0 & 0\\ \nu & 1-\nu & \nu & 0 & 0 & 0\\ \nu & \nu & 1-\nu & 0 & 0 & 0\\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0\\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0\\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2}\end{bmatrix} . \begin{bmatrix}\epsilon_{11}\\ \epsilon_{22}\\ \epsilon_{33}\\ 2\epsilon_{23}\\ 2\epsilon_{13}\\ 2\epsilon_{12}\end{bmatrix}, \qquad \textbf{(2.14)}$$

where Young's modulus E and Poisson's ratio ν is independent of direction. This is the Hooke's law for isotropic material in three dimensions presented in matrix form. This behavior is prominent in most brittle rocks when the effect of discontinuities is negligible (Ismael et al., 2017).

Young's modulus (E) is a scalar that expresses the elastic deformability of a particular point in the material measured in a specific direction. Generally, it describes the proportionality between axial stress and axial strain and is given by:

$$E = \sigma_a/\epsilon_a, \qquad \textbf{(2.15)}$$

in where σ_a is the axial stress, and $\epsilon_a$is the axial strain. The term axial is here referring to the orientation parallel with the applied load and E is consequently a function of loading direction. In addition, if Young's modulus is constant for any load in a particular point in a given direction, the material shows **linear elastic behavior** in that point. The term linear is here referring to the shape of the stress-strain curve which is linear.

Young's modulus is usually attained by uniaxial or triaxial compression tests on small rock samples with cylindrical shape. Typical values for Young's modulus taken in uniaxial compression on intact rock specimens is presented in Figure 11. The variability is dependent on rock type. Strength and stiffness parameters given a big dataset of rocks from North and South America was reviewed by Kulhawy (1975).He found Youngs's modulus of intact rock to vary between 1.2 and 99.4 GPa, which is in good agreement with SINTEF's data. Detailed summary of the Young's modulus of different categories of rock is given in Appendix (1).

Poisson's ratio gives the relation between the transverse and axial strains. Gercek (2007) defines the Poisson's ratio as the negative of the ratio of transverse strain to the axial strain in an elastic material subjected to a uniaxial stress. This definition is closely linked

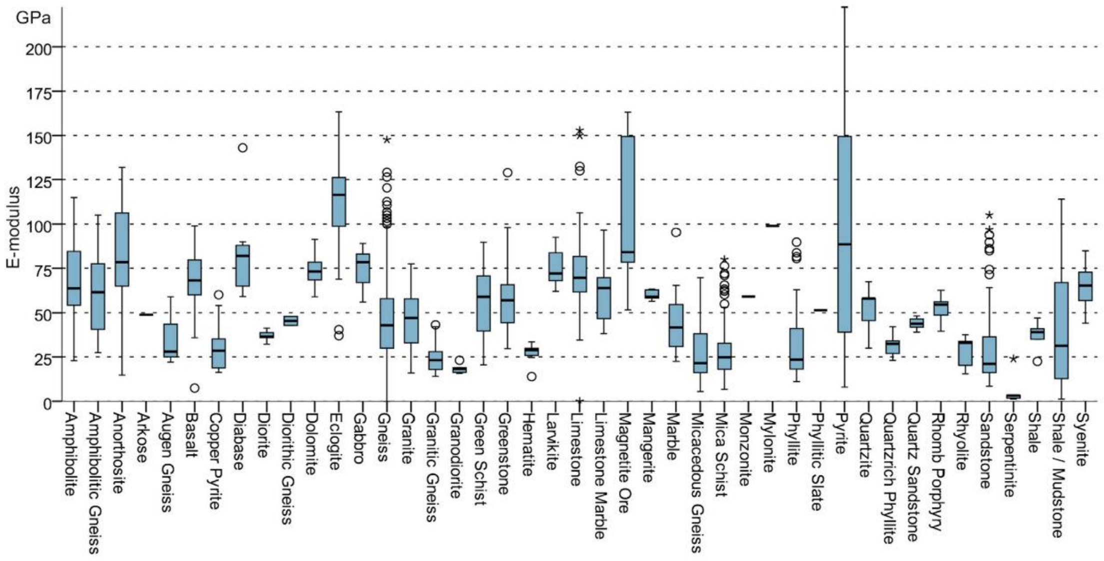


Figure 11: Uniaxial E-modules in GPa for different Norwegian rock types, extracted and made by Høien et al. (2019) from SINTEF rock mechanical properties database (SINTEF, 2016).

to the equation of the ratio which is given as:

$$\nu = -\epsilon_t/\epsilon_a, \tag{2.16}$$

where ν is Poisson's ratio, ϵ_t is the transversal strain, and ϵ_a is the axial strain. The term transverse strain is referring to the relative deformation along one direction normal to the direction of the applied force. The term axial strain is referring to the relative deformation parallel with the direction of the applied force.

An isotropic material has a Poisson's ratio independent of direction. Gercek (2007) states that Poisson's ratio has no significant effect on the stress distribution in plane elasticity problems with no body forces acting. Gercek claims that the effect of Poisson's ratio may be significant in 3-dimensional stress problems despite small variations of its values. Ranges of some Poisson's ratios of intact rock specimens is presented in Figure 12. Kulhawy (1975) also reviewed Poisson's ratios in his analysis of elastic parameters of rocks from the Americas and found it to vary between 0.02 and 0.46 (0.7 if chemical sedimentary rock is included). This is in good agreement with the data presented by Gercek. Detailed summary of the Poisson's of different categories of rock is given in Appendix (1).

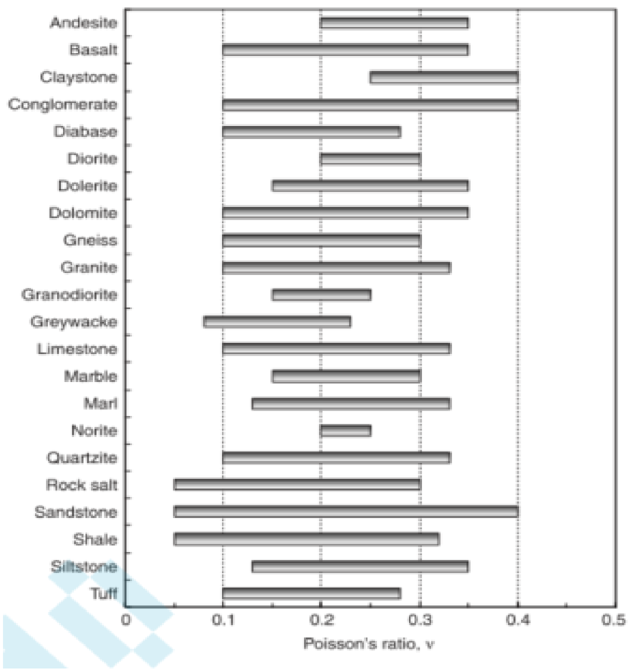


Figure 12: Range of Poisson's ratio for some intact rock types extracted by Gercek (2007) from data presented by (Hatheway & Kiersch, 1986; METU, 1989a, 1989b, 1989c; Vutukuri et al., 1974)

Shear modulus is the ratio of shear stress to the shear strain and is the measure of elastic shear stiffness in a particular point in a given direction. The relation is as follows:

$$G = \tau/\gamma, \qquad \textbf{(2.17)}$$

whereby G is the shear modulus, τ is the shear stress, and γ is the shear strain. As shown above, this modulus can be expressed with Young's modulus and Poisson's modulus if the material behaves isotropic. If not, the shear modulus must be measured. No data on this parameter was found in the literature study.

Strictly speaking, intact rock is does not behave perfectly elastic due to permanent deformation of micro cracks, weak minerals and pores in the rock material under loading in conjunction with the fracturing which occurs when peak-strength is reached. The permanent plastic deformation is referred to as **hysteresis**. Despite of this, elastic rheologic mode is the most frequent description for crystallized rock, given that the load is kept beneath peak strength. Figure 13 illustrates the elastic behavior of crystalline rocks, which consists of the stress-strain curves from uniaxial compression tests on six rock samples obtained by Wawersik (1968). The six different rock samples all show close to linear elastic behavior before fracturing, given by the peak stress value.

Also, Figure 13 presents Wawersik's two classes of rock based on the deformation mode after peak stress is reached. The fracture propagation of class 1 rocks is seen to be stable after initiation because each increment of strain beyond peak-load requires an increment of work to be done on the rock sample. In contrast, class 2 rock types are characterized by unstable fracture growth in where elastic energy release must be controlled to control failure. Class 2 rocks are said to be brittle and may fail due to rock burst in high stress areas, for instance in a road tunnel in the bottom of a steep valley near the surface. Class 1 rocks, however, is said to be ductile and may show squeezing behavior when overly stressed. This is a problem well known for rock engineers working in the alps or in the Himalayas, which consists of relatively young rock types compared to the Norwegian mountain ranges.

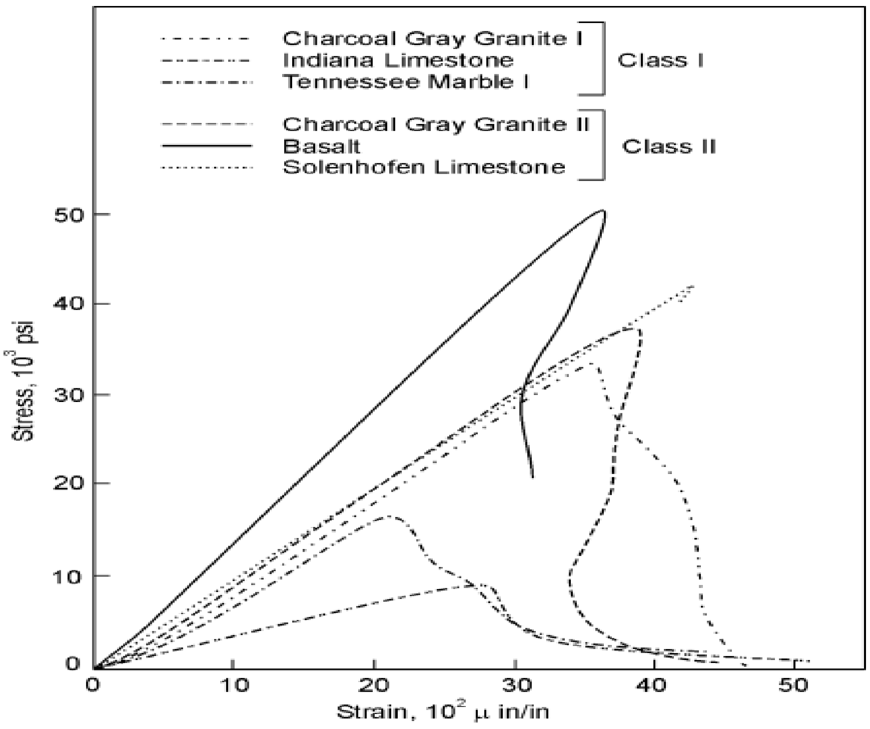


Figure 13: Stress-strain curves for six representative rocks in uniaxial compression, illustration made by Wawersik (1968).

Influence of confining pressure

Jaeger et al. (2009) investigated the effect of confining pressure on strength of rock using conventional triaxial tests where $\sigma_1 > \sigma_2 = \sigma_3$. The result of this study is depicted in Figure 14. Generally, an intact rocks' strength, its linear elastic range and degree of plasticity increases with increasing confining pressure. Though, it does not have a severe impact on the Young's modulus magnitude. Multiple shear fracture is the failure mode which dominates with increasing confinement pressure instead of the axial splitting seen in UCS-tests. Also, the material shows a dilation decrease as a function of increasing confining pressure. The reason for this is the degree of internal fracturing (Elliott, 1982) being lowered. A last remark, Mogi (1967) showed that if $\sigma_2 > \sigma_3$ an even more prominent strengthening of the material will occur indicating that information of σ_2 should not necessarily be neglected.

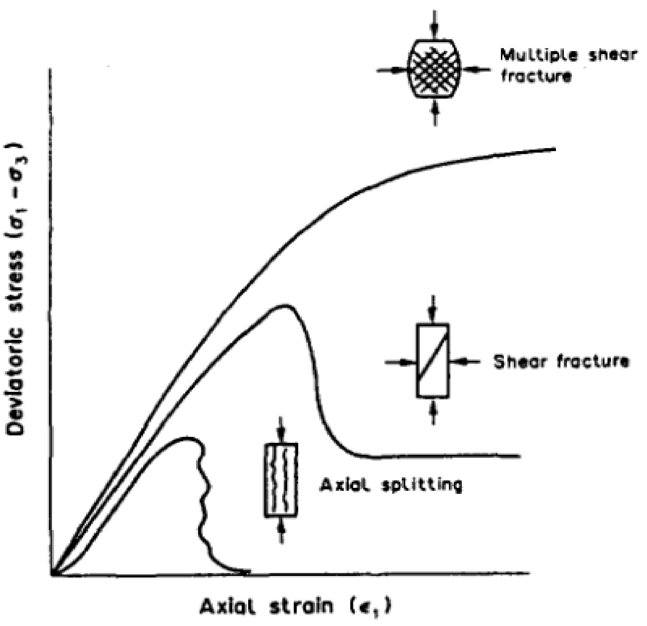


Figure 14: Rock specimen get more ductile with increased confining pressure which also affects the failure mode, borrowed from Jaeger et al. (2009).

Influence of water

Rocks tend to decrease in degree of elasticity and in strength with increasing moisture content. In their review, Wong et al. (2016) found that it was no universally approved explanation for the influence of water on rock strength and elasticity. Yet, two insights of the water-weakening effects were mentioned: (a) A negative exponential and/or negative power relationship is observed between rock strength/Young's modulus and water content, and (b) an exponential relationship has been observed between tensile strength in regard to both surface tension and the dielectric constant of the saturating liquid. According to their result a rock can show strength reductions between 2% observed in some tuffs and granites. In clay shales, siltstones, and mudstones, however, the rock strength was reduced to over 90%. Young's modulus showed near 0% reduction in magnitude in tuff, slate, sandstone. In shale, siltstone, and mudstone the reduction in Young's modulus was over 80%. Thus, laboratory measurements should be done with the same liquid content as in the field.

Water also reduces the strength due to water pressure in pores and discontinuities. Terzaghi (1923) formulated the concept of effective stress for soils in where the intergranular stress controls the strength and deformation of the rock or soil. To apply the concept of effective stress in rock engineering purposes, it is necessary to gather information concerning the water pressure distribution within the rock and rock mass. Hoek and Brown (1997) describes the method of direct measurements, using piezometers, and the methods of indirect measurements estimated from manually constructed or numerically generated flow nets. Nevertheless, in low permeability rocks it may seem that the effective stress law gives unreliable results. Brace and Martin Iii (1968) conducted a triaxial tests of low permeable crystalline silicate rocks. In this study it was found that the effective stress law only was reliable for strain rates under a critical value. Static equilibrium was not established for higher strain rates. Besides, on the word of Hoek and Brown (1997) and Li (2018) low pore pressures along tunnels and other cavities is normal due to their draining effects on the rock mass. Thus, the concept of effective stress can be neglected in such cases.

Influence of temperature

Hudson and Harrison (1997) states that little information is available indicating the effect of temperature on the stress-strain curve. However, a decrease in strength and Young's modulus is observed with increasing temperature. Also, it is observed an increase of the rocks ductile character in the post-peak stage with increasing temperature. Figure 15 presents these effects graphically. In this graph the strength of the material is reduced by 50% when increasing the temperature from 300° to 600°. With the same temperature

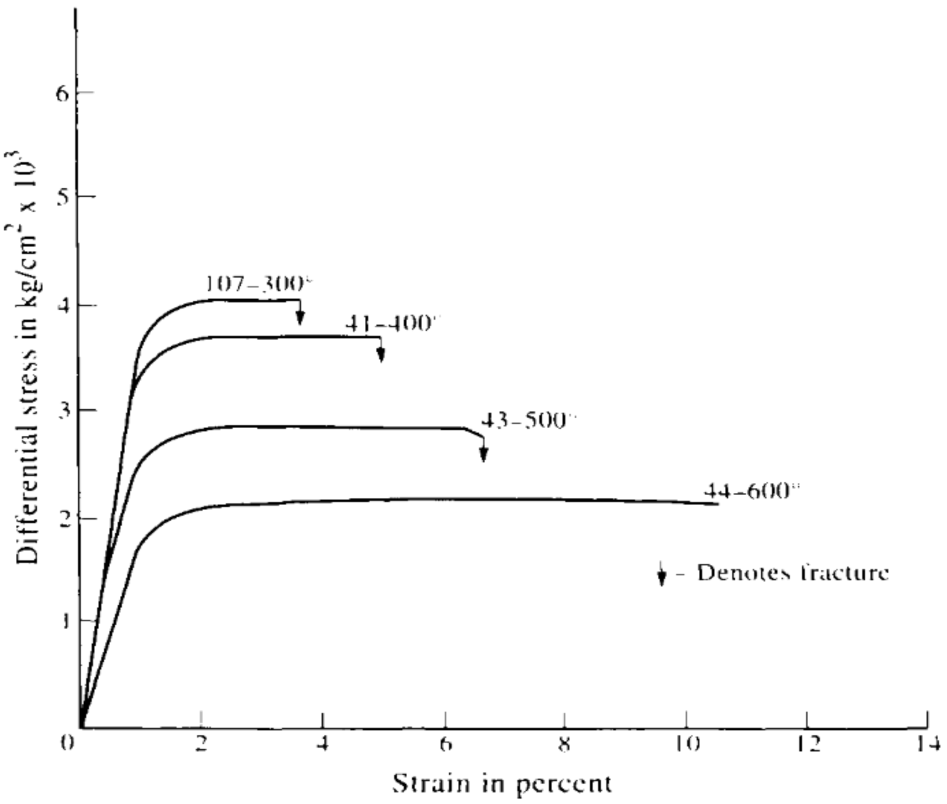


Figure 15: The effect of temperature on the stress-strain curve, taken from Hudson and Harrison (1997).

reduction, the range of the plasticity zone is more than tripled. Lastly, with higher temperatures it is indicating that the magnitude of Young's modulus is reduced significantly. In addition, Sheorey (1994) estimate the average temperature gradient of the earth's crust to be $0.024^{o}\mathrm{C/m}$.

2.2.5 Plasticity

Theoretically speaking, plastic materials' deformation in response to constant stress continues indefinitely without any further increase of this stress. Also, the strain energy is changing its state through permanent deformations only. Plastic behavior is generally a function of distortional strains and deviatoric stresses. Stresses are related to strains in a time-independent manner in where the material undergoes plastic flow when stressed. (Hudson & Harrison, 1997)

Knowing the post-peak behavior of the rock mass is important to define the true deformation mode of a rock mass. The rock mass is often assumed to behave linear elastically before fracture initiation and to some degree behave plastically after fracturing. Plasticity theory is used to describe the post peak behavior often in conjunction with elasticity theory where the defined **failure criterion** is also treated as the **yield criterion**. Plasticity theory postulates that irreversible strains occur whenever the stress state satisfies the yield criterion and is based on experimental evidence (Luenberger & Ye, 1984). The general failure criterion of a material is:

$$\mathrm{f}(\sigma_{\mathrm{ij}}, \kappa) = 0, \tag{2.18}$$

in where σ_{ij} is the stress tensor and κ represents the material parameters and f is the failure criterion function. An elastoplastic model is based on the assumption that the strain state is dependent on elastic and plastic components only, and that their relation is additive (or in some occasions also multiplicative):

$$\epsilon_{\mathrm{tot}} = \epsilon_{\mathrm{e}} + \epsilon_{\mathrm{p}}, \tag{2.19}$$

in where ϵ_{e} is elastic strain state and ϵ_{p} is the plastic strain state. In such a model, if $\mathrm{f} < 0$ the plastic strain state $\mathrm{e_p} = 0$. Vermeer and De Borst (1984) states that there is no one-to-one correspondence between plastic stresses and strains, which is the case for elastic stresses and strains. Thus, the elastic part of the strain obeys Hooke's law. The plastic part, however, is described by a flow rule:

$$\epsilon_{\mathrm{p}} = \lambda \nabla \mathrm{g}, \qquad \nabla \mathrm{g} = \partial \mathrm{g}/\partial \sigma, \tag{2.20}$$

where, λ, is a non-negative multiplier if plastic loading occurs. This multiplier is zero when the deformation mode is purely elastic. Furthermore, g is called the plastic potential function and is a scalar function. For most metals $\mathrm{f} = \mathrm{g}$, this is not the case for most geological materials because of dilatancy in where $\mathrm{f} \neq \mathrm{g}$. The term dilatancy refers to the volume change that happens when granular entities are sliding over each other when plastic deformation occurs. Also, it is necessary to define a hardening rule for most geomaterials. A hardening rule enables the consideration of the strain-softening and/or strain-hardening behavior of the geomaterial.

Hoek and Brown (1997) states that there are no definite parameters to describe the elastoplastic behavior of intact rock. Discontinuities of the rock mass complicates the description of this behavior even further. According to Li (2018), the gathering of the post-peak behavior from uniaxial and triaxial tests is a rather complex task often leading to inconclusive test results. Thus, in modelling purposes in it is normal to assume: (a) Perfectly elastic-brittle behavior for strong rock, (b) strain softening behavior for medium strong rock, and (c) perfect elastic-plastic behavior for weak rock. These post-peak characteristics are given in Figure 16.

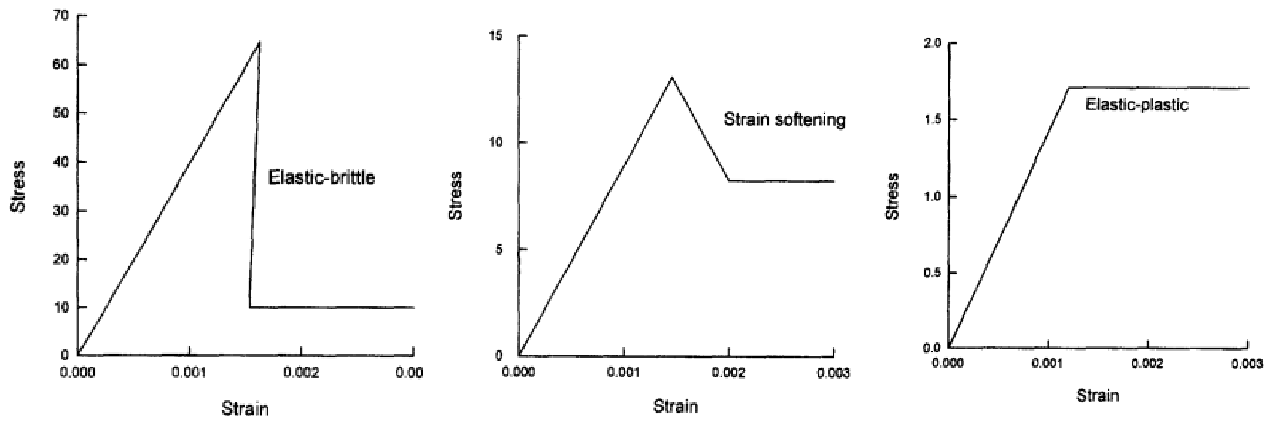


Figure 16: Post-peak behaviour for three different idealistic situations, as illustrated by Hoek and Brown (1997).

2.2.6 Time-dependent deformation

Time-dependent deformations further complicates the analysis of the deformation mode of rock materials (Hudson & Harrison, 1997). In most cases, where time-dependent behavior is present, it is usual to assume that the material behaves elasto-viscoplastic. This term refers to three attributes of such a material: (a) It show time-independent elastic behavior, (b) it shows time-dependent viscous behavior, and (c) if a certain limit is reached plastic flow occurs. Time-dependent viscous behavior consists of strains related to deviatoric stresses and distortional strains in where the stresses is related to strain rates. The term viscous is used to underline that the solid material in some extent has flow properties and behaves in some extent like a fluid. The three constituents of the elasto-viscoplastic model are assumed to be additive and gives the following relation:

$$\epsilon_{\mathrm{tot}} = \epsilon_{\mathrm{e}} + \epsilon_{\mathrm{p}} + \epsilon_{\mathrm{t}}, \qquad \mathbf{(2.21)}$$

in where ϵ_{e} is elastic strain state, ϵ_{p} is the plastic strain state and ϵ_{t} is the time-dependent stress state.

Høien (2018) states that the most prominent time-dependent processes are: (a) mineral and mechanical swelling, (b) creep, and (c) consolidation. All three time-dependent processes leads to convergence of the tunnel walls. Einstein (1996) defined swelling as the: "time-dependent volume increase of the ground, leading to inward movement of the tunnel perimeter". Furthermore, Einstein defined squeezing as the "time-dependent shearing of the ground, leading to inward movement of the tunnel perimeter".

According to Høien (2018), changes in pore pressure may occur as a consequence of changes in the stress state due to excavation processes. Positive excess pore pressure may build up, making the water to flow out of the rock mass. Consequently, the material consolidates. On the other hand, if it leads to negative pore pressure, water will flow toward the rock mass and mechanical swelling will occur.

Deformation occurring from the shear failure over time given constant stress is called creep. This effect is independent of the consolidation/mechanical swelling. This implies that the two processes may occur simultaneously. Also, Høien (2018) states that creep is related to the time-dependent properties of the grain skeleton and is usually observed in drained rock mass. Li (2018) divides the creep process into three successive periods: (a) The primary period where the strain rate decreases with time, (b) the secondary period where the strain rate is constant with time, and (c) the tertiary period where the strain rate increases with time. The components of the creep curve are given in Figure 17.

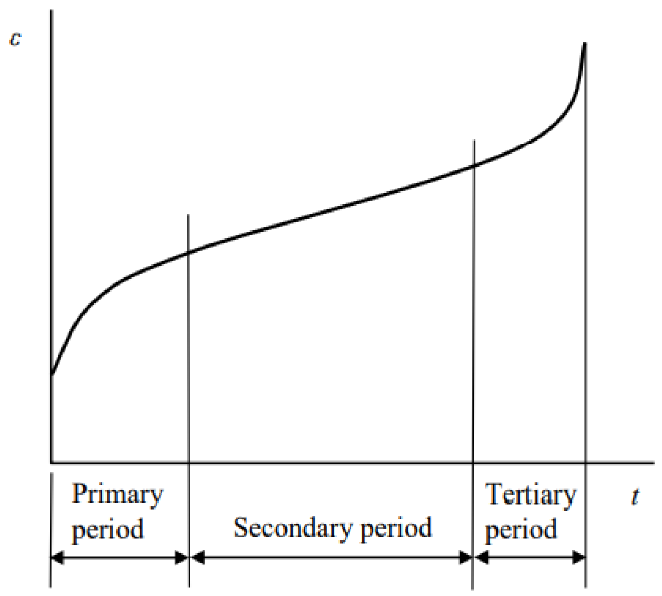


Figure 17: Creeping curve of imaginary overstressed weak rock, taken from Li (2018).

The Burger model can be applied to describe the primary and secondary periods. The Burger model is an idealization of the two first period of the time-strain curve given in Figure 17 and is as follows:

$$\epsilon = \frac{\sigma_c}{E_m} + \frac{\sigma_c t}{3\eta_m} + \frac{\sigma_c}{E_k}\left(1 - \exp\left(-\frac{E_k t}{3\eta_k}\right)\right), \tag{2.22}$$

where the first term refers to the immediate elastic deformation, the second term to the linear part of the creep curve in the second period showing a combination of plastic and time-dependent deformation, and the third term is describing the lowering of strain rate in the primary period.

The third term is for the most due to volumetric stresses, which is the uniform stress in the radial and axial directions. Li (2018) states that it is not possible to model the tertiary state. The reason for this is the lack of rheological models to describe this behavior of creep. The tertiary period only appears when the applied stress is beyond approximately 75% of the UCS. Besides, the tertiary period is linked to resolute failure due to the accelerating nature and must be avoided to avoid tunnel collapse.

According to Høien (2018), a redistribution of stresses around the tunnel profile occurs during time-dependent deformations. The change of stress regime may lead to new deformation processes in areas that before were in static equilibrium. This, again, can be the beginning of a domino effect. Høien underlines that the processes of time-dependent deformations are hard to determine, and several processes may occur simultaneously. There is therefore difficult to separate them. Also, it can be even harder to map each of the processes' contributions in the resultant stress and strain field. Time-dependent deformations are more prominent in overstressed weak rock masses and weakness zones such as faults.

2.2.7 Strength of intact rock

As already established, the rock strength is the tipping point between the linear elastic deformation mode of the rock and the post-failure deformation mode of the rock. A brittle rock is characterized by a narrow or no plastic yield zone and shows high degree of strain-softening behavior leading to violent strain release. A ductile rock has an extensive plastic yield zone where little to no strain-softening behavior occurs. Also, most rocks post-peak time-independent deformation modes resides in between of these two extremes.

According to Hudson and Harrison (1997), several rock strength tests have been developed over the years. Figure 18 illustrates the experimental loading conditions for intact rock strength testing. The main concept of all these tests is to gradually increase the forces exerted on the material until it breaks. Consequently, a test on a specific specimen can only be conducted once and pre-experiment planning is vital.

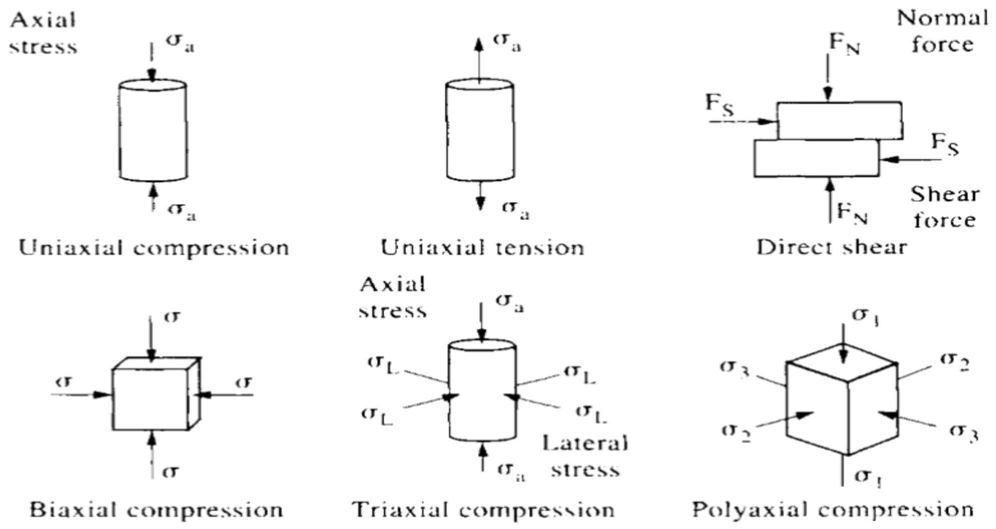


Figure 18: Specimen loading condition for experimental strength tests, from Hudson and Harrison (1997).

Uniaxial, triaxial and poly-axial compression test methods can be used to capture the compressional strength of an intact rock. Uniaxial compression test is the simplest and fastest to conduct, and is therefore the most common. The uniaxial compression test is done without confinement pressure. This implies that $\sigma_2 = \sigma_3 = 0$. Triaxial tests is also extensively used. This test, unlike the uniaxial test, is done with confinement pressure such that $\sigma_1 \neq \sigma_2 = \sigma_3$. The term triaxial is in this regard somewhat vague. It means that all three principal stresses are non-zero. Yet, it does not communicate that $\sigma_2 = \sigma_3$. In practise, this makes for an easier test to conduct compared with the poly-axial test whereby $\sigma_1 \neq \sigma_2 \neq \sigma_3$. The reason for this is two folded. First of all, both uniaxial tests and triaxial tests is done on cylinder shaped specimens, which is the shape of rock specimens after core drilling and only the ends must be prepared. Specimens for poly-axial test specimens, however, must be prism shaped. Second of all, the poly-axial test apparatus and procedure is more complex compared to the triaxial case. method itself and the preparation of test specimens. However, it is the poly-axial test that provides the most realistic results (Wawersik et al., 1997) which is important for more detailed modelling projects.

Biaxial compression tests and uniaxial tension tests is used to decide the intact rocks tensile strength. Rock is well known for the fact that the tensile strength is small compared to the compression strength (e.g. Palmström & Stille, 2010). Shear strength can be found using direct shear test and is mostly used to find strength of discontinuities.

The compressional, tensional and shear strengths of intact rock are closely linked to the lithology. The variation of strength is as severe between rock types as it is within a specific rock type. Accordingly, this depicts the great variability of mineralogical compounds and bonding strengths. Yet again, this illustrates the great variability of the geological setting under which the rocks have been created and resided (Barton, 1976; Kulhawy, 1975). Figure 19: Uniaxial compressive strength (UCS) in MPa for Norwegian rock types extracted

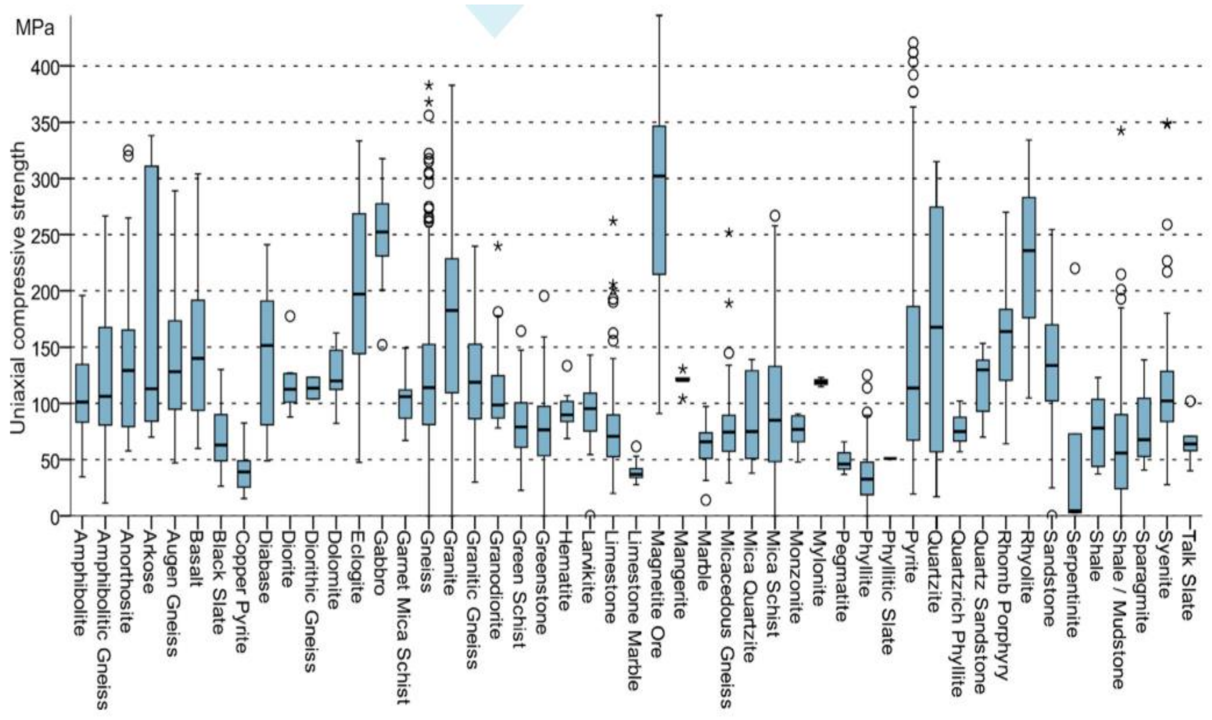


Figure 19: Uniaxial compressive strength (UCS) in MPa for Norwegian rock types extracted and made by Høien et al. (2019) from SINTEF rock mechanical properties database (SINTEF, 2016).

and made by Høien et al. (2019) from SINTEF rock mechanical properties database (SINTEF, 2016). illustrates the mentioned variability of the uniaxial compression strength of Norwegian rocks. This box plot is based on SINTEF's dataset attained with uniaxial compression tests conducted at SINTEF's rock mechanical laboratory. It can be seen that the variability of uniaxial compression strength varies significantly. For instance, compare the magnetite ore and the limestone marble.

As earlier mentioned, Kulhawy (1975) analyzed a vast number of data on rock mechanical parameters of rocks residing in the Americas. It was found that the variability of tensile strength is much less compared to compression strength. For uniaxial stress tests the range of tensile strength of rocks was between 0.55 and 17.4 MPa. For compressive strength, on the other hand, the range was between 3.65 and 355 MPa. Kulhawy did not investigate shear strengths. However, Kulhawy's article presents all the raw data analyzed and can be used to give sensible guesses on the relation between UCS and tensile strength for a given rock. As well, Perras and Diederichs (2014) tried to correlate UCS values to tensile stress to make estimations in a preliminary phase of engineering projects. No correlation was established, yet, they found a correlation between tensile stress and the crack initiation threshold.

Barton (1976) argued that the variability of shear strength is great at low normal stresses. This is indicated in Figure 20, in where the shear stresses are more clustered for low normal stresses. As the normal stress increases the variability of the shear stress increases dramatically. The data was attained by direct shear test on specimens with a single discontinuity.

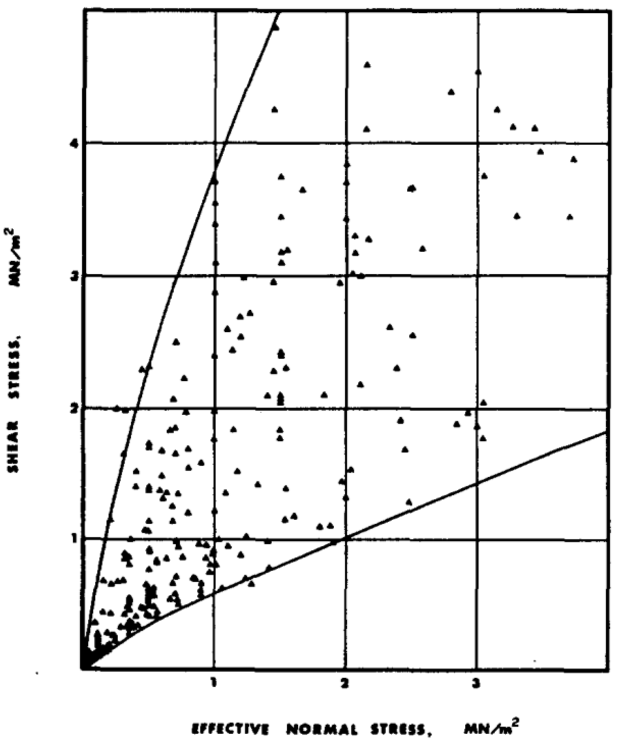


Figure 20: The peak shear strength of unfilled rock joints from, made by Barton (1976).

2.2.8 Failure criteria of intact rock

Hudson and Harrison (1997) states that there is a knowledge gap on the exact physical nature of rock failure. This statement refers to two different aspects. First of all, the precise description of microcrack initiation and propagation is unknown. Second of all, the mode of the structural breakdown which occurs when microcracks propagates and coalesce is unknown. Both processes are complex and hard to describe, and there is no convenient way to create simplified models. Nonetheless, in practical engineering there is necessary to predict how, where and when failure will occur. To accomplish this, the concept of failure criteria is applied.

According to Hudson and Harrison (1997), it has been a tradition in the field of structural mechanics to regard stress as the cause and strain as the effect when deformation mode of materials is analyzed. Hence, it has been normal to express strength of rock in terms of function of stress:

$$\text{strength} = \mathrm{f}(\sigma_1, \sigma_2, \sigma_3). \qquad \textbf{(2.23)}$$

Failure criteria extensively used in the field of rock engineering, such as the Mohr-Coulomb and the Hoek-Brown criteria is of this type. Though, it is possible to either express strength in terms of strains:

$$\text{strength} = \mathrm{f}(\epsilon_1, \epsilon_2, \epsilon_3), \qquad \textbf{(2.24)}$$

or in terms of a combination of stresses and strains:

$$\text{strength} = \mathrm{f}(\sigma_1, \sigma_2, \sigma_3, \epsilon_1, \epsilon_2, \epsilon_3), \qquad \textbf{(2.25)}$$

Still, Hudson and Harrison (1997) states that the two latter criteria is not in extensive use.

It is frequently assumed that the intermediate principal stress has insignificant effect on rock strength in rock mechanical applications. This is the underlaying assumption when mechanical characteristics is studied through triaxial tests on cylindrical specimens in where $\sigma_2 = \sigma_3$. Triaxial tests have been widely used to this day because of the convenience of the equipment, specimen preparation and testing procedures. (ISRM, 2014)

Massive rocks such as granite, gabbro, quartzite, and marble have in practice shown approximately linear elastic and isotropic material behavior. Although, they are in some extent elastically anisotropic (Li, 2018). Consequently, it is normal to assume that these rocks or equivalents are linear elastic and isotropic. This assumption enables the application of failure criteria developed for materials behaving accordingly. Mohr-Coulomb and Hoek-Brown for intact rock are examples of such criteria, and is also frequently used in rock engineering applications.

Mohr-Coulomb failure criterion

A failure criterion based on the shear failure of glass was proposed by Coulomb (1773). It was observed that the shear strength was dependent on the materials **cohesion** and a constant multiplied with the normal stress acting on the shear plane. Cohesion is a measure of the particles' ability to adhere together. Coulomb proposed the following criteria for shear failure of soils:

$$S = ca + 1/n * N, \tag{2.26}$$

whereby c is the cohesion per unit area, a is the area of the shear plane, N is the normal force on the shear plane and $1/n$ is the **coefficient of the internal friction**. The coefficient of the internal friction is a measure of the resistance the particles have when sliding over each other

Mohr (1882) invented a graphical method to describe the stress state in a material in all orientations given a 2-dimensional plane. Two equations were needed; one to describe the normal stress and one to describe the shear stress, both acting on an incrementally small imaginary surface. The equations are:

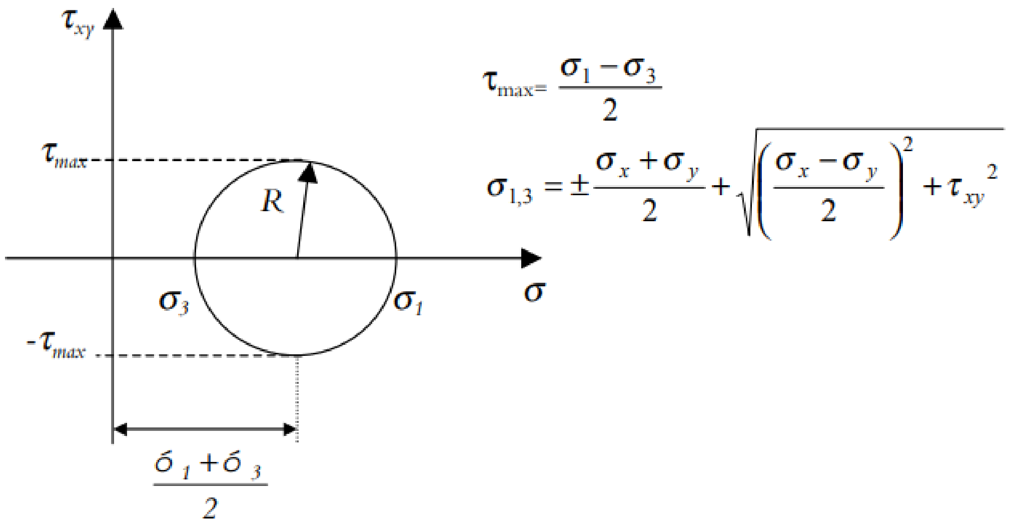
$$\sigma(\theta) = \frac{\sigma_x + \sigma_y}{2} + \frac{\sigma_x - \sigma_y}{2}\cos(2\theta) + \tau_{xy}\sin(2\theta), \tag{2.27}$$

$$\tau(\theta) = -\frac{\sigma_x - \sigma_y}{2}\sin(2\theta) + \tau_{xy}\cos(2\theta), \tag{2.28}$$

in where θ is the orientation of the infinitesimal plane, on which the normal stress σ and the shear stress τ exerts. With mathematical manipulations, Mohr eliminated θ from these equations. This resulted to the expression of Mohr's circle:

$$\left(\sigma(\theta) - \frac{1}{2}(\sigma_x + \sigma_y)\right)^2 + \tau(\theta)^2 = \left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + \tau_{xy}^2 = R^2 \tag{2.29}$$

Mohr's circle is used as a useful tool to graphically illustrate different properties of the stress state. An example is given in Figure 21. For instance, it can be seen that the maximal shear stress occurs when $\sigma(\theta) = \sigma_{vol}$ in where σ_{vol} is the mean stress. An important note is that Mohr's circle is oriented along with the orientation of σ_1 such that $\sigma_{vol} + R(\alpha = 0) = \sigma_1$. In other words, the angle $\alpha = 0$ is defined as the starting point of the stress state description and begins at σ_1 revolving counter-clockwise when $\alpha > 0$.



τ_{xy}
$\tau_{\max} = \frac{\sigma_1 - \sigma_3}{2}$
$\sigma_{1,3} = \pm\frac{\sigma_x + \sigma_y}{2} + \sqrt{\left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + \tau_{xy}^{\ 2}}$
τ_{max}
R
σ_3
σ_1
σ
$-\tau_{max}$
$\frac{\sigma_1 + \sigma_3}{2}$

Figure 21: Mohr's circle, illustration made by Edelbro (2003).

A **Mohr envelope of failure** is constructed by plotting the Mohr's circles at failure belonging to similar specimens for different confining stresses using triaxial tests. A given circle is defined by one pair of σ_1 and σ_3. The trace of the tangents of the Mohr's circles defines the Mohr envelope, as seen in Figure 22. Mohr state that failure occurs when a Mohr's circle for a point within a material surpasses the envelope. Also, in practice, the Mohr envelope must be set by taking a best fit of the data at hand, which is illustrated in Figure 22.

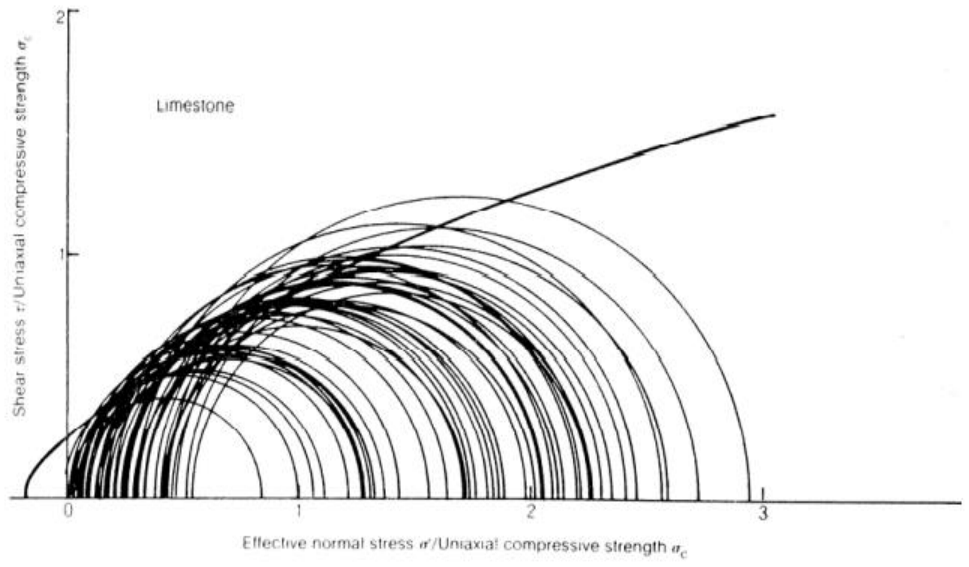


Figure 22: Mohr failure circles and envelope for a limestone given from published triaxial data for intact samples, figure borrowed from Hoek (1983).

The influence of the intermediate principal stress, σ_2, was neglected in the work of Mohr and Coulomb. This implies that the failure plane is parallel with the direction of σ_2. The angle between σ_1 and the failure plane will decrease as σ_3 increases. The final version of the Coulomb equation is:

$$\tau_f = c + \sigma_n \tan(\phi), \qquad \textbf{(2.30)}$$

whereby τ_f is the shear stress along the shear plane at failure, c is the cohesion, σ_n is the normal stress acting on the shear plane, and ϕ is the friction angle of the shear plane. For decades, Equation (2.30) has been referred to as the Mohr-Coulomb failure criterion, to honor their contributions to the method. Mohr-Coulomb failure criterion is applied in rock mechanics to predict shear failure of rock joints, intact rock and rock masses. The assumption of this criterion is that failure occurs along a plane with no dilation.

This method's simplicity is the reason for its wide usage in rock mechanics. It is easy to use, to comprehend, and consist of a simple mathematical expression based on few parameters. When all principal stresses are compressive, experiments have shown that this criterion does it reasonably well to predict intact rock failure, when the uniaxial compressive strength (σ_c) is much greater than the uniaxial tensile strength (σ_t), e.g. $\frac{\sigma_c}{\sigma_t} > 10$. (ISRM, 2014)

However, according to ISRM (2014), there are several factors to consider before using Mohr-Coulomb as failure criteria. First of all, in numerical coding this criterion is not as easy to implement because of the hexagonal shape in the π-plane in where smooth shapes like a circle is preferred. The π-plane is the plane of the deviatoric stresses in the space with the principal stresses as basis vectors. For materials where the deviatoric stresses are the only source to failure the π-plane is a useful tool to analyse its failure mode. If not, the failure mode must be presented in three dimensions. Second of all, if the normal stress is tensile the assumption of an inner friction is meaningless and a limit of $\sigma_n = \sigma_t$ is normally applied. This is called a tension cut-off. Consequently, it is necessary to also measure values for the tensile strength. Third of all, the failure mode must be shear if this method should be used. Fourth of all, the relation between normal and shear stress obtained during triaxial testing normally gives a non-linear behavior opposed to the linear Mohr-Coulomb criterion. To finish, it does not consider the effects of the intermediate principal stress.

Brady and Brown (2012) states that the Coulomb criterion can provide a good representation of residual strength conditions of rock mass. This implies that this method can be used in cases under which the rock mass already shows high degree of fracturing of no preferred orientation which leads to an isotropic strength reduction. This is relevant in cases such as weakness zones.

Hoek-Brown

According to Hoek and Brown (2019), it is evident that failure in brittle materials such as rock or glass originates from micro-cracks or flaws in the intact material. In rock, these flaws are consisting of grain boundaries, or inter-granular cracks and tensile cracks, that propagate from their tips when frictional sliding occurs along the flaw. A parabolic failure mode was proposed by Griffith (1921) based on these observations. Griffith assumed that tensile failure in brittle materials initiates at the tips of defects. An empirical failure criterion was developed by Hoek and Brown (1980a, 1980b) inspired by Griffith's work. The reason behind this was the insights attained using Mohr envelopes. For rocks these envelopes tended to behave in a non-linear fashion. By curve fitting results from triaxial tests on brittle rocks, the non-linear behavior proposed by Griffith was matched with promising results. Following parabolic failure criteria for elastically isotropic intact rock were suggested:

$$\sigma_1 = \sigma_3 + \sigma_{ci}\left(m_i \frac{\sigma_3}{\sigma_{ci}} + 1\right)^{1/2}, \tag{2.31}$$

where σ_1 and σ_3 are the major and minor principal stresses respectively, σ_{ci} is the uniaxial compression strength for intact rock, and m_i is a material constant of the intact rock. The concept of effective stress was later included in Eq. (27), replacing the total stresses proposed in the first version. The reason for this was that experiments indicated that the failure mode was significantly affected by the degree of water saturation.

It was proven by Zou et al.(2008, 2015) that it was possible to derive the Hoek-Brown criteria theoretically from an analysis on fracture propagation:

$$\sigma_1 = \sigma_3 + \sigma_{ci}\left(\left(\frac{\mu}{\kappa}\frac{\sigma_{ci}}{|\sigma_t|}\right)\frac{\sigma_3}{\sigma_{ci}} + 1\right)^{1/2}, \tag{2.32}$$

where $\mu = \tan\phi$, ϕ is the crack surface friction angle, κ is a coefficient used for mixed mode fracture, and $|\sigma_t|$ is the absolute value of the tensile strength. Thus, by comparison of Equation (2.31) and Equation (2.32) it is evident that $m_i = \mu\sigma_{ci}/(\kappa|\sigma_t|)$. Ergo, m_i has physical meaning.

ISRM (2014) gives the following advantages of the Hoek-Brown criterion: (a) in accordance with experimental data over a range of confining pressures the shape of the criterion is non-linear, (b) it is developed on a wide range of experimental data based on a wide range of intact rock types, (c) it provides a simple empirical method to predict rock mass properties, and (d) three decades of experience, in which this criterion has been used extensively on a variety of rock engineering projects, has highlighted its uses and limitations.

The main flaw of the criterion is the neglection of σ_2, which does enhance the strength of the rock. Again, this leads to misinterpretation of the transition between brittle and ductile behavior(e.g. Mogi, 1967). Furthermore, it does not necessarily capture the right strength of tension. However, there are made a version with a tensile strength cut off, where the cut off is based on results from lab tests on tensile strength (Hoek & Brown, 2019).

Drucker-Prager

Originally, the Drucker-Prager failure criterion was established as a generalization of the Mohr-Coulomb criterion for soils (Drucker & Prager, 1952). It is a 3-dimensional failure criterion which implies that both the deviatoric stress tensor and the volumetric stress tensor is believed to control the failure mode of the material. The criterion is given as follows:

$$\sqrt{J_2{}'} = \lambda I_1' + \kappa \tag{2.33}$$

where λ and κ are material constants, $J_2{}'$ is the second invariant of the effective deviatoric stress tensor and I_1' is the first invariant of the effective stress tensor. They are defined as:

$$I_1' = \sigma_1' + \sigma_2' + \sigma_3', \tag{2.34}$$

$$J_2' = \frac{1}{6}[(\sigma_1' - \sigma_2')^2 + (\sigma_2' - \sigma_3')^2 + (\sigma_3' - \sigma_1')^2], \tag{2.35}$$

where λ and κ can be determined from triaxial tests after plotting the results in the space of I_1' and J_2'. Alternatively, internal friction angle and cohesion intercept found in triaxial tests can be used:

$$\lambda = \frac{2\sin\phi}{\sqrt{3}(3 - \sin\phi)} \tag{2.36}$$

$$\kappa = \frac{6c\cos\phi}{\sqrt{3}(3 - \sin\phi)}, \tag{2.37}$$

in where ϕ is the materials internal friction angle and c is the materials cohesion. Vermeer and De Borst (1984) stated that the Drucker-Prager condition makes a poorly fit for material with higher friction angles such as sand, concrete, and rock. However, this approximation is useful for clays with low angle of friction.

ISRM (2014) made a review of failure criteria for intact rock, where it is shown that the Drucker-Prager criterion overestimate the strength of the material. This is because the strength criterion shows an increase in error with increasing difference between σ_2 and σ_3. (ISRM, 2014)

Friction of Rocks

Byerlee (1978) found in an analysis shear behavior of joints and faults, that the friction is dependent on the surface friction of the rock mass in low stress areas. A low stress area is defined as normal stresses ranging between 0 to 50 Bars and with shear stresses ranging between 0 to 60 bars. Also, it was found that there was no strong correlation between friction and rock type. The weak correlation between friction and rock type was attributed to the variation of friction due to variation of surface roughness. An equation describing friction of rocks at low stresses was suggested by (Barton, 1976):

$$\tau = \sigma_n \tan\left[\mathrm{JRC}\log_{10}\left(\frac{\mathrm{JCS}}{\sigma_n}\right) + \phi_b\right], \tag{2.38}$$

where τ is the shear stress of the failure plane, σ_n is the normal stress of the failure plane, JRC is the undulation number, JCS is the strength of the failure plane taken with Schmidt hammer, and ϕ_b is the basic friction angle. The link to Mohr-Coulomb is evident in where no cohesion is assumed. The equation states that slipping will occur if the shear force exceeds the criteria given in Equation (2.33).

For intermediate stress areas ranging between 50 to 1000 bars and high stress areas where the stress is over 1000 bars, there is no correlation between friction and initial surface friction. Furthermore, it seems that the friction behavior are independent of rock type. Byerlee (1978) proposed two equations describing friction: Normal stresses up to 2 kb and above 2 kb respectively.

$$\tau = 0.85\sigma_n \tag{2.39}$$

$$\tau = 0.5 + 0.6\sigma_n \tag{2.40}$$

2.2.9 Anisotropy and deformability of rock mass

Hudson and Harrison (1997) states that thorough examination of the mechanical properties of rocks have proven most rocks to behave anisotropic. Thus, it is argued that the assumption of rock mass behaving continuous, homogeneous, isotropic, and linear-elastic (CHILE) does not hold in most cases.

Features causing anisotropy

Rock anisotropy is attributed to both primary and secondary structures that can be traced back to the formation environment of the intact rock and the rock mass.

The term **primary structures** refer to the micro geological features which are formed during the formation of a specific rock. Features like rock fabric, texture, schistosity, grain size, and fissility affect the degree of anisotropy. Besides, Ullemeyer et al. (2006) states that the texture and fabric of the principal rock-forming minerals in general are the most important factors. According to Bagheripour et al. (2011) the nature of anisotropic intact rock can be divided in three categories. First of all, most foliated metamorphic rocks, such as gneisses, phyllites, slates and schists, have a natural orientation in their flat and/or elongated minerals or a banding phenomenon. This leads to anisotropy in mechanical properties. Second of all, stratified sedimentary rocks, like sandstone or siltstone, often display anisotropic behavior. This is due to the sedimentation processes of the different strata, or different minerals with various grain sizes. Finally, the anisotropy of igneous intact rocks seldom occurs due to the minerals being oriented more chaotic and sporadic. However some extrusions do not behave according to this rule of thumb (e.g. Matsukura et al., 2002; Wahlstrom, 1973).

The term **secondary structures** refer to the discontinuities of the rock mass. These discontinuities are associated with three distinct issues (Bobet et al., 2009): (a) The scale effect illustrated in Figure 23 indicating that the quality of a rock mass is also dependable of the tunnel size, (b) alteration of stress and strain paths due to discontinuities, and (c) relative motions of rock blocks whereby discontinuities limits the elastic behavior of the rock material. According to Hoek and Marinos (2000), anisotropy in strength is a result of either the filling between blocks or the contact surface among blocks being weaker compared to the strength of the intact blocks.

Heavily fractured rock mass may behave isotropic since the discontinuity planes continuity is disrupted. Bray (1967) demonstrated that if a rock mass contains ten or more joint sets, it is valid to treat it as a homogenous and isotropic material. Furthermore, Hoek and Brown (1980b) argued that homogeneity is a characteristic reliant on the sample size. This is also illustrated in Figure 23, assuming that the tunnel is representing a sample.

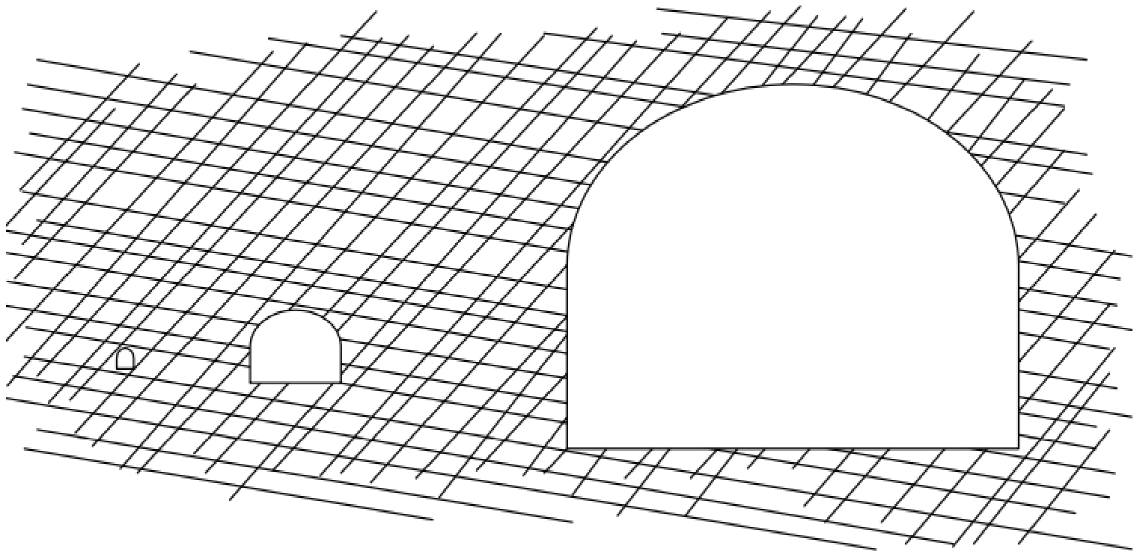


Figure 23: This figure presents the scaling effect as a consequence of different tunnel sizes, as illustrated by Edelbro (2003).

Edelbro (2003) had these effects in mind when she defined continuous rock mass as either being intact or being closely jointed in her literature review on quality descriptions of hard rock. This definition is illustrated in Figure 24. In her definition, discontinuous rock mass is highly affected by the direction of the discontinuities, and failure gets direction dependent.

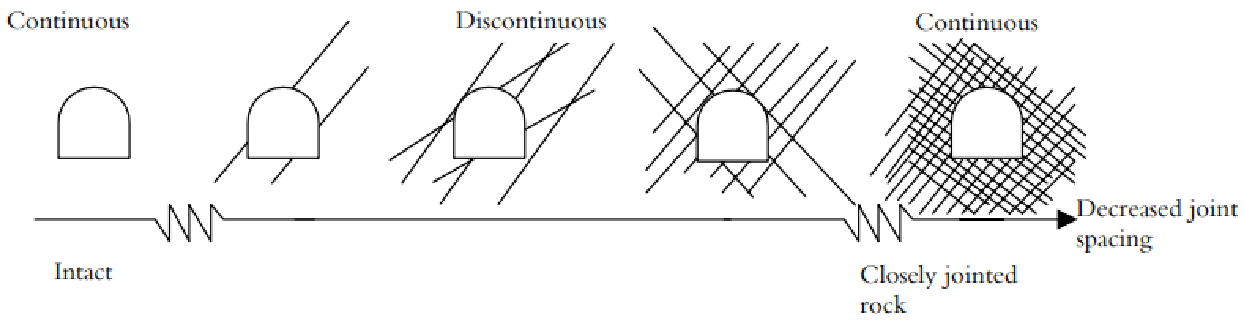
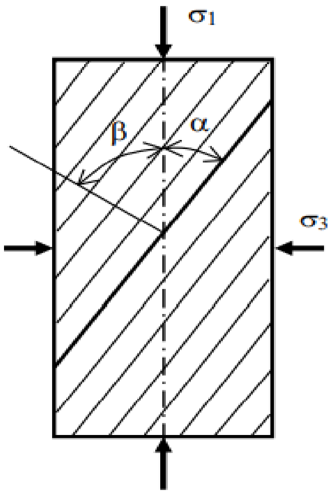


Figure 24: Illustration of the difference between discontinuous and continuous rock mass. The size of the tunnel is held constant . Edelbro (2003) has made the illustration.

Triaxial tests has been conducted on rock samples with a distinct weakness plane (i.e. Brady & Brown, 1993; Donath, 1972). Their result is presented in Figure 25. The tests showed that the anisotropic strength character is dependent on the angle between the weakness plane and the direction of the applied load. These tests were conducted by varying the load angle from being parallel with the weakness plane to be orthogonal to the plane. Angle of zero is the case when the load is parallel. For each change of angle several tests with different confining pressure were conducted. They found that the lowest peak strength was attained with an angle between 30 and 45 degrees. This effect was not affected severely by change in confinement pressure.



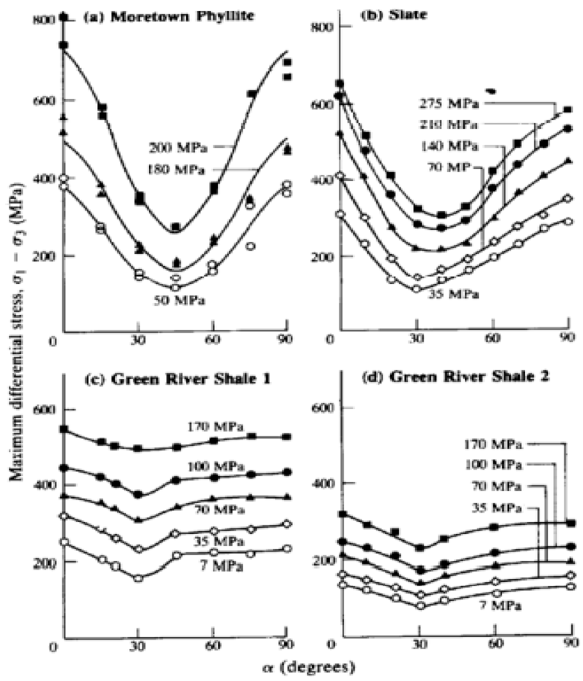


Figure 25: Variation of the peak deviatoric stress with the angle of inclination of the major principal stress to the plane of weakness, given by different confining pressures for (a) a phyllite(Donath, 1972), (b-d) a slate and two shales (Brady & Brown, 1993).

Li (2018) discuss that the weakening effect of a discontinuity is additive. This effect is illustrated in Figure 26. It is assumed a hard rock mass with 4 distinct joints with of weak strength that exists in the same plane. A triaxial compression test is done with varying load angle. If there were only one discontinuity, the behavior would be similar to the cases in Figure 25. With 4 weakness planes, the result is a combined effect where the rock mass shows a more constantly weak character. As the specimen rotates there is always one weakness zone which will dominate the failure mode.

Hudson and Harrison (1997) argues that the stiffness also is affected by the characteristics, angels, and frequencies of discontinuities. They found a following equation for an idealized rock mass with one discontinuity set normal to the direction of the applied force with negligible thickness and one specific frequency:

$$E_{MASS} = \sigma/\epsilon = 1/[(1/E) + (\lambda/E_D)], \quad \textbf{(2.41)}$$

whereby E_{MASS} is the Young's modulus of the rock mass, σ is the stress from uniaxial load, ϵ is the total strain, E is the Young's modulus of intact rock, λ is the frequency of the discontinuity set, and E_D is the Young's modulus of the discontinuity set. This idealized case is shown graphically in Figure 27.

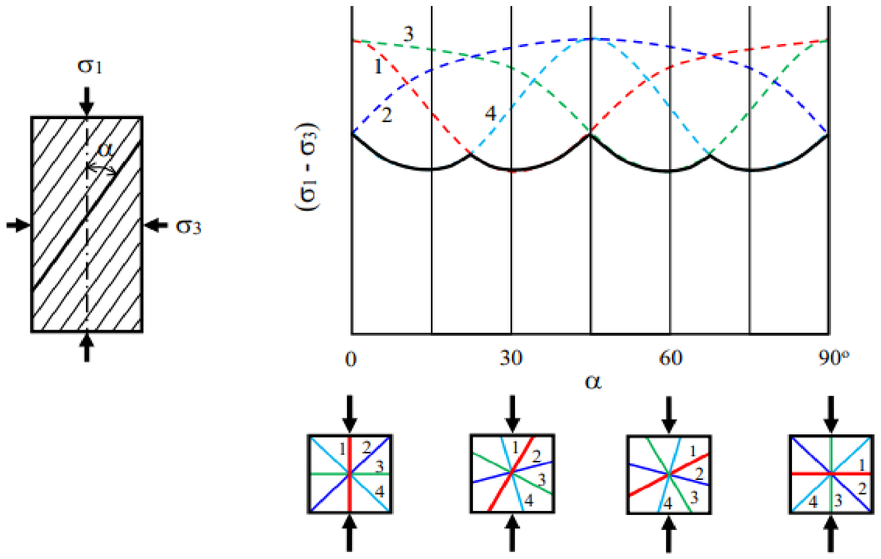


Figure 26: Illustration of joint sets influence on the strength of the rock mass, figure taken from Li (2018).

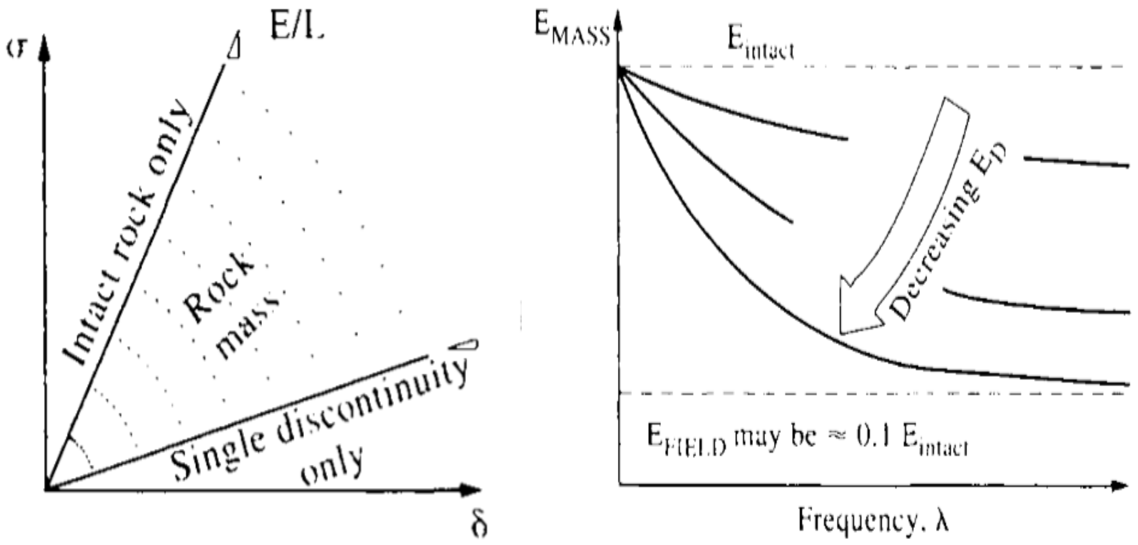


Figure 27: Variation of in situ rock deformability as a function of the frequency of one discontinuity set (idealized) taken from Hudson and Harrison (1997).

Goodman (1989) did the same argument just for shear modulus and showed that:

$$G_{MASS} = \tau/\gamma = 1/[(G) + (\lambda/G_D)], \qquad \textbf{(2.42)}$$

whereby G_{MASS} is the shear modulus, τ is the shear stress from uniaxial shear, γ is the total shear strain, G is the intact shear modulus, λ is the frequency of the discontinuity set, and G_D is the shear modulus of the discontinuity.

According to (Gercek, 2007) the Poisson's ratio is also a weighted sum of the stiffness behavior of the discontinuities and the intact rock. Important factors describing the stiffness of joints are the normal stiffness (k_e), the shear stiffness (k_s), and joint spacing. Several studies have shown that the value of Poisson's ratio for most cases are lower than the intact value. In fact, in some cases, unusually high values were obtained ($\nu > 0.5$ indicating the anisotropy induced by the joints

According to Hudson and Harrison (1997) is the mathematics associated with further extensions to account for discontinuity geometry, loading angle, and number of discontinuity rather complex. However, a complete solution has been provided by Wei (1988). Wei's model can in some extent also consider weakening effects due to impersistent discontinuities.

Deformation prediction of rock mass

It is evident that to capture the true deformation behavior of is a rather cumbersome task, and in engineering projects it will be too time-consuming and expensive to accomplish (e.g. Bieniawski, 1974; Palmström & Stille, 2010). This is not only due to the complexity of the mathematical representation, but it is also due to the heterogeneity of rock mass and its great variability over relative short distances. Extrapolation of input-data from a rock mass measured in one site onto the surrounding rock mass cannot readily be done without sufficient information about similarity of the sites under consideration. Thus, it has been a long tradition for developing methods to predict rock mass deformation based on observations in the field. This approach has especially proven to be effective in the pre-excavation phase in where little is known about the rock mass at hand.

Rock mass classification

Krauland et al. (1989) categories four different approaches to get the rock mass strength: (a) mathematical modelling, (b) rock mass classification, (c) large-scale testing, and (d) back-calculation.

The mathematical models (a) capture the strength through describing both the rock substance and the properties of the discontinuities. The modelling can either be done through simulation of the discontinuities as discrete elements of the rock mass or be regarded as the rock mass as a composite material. The properties of this material are given by the properties of both the intact rock and weaknesses and how these elements are distributed. The mathematical models all require information of many parameters and is all based on several assumptions simplifying the rules of the rock mass behavior.

Classification systems (b) can be divided into two sub-groups based on the purpose of the classification: Stability classification and strength classification.

The goal of stability classification systems is to define the ratio between load and bearing capacity. This ratio is used to classify the response in which the underground structure has on it, and then give an estimation on necessary use of support. The size of the stress field and the influence of the excavations geometry is included within these types of classification systems. The usefulness of such systems is linked to systems which shows little variation in the geometry of the excavation, for instance in tunnelling.

The purpose of strength classification systems is to map the strength of the rock mass only. The classified rock mass strength is used in combination with information about load in a stability evaluation. This type of approach has been useful in projects where the geometry of the excavation, the size of the excavation, and the stress state show large variations, such as in civil engineering applications. Determination of rock mass strength through classification systems are based on a limited number of parameters that are believed to control the rock mass strength. All classification systems are empirically based and are usually developed with a data base of case studies.

In large-scale testing (c), the strength of representative rock mass samples is determined. This includes the interaction between intact rock and the discontinuities, hence providing for thorough description of the properties of the rock mass. High costs and the need for representative test objects makes this an expensive procedure.

Back-calculation (d) is conducted to assess experience on the properties of the rock mass from existing engineering structures. In particular, the maximum bearing capacity of the rock mass can be decided if failure already has occurred.

Edelbro (2003) states that empirically derived failure criteria for rock masses, often used in conjunction with rock mass classification of strength, should be added to Krauland's list. Classification systems is thereby used to assert rock mass properties. Empirical failure criteria for rock mass are mainly based on triaxial testing on small rock samples, and few is verified against test data for rock masses. Such classification systems also provide for an estimation of the rock mass' deformation parameters. When both the deformation response and the failure mode is established, it is possible to predict deformation response through use of numerical modelling. This combined method is the only method which will be further investigated in this chapter.

Rock mass classification systems

Cai and Kaiser (2006) defines rock mass classification systems as systems intended to classify and characterize the rock masses, used as a basis for estimating deformation and strength properties, supply quantitative data for support estimation, and to give a platform for communication between exploration, design, and construction groups. According to Palmstrom and Broch (2006) the difficulties in technically describing rock masses and ground conditions motivated development of an empirical approach of rock mass classification systems at an early stage of geological engineering and rock mechanics. The first system in use was developed in USA by Terzaghi (1946) which applies conservative estimation for loads on the support totally based on the use of steel.

According to Palmström and Stille (2010) rock mass classification systems can be considerably useful in the initial stages of a project when little is known about the details of the rock mass. The classification systems can be divided into two distinct groups: (a) general classification systems, or (b) classification systems for specific applications. The classification systems consider factors believed to affect the stability. Thus, the parameters are often related to the discontinuities such as the number of joint sets, roughness, filling and alteration of joints, joint distance, groundwater conditions, and sometimes the strength of the intact rock and the magnitude of stress.

Hudson and Harrison (1997) proclaim that classification schemes essentially are a compromise between the use of complete theory and ignoring the rock properties entirely – pointing to the fact that all classification systems are based on just a few of the key rock mass parameters. Furthermore, they state that a single number primarily based on field observation cannot describe the rock mass anisotropy and time dependent behavior in a satisfactory manner. Since classification of rock mass is an indirect method, it does not predict the mechanical properties of the rock mass or failure mode directly. In fact, the result of such systems gives an estimate of the stability given in subjective terms such as bad, fairly bad, acceptable, etc. The value from some of the systems can be used to assess the rock mass strength through use of failure criteria or to predict necessary use of rock support.

According to Edelbro (2003) there are three systems that are most often used to predict strength and deformation properties of rock mass: (a) NGI's Q-system developed by Barton et al. (1974), (b) the Rock Mass Rating (RMR) first proposed by Bieniawski (1973), and (c) the Geological Strength Index introduced by Hoek et al. (1995). The Q-system and the RMR can also be used to get predictions for necessary use of rock support.

The Q-system

The original Q-system was based on an analysis of 212 case records where most of the material was based on tunnelling cases from Sweden and Norway. 180 of the 212 case records were from supported excavations, and the rest were from cases permanently unsupported. The entire scale of projects was investigated ranging from unsupported 1.2 m wide pilot tunnels to unsupported 100 m wide mine-caverns. The excavation depths ranged from 5 to 2500 m where the most common depths were between 50 and 250m. The last major update was conducted by Grimstad et al. (2002) and was then based on around 1050 base records. (Edelbro, 2003)

In application of this classification system, a given site should be divided into several geological structural units in such a way that each type of rock mass is represented by a separate geological structural unit. This separation process is mainly based on experiences in the field. The Q-system uses the following six parameters:

(a) Rock Quality Designation (RQD)*,*
(b) Number of joint sets ($\mathrm{J_n}$)*,*
(c) Joint roughness number ($\mathrm{J_r}$) *of least favourable discontinuity or joint set,*
(d) Joint alteration number ($\mathrm{J_a}$) *of least favourable discontinuity or joint set,*
(e) Joint water and pressure reduction factor ($\mathrm{J_w}$)*,*
(f) Stress reduction factor (SRF) *for faulting, strength/stress ratios in hard massive rocks, and squeezing and swelling rock.*

Following empirical equation was defined (Barton et al., 1974):

$$Q = \left[\frac{\mathrm{RQD}}{\mathrm{J_n}}\right] * \left[\frac{\mathrm{J_r}}{\mathrm{J_a}}\right] * \left[\frac{\mathrm{J_w}}{\mathrm{SRF}}\right], \tag{2.43}$$

in where the ratio $[\mathrm{RQD}/\mathrm{J_n}]$ is the relative block size, $[\mathrm{J_r}/\mathrm{J_a}]$ is the relative frictional strength of the least favourable joint set or filled discontinuity, and $[\mathrm{J_w}/\mathrm{SRF}]$ is the active stress ratio. The range of Q-values is between 0.001 and 1000. The details of the parameters and how much each parameter is weighted can be found in the Q-system handbook (NGI, 2015). The parameters are found either through use of either borehole data or underground mapping. Also, the Q-system can be used to estimate Young's modulus for rock mass, but is not recommended by (Palmstrom & Broch, 2006)

Rock Mass Rating (RMR)

This rating system was originally based on Bieniawski's experiences in shallow tunnels in sedimentary rock (Goel & Singh, 2011) and was based on 49 unpublished case histories. It underwent several stages of development until 1989 with addition with over 64 more case studies. Also, Aksoy (2008) states that it has been used in over 350 applications in underground openings, tunnels, underground mines, and open-pit slope designs. Edelbro (2003) stresses that the RMR-system is calibrated using experiences from coalmines, civil engineering excavations and tunnels at shallow depths.

In application of this classification system, a given site should be divided into several geological structural units in such a way that each type of rock mass is represented by a separate geological structural unit. This separation process is mainly based on experiences in the field. The following six parameters are attained for each structural unit:

(a) Uniaxial compressive strength of intact rock,
(b) Rock quality designation (RQD) *(Deere & Miller, 1966),*
(c) Joint or discontinuity spacing,
(d) Joint condition,
(e) Groundwater condition,
(f) Joint orientation adjustment.

Each of the parameter's contribution is added to give the resulting RMR-value ranging from 0 to 100. The details of the parameters and how much each parameter is weighted can be found in the book: "Engineering rock mass classification" (Bieniawski, 1989). The parameters are found either through use of either borehole data or underground mapping. In addition, RMR can also be used to estimate Young's modulus for rock mass and UCS for rock mass (Basarir, 2008), in where the use is recommended for preliminary studies only due to uncertainty of the estimation.

Geological Strength Index (GSI)

Hoek et al. (1995) introduced the Geological Strength Index to complement their generalized rock failure criterion (discussed later). GSI is used to estimate the reduction of strength of the rock mass for different geological conditions for undisturbed rock. In this context the term undisturbed refers to rock mass that has taken little to no damage because of the excavation process. GSI was made to focus on the two principal factors believed to have the most significant influence on the mechanical properties of a rock mass, namely the blockiness and the condition of the joints. This system was developed to deal with rock masses comprised of interlocking angular blocks in which the failure process is dominated by block sliding and rotation without great deal of rock failure.

According to Hoek and Brown (2019), the latest major revision of this system was executed by (Hoek et al., 2002) where it became possible to adjust for blast damage. It is possible to decide the GSI through measurements of Q-values or RMR-values. However, Hoek and Brown (1997) recommends to get GSI-values through usage of their own classification system because of flaws in the equations describing their relation. Details of the GSI-classification is given in Hoek and Brown (2019) in where it is stressed that several adaptations have been made in situations where the rock mass is heterogenous and tectonically deformed. The last adaptation was done by (Marinos & Carter, 2018). GSI can also be used to estimate Young's modulus for rock mass (e.g.Hoek & Diederichs, 2006) and UCS for rock mass (Hoek et al., 2002). Also, in this case is it recommended for preliminary studies only. GSI ranges from 1 to 100.

Høien et al. (2019) states that the equations proposed by Hoek and Diederichs (2006) and Hoek et al. (2002) are probably the most used equations to predict rock mass modulus and strength. The reason is that they are well documented and is easy to implement in computer software.

Failure criteria of rock mass: Generalized Hoek-Brown criterion

Alongside with Mohr-Coulomb, the Generalized Hoek-brown criterion is the most used criterion for rock engineering purposes (Edelbro, 2003).

The Generalized Hoek-Brown failure criterion (Hoek et al., 1995) for jointed rock masses is defined by:

$$\sigma_1' = \sigma_3' + \sigma_{ci}\left(m_b \frac{\sigma_3{}'}{\sigma_{ci}} + s\right)^a \tag{2.44}$$

where m_b, s, and a are the rock mass material constants given by:

$$m_b = m_i \exp[(\mathrm{GSI} - 100)/(28 - 14D)]$$

$$s = \exp[(\mathrm{GSI} - 100)/(9 - 3D)] \tag{2.45}$$

$$a = 1/2 + 1/6(e^{-\mathrm{GSI}/15} - e^{-20/3})$$

Also, m is equivalent to friction strength of rock, s is a measure of degree of fracture of rock, thus analogous to cohesion. D is a factor dependent on the degree of disturbance to which the rock mass has been subjected to blast damage and stress relaxation.

Hoek and Marinos (2000) argued that the generalized Hoek-Brown also could be used for weak rock if it showed isotropic behavior, and which they stated is a criterion for any other of the published criteria that can be used for the purpose rock mass failure. In other words, while the behavior of the rock mass is controlled by movement and rotation of rock elements separated by intersecting structural features such as bedding planes and joints, there are no preferred failure direction.

Equation (2.44) was developed for estimating rock mass comprising interlocking angular blocks in which the failure process is dominated by block sliding and rotation without a great deal of intact rock failure.

According to (ISRM, 2014), there have also been attempts to generalize this criterion even further to also include the intermediate stress. However, this was only for intact rock specimens, and the ISRM concluded that due to either to severe overestimations and underestimations of the rock strength indicated that more empirical data and testing was necessary before they could recommend it. There have also been attempts to make a modified version for anisotropic intact rock (e.g. Saroglou & Tsiambaos, 2008), but not for rock mass.

Critical strain

Sakurai (1981) introduced the concept of critical strain and is the strain of the rock mass at the yield load:

$$\epsilon_0 = \sigma_c / E, \tag{2.46}$$

where σ_c is the UCS, and E is the Young's modulus. If the rock is linear elastic, the critical strain is the strain at peak-load. According to (Sakurai, 1984), the stability of tunnels can be predicted based on the strain of the rock mass close to the tunnel. It was further stated that the critical strains ranged between 0.1% to 1% for rocks and between 1% and 5% for soils. Sakurai (1984) suggested that stability problems in tunnels occurred when the critical strain exceeded approximately 1%.

It has been shown that it is possible to predict strain by the rock mass strength and in/situ stresses (e.g.Hoek, 1999; Hoek & Marinos, 2000). The relation between unsupported rock mass strength and strain is presented in Figure 28: The relation between unsupported rock mass strength and strain. (Hoek & Marinos, 2000). It is based on a Monte Carlo simulation in where: (a) tunnel diameters ranged between 4 and 16m, (b) m_i between 5 and 12, (c) overburden between 80 to 800m, (d) GSI between 10 to 30, and (e) σ_c between 1 to 30 MPa. The model was able to confirm Sakurai's statement that minor stability issues tended to occur for critical strain passed 1%. Hoek (2001) made a chart in where field observation of σ_c of rock specimens were plotted against measured strain. The chart is given in Figure 29. The red dots mark the situations where it was experienced issues with instability. It can be seen from this chart that there was only one occurrence in where it was experienced instability with strain below 1%. The highest strain with no instability problems was 4%.

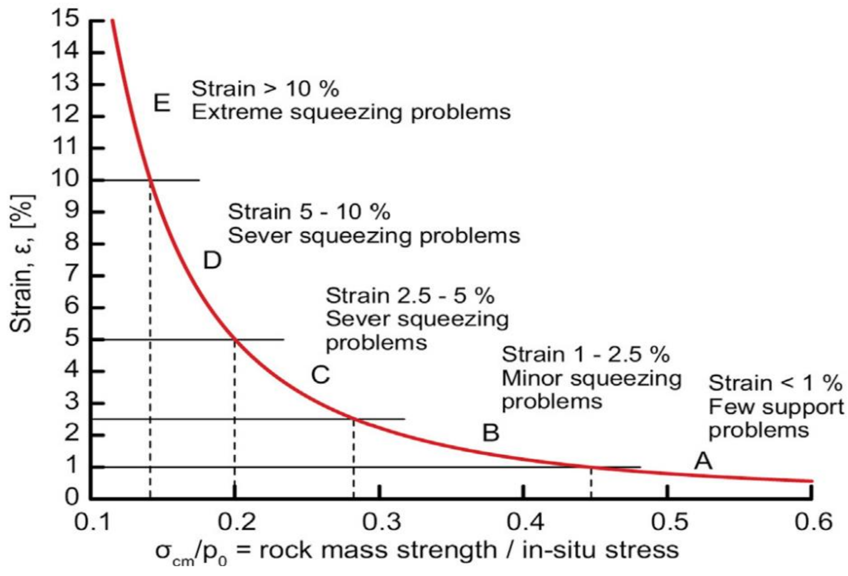


Figure 28: The relation between unsupported rock mass strength and strain. (Hoek & Marinos, 2000)

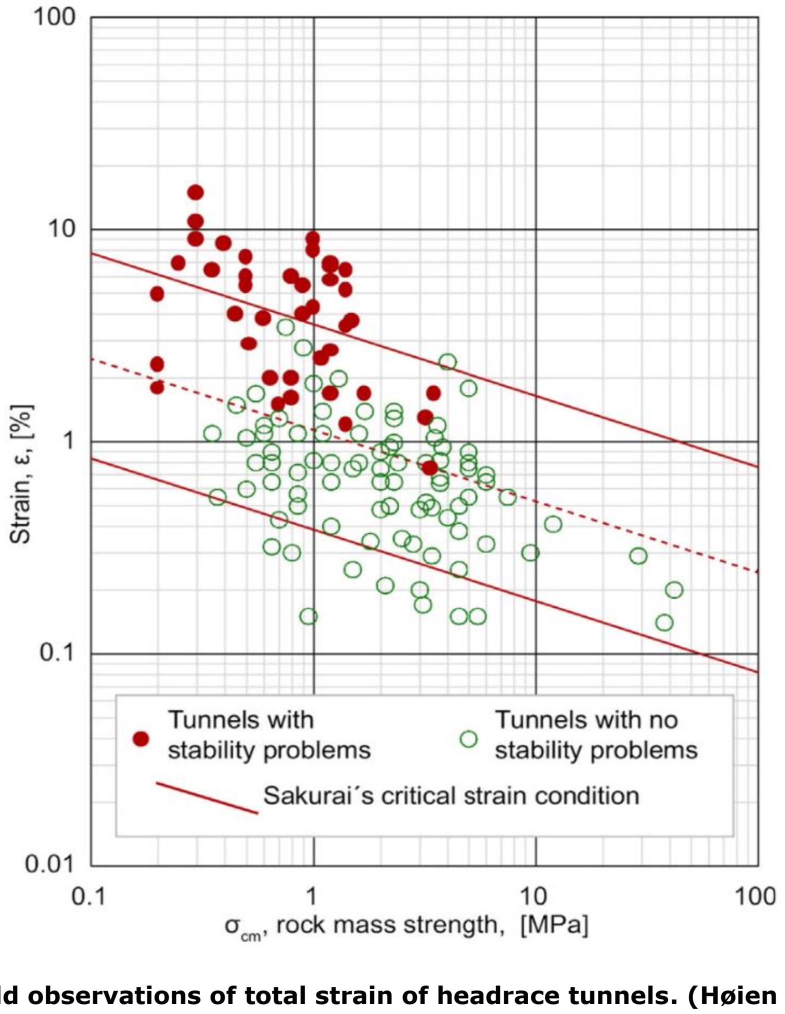


Figure 29: Field observations of total strain of headrace tunnels. (Høien et al., 2019)

2.2.10 The stress distribution

The term stress distribution is referring to the distribution of inner forces that consists within a continuous material, such as a rock mass. These inner forces are a result of several factors such as gravitation, friction between grains, and temperature. Conceptually is the stress distribution a summarization of the stress state in each point of a given continuous material. The convention is that the stress is given with a positive value under compression. Therefore, must the strain be positive when the material contracts.

The stress state of rock mass is comprised from two sources: in situ stresses and induced stresses (Bøgeberg et al., 2021), as seen in Figure 30. In situ stress state is defined as the original stress state in a rock mass before excavation or other disturbances. There are four main categories of in situ stresses referring to their origin: gravitational-, tectonic-, residual-, and terrestrial stresses. The induced stress state is the stress state after excavation or other disturbances. It is apparent that this altogether leads to a complex of stresses in which it is impractical to grasp the details. To get a decent and detailed picture of the stress distribution (Haimson & Cornet, 2003), it is necessary with high quality in-situ stress measurements such as the doorstopper method or hydraulic fracturing. The details of how stress measurements are executed is beyond the scope of this project.

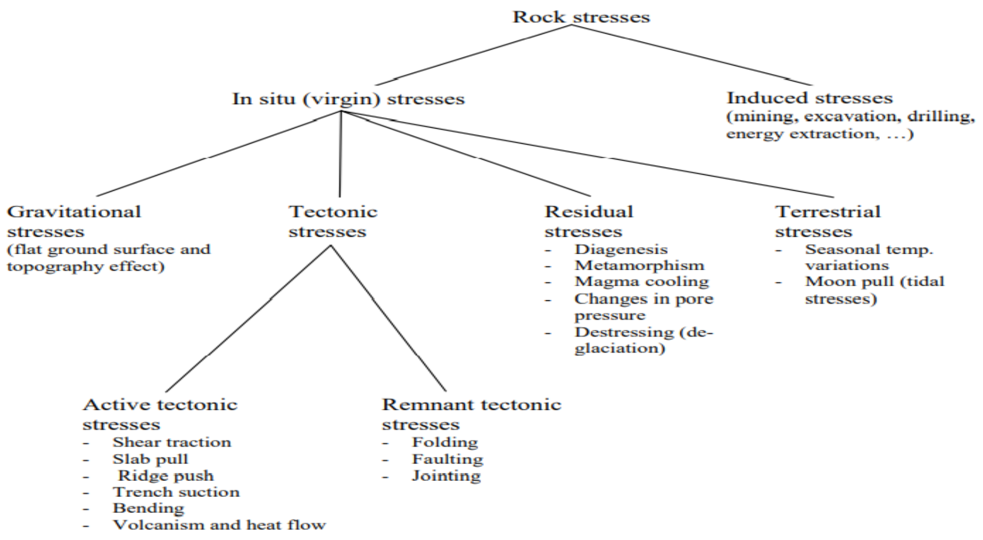

Rock stresses
In situ (virgin) stresses
Induced stresses (mining, excavation, drilling, energy extraction, …)
Gravitational stresses (flat ground surface and topography effect)
Tectonic stresses
Residual stresses
- Diagenesis
- Metamorphism
- Magma cooling
- Changes in pore pressure
- Destressing (de-glaciation)
Terrestrial stresses
- Seasonal temp. variations
- Moon pull (tidal stresses)
Active tectonic stresses
- Shear traction
- Slab pull
- Ridge push
- Trench suction
- Bending
- Volcanism and heat flow
Remnant tectonic stresses
- Folding
- Faulting
- Jointing

Figure 30: Different parameters affecting the stress state(Amadei & Stephansson, 1997).

It is usual to define in-situ rock mass stresses by a vertical stress induced by gravity and a horizontal stress which is given by its ratio compared to the vertical stress. This is given in eq. (41) and (42).

$$\sigma_v = \gamma z \tag{2.47}$$

$$\sigma_h = k\sigma_v \tag{2.48}$$

Terzaghi and Richart Jr (1952) proposed, for a rock mass gravitationally loaded whereby no lateral movement is allowed, the k ratio is independent of depth and is approximated through Poisson's ratio of the rock mass (ν) in the following way:

$$k = \frac{\nu}{1-\nu} \tag{2.49}$$

Numerous measurements has proven this wrong, whereby Hast (1958) was the first to report anomaly high k-values shallow in the crust. Stress measurements from several projects across the globe highlighted the trend of high values of the k-ratio near the surface in which reduces with depth (Brown & Hoek, 1978; Herget, 1988), see Figure 31. In fact, the k-factor contains the effects from all the sources of in-situ stresses mentioned above in addition to Poisson's restraint, major and minor geological structures, rock properties and surface topography.

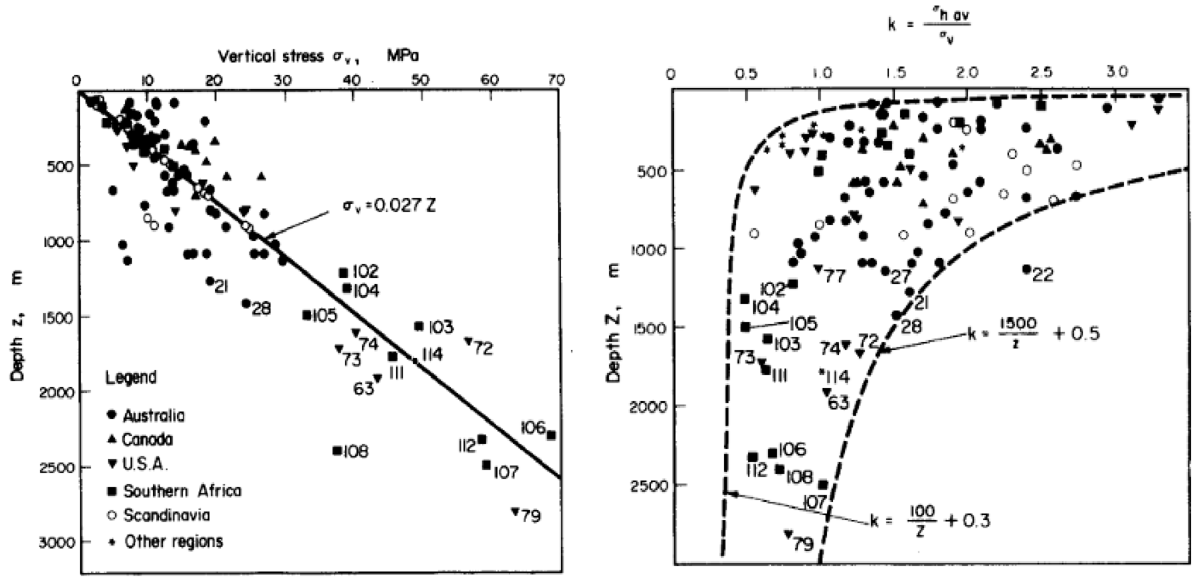


Figure 31: k-values gathered across the world, made by Brown and Hoek (1978).

McCutchen (1982) was one of the first attempting to explain the variability of the k-factor theoretically and showed that it was possible to describe the general behavior of the in-situ stress field through a simple elastic-static analysis. However, McCutchen was unable to satisfactory fit his theory to the work of Hoek and Brown given in Figure 31. Sheorey (1994), inspired by McCutchen, proposed an elastic-static thermal stress model in which reached a more promising fit. In his work it was found that the earth's curvature was the most important factor to create high values for k in the proximity of the surface. Also, he found that Young's modulus has a great influence on this factor. He proposed the following empirical relation:

$$k = 0.25 + 7E\left(0.001 + \frac{1}{H}\right), \tag{2.50}$$

where E is Young's modulus and H is depth beneath the surface. Sheorey stressed that his analysis do not imply that the other factors mentioned above may be ignored. He pointed out that his theory does not explain why vertical stresses can be different from cover pressure. Neither can it explain the occasionally high horizontal stresses, nor the fact that the two horizontal stresses usually are different. Thus, since stress is a complex combination of several factors, he stresses the importance of measurements. Although, the k-values high dependency on Young's modulus indicates that stress measurements done in hard rocks should not necessarily be used in softer rocks. This is shown graphically in Figure 32.

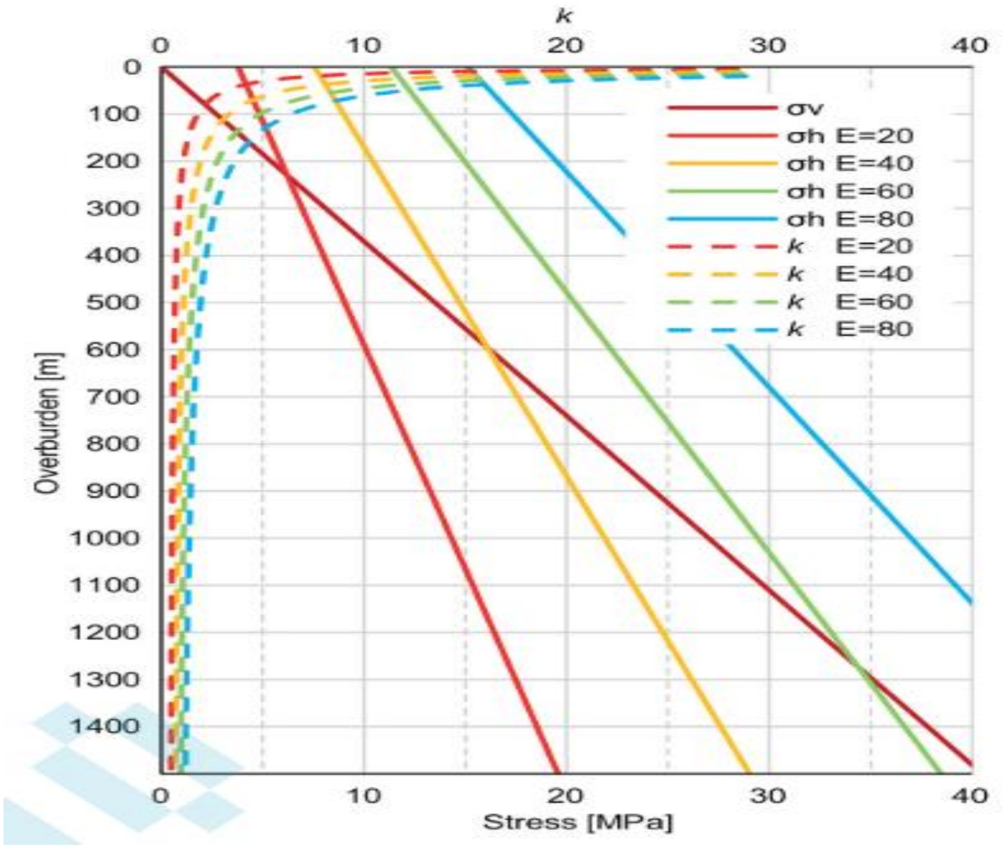


Figure 32: The k-ratio and calculated stress towards depth, made by Høien et al. (2019), based on (Hatheway & Kiersch, 1986; Sheorey, 1994).

Discontinuities is a feature of the rock mass that greatly inflicts the proximate surrounding stress distribution. The alteration of the stress state is a direct consequence of discontinuities low tensile stress which implies low rock mass quality and leads to a concentration of stress in the neighborhood of the zone. An illustration of the alteration is given in Figure 33. (Myrvang, 2001)

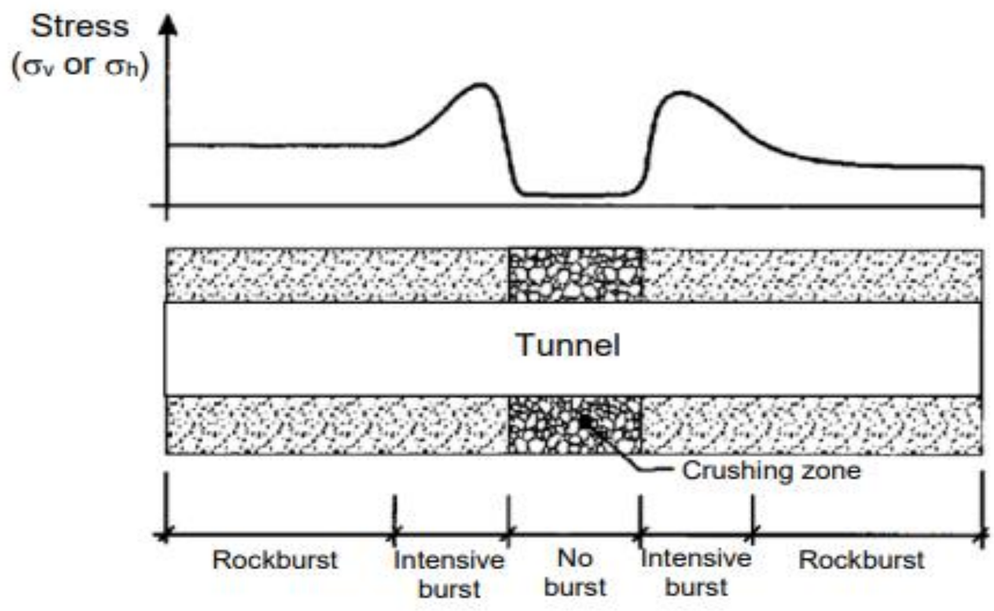


Figure 33: Illustration of how a weakness zone may alter the stress distribution in the proximity of the tunnel, illustration made by Myrvang (2001).

The stress state in the weakness zone is assumed to be hydrostatic. It is a reasonable assumption for very weak rock like in a fault or shear zone, because of such rocks already has reached failure and therefore cannot sustain significant stress differences. Also, this is the case even if the stress distribution in the surrounding competent rock mass is asymmetrical. (Hoek, 1999)

2.2.11 Structure and material behavior of weakness zones

The review of the nature of weakness zone is limited to fault zones in brittle rocks.

Fault zones in brittle rocks

A fault can be defined as a feature where a relative movement between the rock mass on both sides of a discontinuity has occurred. Furthermore, a fault is a brittle deformation of the rocks, and may be a consequence of either compression or extension. (Sæter, 2005). McClay (1987) states that brittle to semi-brittle faults are planar discontinuities along which significant displacement has occurred, and that they generally occur int the upper 10-15 km of the crust. A fault zone is defined as an area with several parallel faults. However, it is also used to refer to a single fault complex that is made up by the fault core and the damage zone, see Figure 34.

Fault material

Gouge, cataclasis, ultra-cataclasis, or a mix of these three is the main constituents of the fault material (Sæter, 2005). Following fault materials are defined:

- *Cataclasite: A type of rock made after lithification of angular fragments in a fine-grained matrix coming from the process of abrasion of a brittle protolith due to shear movement. The fragments show no preferred orientation.*
- *Mylonite: A foliated rock where the original grain size of the rock was reduced due to plastic and semi-plastic deformation.*
- *Breccia: A loose rock of angular fragments in a fine-grained matrix coming from the process of abrasion of a brittle protolith due to shear movement. Normally, the fragments show no preferred orientation. Compared to cataclasite, breccias show less cohesion due to lesser extent of pressurization and bonding of the matrix.*
- *Gouge: A fine-grained mass of clay-like character which is created because of severe crushing of the host rock by deformation in the brittle regime. The grain size is less than 0.1 mm, but there are some rock fragments of greater magnitude prevalent in the matrix. Dry gouge is loose to partly compact. Wet gouge is sticky, implies more cohesion.*

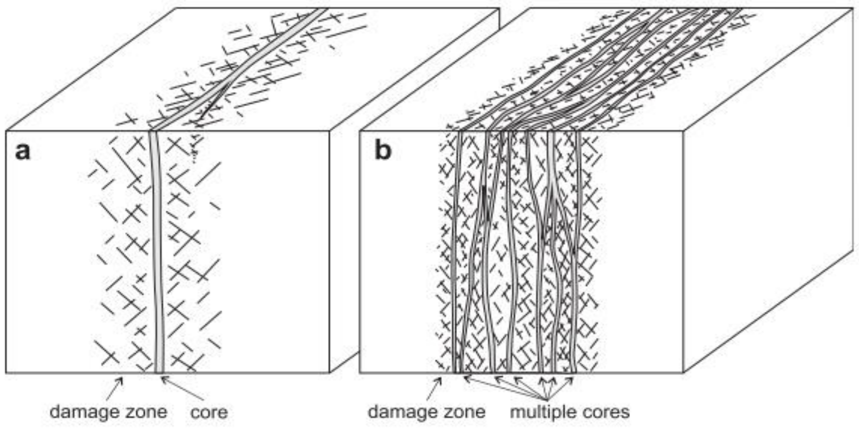
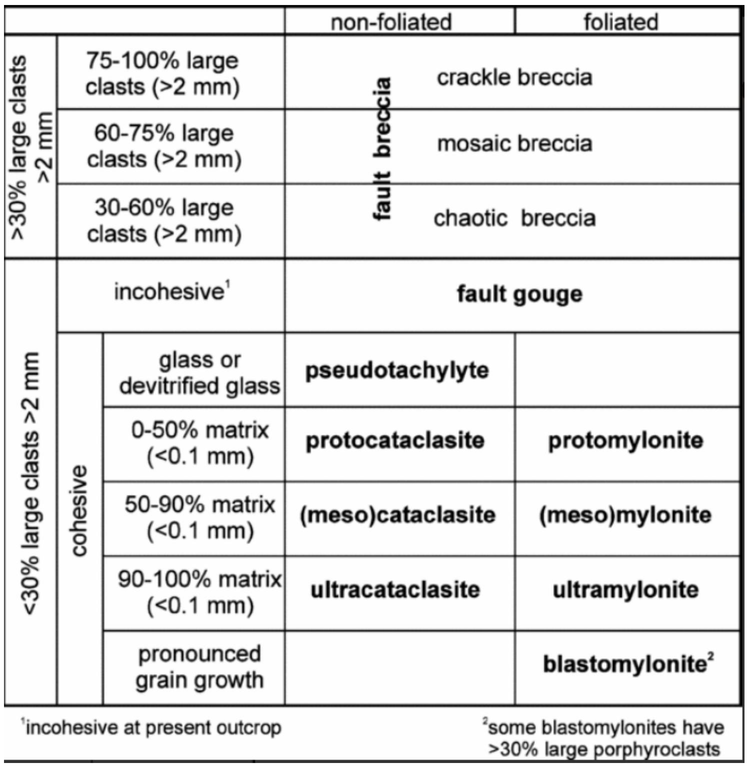


Figure 34: Typical structure of fault zones. (a) Replicates a singular high-strain core surrounded by fractured damage zone and (b) represents a model of multiple high-strain cores in which enclose lenses of fractured protolith. Taken from Faulkner et al. (2010).

Woodcock and Mort (2008) proposed a classification on fault rocks based on grain sizes and extent of orientation. This definition is given in Figure 35. As can be seen, the term ultra refers to a cohesive non-foliated fault rock dominated by fine-grained matrix (90-100%).



				non-foliated	foliated
>30% large clasts >2 mm		75-100% large clasts (>2 mm)	fault breccia	crackle breccia	
		60-75% large clasts (>2 mm)		mosaic breccia	
		30-60% large clasts (>2 mm)		chaotic breccia	
<30% large clasts >2 mm		incohesive[1]		**fault gouge**	
	cohesive	glass or devitrified glass		**pseudotachylyte**	
		0-50% matrix (<0.1 mm)		**protocataclasite**	**protomylonite**
		50-90% matrix (<0.1 mm)		**(meso)cataclasite**	**(meso)mylonite**
		90-100% matrix (<0.1 mm)		**ultracataclasite**	**ultramylonite**
		pronounced grain growth			**blastomylonite**[2]

[1]incohesive at present outcrop

[2]some blastomylonites have >30% large porphyroclasts

Figure 35: A classification of fault rocks based on visual characteristics. Made by (Woodcock & Mort, 2008).

No articles regarding data on strength and stiffness of fault rock, breccia or gouge for engineer geological applications has been found. According to (Riedmüller et al., 2001) the material of fault zones show great heterogeneity, consisting of randomly occurring material of more or less undeformed, unaltered stiff rock fragments surrounded by a soft weak matrix. This matrix can in some instances be altered due to second mineralization and therefore harden and be cohesive. Also, they state that the ratio between matrix and clasts is extremely variable, and the distinction between fragments and matrix is given by the scale of the problem. Another important factor is the water content and hydraulic connectivity which can lead to an extended weakening of the material especially due reduced effective stress of the fault. Riedmüller et al. (2001) states that the extensive complexity of brittle faults makes the geotechnical characterization and investigation difficult.

According to Kalender et al. (2014), measurements of strength and stiffness of fault rock and breccias is nearly impossible to do because of the difficulty in attaining high quality, undisturbed drill core samples. Also, there are problems in preparing laboratory specimens and gather cohesion values internal friction angle and UCS for such complex mixtures. Fasching and Vanek (2011) states that only tectonical breccias and mylonite can be addressed with hard rock behavior, and that regular breccias, fault gouge, and cataclasis can only be attributed with soft rock and/or soil behavior. Cohesion can only be disregarded if the material is believed to not show mineral-bonding between the grains and that there is little of silt and clay-sized grains in the material. Kalender et al. (2014) developed a failure criterion for fault rock with a significant degree of clasts, which is essential due to the complex failure behavior such soil/rock hybrids entail. However, this criterion only works for hybrids with low degree of grain-interlocking under low confinement stress and under-estimates the strength of the rock.

The first 5 km of the crust is dominated by brittle deformation mode, and the ductile zones is usually found deeper than 5-10 km. There is also possible to find a composite zone which show a combined behavior. Thus, a fault, given the definition above, is the same as a brittle shear zone, where a transition zone to a ductile shear zone happens between 5-10 km. An illustration is given in Figure 36. (Sæter, 2005)

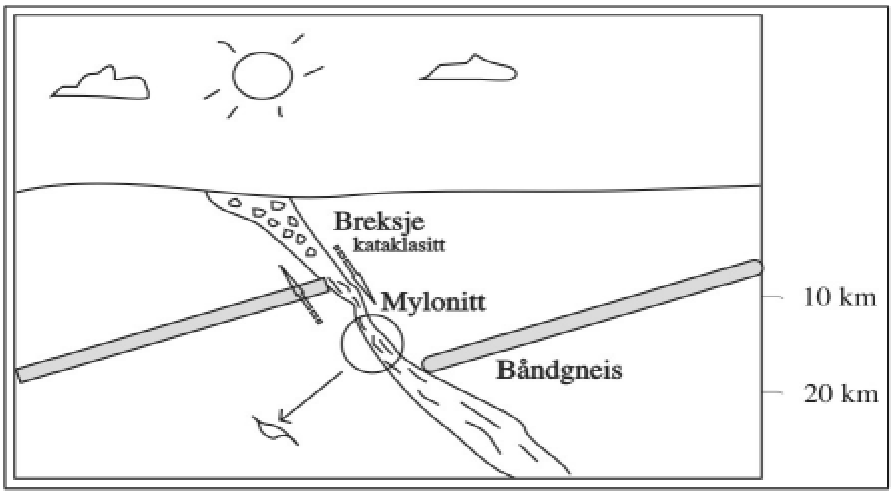


Figure 36: A fault who undergoes a gradual transition from brittle character to a ductile shear zone. Made by Sæter (2005).

Furthermore, as seen in Figure 36 the material of the fault core consolidates with depth, indicating an increase of strength.

Fault structure:

In general, fault zones vary in complexity both along strike and dip (e.g. Caine et al., 1996). This yields also over relatively short distances. In other words, fault zones are 3-dimensional systems which vary significantly both in geometry and material behavior. Furthermore, fault zone structure, mechanics and permeability can vary severely both over geological time, and in a time span relevant for a variety of industrial applications, see Figure 37. A fault zone cores extension, both normal to strike and parallel with dip, is closely linked to the slipping distance.

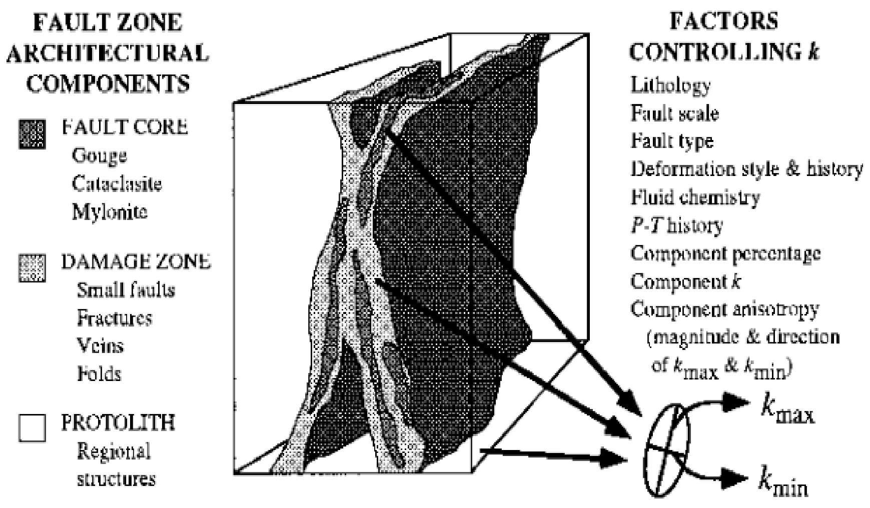


Figure 37: Conceptual model of a fault zone with removal of the protolith showing its complexity. This model focuses on permeability (k) (Caine et al., 1996).

Fault zone structure depends on the depth of formation, protolith, tectonic environment (e.g., strike-slip, extension or compressional), magnitude of displacement and fluid-flow. Generally, faults in low-porosity rocks consists of a fine-grained core surrounded by a fracture dominated damage zone. (Balsamo et al., 2010)

Qualitative description of fracture damage zones (brittle protolith) surrounding a fault core, by e.g. McGrath and Davison (1995) and Berg and Skar (2005), show that damage zones comprises fractures at a range of sizes. Both microfractures and larger fractures that may shows some small shear displacements and consist of cataclasis. Also, it can be hard to distinguish fault damage zone fractures and subsidiary fault structures. Brittle rock damage zones mostly consists of mode 1 fractures which is the same as extension fractures, also called dilatant fractures due to the volume increase they lead to (Blenkinsop, 2008). In low porosity rocks the density of fractures decreases exponentially when moving away from the fault core, for instance showed by Wilson et al. (2003) in their work on mapping of micro fracture on Punchbowl fault in San Andreas in California. This behavior has been linked to the decrease of stress when moving away from the fault tip where fracture mechanic models anticipate stresses of great magnitude. According to Mitchell and Faulkner (2009), maximum microfracture density is often attained in close proximity to the fault core and is dependent on rock type but independent on the displacement of the fault, see Figure 38.

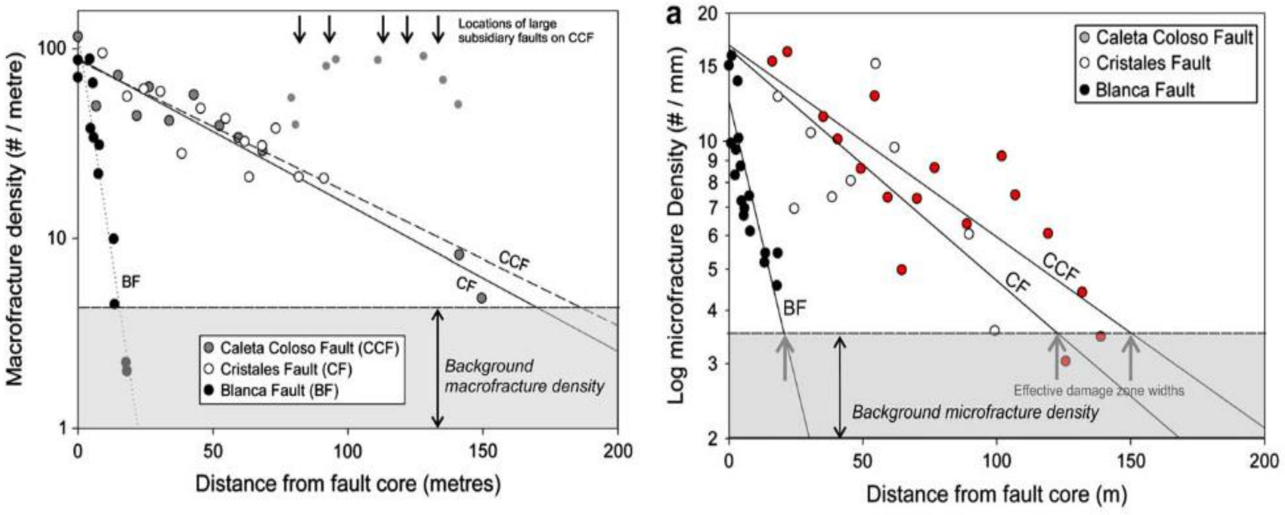


Figure 38: Models showing how density of macro-fracture (left) and microfracture (right) decreases with distance from fault core given from three strike-slip fault zones in low porosity crystalline rocks north in Chile. Bianca fault showed 35 m of displacement, Cristales Fault showed 220 m, and Caleta Coloso showed 5 km, from Mitchell and Faulkner (2009).

However, damage zone width, both in micro and macro scale, tends to be thicker with increasing displacement. Mitchell and Faulkner (2009) studied damage zone widths from several faults in the same granodioritic batholith ranging from centimeter scale and up to kilometer scale showing this effect. Their work indicates a decrease of the width development for high fault displacements. Their work is presented in Figure 39.

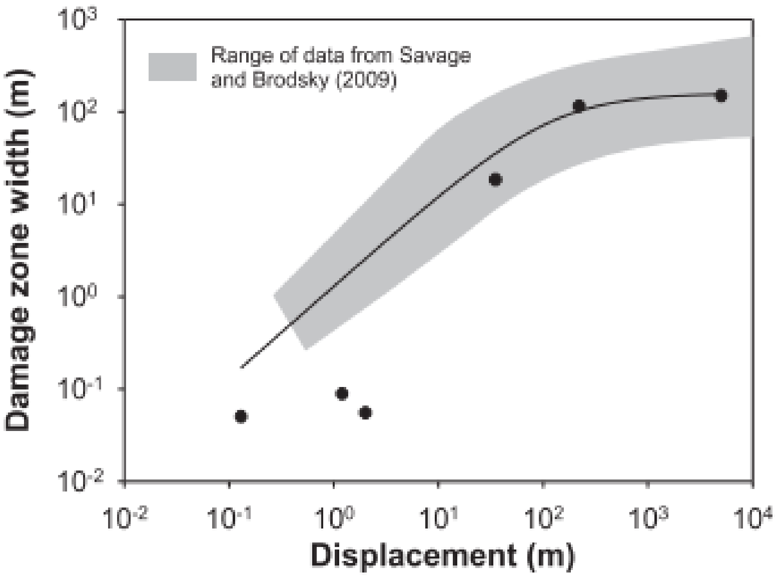


Figure 39: The increase in damage zone width plotted against fault displacement (Mitchell & Faulkner, 2009). The shaded area shows the extent of data compiled by (Hatheway & Kiersch, 1986); Savage and Brodsky (2010).

Faulkner et al. (2010) implies that the prediction of width of fault cores are less good than the prediction of damage zone width because of a more prominent scatter in the data. However, a model relating fault core thickness with displacement may present a good approach to find the upper bound of core thickness. Scholz (1987) is one that have provided such a model and his results is given Figure 40.

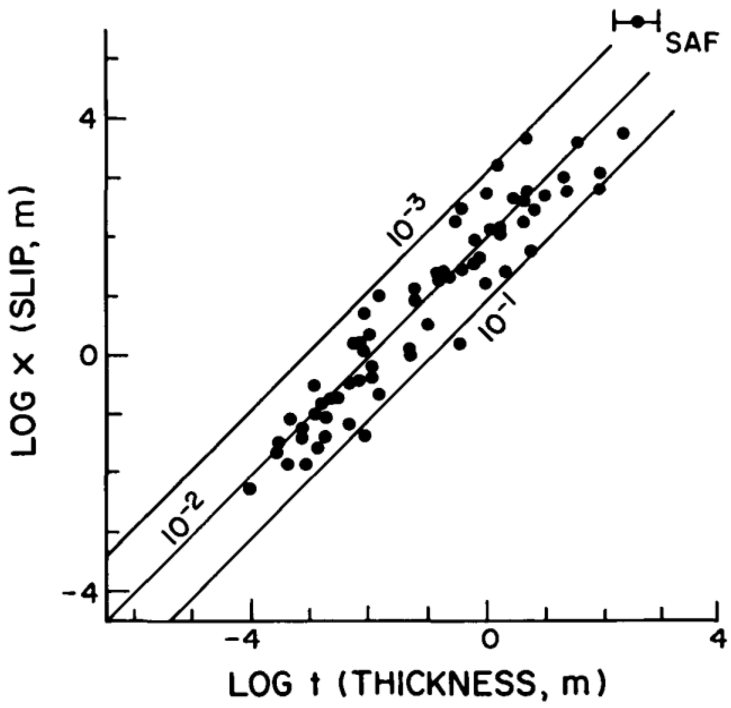


Figure 40: Log-log plot of gouge-zone thickness, t, against total slip, x. Estimated by Scholz (1987).

According to McClay (1987), a faults angle is directly linked to the stress-situation when the fault was created leading to the following: (a) Strike-slip movement of the fault result sub-vertical dip of the fault, (b) dip-slip movement of normal faults leads to dips of around 60°, (c) dip-slip movement of reverse faults leads to dips of around 45°, dip-slip movement of thrust faults give dips less than 45° . Thus, seen together they show the total range of dip of faults between 0 to 90 degrees.

2.3 FEM modelling

The main purpose of this section is to present numerical modelling as a tool to attain a mathematical description of the redistribution of stresses post excavation. This section is divided in three subsections: (a) gives a brief summary of the use of the general use of numerical modelling and focuses then on the use of finite element modelling, (b) presents the dilemma between usage of 2-dimensional and 3-dimensional numerical models, and (c) presents the problem of dynamic unloading.

2.3.1 A brief summary of numerical modelling and finite element method

This subsection is made for two reasons in mind. It begins with a short summary of the general use of numerical methods. It is then narrowed down to a more specific presentation of the finite element method.

It is normal to describe the stress-strain relation of rock mass using partial differential equations and solve it using numerical software such as RS2 (earlier Phase2), FLAC3D and more (e.g.Cai, 2008; Mao & Nilsen, 2013; Trinh et al., 2010). To be able to construct a set of partial equations to describe the stress and strain distribution, it is necessary to define a set of constitutive laws describing the physical behavior of the examined system.

First of all, the rock mass materials' response to forces must be defined. In the field of rock engineering, it is normal to assume that the inspected rock mass' material behavior pre-failure, can be described by **Hooke's law of linear elasticity**, see Equation (2.10). Post-failure it is normal to assume that the rock behaves according to a certain flow rule, generally defined in equation (2.20). In many occasions, it is normal to only calculate the deformation just after excavation. Under such circumstances it can be assumed no time-dependent deformations, which is an important factor in long-term stability, see section 2.1.6.

Second of all, another important aspect of the physical system is the description of the forces that are the source of the deformations. In this case it is normal to assume that the surface and body forces acting on the rock mass follows the **Newton's laws of motion**. One of the most central body forces is in this regard the force of gravity.

These two assumptions are vastly used in the field and makes it possible to describe the deformation response of rock mathematically. For easy problems it is possible to calculate this response analytically. For example, Kirsch (1898) found an analytical description of the elastic stresses around a circular tunnel in an infinite continuous rock mass. However, most practical applications cannot be predicted using this analytical solution. Thus, for most rock engineering applications it is necessary to calculate the equations numerically. This means that a **numerical method** is used to approximate the true solution.

Yet, an important note is that the true solution is not necessarily known for a certain problem. It is therefore important in practical engineer applications to verify the numerical model by comparing it with similar numerical models already verified, or by verifying it by comparing the predictions with deformation monitoring of the tunnel of a specific project post excavation. If there are significant deviations, the numerical model is adjusted accordingly (Palmström & Stille, 2010)

There are many numerical methods developed over the years which suits a variety of physical applications. According to Jing and Hudson (2002) the most commonly applied numerical methods for rock mechanic problems are: (a) continuum methods; the finite difference method (FDM), the finite element method (FEM), and the boundary element method (BEM), (b) discrete methods; the discrete element method (DEM), and discrete fracture network (DFN) methods, and (c) hybrids of the methods presented in (a) and (b).

Furthermore, Jing and Hudson (2002) states in their review that the choice of modelling with a continuum or discrete method depends mainly on the problem scale and the geometry of the fracture system. The continuum approach is applied if there are few joints present in the rock mass, or if fracture opening and full-scale block detachments do not dominate the failure mode of the rock mass. The discrete approach is mainly applied for moderately jointed rock mass in where detachments and rotation of blocks dominates the failure mode. In later years, a hybrid approach has been developed to utilize the strengths of each method in complex cases, where the rock mass alternates between being dominated by fractures and not. As stated by Onah (2012), a common denominator of element methods, is that they all approximates the numerical solution by dividing the problem into small elements, for then to synthesize the solutions of each element into a general solution of the system as a whole.

It is not within the scope of the thesis to go into the details of each model. However, both Onah (2012) and Jing and Hudson (2002) states that the best numerical models are based on a hybrid of continuum and discrete approaches. Also, it has been stated that the rapid development of hardware over the two last decades has boosted the use and development of numerical element methods due to reduction of computational time. This also has paved the road for complex methods of the abovementioned hybrids.

Despite of the vast number of numerical methods, has the **Finite Element Method** (FEM) been the most used approach in stress analysis of rock mass in the field of rock engineering over the last decades (e.g. Cai, 2008; Mao & Nilsen, 2013) and is in wide use in the field of rock engineering. One reason for this is that FEM-tools enables analysis of rather complex geometries and variable material behavior, which is easy to define using a CAD-based software. Here complex material behavior refers to the elastic-plastic properties of the rock mass can be varied to fit the problem at hand.

Finite element method is a popular method for solving differential equations describing a 2-dimensional or 3-dimensional space numerically (e.g. Saabye Ottosen & Petersson, 1992). It is a versatile method and can, e.g., be used for applied problem-solving in the fields of structural analysis, fluid flow, mass transport, and more. This method also applies for stability assessments of rock in where Newton's laws and Hooke's law is assumed to describe the constitutive behavior. FEM is designed to solve a big and complex problem by: (a) subdividing the problem into subdomains, called **finite elements**, (b) defining approximation functions describing each subdomain's behavior, (c) assemble each approximation function into a larger system of equations for the final calculation, and (d) the approximation of the global system is then done by minimizing an associated error function. The global system of equations has known solution-techniques, and can be calculated by defining an appropriate set of boundary conditions.

The finite element method assumes that a mathematical continuum can be divided into discrete geometrical entities in which each entity behaves under the same mathematical rule. An example of such a rule is the failure criterion, which describes when and where a given material undergoes failure. The complete system of the discretized finite elements is referred to as the **mesh**, which can be tailored to present more detailed values in areas considered to be crucial in a project. In fact, it is the mesh that is the reason for the high computational time of FEM. Thus, by controlling the size of the finite elements, the accuracy of the predictions can be controlled, which again controls the time it takes to reach numerical convergence. The smaller the elements, the more demanding it is to calculate the model and reach convergence.

Inner forces in the discretized continuum, is transferred from one entity to another by nodes defined on the rim of each finite element. The source of the inner forces is often due to impact of outer forces such as gravity. A key concept of the force transfers is the superposition principle. This states that all forces working on a geometrical entity are independent and can be summed together making a resultant response called a resultant force. Thus, the principal of super position demands that the mathematical continuum to be a vector space, which implies that the continuum behaves linearly. Thus, the rock mass modelled is assumed to be a mathematical continuum and vector space abiding the laws of Newton and Hooke.

To sum it up, it is made a list of some pros and cons of FEM based on the information presented above. The pros are as follows:

(a) FEM is applied to numerous problems which means there are much information regarding its uses, strengths and weaknesses for given engineering applications.
(b) It is possible to model physical properties too complex for any closed bound analytical solutions.
(c) There are no limitations regarding the complexity of the geometry of the problem.
(d) There are no limitations regarding which physical problem to be modelled as long as there is a set of constitutive laws which is applicable for the problem at hand.
(e) It is possible to model material anisotropy and non-homogeneity.
(f) There are no limitations of the loading and boundary conditions.
(g) The design of the method is time-saving. It is faster to solve small domains and then synthesize each contribution into a general solution. The reason for this is that the equations describing each finite element's response to a certain force is easier to calculate than solving one equation describing the entire domain.

The cons are:

(a) Large amount of data is required as input for defining the mesh, which is demanding for the computer. It is imperative for the user to create a mesh which is refined to only show details in where it is necessary to know the details.
(b) Even though a mesh is optimized is the calculation time of the finite element method demanding compared to other numerical element methods. This again leads to high computational time of each method calculated. It is therefore important to have access to computers of high performance.
(c) The resulting output will vary considerably, depending on the quality of input parameters and the defined mesh. Thus, it is important to verify the model at hand to ensure the quality of the output.

2.3.2 A comparison of 2D and a 3D numerical modelling

Mao and Nilsen (2013) states that 3-dimensional numerical methods are frequently used in the industry when tunnelling through major weakness zones or faults with complex geometries. It is further stated that 2-dimensional methods are fast and convenient to use, but simulation results indicates that the results may deviate significantly from reality if not used with care.

In their article, a comparison between the 3-dimensional finite difference method program FLAC^{3D} and the 2-dimensional finite element method program Phase^2 (the old name for RS2). Widths of weakness zones, different sets of stress ratios, and different strengths of weakness zones were all varied using both methods. These variations set the base of the analysis. Areas of yielded zones and vertical displacements at the crown and invert was compared in where the differences in performance were presented.

One important conclusion was that for analysis of the effects of weakness zones perpendicular to the tunnel alignment, a realistic representation is only ensured if the thickness of the zone is much wider than the span of the tunnel. The wider the span of the tunnel, the wider must the weakness zone be. It is further stated that it is rather unusual for the most commonly used excavation spans to face zones of such attribute. Also, it is seen that the deviation between 2-dimensional and 3-dimensional methods increases with decreasing strength of the zone material. It is remarked that it is hard to define deformations of tunnels correctly before rock support is installed when Phase^2 is used. Finally, it is stated that it is not realistic to develop empirical diagrams for tunnelling in a weakness zone due to, e.g.: (a) the geometry of weakness zone and tunnel, (b) the anisotropic stress state, (c) the material properties of weakness zone, and (d) the influence of side rock.

2.3.3 The effect of dynamic unloading

This subsection is taken from the specialization project (Kaasbøll Andresen, 2021). It is included to present the errors connected with numerical models that do not take into account the effect of dynamical unloading. The term dynamic unloading is referring to the transition state of stress, between pre- and post-excavation. For instance, in drill and blast methods, it is a short period of gradual unloading during blasting.

Cai (2008) conducted an analysis of the influence of stress path on tunnel excavation response, to broaden the understanding on numerical tool selection and modelling strategy. He compared the two commercial 2-dimensional continuum modelling programs FLAC and Phase^2 (now named RS2). Both programs are used for modelling soil, rock, and structural behavior in the fields geomechanics, geotechnics, and in civil and mining engineering. The programs differ in their theoretical formulation. Phase^2 is formulated with an implicit finite element scheme while FLAC is based on explicit finite difference formulation. Cai showed in his article that differences in modelling outcome can be traced back to the difference in their formulations in which leads to differences in the stress paths. No difference was found when the problem was considered linear elastic because of the unimportance of the stress path. However, for a tunnel excavation in an elastic-plastic material simulating the drill and blast method, a significant difference in yielding zone distribution was obtained, especially in material with low strength relative to the magnitude of the in-situ stress.

The differences in yielding zone were attributed to the effect of dynamic unloading, and hence differences in their description of the stress path. Cai followed a traditional modelling procedure where a "sudden" excavation (modelling drill and blast) is performed. This means that material where the excavation is planned suddenly is deleted, and the response of this deletion is calculated. Cai points out that FLAC has an explicit finite difference formulation which makes of a smoother transition from the in-situ stress distribution to stress distribution after excavation, resulting in a more realistic description of the dynamic unloading which happens in practice. Phase^2 does not take this dynamic unloading into account because of its implicit finite element formulation. On the other hand, deduced from the results of his models, the importance of this dynamic effect faded when the strength of the material increased. Also, dynamic unloading was less important given shear failure, even though it showed some significance at shallow depths.

2.4 Programming

This section is added for two reasons. First of all, a set of relevant definitions of programming is made, to create the foundation of the discussion of the scripting process. This will also help readers which is not that fluent in the world of programming. Second of all, the software utilized in the thesis is presented. This will give the reader some insight of the development process and in the method used in processing the data.

If the reader wants to know more of the script, see appendix (XX). It is out of scope of this thesis to go into the details of its functionality. However, see chapter (XX) to get some insight of the development process.

2.4.1 Definitions

The main purpose of this subsection is to define relevant programming terms to set the foundation of the discussion of the scripting process.

A **computer** is an electronic device for storing and processing data, typically in binary form, according to instructions given to it often most by a user.

A **user** is a person who utilizes a computer or network services.

Hardware is the machines, wiring, and other physical components of a computer or other electronic system. For instance, the motherboard which enables the physical connections between the other components of the computer and the central processing unit (CPU) that executes the instructions given to the computer.

Software is a set of computer programs. It is categorized in two. **System software** is the software needed for the communication between the user and the computer to happen, and for the communication of the computer's hardware to happen. Operating Systems such as windows is an example of a system software. **Application** software, application or apps for short, is software made for solve specific tasks. Microsoft Office is an example of application software helping users doing office related tasks.

A **computer program** is a set or sequences of instructions of a programming language for a computer to execute and is most often defined in binary form. A computer program in the readable form for users is called source code. A computer program is a conglomerate of modules.

Source code or **code** is a term used to describe text that is written in a particular programming language.

Program language is a language designed for people to communicate with a computer. A computers language is the binary number system. This system is hard for users to communicate with. This motivates the need for programming languages. There are many programming languages for different purposes along with a set of pros and cons.

Structured programming is a programming paradigm made to improve the clarity, quality, and development time of a computer program by extensive use of the control flow constructs of selection and repetition, block structures, and subroutines. The most used programming languages today are structured programming languages.

Block structure of a code is a lexical structure of source code which is grouped together. One block consists of one or several lines of code called statements. Block structures can be nested, which means that a block can consist smaller blocks, etc.

A **statement** is the smallest entity of a code. It resembles one action to be carried out. For example, allocating the value 60 to the variable a.

A **command** or an **instruction** is referring to the action the computer is tolled to execute by a given statement.

Subroutines is a sequence of program instructions that performs a specific task, packaged as a unit

An **If/then/else** statements is a programming structure where a block of code is executed if a certain condition is met. This is a construct of selection.

A **loop** is a programming structure of repetition that repeats a block of code until a specific condition is met.

An **iteration** refers to one repetition of a loop. It can also be used as a verb. Then it refers to the action of looping/**iterate** over a given block of code or over a set of files, etc.

A **module** is a software component or part of a program that contains one or more routines. These routines may be other modules or functions. So, if a programmer has several functions and/or modules that can be grouped in a certain way, these functions is put in a module. This creates a folder structure making it possible to reuse the functions it resembles efficiently, and to share the work with other programmers in a structured and logical manner. One or more independently developed modules makes a computer program.

A **function** is defined as a portion of code that does a specific task. It is a goal of the programmer to make a function as general as possible. The reason is to reuse the function, when possible, to save work in the future. The same philosophy goes for the development of modules and computer programs, and for the entire field of data science. Reusing earlier work is gold.

A **data file**, or file for short, is an entity to store data to be used by a computer application or system. This can conclude both input and output data. A file does not usually contain instructions to be executed by the computer. Computer programs often works with files. Either when certain input data is needed for the execution of the program, or when output data needs a place to be stored. A file is given a specific place in the memory of the computer and only one file can exist in this particular position.

A **file extension** is the suffix to the name of a data file. For instance, .txt, .exe or .docx. The suffix defines the use of the data file. For example, .docx file says that it is a word file to be used by the computer program Microsoft Word.

A **directory**, better known as a folder, is a file system cataloguing structure which contains references to positions in the computer's memory of certain files. Directories enables the user to organize the content stored by the user.

A **compiler** or an **interpreter** is a translator of a programming language. It translates lines of a specific programming language to binary code understood by computers.

A **script** is a text-file containing all a sequence of instructions of a programming language not yet interpreted by a compiler.

The term **open-source** refers to computer programs in where the source code is publicly available to be downloaded. This means that the computer program is free to use and can also be altered after downloading.

A **commercial** computer program is the opposite of an open-source program. The use of it is protected by a pay wall. Also, it is seldom possible to download the source-code after paying.

Implementation is the realization of an application. In other words, it is the action of making the code necessary for the machine to do a certain task.

A **crash**, or a **system crash**, occurs when a computer program or operating systems stops functioning and stops, often accompanied with an error message to help the user find a solution to the problem. A crash can be forced through the action of the user, or it can be as a result of actions beyond the control of the user. An example of the former is when the user opens to many programs at a time which ends with the computer crashing. An example of the latter is when a commercial software update leads to new bugs introduced by the changes included in the source code.

A **bug** is a coding error in a program. To **debug** is the action of fixing bugs.

A **debugger** is a computer program designed to help a programmer to test and debug a script. A debugger is mainly used to run a script under controlled conditions and makes it possible to track the progression of the instructions and monitor changes in memory areas used by the script. This can be used to indicate in which segments of the script errors are.

Version control is a system to manage changes to computer programs, documents, or other collections of information. It works as a library in where earlier versions of, e.g., source code is stored whenever the user wants it to be stored. Version control makes it possible to go back to earlier versions if the programmer manages to introduce major bugs in a source code. It works therefore as a safety net. There are more advantages, but they are not relevant for this project.

To **run** a program or a script is an action done for making the computer to execute the sequence of instructions contained in the program or script.

To run a **batch** means that a computer program runs a sequence of files in where the same instructions is executed on each file.

The **user interface** (UI) is the area where interactions between users and computers occur.

Application Programming Interface (API) is a term referring to ways for computers or computer programs to communicate. An API is a document or standard that describes how to make this connection. A programmer uses this standard to implement some features of a computer or software into another computer or software. Also, one purpose of APIs is to hide certain parts of a system, only showing the details needed to get the wanted connection or functionality. For example, Spotify's and Facebook's developers implemented a connection with a person's Facebook profile and Spotify profile. Both software's' API was needed to establish this connection. Most likely, both companies only gave the information needed to establish this functionality guarding the rest of their source code.

Text User Interface (TUI) is a user interface where the interactions are done by text commands. For instance, most programming languages are TUIs. The navigation and communication with TUI-based programs is mostly the keyboard, but also with the mouse in some circumstances.

Graphical User Interface (GUI) is a user interface where the interactions happen using graphical icons and short text-based operations in rubrics. GUIs was developed to lower the steep learning curve of TUIs. Both the icons and rubrics are designed to explain what is needed by the user of the program. The keyboard and mouse are two devices used to navigate in and communicate with a given GUI-based program

A **Text editor** is a computer program to edit plain text for example stored in a .txt-file. Text editors is often used for programming purposes. Notepad++ is an example of a popular open-source text editor.

An **Integrated Development Environment** (IDE) is a software application that provides a source code editor, a compiler, and debugger in a seamless environment to increase productivity.

A **word processor** is a specialized text editor for the purposes of text publishing. For example, it is normal to use a word processor when writing an article or book. A word processor has specialized functions and user interface for this purpose in mind. Microsoft Word is an example of a commercial computer program.

Automation is in this thesis defined as processes, scripts or computer programs designed to reduce human labor and intervention.

The **screen** or **monitor coordinate system** of the computer is needed to define graphical representation of computer programs. This coordinate system differs from the cartesian coordinates system. The origin is on the top left of the monitor. There are only positive values on both on both axes. The x-axis has the same orientation as the cartesian system. However, y-axis is flipped 180 degrees with the x-axis as the rotation axis. The coordinate system of the monitor is compared to a 2-dimensional cartesian system in Figure 41.

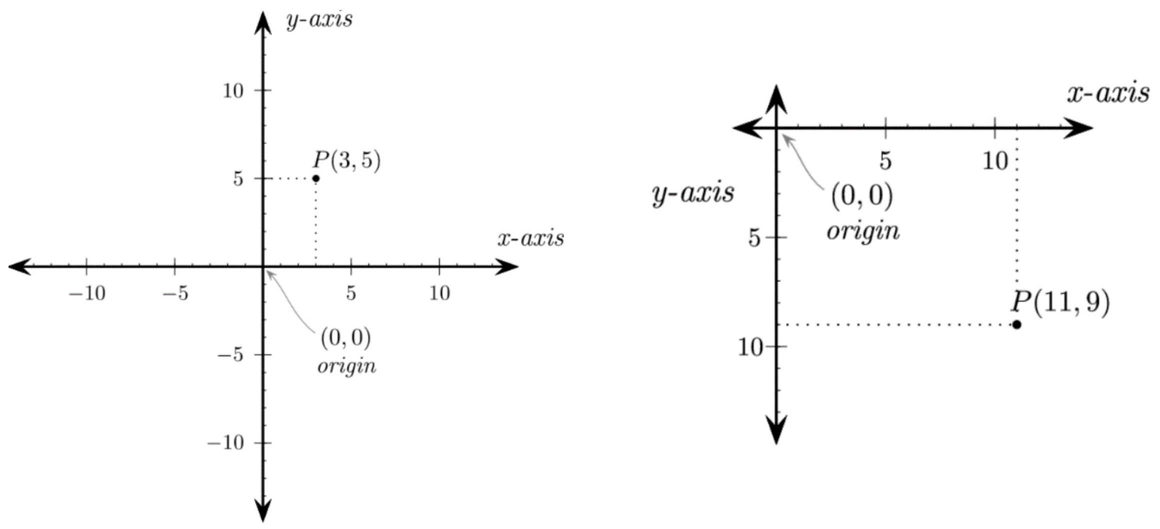


Figure 41: Illustration of Cartesian coordinate system to the left, and computer coordinate system to the right. Taken from (Vincent, 2017)

The ranges of the coordinate system axes are given by the monitor's resolution. Modern computers have resolution 1920/1080. This means that the length of the x-axis is 1920 and the length of the y-axis is 1080. The resolution is given by the number of pixels of the monitor. Pixel is short for picture element and is normally square shaped. Thus, the size of the pixel is the smallest possible to make on screen.

Input/Output manipulations is referring to reading of an input such as a text-file and to append or writing operations to an output, for instance, to the same text-file.

A **regular expression** is a sequence of characters that defines a search pattern of a text. Regular expressions are utilized to manipulate, find, or store substrings of a file or files, that are matching with the defined pattern. For example, a regular expression can define a pattern that finds all numbers in a text-file with 4 digits or more. It can also define a pattern that replaces all commas with dots. Thus, regular expressions provide a dynamic method of searching and manipulating strings. Regular expressions are given by a specific set of syntax.

The phrase **by hand** is used in the thesis when models is constructed using the RS2 software, and is used as the opposite to running the automation script developed in this thesis.

2.4.2 Software and tools used the in the thesis

The main purpose of this subsection is to give a brief summary of the software and tools used in the thesis work.

RS2 is a commercial 2-dimensional implicit FEM modelling program distributed by RockScienceTM. The term implicit refers to the category of methods used to solve partial differential equations. It is specifically designed for geotechnical and engineer geological applications. According to Mao and Nilsen (2013) and (Trinh et al., 2010), RS2 is a popular choice out of several FEM-based software used in the industry of rock engineering. Also, it is to this day widely used in the stability assessment of rock tunnels. In addition to be relatively easy to learn and use, it has big libraries of support, failure criteria and is linked to other programs distributed by RockScienceTM that makes the definition of input parameters of rock material, stress, etc. easier and more seamless.

RS2 Compute is the program RS2 launches to calculate a numerical model. This program can be opened independently, which makes it possible to calculate multiple files successively as a batch.

RS2 Interpret is the program RS2 launches to let the user visualize, interpret, and process data. It can be opened independently, which makes it possible to open RS2-files that are calculated without running RS2. By default, the results are presented in iso-contour plots in where colors are used to show areas where the investigated parameter has the same magnitude. Two examples of such parameters are: (a) the principal stresses, and (b) total deformations. RS2 Interpret comes with different visualization tools to further enhance the analysis. Three strong tools that have been central in this thesis are: (a) display the deformation vectors, to visualize the vector field, (b) display deformed boundaries, to visualize the total deformation of the tunnel periphery, and (c) excavation query line, to define and fetch the datapoints along the tunnel periphery.

Notepad++ is a widely used open-source text-editor. This implies that there are many modules implemented that are free to download. Notepad++ is a versatile tool to manage large datasets, e.g., stored in ".csv"- or".txt"-format. An example of a useful module is the compare plugin. This plugin compares two files, lines them up, and visualizes the differences. This makes it easy for the user to detect and analyze these differences.

Git is a an open-source software to track changes in any types of files and folders. It is mainly used to coordinate work among programmers that works on the same projects.

GitHub provides internet hosting for software development and version control for git users. In short terms, it makes it possible to store a development project online to share the work with other programmers. The git and GitHub functionality make it possible for others to fix bugs or make suggestions for further development of a project. The owner of the project gets the bugfixes and development suggestions and decides which of them to implement. The modules used in this thesis is mainly downloaded from GitHub. Another strength of git and GitHub is that it makes it possible for one user to work with a single project on several computers at once due to its push and pull functionality. When a change is made on the script in one computer, a push command is executed. This command sends the changes to GitHub to be stored. On the other computers, a pull command is initiated. This results in an implementation of those changes.

Excel is a software for managing spreadsheets and is vastly used. Excel was used in the data analysis of the thesis. To process the data gathered using RS2 Interpret, which was stored in ".csv"-files, the **Power Query** module was used. This is a tool to prepare large datasets before it is imported into the spreadsheet. The preparations are stored in a **query** that is linked to the path of a specific ".csv"-file. Therefore, if changes are introduced to the ".csv"-file, these changes will be automatically introduced into the table stored in excel. Another important tool of Excel has been **Visual Basic.** This is a scripting language that makes it possible to automize all aspects of work done in the excel environment. This tool was central in the creation of the scatter plots which was grouped in several subsets. If changes to the dataset was introduced, excel do not manage these changes well regarding subsets in scatter plots. Visual basic made it possible to introduce these changes automatically.

Python is a program language accompanied with a compiler. There are several versions both of the language and compilers Python is regarded as a **high-level language** since its syntax is very close to the English language. A high-level language is therefore easier to read for users, but is harder to read for computers. Consequently, the execution time of python is relatively slow. The reason is that it takes longer time to translate into binary code for the compiler. Lower-level languages such as C is faster. The fastest execution time would be a code of pure binary. To write in binary is however too impractical. Python is open-source and widely used in where developers share modules and computer programs, and to cooperate with other developers for free.

PyCharm is an IDE used in the development of the script. It was chosen due to being integrated with git and GitHub. One of the best functionalities was the monitoring of conflicts of implementation after a pull request. It worked by comparing the local file with the file stored on GitHub. The differences were shown graphically with different colors depending on the type of conflict. The user goes through the conflicts line by line and chooses to either implement a particular change to a given line or not.

PyAutoGUI is an open-source module, which enables control of the mouse cursor and the keyboard commands by writing a python script. It was developed by Sweigart (2014). All operations needed for full resemblance of the mouse and keyboard is achieved through a set of implemented functions. For example, the command of right clicking the mouse or the typing of the letter c three consecutive times.

The **Popen** constructor from the subprocess module can be used to open computer programs stored in the computer. The path of the specific computer program and the path of the file to be opened must be known to call the constructor.

The **OS** module enables functionality of the windows operating system. In the thesis, the following functionality was used: (a) creation and removal of directories, (b) fetching of directory content such as ".csv"-files, and (c) to check if certain files exist. Also, the **Shutil** module enables more intricate OS-functionality such as copying of files and file removal.

The **Time** module consists of functions related to time and date. Two notable functions are: (a) the time function, which can be used to record the computation time of scripts, and (b) the Sleep function used to postpone the execution of succeeding lines of a script.

The **numpy** module provides arithmetic operations and a diverse set of mathematical equations.

The **pandas** module provides functions to work with data structures such as excel sheets and ".csv"-files. One import aspect of this module is the matrix-functionality, in where matrix addition, multiplication etc. is well defined.

The **re** module provides functions that utilize regular expressions. For instance, the function **findall** returns all instances of a string that are given by the pattern defined by a regular expression. Another example is the function **replace**, which replaces substrings of strings given by the pattern defined by a regular expression.

3 Parameter study setup

In the following chapter, all parameters needed to replicate the experiments are presented.

There is one experiment for each weakness zone thickness T. From Table 1 it can be seen that there were 5 experiments in total, where the overburden H, the weakness zone angle Θ, and the shortest distance from tunnel center to weakness zone Γ is the parameters varied for each experiment. Thus, for each experiment T is constant. In the last column of Table 1, the numbers of models required for a sensitivity study for each parameter is presented. The total of models required to conduct this study is 3696, 924 models for each T. All combinations of the parameters in Table 1 are defined in the main file of the python script before the compilation and execution.

PARAMETERS OF GEOMETRY AND STRESS - SENSITIVITY

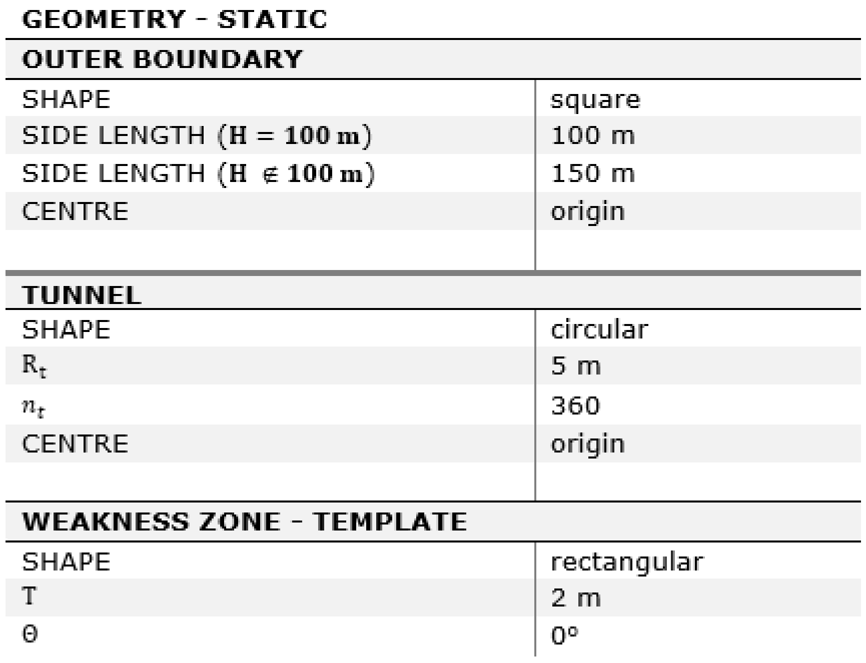
																						n
T [m]	1	2	4	8																		4
H [m]	100	200	300	500	800	1200																6
Θ [°]	45	52.5	60	67.5	75	82.5	90															7
Γ [m]	ZC*	0	0.25	0.5	0.75	1	2	3	4	4.5	5	5.5	5.75	6	7	8	9	10	11	13	15	20	22
																						3696

Table 1: The table enlists the parameters varied in the sensitivity experiment. In the last column, the number of values for each parameter are given.The value of 3696 is the total number of models constructed in the thesis.

All parameters given in Tables (Table 2-Table 5) is defined in the template models, defined in RS2 by hand. An important note is that the only difference of the template models is the size of the outer boundary.

In Table 2 all information regarding the geometries of the template models is defined. There are in total two template models; one for $\mathrm{H} = 100\ \mathrm{m}$, and one for $\mathrm{H} \notin 100\ \mathrm{m}$. The only difference between the two template models is the length of the sides of the square shaped outer boundary. Both template models have; (a) square-shaped outer boundary centered in the origin, (b) circular shaped tunnel centered in the origin with $\mathrm{R_t} = 5\ \mathrm{m}$ and is defined by $\mathrm{n_t} = 360$ points, and (c) a rectangular shaped weakness zone in where $\mathrm{T} = 2\ \mathrm{m}$ and $\Theta = 0^{\circ}$ centered in the origin.

Table 2: The parameters defining the geometries of the template models are presented here.



GEOMETRY - STATIC

OUTER BOUNDARY	
SHAPE	square
SIDE LENGTH ($\mathbf{H = 100\ m}$)	100 m
SIDE LENGTH ($\mathbf{H \notin 100\ m}$)	150 m
CENTRE	origin
TUNNEL	
SHAPE	circular
$\mathrm{R_t}$	5 m
n_t	360
CENTRE	origin
WEAKNESS ZONE - TEMPLATE	
SHAPE	rectangular
T	2 m
Θ	0°

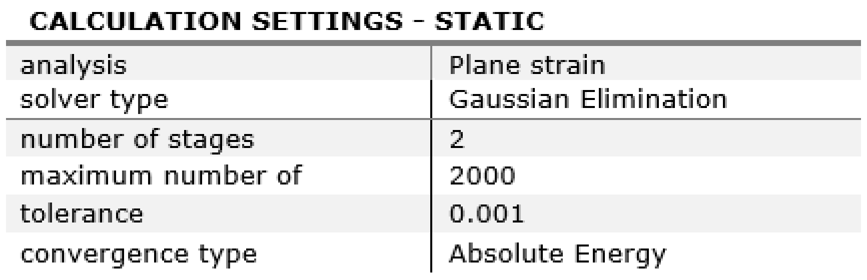
In Table 3, all the parameters defining the stress state and the material behavior of the geological entities are defined. The general picture are as follows; (a) isotropic hydrostatic stress, (b) drained rock mass with no joints, (c) host rock with isotropic and post-peak strain softening material behavior with failure described with GHB-failure criterion, and (d) a weakness zone material which is isotropic and plastic in failure is described by MC failure criterion.

Table 3: The parameters defining the stress state and material behavior of the geology are presented below.

MATERIAL PARAMETERS AND STRESS RATIO - STATIC	
Stress field	hydrostatic
k, both directions	1
Water	no
Joints	no
HOST ROCK	
ρ	0,027 MN/m^3
failure criteria	GHB, isotropic
material type	elasto-perfectly plastic
GSI	80
E_{rm}	17607 MPa
ν_{rm}	0.3
σ_c	125 MPa
m_b	13.7
s	0.11
a	0.5
tensile cut-off	no
mb_r	13.7
s_r	0.11
a_r	0.5
ψ	0
WEAKNESS ZONE	
ρ	0,020 MN/m^3
FAILURE CRITERIA	MC, isotropic
Material type	elasto-perfectly plastic
E_{rm}	300 MPa
ν_{rm}	0.3
t_0	0 MPa
φ	30
c	0.34 MPa
t_r	0 MPa
φ_r	30
c_r	0.34 MPa
ψ	0

In Table 4, the calculation settings are defined. It is assumed plain strain behavior. The convergence is attained using the absolute energy convergence type solved by Gaussian elimination. It is defined two stages; one for before excavation, and one for after excavation. The tolerance is default, but the number of iterations is changed to 2000.

Table 4: The parameters defining the settings of the calculation is presented here.

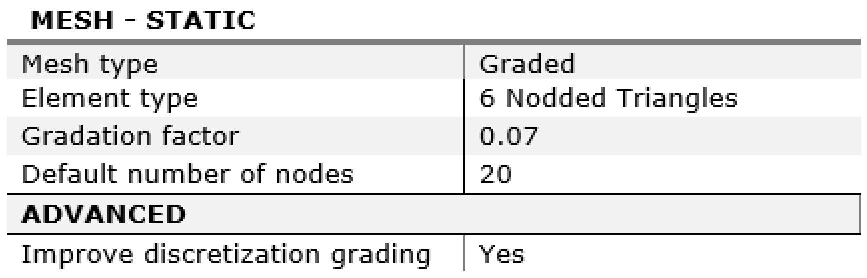


CALCULATION SETTINGS - STATIC

analysis	Plane strain
solver type	Gaussian Elimination
number of stages	2
maximum number of	2000
tolerance	0.001
convergence type	Absolute Energy

In Table 5, the parameters defining the discretization and mesh is presented. The following can be seen: (a) the mesh type is graded, (b) it is used 6 nodded triangle-shaped elements, (c) the gradation factor is set to 0,07, (d) the number of nodes is set to 20, and (e) improve discretization grading is turned on in the advanced section of the define mesh window.

Table 5: The parameters defining the mesh settings are given below.



MESH - STATIC

Mesh type	Graded
Element type	6 Nodded Triangles
Gradation factor	0.07
Default number of nodes	20
ADVANCED	
Improve discretization grading	Yes

4 Results

In this chapter, the results of the sensitivity study are presented. It is chosen a graphical approach in where four charts, one for each zone thickness, is created. In each chart the maximal total strain is plotted against the normalized shortest distance between tunnel center and weakness zone. The distance is normalized by dividing the distance on the radius of the tunnel (Γ/r_t). Thus, both the total strain (Type equation here. and the distance is normalized using the same parameter. The model is designed such that a given overburden is given by a line of distinct color. When moving from one cluster of overburdens to another, the zone angle is changed.

Figure 42 represents the chart in where the zone thickness is equal to 1m. Figure 43 represents the chart in where the zone thickness is equal to 2m. Figure 44 represents the chart in where the zone thickness is equal to 4m. Figure 45 represents the chart in where the zone thickness is equal to 8m. It was necessary to have each chart as large as possible. Therefore, it is dedicated one page for each chart.

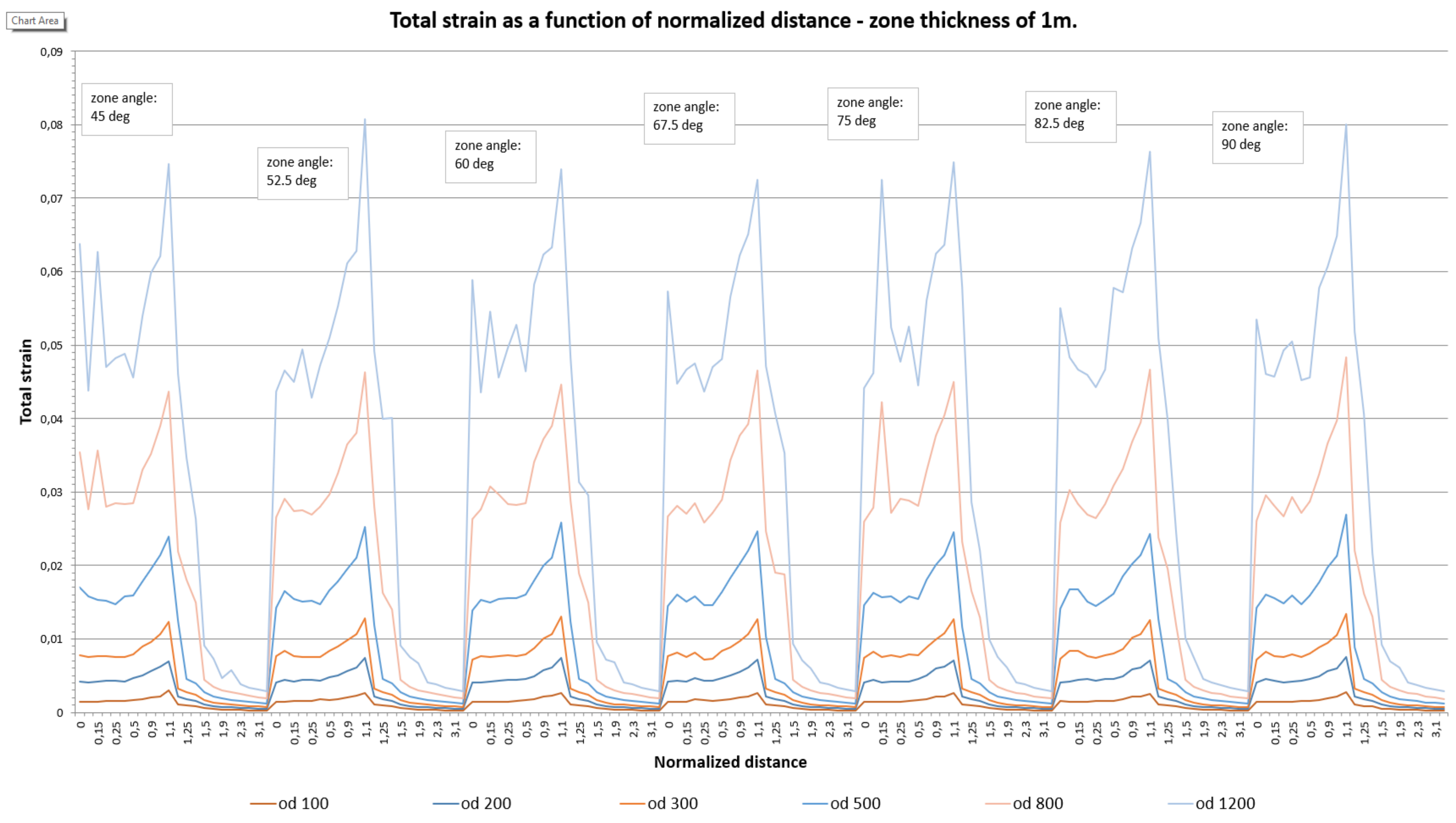


Figure 42: A line chart presenting the total strain plotted against the normalized shortest distance from tunnel center to weakness zone for zone thickness of 1m. The difference of overburden is given by the different color, and distinct entity marks a new zone angle.

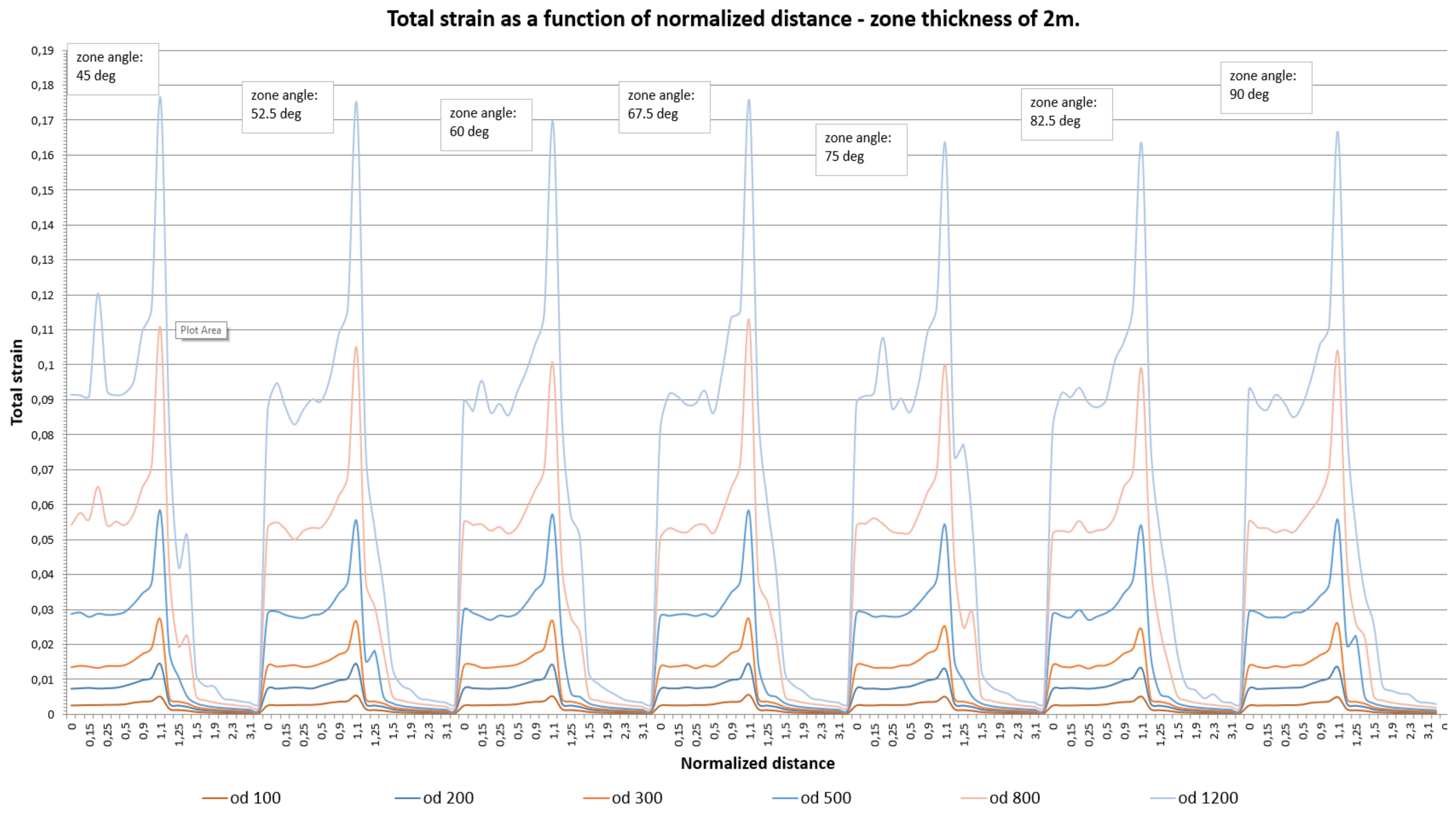


Figure 43: A line chart presenting the total strain plotted against the normalized shortest distance from tunnel center to weakness zone for zone thickness of 2m. The difference of overburden is given by the different color, and distinct entity marks a new zone

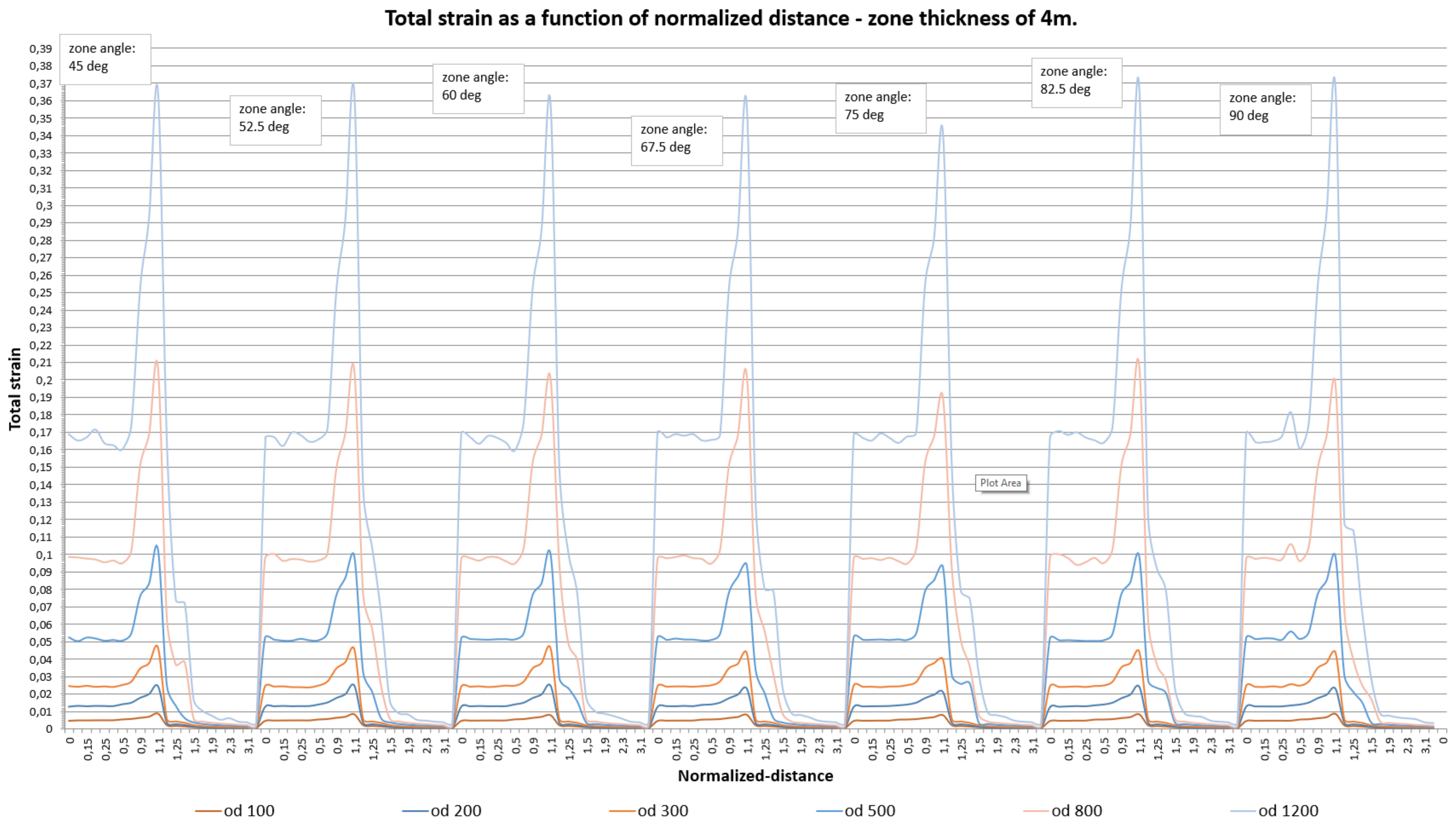


Figure 44: A line chart presenting the total strain plotted against the normalized shortest distance from tunnel center to weakness zone for zone thickness of 4m. The difference of overburden is given by the different color, and distinct entity marks a new zone

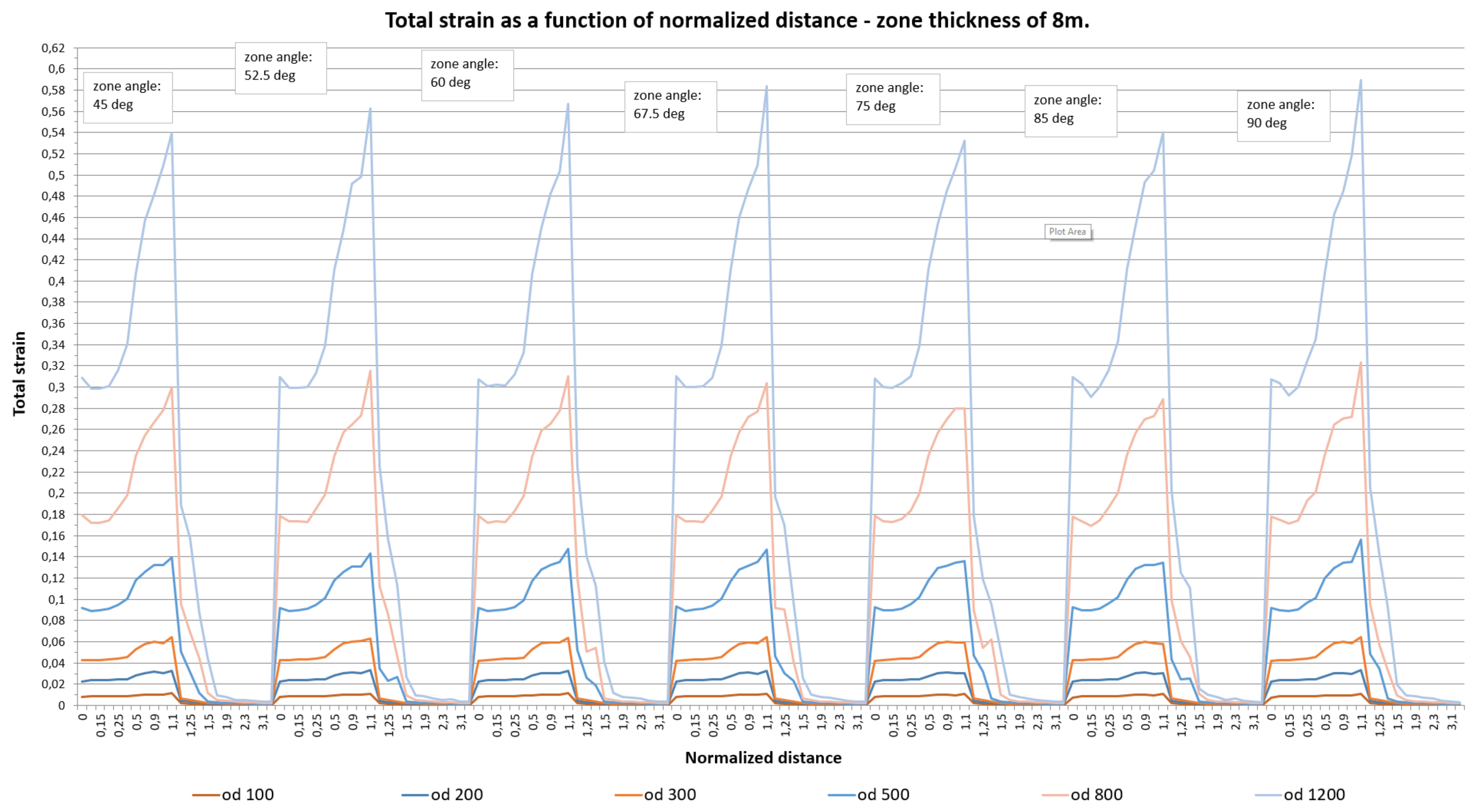


Figure 45: A line chart presenting the total strain plotted against the normalized shortest distance from tunnel center to weakness zone for zone thickness of 8m. The difference of overburden is given by the different color, and distinct entity marks a new zone

5 The numerical model and experiences of the automation process

This chapter is included to highlight some of the most central dilemmas faced in the choice of software and the following development process of the automation script. This chapter is divided in three sections: (a) the choice of numerical method is discussed, (b) the choice of numerical software is discussed, and (c) the implications the choice of numerical software had on the development on the sensitivity study, which lead to the development of the automation script is discussed.

The details of the script and its performance is not within the scope of this thesis. However, the whole python project developed in the thesis is included in the appendix, for those who are interested. Furthermore, a zip file containing all the models created with this script together with the calculated data sets are to be found in NTNU Open.

5.1 Choice of numerical method

This discussion is mainly based on section 2.2 and 2.3.

It was found no analytical solution describing weakness zones influence on stability of tunnels in the literature study of the specialization project (Kaasbøll Andresen, 2021). Thus, to be able to analyze how weakness zones affects stability it was necessary to describe the stress-strain relation using a numeric-mathematical approach based on solving a set of partial differential equations.

To be able to construct a set of partial differential equations, it is normal in rock engineering to assume that the rock mass behaves according to Hooke's law of linear elasticity pre-failure, according to a flow rule of plasticity post-failure (Hudson & Harrison, 1997), and by for instance the Burger rule when including time-dependent deformations(Li, 2018). The reason for this is simple. Accompanied with representative input data and qualitative knowledge of the deformation behavior of similar rock mass, these three approaches combined resembles the deformation behavior of rock well enough to enable creation of numerical models (Hoek & Brown, 1997; Hudson & Harrison, 1997; Li, 2018; Wawersik, 1968).

These three constitutive models, combined with the assumption of the system behaving in accordance with Newton's laws and suitable failure criteria, it is possible to create the necessary theoretical base to describe the stress distribution changes post excavation numerically. These assumptions were therefore necessary for the thesis to be able to conduct it.

It is imperative for the choice of numerical method that is handles creation of rather complex geological geometries and material compositions. The reason for this is the complexity of the geometry and material behavior of weakness zones (Caine et al., 1996; Riedmüller et al., 2001). In fact, Faulkner et al. (2010) states that a fault zone seldom is one single entity, but should instead be seen as a system of several fault cores with an intense state of fracturing in between. Also, according to Mitchell and Faulkner (2009), the fracture density does not abruptly reach zero with a given distance, but show a non-linear decrease with distance. The magnitude of a fault zone highly depends on the travel distance of the slip, and the material mode of the host rock, which also is highly variable (Hatheway & Kiersch, 1986; Savage & Brodsky, 2010). The same complexity also shows regarding the fault's material behavior. According to (Riedmüller et al., 2001) the material of fault zones show great heterogeneity, consisting of randomly occurring material of more or less undeformed, unaltered stiff rock fragments surrounded by a soft weak matrix. Also, Sæter (2005) argues that the material behavior of the fault is dependable on the depth, in where the ductility increases with depth and where the amount of clasts decreases with depth.

Finite Element Method has been the most used approach in stress analysis of rock mass in the field of rock engineering over the last decades (e.g. Cai, 2008; Mao & Nilsen, 2013). Yet, it is evident that there are several other numerical methods such as: FDM, BEM, DEM, and DFN (Jing & Hudson, 2002). The strength of FEM-based tools is in fact due to its capabilities of creating rather complex geometries and variable material behavior showing both time-independent elastic-plastic and time-dependent deformation behavior. Also, since the mesh can be defined into the last detail and is presiding over the entire domain, it is possible to control the accuracy of the numerical solution quite easily.

The finite element method was suitable as the numerical equation solver for the experiments of the thesis, and was therefore chosen. As discussed, the modelling of weakness zones comes with potential needs for defining complex material behaviors and geometries. The same goes with the geometry of the excavation, which is not necessarily circular. Lastly, since finite element method has been vastly used in the field of rock engineering over the decades, it implied that there was a great deal of experience developed using this tool for rock engineering purposes.

5.2 The choice of numerical software and verification of the numerical models

This section is dived in two. In the first subsection the choice of numerical software is presented. After this a brief discussion over the possibilities for verification of the numerical models produced in the thesis work is done.

5.2.1 The choice of numerical software

RS2, earlier known as Phase 2, was chosen as the numerical software of the thesis experiments. According to Mao and Nilsen (2013), the commercial software package RS2 made by RockScienceTM is a popular choice when a 2-dimensional FEM-analysis is to be conducted in the tunnelling industry. It is a software that have been used for decades due to its user-friendly GUI. Also, it provides a complete library including: commonly used failure criteria, a variety of bolts and liners to be used in modelling of security measures, and much more. Since it have been used frequently in the industry, it is also much material regarding its performance and limitations (e.g. Cai, 2008; Mao & Nilsen, 2013). This also implies that there have been many contributors to verify its modelling performance.

5.2.2 Verification of the numerical models

There was not found any analytic method to verify a theoretical model with one weakness zone within the range of influence of the tunnel in the specialization project (Kaasbøll Andresen, 2021). The reason for this is simple. It is hard to do a general verification of a model in where there is no known analytical solution. Furthermore, since the thesis is completely theoretical, it is not possible to verify it by comparing its performance to known practical cases. However, RS2 has been verified compared with many analytical solutions, for instance Kirch, GHB, and MC for isotropic, continuous rock mass without weakness zone (RocScience, 1989-2021). There are many more theoretical verifications in this document for a variety of geotechnical and rock engineering problems.

5.3 Factors regarding the development of the automation script

A sensitivity study using RS2 quickly revealed some impracticalities in the beginning of the development of the experiment setup of the thesis. RS2 is neither designed for batch running files nor for sequencing through these files to do repeatable operations. The only part of RS2's interface that supports batch running of files is RS2 Compute. The reason behind the lack of batch functionality is unclear. One reason may be that the RockScienceTM's user base has not asked for more batch functionality. Furthermore, RS2 is GUI-based and does not support instructions given from scripts. Therefore, to create a FEM-model using RS2, changes in material parameters, changes in geometry of the geology, creation of discretization and mesh, execution of calculation, and interpretation of data, must be done explicitly by the user with keyboard and mouse.

This section has two subsections: (a) an investigation of approaches for development of the script, and (b) a few notes regarding the implementation of the final approach.

5.3.1 The investigation of approaches of development

In this subchapter there will be presented three approaches which was investigated during the thesis work. The last approach presented was the one that became implemented.

It was decided to investigate the possibilities of automation of the modelling process of RS2 using third-party programming. RS2's lack of batch and scripting functionality limited the possibilities of the design of the sensitivity study. It is limited how many models that are possible to make by hand. Scripting enables batch solutions, where rules for small changes in the model construction can be defined in advance. To elaborate, assume the construction of ten models. The only parameter that is differing the models are the overburden. With scripting it would be possible to ask for 10 models and define a rule for the implementation of the overburden. An example of a rule could be that the overburden increased with 100 meters for each model.

The modelling process was subdivided into five stages: (a) creation of the template models, (b) implementation of geometry alterations, (c) calculation, (d) gathering and storage of tunnel periphery data, and (e) processing and representation of tunnel periphery data. In stage (a), all parameters defining parameters that are shared by all models of the sensitivity experiment are defined. For example, the definition of tolerance and number of iterations of the defining the rules of the calculation. In stage (b), the parameters unique to a model is set. These parameters are given by the sensitivity experiment setup. An example of a parameter is the angle of a weakness zone. Also, the discretization and mesh are defined in stage (b). In stage (c), the calculation of every model of the sensitivity study are executed. In stage (d), the gathering and storage of tunnel periphery data is done. A note, the results are by default represented in contour plots of, e.g., principal stresses or strains. So, instructions were defined to only store the values around the tunnel periphery. In stage (e), the data processing and data representation is done. It was stage (b), (d) and (e) where it was possible to automate the processes. There are two reasons for not to automate stage (a). Redefinitions of the template models happened seldom, and there was need for high degree of flexibility in the testing of these templates. Automating this stage was therefore not necessary and would only slow down the process. Stage (c) is not automated since RS2 Compute already provide for batch functionality.

Two approaches to learn from

Two solutions to the development of the automation script were found. The most efficient of the two solutions was scripting using API. However, RockScienceTM had not developed API for their software. The API of RS2 would give access to call its functions from a script. For instance, it would be possible to execute commands for changing weakness zone size, rotation of the zone, construction of mesh and more, using python script. This was a setback in the development process, since no knowledge of the specifics of RS2's implementation is needed with this solution.

The other possible solution was to implement the application of an automation package called PyAutoGUI developed in the language of python (Sweigart, 2014). This solution is more tedious and laborious compared to the solution using API. The idea was to record the operations used when making a model by hand and then translate it to a script. Loops makes it possible to iterate over the unique input parameters and make a model for each specific case. This method leaned on the idea of defining a set of template models, one for each overburden, and to implement changes on copies of those templates. Also, it was decided to vary the material parameters in the template model since the geometrical features were deemed most interesting to include in the sensitivity study. In this approach all changes of geometry, stage (b), were attained using the tools in RS2's graphical interface by scripting each command resembling the behavior of the mouse and keyboard operations.

However, PyAutoGUI processes showed to be more problematic than anticipated. The reason for this were two folded.

First of all, when the number of actions controlling keyboard and cursor movement increased, the risk for the script to crash increased. The reason for this instability lies in the nature of the PyAutoGUI module. Computers share an important trait with the human being; it cannot multi-task. However, unlike the human being, it is rather good at doing an enormous number of operations fast in a successive manner. Consequently, one command must be completed before the next is initiated or else the program will likely crash or behave unexpectedly. Thus, problems will surely arrive if a command is tried executed before the predecessor is fulfilled.

A PyAutoGUI based script must be tailored to cope with the computers lack of multi-task functionality. Operations done with a mouse and keyboard is directly linked to the computer, making the chain of communication short. In comparison, Auto-GUI based scripts has a longer communication chain. This chain begins with a command given in the script. The PyAutoGUI module is itself a python-based script, which consists of several packages. Each of these packages also consists of several packages and so on. The overall module connectivity can be depicted as a tree spanning out branches, where each branch has its own branches. So, several packages are communicating before the communication with the operative system has been initiated. This system of information makes the communication from the automation script to the OS slower compared to the communication from keyboard and mouse, which is seamlessly intertwined with the functionality of the computer.

Thus, it is possible for the script to send the next command in line before the previous command is executed by the machine. If too many commands are given before execution of the previous ones two scenarios may happen. The first scenario is that the interpretation and execution of the commands gets interrupted in a way leading to latter commands being executed before preceding commands but the computer do not crash. This leads the automation script to behave unexpectedly. The other scenario is that the accumulation of commands reaches a point where the computer no longer can interpret the commands and the automation script crashes.

Therefore, it was necessary to make small delays in the script by implementing artificial pauses. The purpose of the break is to give the computer enough time to interpret and execute a command before the next arrives. This stabilizes the code. However, the length of each artificial break could only be defined by trial and error. Changes of the length of the artificial pauses was tested by running the script multiple times until the script was stabilized. A too long break increases the execution time with no benefits. A too short break increases the risk of program crash. This process of setting the artificial pauses was slow. Also, a setup of artificial pauses could work one day for so not work the next. Furthermore, the need of artificial breaks varied from one pc to another depended on the quality of the specs of the computer at hand. The faster the computer the shorter the breaks. To conclude, the risk of accumulation of unexecuted commands increased when the number of PyAutoGUI-operations increased and decreased with a faster computer.

Second of all, RS2's implementation was problematic when used in conjunction with PyAutoGUI. The example presented below is included for two reasons: Both to show the limitation due to the software and to give some insight of the workflow of the modelling process.

The main problem was to manage interactions between the script and RS2's material assignment tool. There are two ways to set the material category of an enclosed area defined by boundary lines in RS2: (a) It is possible to right click on the given area, move the cursor down to assign material, and a drop down menu reveals the different material categories and the wanted category is then chosen by left clicking that category; (b) The other method is to open the window assign by for instance using the shortcut "ctrl" + "a" and choose category from the window, for then to left click on the enclosed areas you want to assign . Snapshots of the two methods is given in Figure 46: Snapshots of two methods of assigning materials in RS2. Method (a) on the left and method (b) the right..

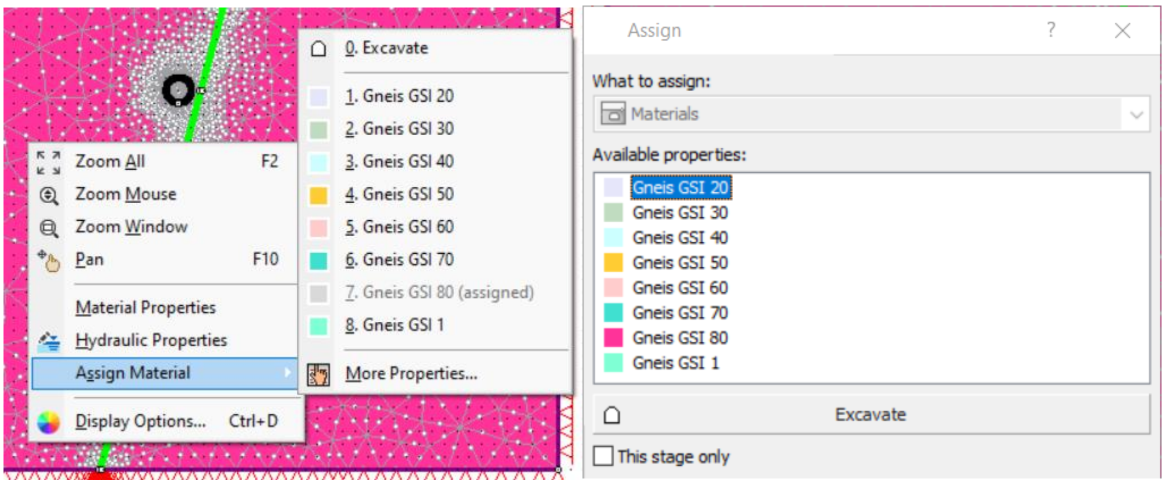


Figure 46: Snapshots of two methods of assigning materials in RS2. Method (a) on the left and method (b) the right.

Method (b) would be the easiest to implement because it leads to fewer clicks and fewer special cases of clicks. Method (a) leads to three clicks per assigned enclosed area. The following equation holds:

$$n_{(a)} = 3n_a, \tag{5.1}$$

whereby $n_{(a)}$ is total number of clicks of a model with method (a) and n_a is the number of assigned enclosed areas of a model. Method (b) lead to one click for each material assignment category given in the assign window together with one click for each assigned enclosed area. The following statement holds:

$$n_{(b)} = N_w + n_a, \tag{5.2}$$

in where $n_{(b)}$ is total number of clicks of a model with method (b), N_w is the number of assigned material categories, and n_a is the number of assigned closed areas of a model. It is evident that $n_{c,a} = 3(n_{c,b} - N_w)$. N_w is constant for a given experiment. Also, in the thesis $N_w = 3$ in all experiments. Thus, the number of clicks in method (a) increases close to three times more than in method (b) by increasing n_a by one.

In the project there was at least four and maximum six enclosed areas. Thus, if a model had four enclosed areas to assign (n_a) and three material categories (N_w), method (a) would lead to twelve clicks and method (b) would lead to 7 clicks. If there were 6 assigned areas the numbers would have been 18 and 9 respectively. This do not sound much, however, when there are several thousand models the time consumption due to these clicks accumulates. The number of assigned areas varies from model to model. The following can be defined:

$$N_a = \sum_{i=1}^{k} n_{a,i}, \tag{5.3}$$

where N_a is the sum of the number of assigned enclosed areas for each model in the experiment and $n_{a,i}$ is the number of assignments of model i, and k is the number of models of the experiment. The number of material categories, however, is constant for a given experiment. The following number series can be established:

$$N_{(a)} = 3N_a, \tag{5.4}$$

$$N_{(b)} = kN_w + N_a, \tag{5.5}$$

where $N_{(a)}$ and $N_{(b)}$ is the total number of clicks in the entire experiment given experiment (a) and (b) respectively, k is the number of models of an experiment, and $N_w$is the number of material categories. In the experiments done in the thesis, $k = 923$. The exact value of N_a is not known, but assume that $n_{a,i} = 5$ for all models i. This gives $N_{(a)} = 3 * 5 * 923 = 13\,845$ clicks, and $N_{(b)} = 923 * 3 + 5 * 923 = 7384$ clicks. Thus, $N_{(b)} \approx 0{,}53 * N_{(a)}$ in this case. Also, six experiments were conducted in the thesis, so there is a significant difference of the two methods.

Furthermore, from Equation (**5.4**), describing the accumulated clicks for method (a), do not account for the different material categories (N_w). The reason for this, is that the user right-clicks in the enclosed area that wants to be changed and must choose the category each time. Thus, this operation is one of three clicks in the drop-down menu used in method (a). Besides, the drop-down menu changes position when close to the periphery of the bottom and right side of the screen. This even further complicates the implementation of method (a) which demands that this changing position of the drop-down menu is accounted for. To conclude, method(a) was disregarded because of the severe accumulation of clicks and due to the drag-down menu not appearing on the same relative position each time. To remind, the accumulation of clicks is a severe problem because of the instability it brings with it as discussed earlier in this section.

However, method (b) was also problematic. The reason for this was that the assign window did not appear in the same position of the screen each time it was opened.

It became clear that Auto-GUI operations should be considered as an emergency solution and should therefore be used with care and kept to a minimum. The PyAutoGUI module is defined by the same coordinate system as the monitor of the computer. This is the only coordinate system used in the module. Thus, it is not possible to define coordinate systems relative to a specific window. So, to be able to use the assign window the position of this window must be known. However, when the assign window was opened it did not open on the same place each time it was opened. It was not found a solution to find the window after it was opened. Consequently, since the assign window does not appear on the same place each time, and PyAutoGUI does not manage to detect the assign window and create a relative coordinate system to this window, it is not possible to define where the mouse should click. Hence, the approach of method (b) did not work. To express it a bit blunt; the best way to use Auto-GUI processes is to avoid using them.

The final approach

A third approach based on the approach using PyAutoGUI was developed. The idea was to make use of the fact that is possible to edit ".fea"-files in a text editor such as notepad or notepad++. Thus, induce changes in the lines defining the file using python's Open-function was possible. The idea was to combine the Open functionality with the PyAutoGUI module. The goal was to use the Open function whenever possible, and supplement with the PyAutoGUI package when it was beneficial. This approach was the one that worked.

The default file of RS2 is a compressed file which contains 13 text-files after calculation. All of the files are written in English, which opened for the usage of I/O-manipulations. The file of interest was the ".fea"-file, since the other files only consisted of meta data or functioned as storage of the results. ".fea"-file is where all input parameters describing the project is stored, i.e., material parameters, geometric features, mesh description, stress definitions, yield and strength criteria etc. This is a dynamic file that varies in length depending on the stage of the project. When the mesh is defined the number of lines increases, when the calculation is done even more lines is added etc. Conveniently, the size of the file is constant in each stage, and I/O-manipulations can still be used.

A ".fea"-file was opened and examined using notepad++. The source file was systemized with a description of each part. This made it possible to navigate the sections of the file and understand how each section worked. The file consists of around 47 000 lines before and 81 200 after calculation. Manipulations of the file was only done before calculation and almost all of the manipulations were done on the last 2000 lines, which is the lines in where the geometry of the model is set and where the material of the enclosed areas is allocated.

The early workflow was based on trial and error on replicating a test model created in RS2 by making changes of another model in the source code. Notepad++ was used fluently in this comparison. The compare plugin was the main tool used. This tool compares two text-files and shows graphically what is similar and what is not.

It was experienced in the work with the PyAutoGUI based script that some of the operations was rather fast to execute by using keyboard shortcuts. This was the case for meshing, calculations, and for the data processing and gathering. If a function lacked a shortcut, it was often possible to customize a shortcut for it. Also, it was experienced that it was not as easy or possible to create the mesh, do the calculations, and to the data processing by the I/O-manipulations executed with the script. Thus, it became evident that it was most efficient to make a hybrid script, that was using both approaches combined. This approach was believed to be the most efficient and easiest to develop given the need for a stable and reliable code.

5.3.2 Implementation of the automation script – a few notes

The workflow of the development of the script followed the workflow of the modelling process of RS2. It began with; a) the geometry constructions and material allocations, followed by b) the implementation of mesh construction, then c) the calculations, and was finished with d) development of the data processing, data storage and data visualization. The reason for the separation of part a) and b) was that a) was attained using I/O-manipulations where b) was done by opening RS2 from the script and running keyboard shortcut commands using PyAutoGUI. The development of c) and d) was similar to b), except that the programs opened were different and with different commands.

In the following, a brief description of each stage will be presented. The general goal of the implementation of the script was to make all stages together in one script. This means that the experiment starts by clicking run, and the entire experiment is finished by the execution of this script.

Geometry construction and material allocations

The script begins with reading which parameters and in which sequence these parameters is to be changed. Then, the filenames are created, which contains the information of the specific changes of a given file. An example of a filename is: "S_rm80_ws20_k1_ob100_t2_a45_d3". S is referring to the circular shape of the tunnel, rm80 refers to a rock mass with GSI 80, ws20 refers to a weakness zone with GSI 20, k1 refers to a stress ratio of 1, ob100 refers to an overburden of 100m, t2 refers to a weakness zone thickness of 1m, a45 refers to an angle of 45^{o}, and d3 refers to the shortest distance between the center of the tunnel and the weakness zone. This was the filename template used in the thesis. This filename structure was practical in two ways. It made it easy for the user to differ the files, and it was used to tell the script which changes to be done with a specific file.

After this, the script creates all the ".fea"-files used in the experiment, by copying the relevant template files and assigning their respective filenames. There was created one template file for each unique experiment setup. In the experiment, two template files were created. One for the overburden of 100m and one for the rest. The reason for this was that the outer boundary was a square of size 150m for all overburdens except the one with overburden of 100m.

When all the files with their respective filenames are created, the script loops through each file and implements the changes given in the filename in the order from left to right. It was the changes related to the geometry of the weakness zone that was the challenge to implement. The reason for this is that all the geometries in RS2 is defined by points. Unexpected behavior would occur if just one point was not placed correctly. An incorrectly placed point would lead to openings in the geometries merging to geometries together. It could also lead to unwanted additional closed geometries. Both cases lead to unexpected behavior where the material allocation did not work as intended or that the file could not be opened by RS2. The implementation of the geometry alteration was the most demanding part of the development of the script. It was challenges in the implementation of rotation of the zone, scaling of width of the zone, and translation of the zone. Figure 47 shows an example of how the model alteration should look like.

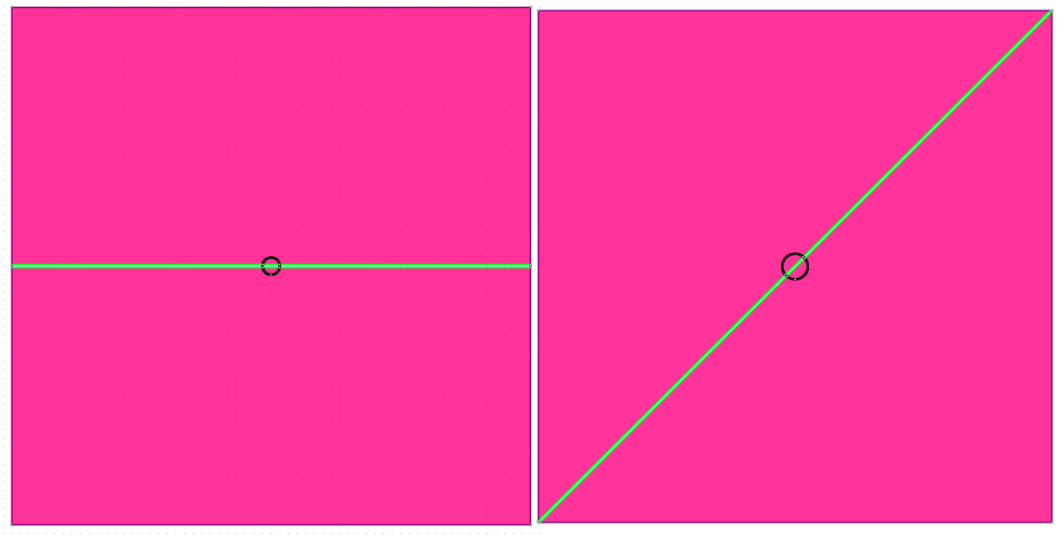


Figure 47: Two examples on how a successful model creation in RS2 should look like.

Figure 48 shows some examples of some interesting geometry alterations in where it was wondered if a pursuit as an artist would be more suitable for the author. To say, there were many more examples, but they were neither picturesque enough nor intriguing enough to be shared.

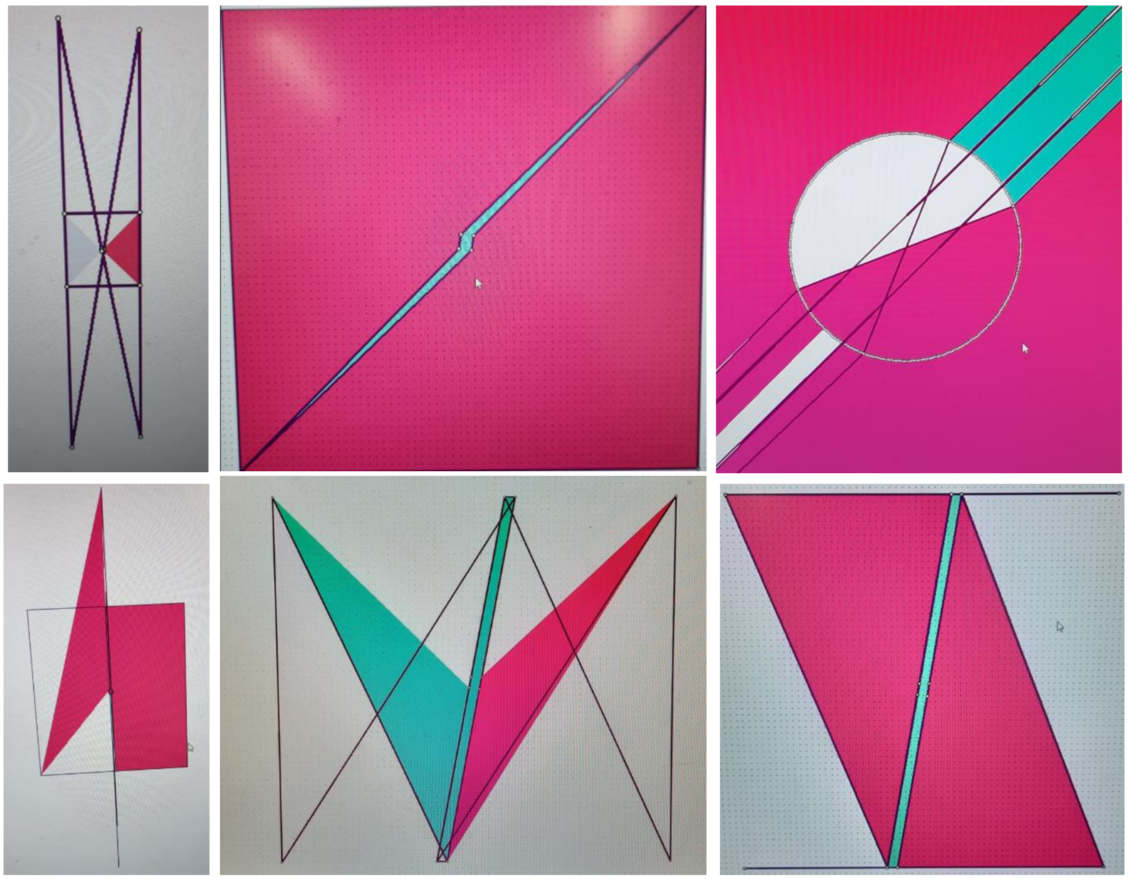


Figure 48: Six examples of unsuccessful implementation of geometry construction is here presented. RS2 is very sensitive regarding of the positioning of the points defining the geometries.

The reason behind these challenges was that the structure of the ".fea"- file was unknown. The only way to gain this knowledge was by testing using notepad++'s compare functionality. First, an alteration of a model was done by hand. Then, this alteration was tried replicated using the script. Finally, the two files were opened in notepad++ and compared. This process was repeated until the script could replicate the handmade model with no mistakes and for all categories of changes relevant for the experiment. This was tedious work and loads of testing too. Many mathematical operations and functions were necessary to be defined in the script to be able to make the proper changes. Especially in the transformation of geometry. Furthermore, the core of the problem was to ensure that there was no leakage in the geometry which would lead to random behavior as depicted in Figure 48.

Mesh construction

After the script has defined the geometries and allocated the materials, it iterates over the models defining the mesh. The specifications of the mesh were set in the template files. Thus, all the models had the same mesh since all were copies of the template files. The process was rather simple to implement. For each iteration, a model is opened in RS2 using the Popen-function from the subprocess module. Then a keyboard shortcut command linked to the mesh creation is executed followed by a save-file command. Then the software is closed. This process was repeated until all the files created had a mesh defined. In Figure 49 an example model with mesh is presented.

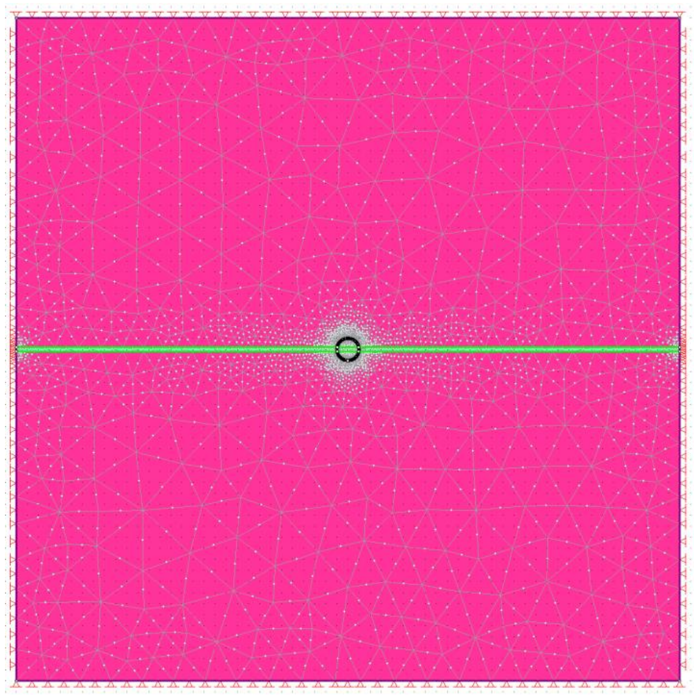


Figure 49: A model with mesh.

The calculation

When the mesh creation script was finished the calculation functionality was implemented. The concept is rather simple. The program opens the RS2 Compute, allocates the path in where the models are stored, opens them into the software, for then to start the calculation process. After that, RS2 Compute does the rest.

Yet, there was no immediate way to make the script understand when RS2 Compute were finished with its calculations. The consequence of this problem was that it was not possible for the script to know when to initiate the next command since there is not possible to establish a direct connection with RS Compute. The calculation time of the script depends on the number and the complexity of the models. It was therefore not convenient to use an artificial break, where the length of the break must be known in advance. The solution was to get the processor usage data using the Shututil module. RS2 Computes' processor usage were over 20%, as long as the calculation process were still going. When the calculation of the last model was finished, then the process usage dropped down below 1%. This attribute was used in an if/then/else-statement. The script would go to the next section when the CPU-usage dropped below 1%.

Due to the high number of numerical models, it was hard to check the quality of each them. RS2 Compute creates a log file for each computed model. This log file consists of all the calculation stages of the model in where the accomplished lowest tolerance is given for each stage. Knowing this information is useful in the determination of the quality of the calculation. If a model shows a tolerance that is long away from reaching the tolerance, it indicates that there may be a problem of the model definition and the user should investigate this model more closely. However, open the log file over 3000 times is not ideal. The solution was to let the script scan each logfile, and store the path of the files in where the tolerance was higher than the tolerance set by the user.

Data processing, storage, and visualization

The last piece of the development process was the data processing, storage, and visualization script. The data processing has two layers. The first layer is the processing done by RS2 Interpret which is directly visualized in terms of contour plots. In Figure 50 two different contour plots are presented. The one to the left is presenting the calculated sigma 1 values around the tunnel periphery, and the one to the right is presenting total deformations around the tunnel periphery. This is the two data types used in the experiments of the thesis.

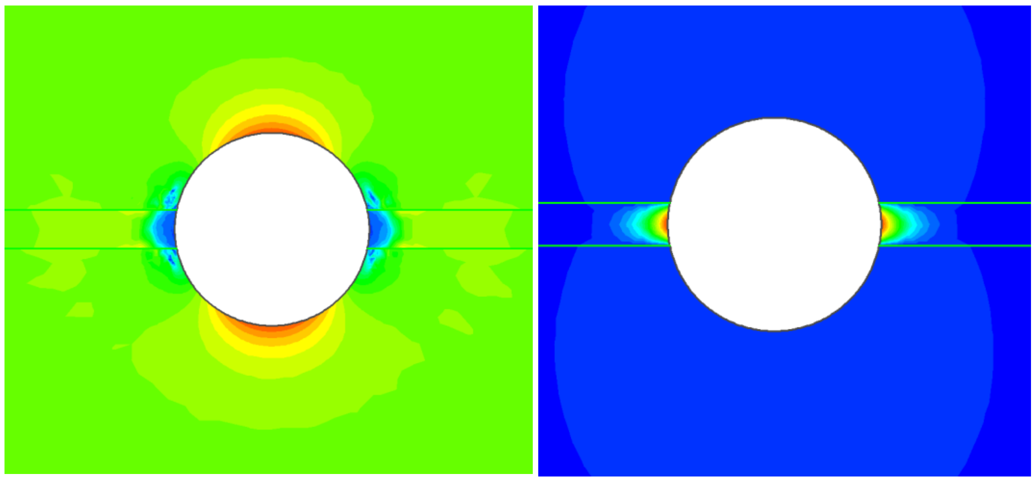


Figure 50: Examples of contour plots.

It was the data along the tunnel periphery that was deemed interesting in the thesis. Therefore, the script had commands to open each model in RS2 Interpret, construct an excavation query along with the tunnel periphery, and store each dataset in a csv-file by extracting the copied query from clipboard. It was one csv-file for each model. The values selected was: (a) the maximal deformations with the accompanied stresses, and (b) the total deformation and stresses of the intersection points between zone and tunnel periphery. Value selection (b) was only relevant when the entire thickness of the zone was crossing the tunnel. Between 4 and 12 values was fetched from each csv-file and stored in another csv-file. The contents of the last file were the one to be plotted. The data produced in the thesis is to be found in NTNU Open, stored in a zip-file.

Finally, a system for visualization of the data was developed. Python was first chosen. However, it became clear that it did not work as intended. The reason was the implementation of the Matplotlib module. With this module it was only possible to define changes of the charts with the script. Thus, it was not possible to make changes to the charts after the charts were created. This made the process of data visualization static and slow. It may be a module or another solution to enable dynamic charts, but it was not found.

It was decided to use excel, which does have dynamic charts. However, excel has problems when it comes to creating charts based on large datasets that is grouped in several subsets. Particularly, when the subsets are dynamically changing in size according with changes of the experiment layout. To elaborate this problem an example is created. Assume the steering system of a type of drone is under development. Two drones are tested simultaneously with a wind of certain magnitude coming normal to the flight path. The position of the drone is sampled every millisecond and is to be monitored in real time. A table related to this example is given in Table 6 and the chart based on this table is given in Figure 51.

For every sample, a column is added to Table 6. In this table the last column is colored with green to emphasize that this column just has been added. Since the data must be monitored in real time, and the sampling frequency is short, it is necessary for this to happen automatically. This is a behavior that could be implemented in the excel sheet quite easily using Power Query.

In Figure 51 the chart before the sampling of column 11 is on the left, and the updated chart with column 11 is on the right. This inclusion in an excel chart is, however, not possible to do automatically. In other words, it is no easy way to include the data from the new column of the table in the groups "x, drone a" and "x, drone b" as defined in the chart without doing it by hand. This is a problem relevant for the experiment in the thesis, in where small changes of the experiment setup can redefine the ranges of the group in similar fashion. There is indeed no demand for real time updates, but, due to big datasets, changing the ranges manually is too time consuming. Especially, when regarding how hard it is to update the datasets given Excel's implementation.

Table 6: Imagined position values of two drones used to analyze its steering system when an artificial wind is blowing in x-direction. This is used as an analogy to clarify the need of VBA to automize the construction of charts in excel.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
x, drone a [cm]	0,42	0,32	0,33	0,31	0,23	0,29	0,39	0,41	0,49	0,42	0,28	0,58
x, drone b [cm]	0,72	0,62	0,63	0,61	0,73	0,59	0,79	0,81	0,89	0,82	0,98	0,78
y [m]	0,00	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,0	11,0

The answer to this problem was Visual Basic, which is Microsoft's very own scripting language. This language became central in managing datasets of the size in this project. VBA scripts, called macros, made it possible to make the excel sheets behave dynamically. The term dynamically is here referring to excel being able to detect changes in the dataset

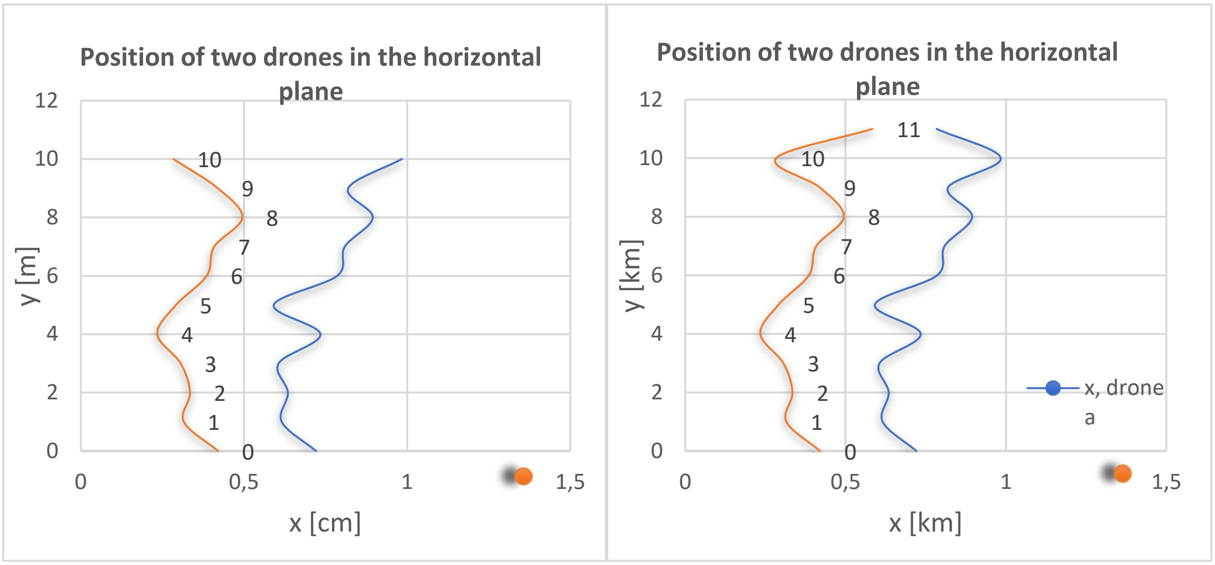


Figure 51: This is used as an analogy to clarify the need of VBA to automize the construction of charts in excel.

and its subsets, and implement those changes into the different charts. The excel workbook is implemented to be directly linked to the CSV-files in where the datasets are stored. Thus, it is only necessary to refresh the excel workbook and change three variables to update the changes due to changes of the experiment. Visual Basic was not known by the author in advance. However, excel enables recording of the commands the user does in the interface of the workbook and stores it in a macro-script. This resulted in a swift learning process. Also, VBA is frequently used with a large user base which meant that it also was easy to search for many answers in where many posts their macros open source and shares their work.

The final solution proposed was successfully implemented and worked as the cornerstone for the experimental method of the thesis. Consequently, this script enabled the creation of thousands of numerical models in matter of a couple of days. Also, the relative short calculation time made it possible to use it as an integrated part in the development of the method, in a dynamic trial and error-based workflow.

6 Aspects regarding the sensitivity study setup

The purpose of this chapter is to discuss the choices made regarding the different aspects of the sensitivity study setup. The chapter is divided in three sections: (a)

An important note, all the numerical models presented in the following chapter are based on the parameters given in Chapter 3: Parameter Study Setup.

6.1 The choices of sensitive parameters

In this section the different aspects leading to the final experiment setup is presented.

First, to repeat from the theory section, parameters deemed to be investigated in the experiment of the master thesis were:

(a) The strength and stiffness of host rock.
(b) Failure criteria for host rock
(c) The stress distribution.
(d) The strength and stiffness of weakness zone.
(e) Failure criteria for weakness zone.
(f) Number of weakness zones.
(g) Width of weakness zone.
(h) Orientation of weakness zone.
(i) Position of weakness zone relative to tunnel

The goal of the development of the experiment setup is to find something fundamental regarding how the weakness zone alters the stress distribution close to tunnels, and how this affects the tunnels stability. To achieve this, it was decided to take a bottom-up approach in where it is necessary to investigate parameters believed to be more vital. In In his article, Basarir (2008) argues that momentum forces needs to be included to be able to predict appropriate security measures. Thus, it is important to gain the details of the deformation distribution. If the distribution gets skewed, this indicates force moments in the rock mass. Thus, in a bottom-up approach, it is imperative to exclude parameters that overshadows the effects the weakness zone has on the deformation distribution around the tunnel alignment. The gathered knowledge of the idealized case can be used as a tool to understand more complex practical cases.

Høien (2018) investigated the effect the thickness of weakness has on the total deformation along the crown of a road tunnel, which showed that the deformation increase was proportional with an increase of thickness. Thicknesses between 1 and 50m were modelled. Furthermore, it was also seen that the shape of the deformation curve was more or less the same. This behavior was prominent for the modelled overburdens of 100 and 500m respectively. This gave enough indices to develop a hypothesis that there is something fundamental with how the geometry of weakness zones affects the stress distribution. In this experiment the strength of the host rock and weakness zone were held constant. Both materials were assumed to behave according to the Generalized Hoek-Brown failure criteria which implies a material that can be assumed continuous, as depicted in Figure 24. It was also assumed perfectly elastic-perfectly plastic material behavior for both rocks, homogenous stress field, and both rocks consisting of isotropic materials. In other words, the model resembles a relatively ideal geology with little variation in behavior.

Brady and Brown (1993); Donath (1972) presentation of triaxial tests on rock samples with a distinct weakness plane, as seen in Figure 25, indicates a situation where the strength of the rock mass is highly dependable of the orientation of the weakness plane. If the simulated rock mass was to behave transversely isotropic this would lead to a model in where the effect of the weakness zone would be dependable on the orientation of the weakness plane. This would lead to a situation in where it would be hard to be sure what parts of the deformation distribution that can be traced back to the existence of the weakness zone alone, and not to the combined effect of joint orientation and the zone. Also, failure criteria go under the same category. A criterion that depicts anisotropic failure behavior leads a preferred failure direction and do therefor contribute to mask the weakness zones contribution to the alteration of the deformation field.

The same argument holds for the induced stress distribution. A complex stress distribution with gravitational-, tectonic-, residual-, and terrestrial stress components (Amadei & Stephansson, 1997), often leads to an uneven and anisotropic stress distribution. Figure 31 presents stress-ratios across the globe, which indicates that there is normal with stress anisotropy in where the variance decreases with depth. However, a complex stress distribution would make it hard to analyze the weakness zones contribution to the altered stress state.

In conclusion, the following parameters were chosen to be varied in the numerical experiments of the thesis:

(a) The isotropic gravitational induced stress distribution.
(b) Width of weakness zone.
(c) Orientation of weakness zone.
(d) Position of weakness zone relative to tunnel

Thus, the sensitivity study is predominantly investigating how the geometrical features alters the stress distribution, and how these geometrical features interact with gravity induced stresses.

6.2 The experiment layout

In this section, the choices regarding the experiment setup and the data processing of the relatively big dataset gain from the experiments of the thesis is discussed. There was done several test-runs of the experiments to make sure that the values of the parameters captured all the traits of the final plots, which resulted in the values given in Table 1.

In Table 1 it can be seen that the chosen parameter study setup leads to a total number of 3696 models. Each of the models is on its own a rather big dataset, containing thousands of output-values of several parameters based on the calculation of the given set of equations.

It was decided to divide the parameter study into four smaller experiments; one for each thickness, see Table 1. It was experienced somewhat instabilities with the developed script when the dataset became large. Therefore, it was decided to categorize the experiment based on the sensitivity parameter with the fewest values, and which already had been exanimated in some extent before, by Høien et al. (2019). Thus, for each experiment there are three values to be varied: (a) the overburden, (b) the zone angle, and (c) the position of the zone.

Høien et al. (2019) focused on total deformations and the critical strain concept proposed by (Sakurai, 1981) in where he back-calculated the strain response of several known tunneling projects. In the article the strain responses was compared to the proposed 1%-critical strain limit confirmed by the works of Hoek and Marinos (2000), see Figure 28. It was a useful approach in where it was shown that most weakness zones, in where it was installed heavy security measures, did not exceed the 1% percent limit. It was shown in Figure 29, by (Hoek, 2001), that only one incidence of instability was experienced with a critical strain lower than 1%.

It was therefore decided to use this approach in the master thesis also. It was observed during test-modelling that the highest total deformations occurred within the weakness zones, normally with only one peak. Therefore, it was decided to use the maximum total deformation of each model to create the charts. One of the advantages of this approach is that it is possible to reduce the values gathered from each model of the experiments from 360 values to 1 or 2 for each model. There are 2 values only if the entire zone crosses the tunnel. This is a great reduction in values, given that the total number of models used in this thesis is over 3696.

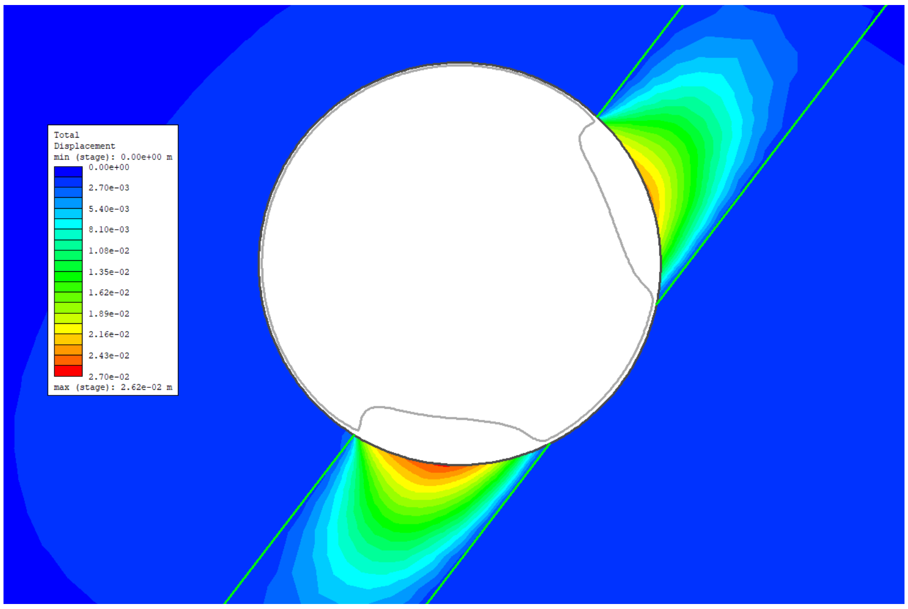


Figure 52: Example of deformation distribution of one of the models created in the experiments of the thesis. Observe the skewness of the deformation envelope, indicating a force momentum that would be developed if security measures were installed.

It was also during test-modelling observed that the weakness zones of the models lead to skewness of the deformation envelopes along the rim of the tunnels. An example of one model created in the experiments of the thesis is presented in Figure 52. It is therefore suggested to analyze the skewness of the models. The same dilemma was occurring in this situation as well. It was imperative to reduce the number of values attained from each model, to be able to make charts that are meaningful.

It was investigated if it was possible to calculate force moments as defined in static mechanics, given the equation:

$$\boldsymbol{M} = \boldsymbol{r}x\boldsymbol{F}, \tag{6.1}$$

where $\boldsymbol{M}$ is the momentum about a given point, $\boldsymbol{r}$ is the distance from the point of rotation, and $\boldsymbol{F}$ is the force exerting perpendicular to the distance vector $\boldsymbol{r}$. A graphical representation of the idea is presented in However, it became evident that this approach was too hard to define. The idea was to define the point of momentum where the change of strain was higher than a certain limit, and that the calculation stopped when the difference in strain went below that same limit. This idea is presented in Figure 53. The goal is to find the resultant force R and the arm s, which rotates about the point given in the top left corner, and about the bottom right corner, where $\epsilon(s)$=0. The arm s is here given by the arc length measured from the two points where $\epsilon(s)$ becomes zero. If there are skewed loads, the two calculated force moments should be different. This is depicted in Figure 53, whereby the arm s to the resultant force R is longer about the point in the top left corner, compared to the s of R from the point in the bottom right corner. I there are no skewed load, this distance would be the same. This implies that by this method there will only be two or four values left after calculation. If the difference of the momentums is plotted, this will reduce to one or two values. There are four values, only when the entire zone crosses the tunnel alignment.

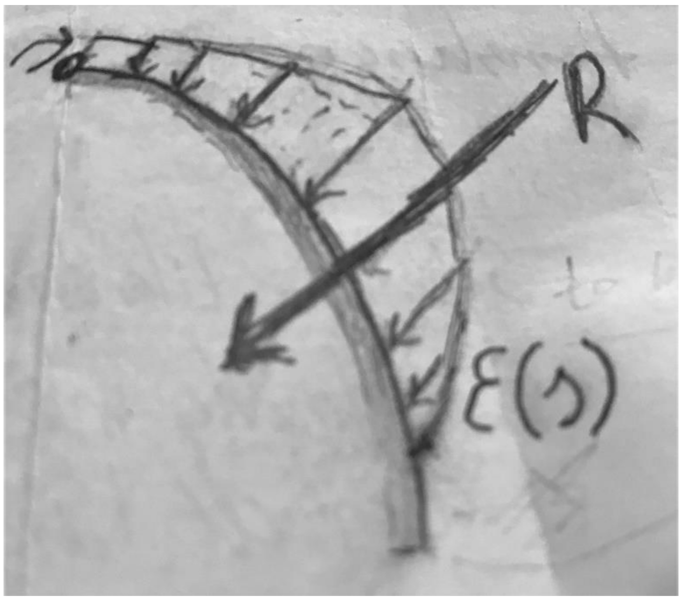


Figure 53: Graphical representation of the force moment calculation.

Following assumptions was made to create a momentum equation given by the arc length: (a) tunnel is cylinder with infinite extension along the tunnel axis, (b) the forces are always normal to the curvature of the tunnel, which implies that the resultant force intersects with the tunnel center, (c) the rock mass is linear elastic, and (d) there are only static force interactions. This gives the following equation:

$$M = \oint \sigma(s)(s - s_0)dS \;=\; \oint E(s)\epsilon(s)(s - s_0)dS \tag{6.2}$$

The only conflicting assumption is assumption (c), in where it is assumed that the forces always are normal to the arc of the tunnel. However, investigations on the total deformation distributions of several models created in the thesis showed that this assumption could be made. An example is given in Figure 54 where the red arrows represent the direction of the total deformation distribution. It can be seen that the arrows are near to be normal to the tunnel arc at any s along the tunnel rim.

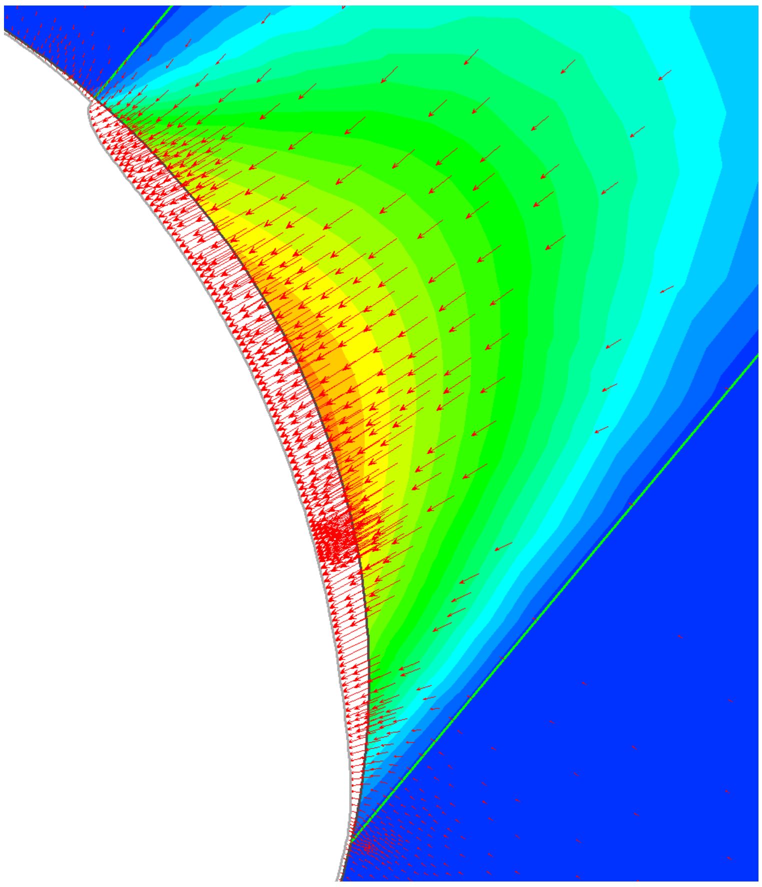


Figure 54: It can be seen from the red arrows indicating direction of deformation, that it can be assumed that the direction of force along the tunnel rim is normal to the rim.

However, there was problems to establish the equation of Young's modulus, $E(s)$, as given in Equation (6.2). Hudson and Harrison (1997) in Equation (2.41), for an idealized rock mass with one discontinuity set normal to the direction of the applied force with negligible thickness and one specific frequency, that the Young's modulus was a weighted sum of the Young's modulus of the joint and the Young's modulus of the rock. The weight was given by the extension of each entity. This is precented graphically in Figure 27. This idealized case can be used as an analogy for the case of the thesis wherein the Young's modulus $E(s)$ is a weighted sum of all Young's moduli inflicting on this points, which is altered when fracturing occurs. In other words, this process is rather chaotic, and is hard to deal with.

It was therefore suggested a non-physical approach which has been given the name deformation momentum. A hint was given in Figure 53, in where the force field has been swapped with the deformation field. The idea is simple. Instead of crossing the distance with the resultant force, it is instead crossed with the resultant deformation. It is underlined that this approach has no physical meaning. However, it will be useful to be able to indicate skewness in the models of the thesis in a way that is applicable on the relatively large dataset given by for instance the 3696 models created in the thesis.

6.3 Geometry

In this section, different aspects regarding geometry definitions of the numerical models constructed in the thesis is discussed.

6.3.1 Tunnel

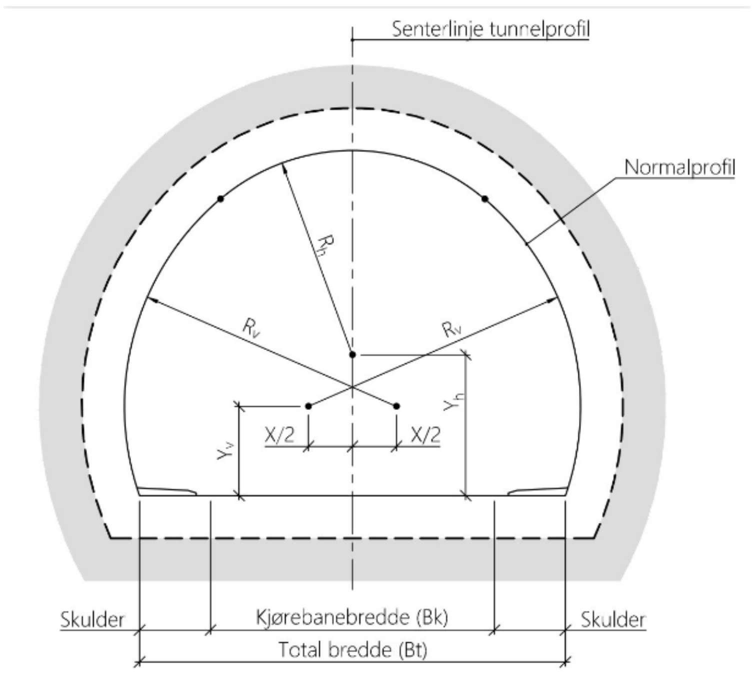


Figure 55: Example of a tunnel geometry in accordance with the Norwegian national standard N500. (SVV, 2022)

Most road tunnels are driven conventional in Norway (Bruland, 2016). According to N500 (SVV, 2022), the tunnel shall be defined by a flat bottom, whereby the walls and roof shall resemble a quasi-circular shape. This is done by defining two tunnel radii, one for the walls and one for the roof, see Figure 55. The most accurate analysis would have been by defining a tunnel geometry given by these standards, since the geometry of the excavation is an import factor considering the redistribution of stresses (Nilsen, 2016). However, the circular shape has some clear modelling practical advantages: (a) it has the highest degree of symmetry when residing in a homogenous rock mass (b) it is easier to implement when making the automation script in python, (c) it is a strong shape; so if the problem shows in these models there will most likely be problems for less optimized shapes, and (d) it distributes the stresses evenly around the periphery; to be sure that the stress alterations measured is only because of the contributions of the weakness zone. Thus, it was decided to model a circular tunnel. The radius was chosen to represent an average size in Norwegian tunneling and was therefore given the magnitude as seen in Table 2, inspired by the values of radii in N500.

6.3.2 Fault Zone

In the specialization project (Kaasbøll Andresen, 2021) the investigations on weakness zones was kept to only include faults to limit the scope. Because, it would be logical to also include the effects of deep erosion, which is common in hard rock areas such as the Norwegian. However, fault zones were chosen since it is a more frequent problem in tunneling.

Faults are often assumed to have the shape of a linear sheet, which often is the case in Norwegian-like settings, as long as the tunneling project is not close to the surface (Nilsen, 2016). For project near the surface, the shape of weakness zone often can resemble the shape of a wedge. Thus, it is assumed in the thesis that the modelled fault has the shape of a linear sheet, which again limits how close to the surface a model can be created. According to Nilsen, this wedge shape often does not go deeper than 20 meters.

The weakness zone's strike is assumed parallel to the tunnel axis in the thesis. This is the worst-case scenario in tunneling, if the zone is close to the tunnel, since it is experienced to lead to the most severe stability issues (Nilsen, 2016). Also, this assumption makes it possible to go clear of the addressed by (Mao & Nilsen, 2013). In their comparison of FLAC3D and Phase2 (today: RS2) a zone, modelled in Phase2, normal to the tunnel axis must be much thicker than the tunnel span to not deviate significantly from the more accurate 3D model created in FLAC3D.

It is assumed an infinitely long tunnel when using 2D FEM, which is the case of this thesis. This implies that effects close surface is not investigated (Cai, 2008).

6.3.3 The outer boundary

The outer boundary was given a square shape, which is normal in tunneling applications (Basarir, 2008; Høien et al., 2019). However, it is a rule of thumb saying that the dimensions of this square should be between 3 and 5 times the tunnel diameter. As seen in Table 2, the size was set to either 10 times or 15 times the tunnel diameter. The reason for this great deviation is simple. Position of the zone is one of the parameters varied in the experiments of the thesis. A weakness zone can only be modelled within the area enclosed by the outer boundaries. In an early stage of the project, it was unclear when a zone was out of the influence zone of the tunnel. Also, with increased thickness of the weakness zone and/or increased overburden expands the influence zone. Thus, the number of 10 and 15 times the tunnel diameter was chosen to be sure it was enough space. Since it is possible to control the mesh to be more course outside area of interest, this did not significantly raise the computational time.

There are two different outer boundaries defined, since it was not possible to define an outer boundary with lengths 150 meters, in the case of overburden equal to 100 meters due to how RS2 functions.

6.4 Mesh and sampling considerations

This section has two subsections. The first subsection discusses the choice of mesh. The other subsection discusses the choice of the number of points defining the tunnel, which in theory can affect the definition of the mesh.

6.4.1 The definition of mesh

In this subsection the choice in the definition of mesh is being discussed. The models presented in Figure 57 and Figure 56 are both based on the experiment setup presented in chapter three. They both has overburden of 100m, the weakness zone is centered, the thickness is 4m, and the angle is 22.5°. The angle is not included in the range of the experiment setup. The choice of the geometries is not important of this discussion and is analogous to all the models of the thesis.

Figure 56 shows the mesh defined in Table 5. The mesh of Figure 57 is defined in Table 7. It can be observed that the mesh of Figure 56 is more specified to be fine-grained only close to the tunnel. Figure 57, however, has the same gradation around the tunnel as Figure 56, but is more fine-grained over the entire domain.

Table 7: The definition of the mesh used in the definition of Figure 57. The purpose of this model is to investigate the effect of a finer mesh than the mesh used in the models created in the experiments of the thesis.

MESH - STATIC	
Mesh type	Graded
Element type	6 Nodded Triangles
Gradation factor	0.07
Default number of nodes	40
ADVANCED	
Improve discretization	Yes

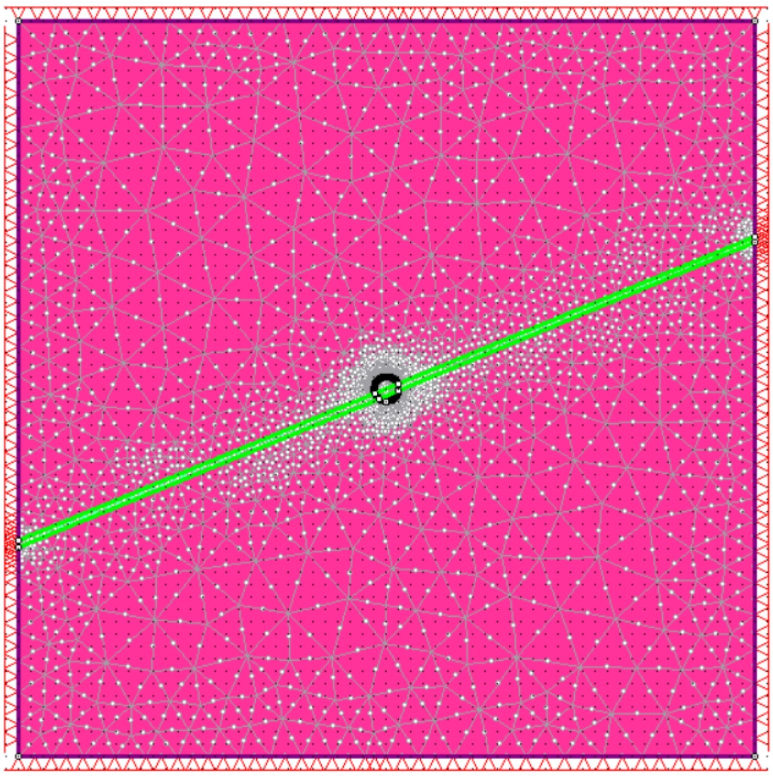


Figure 56: The mesh of the experiments of the thesis looks like this. The clustering of points is kept around the neighborhood of the tunnel.

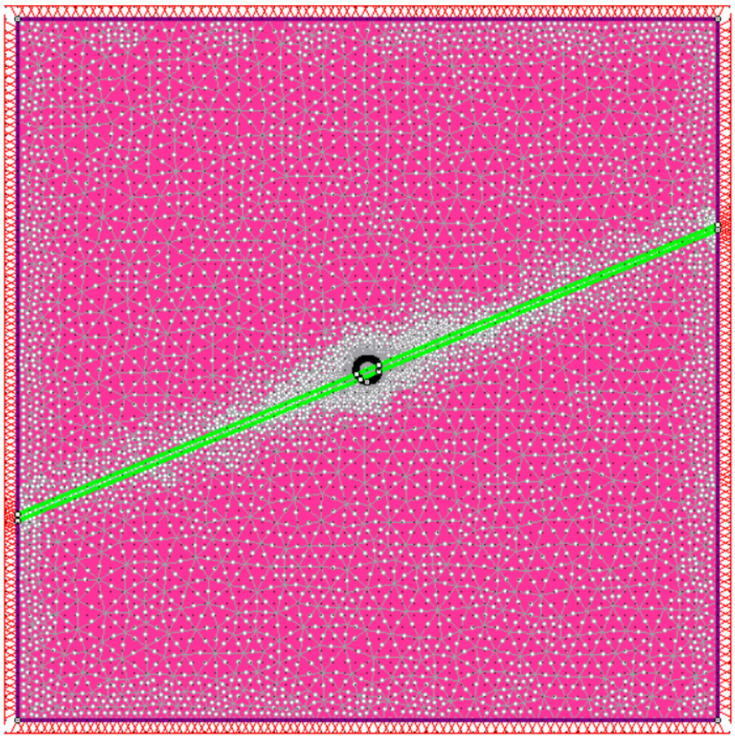


Figure 57: The looks of the mesh defined in Error! Reference source n
ot found.**. It is relatively fine-grained in comparison with the mesh**

Figure 58 and Figure 59, are both defined with the same mesh as depicted in Figure 56, except that Figure 58 is based on a 3-nodded mesh. This gives each element 6 degrees of freedom instead of 12, which again means a lowering in the quality of the approximations. The benefit would be to decrease the computational time. Both Figure 58 and Figure 59 depicts the deformation distribution shown graphically by contour plot.

In the comparison of Figure 58 and Figure 59, it can be seen that the 6-nodded mesh was found to give significant better results than the 3-nodded. The 3-nodded mesh gives more uneven and round deformation envelope (the gray line). Wrong depiction of the

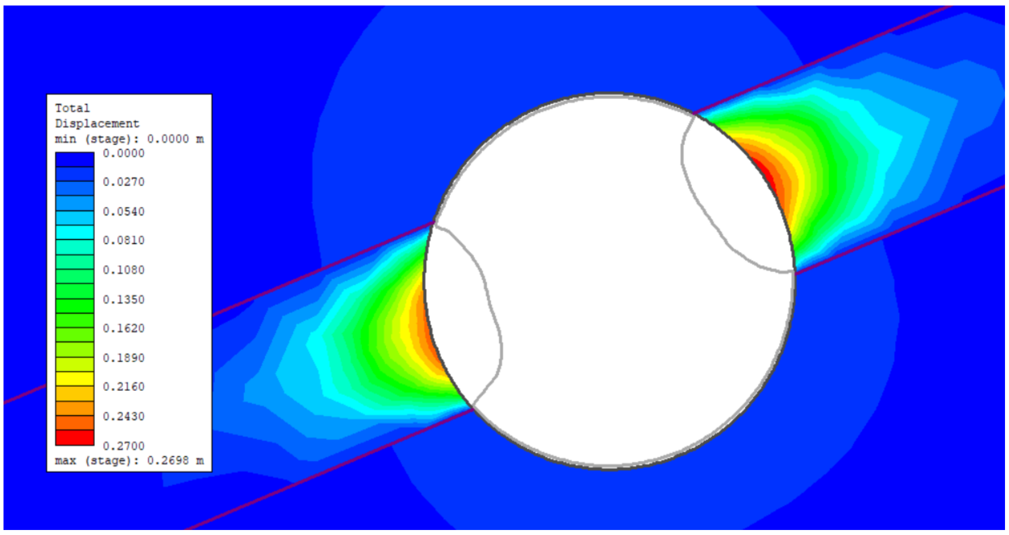


Figure 58: The deformation distribution calculated based on Figure 56, but with a 3-noded mesh. See Error! Reference source not found. **for definition of mesh. The d eformation envelope deviates significantly from the one in Figure 59.**

deformation envelope, would be a problem, since the skewness-calculations is based on this shape. Also, the total deformation is 5 centimeters less, which gives a difference of about 15%. This is not ideal, since the total deformation of each model of the experiments, which always resides on the tunnel periphery, is gathered for later to be plotted.

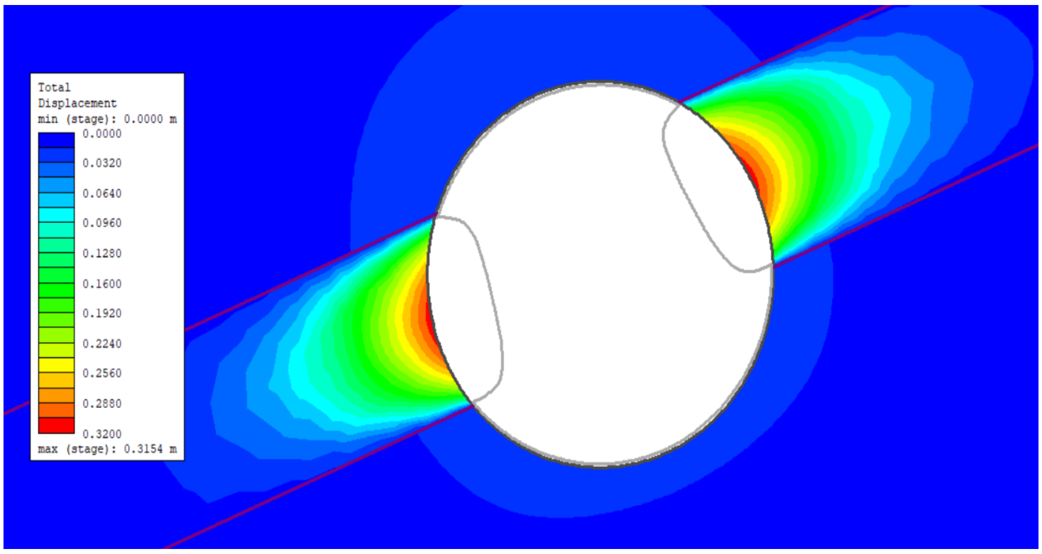


Figure 59: The deformation distribution calculated based on Figure 56. See Table 5 for the definition of mesh.

Figure 60, shows the deformation distribution around a tunnel with a centered weakness zone. However, it is based on the calculations of Figure 57, which again is based on the mesh given in Table 7. In the comparison of Figure 59 and Figure 60, it can be seen a good fit between the two deformation envelopes. Even though it resembles a more rounded shape in the tails of the deformation contour-lines, it is not important in this case. The reason for this is that it is only the values along the tunnel periphery that are analyzed. Therefore, by choosing the mesh definition as seen in Table 5, it will lead to lower computational time compared to the definition given in Table 7, without compromising of the accuracy of the total deformation along the tunnel periphery.

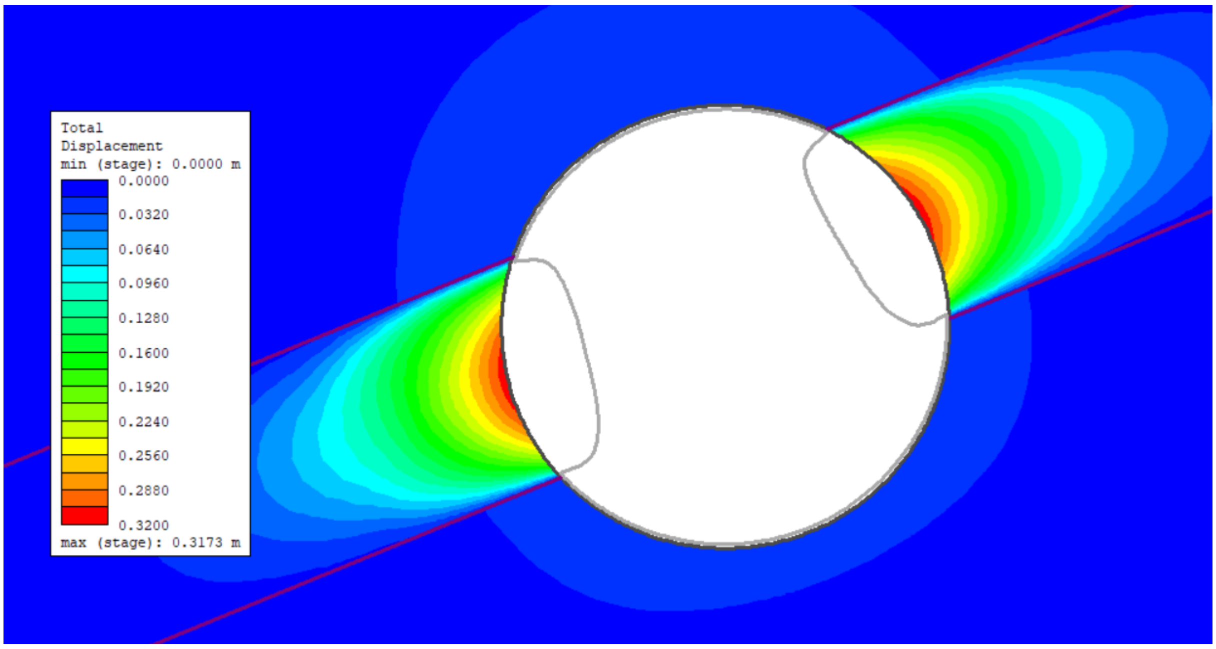


Figure 60: The deformation distribution calculated based on Figure 57. The mesh is defined in Error! Reference source not found.**. The deformation envelope is almost identical w ith the one in Figure 59.**

6.4.2 Sampling of data

In this section, a brief discussion of the data sampling of the tunnel delimiter is made.

In this subsection the sampling of data is briefly discussed. The models presented in Figure 61 and Figure 62 are both based on the experiment setup presented in chapter three. Their geometric features are the same as Figure 57 and Figure 56, except of having an angle of 45º. As the discussion of mesh the choice of geometric features are not important to this discussion.

For technicality reasons regarding the implementation of the automation script it was the excavation query function that worked best to get the data along the rim of the tunnel. However, this function only creates a datapoint for each point defining the tunnel. Both Figure 61 and Figure 62 presents the deformation distribution around a tunnel with a weakness zone positioned in the center of the tunnel. The only difference is that Figure 61 has a tunnel defined by 90 points and that Figure 62 has a tunnel defined by 360 points.

The deformation envelopes of Figure 61 and Figure 62 are almost identical. Logically, it would be vise to choose Figure 61, since it would lead to a slightly more efficient calculation of the model. However, it is imperative to get a decent number of data points in the calculation of skewness, which will be more accurate when there are more data points. Therefore, the version with 360 tunnel defining points was chosen.

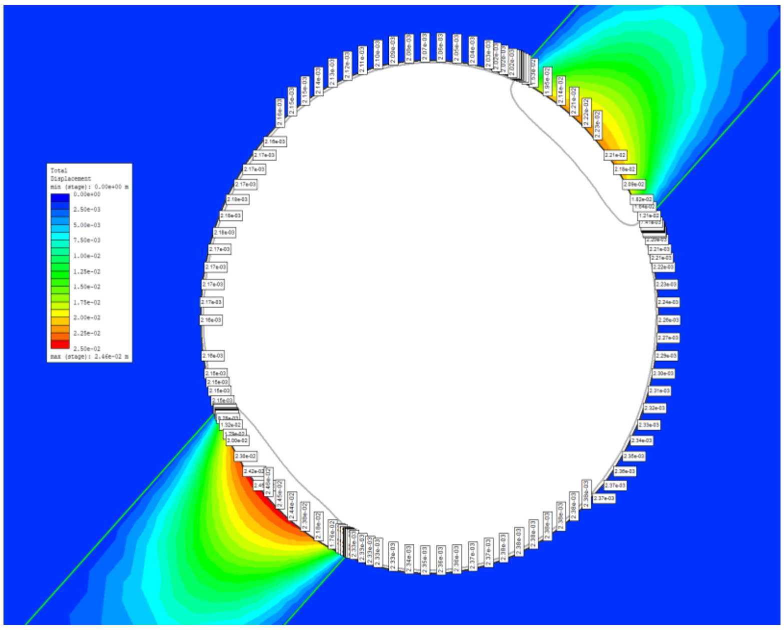


Figure 61: Tunnel with 90 points only sample 90 datapoints when using the query excavations function in RS2 Interpret. Else, it gives almost identical deformation distribution compared with Figure 62.

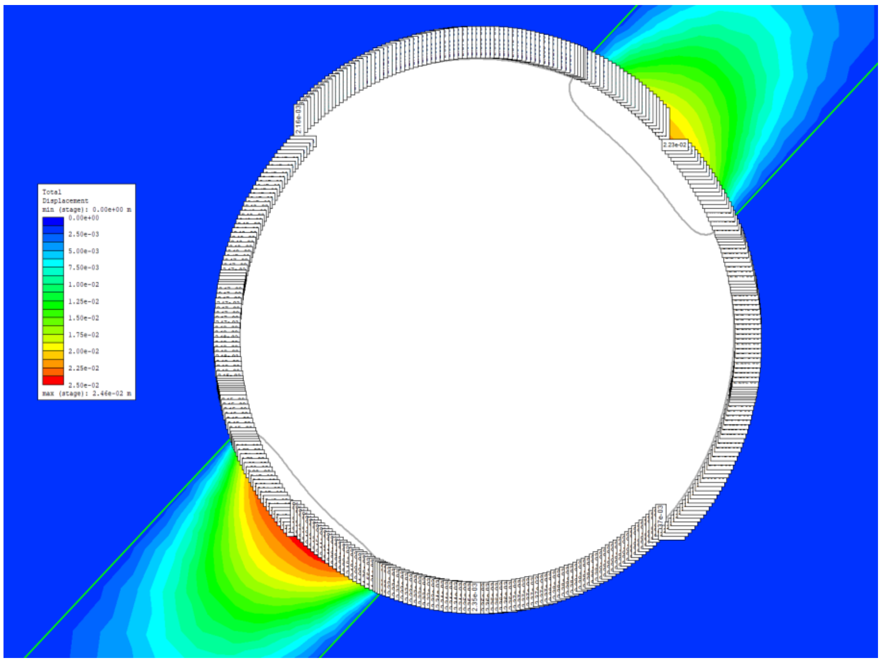


Figure 62: Tunnel with 360 points gives 360 datapoints when using query excavations function in RS2 Interpret. Else, it gives almost identical deformation distribution compared with Figure 61.

6.5 Material behaviour of host rock and weakness zone

In this section a discussion regarding the material definitions of the host rock and weakness zone of the models are discussed. It is divided in two subsections: one for the host rock, and one for the weakness zone.

The definition of the material behavior of both host rock and weakness zone was chosen to be static and the same for all models of the project. The reasons are given in section 6.1. The material definitions of the thesis are given in Table 3. To make the results comparable with the sensitivity study conducted by Høien et al. (2019).

It was assumed that the experiments were to model the deformation distribution around the tunnel just after exaction. The reason for this was to exclude the effects of time-dependent deformations, which is an important factor when considering long term stability (Hudson & Harrison, 1997). Also, this assumption is a necessity when the FEM-method was chosen, since RS2's implementation of FEM does not directly include time-dependency. However, this assumption does not come without its complications. According to (Cai, 2008), the stress distribution change when modelling a procedure of "sudden" excavation, e.g. drill and blast, the transition from the state of in-situ stresses and the post-excavation stress-state should be taken into account. This cannot be readily attained with RS2 which do not resemble time in a direct manner. However, by assuming an excavation process done with TBM solves this issue.

It was assumed drained condition, which often is the case in tunneling projects as stated by Hoek and Brown (1997) and Li (2018). The joints are assumed to be integrated in the rock mass by using the rock mass classification of GSI.

6.5.1 The host rock

The host rock was assumed to have a density of 2,7 g/cm^3, which is a normal assumption of rock mass density. It was assumed no distinct orientation of the rock mass, which implies that the rock mass can be assumed to have isotropic material behavior. This resembles the case shown in Figure 26, where Li (2018) illustrated how the deviatoric stress becomes constant when the number of rock joints of different orientation increases. This assumption makes it possible to use the Generalized Hoek-Brown to describe the host rocks failure mode.

The Hoek-Brown parameters m_b, a, and s was defined using the equation-set (2.45). It was assumed no damage zone due to the excavation process, and since the tunnel is circular, the tunnel would most likely be excavated with TBM, which is known to be a gentle excavation method. Thus, the two parameters necessary to define was the GSI and the m_i of the host rock. It was assumed a representative Norwegian granitic gneiss, which was the assumption made by Høien et al. (2019). The reason behind this choice Is its relatively low variability for both strength and stiffness. The m_i was then chosen in the chart given in the RS2 software. The mean value of 28 was chosen. The GSI was set to 80, which was the same Høien did.

The σ_c was chosen based on the values given in Figure 19 and E_i was chosen based on the data set presented in Figure 11. The mean values of the granitic gneiss were chosen for both strength (125 MPa) and stiffness (20000 MPa). The tensile strength was assumed to be 0, which would only be a good assumption if the deformation mode is controlled by the joints and fracture of the rock. It could be argued that a rock mass of GSI 80 would not only be dominated by the strength of its joints. However, this assumption does not have a huge impact since all of the models is based on the same materials, in where the study is a comparative study.

The assumption of perfectly elastic – perfectly plastic rheology of the host rock was done by Høien et al. (2019) and Basarir (2008). It is a normal assumption to do in the industry when numerical models of rock engineering projects are created, also for rock mass that show more brittle behavior. The reason for this is that it increases the chances of the model to converge, even though it is not physically correct. If the rock mass is strong, and there is little chance of it to fail, most of the rock will reside in the linear elastic part of the deformation curve, see Figure 10. This implies that the assumption can be made without great errors. This assumption means that the residual strength values is the same as the peak value.

This assumption was also made in this thesis. The reason for this is that during the test-modelling it was observed that the fracture zones around the tunnel rim was dominated by the weakness zones in most cases, and was only starting to show at high overburdens. A model resembling the worst-case scenario is given in Figure 63. In this model the overburden is 1200m, the zone thickness is 8m, the zone angle is at 60° and the zone is tangent to the tunnel. It can be seen that the fracture zone is compacted around the rim of the tunnel. Also, by studying the deformation envelope it can be deduced that the weakness zone still dominates. This indicates that the assumption of perfectly elastic – perfectly plastic material behavior also holds in the thesis. Furthermore, since this is a comparative study in where the material parameters and failure modes are the same for all models, the details of the material behavior in itself are not that important.

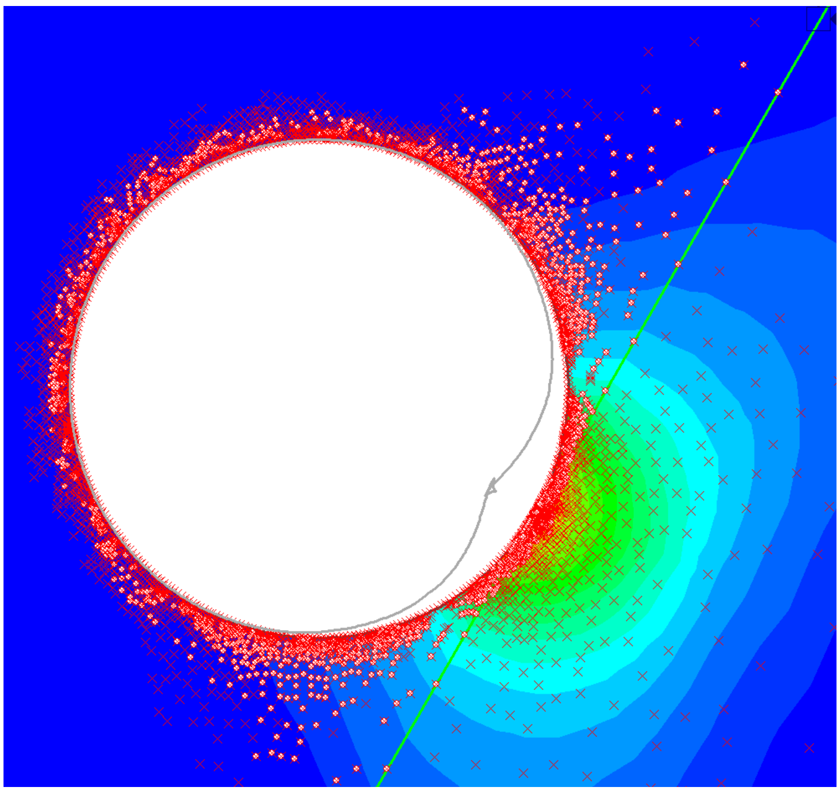


Figure 63: The distribution of fractures around the tunnel rim in a worst case-scenario model in where the overburden is 1200m, the zone thickness is 8m and the zone is tangent to the tunnel. It can be seen that most of fractures are close to the tunnel rim, and that deformation envelope is dominated by the weakness zone.

Basarir (2008) assumed no dilation, since the rock mass model was assumed to be relatively brittle rock. Thus, such a rock show little to no volume increase. It is based on assumption with no volume change when slip along intersecting continuities. The same assumption is made in the thesis since the rock is assumed to be a relatively brittle granitic gneiss.

The poisons ratio was assumed to be 0.3 which was done by (Høien et al., 2019).

6.5.2 Weakness zone

Fasching and Vanek (2011) states that only tectonical breccias and mylonite can be addressed with hard rock behavior, and that regular breccias, fault gouge, and cataclasis can only be attributed with soft rock and/or soil behavior. By assuming a weakness zone consisting of mostly fault gouge will therefore imply that the zone material can be regarded as a soil-like material, with little to no cohesion, and where the friction of the soil parameters. According to Brady and Brown (2012), the Mohr-Coulomb failure criterion is applicable, since the failure envelope most likely show linear behavior. The MC criterion assumes that the failure mode of the material is sheer, and that there is no dilation. It is therefore assumed no dilation in the material. According to ISRM (2014), the tensile stress must be assumed to be zero, if not, the assumption of inner friction is meaningless. Vermeer and De Borst (1984) states that it is normal to assume linear elastic–perfectly plastic rheology of soils. This is assumed to be the case of the assumed fault gouge of the weakness zone defined in the thesis.

It was hard to find literature on measured values of strength and stiffness parameters of weakness zone material (Kaasbøll Andresen, 2021). Therefore, the choice of strength of the fault gouge was based on triaxial tests done on fault gouge containing swelling clay (Høien et al., 2020). This article consisted of values regarding: (a) stiffness of the gouge, (b) cohesion of the gouge, (c) friction angle of the gouge, and (d) density of the gouge. The final values are presented in Table 3. Note that the residual values are the same as the peak values, due to the assumption of linear elastic–perfectly plastic rheology.

The poisons ratio was assumed to be 0.3.

6.6 Stress-inducing forces

It was assumed only gravity induced stresses in the models of the thesis. It was also assumed that the stress ratio $k = 1$ for all overburdens. The reason for these assumptions is elaborated in section 6.1.

Since it is assumed $k = 1$, it sets a limitation of which overburdens that can be modelled. Sheorey (1994) developed an equation based on the field measurements presented by Brown and Hoek (1978) which resembled that the k-value behaved asymptotic in where the k-value converged to 1 as the depth increased. Especially high k-values was attributed to near-surface conditions. Høien et al. (2019) made a chart based on equation (2.50) (Sheorey), in where the stiffness of the rock mass was varied. In the thesis, the stiffness of the granitic gneiss is 20000 MPa. Thus, it can be seen that it is possible to assume $k = 1$ for overburdens from 100 meters and below. The same assumption can be made of the weakness zone which has a stiffness of 300 MPa.

6.7 Symmetry – reducing number of models

The symmetry of the combination of circular tunnel, homogenous and isotropic rock mass, and hydrostatic stress field is evident. There are defined two models with material behavior as, tunnel geometry, and outer boundary as defined in the models of the experiments of the thesis. Both models have: (a) overburden of 100 meters, (b) zone thickness of 2 meters, (c) shortest distance from tunnel center to weakness zone of 1 meter. They only differ in the zone angle, in where the model of Figure 65 has zone angle of 22.5°, and the model of Figure 64 has a zone angle of 67.5°. The symmetry of the deformation envelopes is striking when comparing Figure 65 and Figure 64. Thus, it was decided to model angles

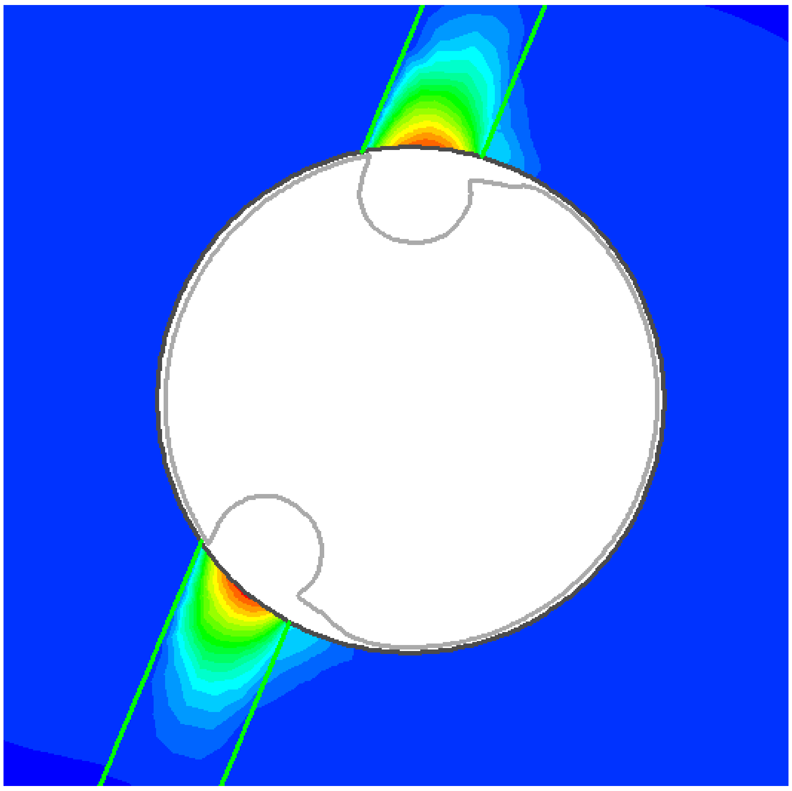


Figure 64: Deformation distribution of a model with overburden of 100 meters, zone thickness of 2 meters, zone angle of 67.5°, and shortest distance from tunnel center to zone of 1 meter.

between 45 and 90° and with only positive shortest distances between tunnel center and weakness zone.

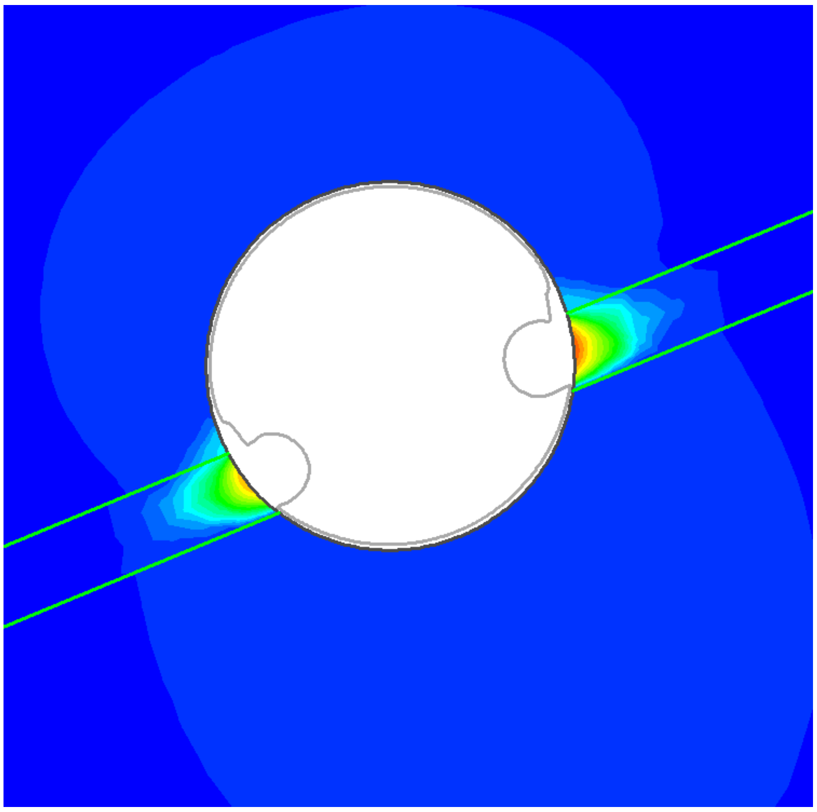


Figure 65: Deformation distribution of a model with overburden of 100 meters, zone thickness of 2 meters, zone angle of 22.5°, and shortest distance from tunnel center to zone of 1 meter.

7 Discussion of results

In this chapter, the results given by the charts of Figure 42 to Figure 45 is discussed.

It was chosen to plot the total strain ϵ_{tot} against the normalized shortest distance between tunnel center and weakness zone $\Gamma_{\mathrm{norm}} = \Gamma/\mathrm{r_t}$. The benefit of this is that both are normalized by the radius of the tunnel $\mathrm{r_t}$. This makes the charts independent of the tunnel size, which can be useful for later comparisons in where sensitivity studies on tunnels with other radii is done.

A first notion when examining the charts is that each angle cluster is quite similar to the other clusters contained in the same chart. Thus, this implies that zone angle (Θ) has little significance in the contribution on the stability for any H, T or Γ_{norm}. Indeed, there are differences for lower values of H, but the differences rapidly vanquish with an increase of T. Looking thoroughly, it can be seen that there is a slight proportionality between Θ and H for all Γ_{norm} and T, in where there is bigger variance between the angle clusters for increasing H. However, when comparing all the charts, it does seem random which angle cluster that shows the greatest increase. It is therefore believed that these variations are due to numerical noise.

By comparing each of the charts it is seen that they all show the same pattern, depicting two different exponential curves which meets in a singular point where $\Gamma_{\mathrm{norm}} = 1.1$. Also, in all charts, a rapid decrease in ϵ_{tot} is observed when $\Gamma_{\mathrm{norm}} > 1.1$. This indicates that the weakness zone quite fast stops influencing the tunnel stability, where it is out of influence when $\Gamma_{\mathrm{norm}} >\approx 2$. For all charts, except of T=1, $\epsilon_{\mathrm{tot}}(\Gamma_{\mathrm{norm}} = 0) \approx 0.5 * \epsilon_{\mathrm{max}}$. Also, it can be observed severe differences in heights of the clusters, severe differences of starting points of the clusters, but not in the width of the clusters. The similar widths are due to the fact that the zone fast leaves the influence distance of the zone.

For low overburdens the effect of zone thickness (T) is less severe. In fact, $\epsilon_{max}(\mathrm{H} = 100,\ T = 8) - \epsilon_{max}(\mathrm{H} = 100,\ T = 1) \approx 0{,}007$, in comparison to $\epsilon_{max}(\mathrm{H} = 1200,\ T = 8) - \epsilon_{max}(\mathrm{H} = 1200,\ T = 1) \approx 0{,}5$. Thus, the difference in distribution of ϵ_{tot} increases exponentially with an increase of H for all T. There is also observed that H and T proportionally enhances their effect in where $\max(\epsilon_{\mathrm{max}}) = \epsilon_{max}(\mathrm{H} = 1200,\ T = 8)$. For an increase of T, the exponential growth of H increases rapidly. The difference $\epsilon_{max}(\mathrm{H} = 1200,\ T = 1) - \epsilon_{max}(\mathrm{H} = 100,\ T = 1) \approx 0{,}045$, in where the difference $\epsilon_{max}(\mathrm{H} = 1200,\ T = 8) - \epsilon_{max}(\mathrm{H} = 100,\ T = 1) \approx 0{,}55$.

Also, it is observed that the maximal total strain of the models ϵ_{max} peaked when the normalized distance Γ/r_t=1.1 for all four thicknesses. This does, however, not necessarily resemble the true peak. There were tested for 22 different Γ. Thus, it can only be concluded that the peak happens somewhere between $\Gamma_{\mathrm{norm}} \in [1,\ 1.25]$. There are reasons to believe that the graph gets smoother with an increase in sampling frequency.

It is important to note that there was a severe difference in the strength of the zone and the strength of the host rock. Using the critical strain concept proposed by (Sakurai, 1981), and comparing the total strains of the charts with Figure 28 created by (Hoek & Diederichs, 2006) over 70% of the models show some extent of squeezing behavior in where the severity increases with both overburden and thickness, and can be further enhanced for certain positions of the zone.

A last note, the symmetry shown in the charts created by the automation resembles a symmetry that leads to thoughts that there has been found something fundamental regarding weakness zones affect the tunnel stability. A hypothesis has been made, that it could be possible to create a dimensionless constant, analogous to the Reynolds number in fluid mechanics, in where the purpose of this number would be to predict when a weakness zone no longer afflicts the stability. If this was to be done, a more thorough study should be initiated, in where the tunnel radius and material parameters also should be included. Also, the lines of the charts are not very smooth, which implies a need for a higher sampling frequency.

8 Conclusion

In the thesis it was successfully developed a method to automize the model creation process of the 2D finite element program RS2. This automation script was used to create the total number of 3696 models to be used in an extensive sensitivity analysis on which weakness zone-defining parameters that affects tunnel stability, and in what extent each parameter contributes to this instability. Four parameters were investigated: (a) zone thickness, (b) overburden, (c) zone angle, (d) distance between zone and tunnel center.

An important finding was that the zone angle Θ did not have any significance on the tunnel stability for any zone thicknesses T, overburdens H, and normalized shortest distances between tunnel center and weakness zones Γ_{norm}. Another important insight was that maximum strain $\epsilon_{max} = \epsilon_{tot}(\Gamma_{norm} = 1.1) \forall \mathrm{T}, \mathrm{H}$. A third curiosity was that the weakness zone was out of the influence zone of the tunnel when $\Gamma_{\mathrm{norm}} >\approx 2$. The exponential increase of ϵ_{tot} with increasing H is severely primed when also T is increased.

The following suggestions are made for further work with this script

- *By varying tunnel radius and material parameters, it can be possible to develop a "Reynolds number" with the purpose of predict the distance when the weakness zone is out of the influence zone of the tunnel. It is believed that this study would be quite extensive, but quite rewarding.*
- *A more complex definition of weakness zones could be implemented, to get a more realistic approach, in where zone with damage zone and fault cores is investigated.*
- *Development of an equation to better quantify the true sensitivity of each parameter. It should be possible to use regression to define an equation that describes the results of the model. With this done it would be possible to define a more mathematical approach regarding a parameter study.*
- *In section 6.2, it was suggested a method to capture the skewness of the deformation distributions of the models created. This would be interesting to investigate, to see if the approach is useful.*

9 References

Aksoy, C. (2008). Review of rock mass rating classification: historical developments, applications, and restrictions. *Journal of mining science*, *44*(1), 51-63.

Amadei, B., & Stephansson, O. (1997). *Rock stress and its measurement*. Springer Science & Business Media.

Bagheripour, M. H., Rahgozar, R., Pashnesaz, H., & Malekinejad, M. (2011). A complement to Hoek-Brown failure criterion for strength prediction in anisotropic rock. *Geomechanics and Engineering*, *3*(1), 61-81.

Balsamo, F., Storti, F., Salvini, F., Silva, A., & Lima, C. (2010). Structural and petrophysical evolution of extensional fault zones in low-porosity, poorly lithified sandstones of the Barreiras Formation, NE Brazil. *Journal of Structural Geology*, *32*(11), 1806-1826.

Barton, N. (1976). The shear strength of rock and rock joints. *International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts*, *13*(9), 255-279. https://doi.org/10.1016/0148-9062(76)90003-6

Barton, N., Lien, R., & Lunde, J. (1974). Engineering classification of rock masses for the design of tunnel support. *Rock mechanics*, *6*(4), 189-236.

Basarir, H. (2008). Analysis of rock–support interaction using numerical and multiple regression modeling. *Canadian Geotechnical Journal*, *45*(1), 1-13. https://doi.org/10.1139/t07-053

Berg, S. S., & Skar, T. (2005). Controls on damage zone asymmetry of a normal fault zone: outcrop analyses of a segment of the Moab fault, SE Utah. *Journal of Structural Geology*, *27*(10), 1803-1822.

Bieniawski, Z. (1973). Engineering classification of jointed rock masses. *Civil Engineering = Siviele Ingenieurswese*, *1973*(12), 335-343.

Bieniawski, Z. T. (1974). Estimating the strength of rock materials. *Journal of the Southern African Institute of Mining and Metallurgy*, *74*(8), 312-320.

Bieniawski, Z. T. (1989). *Engineering rock mass classifications : a complete manual for engineers and geologists in mining, civil, and petroleum engineering*. Wiley.

Blenkinsop, T. G. (2008). Relationships between faults, extension fractures and veins, and stress. *Journal of Structural Geology*, *30*(5), 622-632. https://doi.org/10.1016/j.jsg.2008.01.008

Bobet, A., Fakhimi, A., Johnson, S., Morris, J., Tonon, F., & Yeung, M. R. (2009). Numerical models in discontinuous media: review of advances for rock mechanics applications. *Journal of Geotechnical and Geoenvironmental Engineering*, *135*(11), 1547-1561.

Brace, W., & Martin Iii, R. (1968). A test of the law of effective stress for crystalline rocks of low porosity. *International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts*, *5*(5), 415-426.

Brady, B. H., & Brown, E. T. (1993). *Rock mechanics: for underground mining*. Springer science & business media.

Brady, B. H. G., & Brown, E. T. (2012). *Rock Mechanics: For Underground Mining*. Springer.

Bray, J. (1967). A study of jointed and fractured rock. *Rock mechanics and engineering geology*, *5*(2-3), 117-136.

Brown, E. T., & Hoek, E. (1978). Trends in relationships between measured in-situ stresses and depth. *International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts*, *15*(4), 211-215. https://doi.org/10.1016/0148-9062(78)91227-5

Bruland, A. (2016). Anleggsteknikk GK.

Byerlee, J. (1978). Friction of Rocks. In (pp. 615-626). Birkhäuser Basel. https://doi.org/10.1007/978-3-0348-7182-2_4

[Record #161 is using a reference type undefined in this output style.]

Cai, M. (2008). Influence of stress path on tunnel excavation response–Numerical tool selection and modeling strategy. *Tunnelling and Underground Space Technology*, *23*(6), 618-628.

Cai, M., & Kaiser, P. (2006). Visualization of rock mass classification systems. *Geotechnical & Geological Engineering*, *24*(4), 1089-1102.

Caine, J. S., Evans, J. P., & Forster, C. B. (1996). Fault zone architecture and permeability structure. *Geology*, *24*(11), 1025-1028.

Chapple, W. M. (1987). Strength of rocks. In *Structural Geology and Tectonics* (pp. 746-748). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-31080-0_106

Coulomb, C. A. (1773). Essai sur une application des regles de maximis et minimis a quelques problemes de statique relatifs a l'architecture (essay on maximums and minimums of rules to some static problems relating to architecture).

Crowder, J., & Bawden, W. (2004). Review of post-peak parameters and behaviour of rock masses: current trends and research. *Rocnews, fall*.

Deere, D. U., & Miller, R. P. (1966). *Engineering classification and index properties for intact rock* (Vol. 65-116). University of Illinois.

Donath, F. A. (1972). Effects of cohesion and granularity on deformational behavior of anisotropic rock. *Studies in mineralogy and precambrian geology*, *135*, 95-128.

Drucker, D. C., & Prager, W. (1952). Soil mechanics and plastic analysis or limit design. *Quarterly of applied mathematics*, *10*(2), 157-165.

Edelbro, C. (2003). Rock mass strength: a review. *Teknisk rapport - Luleå tekniska universitet*, 92.

Einstein, H. H. (1996). Tunnelling in difficult ground: Swelling behaviour and identification of swelling rocks. *Rock Mechanics and Rock Engineering*, *29*(3), 113-124. https://doi.org/10.1007/bf01032649

Elliott, G. M. (1982). *An investigation of a yield criterion for porous rock [Doctoral thesis, University of London].* London. https://spiral.imperial.ac.uk/bitstream/10044/1/36259/2/Elliott-GM-1983-PhD-Thesis.pdf

Fasching, F., & Vanek, R. (2011). Engineering geological characterisation of fault rocks and fault zones/Ingenieurgeologische Charakterisierung von Störungsgesteinen und Störungszonen. *Geomechanics and Tunnelling*, *4*(3), 181-194.

Faulkner, D. R., Jackson, C. A. L., Lunn, R. J., Schlische, R. W., Shipton, Z. K., Wibberley, C. A. J., & Withjack, M. O. (2010). A review of recent developments concerning the structure, mechanics and fluid flow properties of fault zones. *Journal of Structural Geology*, *32*(11), 1557-1575. https://doi.org/10.1016/j.jsg.2010.06.009

Gercek, H. (2007). Poisson's ratio values for rocks. *International Journal of Rock Mechanics and Mining Sciences*, *44*(1), 1-13. https://doi.org/10.1016/j.ijrmms.2006.04.011

Goel, R. K., & Singh, B. (2011). *Engineering Rock Mass Classification*. https://doi.org/10.1016/C2010-0-64994-7

Goodman, R. E. (1989). *Introduction to rock mechanics* (Vol. 2). Wiley

Griffith, A. A. (1921). The phenomena of rupture and flow in solids. *Philosophical transactions of the royal society of london. Series A - mathematical, physical and engineering sciences*, *221*(582-593), 163-198.

Grimstad, E., Kankes, K., Bhasin, R., Magnussen, A. W., & Kaynia, A. (2002). Rock mass quality Q used in designing reinforced ribs of sprayed concrete and energy absorption. *Report, Norwegian Geotechnical Institute*, *19*.

Haimson, B., & Cornet, F. (2003). ISRM suggested methods for rock stress estimation—part 3: hydraulic fracturing (HF) and/or hydraulic testing of pre-existing fractures (HTPF). *International Journal of Rock Mechanics and Mining Sciences*, *40*(7-8), 1011-1020.

Hast, N. (1958). *The measurement of rock pressure in mines*. Generalstabens Litografiska Anstalts Förlag.

Hatheway, A. W., & Kiersch, G. A. (1986). Engineering properties of rocks. In R. S. Carmichael (Ed.), *Handbook of physical properties pf rocks.* (Vol. 2, pp. 289-331). CRC press.

Herget, G. (1988). *Stresses in rock*. Balkema Rotterdam.

Hoek, E. (1983). Strength of jointed rock masses. *Geotechnique*, *33*(3), 187-223.

Hoek, E. (1999). *Support for very weak rock associated with faults and shear zones*. Routledge.

Hoek, E. (2001). Big tunnels in bad rock. *Journal of Geotechnical and Geoenvironmental Engineering*, *127*(9), 726-740.

Hoek, E., & Brown, E. T. (1980a). Empirical strength criterion for rock masses. *Journal of the geotechnical engineering division*, *106*(9), 1013-1035.

Hoek, E., & Brown, E. T. (1980b). *Underground excavations in rock*. CRC Press.

Hoek, E., & Brown, E. T. (1997). Practical estimates of rock mass strength. *International Journal of Rock Mechanics and Mining Sciences*, *34*(8), 1165-1186. https://doi.org/10.1016/s1365-1609(97)80069-x

Hoek, E., & Brown, E. T. (2019). The Hoek–Brown failure criterion and GSI – 2018 edition. *Journal of Rock Mechanics and Geotechnical Engineering*, *11*(3), 445-463. https://doi.org/10.1016/j.jrmge.2018.08.001

Hoek, E., Carranza-Torres, C., & Corkum, B. (2002). Hoek-Brown failure criterion-2002 edition. *Proceedings of NARMS-Tac*, *1*(1), 267-273.

Hoek, E., & Diederichs, M. S. (2006). Empirical estimation of rock mass modulus. *International Journal of Rock Mechanics and Mining Sciences*, *43*(2), 203-215.

Hoek, E., Kaiser, P. K., & Bawden, W. F. (1995). *Support of underground excavations in hard rock* (1 ed.). CRC Press.

Hoek, E., & Marinos, P. (2000). Predicting tunnel squeezing problems in weak heterogeneous rock masses. Retrieved 14.11.2021, from https://www.rocscience.com/assets/resources/learning/hoek/Predicting-Tunnel-Squeezing-Problems-in-Weak-Heterogeneous-Rock-Masses-2000.pdf

Hudson, J. A., & Harrison, J. P. (1997). *Engineering Rock Mechanics: An Introduction to the Principles* (1st ed. ed.). Amsterdam: Elsevier Science & Technology.

Huston, R. L., & Josephs, H. (2009). *Practical stress analysis in engineering design* (3rd ed. ed.). CRC Press.

Høien, A. H. (2018). *Applicability of reinforced ribs of sprayed concrete in sections of poor quality and swelling rock mass [Doctoral thesis, Norwegian University of Science and Tecnology].* Trondheim. https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2584189

Høien, A. H., & Nilsen, B. (2019). Analysis of the stabilising effect of ribs of reinforced sprayed concrete (RRS) in the Løren road tunnel. *Bulletin of Engineering Geology and the Environment*, *78*(3), 1777-1793. https://doi.org/10.1007/s10064-018-1238-1

Høien, A. H., Nilsen, B., & Olsson, R. (2019). Main aspects of deformation and rock support in Norwegian road tunnels. *Tunnelling and Underground Space Technology*, *86*, 262-278. https://doi.org/10.1016/j.tust.2019.01.026

Høien, A. H., Nilsen, B., Vistnes, G., & Olsson, R. (2020). Experimental triaxial testing of swelling gouge materials. *Bulletin of Engineering Geology and the Environment*, *79*(1), 355-370. https://doi.org/10.1007/s10064-019-01547-6

Ismael, M., Chang, L., & Konietzky, H. (2017). Behaviour of anisotropic rocks. Retrieved 05.01.2022, from https://tu-freiberg.de/sites/default/files/media/professur-felsmechanik-32204/E-book/24_behaviour_of_anisotropic_rocks_0.pdf

ISRM. (1978). International society for rock mechanics commission on standardization of laboratory and field tests: Suggested methods for the quantitative description for discontinuities in rock masses. *International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts*, *15*(6), 319-368. https://doi.org/10.1016/0148-9062(78)91472-9

ISRM. (2014). *The ISRM suggested methods for rock characterization, testing and monitoring: 2007-2014*. Springer.

Jaeger, J. C., Cook, N. G., & Zimmerman, R. (2009). *Fundamentals of rock mechanics*. John Wiley & Sons.

Jing, L., & Hudson, J. (2002). Numerical methods in rock mechanics. *International Journal of Rock Mechanics and Mining Sciences*, *39*(4), 409-427.

Kalender, A., Sonmez, H., Medley, E., Tunusluoglu, C., & Kasapoglu, K. (2014). An approach to predicting the overall strengths of unwelded bimrocks and bimsoils. *Engineering geology*, *183*, 65-79.

Kirsch, C. (1898). Die theorie der elastizitat und die bedurfnisse der festigkeitslehre. *Zeitschrift des Vereines Deutscher Ingenieure*, *42*, 797-807.

Krauland, N., Söder, P., & Agmalm, G. (1989). Determination of rock mass strength by rock mass classification—Some experiences and questions from Boliden mines. *International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts*, *26*(1), 115-123. https://doi.org/10.1016/0148-9062(89)90531-7

Kulhawy, F. H. (1975). Stress deformation properties of rock and rock discontinuities. *Engineering Geology*, *9*(4), 327-350. https://doi.org/10.1016/0013-7952(75)90014-9

Kaasbøll Andresen, E. (2021). *A literature review on important factors considering a numerical sensitivity study on weakness zones* (Specialization project, Issue.

Lamuta, C. (2019). Elastic constants determination of anisotropic materials by depth-sensing indentation. *SN Applied Sciences*, *1*(10), 1263. https://doi.org/10.1007/s42452-019-1301-y

Li, C. C. (2018). *TGB4210 Rock Mechanics Compendium*.

Luenberger, D. G., & Ye, Y. (1984). *Linear and nonlinear programming* (Vol. 2). Springer.

Mao, D., & Nilsen, B. (2013). Numerical analysis of effects of weakness zones on tunnel stability 2D versus 3D. Advances in underground space development. *Research publishing service*, 388.

Marinos, V., & Carter, T. G. (2018). Maintaining geological reality in application of GSI for design of engineering structures in rock. *Engineering Geology*, *239*, 282-297.

Matsukura, Y., Hashizume, K., & Oguchi, C. (2002). Effect of microstructure and weathering on the strength anisotropy of porous rhyolite. *Engineering Geology*, *63*(1-2), 39-47.

McClay, K. R. (1987). *The mapping of geological structures*. John Wiley & Sons.

McCutchen, W. (1982). Some elements of a theory for in-situ stress. International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts,

McGrath, A. G., & Davison, I. (1995). Damage zone geometry around fault tips. *Journal of Structural Geology*, *17*(7), 1011-1024.

METU. (1989a). *Investigation on the determination of rock mechanics and design parameters for coal and coal measure rocks at Asma Mine. Report prepared for TTK, Department of Mining Engineering, Ankara.*

METU. (1989b). *Investigation on the determination of rock mechanics and design parameters for coal and coal measure rocks at Gelik Mine. Report prepared for TTK, Department of Mining Engineering, Ankara.*

METU. (1989c). *Investigation on the determination of rock mechanics and design parameters for coal and coal measure rocks at Kandilli Mine. Report prepared for TTK, Department of Mining Engineering, Ankara.*

Mitchell, T. M., & Faulkner, D. R. (2009). The nature and origin of off-fault damage surrounding strike-slip fault zones with a wide range of displacements: A field study from the Atacama fault system, northern Chile. *Journal of Structural Geology*, *31*(8), 802-816. https://doi.org/10.1016/j.jsg.2009.05.002

Mogi, K. (1967). Effect of the intermediate principal stress on rock failure. *Journal of Geophysical Research*, *72*(20), 5117-5131.

Mohr, O. (1882). Über die Darstellung des Spannungszustandes und des Deformationszustandes eines Körperelementes und über die Anwendung derselben in der Festigkeitslehre. *Der Civilingenieur*, *28*(2), 113-156.

Myrvang, A. M. (2001). *Bergmekanikk*. Institutt for geologi og bergteknikk, NTNU.

NGI. (2015). *Bruk av Q-systemet*. Norges Geotekniske Institutt. https://www.ngi.no/Tjenester/Fagekspertise/Ingenioergeologi-og-bergteknikk/Q-systemet

Nilsen, B. (2016). *Ingeniørgeologi Berg Grunnkurskompendium* (3 ed.). Akademika.

Onah, H. (2012). Different element methods in engineering practice. *Nigerian Journal of Technology*, *31*(3), 288-292.

Palmstrom, A. (1996). RMi-a rock mass characterization system for rock engineering purposes. *Journal of Rock Mechanics and Tunnelling Technology*, *1*, 69-108.

Palmstrom, A., & Broch, E. (2006). Use and misuse of rock mass classification systems with particular reference to the Q-system. *Tunnelling and underground space technology*, *21*(6), 575-593.

Palmström, A., & Stille, H. (2010). *Rock Engineering*. London: Institution of Civil Engineers.

Perras, M. A., & Diederichs, M. S. (2014). A review of the tensile strength of rock: concepts and testing. *Geotechnical and geological engineering*, *32*(2), 525-546.

Riedmüller, G., Brosch, F. J., Klima, K., & Medley, E. W. (2001). Engineering geological characterization of brittle faults and classification of fault rocks. *Felsbau*, *19*(4), 13-19.

RocScience. (1989-2021). *Stress Analysis Verification Manual*. Retrieved 02.04.2022 from https://www.rocscience.com/help/rs2/verification-theory

RocScience. (2021). *Preliminaries on Constitutive Models*. Retrieved 10.12.2021 from https://static.rocscience.cloud/assets/verification-and-theory/RS2/1-Preliminaries-on-Constitutive-Models.pdf

Roylance, D. (2001). Transformation of Stresses and Strains. Retrieved 15.10.2021, from https://web.mit.edu/course/3/3.11/www/modules/trans.pdf

Sakurai, S. (1981). Direct strain evaluation technique in construction of underground opening. The 22nd US Symposium on Rock Mechanics (USRMS),

Sakurai, S. (1984). Displacement measurements associated with the design of underground openings. Field measurements in geomechanics. International symposium,

Saltelli, A. (2002). Sensitivity analysis for importance assessment. *Risk analysis*, *22*(3), 579-590.

Saroglou, H., & Tsiambaos, G. (2008). A modified Hoek–Brown failure criterion for anisotropic intact rock. *International Journal of Rock Mechanics and Mining Sciences*, *45*(2), 223-234.

Savage, H., & Brodsky, E. (2010). Collateral damage: capturing slip delocalization in fracture profiles. *Journal of Geophysical Research*.

Scholz, C. H. (1987). Wear and gouge formation in brittle faulting. *Geology*, *15*(6), 493-495.

Sheorey, P. R. (1994). A theory for In Situ stresses in isotropic and transverseley isotropic rock. *International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts*, *31*(1), 23-34. https://doi.org/10.1016/0148-9062(94)92312-4

SINTEF. (2016). *Rock Mechanic Properties of Rocks Tested in SINTEF Rock Mechanics Laboratory* Infrastructure, S.B.a. (Ed.).

SVV. (2022). N500 Vegtunneler. https://viewers.vegnorm.vegvesen.no/product/859938/nb

Sweigart, A. (2014). *PyAutoGUI*. In (Version 0.9.53) https://github.com/asweigart/pyautogui/commits/master/docs/index.rst

Sæter, H. S. B. (2005). Kompendium: Innføringskurs i strukturgeologi.

Saabye Ottosen, N., & Petersson, H. (1992). *Introduction to the finite element method*. Pearson Prentice Hall.

Terzaghi, K. (1923). Die Berechnung der Durchlassigkeitsziffer des Tones aus Dem Verlauf der Hidrodynamichen Spannungersheinungen *Akademie der Wissenschaften in Wien*, *132*, 13.

Terzaghi, K. (1946). Introduction to tunnel geology. *Rock tunnelling with steel supports*, 17-99.

Terzaghi, K., & Richart Jr, F. (1952). Stresses in rock about cavities. *Geotechnique*, *3*(2), 57-90.

Trinh, Q. N., Myrvang, A., & Sand, N. S. (2010). Rock Excavation And Support For a Crusher Hall At Rana Gruber, Norway. 44th U.S. Rock Mechanics Symposium and 5th U.S.-Canada Rock Mechanics Symposium,

Ullemeyer, K., Siegesmund, S., Rasolofosaon, P. N., & Behrmann, J. H. (2006). Experimental and texture-derived P-wave anisotropy of principal rocks from the transalp traverse: an aid for the interpretation of seismic field data. *Tectonophysics*, *414*(1-4), 97-116.

Vermeer, P. A., & De Borst, R. (1984). Non-associated plasticity for soils, concrete and rock. *HERON*, *29*(3), 1-64.

Vincent, C. P. (2017). *Program Arcade Games - with Python And Pygame* http://programarcadegames.com/

Voight, W. (1928). Lehrbuch der kristallphysik. *Teubner, Leipzig*.

Vutukuri, V. S., Lama, R. D., & Saluja, S. S. (1974). *Handbook on mechanical properties of rocks* (Vol. 1). Trans Tech Publications.

Wahlstrom, E. E. (1973). *Tunneling in rock* (Vol. 3). Elsevier.

Walton, G., Labrie, D., & Alejano, L. R. (2019). On the Residual Strength of Rocks and Rockmasses. *Rock Mechanics and Rock Engineering*, *52*(11), 4821-4833. https://doi.org/10.1007/s00603-019-01879-5

Wawersik, W., Carlson, L., Holcomb, D., & Williams, R. (1997). New method for true-triaxial rock testing. *International Journal of Rock Mechanics and Mining Sciences*, *34*(3-4), 1-14.

Wawersik, W. R. (1968). *Detailed analysis of rock failure in laboratory compression tests*. University of Minnesota.

Wei, Z. (1988). *A fundamental study of the deformability of rock masses [Doctoral Thesis, University of London].* London.

Wilson, J., Chester, J., & Chester, F. (2003). Microfracture analysis of fault growth and wear processes, Punchbowl Fault, San Andreas system, California. *Journal of Structural Geology*, *25*(11), 1855-1873.

Wong, L. N. Y., Maruvanchery, V., & Liu, G. (2016). Water effects on rock strength and stiffness degradation. *Acta Geotechnica*, *11*(4), 713-737.

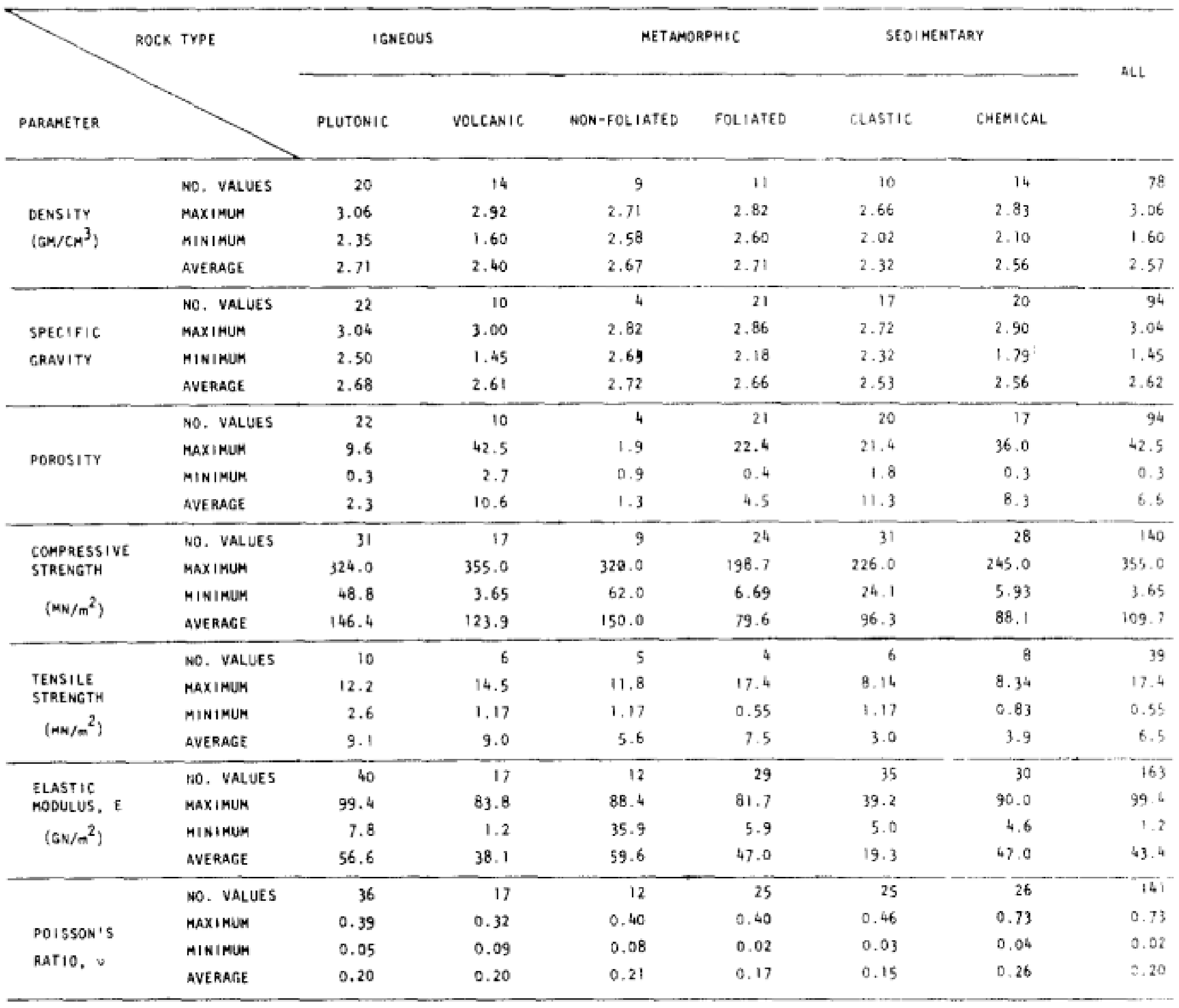
Woodcock, N., & Mort, K. (2008). Classification of fault breccias and related fault rocks. *Geological Magazine*, *145*(3), 435-440.

Zuo, J.-p., Li, H.-t., Xie, H.-p., Ju, Y., & Peng, S.-p. (2008). A nonlinear strength criterion for rock-like materials based on fracture mechanics. *International Journal of Rock Mechanics and Mining Sciences*, *45*(4), 594-599.

Zuo, J., Liu, H., & Li, H. (2015). A theoretical derivation of the Hoek–Brown failure criterion for rock materials. *Journal of Rock Mechanics and Geotechnical Engineering*, *7*(4), 361-366.

10 Appendix

A1: **Summary of uniaxial stress-strain parameters.**



PARAMETER \ ROCK TYPE		IGNEOUS		METAMORPHIC		SEDIMENTARY		ALL
		PLUTONIC	VOLCANIC	NON-FOLIATED	FOLIATED	CLASTIC	CHEMICAL	
DENSITY (GM/CM^3)	NO. VALUES	20	14	9	11	10	14	78
	MAXIMUM	3.06	2.92	2.71	2.82	2.66	2.83	3.06
	MINIMUM	2.35	1.60	2.58	2.60	2.02	2.10	1.60
	AVERAGE	2.71	2.40	2.67	2.71	2.32	2.56	2.57
SPECIFIC GRAVITY	NO. VALUES	22	10	4	21	17	20	94
	MAXIMUM	3.04	3.00	2.82	2.86	2.72	2.90	3.04
	MINIMUM	2.50	1.45	2.69	2.18	2.32	1.79	1.45
	AVERAGE	2.68	2.61	2.72	2.66	2.53	2.56	2.62
POROSITY	NO. VALUES	22	10	4	21	20	17	94
	MAXIMUM	9.6	42.5	1.9	22.4	21.4	36.0	42.5
	MINIMUM	0.3	2.7	0.9	0.4	1.8	0.3	0.3
	AVERAGE	2.3	10.6	1.3	4.5	11.3	8.3	6.6
COMPRESSIVE STRENGTH (MN/m^2)	NO. VALUES	31	17	9	24	31	28	140
	MAXIMUM	324.0	355.0	320.0	198.7	226.0	245.0	355.0
	MINIMUM	48.8	3.65	62.0	6.69	24.1	5.93	3.65
	AVERAGE	146.4	123.9	150.0	79.6	96.3	88.1	109.7
TENSILE STRENGTH (MN/m^2)	NO. VALUES	10	6	5	4	6	8	39
	MAXIMUM	12.2	14.5	11.8	17.4	8.14	8.34	17.4
	MINIMUM	2.6	1.17	1.17	0.55	1.17	0.83	0.55
	AVERAGE	9.1	9.0	5.6	7.5	3.0	3.9	6.5
ELASTIC MODULUS, E (GN/m^2)	NO. VALUES	40	17	12	29	35	30	163
	MAXIMUM	99.4	83.8	88.4	81.7	39.2	90.0	99.4
	MINIMUM	7.8	1.2	35.9	5.9	5.0	4.6	1.2
	AVERAGE	56.6	38.1	59.6	47.0	19.3	47.0	43.4
POISSON'S RATIO, ν	NO. VALUES	36	17	12	25	25	26	141
	MAXIMUM	0.39	0.32	0.40	0.40	0.46	0.73	0.73
	MINIMUM	0.05	0.09	0.08	0.02	0.03	0.04	0.02
	AVERAGE	0.20	0.20	0.21	0.17	0.15	0.26	0.20

```
"""
Import of modules created for this project is the ones starting with from.
Important note, these modules is again using modules created by others in conjunction with own modules etc.
So, thanks to all that contributes with useful tools on an open-source basis!!!! Without the python community
this project would not have been possible.
"""
import time

import pandas as pd

from Automatisering_RS2 import module_calculate as mcal, module_execute_model_alteration as mema, mouse_tracker
as mt
from Automatisering_RS2 import module_create_mesh as mcm
from Automatisering_RS2 import module_execute_data_processing as medp
from Automatisering_RS2 import module_main_functions as mmf
from Automatisering_RS2 import module_plan_experiment as mpe
from Automatisering_RS2 import module_store_data as msd

"""
This is the main file of the automation script made automize the modelling process of the 2-dimensional FEM programme
RS2 by RocScience Inc.

It was created as an integral part of the master thesis: "" written by Eirik Kaasbøll Andresen the spring 2022. This
Thesis is available at NTNU Open. Rweading of the thesis will give the context of this script and may give the reader
new ideas thqat can maker use of this script and lead to further developments. You can get the script from my github
account.

The main file works as a hub and is putting togehter the contribution of several scripts developed in this project. The
scripts is categorized and is developed to face certain needs.

If a certain script or module is not used for a paricular reason, it is commented out.
The script is designed with a memory storage solution. If one part of the script is finnished, all information is needed
stop the script, for then to comment out certain blocks and keep on from the point the script was stopped.

However, this script is not optimized for being userfriendly. So, feel free to contact me if there are any questions.

BR
   Eirik Kaasbøll Andresen - the developer of this script.


PS
   All communinication through monitor and some variables are written in Norwegian, so have fun with google translate
   and enjoy some words and sentences of this beautful language!
"""

"""
This is the controll panel stearing which operations to be done when running the script. If all is False, this
script does nothing. If no changes is made to input parameters defining the models there is no need to go further.

#1 is True if the project is moved over to another screen. If there are difference from the new screen compared to old
this will mean that the mouse coordinates must be updated.

#2 is True if thwe mouse coordinates must be redefined

#3 is True if a new project is to be made. This will set up the folder strucure needed

#(4 to 10) with name bool_shall_execute_... is used to turn on and of different operations of the script in where all
```

```
seqence through all models of the experiment and does different tasks.

#4 True turns on the model alteration process
#5 True True turns on the create mesh process
#6 True turns on the calculation process
#8 True turns on the data storing process
#10 True turns on the first data processing process, in where the second data processing is done using Excel,
see thesis.

#7 If true, the function calculate pauses after the creation of the log file analysis, so that the user can
   check the quality.

#9 is False if data storing is done a second time or more. This is due to how rs2 interpret works. The second time it is
run, it opens directly wit total deformation as parameter. However, the first parameter stored must be sigma 1.

NB!!! If there are done any changes to the template files on which all the models is based, all the parameters
between #4 - 10 must  set to True. This is all from changing the parameters defining the calculation
(iteration, tolerance), mesh, material parameterers and more.

Another important note. If the script must be stopped after for instance mesh creation this is fine. Just turn the
processes already executed to False and the scripts run from where you stopped.

Furthermore, let say there was something going wrong with the mouse operations of the data storage after completing
overburden 100, 200, and 500, where the bug happened when overburden 800. It is not necessary to do the entire
data storage process again. Using files_to_skip enables the user to controll which clusters of files to be operated
by refering to overburdens, which is the coarsest sorting defined in this thesis.
An example; files_to_skip = [0,1,2] skips the overburdens 100, 200, 300 of the list
overburdens = [100, 200, 300, 500, 800, 1200]. overburdens is allocated in line 220.

However, files_to_skip cannot be used on the function calculate from module_calculate. This is not as big issue since
calculate for the most case uses rs2's own batch functionality. Just set #4,5 and 6 to false. Open rs2 calculate and
choose the files that needs to be calculated. And then run the rest when the calculations are finnished. It is
adviced to check the file containing log-files exceeding the given tolerance, to see if the calculations are reliable by
setting #7 to true.

For yes, the calculate operation ends with analysing all the log-files created by rs2 compute after each completed
calculation. If there is found tolerances greater than a certain error set by the user, it stores the path of these
log-files, and the user can check if the differences are problematic.
"""
# 1
bool_know_size_monitor = False
# 2
bool_shall_reedit_mouse_coordinates = False
# 3
bool_create_new_project = False

# 4 to 10 Settings for the operations
bool_shall_execute_model_alteration = False

bool_shall_execute_create_mesh = False

bool_shall_execute_calculate = False
bool_stop_to_check_logs = False

bool_shall_execute_data_store = False
bool_is_first_time_execute_data_store = False
```

```
bool_shall_execute_data_processing = False

# 11 skip overburdens
files_to_skip = []

"""NB!!!!!!!!!!!
It two functions in the modules created that must be updated if there are changes in
the setup of the sensitivity experiment. The functions are:

    which_overburden and get_ob_index in module_plan_experiment.py, in where changes in overburden values
    must be implemented.

"""

"""
It will be calculated computation time of each operation seperatly, due to the script is forced to take break
two times.

The first break is when the quality of the models is to be checked. There are some few bugs in the material allocation
which must be attoned for. It is only necessary to check the models of overburden 100 and 200. The rest is based
on the model of overburden 200 in where the only difference is the overburden. The differnece betweeen overburden 100
and 200 is the outerboundary. Thus, the two models experience differneces in their bugs. It is not knwon why the bugs
exist, however it does not affect many models.

The second break is under calculations. The reason for this is that there appeared some complications with rs2 compute.
It must be implemented the path of where rs2 compute finds its data. It is not done now due not having time, but it is
not a complicated thing to implement.

Also, due to symmetry reasons it was only necessary to caclulate for angles between 45 to 90 degrees. So, the problems
faced with the material allocations is more severe if the angles are not between this range. If the script is to be
used for more complex purposes, this problem must be resolved. It is not known why this occur at this stage.
"""

"""
This aks the user if the programme should be run. If not true, the script is exited.
"""

command = mmf.procede_script()

if command == 'j':
    """
    This starts the timer used to calculate the running time of the entire script.
    This will include the time used to check models and the delay if the user do not detect early enough that the
    calculation is over.
    """
    time_start = time.time()

    """
    Settings for displaying panda dataframes is set. This helped the workflow quit a bit.
    """
    pd.set_option('display.max_rows', None)
    pd.set_option('display.max_columns', None)
    pd.set_option('display.width', None)
    # pd.set_option('display.max_colwidth', -1)

    """
    Do you want to know the size of the monitor, set bool_know_size_monitor = True in controll panel line 86.
```

```
This must be known to check if the mouse coordinates must be redefined if a ne sreen is to be used.
"""
mmf.get_screen_width(bool_know_size_monitor)

"""
liste_stier_PycharmProjects_automatisering contains the paths of the csv containing all paths used for a range
of purposes of this script, and a path to
"""
"""
main_stringobjects consist  all paths necessary to be defined for the script to run given by a list
of paths with metadata taken from liste_stier_PycharmProjects_automatisering
"""
main_stringobjects = pd.read_csv(r'C:\temp\thesis\eksperimenter\mektighet_1'
                                 r'\base_modeler\Pycharm_automatisering'
                                 r'\liste_stier_PycharmProjects_automatisering.txt',
                                 sep=';')
"""
mt.mouse_tracker updates mouse operations if neccessary, if that the case,
bool_shall_reedit_mouse_coordinates = True
"""
mt.mouse_tracker(main_stringobjects['object'][7], bool_shall_reedit_mouse_coordinates)

"""
This part defines the input parameters used in the numerical models.

the plan_experiment module (shortened: pe) detects which defined parameters that is to be static and which that is
to be dynamic.

Then it creates the ranges of each parameter. These parameters is later used to define the material behaviour and
geometry of the models.
"""

# defines some general aspects regarding calculation behaviour used by rs2 compute,
tolerance = 0.001
maxiter = 10000

# defines parameters regarding the output values to be stored in store_data function
ant_parametere_interpret = 2
parameter_navn_interpret = ['sigma 1:', 'total deformasjon:', 'end']  # also used in execute_data_processing

# defines two parameters regarding the dynamic parameters of this experiment
parameters_varied = ['od', 'v', 'x']  # used in execute_data_processing
list_change_fieldstress = [300, 500, 800, 1200]  # used in execute_model_alteration

"""
These parameters defines the most central parameters describing geometry, stress situation and material parameters
In this script, the shape of tunnel and material parameters is equal for all models, so these parameters is only
set in the template files in which the models of the experiment is based. Thus, it is only stress, thickness of
zone, angle of zone, and translation of zone that are implemented here. It would be possible to include material
changes in the script at a later stage.

Also, in the thesis, the thickness is only changed when a new experiment is done, just to reduce the size of each
dataset.

The filenames of the fea files is defined by the parameters below.
The filenames are central in the implementation of the script for two reasons:
1: To make it easy to differ the files from each other
```

```
2: To let the script know which values needed to construct the varied geometry and stress

The structure of the filenames is as follows (NB! Norwegian acronyms):
S_bm80_ss1_k1_od500_m4_v22.5_x0_y0. In where,

S (sirkulær): circular tunnel shape,
bm (bergmassekvaliteten): rock mass quality given by GSI
ss (svakhetssone): material quality of zone given by GSI (in the thesis it is given by mohr-columb parameters
                                          , so the name is somewhat misleading)
k: constant for the isotropic ratio of vertical to horizontal stress,
od (overdekningen): the overburden,
m (mektigheten): thickness of zone,
v (vinkel): angle of zone,
x: horizontal translation of zone,
y: the vertical translation of zone.



NB!!!!!!
The y_attributes is not used. It was, during the method development of the thesis, realised that the
radial distance was the most practical to use, not x and y.
Thus, y is 0 because it is not used, and x_attributes will be normalised later in script. For instance,
x = [0, 0, 0.25, 0.5, 0.75, 1, 2, 3, 4, 4.5, 5, 5.5, 5.75, 6, 7, 8, 9, 10, 11, 13, 15, 20], is the magnitude
of the radial lengths, to be normalized later.
Normalized:
It is the shortest distance from tunnel centre to zone we want to define, it is a radial vector which has the same
magnitude for all angles of the zone, and is same for all thicknesses too since it is the lineament closest to the
zone we are measuring from. Pythagoras is used to express this radial distance with its x-value,
which can be done since the shortest distance is the normal normal to the lineamnet of the zone intersecting the
tunnel centre.

To conclude, the x-values is later normalised using pythagoras to transform the radial distance.
And y is kept to zero.
This is done in set_model_csv_attributes_batch from module_plan_experiment, where all the model names of the
experiment is defined. Thus the x-value will be dependent on the zone angle.
"""

rock_mass_material, weakness_zone_material, stress_ratio, overburdens, thickness_attributes, angle_attributes, \
    y_attributes, x_attributes = 80, 1, 1, [100, 200, 300, 500, 800, 1200], \
    4, [45, 52.5, 60, 67.5, 75, 82.5, 90], 0, \
    [0, 0, 0.25, 0.5, 0.75, 1, 2, 3, 4, 4.5, 5, 5.5, 5.75, 6, 7, 8, 9, 10, 11, 13, 15, 20]

# True lengths is the magnitude of distance between tunnel centre and zone centre
true_lengths = [x + thickness_attributes / 2 if i > 0 else x for i, x in enumerate(x_attributes)]

# list contains the length of the sides of the quadratic outer boundary
ytre_grenser_utstrekning = [100, 150, 150, 150, 150, 150]

# the number of points defining the tunnel boundary of the model. For the script to function properly,
# it must be possible to divide it by 4. The high number is due to it increases the accuracy around tunnel
# periphery.
n_points_tunnel_boundary = 360

# this list defines the values for material allocation. 15 is weakness zone material and 16 is the rock mass
# the smallest lsit element refers to one geometrical element of a model, and the two numbers refers to before exc
# and after excavation respectively.
# there are four scenarios:
```

```python
# 1:The zone do not cross tunnel at all.   -> 4 geometrical elements, excavation throug rock mass
# 2:The zone is thicker than tunnel diameter and completely submerges the tunnel  -> 4 geometrical elements,
#                                                          excav through weaknes zone
# 3:Part of the zone cross tunnel -> 5 elements
# 4:whole zone cross tunnel but is thinner than tunnel diameter -> 7 elements
list_which_material = [[[[15, 15], [15, 15], [16, 16], [16, 0]], [[15, 15], [15, 15], [16, 16], [15, 0]]],
                       [[15, 15], [15, 15], [15, 0], [16, 0], [16, 16]],
                       [[15, 15], [15, 15], [15, 0], [15, 0], [16, 16], [16, 16], [16, 0]]]

# the names of the columns of the parameters written to the csv containing the first round processed data, see
# discussion in thesis for what this means
valnavn = ['file_name', 'true_lengths', 'od', 'v', 'x', 'sigma 1, max', 'totaldeformasjon, max',
           'quad_high - sigma 1, inbetween', 'quad_high - totaldeformasjon, inbetween',
           'quad_low - sigma 1, inbetween', 'quad_low - totaldeformasjon, inbetween']
list_valnavn = []
list_valnavn += 7 * [valnavn]

# this parameter contains the path to csv containing the descripton of variables of each model, as discussed in
# line 217.
path_csv_parameter_verdier_fil = main_stringobjects['object'][0]

# set_model_csv_attributes_batch interprets the values given in line 267 and creates a csv file describing
# the definition of each model of the experiment.
number_of_files = mpe.set_model_csv_attributes_batch(path_csv_parameter_verdier_fil, rock_mass_material,
                                                     weakness_zone_material, stress_ratio, overburdens,
                                                     thickness_attributes, angle_attributes, y_attributes,
                                                     x_attributes)

"""
In this section rest of paths given in main_stringobjects is allocated. They are explained as they come up.
"""

# this path points to the csv containing the parameters describing the foldernames in where the results are stored,
# categorized regarding overburden. The experiments are categorized regarding thickness of zone
path_csv_parameter_verdier_mappe = main_stringobjects['object'][1]

# shell for the fea models and the csv containg the results repectively.
paths_shell_rs2 = main_stringobjects['object'][2]

# paths of the programmes to be executed in the script:
path_rs2 = main_stringobjects['object'][3]
path_rs2_compute = main_stringobjects['object'][4]
path_rs2_interpret = main_stringobjects['object'][5]

# path to csv containing mouse coordinates
sti_koordinater_mus = main_stringobjects['object'][6]

# path for folder to conmtain the fea files
sti_til_mappe_for_arbeidsfiler = main_stringobjects['object'][7]
# path for folder to contain the folders that containes the categorized result files
sti_til_mapper_endelige_filer = main_stringobjects['object'][8]

# sti_csv_gamle_rs2stier og sti_csv_gamle_csvStier:
# is the paths for the csv's containing the paths of the fea files and result csv files from the last run.
sti_csv_gamle_rs2stier = main_stringobjects['object'][9]
sti_csv_gamle_csvstier = main_stringobjects['object'][10]
```

```python
# sti_list_variables_2lines_calculations: a list containing the paths of the geometrical data needed for the
# script to work without defining the geometri of all files each time.
sti_list_variables_2lines_calculations = [main_stringobjects['object'][11], main_stringobjects['object'][12],
                                          main_stringobjects['object'][13], main_stringobjects['object'][14],
                                          main_stringobjects['object'][15], main_stringobjects['object'][16],
                                          main_stringobjects['object'][17], main_stringobjects['object'][18],
                                          main_stringobjects['object'][19]]

# sti_tolerance_too_high: path to csv containing the list of log-files with tolerance exceeded
sti_tolerance_too_high = main_stringobjects['object'][21]

# sti_values_toplot: paths containing the first round of processed data as described in the thesis.
sti_values_toplot = main_stringobjects['object'][22]

# storage_calculation_times contains the path in where the caclulation times of the entire script and each
# operation is stored. It is emptied for each run, so, if the calculation time of each run is to be stored, do
# that in another file
storage_calculation_times = main_stringobjects['object'][23]

"""
the coordinates of the mouse operations is fetched here. The colnames is also defined.
"""
df_koordinater_mus = pd.read_csv(sti_koordinater_mus, sep=';')
navn_kol_df_koord_mus = ['Handling', 'x', 'y']

"""
paths to the template files is defined
"""
sti_kildefil_rs2, sti_kildefil_csv = mmf.get_file_paths_batch(paths_shell_rs2, path_csv_parameter_verdier_fil)

"""
create_work_and_storage_folders
"""
mmf.create_work_and_storage_folders(sti_til_mappe_for_arbeidsfiler, sti_til_mapper_endelige_filer)

"""
mappenavn_til_rs2/csv: containes the column names of df_stier_rs2/csvfiler
"""
mappenavn_til_rs2, mappenavn_til_csv = mmf.get_name_folders(sti_til_mapper_endelige_filer)
df_filnavn_rs2, df_filnavn_csv = mmf.make_file_name(path_csv_parameter_verdier_fil)

"""
get file paths of fea-files and csv to store ecxcavation query data. If bool_create_new_project True, old content is
deleted and new files and folder system is created, if False the olde ystem is kept and the file paths is fetched.
The fea files, if bool_create_new_project True, is created by copying the template files.
"""

df_stier_rs2filer, df_stier_csvfiler = \
    mmf.get_paths_df(sti_til_mappe_for_arbeidsfiler, sti_til_mapper_endelige_filer, sti_kildefil_rs2,
                     sti_kildefil_csv, sti_csv_gamle_rs2stier, sti_csv_gamle_csvstier,
                     path_csv_parameter_verdier_mappe, ytre_grenser_utstrekning,
                     bool_create_new_project)

"""
df_endrede_attributter_rs2filer is dataframe containing the values defining all the models used in the geometry
construction. The columns are given by the overburdens.
"""
```

```
df_endrede_attributter_rs2filer = mmf.get_changing_attributes(df_stier_rs2filer, mappenavn_til_rs2)

"""
used by autogui processes to define how long the script shall wait before next operation is initiated. Given in
seconds. Important to ensure that one command is completed before the next arrives. See, thesis for a more
thorough description.
"""
time0 = [0, 0.7, 1, 2, 5]

"""
the geometries of fea models is created, or if bool_shall_execute_model_alteration is False, necessary
geometry data is fetched from pickle-files
"""
#
list_of_df_2lines_info, colnames_of_dfs_2lines_info = \
    mema.execute_model_alteration(ytre_grenser_utstrekning, n_points_tunnel_boundary, overburdens,
                                  list_change_fieldstress,
                                  mappenavn_til_rs2, mappenavn_til_csv, df_stier_rs2filer,
                                  df_stier_csvfiler, df_endrede_attributter_rs2filer, list_which_material,
                                  sti_list_variables_2lines_calculations, x_attributes.copy(),
                                  len(angle_attributes), bool_shall_execute_model_alteration, files_to_skip,
                                  storage_calculation_times)

# geometrical data used in execute_data_storage and execute_data_processing
list_0lines_inside, list_1line_inside, list_2lines_inside, list_excluded_files_2linescalc, list_points_to_check, \
    list_iternumber_0, list_iternumber_1, list_iternumber_2, ll_inner_points = \
    list_of_df_2lines_info[0], list_of_df_2lines_info[1], list_of_df_2lines_info[2], list_of_df_2lines_info[3], \
    list_of_df_2lines_info[4], list_of_df_2lines_info[5], list_of_df_2lines_info[6], list_of_df_2lines_info[7], \
    list_of_df_2lines_info[8]

"""
her lages diskretisering og mesh til alle modellene
"""

mcm.create_mesh(mappenavn_til_rs2, mappenavn_til_csv, df_stier_rs2filer, df_stier_csvfiler, path_rs2, time0,
                files_to_skip, bool_shall_execute_create_mesh, storage_calculation_times)

"""
her kjøres alle kalkulasjonene, med en dynamisk while-løkke slik at når alle kalkulasjonene er ferdig,
så fortsetter scriptet. Det er viktig å sørge for at rs2_compute allerede finner den mappen som filene ligger i.
"""

mcal.calculate(path_rs2_compute, time0, df_filnavn_rs2, sti_til_mappe_for_arbeidsfiler, sti_tolerance_too_high,
               tolerance, number_of_files, bool_shall_execute_calculate, df_koordinater_mus, navn_kol_df_koord_mus,
               bool_stop_to_check_logs, storage_calculation_times)

"""
åpner interpret, der alle resultater som skal benyttes hentes ut og lagres i csv-format
"""

msd.store_data(mappenavn_til_rs2, mappenavn_til_csv, df_stier_rs2filer, df_stier_csvfiler, path_rs2_interpret,
               df_koordinater_mus, navn_kol_df_koord_mus, ant_parametere_interpret,
               parameter_navn_interpret, time0, ll_inner_points, bool_is_first_time_execute_data_store,
               bool_shall_execute_data_store, files_to_skip, storage_calculation_times)

"""
```

```
here the first processing of data, as explained in the thesis in page , is executed and the fetched data is stored
in csv-files
"""

medp.execute_data_processing(parameter_navn_interpret, mappenavn_til_rs2, mappenavn_til_csv,
                             df_stier_csvfiler, list_points_to_check, sti_til_mapper_endelige_filer,
                             list_excluded_files_2linescalc, list_valnavn, sti_values_toplot, list_0lines_inside,
                             list_1line_inside, parameters_varied, true_lengths, bool_shall_execute_data_processing,
                             files_to_skip, storage_calculation_times)

category = 'end of script'
# this will only be calculated if the script is not interrupted
mmf.calculate_computation_time(time_start, category, storage_calculation_times)
# if so, add the time used for each operation, which will give a good estrimate of the total computation time.
```

```python
import pandas as pd
import pyautogui as pag

"""
This script tracks mouse coordinates, adds metadata to the coordinate, and stores it to a csv-file

"""


def mouse_tracker(path, bool_shall_reedit_mouse_coordinates):
    navn_kolonne = ['Handling', 'x', 'y']
    command = 'n'  # spore musas koordinater
    while bool_shall_reedit_mouse_coordinates:
        try:
            command = input("Vil du spore musas koordinater? j for ja og n for nei: ")
            if command == 'j':
                while True:
                    try:
                        command = input('Plasser mus over ønsket felt og press j: ')
                        if command == 'j':
                            break
                        else:
                            print('j for ja og n for nei, prøv igjen!')
                    except NameError:
                        print("Oppstod en feil!")
                        continue
                break
            elif command == 'n':
                break
            else:
                print('j for ja og n for nei, prøv igjen!')
        except NameError:
            print("Oppstod en feil!")
            continue

    if command == 'n':
        print('Ingen posisjon ble lagret!')
    else:
        teller = 0
        liste = [[], [], [], []]
        while command == 'j':

            x, y = pag.position()
            liste[0].append(input('Gi beskrivelse av handling: '))
            liste[1].append(x)
            liste[2].append(y)

            while True:
                try:
                    command2 = input('Ønsker du å slette lagret punkt? j for ja og n for nei: ')
                    if command2 == 'j':
                        del liste[0][-1]
                        del liste[1][-1]
                        del liste[2][-1]
                        break
                    elif command2 == 'n':
                        teller += 1
                break
            else:
                print('j for ja og n for nei fjompenisse!')
        except NameError:
            print('Det er et eller anna som har gått gæli!')

    while True:
        try:
            command = input('Plasser mus over ønsket posisjon og press j for å lagre, eller n for å avslutte: ')
            if command == 'j':
                break
            elif command == 'n':
                break
            else:
                print('j for ja og n for nei, prøv igjen!')
        except NameError:
            print("Oppstod en feil!")
            continue

  data = {navn_kolonne[0]: liste[0], navn_kolonne[1]: liste[1], navn_kolonne[2]: liste[2]}
  df = pd.DataFrame(data)
  df.to_csv(path, mode='a', sep=';', index=False, header=False)
return
```

```
import itertools
from csv import writer
from sys import exit

import numpy as np

"""
This modulet is made to detect which defined parameters that is to be static and which to be dynamic, create the
ranges of each parameter, for then to create a set of values defining each model which is stored in a csv-file.

This script is called by the main file

"""

"""
This function gets the range of each parameter. If the range is greater than two it is dynamic, is it one it is static,
and is it zero it is not included. This is the function that is doing the detection work.
It is called in the function set_model_csv_attributes below
"""


def get_shape_matrix(rock_mass_material, weakness_zone_material, stress_ratio, overburden,
                     thickness_attributes, angle_attributes, y_attributes, x_attributes):
    """

    @param rock_mass_material:
    @param weakness_zone_material:
    @param stress_ratio:
    @param overburden:
    @param thickness_attributes:
    @param angle_attributes:
    @param y_attributes:
    @param x_attributes:
    @return:
    """
    iter_list = [rock_mass_material, weakness_zone_material, stress_ratio, overburden,
                 thickness_attributes, angle_attributes, y_attributes, x_attributes]
    shape_matrix_list = []
    for iter_object in iter_list:
        if isinstance(iter_object, (int, float, str)):
            shape_matrix_list.append(1)
        else:
            shape_matrix_list.append(len(iter_object))
    return shape_matrix_list


"""
This function gets the ranges of the variables consisting of minimum a non-zero value. With one value this value is
put into a vector with length 1. If two or more values a list is created out of the iter_object which must consist of
three values: (a)start value, (b) end value, and (c) the incremental step.
It is calles in the function set_model_csv_attributes below
"""


def get_range_changing_attributes(rock_mass_material, weakness_zone_material, stress_ratio, overburden,
                                  thickness_attributes, angle_attributes, y_attributes, x_attributes):
    """
    @param rock_mass_material:
    @param weakness_zone_material:
    @param stress_ratio:
    @param overburden:
    @param thickness_attributes:
    @param angle_attributes:
    @param y_attributes:
    @param x_attributes:
    @return:
    """
    iter_list = [rock_mass_material, weakness_zone_material, stress_ratio,
                 thickness_attributes, angle_attributes, y_attributes, x_attributes]
    for idx, iter_object in enumerate(iter_list):
        if not isinstance(iter_object, (list, float, int, tuple, str)) or (isinstance(iter_object, (list, tuple)) and
                                                                        len(iter_object) != 3):
            print(
                "Advarsel!!!!! Input må være enten type int, float, list eller tupple. Dessuten må list eller tuple ha "
                "lengde 3 og være på formen [start_element, slutt_element, steg for element]")
            exit()
        elif isinstance(iter_object, (int, float, str)):
            iter_list[idx] = [iter_object]
        else:
            iter_list[idx] = np.arange(iter_object[0], iter_object[1], iter_object[2]).tolist()
    iter_list.insert(3, overburden)

    return iter_list[0], iter_list[1], iter_list[2], iter_list[3], iter_list[4], iter_list[5], iter_list[6], \
        iter_list[7]


"""
The two functions below is intertwined. Together they provide an integer that represents an overburden(ob) value.
0 is ob 25m and 6 is ob 1200m.

!!!!!!!!!!Thus, this must be updated if there are changes in which overburdens to be modelled!!!!!!!!!!!!!!

get_ob_index is included in the function set_model_csv_attributes below
"""


def which_overburden(element):
    if element == 25:
        return 'ob_25'
    elif element == 100:
        return 'ob_100'
    elif element == 200:
        return 'ob_200'
    elif element == 300:
        return 'ob_300'
    elif element == 500:
        return 'ob_500'
    elif element == 800:
        return 'ob_800'
    elif element == 1200:
        return 'ob_1200'
    else:
        return None
```

```python
def get_ob_index(element):
    switcher = {
        'ob_25': 0,
        'ob_100': 1,
        'ob_200': 2,
        'ob_300': 3,
        'ob_500': 4,
        'ob_800': 5,
        'ob_1200': 6,
    }
    return switcher.get(which_overburden(element), None)


"""
Add it is an iterator class. Purpose: iterate over list containing the distance between centre zone and
centre of tunnel and add the half of the thickness of zone such that the lineament closest to tunnel centre is
always on the same spot of a given distance for any thicknesses. Used by get_x_distance below.

Important:
Sentinel: marks end of object and is allocated sentinel = object() if nothing else is defined. Do not change this if
not being sure about it.
"""


class AddIt:

    def __init__(self, iter_object, mektighet, sentinel=object()):
        self.count = 0
        self.len_iter_object = len(iter_object)
        self.iter_object = iter_object
        self.mektighet = mektighet
        self.sentinel = sentinel

    def __iter__(self):
        return self

    def __next__(self):
        if self.count >= self.len_iter_object:
            return self.sentinel
        if self.count == 0:
            ret = self.iter_object[self.count]
        else:
            ret = self.iter_object[self.count] + self.mektighet / 2
        self.count += 1
        return ret

    __call__ = __next__


"""get_x_distance uses add it class to iterate over the distances given from the experiment setup, leading to proper
changes. The reason is given in the description of the class. This function is used by set_model_csv_attributes_batch
below.
"""
```

```
def get_x_distance(normalized_distance_list, zone_angle, mektighet):
    zone_angle = np.deg2rad(zone_angle)
    sentinel = object()
    ai = AddIt(normalized_distance_list, mektighet, sentinel)
    x_distance_list = []
    for i in iter(ai, sentinel):
        x_distance_list.append(round(i / np.sin(zone_angle), 2))
    return x_distance_list


"""
This function writes the set of values defining each modeland writes it to a specified csv-file.
"""


def set_model_csv_attributes_batch(path_csv_attributes, rock_mass_material, weakness_zone_material, stress_ratio,
                                   overburden, thickness_attributes, angle_attributes, y_attributes, x_attributes):
    # rmm, wzm, sr, ob, m, v, y, x = get_range_changing_attributes(rock_mass_material, weakness_zone_material,
    #                                                               stress_ratio,
    #                                                               overburden, thickness_attributes, angle_attributes,
    #                                                               y_attributes, x_attributes)
    rmm, wzm, sr, ob, m, v, y, x = [rock_mass_material], [weakness_zone_material], [stress_ratio], overburden, \
                                   [thickness_attributes], angle_attributes, [y_attributes], x_attributes
    shape_matrix_list = get_shape_matrix(rmm, wzm, sr, ob, m, v, y, x)
    number_of_files = 0
    with open(path_csv_attributes, 'w', newline='') as file:
        writer_object = writer(file, delimiter=";")
        list_data = [['bm', 'ss', 'k', 'od', 'm', 'v', 'y', 'x']]
        for idx in itertools.product(*[range(s) for s in shape_matrix_list]):
            rmm_i, wzm_i, sr_i, ob_i, m_i, v_i, y_i, x_i = idx
            x_true = get_x_distance(normalized_distance_list=x.copy(), zone_angle=v[v_i], mektighet=m[m_i])
            list_data.append(['{}'.format(rmm[rmm_i]), '{}'.format(wzm[wzm_i]),
                              '{}'.format(sr[sr_i]), '{}'.format(ob[ob_i]),
                              '{}'.format(m[m_i]), '{}'.format(v[v_i]),
                              '{}'.format(y[y_i]), '{}'.format(x_true[x_i])])
            number_of_files += 1
        writer_object.writerows(list_data)
        file.close()
    return number_of_files
```

```
import os
import re
import shutil as st
import time
from datetime import timedelta
from subprocess import check_call

import numpy as np
import pandas as pd
import psutil
import pyautogui as pag
from colorama import Fore
from colorama import Style


"""
Two purposes of this module.

1) To stor functions that are general and can be used by several of the other modules
2) To contain functions that prepare the main file to conduct the automation operations. Most of these functions
   are os-based functions.

"""



"""
get_screen_width gets screen size and monitors it
"""



def get_screen_width(bool_know_size_monitor):
    if bool_know_size_monitor is False:
        return
    screenwidth, screenheight = pag.size()  # the size of main monitor
    print("the size of the main monitor is [" + str(screenwidth) + ", " + str(screenheight) + "].")
    return


"""
get_time_increments returns a list of time-increments used to make artificial breaks when any pag.'function'
is to be used. It is commented in my thesis the reason for this at page 78.
The full name of the thesis is given in the head of the main file.
"""



def get_time_increments():
    time_list = [0, 0.5, 1, 2, 5]
    return time_list


"""
pause_script is used whenever it is handy to force the script to wait. When j is given as a command in python console
the scripts continues.
"""



def procede_script():
    while True:
        try:
            command = input('fortsette script? j for ja: ')
            if command == 'j' or command == 'n':
                return command
            else:
                print('j for ja din nisse!')
        except NameError:
            print('implementert verdi ukjent')
            continue
    return


"""make_file_empty deletes the content of a file"""


def make_file_empty(file):
    with open(file, 'w'):
        pass
    return


"""
calculate_computation_time, calculates the computation time when called, monitored in python console
used after each bigger operation in main, such as after calculation of the models
"""


def calculate_computation_time(time_start, category, storage_calculation_times):
    if is_file_empty(storage_calculation_times):
        mode = 'w'
    else:
        mode = 'a'
    time_mid = time.time()
    time_diff_in_seconds = time_mid - time_start
    time_diff_in_hours_min_sec = timedelta(seconds=time_diff_in_seconds)
    line = "Tid brukt for kjøring av script etter {}:     {}. (#hours#:#minutes#:#seconds#)\n" \
        .format(category, time_diff_in_hours_min_sec)
    print(line)
    with open(storage_calculation_times, mode=mode) as file:
        file.writelines(line)
    return


"""
Checks if process has started to run
"""


def check_if_process_running(process_name):
    """
    Check if there is any running process that contains the given name processName.
    """
    # Iterate over the all the running process
    for proc in psutil.process_iter():
        try:
            # Check if process name contains the given name string.
            if process_name.lower() in proc.name().lower():
                return True
        except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
            pass
    return False


"""
gets the PID number of
"""


def get_pid(procname):
    for proc in psutil.process_iter():
        if proc.name() == procname:
            return proc.pid

    return None


"""
this function copies a variable to clipboard
"""


def copy2clip(txt):
    cmd = 'echo ' + txt.strip() + '|clip'
    return check_call(cmd, shell=True)


"""
separates parameters of the sensitivity study that are lists from the parameters that are single values. Used in main
line 119
"""


def separate_lists_and_values(list_attributes):
    checker = ['bm', 'ss', 'k', 'od', 'm', 'v', 'y', 'x']
    res = [elem for elem in zip(list_attributes, checker) if (isinstance(elem[0], list) and elem[1] not in
                                                    ['bm', 'ss', 'k', 'od'])]
    res = list(zip(*res))
    return list(res[0]), list(res[1])


"""
is_file_empty check if a specific file is empty
"""


def is_file_empty(file_path):
    """ Check if file is empty by confirming if its size is 0 bytes"""
    # Check if file exist and it is empty
    return os.path.exists(file_path) and os.stat(file_path).st_size == 0


"""
get_file_paths_batch gets the paths of the template models used to create the fea models of the experiment, and the
```

```python
path to the csv where the model definition of each model is stored.
"""


def get_file_paths_batch(paths_shale_rs2, path_csv):
    overburdens = [25, 100, 200, 300, 500, 800, 1200]
    sti_kildefil_rs2 = []
    sti_kildefil_csv = []
    for i in overburdens:
        sti_kildefil_rs2.append(
            paths_shale_rs2.format(i, i, i))
        sti_kildefil_csv.append(path_csv)
    return sti_kildefil_rs2, sti_kildefil_csv


"""get file paths of already existing fea-files"""


def get_old_paths_df(sti_csv_gamle_rs2stier, sti_csv_gamle_csvstier):
    # df_gamle_rs2filer/csvfiler:
    # er en dataframe som inneholder stiene lest fra sti_csv_gamle_rs2stier
    df_gamle_stier_rs2filer = pd.read_csv(sti_csv_gamle_rs2stier, sep=';')
    df_gamle_stier_csvfiler = pd.read_csv(sti_csv_gamle_csvstier, sep=';')
    # Tomme elementer får verdien None.
    df_gamle_stier_rs2filer = df_gamle_stier_rs2filer.fillna(np.nan).replace([np.nan], [None])
    df_gamle_stier_csvfiler = df_gamle_stier_csvfiler.fillna(np.nan).replace([np.nan], [None])
    df_stier_rs2filer = df_gamle_stier_rs2filer.copy()
    df_stier_csvfiler = df_gamle_stier_csvfiler.copy()
    return df_stier_rs2filer, df_stier_csvfiler


"""
get file paths of fea-files and csv to store ecxcavation query data. If bool_create_new_project True, old content is
deleted and new files and folder system is created, if False the olde ystem is kept and the file paths is fetched.
The fea files, if bool_create_new_project True, is created by copying the template files.
"""


def get_paths_df(sti_til_mappe_for_arbeidsfiler, sti_til_mapper_endelige_filer, sti_kildefil_rs2, sti_kildefil_csv,
                 sti_csv_gamle_rs2stier, sti_csv_gamle_csvstier, parameter_verdier_mappenavn,
                 ytre_grenser_utstrekning, bool_create_new_project):
    if bool_create_new_project:
        # I create_folders:
        # vil mapper fra forrige prosjekt eventuelt bli slettet og mappene
        # til det nye prosjekt blir laget hvis dette er tilfelle
        delete_and_create_folders(sti_til_mappe_for_arbeidsfiler, sti_til_mapper_endelige_filer,
                                  parameter_verdier_mappenavn)
        # mappenavn_til_rs2, mappenavn_til_csv = get_name_folders(sti_til_mapper_endelige_filer)
        # copy_and_store:
        # lager alle kopiene av kildefilene og lagrer filene i rett mappe.
        # Dette gjøres ved å bruke mappenavn og filnavn som markør.
        df_nye_stier_rs2filer, df_nye_stier_csvfiler = copy_and_store(sti_kildefil_rs2,
                                                                      sti_til_mappe_for_arbeidsfiler,
                                                                      sti_til_mapper_endelige_filer,
                                                                      sti_kildefil_csv,
                                                                      ytre_grenser_utstrekning)
        # to_csv:
```

```
        # Her blir alle stiene til de nylagete rs2-filene lagret.
        df_nye_stier_rs2filer.to_csv(alternate_slash([sti_csv_gamle_rs2stier])[0], sep=';')
        df_nye_stier_csvfiler.to_csv(alternate_slash([sti_csv_gamle_csvstier])[0], sep=';')
        df_stier_rs2filer = df_nye_stier_rs2filer.copy()
        df_stier_csvfiler = df_nye_stier_csvfiler.copy()
    else:
        df_stier_rs2filer, df_stier_csvfiler = get_old_paths_df(sti_csv_gamle_rs2stier, sti_csv_gamle_csvstier)

    return df_stier_rs2filer, df_stier_csvfiler


"""
create_work_and_storage_folders creates the folders for storing the fea-files to be created and its results
is they do not exist.
"""


def create_work_and_storage_folders(sti_til_mappe_for_arbeidsfiler, sti_til_mapper_endelige_filer):
    # hvis mapper for arbeidsfiler og lagring av resultater ikke eksisterer så lages dette
    if not os.path.exists(alternate_slash([sti_til_mappe_for_arbeidsfiler])[0]):
        os.mkdir(alternate_slash([sti_til_mappe_for_arbeidsfiler])[0])
    if not os.path.exists(alternate_slash([sti_til_mapper_endelige_filer])[0]):
        os.mkdir(alternate_slash([sti_til_mapper_endelige_filer])[0])
    return


"""make_file_name har til hensikt å lage de filnavn tilhørende RS2-prosjekter
funksjonen skal returnere en liste med  alle filnavn på denne formen her:

Input:

parameter_navn:
er en liste av strenger som inneholder de parametere som er definert i excel-fila anvist over, foruten
geometri som ikke har en tallverdi knyttet til seg.
parameter_verdier_excel:
er en streng som inneholder pathen til excel-fila der verdiene til de tilhørende parameternavnene er lagret.
geometri:
inneholder informasjon om tunnelens geometri: s for sirkulær og hs for hestesko. Denne er satt deafult til sikrulær


Andre parametere: df_verdier: dataframe som inneholder verdiene som tilhører de ulike parameternavnene
file_name_liste: liste som tilslutt inneholder alle filnavnene tilhørende rs2-prosjektene som skal opprettes. path:
er variabelen som brukes til å bygge opp hver enkelt streng som tilsammen resulterer i et filnavn for et gitt
rs2-prosjekt.

returnerer:
file_name_list"""


def make_file_name(parameter_verdier_csv, geometri='S'):
    df_verdier = pd.read_csv(parameter_verdier_csv, sep=';')
    parameter_navn = df_verdier.columns.values.tolist()
    file_name_rs2_list = []
    file_name_csv_list = []
    for i in range((df_verdier.shape[0])):
        path = ''
        path += (geometri + "_")
        for navn in parameter_navn:
            path += (navn + str(df_verdier[navn][i]) + "_")
        path = path[:-1]
        path1 = path
        path += '.fea'  # denne er eneste forskjellen mellom make_folder_name og make_file_name
        path1 += '.csv'
        file_name_rs2_list.append(path)
        file_name_csv_list.append(path1)
    return file_name_rs2_list, file_name_csv_list


"""beskrivelsen for denne er identisk med den over bare at denne er tilpasset for mappenavn
parameter_verdier_excel har en annen sti og det blir ikke lagt til .fea i enden av navnet."""


def make_folder_name(parameter_verdier_mappenavn, geometri='S'):
    df_verdier = pd.read_csv(parameter_verdier_mappenavn, sep=';')
    parameter_navn = df_verdier.columns.values.tolist()
    folder_name_list = []
    for i in range((df_verdier.shape[0])):
        path = ''
        path += (geometri + "_")
        for navn in parameter_navn:
            path += (navn + str(df_verdier[navn][i]) + "_")
            if navn == 'y' or navn == 'x':
                if df_verdier[navn][i] == 0:
                    path = path.replace((navn + str(df_verdier[navn][i]) + "_"), "")
        path = path[:-1]
        folder_name_list.append(path)
    return folder_name_list


"""
delete_and_create_folders asks if the user wants to delete content already stored in the working directory. If yes,
it deletes the old and creates
"""


def delete_and_create_folders(storage_path, storage_final_folders, parameter_verdier_mappenavn):
    storage_path = alternate_slash([storage_path])[0]
    storage_final_folders = alternate_slash([storage_final_folders])[0]
    folder_names = make_folder_name(parameter_verdier_mappenavn)
    folder_paths = folder_names
    for i in range(len(folder_names)):
        folder_paths[i] = (storage_final_folders + '/' + folder_names[i] + '/')
    while True:
        try:
            x = input('Vil du slette mapper/filer på denne stien? ')
            if x == 'j':
                if os.path.exists(storage_path):
                    st.rmtree(storage_path, ignore_errors=True)  # folder_paths[i]
                    os.mkdir(storage_path)
                if os.path.exists(storage_final_folders):
                    st.rmtree(storage_final_folders, ignore_errors=True)
                    os.mkdir(storage_final_folders)
                for folder in folder_paths:
                    os.mkdir(os.path.join(storage_final_folders, folder))
                    os.mkdir(os.path.join(folder, folder + '/csv/'))
                    os.mkdir(os.path.join(folder, folder + '/rs2/'))
                break
            elif x == 'n':
                if not os.path.exists(storage_final_folders):
                    for folder in folder_paths:
                        os.mkdir(os.path.join(storage_final_folders, folder))
                        os.mkdir(os.path.join(folder, folder + '/csv/'))
                        os.mkdir(os.path.join(folder, folder + '/rs2/'))
                break
            else:
                print('j for ja din fjomp!')
        except NameError:
            print('implementert verdi ukjent')
            continue
    return


"""hensikten med get_path_folders er å hente stier på de mapper som RS2-filene skal bli lagret i
mappenavn kommer til å være definert utifra de parametere som til en hver tid holdes konstant/skiftes sjeldent,
samt en hovedsti.


parametere:
main_path:
er en streng og er pathen til der hvor de ulike mappene og filene skal lagres. Må sørge for at det er en
backslash i slutten av strengen
list_folders:
liste som inneholder navnene til alle mapper som er relevante for lagring av de produserte RS2-filer


returnerer:
list_folders"""


def get_name_folders(path_storage_files):
    path_storage_files = alternate_slash([path_storage_files])[0]
    list_folders = os.listdir(path_storage_files)  # henter de mappenavn som ligger i main_path
    # list_folders.pop(0)
    # list_folders.pop(0)
    list_folders.sort(key=len)  # sørger for at mappenavnene blir sortert i stigende rekkefølge
    list_csv_folders, list_rs2_folders = [s + '/csv/' for s in list_folders], [s + '/rs2/' for s in list_folders]
    # må endres hvis mappestrukturen endres!!!!!!
    return list_rs2_folders, list_csv_folders


"""copy_and_store:
er en funksjon som tar get_path_folders, make_file_name og alternate_slash i bruk.
Hensikten er å gjøre klar alle filer som skal igjennom sine respektive endringer i RS2.
Gi de tenkte RS2-filene navn som tilsvarer hvordan modellen i et gitt tilfelle skal se ut,
så bestemme i hvilken mappe en gitt fil tilhører for tilslutt å kopiere kildefila n ganger, der hver kopi blir
tilordnet hver sin sti. Mao. ingen av filene er klargjort etter å ha kalt på denne funksjonen
Denne funksjonen oppretter også et excel ark til hver enkelt fil. Der skal lister med resultater lagres.


input:
de som er gitt i funksjonene over.
```

```
path_file0 viser til kildefilens sti


andre parametere:
list_name_folders:
Liste av navnene til mappene som RS2_fil_stiene skal lagres i
list_rs2_file_names:
Liste over alle stiene til RS2_filene
df_name_files:
En pandas dataframe som er en 2-dimensjonal matrise med mange tillegsfunksjoner.
I denne er rs2_fil-lagret lagret i de mappene de hører hjemme ved at rs2_filnavnet blir tilordnet den kolonne
som har den rette label gitt av mappenavnet.
df_list_path_files:
Er eksakt lik som df_name_files bare at denne inneholder stien til RS2-fila.


returnerer:
navnet til mappene og dataframe med filplasseringene, samt dataframe med filnavnene."""


def copy_and_store(path_file0_rs2, path_storage_files, path_final_folders, parameter_verdier_csv,
                   ytre_grenser_utstrekning, geometri='S'):
    path_file0_rs2 = [alternate_slash([path])[0] for path in
                      path_file0_rs2]  # alternate_slash kun laget for å funke på lister
    name_rs2_folders, name_csv_folders = get_name_folders(path_final_folders)
    path_storage_files = alternate_slash([path_storage_files])[0]
    df_name_rs2_files = pd.DataFrame(columns=name_rs2_folders)
    df_name_csv_files = pd.DataFrame(columns=name_csv_folders)
    for rs2, csv, attributes in zip(name_rs2_folders, name_csv_folders,
                                    parameter_verdier_csv):  # sammenlikner mappenavn med RS2-fil-navn.
        list_rs2_file_names, list_csv_file_names = make_file_name(attributes, geometri)
        rs21 = rs2.replace('/rs2/', '')
        res_rs2 = [i for i in list_rs2_file_names if rs21 in i]  # Når det matcher blir filnavnet lagret i kolonna til
        csv1 = csv.replace('/csv/', '')  # mappenavnet. Lagres i en dataframe.
        res_csv = [i for i in list_csv_file_names if csv1 in i]
        df_name_rs2_files.loc[:, rs2] = pd.Series(res_rs2, dtype=str)
        df_name_csv_files.loc[:, csv] = pd.Series(res_csv, dtype=str)
    df_name_rs2_files = df_name_rs2_files.fillna(np.nan).replace([np.nan], [None])  # Tomme elementer får verdien None.
    df_name_csv_files = df_name_csv_files.fillna(np.nan).replace([np.nan], [None])
    df_list_path_rs2 = df_name_rs2_files.copy()
    df_list_path_csv = df_name_csv_files.copy()
    i = 0
    for k, (rs2, csv, ytre_grense) in enumerate(zip(name_rs2_folders, name_csv_folders, ytre_grenser_utstrekning)):
        # tilordner filnavn sine stier og copierer mal. Tomme elementer forblir tomme.
        for file in df_list_path_rs2.index.values:
            if df_name_rs2_files[rs2][file] is not None and df_name_csv_files[csv][file] is not None:
                df_list_path_rs2.loc[file, rs2] = path_storage_files + '/' + df_name_rs2_files[rs2][file]
                df_list_path_csv.loc[file, csv] = path_storage_files + '/' + df_name_csv_files[csv][file]
                if ytre_grense == ytre_grenser_utstrekning[i - 1]:
                    continue
                else:
                    st.copyfile(path_file0_rs2[i], df_list_path_rs2[rs2][file])
                pd.DataFrame({}).to_csv(df_list_path_csv[csv][file])
        i += 1
    return df_list_path_rs2, df_list_path_csv
```

```python
"""get_changing_attribute:
henter ut de attributter som skal endres i RS2-fila, disse blir returnert som en streng


input:
df_name_files:
en dataframe som inneholder alle filnavn kategorisert etter mappe. Hver mappe har sin egen kolonne
Tomme celler har verdien None.
folder_names:
innehar navnene på hver enkelt mappe


andre parametere:
df_marker_of_change:
dataframe som til slutt kun innehar informasjonen om hvordan hver enkelt fil skal endres.
Denne informasjonen er selv lagret i en dataframe. 1. kolonne inneholder type-informasjon
og 2. kolonne inneholder verdien til denne typen. Tilsammen beskriver en rad i df hvilken endring
som gjelder for en bestemt type.


returnerer:
df_marker_of_change"""


def get_changing_attributes(df_path_files, folder_names):
    df_marker_of_change = df_path_files.copy()  # copy for at lhs blir uavhengig av rhs
    for folder in folder_names:
        for file in df_path_files.index.values:
            if df_path_files[folder][file] is None:  # hoppe over tomme plasseringer
                continue
            y = df_path_files[folder][file].rsplit('/', 1)[0] + '/' + folder.replace('/rs2/', '') + '_'
            x = df_path_files[folder][file].replace(y, '')
            x = x.replace('.fea', '')
            num = []
            char = []
            while len(x) != 0:
                x = x.partition('_')
                q = str(x[0])
                num1 = re.findall(r'[+-]?\d+\.?\d*', q)[0]
                char.append(q.replace(num1, ''))
                num.append(num1)
                x = x[2]
            attributes = pd.Series(data=num, name='values', index=char)
            df_marker_of_change.at[file, folder] = attributes
    return df_marker_of_change


"""alternate_slash bytter ut alle bakstreker i hver streng av en liste med strenger og gjør dem om til skråstreker
og omvendt.
hensikt: formatet til stiene som skiller mappenavn med bakstreker blir ikke forstått av python som forstår skråstreker
        Derfor er det nødvendig med en kornvertering.
Hvis input ikke kun består av stier der mappenavn blir kun skilt av bakstreker eller skråstreker,
så stopper funksjonen å kjøre og ingen endring blir oppnådd.


input:
list_path:
```

```
liste over stier til gitte objekter.


andre parametere:
find_backslash:
for hver sti som inneholder bakstrek blir et nytt element lagt til i denne lista.
find_frontslash:
for hver sti som inneholder skråstrek blir et nytt element lagt til i denne lista.


returnerer:
-1 hvis en feil har oppstått
list_path hvis alt gikk bra"""


def alternate_slash(list_path):
    try:
        backslash = r"\ "
        backslash = backslash[:-1]
        find_backslash = [i for i in list_path if backslash in i]
        find_frontslash = [j for j in list_path if '/' in j]
        if len(find_backslash) == len(list_path) and len(find_frontslash) == 0:  # kjører kun hvis inputfiler
            find_backslash = [sub.replace(backslash, '/') for sub in find_backslash]  # er i rett format (se over)
            list_path = find_backslash
            for i in range(len(list_path)):
                if list_path[i][-1] == ' ':  # sørger for at formatet til strengen blir rett etter konverteringen
                    list_path[i] = list_path[i][:-1]  # python klarer ikke å lese filplasseringer som er delt med en
                # if path[-1] != '/':               # enkelt bakstrek.
                #     path += '/'
        elif len(find_backslash) == 0 and len(find_frontslash) == len(list_path):  # kjører kun hvis inputfiler
            find_frontslash = [sub.replace('/', backslash) for sub in find_frontslash]  # # er i rett format (se over)
            list_path = find_frontslash
        else:
            print(f'{Fore.RED}Feil!  Listen av stier er ikke ensartet med skråstrek eller ensartet med bakstrek, eller '
                  f'så er det ikke en liste. Funksjonen ble derfor stoppet '
                  f'stoppet{Style.RESET_ALL}')  # Fore og style sørger for feilmelding med rød skrift
            return -1
    except TypeError:
        print(f'{Fore.RED}Feil!  Input er enten ikke en liste, eller en liste av strenger.{Style.RESET_ALL}')
        return -1  # Fore og style sørger for feilmelding med rød skrift

    return list_path
```

```
import re
import shutil as st
import time

import numpy as np
import pandas as pd

from Automatisering_RS2 import module_main_functions as mmf, module_model_construction_rs2 as mmcr

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)

"""
This module does two things. First of all, it creates the geometry and material allocation of a given model and stores
the geomtrical data of this model needed for later operations in the main file.

Second of all, if called it returnes lists of the stored geometrical data needed in later operations of the main file.
This makes it possible to reuse experiments without doing all the geometry and material allocations.
"""

"""
alter_geometry executes all alterations to be made for each .fea-file. It reads the respective filenames
which consists of the information of the value of each parameter which defines the file. After this, it implements these
values into the respective fea file by the Open-constructor.

zone is used interchangably with weakness zone
"""



def alter_geometry(zone_angle, x_translation_zone, y_translation_zone, thickness_zone, path_of_rs2_file,
                   list_which_material, _0lines_inside, _1line_inside, _2lines_inside,
                   _excluded_files_2linescalc, iterationnumber, _points_to_check, path_of_csv_file,
                   l_inner_points, _iternumber_0, _iternumber_1, _iternumber_2, x_attribute, num_outerpoints_zone=4,
                   num_lines_zone=2, tunnel_diameter=10, n_points_tunnel_boundary=360, extension_outerboundary=150):
    """

    @param zone_angle:
    @param x_translation_zone:
    @param y_translation_zone:
    @param thickness_zone:
    @param path_of_rs2_file:
    @param list_which_material:
    @param _0lines_inside:
    @param _1line_inside:
    @param _2lines_inside:
    @param _excluded_files_2linescalc:
    @param iterationnumber:
    @param _points_to_check:
    @param path_of_csv_file:
    @param l_inner_points:
    @param _iternumber_0:
    @param _iternumber_1:
    @param _iternumber_2:
    @param x_attribute:
    @param num_outerpoints_zone:
    @param num_lines_zone:
```

```
@param tunnel_diameter:
@param n_points_tunnel_boundary:
@param extension_outerboundary:
@return:
"""

"""
Data from the .fea gile is extracted and prepared for the alternation proccess.
"""

# fetches the content of .fea file and store it as a list.
with open(path_of_rs2_file, 'r') as file:
    data = file.readlines()

# fetches the key-elements to be used to navigate in the list data. These keywords were found by examining the
# fea file using notepad++.
index_material_mesh = data.index("materials mesh start:\n") + 3
index_boundary1 = data.index("  boundary 1 start:\n") + 6
points_tunnel_boundary0 = data[index_boundary1:index_boundary1 + n_points_tunnel_boundary].copy()

# making list of the points defining the tunnel stored as float:
points_tunnel_boundary = mmcr.prep_points_tunnel_boundary(points_tunnel_boundary0, data, index_boundary1)

# defines which points in tunnel boundary that belongs to a specific mathematical quadrant. There are four in trotal
fourth_quad = points_tunnel_boundary[0:int(n_points_tunnel_boundary / 4)]
first_quad = points_tunnel_boundary[int(n_points_tunnel_boundary / 4):int(n_points_tunnel_boundary / 2)]
second_quad = points_tunnel_boundary[int(n_points_tunnel_boundary / 2):int(n_points_tunnel_boundary * (3 / 4))]
third_quad = points_tunnel_boundary[int(n_points_tunnel_boundary * (3 / 4)):n_points_tunnel_boundary]

# saves the categorized points in a list called quad
quad = (fourth_quad, first_quad, second_quad, third_quad)

"""
In this section the change in thickness, rotation and translation is executed
"""

# thickness of weakness zone is changed before translation and rotation of the zone.
# defines the weakness zone based on thickness and on the intersection between the zone and the outer boundary.
# the zone is horizontal at first.
x_right_def_zone = extension_outerboundary
x_left_def_zone = -extension_outerboundary
y_top_def_zone = thickness_zone / 2
y_bot_def_zone = -thickness_zone / 2

# defines the four points defining the zone. They work as pairs where each pair defines a line. Together, the two
# lines spans out the zone. In the script they are systemized based on the sequence of points defining the tunnel.
# There point zero is the point at the bottom of the circle and the sequence goes counter-clockwise.
# Also, the third value of the lists is set to 1, to include the translation in the matrix equation.

point_bot_right = [x_right_def_zone, y_bot_def_zone, 1]
point_top_right = [x_right_def_zone, y_top_def_zone, 1]
point_top_left = [x_left_def_zone, y_top_def_zone, 1]
point_bot_left = [x_left_def_zone, y_bot_def_zone, 1]

# instability occurs if the zone angle is too close [90,180,270,360] for some reason.
# mmcr.prepare_angel solves that issue by making the angle close to the wanted value.
# See modules_model_construction line 10 for more.
```

```python
zone_angle = mmcr.prepare_angel(zone_angle)
# converts the zone angle to radians
zone_angle_rad = np.deg2rad(zone_angle)

# defines the conjoined translation and rotation matrix in where rotation happens first, then translation, or else,
# no translation would happen.
rottrans_matr = np.array([[np.cos(zone_angle_rad), -np.sin(zone_angle_rad), x_translation_zone],
                          [np.sin(zone_angle_rad), np.cos(zone_angle_rad), y_translation_zone]])

# the rotation of the zone and translation is done simultaniously.
# After rotation and translation the zone must be extended to intersect the outer boundary.
if zone_angle != 0:
    # the rotation and translation of zone is done using np.matmul which does matrix product of two arrays
    outerpoint_bot_right0 = np.matmul(rottrans_matr, np.array(point_bot_right))
    outerpoint_top_right0 = np.matmul(rottrans_matr, np.array(point_top_right))
    outerpoint_top_left0 = np.matmul(rottrans_matr, np.array(point_top_left))
    outerpoint_bot_left0 = np.matmul(rottrans_matr, np.array(point_bot_left))

    # the finnished transformed points is categorised into the two line segments
    points_top = [outerpoint_top_right0, outerpoint_top_left0]
    points_bot = [outerpoint_bot_right0, outerpoint_bot_left0]

    # check if zone exeeds the outer boundary, which crashes RS2. If so the code is stopped.
    if mmcr.OuterBoundary.check_points_ob(points_top, points_bot, extension_outerboundary):
        print('Advarsel: svakhetssonen er utenfor de ytre grensene tilhørende modellen. Scriptet ble stanset.')
        quit()
    # the extension of the zone is executed
    outerpoint_top_right, outerpoint_top_left = \
        mmcr.OuterBoundary.find_points_on_outer_boundary(outerpoint_top_right0, outerpoint_top_left0,
                                                         extension_outerboundary, num_lines_zone)
    outerpoint_bot_right, outerpoint_bot_left = \
        mmcr.OuterBoundary.find_points_on_outer_boundary(outerpoint_bot_right0, outerpoint_bot_left0,
                                                         extension_outerboundary, num_lines_zone)
    # checking if the extension is succesfull
    len_topp = np.sqrt(
        (outerpoint_bot_left[0] - outerpoint_bot_right[0]) ** 2 +
        (outerpoint_bot_left[1] - outerpoint_bot_right[1]) ** 2)
    len_bunn = np.sqrt(
        (outerpoint_top_left[0] - outerpoint_top_right[0]) ** 2 +
        (outerpoint_top_left[1] - outerpoint_top_right[1]) ** 2)

    if len_topp < 5 or len_bunn < 5:
        print('Advarsel: svakhetssonens utstrekning er for liten, prøv igjen. Scriptet ble stanset.')
        quit()
else:
    # defining zone if no rotation, only thickness is changed and true length is added to y,
    # the reason for this is that this situation can not be normalized since an x translation is not possible when
    # zone has no angle. See explaination for the term normalized in main file, line 244.
    outerpoint_bot_right = [x_right_def_zone, y_bot_def_zone + x_attribute]
    outerpoint_top_right = [x_right_def_zone, y_top_def_zone + x_attribute]
    outerpoint_top_left = [x_left_def_zone, y_top_def_zone + x_attribute]
    outerpoint_bot_left = [x_left_def_zone, y_bot_def_zone + x_attribute]

    # check if zone exeeds the outer boundary, which crashed RS2. If so the code is stopped.
    if outerpoint_bot_right[1] < -extension_outerboundary or outerpoint_top_right[1] > extension_outerboundary:
        print('Advarsel: svakhetssonen er utenfor de ytre grensene tilhørende modellen. Scriptet ble stanset.')
        quit()
    # the outer points of weakness zone is stored in a list
    outer_points = [outerpoint_bot_right, outerpoint_top_right, outerpoint_top_left, outerpoint_bot_left]

    """
    It must also be defined inner points of the weakness zone if the zone intersects the tunnel periphery. It will be
    zero, two or four points defined, depending if the none, one or two lineaments defing the zone are intersecting.

    The reason for this is due to RS2. If this is not defined, the material allocation will not work.
    """

    # an object is constructed containing all functions and variables needed to define the inner intersectionpoints
    # of a model. If there are innerpoints, theese are created and implemented in the fea-file in the section regarding
    # the specification of the tunnel boundary. See, the class mmcr.InnerBoundary in modules_model_construction_rs2.py
    # line 38 # for details.
    ib = mmcr.InnerBoundary(num_lines_zone, quad, outer_points, data, n_points_tunnel_boundary,
                            index_boundary1, tunnel_diameter, zone_angle, points_tunnel_boundary,
                            extension_outerboundary)
    if ib.inner_points:
        inner_points = ib.get_points_on_circular_boundary_2()
        ib.set_inner_boundary()
    else:
        inner_points = ib.get_points_on_circular_boundary_2()

    """
    In this section, the outerpoints defining the weakness zone is implemented into the.fea-file in the section
    regarding the defintion of the outerboundary.
    """

    # the tokens needed to navigate the list data to implement the changes is defined.
    index_boundary1 = data.index("  boundary 1 start:\n") + 6
    index_boundary2 = data.index("  boundary 2 start:\n") + 6
    index_boundary3 = data.index("  boundary 3 start:\n") + 6
    index_boundary4 = data.index("  boundary 4 start:\n") + 6

    # creation of the object implements the changes. The details are given in modules_model_construction.py in the class
    # OuterBoundary in line 344.
    mmcr.OuterBoundary(outer_points, data, index_boundary2, zone_angle, num_outerpoints_zone,
extension_outerboundary)

    """
    In this section, the outerpoints defining the weakness zone is implemented into the.fea-file in the section
    regarding the defintion of the weakness zone. See modules_model_construction class BoundaryLines line 474
    for details.
    """
    bl = mmcr.BoundaryLines(outer_points, inner_points, index_boundary3, index_boundary4, data, num_lines_zone)
    bl.set_weakness_points()

    """
    In this section the material allocations is done. See modules_model_construction class Materials line 752
    for details.
    """

    # the updated list of the points defining the tunnel boundary is needed due to creation of new points above.
    points_tunnel_boundary0 = data[index_boundary1:index_boundary1 + ib.n_points_ib].copy()
    # making list of points stored as float:
    points_tunnel_boundary = mmcr.prep_points_tunnel_boundary(points_tunnel_boundary0, data, index_boundary1)
```

```python
    # allocating the materials.
    mls = mmcr.Materials(index_material_mesh, inner_points, extension_outerboundary,
                         num_lines_zone, quad, outer_points, data, n_points_tunnel_boundary, index_boundary1,
                         tunnel_diameter, zone_angle, points_tunnel_boundary, y_translation_zone, x_translation_zone,
                         list_which_material)
    mls.setmaterialmesh()

    """
    In this section the alterations is written into the given models fea-file
    """

    with open(path_of_rs2_file, 'w') as file:
        file.writelines(data)

    """
    in this section, the geometry data needed in later operations of the main file is stored in lists later to be stored
    in csv or pickle format
    """
    if all(points is None for points in inner_points):
        _0lines_inside.append([path_of_rs2_file, path_of_csv_file])
        _iternumber_0.append(iterationnumber)
        _excluded_files_2linescalc.append(iterationnumber)
        # _points_to_check.append(None)
    elif any(points is None for points in inner_points):
        _1line_inside.append([path_of_rs2_file, path_of_csv_file])
        _excluded_files_2linescalc.append(iterationnumber)
        _iternumber_1.append(iterationnumber)
        # _points_to_check.append(None)
    else:
        _2lines_inside.append([path_of_rs2_file, path_of_csv_file])
        _points_to_check.append(inner_points)
        _iternumber_2.append(iterationnumber)
    l_inner_points.append(inner_points)
    return


"""
create_pickle_2lines_info stores the necessary geometry data used by operations succeding ea.execute_model_alteration
It is called from ea.execute_model_alteration in experiment_actions in line 109.

A pickle is a file format created to store python objects in the format of the python language. For example, pandas
datdaframes must be stored in this format if to be used later in another run or other scripts. If the dataframe was
stored in the csv format, the dataframe was altered in a awy that the information was destroyed.
"""


def create_pickle_2lines_info(list_0lines_inside, list_1line_inside, list_2lines_inside, list_excluded_files_2linescalc,
                              list_points_to_check, sti_list_variables_2lines_calculations, mappenavn_til_rs2,
                              list_iternumber_0, list_iternumber_1, list_iternumber_2, ll_inner_points):
    iterable = [list_0lines_inside, list_1line_inside, list_2lines_inside, list_excluded_files_2linescalc,
                list_points_to_check, list_iternumber_0, list_iternumber_1, list_iternumber_2, ll_inner_points]
    list_of_df_2lines_info, colnames_of_dfs_2lines_info = [], []

    for sti_variables, it in zip(sti_list_variables_2lines_calculations, iterable):
        # list_of_df_2lines_info.append([]), colnames_of_dfs_2lines_info.append([])
        d = {navn.replace('/rs2', ''): i for navn, i in zip(mappenavn_til_rs2, it)}
        df = pd.DataFrame({k: pd.Series(v) for k, v in d.items()})
        df.to_pickle(path=sti_variables)
        list_of_df_2lines_info.append(df), colnames_of_dfs_2lines_info.append(df.head())
    return list_of_df_2lines_info, colnames_of_dfs_2lines_info


"""
gets values of material parameters and geometry for a rs2-model extracted from its filename and calls
alter_geometry
"""


def alter_model(extension_outerboundary, n_points_tunnel_boundary,
                path_of_rs2_file, path_of_csv_file, df_changing_attributes_rs2files,
                foldername_of_pathcategory, list_which_material, _0lines_inside, _1line_inside, _2lines_inside,
                _excluded_files_2linescalc, _points_to_check, i, j, _iternumber_0, _iternumber_1, _iternumber_2,
                l_inner_points, x_attributes):
    """

    @param extension_outerboundary:
    @param n_points_tunnel_boundary:
    @param path_of_rs2_file:
    @param path_of_csv_file:
    @param df_changing_attributes_rs2files:
    @param foldername_of_pathcategory:
    @param list_which_material:
    @param _0lines_inside:
    @param _1line_inside:
    @param _2lines_inside:
    @param _excluded_files_2linescalc:
    @param _points_to_check:
    @param i:
    @param j:
    @param _iternumber_0:
    @param _iternumber_1:
    @param _iternumber_2:
    @param l_inner_points:
    @param x_attributes:
    @return:
    """
    # fetching which values to be changed for a specific model
    angle = df_changing_attributes_rs2files[foldername_of_pathcategory[i]][j]['v']
    angle = float(angle)
    x_attribute = x_attributes[j]
    y_translation_zone = float(df_changing_attributes_rs2files[foldername_of_pathcategory[i]][j]['y'])
    x_translation_zone = float(df_changing_attributes_rs2files[foldername_of_pathcategory[i]][j]['x'])
    thickness_zone = float(df_changing_attributes_rs2files[foldername_of_pathcategory[i]][j]['m'])

    # function which executes the alterations by changing certain lines of the .fea-files using the Open-constructor
    alter_geometry(angle, x_translation_zone, y_translation_zone, thickness_zone, path_of_rs2_file,
                   list_which_material, _0lines_inside, _1line_inside, _2lines_inside,
                   _excluded_files_2linescalc, i, _points_to_check, path_of_csv_file,
                   _iternumber_0, _iternumber_1, _iternumber_2, l_inner_points, x_attribute,
                   extension_outerboundary=extension_outerboundary,
                   n_points_tunnel_boundary=n_points_tunnel_boundary)
    return
```

```
"""
get_parameters_2lines_inside is called if there there are going to be done operations on files that already have their
geometry been defined in an earlier run of the script. For example, if the mesh of the models must be refined, leading
to new calculations and fetching of values with interpret, this function can be called. The geometrical data needed
in the comming operations is stored in pickle-files in the end of alter_geometry.
"""


def get_parameters_2lines_inside(sti_list_variables_2lines_calculations):
    list_of_df_2lines_info = []
    colnames_of_dfs_2lines_info = []
    for sti_variables_2lines in sti_list_variables_2lines_calculations:
        df = pd.read_pickle(filepath_or_buffer=sti_variables_2lines)
        list_of_df_2lines_info.append(df)
        colnames_of_df = df.head()
        colnames_of_dfs_2lines_info.append(colnames_of_df)
    return list_of_df_2lines_info, colnames_of_dfs_2lines_info


"""
execute_model_alteration either sequence through each model and creates them based opn the content of their filenames
calling alter_model, or it retrieves the necessary geometry data for the rest of the operations of the main file calling
get_parameters_2lines_inside.
"""


def execute_model_alteration(ytre_grenser_utstrekning, n_points_tunnel_boundary, overburdens, list_change_fieldstress,
                             mappenavn_til_rs2, mappenavn_til_csv, df_stier_rs2filer,
                             df_stier_csvfiler, df_endrede_attributter_rs2filer, list_which_material,
                             sti_list_variables_2lines_calculations, x_attributes, len_angle_attributes,
                             bool_shall_execute_model_alteration, files_to_skip, storage_calculation_times):
    """

    @param ytre_grenser_utstrekning:
    @param n_points_tunnel_boundary:
    @param overburdens:
    @param list_change_fieldstress:
    @param mappenavn_til_rs2:
    @param mappenavn_til_csv:
    @param df_stier_rs2filer:
    @param df_stier_csvfiler:
    @param df_endrede_attributter_rs2filer:
    @param list_which_material:
    @param sti_list_variables_2lines_calculations:
    @param x_attributes:
    @param len_angle_attributes:
    @param bool_shall_execute_model_alteration:
    @param files_to_skip:
    @param storage_calculation_times:
    @return:
    """
    if bool_shall_execute_model_alteration is True:
        time_operation = time.time()
        category = 'geometri'

        # x_attributes is used if zone angle is zero and must be repeated as defined under, to work with alter_model
        x_attributes = len_angle_attributes * x_attributes
```

```
# lists to contain geometry data later to be used in data_storage and data_processing
list_0lines_inside, list_1line_inside, list_2lines_inside, list_iternumber_0, list_iternumber_1, \
   list_iternumber_2, list_excluded_files_2linescalc, list_points_to_check, ll_inner_points = \
   [], [], [], [], [], [], [], [], []
q = 0
for k, (navn_rs2, navn_csv, utskrekning, overdekning) in enumerate(zip(mappenavn_til_rs2, mappenavn_til_csv,
                                                                       ytre_grenser_utstrekning, overburdens)):
   if k in files_to_skip:
      continue
   list_0lines_inside.append([]), list_1line_inside.append([]), list_2lines_inside.append([]),
   list_excluded_files_2linescalc.append([]), list_points_to_check.append([]),
   list_iternumber_0.append([]), list_iternumber_1.append([]), list_iternumber_2.append([]),
   ll_inner_points.append([])
   p = 2
   if utskrekning == ytre_grenser_utstrekning[k - 1]:
      q += 1
      overburden = list_change_fieldstress[q - 1]
      if q == 1:
         print('ferdig å undersøke filer?')
         mmf.procede_script()

      list_0lines_inside[k], list_1line_inside[k], list_2lines_inside[k], \
         list_excluded_files_2linescalc[k], list_points_to_check[k], list_iternumber_0[k], \
         list_iternumber_1[k], list_iternumber_2[k], ll_inner_points[k] = \
         list_0lines_inside[p], list_1line_inside[p], list_2lines_inside[p], \
         list_excluded_files_2linescalc[p], list_points_to_check[p], list_iternumber_0[p], \
         list_iternumber_1[p], list_iternumber_2[p], ll_inner_points[p]

      list_0lines_inside[k] = [[paths[0].replace('od200', 'od{}'.format(overdekning)),
                                paths[1].replace('od200', 'od{}'.format(overdekning))]
                               for paths in list_0lines_inside[k]]
      list_1line_inside[k] = [[paths[0].replace('od200', 'od{}'.format(overdekning)),
                               paths[1].replace('od200', 'od{}'.format(overdekning))]
                              for paths in list_1line_inside[k]]
      list_2lines_inside[k] = [[paths[0].replace('od200', 'od{}'.format(overdekning)),
                                paths[1].replace('od200', 'od{}'.format(overdekning))]
                               for paths in list_2lines_inside[k]]
      for j in range(df_stier_rs2filer.shape[0]):
         path_fil_rs2 = df_stier_rs2filer[navn_rs2][j]
         path_fil_csv = df_stier_csvfiler[navn_csv][j]
         if isinstance(path_fil_rs2, str):
            path_to_copy_rs2 = df_stier_rs2filer['S_bm80_ss1_k1_od200/rs2/'][j]
            path_to_copy_csv = df_stier_csvfiler['S_bm80_ss1_k1_od200/csv/'][j]
            st.copyfile(path_to_copy_rs2, path_fil_rs2)
            st.copyfile(path_to_copy_csv, path_fil_csv)
      key_word = 'field stress:\n'
      for j in range(df_stier_rs2filer.shape[0]):
         path_fil_rs2 = df_stier_rs2filer[navn_rs2][j]
         if isinstance(path_fil_rs2, str):
            with open(path_fil_rs2, 'r') as file:
               data = file.readlines()
               index_fieldstress = data.index(key_word) + 1
               data[index_fieldstress] = re.sub(r'^(\s*(?:\S+\s+){5})\S+', r'\1 '
                                                + str('{}'.format(overburden)), data[index_fieldstress])
            with open(path_fil_rs2, 'w') as file:
               file.writelines(data)
```

```
        else:
            for j in range(df_stier_rs2filer.shape[0]):
                path_fil_rs2 = df_stier_rs2filer[navn_rs2][j]
                path_fil_csv = df_stier_csvfiler[navn_csv][j]
                print(path_fil_rs2)
                # print(path_fil_csv)
                if isinstance(path_fil_rs2, str) and isinstance(path_fil_csv, str):
                    alter_model(utskrekning, n_points_tunnel_boundary,
                                path_fil_rs2, path_fil_csv, df_endrede_attributter_rs2filer,
                                mappenavn_til_rs2, list_which_material, list_0lines_inside[k],
                                list_1line_inside[k], list_2lines_inside[k], list_excluded_files_2linescalc[k],
                                list_points_to_check[k], k, j, list_iternumber_0[k], list_iternumber_1[k],
                                list_iternumber_2[k], ll_inner_points[k], x_attributes)
                else:
                    ll_inner_points[k].append(None)
    list_of_df_2lines_info, colnames_of_dfs_2lines_info = \
        create_pickle_2lines_info(list_0lines_inside, list_1line_inside, list_2lines_inside,
                                  list_excluded_files_2linescalc, list_points_to_check,
                                  sti_list_variables_2lines_calculations, mappenavn_til_rs2,
                                  list_iternumber_0, list_iternumber_1, list_iternumber_2, ll_inner_points)
    mmf.calculate_computation_time(time_operation, category, storage_calculation_times)
else:
    list_of_df_2lines_info, colnames_of_dfs_2lines_info = \
        get_parameters_2lines_inside(sti_list_variables_2lines_calculations)

return list_of_df_2lines_info, colnames_of_dfs_2lines_info
```

```
import re

import numpy as np
from sympy import Point, Segment, Circle, geometry, Line


def prepare_angel(angel):
    if not -1 < angel / 360 < 1:
        if angel % 360 != 0:
            angel = angel % 360
        else:
            angel = 0
    if 89.999 < angel < 90.111 or -89.999 > angel > -90.111:
        angel = 89.99
    elif 179.9 < angel < 180.1 or -179.9 > angel > -180.1:
        angel = 179
    elif 269.9 < angel < 270.1 or -269.9 > angel > -270.1:
        angel = 269
    elif angel in [10, -10, 170, -170]:
        angel = angel + 1
    return angel


def prep_points_tunnel_boundary(points_tunnel_boundary0, data, index_boundary1):
    points_tunnel_boundary = []
    for index, points in enumerate(points_tunnel_boundary0):
        data[index_boundary1 + index] = re.sub(r'^(\s*(?:\S+\s+){0})\S+', r'\1 ' + str(index) + ':',
                                               data[index_boundary1 + index])
        points_string = re.findall(r"[-+]?(?:\d*\.\d+|\d+\b(?!:))", points)
        points = [float(points_string[0]), float(points_string[1])]
        points_tunnel_boundary.append(points)
    return points_tunnel_boundary


class InnerBoundary:
    # inner boundary er en class der hensikten er å tilordne korrekt plassering for de indre punktene som beskriver
    # svakhetssonen i rs2. Dette skal kun inntreffe hvis sonen treffer tunnelen. Med andre ord, denne classen må ta
    # høyde for situasjon der enten 0, 1, 2 etc. linjeelement treffer tunnelåpningen.
    def __init__(self, ant_linjer, nth_quad, punkter_ytre, data, number_points_inner_boundary, index_boundary1,
                 diameter, vinkel, points_tunnel_boundary, ytre_grenser_utstrekning):
        self.nth_quad = nth_quad
        self.punkter_ytre = punkter_ytre.copy()
        self.data = data
        self.n_points_ib = number_points_inner_boundary
        self.index_boundary1 = index_boundary1
        self.ant_linjer = ant_linjer
        self.diameter = diameter
        self.vinkel = vinkel
        self.points_tunnel_boundary = points_tunnel_boundary.copy()
        self.ytre_grenser = ytre_grenser_utstrekning
        circle = self.get_tunnel_esc_circle_sympy()
        seg1, seg2 = Segment(self.punkter_ytre[0], self.punkter_ytre[3]), \
            Segment(self.punkter_ytre[1], self.punkter_ytre[2])
        seg_check = [seg1, seg2]
        inside = [self.is_line_inside_circle_sympy(seg, circle) for seg in seg_check]
        if any(inside):
            self.inner_points = True
    else:
        self.inner_points = False

def get_middle_points(self):
    p = []
    for i in range(self.ant_linjer):
        k = [3, 1]
        p.append(((self.punkter_ytre[i][0] + self.punkter_ytre[i + k[i]][0]) / 2,
                  (self.punkter_ytre[i][1] + self.punkter_ytre[i + k[i]][1]) / 2))
    return p

@staticmethod
def which_quad(punkt):
    if punkt[0] > 0 and punkt[1] >= 0:
        return '1st quad'
    elif punkt[0] <= 0 and punkt[1] > 0:
        return '2nd quad'
    elif punkt[0] < 0 and punkt[1] <= 0:
        return '3rd quad'
    elif punkt[0] >= 0 and punkt[1] < 0:
        return '4th quad'
    else:
        return None

def get_quad_index(self, punkt):
    switcher = {
        '4th quad': 0,
        '1st quad': 1,
        '2nd quad': 2,
        '3rd quad': 3,
    }
    return switcher.get(self.which_quad(punkt), None)

@staticmethod
def which_outer_boundary_point(element):
    if element == 0 or element == 3:
        return 'nedre grense'
    elif element == 1 or element == 2:
        return 'ovre grense'
    else:
        return None

def get_indices_outer_boundary(self, element):
    switcher = {
        'nedre grense': (0, 3),
        'ovre grense': (1, 2),
    }
    index_point = switcher.get(self.which_outer_boundary_point(element), None)
    return index_point

def get_index_lowest_diff_points(self, quad_index, punkt):
    diff_nth = [abs(np.sqrt(
        (punkt[0] - float(point[0])) ** 2 + (punkt[1] - float(point[1])) ** 2)) for
        point in self.nth_quad[quad_index]]
    sorted_diff_nth = sorted(diff_nth)
    index_lowest_diff = [diff_nth.index(sorted_diff_nth[1]),
                         diff_nth.index(sorted_diff_nth[0])]
        return index_lowest_diff

    def get_linfunc_outer_boundary(self, element):
        indices_points_outer_boundary = self.get_indices_outer_boundary(element)
        a_line = (self.punkter_ytre[indices_points_outer_boundary[0]][1] -
                  self.punkter_ytre[indices_points_outer_boundary[1]][1]) / (
                      self.punkter_ytre[indices_points_outer_boundary[0]][0] -
                      self.punkter_ytre[indices_points_outer_boundary[1]][0])
        b_line = self.punkter_ytre[indices_points_outer_boundary[0]][1] - a_line * \
            self.punkter_ytre[indices_points_outer_boundary[0]][0]
        return a_line, b_line

    def get_tunnel_esc_circle_sympy(self):
        rad = self.diameter/2
        circle = Circle((0, 0), rad)
        return circle

    def get_outerboundary_line_seg_sympy(self, element):
        indices_points_outer_boundary = self.get_indices_outer_boundary(element)
        point_left = Point(self.punkter_ytre[indices_points_outer_boundary[1]][0],
                           self.punkter_ytre[indices_points_outer_boundary[1]][1])
        point_right = Point(self.punkter_ytre[indices_points_outer_boundary[0]][0],
                            self.punkter_ytre[indices_points_outer_boundary[0]][1])
        seg = Segment(point_left, point_right)
        return seg

    @staticmethod
    def is_line_inside_circle_sympy(seg, circle):
        if circle.intersection(seg):
            return True
        else:
            return False

    def is_line_inside_circle(self, a_line, b_line):
        epsilon = 10 ** -13
        a = a_line ** 2 + 1
        b = 2 * a_line * b_line
        c = b_line ** 2 - self.diameter ** 2 / 4
        test = b ** 2 - 4 * a * c
        if test > epsilon:
            return True
        else:
            return False

    @staticmethod
    def float_of_rational(rational):
        _float = rational.p/rational.q
        return _float

    def calculate_inner_points_sympy(self, seg, circle):
        a, b = geometry.intersection(circle, seg)
        a, b = a.evalf(), b.evalf()
        a, b = [float(a[0]), float(a[1])], \
               [float(b[0]), float(b[1])]
        return b, a

    def calculate_inner_points(self, a_line, b_line):
        a = a_line ** 2 + 1
        b = 2 * a_line * b_line
        c = b_line ** 2 - self.diameter ** 2 / 4
        x_pos = (-b + np.sqrt(b ** 2 - 4 * a * c)) / (2 * a)
        x_neg = (-b - np.sqrt(b ** 2 - 4 * a * c)) / (2 * a)
        y_pos = a_line * x_pos + b_line
        y_neg = a_line * x_neg + b_line
        point_pos = [x_pos, y_pos]
        point_neg = [x_neg, y_neg]
        return point_pos, point_neg

    def get_theoretical_inner_points(self, element):
        a_line_nedre, b_line_nedre = self.get_linfunc_outer_boundary(element)
        point_pos, point_neg = self.calculate_inner_points(a_line_nedre, b_line_nedre)
        points = [point_pos, point_neg]
        return points

    def get_theoretical_inner_points_sympy(self, element):
        seg = self.get_outerboundary_line_seg_sympy(element)
        circle = self.get_tunnel_esc_circle_sympy()
        point_pos, point_neg = self.calculate_inner_points_sympy(seg, circle)
        points = [point_pos, point_neg]
        return points

    def calculate_intersection(self, punkt, element):
        quad_index = self.get_quad_index(punkt)
        index_lowest_diff = self.get_index_lowest_diff_points(quad_index, punkt)
        a_circ = (self.nth_quad[quad_index][index_lowest_diff[1]][1] - self.nth_quad[quad_index][index_lowest_diff[0]][
            1]) / (
                         self.nth_quad[quad_index][index_lowest_diff[1]][0] -
                         self.nth_quad[quad_index][index_lowest_diff[0]][0])
        b_circ = self.nth_quad[quad_index][index_lowest_diff[1]][1] - a_circ * \
            self.nth_quad[quad_index][index_lowest_diff[1]][0]
        a_line, b_line = self.get_linfunc_outer_boundary(element)
        x = (b_circ - b_line) / (a_line - a_circ)
        y = (b_circ * a_line - b_line * a_circ) / (a_line - a_circ)
        point = [x, y]
        return point

    def calculate_intersection_sympy(self, punkt, element):
        seg_line = self.get_outerboundary_line_seg_sympy(element)
        quad_index = self.get_quad_index(punkt)
        index_lowest_diff = self.get_index_lowest_diff_points(quad_index, punkt)
        p_circ00 = Point(self.nth_quad[quad_index][index_lowest_diff[3]][0],
                         self.nth_quad[quad_index][index_lowest_diff[3]][1])
        p_circ01 = Point(self.nth_quad[quad_index][index_lowest_diff[2]][0],
                         self.nth_quad[quad_index][index_lowest_diff[2]][1])
        p_circ10 = Point(self.nth_quad[quad_index][index_lowest_diff[0]][0],
                         self.nth_quad[quad_index][index_lowest_diff[0]][1])
        p_circ11 = Point(self.nth_quad[quad_index][index_lowest_diff[1]][0],
                         self.nth_quad[quad_index][index_lowest_diff[1]][1])
        seg_circ0 = Segment(p_circ00, p_circ01)
        seg_circ1 = Segment(p_circ01, p_circ11)
        seg_circ2 = Segment(p_circ11, p_circ10)
        check_circ = [seg_circ0, seg_circ1, seg_circ2]
        point = [seg_circ.intersection(seg_line)[0] for seg_circ in check_circ if seg_circ.intersection(seg_line)]
        point = point[0]
        point = point.evalf()
        point = [float(point[0]), float(point[1])]
        return point

    def get_points_on_circular_boundary_2(self):
        points = []
        # path = mplt_path.Path(self.points_tunnel_boundary)
        for i in range(self.ant_linjer):
            # inside = path.contains_point(((self.punkter_ytre[i][0] + self.punkter_ytre[i + k[i]][0]) / 2,
            #                               (self.punkter_ytre[i][1] + self.punkter_ytre[i + k[i]][1]) / 2))
            a, b = self.get_linfunc_outer_boundary(i)
            if self.is_line_inside_circle(a, b):
                points_theoretical = self.get_theoretical_inner_points(i)
                point = self.calculate_intersection(points_theoretical[0], i)
                point1 = self.calculate_intersection(points_theoretical[1], i)
                points.append(point)
                points.append(point1)
            else:
                point, point1 = None, None
                points.append(point)
                points.append(point1)
        p = points.copy()
        points = [p[0], p[2], p[3], p[1]]
        return points

    def get_points_on_circular_boundary_sympy(self):
        points = []
        circle = self.get_tunnel_esc_circle_sympy()
        for i in range(self.ant_linjer):
            seg = self.get_outerboundary_line_seg_sympy(i)
            if self.is_line_inside_circle_sympy(seg, circle):
                points_theoretical = self.get_theoretical_inner_points_sympy(i)
                point = self.calculate_intersection_sympy(points_theoretical[0], i)
                point1 = self.calculate_intersection_sympy(points_theoretical[1], i)
                points.append(point)
                points.append(point1)
            else:
                point, point1 = None, None
                points.append(point)
                points.append(point1)
        p = points.copy()
        points = [p[0], p[2], p[3], p[1]]
        return points

    def get_start_quad(self, punkt):
        switcher = {
            '4th quad': 0,
            '1st quad': int(self.n_points_ib/4),
            '2nd quad': int(self.n_points_ib/2),
            '3rd quad': int(self.n_points_ib*(3/4)),
        }
        return switcher.get(self.which_quad(punkt), None)

    def sort_boundary_points(self):
        points = self.get_points_on_circular_boundary_2()
        points = [value for value in points if value is not None]
        i_data_list = []
        for i in range(len(points)):
            quad_index = self.get_quad_index(points[i])
            index_lowest_diff = self.get_index_lowest_diff_points(quad_index, points[i])
            index_lowest_diff.sort()
            i_data_list.append(self.index_boundary1 + self.get_start_quad(points[i]) + index_lowest_diff[1])
        i_data_list, points = (list(t) for t in zip(*sorted(zip(i_data_list, points))))
        # self.n_points_ib = self.n_points_ib + len(points)
        return i_data_list, points

    def remove_neighbour(self, i_data, point):
        point_string1 = re.findall(r"[-+]?(?:\d*\.\d+|\d+\b(?!:))", self.data[i_data - 1])
        point_string2 = re.findall(r"[-+]?(?:\d*\.\d+|\d+\b(?!:))", self.data[i_data + 1])
        point_check1 = [float(point_string1[0]), float(point_string1[1])]
        point_check2 = [float(point_string2[0]), float(point_string2[1])]
        len1 = np.sqrt((point[0] - point_check1[0]) ** 2 + (point[1] - point_check1[1]) ** 2)
        len2 = np.sqrt((point[0] - point_check2[0]) ** 2 + (point[1] - point_check2[1]) ** 2)
        if len1 < len2:
            self.data.pop(i_data - 1)
        else:
            self.data.pop(i_data + 1)
        return

    def set_inner_boundary(self):
        i_data_list, points = self.sort_boundary_points()
        # p = 0
        for i_data, point in zip(i_data_list, points):
            self.data.insert(i_data, "        {}: ".format(0) + str(point[0]) + ', ' + str(point[1]) + '\n')
            self.remove_neighbour(i_data, point)
            # p += 1
        # self.data[self.index_boundary1 - 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+', r'\g<1>' + str(self.n_points_ib),
        #                                              self.data[self.index_boundary1 - 1])
        # rette opp i nummerering av punkter
        for index in range(self.n_points_ib):
            self.data[self.index_boundary1 + index] = re.sub(r'^(\s*(?:\S+\s+){0})\S+', r'\g<1>' + str(index) + ':',
                                                             self.data[self.index_boundary1 + index])[1:]
            self.data[self.index_boundary1 + self.n_points_ib + 10 + index] = \
                re.sub(r'^(\s*(?:\S+\s+){1})\S+', r'\g<1>' + str(index), self.data[self.index_boundary1 +
                                                                                    self.n_points_ib + 10 + index])
        return

    def get_index_inner_points(self, tunnel_boundary_points):
        points = self.get_points_on_circular_boundary_2()
        points = [value for value in points if value is not None]
        indices = []
        for point in points:
            idx = tunnel_boundary_points.index(point)
            indices.append(idx)
        return indices


class OuterBoundary:
    def __init__(self, punkter_ytre, data, index_boundary2, vinkel, ant_pkt_ytre, ytre_grenser_utstrekning):
        self.punkter_ytre = punkter_ytre.copy()
        self.data = data
        self.index_boundary2 = index_boundary2
        self.ant_pkt_ytre = ant_pkt_ytre
        self.ytre_grenser = ytre_grenser_utstrekning
    if vinkel != 0:
        self.del_points()
        self.set_points_outer_boundary()

@staticmethod
def get_linfunc(punkter):
    a_line = (punkter[0][1] - punkter[1][1]) / (punkter[0][0] - punkter[1][0])
    b_line = punkter[0][1] - a_line * punkter[0][0]
    return a_line, b_line

@staticmethod
def find_points_on_outer_boundary(point_r, point_l, ytre_grenser, ant_linjer):
    point_r = np.array(point_r)
    point_l = np.array(point_l)
    test = [ytre_grenser, -ytre_grenser]
    a = (point_r[1] - point_l[1]) / (point_r[0] - point_l[0])
    b = point_r[1] - a * point_r[0]
    point = []
    # finder hvilken begrænsning som gjelder for de to punkter som bæskriver linja og lagrer den ferdige
    # beskrivelsen i en vector
    for i in range(ant_linjer):
        y = a * test[i] + b
        x = (test[i] - b) / a
        if abs(y) <= ytre_grenser:
            point.append(np.array([test[i], y]))
        if abs(x) <= ytre_grenser:
            point.append(np.array([x, test[i]]))
    # sørger for at det endrede punkt til høyre er lagret i første element av vektoren
    v = np.sqrt(np.dot(np.linalg.norm(point[0] - point_r), (np.linalg.norm(point[0] - point_r))))
    v2 = min(np.sqrt(np.dot(np.linalg.norm(point[0] - point_r), (np.linalg.norm(point[0] - point_r)))),
             np.sqrt(np.dot(np.linalg.norm(point[1] - point_r), (np.linalg.norm(point[1] - point_r)))))
    if v != v2:
        point = [point[1], point[0]]
    return point[0].tolist(), point[1].tolist()

@staticmethod
def check_points_ob(punkter_over, punkter_under, ytre_grenser):
    a, b = OuterBoundary.get_linfunc(punkter_over)
    c, d = OuterBoundary.get_linfunc(punkter_under)
    func1 = [-abs(a), abs(b)]
    func2 = [-abs(c), abs(d)]
    check = [abs(func1[0] * ytre_grenser + func1[1]), abs((ytre_grenser - func1[1]) / func1[0]),
             abs(func2[0] * ytre_grenser + func2[1]),
             abs((ytre_grenser - func2[1]) / func2[0])]
    if any(z <= ytre_grenser for z in check[0:2]) and any(z <= ytre_grenser for z in check[2:4]):
        return False
    else:
        return True

def which_line(self, element):
    if self.punkter_ytre[element][1] == -self.ytre_grenser and \
            -self.ytre_grenser < self.punkter_ytre[element][0] <= self.ytre_grenser:
        return '1st line'
    elif self.punkter_ytre[element][0] == self.ytre_grenser and \
            -self.ytre_grenser < self.punkter_ytre[element][1] <= self.ytre_grenser:
        return '2nd line'
    elif self.punkter_ytre[element][1] == self.ytre_grenser and \
            -self.ytre_grenser <= self.punkter_ytre[element][0] < self.ytre_grenser:
        return '3rd line'
    elif self.punkter_ytre[element][0] == -self.ytre_grenser and \
            -self.ytre_grenser <= self.punkter_ytre[element][1] < self.ytre_grenser:
        return '4th line'
    else:
        return None

def get_boundary_index(self, element):
    switcher = {
        '1st line': 1,
        '2nd line': 2,
        '3rd line': 3,
        '4th line': 4,
    }
    return switcher.get(OuterBoundary.which_line(self, element), None)

def del_points(self):
    self.data.pop(self.index_boundary2 + 7)
    self.data.pop(self.index_boundary2 + 6)
    self.data.pop(self.index_boundary2 + 3)
    self.data.pop(self.index_boundary2 + 2)

def key_sorter(self, item):
    if item[0] == -self.ytre_grenser and -self.ytre_grenser <= item[1] < self.ytre_grenser:
        return item[1] * -1
    elif item[1] == self.ytre_grenser and -self.ytre_grenser <= item[0] < self.ytre_grenser:
        return item[0] * -1
    elif item[0] == self.ytre_grenser and -self.ytre_grenser < item[1] <= self.ytre_grenser:
        return item[1]
    elif item[1] == -self.ytre_grenser and -self.ytre_grenser < item[0] <= self.ytre_grenser:
        return item[0]
    else:
        return False

def sort_ob_points(self, item):
    item, self.punkter_ytre = (list(t) for t in zip(*sorted(zip(item, self.punkter_ytre), reverse=True)))
    for i in range(4, 0, -1):
        indices = [j for j, x in enumerate(item) if x == i]
        if len(indices) > 1:
            to_sort = self.punkter_ytre[indices[0]:indices[-1] + 1]
            self.punkter_ytre[indices[0]:indices[-1] + 1] = sorted(to_sort, key=self.key_sorter)
    return item

def set_points_outer_boundary(self):
    placement_new_point = []
    for i in range(self.ant_pkt_ytre):
        placement_new_point.append(self.get_boundary_index(i))
    placement_new_point = self.sort_ob_points(placement_new_point)
    dummy = placement_new_point.copy()
    k = 1
    for i in range(0, self.ant_pkt_ytre):
        if i > 0 and placement_new_point[i] == dummy[i - 1]:
            placement_new_point[i] += k
            k += 1
        else:
            k = 1
            self.data.insert(placement_new_point[i] + self.index_boundary2,
                             "       {}: ".format(0) + str(self.punkter_ytre[i][0]) + ', ' + str(
                                 self.punkter_ytre[i][1]) + '\n')
        for index in range(len(self.data[self.index_boundary2:(self.index_boundary2 + 8)])):
            self.data[self.index_boundary2 + index] = re.sub(r'^(\s*(?:\S+\s+){0})\S+', r'\g<1>' + str(index) + ':',
                                                              self.data[self.index_boundary2 + index])
        return


class BoundaryLines:
    def __init__(self, punkter_ytre, punkter_indre, index_boundary3, index_boundary4, data, ant_linjer):
        self.index_boundary3 = index_boundary3
        self.index_boundary4 = index_boundary4
        self.punkter_ytre = punkter_ytre
        self.punkter_indre = punkter_indre
        self.data = data
        self.ant_linjer = ant_linjer

    @staticmethod
    def inner_points_cleanup(punkter):
        for i in range(2):
            v3 = np.sqrt(np.dot(np.linalg.norm(np.array(punkter[0]) - np.array(punkter[1])),
                                (np.linalg.norm(np.array(punkter[0]) - np.array(punkter[1])))))
            v4 = np.sqrt(np.dot(np.linalg.norm(np.array(punkter[0]) - np.array(punkter[2])),
                                (np.linalg.norm(np.array(punkter[0]) - np.array(punkter[2])))))
            if v3 > v4:
                x = punkter.copy()
                punkter[2] = x[1]
                punkter[1] = x[2]
        return punkter

    def sort_weakness_points(self):
        punkter = []
        p = [3, 1]
        for i in range(len(p)):
            punkt = [self.punkter_ytre[i], self.punkter_indre[i], self.punkter_indre[i + p[i]],
                     self.punkter_ytre[i + p[i]]]
            if punkt[1] is not None:
                punkt = self.inner_points_cleanup(punkt)
            punkter.append(punkt)
        return punkter

    def set_weakness_points(self):
        punkter = self.sort_weakness_points()
        if all(elem is None for elem in self.punkter_indre):
            self.set_weakness_exl_inner_points()
        elif all(elem is not None for elem in self.punkter_indre):
            self.set_weakness_with_inner_points(punkter)
        else:
            self.set_weakness_with_inner_point(punkter)
        return

    def set_weakness_with_inner_points(self, punkter):
        punkter_under = punkter[0]
        punkter_over = punkter[1]
        for i in range(len(punkter_over)):
            self.data[self.index_boundary3 + i] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                       r'\g<1>' + str(punkter_under[i][0]) + ',',
                                       self.data[self.index_boundary3 + i])
            self.data[self.index_boundary3 + i] = re.sub(r'^(\s*(?:\S+\s+){2})\S+', r'\g<1>' + str(punkter_under[i][1]),
                                       self.data[self.index_boundary3 + i])
            self.data[self.index_boundary4 + i] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                       r'\g<1>' + str(punkter_over[i][0]) + ',',
                                       self.data[self.index_boundary4 + i])
            self.data[self.index_boundary4 + i] = re.sub(r'^(\s*(?:\S+\s+){2})\S+', r'\g<1>' + str(punkter_over[i][1]),
                                       self.data[self.index_boundary4 + i])
        return

    def set_weakness_with_inner_point(self, punkter):
        index_boundary = [self.index_boundary4, self.index_boundary3]
        support_points = self.weaknesszone_prep()
        # support points bidrar til at svakhetsoner nær periferi blir detektert av rs2. Dette er punkter som ligger
        # på svakhetssonen på halveispunktene mellom midtpunktet til hvert linjeelement og hvert enkelt ytterpunkt.
        p = [3, 1]
        for i in range(len(punkter)):
            if punkter[i][1] is not None:
                for j in range(len(punkter[i])):
                    self.data[index_boundary[i] + j] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                          r'\g<1>' + str(punkter[i][j][0]) + ',',
                                          self.data[index_boundary[i] + j])
                    self.data[index_boundary[i] + j] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                          r'\g<1>' + str(punkter[i][j][1]),
                                          self.data[index_boundary[i] + j])
                self.data.insert(index_boundary[i] + 2 + 15, self.data[index_boundary[i]+15+2])
                self.data.insert(index_boundary[i] + 3, self.data[index_boundary[i]])
                self.data[index_boundary[i] + 3] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                        r'\g<1>' + str(support_points[i][1][0]) + ',',
                                        self.data[index_boundary[i] + 3])
                self.data[index_boundary[i] + 3] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                        r'\g<1>' + str(support_points[i][1][1]),
                                        self.data[index_boundary[i] + 3])

                for idx in range(len(punkter[i])+1):
                    self.data[index_boundary[i] + idx] = re.sub(r'^(\s*(?:\S+\s+){0})\S+',
                                           r'\g<1>' + '{}:'.format(idx),
                                           self.data[index_boundary[i] + idx])
                    self.data[index_boundary[i] + 15 + idx] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                             r'\g<1>' + '{}'.format(idx),
                                             self.data[index_boundary[i] + 15 + idx])
                self.data[index_boundary[i] - 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+', r'\g<1>' +
                                        '{}'.format(len(punkter[i])+1),
                                        self.data[index_boundary[i] - 1])
            else:
                self.data.pop(index_boundary[i] + 17)
                self.data.pop(index_boundary[i] + 2)
                self.data[index_boundary[i]] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                      r'\g<1>' + str(self.punkter_ytre[i][0]) + ',',
                                      self.data[index_boundary[i]])
                self.data[index_boundary[i]] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                      r'\g<1>' + str(self.punkter_ytre[i][1]),
                                      self.data[index_boundary[i]])
                self.data[index_boundary[i] + 1] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                        r'\g<1>' + str(support_points[i][1][0]) + ',',
                                        self.data[index_boundary[i] + 1])
            self.data[index_boundary[i] + 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                   r'\g<1>' + str(support_points[i][1][1]),
                                                   self.data[index_boundary[i] + 1])

            self.data[index_boundary[i] + 2] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                                   r'\g<1>' + str(self.punkter_ytre[i + p[i]][0]) + ',',
                                                   self.data[index_boundary[i] + 2])
            self.data[index_boundary[i] + 2] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                   r'\g<1>' + str(self.punkter_ytre[i + p[i]][1]),
                                                   self.data[index_boundary[i] + 2])
            self.data[index_boundary[i] - 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+', r'\g<1>' + '3',
                                                   self.data[index_boundary[i] - 1])
            for idx in range(3):
                self.data[index_boundary[i] + idx] = re.sub(r'^(\s*(?:\S+\s+){0})\S+',
                                                         r'\g<1>' + '{}:'.format(idx),
                                                         self.data[index_boundary[i] + idx])
        return

    def get_middle_points(self):
        p = []
        for i in range(self.ant_linjer):
            k = [3, 1]
            p.append(((self.punkter_ytre[i][0] + self.punkter_ytre[i + k[i]][0]) / 2,
                      (self.punkter_ytre[i][1] + self.punkter_ytre[i + k[i]][1]) / 2))
        return p

    def set_weakness_exl_inner_points0(self):
        index_boundary = [self.index_boundary4, self.index_boundary3]
        middlepoints = self.get_middle_points()
        p = [3, 1]
        for i in range(len(index_boundary)):
            self.data.pop(index_boundary[i] + 17)
            self.data.pop(index_boundary[i] + 2)
            self.data[index_boundary[i] + 1] = re.sub(r'^(\s*(?:\S+\s+){0})\S+', r'\g<1>' + '1:',
                                               self.data[index_boundary[i] + 1])
            self.data[index_boundary[i] + 1] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                               r'\g<1>' + str(middlepoints[i][0]) + ',',
                                               self.data[index_boundary[i] + 1])
            self.data[index_boundary[i] + 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                               r'\g<1>' + str(middlepoints[i][1]),
                                               self.data[index_boundary[i] + 1])
            self.data[index_boundary[i] + 2] = re.sub(r'^(\s*(?:\S+\s+){0})\S+', r'\g<1>' + '2:',
                                               self.data[index_boundary[i] + 2])
            self.data[index_boundary[i] + 2] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                               r'\g<1>' + str(self.punkter_ytre[i + p[i]][0]) + ',',
                                               self.data[index_boundary[i] + 2])
            self.data[index_boundary[i] + 2] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                               r'\g<1>' + str(self.punkter_ytre[i + p[i]][1]),
                                               self.data[index_boundary[i] + 2])
            self.data[index_boundary[i]] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                           r'\g<1>' + str(self.punkter_ytre[i][0]) + ',',
                                           self.data[index_boundary[i]])
            self.data[index_boundary[i]] = re.sub(r'^(\s*(?:\S+\s+){2})\S+', r'\g<1>' + str(self.punkter_ytre[i][1]),
                                           self.data[index_boundary[i]])
            self.data[index_boundary[i] - 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+', r'\g<1>' + '3',
                                               self.data[index_boundary[i] - 1])
        return
```

```
def set_weakness_exl_inner_points(self):
    index_boundary = [self.index_boundary4, self.index_boundary3]
    support_points = self.weaknesszone_prep()
    # support points bidrar til at svakhetsoner nær periferi blir detektert av rs2. Dette er punkter som ligger
    # på svakhetssonen på halveispunktene mellom midtpunktet til hvert linjeelement og hvert enkelt ytterpunkt.
    p = [3, 1]
    for i in range(len(index_boundary)):
        # self.data.pop(index_boundary[i] + 17)
        # self.data.pop(index_boundary[i] + 2)
        self.data[index_boundary[i]] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                              r'\g<1>' + str(self.punkter_ytre[i][0]) + ',',
                                              self.data[index_boundary[i]])
        self.data[index_boundary[i]] = re.sub(r'^(\s*(?:\S+\s+){2})\S+', r'\g<1>' + str(self.punkter_ytre[i][1]),
                                              self.data[index_boundary[i]])
        self.data[index_boundary[i] + 1] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                                  r'\g<1>' + str(support_points[i][0][0]) + ',',
                                                  self.data[index_boundary[i] + 1])
        self.data[index_boundary[i] + 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                  r'\g<1>' + str(support_points[i][0][1]),
                                                  self.data[index_boundary[i] + 1])
        self.data[index_boundary[i] + 2] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                                  r'\g<1>' + str(support_points[i][1][0]) + ',',
                                                  self.data[index_boundary[i] + 2])
        self.data[index_boundary[i] + 2] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                  r'\g<1>' + str(support_points[i][1][1]),
                                                  self.data[index_boundary[i] + 2])

        self.data[index_boundary[i] + 3] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                                  r'\g<1>' + str(self.punkter_ytre[i + p[i]][0]) + ',',
                                                  self.data[index_boundary[i] + 3])
        self.data[index_boundary[i] + 3] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                  r'\g<1>' + str(self.punkter_ytre[i + p[i]][1]),
                                                  self.data[index_boundary[i] + 3])

        self.data[index_boundary[i] - 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+', r'\g<1>' + '4',
                                                  self.data[index_boundary[i] - 1])
        for idx in range(4):
            self.data[index_boundary[i] + idx] = re.sub(r'^(\s*(?:\S+\s+){0})\S+',
                                                        r'\g<1>' + '{}:'.format(idx),
                                                        self.data[index_boundary[i] + idx])
    return

# def weaknesszone_prep(self):
#     middlepoints = self.get_middle_points()
#     p = []
#     for i in range(self.ant_linjer):
#         k = [3, 1]
#         p1 = ((self.punkter_ytre[i][0] + middlepoints[i][0])*(2/4),
#               (self.punkter_ytre[i][1] + middlepoints[i][1])*(2/4))
#         p2 = ((self.punkter_ytre[i + k[i]][0] + middlepoints[i][0])*(2/4),
#               (self.punkter_ytre[i + k[i]][1] + middlepoints[i][1])*(2/4))
#         p1 = ((p1[0] + middlepoints[i][0]) * (2 / 4),
#               (p1[1] + middlepoints[i][1]) * (2 / 4))
#         p2 = ((p2[0] + middlepoints[i][0]) * (2 / 4),
#               (p2[1] + middlepoints[i][1]) * (2 / 4))
#         p.append([p1, p2])
```

```
    #     return p

    def weaknesszone_prep(self):
        middlepoints = self.get_middle_points()
        p = []
        p1 = ((self.punkter_ytre[0][0] + middlepoints[0][0])*(2/4),
              (self.punkter_ytre[0][1] + middlepoints[0][1])*(2/4))
        p2 = ((self.punkter_ytre[3][0] + middlepoints[0][0])*(2/4),
              (self.punkter_ytre[3][1] + middlepoints[0][1])*(2/4))
        for i in range(2):
            p1 = ((p1[0] + middlepoints[0][0]) * (2 / 4),
                  (p1[1] + middlepoints[0][1]) * (2 / 4))
            p2 = ((p2[0] + middlepoints[0][0]) * (2 / 4),
                  (p2[1] + middlepoints[0][1]) * (2 / 4))
        # p1 = ((p1[0] + self.punkter_ytre[0][0]) * (2 / 4),
        #       (p1[1] + self.punkter_ytre[0][1]) * (2 / 4))
        # p2 = ((p2[0] + self.punkter_ytre[3][0]) * (2 / 4),
        #       (p2[1] + self.punkter_ytre[3][1]) * (2 / 4))
        p3 = self.get_closest_point_on_line(self.punkter_ytre[1], self.punkter_ytre[2], p1)
        p4 = self.get_closest_point_on_line(self.punkter_ytre[1], self.punkter_ytre[2], p2)
        p.append([p1, p2])
        p.append([p3, p4])
        return p

    @staticmethod
    def get_closest_point_on_line(p_line1, p_line2, outer_point):
        x1, y1 = p_line1
        x2, y2 = p_line2
        x3, y3 = outer_point
        dx, dy = x2 - x1, y2 - y1
        det = dx * dx + dy * dy
        a = (dy * (y3 - y1) + dx * (x3 - x1)) / det
        return x1 + a * dx, y1 + a * dy


class Materials(InnerBoundary):
    def __init__(self, index_materials, punkter_indre, ytre_grenser_utstrekning, ant_linjer, nth_quad, punkter_ytre,
                 data,
                 number_points_inner_boundary, index_boundary1,
                 diameter, vinkel, points_tunnel_boundary, forflytning_y_sone, forflytning_x_sone, list_which_material):
        super().__init__(ant_linjer, nth_quad, punkter_ytre, data, number_points_inner_boundary, index_boundary1,
                         diameter, vinkel, points_tunnel_boundary, ytre_grenser_utstrekning, )
        self.index_materials = index_materials
        self.punkter_indre = punkter_indre.copy()
        self.forflytning_y_sone = forflytning_y_sone
        self.forflytning_x_sone = forflytning_x_sone
        self.list_which_material = list_which_material

    def calculate_inner_points_sympy(self, seg, circle):
        a, b = geometry.intersection(circle, seg)
        a, b = a.evalf(), b.evalf()
        a, b = [float(a[0]), float(a[1])], \
               [float(b[0]), float(b[1])]
        return b, a

    def setmaterialmesh(self):
        if all(points is None for points in self.punkter_indre):
            self.__setmaterialmesh0_sympy()
        elif any(points is None for points in self.punkter_indre):
            self.__setmaterialmesh1_sympy()
        else:
            self.__setmaterialmesh2_sympy()
        return

    def __setmaterialmesh0(self):
        del self.data[self.index_materials + 36:self.index_materials + 63]
        i_material = self.index_materials
        mid_points = self.get_middle_points()
        normaler = self.get_normal_lines(mid_points)
        ytre_punkt_under, ytre_punkt_over = self.checker_ob_exl_innerb(normaler)
        list_material = self.list_which_material[0]
        list_iterate, list_iterate1 = self.__get_material_list0(ytre_punkt_under, ytre_punkt_over, list_material)
        self.data[i_material - 2] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                           r'\g<1>' + str(4),
                                           self.data[i_material - 2])
        for i in range(4):
            for j in range(3):
                self.data[i_material + j + 1] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                                       r'\g<1>' + str(list_iterate[i][j][0]) + ',',
                                                       self.data[i_material + j + 1])
                self.data[i_material + j + 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                       r'\g<1>' + str(list_iterate[i][j][1]),
                                                       self.data[i_material + j + 1])
                self.data[i_material + 5] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                   r'\g<1>' + str(list_iterate1[i][0]),
                                                   self.data[i_material + 5])
                self.data[i_material + 6] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                   r'\g<1>' + str(list_iterate1[i][1]),
                                                   self.data[i_material + 6])
            i_material += 9
        return

    def __get_material_list0(self, ytre_punkt_under, ytre_punkt_over, list_material):
        list0 = [[self.punkter_ytre[0], self.punkter_ytre[3], ytre_punkt_under],
                 [self.punkter_ytre[1], self.punkter_ytre[2], ytre_punkt_over],
                 [self.punkter_ytre[3], self.punkter_ytre[2], self.punkter_ytre[1]],
                 [self.points_tunnel_boundary[0], self.points_tunnel_boundary[int(self.n_points_ib / 2)],
                  self.points_tunnel_boundary[int(self.n_points_ib * (3 / 4))]]]
        if self.forflytning_x_sone == 0 and self.forflytning_y_sone == 0:
            list1 = list_material[0]
        else:
            list1 = list_material[1]
        return list0, list1

    def __setmaterialmesh0_sympy(self):
        del self.data[self.index_materials + 36:self.index_materials + 63]
        i_material = self.index_materials
        ytre_punkt_under, ytre_punkt_over = self.checker_ob_sympy()  # sympy Point2D er formatet
        ytre_punkt_under, ytre_punkt_over = list(ytre_punkt_under), list(ytre_punkt_over)
        list_material = self.list_which_material[0]
        list_iterate, list_iterate1 = self.__get_material_list0(ytre_punkt_under, ytre_punkt_over, list_material)
        self.data[i_material - 2] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                           r'\g<1>' + str(4),
                                           self.data[i_material - 2])
        for i in range(4):
            for j in range(3):
                self.data[i_material + j + 1] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                                       r'\g<1>' + str(list_iterate[i][j][0]) + ',',
                                                       self.data[i_material + j + 1])
                self.data[i_material + j + 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                       r'\g<1>' + str(list_iterate[i][j][1]),
                                                       self.data[i_material + j + 1])
                self.data[i_material + 5] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                   r'\g<1>' + str(list_iterate1[i][0]),
                                                   self.data[i_material + 5])
                self.data[i_material + 6] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                   r'\g<1>' + str(list_iterate1[i][1]),
                                                   self.data[i_material + 6])
            i_material += 9
        return

    def __setmaterialmesh1_sympy(self):
        del self.data[self.index_materials + 45:self.index_materials + 63]
        if self.vinkel == 0:
            ytre_punkt_over = [0, self.ytre_grenser]
            ytre_punkt_under = [0, -self.ytre_grenser]
            if self.punkter_indre[0] is not None:
                punkt_i_sone, punkt_u_sone = [0, 5], [0, -5]
            else:
                punkt_i_sone, punkt_u_sone = [0, -5], [0, 5]
        else:
            ytre_punkt_under, ytre_punkt_over = self.checker_ob_sympy()  # sympy Point2D er formatet
            if self.punkter_indre[0] is not None:
                punkt_i_sone, punkt_u_sone = self.checker_ib_sympy(0)
            else:
                punkt_i_sone, punkt_u_sone = self.checker_ib_sympy(1)
        list_iterate = self.__get_material_list1(ytre_punkt_under, ytre_punkt_over, punkt_i_sone, punkt_u_sone)
        list_iterate1 = self.list_which_material[1]
        i_material = self.index_materials
        self.data[i_material - 2] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                           r'\g<1>' + str(5),
                                           self.data[i_material - 2])
        for i in range(5):
            for j in range(3):
                self.data[i_material + j + 1] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                                       r'\g<1>' + str(list_iterate[i][j][0]) + ',',
                                                       self.data[i_material + j + 1])
                self.data[i_material + j + 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                       r'\g<1>' + str(list_iterate[i][j][1]),
                                                       self.data[i_material + j + 1])
            self.data[i_material + 5] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                               r'\g<1>' + str(list_iterate1[i][0]),
                                               self.data[i_material + 5])
            self.data[i_material + 6] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                               r'\g<1>' + str(list_iterate1[i][1]),
                                               self.data[i_material + 6])
            i_material += 9
        return

    def __setmaterialmesh1(self):
        del self.data[self.index_materials + 45:self.index_materials + 63]
        if self.vinkel == 0:
            ytre_punkt_over = [0, self.ytre_grenser]
            ytre_punkt_under = [0, -self.ytre_grenser]
            if self.punkter_indre[0] is not None:
                punkt_i_sone, punkt_u_sone = [0, 5], [0, -5]
            else:
                punkt_i_sone, punkt_u_sone = [0, -5], [0, 5]
        else:
            middlepoints = self.get_middle_points()
            normaler = self.get_normal_lines(middlepoints)
            under, over = middlepoints[0], middlepoints[1]
            if self.origo_is_between(under, over):
                ytre_punkt_under = self.checker_ob(normaler[0], 0)
                ytre_punkt_over = self.checker_ob(normaler[1], 1)
            else:
                ytre_punkt_under, ytre_punkt_over = self.checker_ob_exl_innerb(normaler)
            if self.punkter_indre[0] is not None:
                punkt_i_sone, punkt_u_sone = self.checker_ib(normaler[0], 0)
            else:
                punkt_i_sone, punkt_u_sone = self.checker_ib(normaler[1], 1)
        list_iterate = self.__get_material_list1(ytre_punkt_under, ytre_punkt_over, punkt_i_sone, punkt_u_sone)
        list_iterate1 = self.list_which_material[1]
        i_material = self.index_materials
        self.data[i_material - 2] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                            r'\g<1>' + str(5),
                                            self.data[i_material - 2])
        for i in range(5):
            for j in range(3):
                self.data[i_material + j + 1] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                                        r'\g<1>' + str(list_iterate[i][j][0]) + ',',
                                                        self.data[i_material + j + 1])
                self.data[i_material + j + 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                        r'\g<1>' + str(list_iterate[i][j][1]),
                                                        self.data[i_material + j + 1])
            self.data[i_material + 5] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                r'\g<1>' + str(list_iterate1[i][0]),
                                                self.data[i_material + 5])
            self.data[i_material + 6] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                r'\g<1>' + str(list_iterate1[i][1]),
                                                self.data[i_material + 6])
            i_material += 9
        return

    def __get_opposite_circb_point(self):
        q = [elem for elem in self.punkter_indre if elem is not None]
        r = [round(q[0][0], 17), round(q[0][1], 17)]
        r = self.points_tunnel_boundary.index(r)
        s = [round(q[1][0], 17), round(q[1][1], 17)]
        s = self.points_tunnel_boundary.index(s)
        u = round((r + s) / 2)
        point = self.points_tunnel_boundary[u]
        return point

    def __get_material_list1(self, ytre_punkt_under, ytre_punkt_over, punkt_i_sone, punkt_u_sone):
        # q til u, finne et punkt på tunnel boundary som gjør at materialmeshet for dette materialet blir veldefinert
        p = [3, 2]
        if self.punkter_indre[0] is not None:
            i = 0
        else:
            i = 1
        _list = [[self.punkter_ytre[0], self.punkter_ytre[3], ytre_punkt_under],
                 [self.punkter_ytre[1], self.punkter_ytre[2], ytre_punkt_over],
                 [self.punkter_indre[i], self.punkter_indre[p[i]], punkt_i_sone],
                 [self.punkter_indre[i], self.punkter_indre[p[i]], punkt_u_sone],
                 [self.punkter_ytre[3], self.punkter_ytre[2], self.punkter_indre[p[i]]]]
        return _list

    def get_middle_point(self, element):
        k = [3, 1]
        middlepoint = ((self.punkter_ytre[element][0] + self.punkter_ytre[element + k[element]][0]) / 2,
                       (self.punkter_ytre[element][1] + self.punkter_ytre[element + k[element]][1]) / 2)
        return middlepoint

    def get_middle_points_inner(self):
        p = []
        k = [3, 1]
        for i in range(self.ant_linjer):
            p.append(((self.punkter_indre[i][0] + self.punkter_indre[i + k[i]][0]) / 2,
                      (self.punkter_indre[i][1] + self.punkter_indre[i + k[i]][1]) / 2))
        return p

    @staticmethod
    def get_normal_line(a0, midpoint):
        a, b = -(1 / a0), midpoint[1] + (midpoint[0] / a0)
        return [a, b]

    @staticmethod
    def origo_is_between(middlepoint_under, middlepoint_over):
        origo = [0, 0]
        epsilon = 10 ** -13
        crossproduct = (origo[1] - middlepoint_under[1]) * (middlepoint_over[0] - middlepoint_under[0]) - (
                origo[0] - middlepoint_under[0]) * (middlepoint_over[1] - middlepoint_under[1])
        # compare versus epsilon for floating point values, or != 0 if using integers
        if abs(crossproduct) > epsilon:
            return False
        dotproduct = (origo[0] - middlepoint_under[0]) * (middlepoint_over[0] - middlepoint_under[0]) + (
                    origo[1] - middlepoint_under[1]) * (middlepoint_over[1] - middlepoint_under[1])
        if dotproduct < 0:
            return False
        squaredlengthba = (middlepoint_over[0] - middlepoint_under[0]) ** 2 + (
                    middlepoint_over[1] - middlepoint_under[1]) ** 2
        if dotproduct > squaredlengthba:
            return False
        return True

    def get_normal_lines(self, mid_points):
        a0, b0 = self.get_linfunc_outer_boundary(0)
        normal_under = self.get_normal_line(a0, mid_points[0])
        c0, d0 = self.get_linfunc_outer_boundary(1)
        normal_over = self.get_normal_line(c0, mid_points[1])
        return [normal_under, normal_over]

    @staticmethod
    def key_checker(elem):
        return abs(elem[0])

    def checker_ob(self, normal, element):
        middlepoint = self.get_middle_point(element)
        points = [[self.ytre_grenser, normal[0] * self.ytre_grenser + normal[1]],
                  [(self.ytre_grenser - normal[1]) / normal[0], self.ytre_grenser],
                  [-self.ytre_grenser, normal[0] * -self.ytre_grenser + normal[1]],
                  [(-self.ytre_grenser - normal[1]) / normal[0], -self.ytre_grenser]]
        check = [np.sqrt((points[0][0] - middlepoint[0]) ** 2 + (points[0][1] - middlepoint[1]) ** 2),
                 np.sqrt((points[1][0] - middlepoint[0]) ** 2 + (points[1][1] - middlepoint[1]) ** 2),
                 np.sqrt((points[2][0] - middlepoint[0]) ** 2 + (points[2][1] - middlepoint[1]) ** 2),
                 np.sqrt((points[3][0] - middlepoint[0]) ** 2 + (points[3][1] - middlepoint[1]) ** 2)]
        check, point = [list(t) for t in zip(*sorted(zip(check, points)))]
        point = point[0]
        return point

    def checker_ob0(self, normal):
        under = 0
        over = 1
        middlepoint_under = self.get_middle_point(under)
        middlepoint_over = self.get_middle_point(over)
        points_under = [[self.ytre_grenser, normal[under][0] * self.ytre_grenser + normal[under][1]],
                        [(self.ytre_grenser - normal[under][1]) / normal[under][0], self.ytre_grenser],
                        [-self.ytre_grenser, normal[under][0] * -self.ytre_grenser + normal[under][1]],
                        [(-self.ytre_grenser - normal[under][1]) / normal[under][0], -self.ytre_grenser]]
        points_over = [[self.ytre_grenser, normal[over][0] * self.ytre_grenser + normal[over][1]],
                       [(self.ytre_grenser - normal[over][1]) / normal[over][0], self.ytre_grenser],
                       [-self.ytre_grenser, normal[over][0] * -self.ytre_grenser + normal[over][1]],
                       [(-self.ytre_grenser - normal[over][1]) / normal[over][0], -self.ytre_grenser]]
        check_under = [np.sqrt(
            (points_under[0][0] - middlepoint_under[0]) ** 2 + (points_under[0][1] - middlepoint_under[1]) ** 2),
                       np.sqrt((points_under[1][0] - middlepoint_under[0]) ** 2 + (
                               points_under[1][1] - middlepoint_under[1]) ** 2),
                       np.sqrt((points_under[2][0] - middlepoint_under[0]) ** 2 + (
                               points_under[2][1] - middlepoint_under[1]) ** 2),
                       np.sqrt((points_under[3][0] - middlepoint_under[0]) ** 2 + (
                               points_under[3][1] - middlepoint_under[1]) ** 2)]
        check_over = [
            np.sqrt((points_over[0][0] - middlepoint_over[0]) ** 2 + (points_over[0][1] - middlepoint_over[1]) ** 2),
            np.sqrt((points_over[1][0] - middlepoint_over[0]) ** 2 + (points_over[1][1] - middlepoint_over[1]) ** 2),
            np.sqrt((points_over[2][0] - middlepoint_over[0]) ** 2 + (points_over[2][1] - middlepoint_over[1]) ** 2),
            np.sqrt((points_over[3][0] - middlepoint_over[0]) ** 2 + (points_over[3][1] - middlepoint_over[1]) ** 2)]
        check_under, points_under = [list(t) for t in zip(*sorted(zip(check_under, points_under)))]
        check_over, points_over = [list(t) for t in zip(*sorted(zip(check_over, points_over)))]
        p, q, r, s = [], [], [], []
        for i in range(len(points_under)):
            if abs(points_under[i][0]) <= self.ytre_grenser and abs(points_under[i][1]) <= self.ytre_grenser:
                p.append(points_under[i])
                q.append(check_under[i])
            if abs(points_over[i][0]) <= self.ytre_grenser and abs(points_over[i][1]) <= self.ytre_grenser:
                r.append(points_over[i])
                s.append(check_over[i])
        points_under, check_under, points_over, check_over = p, q, r, s
        if self.origo_is_between(middlepoint_under, middlepoint_over):
            point_under = points_under[0]
            point_over = points_over[0]
        else:
```

```
        if check_under[0] < check_over[0]:
            point_under = points_under[0]
            point_over = points_over[1]
        else:
            point_under = points_under[1]
            point_over = points_over[0]
    return point_under, point_over

def checker_ob_exl_innerb(self, normal):
    under = 0
    over = 1
    middlepoint_under = self.get_middle_point(under)
    middlepoint_over = self.get_middle_point(over)
    points_under = [self.ytre_grenser, normal[under][0] * self.ytre_grenser + normal[under][1]], [
        -self.ytre_grenser, normal[under][0] * -self.ytre_grenser + normal[under][1]]
    points_over = [self.ytre_grenser, normal[over][0] * self.ytre_grenser + normal[over][1]], \
                  [-self.ytre_grenser, normal[over][0] * -self.ytre_grenser + normal[over][1]]
    check_under = [np.sqrt(
        (points_under[0][0] - middlepoint_under[0]) ** 2 + (points_under[0][1] - middlepoint_under[1]) ** 2),
        np.sqrt((points_under[1][0] - middlepoint_under[0]) ** 2 + (
                points_under[1][1] - middlepoint_under[1]) ** 2)]
    check_over = [
        np.sqrt((points_over[0][0] - middlepoint_over[0]) ** 2 + (points_over[0][1] - middlepoint_over[1]) ** 2),
        np.sqrt((points_over[1][0] - middlepoint_over[0]) ** 2 + (points_over[1][1] - middlepoint_over[1]) ** 2)]
    check_under, points_under = [list(t) for t in zip(*sorted(zip(check_under, points_under)))]
    check_over, points_over = [list(t) for t in zip(*sorted(zip(check_over, points_over)))]
    if check_under[0] < check_over[0]:
        point_under = points_under[0]
        point_over = points_over[1]
    else:
        point_under = points_under[1]
        point_over = points_over[0]
    return point_under, point_over

def checker_ob_sympy(self):
    under = 0
    over = 1
    list_p_under, list_p_over = [], []
    seg_ob_bunn, seg_ob_topp, seg_ob_hoyr, seg_ob_vens = self.get_ob_segments_sympy()
    list_check = [seg_ob_bunn, seg_ob_topp, seg_ob_hoyr, seg_ob_vens]
    weak_seg_under, weak_seg_over = self.get_weakness_lines_sympy(under), self.get_weakness_lines_sympy(over)
    middlepoint_under, middlepoint_over = self.get_middle_point_sympy(under), self.get_middle_point_sympy(over)
    normal_seg_under = self.get_normal_line_sympy(middlepoint_under, weak_seg_under)
    normal_seg_over = self.get_normal_line_sympy(middlepoint_over, weak_seg_over)
    self.get_intersections_ob_sympy(list_check, normal_seg_under, normal_seg_over, list_p_under, list_p_over)
    point_under = list(self.get_nearest_intersection_under_sympy(list_p_under, middlepoint_under))
    point_over = list(self.get_nearest_intersection_over_sympy(list_p_over, middlepoint_over, middlepoint_under))
    return point_under, point_over

def get_intersections_ob_sympy(self, list_check, normal_seg_under, normal_seg_over, list_p_under, list_p_over):
    for check in list_check:
        point_under = self.get_normal_intersection_sympy(normal_seg_under, check)
        if point_under:
            list_p_under.append(point_under)
        point_over = self.get_normal_intersection_sympy(normal_seg_over, check)
        if point_over:
            list_p_over.append(point_over)
    if len(list_p_over) == 1:
        list_p_over.append([Point(-list_p_over[0][0][0],
                                  -list_p_over[0][0][1], evaluate=False)])
    if len(list_p_under) == 1:
        list_p_under.append([Point(-list_p_under[0][0][0],
                                   -list_p_under[0][0][1], evaluate=False)])
    return

def get_nearest_intersection_under_sympy(self, list_intersection_points, midpoint):
    if (self.forflytning_x_sone == 0 and self.forflytning_y_sone == 0) and \
            (self.vinkel == 45 or self.vinkel == 225 or self.vinkel == -135 or self.vinkel == -315):
        p1 = Point(-self.ytre_grenser, self.ytre_grenser)
        p2 = Point(self.ytre_grenser, -self.ytre_grenser)
        seg1 = Segment(p1, midpoint)
        seg2 = Segment(p2, midpoint)
        if seg1.length > seg2.length:
            point = p1
        else:
            point = p2
    elif (self.forflytning_x_sone == 0 and self.forflytning_y_sone == 0) and \
            (self.vinkel == -45 or self.vinkel == -225 or self.vinkel == 135 or self.vinkel == 315):
        p1 = Point(self.ytre_grenser, self.ytre_grenser)
        p2 = Point(-self.ytre_grenser, -self.ytre_grenser)
        seg1 = Segment(p1, midpoint)
        seg2 = Segment(p2, midpoint)
        if seg1.length > seg2.length:
            point = p2
        else:
            point = p1
    else:
        seg1 = Segment(list_intersection_points[0][0], midpoint)
        seg2 = Segment(list_intersection_points[1][0], midpoint)
        if seg1.length > seg2.length:
            point = list_intersection_points[1][0]
        else:
            point = list_intersection_points[0][0]
    return point

def get_nearest_intersection_over_sympy(self, list_intersection_points, midpoint_over, midpoint_under):
    if (self.forflytning_x_sone == 0 and self.forflytning_y_sone == 0) and \
            (self.vinkel == 45 or self.vinkel == 225 or self.vinkel == -135 or self.vinkel == -315):
        p1 = Point(-self.ytre_grenser, self.ytre_grenser)
        p2 = Point(self.ytre_grenser, -self.ytre_grenser)
        seg1 = Segment(p1, midpoint_over)
        seg2 = Segment(p2, midpoint_over)
        if seg1.length > seg2.length:
            point = p1
        else:
            point = p2
    elif (self.forflytning_x_sone == 0 and self.forflytning_y_sone == 0) and \
            (self.vinkel == -45 or self.vinkel == -225 or self.vinkel == 135 or self.vinkel == 315):
        p1 = Point(self.ytre_grenser, self.ytre_grenser)
        p2 = Point(-self.ytre_grenser, -self.ytre_grenser)
        seg1 = Segment(p1, midpoint_over)
        seg2 = Segment(p2, midpoint_over)
        if seg1.length > seg2.length:
            point = p2
        else:
            point = p1
    elif Segment(midpoint_under, midpoint_over).contains(Point(0, 0)):
        seg1 = Segment(list_intersection_points[0][0], midpoint_over)
        seg2 = Segment(list_intersection_points[1][0], midpoint_over)
        if seg1.length > seg2.length:
            point = list_intersection_points[1][0]
        else:
            point = list_intersection_points[0][0]
    else:
        seg1 = Segment(list_intersection_points[0][0], midpoint_over)
        seg2 = Segment(list_intersection_points[1][0], midpoint_over)
        if seg1.length > seg2.length:
            point = list_intersection_points[0][0]
        else:
            point = list_intersection_points[1][0]
    return point

def get_weakness_lines_sympy(self, element):
    indices_points_outer_boundary = self.get_indices_outer_boundary(element)
    p1 = Point(self.punkter_ytre[indices_points_outer_boundary[0]][0],
               self.punkter_ytre[indices_points_outer_boundary[0]][1], evaluate=False)
    p2 = Point(self.punkter_ytre[indices_points_outer_boundary[1]][0],
               self.punkter_ytre[indices_points_outer_boundary[1]][1], evaluate=False)
    weak_seg = Segment(p1, p2)
    return weak_seg

def get_ob_segments_sympy(self):
    pkt_ytre_4 = Point(self.ytre_grenser, -self.ytre_grenser)
    pkt_ytre_1 = Point(self.ytre_grenser, self.ytre_grenser)
    pkt_ytre_2 = Point(-self.ytre_grenser, self.ytre_grenser)
    pkt_ytre_3 = Point(-self.ytre_grenser, -self.ytre_grenser)
    seg_ob_bunn = Segment(pkt_ytre_4, pkt_ytre_3)
    seg_ob_topp = Segment(pkt_ytre_1, pkt_ytre_2)
    seg_ob_hoyr = Segment(pkt_ytre_4, pkt_ytre_1)
    seg_ob_vens = Segment(pkt_ytre_2, pkt_ytre_3)
    return seg_ob_bunn, seg_ob_topp, seg_ob_hoyr, seg_ob_vens

def get_middle_point_sympy(self, element):
    k = [3, 1]
    middlepoint = Point((self.punkter_ytre[element][0] + self.punkter_ytre[element + k[element]][0]) / 2,
                        (self.punkter_ytre[element][1] + self.punkter_ytre[element + k[element]][1]) / 2,
                        evaluate=False)
    return middlepoint

@staticmethod
def get_normal_line_sympy(midpoint, weak_seg):
    normal_line = weak_seg.perpendicular_line(midpoint)
    return normal_line

@staticmethod
def get_normal_intersection_sympy(normal_line, boundary_line_seg):
    intersection_point = normal_line.intersection(boundary_line_seg)
    return intersection_point

def checker_ib(self, normal, element):
    middlepoint = self.get_middle_point(element)
    middlepoints = self.get_middle_points()
    point_pos, point_neg = self.calculate_inner_points(normal[0], normal[1])
    points = [point_pos, point_neg]
    q = self.calculate_intersection_ib(point_pos, normal)
    p = self.calculate_intersection_ib(point_neg, normal)
    check = [np.sqrt((q[0] - middlepoint[0]) ** 2 + (q[1] - middlepoint[1]) ** 2),
             np.sqrt((p[0] - middlepoint[0]) ** 2 + (p[1] - middlepoint[1]) ** 2)]
    check, point = [list(t) for t in zip(*sorted(zip(check, points)))]
    if self.origo_is_between(middlepoints[0], middlepoints[1]):
        point_weakness = point[1]
        point_exl_weakness = point[0]
    else:
        point_weakness = point[0]
        point_exl_weakness = point[1]
    return point_weakness, point_exl_weakness

@staticmethod
def unrationalize_point(point):
    point = Point(point[0], point[1], evaluate=False)
    return point

def checker_ib_centered_sympy(self):
    under, over = 0, 1
    circle = Circle(Point(0.0, 0.0, evaluate=False), self.diameter / 2, evaluate=False)
    weak_seg_under, weak_seg_over = self.get_weakness_lines_sympy(under), self.get_weakness_lines_sympy(over)
    middlepoint_under, middlepoint_over = self.get_middle_point_sympy(under), self.get_middle_point_sympy(over)
    normal_seg_under = self.get_normal_line_sympy(middlepoint_under, weak_seg_under)
    normal_seg_over = self.get_normal_line_sympy(middlepoint_over, weak_seg_over)
    point_pos_under, point_neg_under = self.calculate_inner_points_sympy(normal_seg_under, circle)
    point_pos_over, point_neg_over = self.calculate_inner_points_sympy(normal_seg_over, circle)
    points_under = [point_pos_under, point_neg_under]
    points_over = [point_pos_over, point_neg_over]
    q = self.calculate_intersection_ib_sympy(points_under[0], normal_seg_under)[0]
    p = self.calculate_intersection_ib_sympy(points_under[1], normal_seg_under)[0]
    # r = self.calculate_intersection_ib_sympy(points_over[0], normal_seg_over)[0]
    # s = self.calculate_intersection_ib_sympy(points_over[1], normal_seg_over)[0]
    origo = Point(0, 0)
    point_check11 = Point(q[0], q[1])
    point_check12 = Point(p[0], p[1])
    # point_check21 = Point(r[0], r[1])
    # Point_check22 = Point(s[0], s[1])
    seg_middle = Segment(middlepoint_under, middlepoint_over)
    seg_check11 = Segment(origo, point_check11)
    seg_check12 = Segment(origo, point_check12)
    # seg_check21 = Segment(origo, point_check21)
    # seg_check22 = Segment(origo, Point_check22)

    # if self.origo_is_between(middlepoint_under, middlepoint_over):
    #     point_under = points_under[0]
    #     point_over = points_over[0]
    if seg_check11.contains(seg_middle) or seg_check12.contains(seg_middle):
        point_under = points_under[0]
        point_over = points_over[1]
    else:
        point_under = points_under[0]
        point_over = points_over[1]
    return point_under, point_over
```

```python
def checker_ib_sympy(self, element):
    circle = Circle(Point(0.0, 0.0, evaluate=False), self.diameter/2, evaluate=False)
    weak_seg = self.get_weakness_lines_sympy(element)
    middlepoint = self.get_middle_point_sympy(element)
    if element == 0:
        middlepoint_other = self.get_middle_point_sympy(1)
    else:
        middlepoint_other = self.get_middle_point_sympy(0)
    normal_seg = self.get_normal_line_sympy(middlepoint, weak_seg)
    point_pos, point_neg = self.calculate_inner_points_sympy(normal_seg, circle)
    points = [point_pos, point_neg]
    q = self.calculate_intersection_ib_sympy(points[0], normal_seg)[0]
    p = self.calculate_intersection_ib_sympy(points[1], normal_seg)[0]
    origo = Point(0, 0)
    point_check11 = Point(q[0], q[1])
    point_check12 = Point(p[0], p[1])
    seg_middle = Segment(middlepoint, middlepoint_other)
    seg_check11 = Segment(origo, point_check11)
    seg_check12 = Segment(origo, point_check12)
    # seg_check21 = Segment(origo, point_check21)
    # seg_check22 = Segment(origo, Point_check22)

    # if self.origo_is_between(middlepoint_under, middlepoint_over):
    #     point_under = points_under[0]
    #     point_over = points_over[0]
    if seg_check11.contains(seg_middle) or seg_check12.contains(seg_middle):
        point_weakness = points[0]
        point_exl_weakness = points[1]
    else:
        point_weakness = points[1]
        point_exl_weakness = points[0]
    return point_weakness, point_exl_weakness

def checker_ib_centered(self, normal):
    under, over = 0, 1
    middlepoints = self.get_middle_points_inner()
    middlepoint_under, middlepoint_over = middlepoints[0], middlepoints[1]
    point_pos_under, point_neg_under = self.calculate_inner_points(normal[under][0], normal[under][1])
    point_pos_over, point_neg_over = self.calculate_inner_points(normal[over][0], normal[over][1])
    points_under = [point_pos_under, point_neg_under]
    points_over = [point_pos_over, point_neg_over]
    q = self.calculate_intersection_ib(points_under[0], normal[under])
    origo = Point(0, 0)
    mid_under = Point(middlepoint_under[0], middlepoint_under[1])
    mid_over = Point(middlepoint_over[0], middlepoint_over[1])
    point_check11 = Point(q[0], q[1])
    seg_middle = Segment(mid_under, mid_over)
    seg_check11 = Segment(origo, point_check11)

    if seg_check11.contains(seg_middle):
        point_under = points_under[1]
        point_over = points_over[0]
    else:
        point_under = points_under[0]
        point_over = points_over[1]
    return point_under, point_over
```

```
def calculate_intersection_ib(self, punkt, normal):
    quad_index = self.get_quad_index(punkt)
    index_lowest_diff = self.get_index_lowest_diff_points(quad_index, punkt)
    a_circ = (self.nth_quad[quad_index][index_lowest_diff[1]][1] - self.nth_quad[quad_index][index_lowest_diff[0]][
        1]) / (
                 self.nth_quad[quad_index][index_lowest_diff[1]][0] -
                 self.nth_quad[quad_index][index_lowest_diff[0]][0])
    b_circ = self.nth_quad[quad_index][index_lowest_diff[1]][1] - a_circ * \
        self.nth_quad[quad_index][index_lowest_diff[1]][0]
    x = (b_circ - normal[1]) / (normal[0] - a_circ)
    y = (b_circ * normal[0] - normal[1] * a_circ) / (normal[0] - a_circ)
    point = [x, y]
    return point

def calculate_intersection_ib_sympy(self, punkt, normal_line):
    quad_index = self.get_quad_index(punkt)
    index_lowest_diff = self.get_index_lowest_diff_points(quad_index, punkt)
    p_circ0 = Point(self.nth_quad[quad_index][index_lowest_diff[1]][0],
                    self.nth_quad[quad_index][index_lowest_diff[1]][1])
    p_circ1 = Point(self.nth_quad[quad_index][index_lowest_diff[0]][0],
                    self.nth_quad[quad_index][index_lowest_diff[0]][1])
    seg_circ = Line(p_circ0, p_circ1)
    point = seg_circ.intersection(normal_line)
    return point

def __get_material_list2(self, ytre_punkt_under, ytre_punkt_over, indre_punkt_under, indre_punkt_over):
    list0 = [[self.punkter_ytre[0], self.punkter_ytre[3], ytre_punkt_under],
             [self.punkter_ytre[1], self.punkter_ytre[2], ytre_punkt_over],
             [self.punkter_indre[0], self.punkter_indre[3], indre_punkt_under],
             [self.punkter_indre[1], self.punkter_indre[2], indre_punkt_over],
             [self.punkter_ytre[3], self.punkter_ytre[2], self.punkter_indre[2]],
             [self.punkter_ytre[0], self.punkter_ytre[1], self.punkter_indre[1]],
             [self.punkter_indre[3], self.punkter_indre[2], self.punkter_indre[1]]]
    return list0

def __setmaterialmesh2_sympy(self):
    if self.vinkel == 0:
        ytre_punkt_under = [0, -self.ytre_grenser]
        ytre_punkt_over = [0, self.ytre_grenser]
        indre_punkt_under = [0, -5]
        indre_punkt_over = [0, 5]
    else:
        ytre_punkt_under, ytre_punkt_over = self.checker_ob_sympy()
        indre_punkt_under, indre_punkt_over = self.checker_ib_centered_sympy()
    list_iterate = self.__get_material_list2(ytre_punkt_under, ytre_punkt_over, indre_punkt_under,
                                             indre_punkt_over)
    list_iterate1 = self.list_which_material[2]
    i_material = self.index_materials
    for i in range(7):
        for j in range(3):
            self.data[i_material + j + 1] = re.sub(r'^(\s*(?:\S+\s+){1})\S+',
                                                   r'\g<1>' + str(list_iterate[i][j][0]) + ',',
                                                   self.data[i_material + j + 1])
            self.data[i_material + j + 1] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                                   r'\g<1>' + str(list_iterate[i][j][1]),
                                                   self.data[i_material + j + 1])
            self.data[i_material + 5] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                               r'\g<1>' + str(list_iterate1[i][0]),
                                               self.data[i_material + 5])
            self.data[i_material + 6] = re.sub(r'^(\s*(?:\S+\s+){2})\S+',
                                               r'\g<1>' + str(list_iterate1[i][1]),
                                               self.data[i_material + 6])
            i_material += 9
        return


def get_indices_periphery(points_tunnel_boundary, points_wb_3, points_wb_4):
    indices = []
    to_check = [points_wb_3, points_wb_4]
    for check in to_check:
        for point in check:
            idx = [i for i, e in enumerate(points_tunnel_boundary)
                   if e == [round(point[0], len(str(e[0]).replace('-', '')) - 2),
                            round(point[1], len(str(e[1]).replace('-', '')) - 2)]]
            if not idx:
                continue
            indices.append(idx[0])
    indices.sort()
    return indices
```

```
from subprocess import Popen
from time import sleep

import pandas as pd
import pyautogui as pag

from Automatisering_RS2 import module_main_functions as mmf

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)

"""
create mesh opens rs2 of each file and creates the mesh, defined by the settings given in the template files,
succesively.
"""


def create_mesh(mappenavn_til_rs2, mappenavn_til_csv, df_stier_rs2filer, df_stier_csvfiler, path_rs2, time,
                files_to_skip, bool_shall_execute_create_mesh, storage_calculation_times):
    """

    @param mappenavn_til_rs2:
    @param mappenavn_til_csv:
    @param df_stier_rs2filer:
    @param df_stier_csvfiler:
    @param path_rs2:
    @param time:
    @param files_to_skip:
    @param bool_shall_execute_create_mesh:
    @param storage_calculation_times:
    @return:
    """

    if bool_shall_execute_create_mesh is False:
        return

    time_operation = time.time()
    category = 'mesh'

    mmf.procede_script()
    for k, (navn_rs2, navn_csv) in enumerate(zip(mappenavn_til_rs2, mappenavn_til_csv)):
        if k in files_to_skip:
            continue
        for j in range(df_stier_rs2filer.shape[0]):
            path_fil_rs2 = df_stier_rs2filer[navn_rs2][j]
            path_fil_csv = df_stier_csvfiler[navn_csv][j]
            if isinstance(path_fil_rs2, str) and isinstance(path_fil_csv, str):
                Popen([path_rs2, path_fil_rs2])
                sleep(5)

                # chooses the rs2 window so hotkeys can be used
                pag.leftClick(927, 490, interval=time[1])
                # create discretization and mesh
                pag.hotkey('ctrl', 'm', interval=time[1])
                # save
                pag.hotkey('ctrl', 's', interval=time[1])
        # close window
        pag.hotkey('alt', 'f4', interval=time[1])

mmf.calculate_computation_time(time_operation, category, storage_calculation_times)
return
```

```
import re
from subprocess import Popen
from time import sleep

import pandas as pd
import psutil
import pyautogui as pag

from Automatisering_RS2 import module_main_functions as mmf

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)

"""
get_files_unsuc_tolerance iterates over the log files created after each calculated fea-file, and stores the filenames
of log-files in where the tolerance is not below the limit set by the user. These filenames is stored in a csv file
set by the user. It is used by the function ea.calculate in experiment_actions line 148.
"""


def get_files_unsuc_tolerance(path_arbeidsfiler, list_navn_modell, path_store_unsuc_tol_models, tolerance):
    path_arbeidsfiler = mmf.alternate_slash([path_arbeidsfiler])[0]
    file_suffix = '.log'
    list_unsucsesful_tolerance = []
    subs_tol = 'Tolerance:'
    for navn in list_navn_modell:
        navn = navn.replace('.fea', '')
        check_path = path_arbeidsfiler + '/' + navn + file_suffix
        with open(check_path, 'r') as file:
            data = file.readlines()
        list_tol = [line for line in data if subs_tol in line]
        plist = []
        for line in list_tol:
            p = re.findall(r"([-+]?(\d*\.\d+[Ee]?[+-]?\d*\b(?!:))|[-+]?(\d*[Ee]?[+-]?\d*\b(?!:)))", line)[1]
            plist.append(p)
        list_tol = plist.copy()
        list_tol = [float(tol) for tol in list_tol if float(tol) > tolerance]
        if list_tol:
            list_unsucsesful_tolerance.append(check_path)

    with open(path_store_unsuc_tol_models, 'w') as file:
        file.writelines([f"{var1}\n" for var1 in list_unsucsesful_tolerance])
    return


"""
The three next functions is used to automize the calculation process. When the calculations is over, the main file
is allowed to continue with the next line.
"""

"""RS2 Calculate, with filename 'feawin.exe', used to get its process data."""

"""
This function opens RS2 Compute, opens the files to be calculated in RS2 Compute, executes calculations, and lastly,
tracks the cpu usage; when below 5 percent it lets the main script continue whit the next line
"""
```

```
def automation_actions_calculation(number_of_files, sti_til_mappe_for_arbeidsfiler, coordinate_rs2_compute,
                                   name_col_df_mouse, time=None, i=0):
    # if there is not defined a vector of time increments it is set to be time_list which is allocated on top of this
    # file.
    if time is None:
        time = mmf.get_time_increments()

    # sends the path of the workfolder to clipboard
    mmf.copy2clip(sti_til_mappe_for_arbeidsfiler)

    # For some reason rs2 can only add roughly 600 files at a time, thus since the thesis has 924 models for one
    # thickness, the process must be done twice. If the models for one experiment exceeds 1200 this function must
    # be reedited.
    num_first_batch = round(number_of_files / 2)
    num_second_batch = number_of_files - num_first_batch + 1

    # opening the workfolder
    pag.hotkey('alt', 'd', interval=time[2])
    pag.hotkey('ctrl', 'v', interval=time[2])
    # the first opening of files begin
    sleep(5)
    pag.press('enter', interval=time[1])
    sleep(5)
    pag.press('tab', presses=4, interval=time[2])
    pag.keyDown('shiftleft')
    pag.keyDown('shiftright')
    pag.press('down', presses=num_first_batch)
    pag.keyUp('shiftleft')
    pag.keyUp('shiftright')
    sleep(5)
    pag.press('enter', interval=time[1])
    # the first openings of files end
    sleep(60)
    # second openings of files begin
    pag.press('enter', interval=time[3])
    pag.keyDown('shiftleft')
    pag.keyDown('shiftright')
    pag.press('tab', presses=2, interval=time[1])
    pag.keyUp('shiftleft')
    pag.keyUp('shiftright')
    pag.press('down', presses=num_first_batch)
    pag.keyDown('shiftleft')
    pag.keyDown('shiftright')
    pag.press('down', presses=num_second_batch)
    pag.keyUp('shiftleft')
    pag.keyUp('shiftright')
    # second openings of files end
    sleep(15)
    pag.press('enter', interval=time[1])
    # kjÃ¸re kalkulasjon
    sleep(60)
    pag.press('tab', presses=2, interval=time[2])
    pag.press('space', interval=time[2])

    # if there is experienced trouble whith the automated calculation setup, uncomment below, and do it the hardway!
```

```python
    # or solve the bug, up to you :))))
    # pause_script()

    # the last part checks the cpu-percentage used to the calculation operations. When it is below a limit of 5 percent
    # the while loop is exited and the script continues with the next step. Also, if there are problems with the file
    # fetching described above, this section can be commented out.
    procname = 'feawin.exe'  # feawin is the filename of RS2 Compute
    bool_proc = mmf.check_if_process_running(procname)
    sleep(5)
    while bool_proc:
        pid = mmf.get_pid(procname)
        process_compute = psutil.Process(pid)
        # this makes sure that the screen do not go to powersave mode for PCs lacking admin controll or lacking
        # controll over the functionality of the lock and sleep beaviour.
        pag.click(coordinate_rs2_compute[name_col_df_mouse[1]][i],
                  coordinate_rs2_compute[name_col_df_mouse[2]][i], interval=time[0])
        if process_compute.cpu_percent(interval=15) > 5.0:
            bool_proc = True
        else:
            bool_proc = False
    return


"""
calculate calls the Auto.automation_actions_calculation
when calculation finnished it calls get_files_unsuc_tolerance to get files that is exceeding allowed tolerance.
"""


def calculate(path_rs2_compute, time, df_filnavn_rs2, sti_til_mappe_for_arbeidsfiler, path_store_unsuc_tol_models,
              tolerance, number_of_files, bool_shall_execute_calculate, coordinate_rs2_compute, name_col_df_mouse,
              bool_stop_to_check_logs, storage_calculation_times):
    """

    @param path_rs2_compute:
    @param time:
    @param df_filnavn_rs2:
    @param sti_til_mappe_for_arbeidsfiler:
    @param path_store_unsuc_tol_models:
    @param tolerance:
    @param number_of_files:
    @param bool_shall_execute_calculate:
    @param coordinate_rs2_compute:
    @param name_col_df_mouse:
    @param bool_stop_to_check_logs:
    @param storage_calculation_times:
    @return:
    """
    if bool_shall_execute_calculate is False:
        return
    time_operation = time.time()
    category = 'calculation'
    Popen([path_rs2_compute])
    sleep(5)
    automation_actions_calculation(number_of_files, sti_til_mappe_for_arbeidsfiler, coordinate_rs2_compute,
                                   name_col_df_mouse)
    # lukke RS2 Compute
```

```
pag.hotkey('alt', 'f4', interval=time[2])
get_files_unsuc_tolerance(sti_til_mappe_for_arbeidsfiler, df_filnavn_rs2, path_store_unsuc_tol_models, tolerance)
mmf.calculate_computation_time(time_operation, category, storage_calculation_times)
if bool_stop_to_check_logs is True:
    mmf.procede_script()
return
```

```python
from subprocess import Popen
from time import sleep

import pandas as pd
import pyautogui as pag

from Automatisering_RS2 import module_main_functions as mmf

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)

"""
The four functions beneath is used to store the excavation query line values fetched from a specific file
"""
"""
This function should be used the first time the storing process is initiated.

It sets the stage to the excavation stage.
"""



def store_results_csv_prep_init(df_koordinates_mouse, name_col_df, i=1, time=None):
    if time is None:
        time = mmf.get_time_increments()
    pag.click(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
              interval=time[1])  # velg excavation stage
    i += 1
    pag.hotkey('f6', interval=time[1])
    # pag.leftClick(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
    #               interval=time[1])  # change type
    i += 1
    # pag.leftClick(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
    #               interval=time[1])  # choose tot sigma
    i += 1
    # pag.rightClick(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
    #          interval=time[1])  # choose sig1
    i += 1
    pag.hotkey('ctrl', 'e', interval=time[1])  # generates excavation query line
    return i


"""
Same as function over, but:

This function should be used if there happened some complications during the calculation process and the operation
must be repeated. This function sets the interpreter into showing stress which is the first parameter to be stored. If
this was not done, the interpreter is saved to show total deformation.
"""



def store_results_csv_prep(df_koordinates_mouse, name_col_df, i=1, time=None):
    if time is None:
        time = mmf.get_time_increments()
    pag.click(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
              interval=time[1])  # choose stage 2
    i += 1
    pag.hotkey('f6', interval=time[1])
    pag.leftClick(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
                  interval=time[1])  # change type
    i += 1
    pag.leftClick(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
                  interval=time[1])  # choose tot sigma
    i += 1
    pag.rightClick(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
                   interval=time[1])  # choose sig1
    i += 1
    pag.hotkey('ctrl', 'e', interval=time[1])  # generates excavation query line
    return i


"""
saves the content of excavation line to clipboard, later to be saved in csv-file
"""


def interpret_resultat_til_clipboard(time=None):
    if time is None:
        time = mmf.get_time_increments()
    pag.press('f6', interval=time[2])
    pag.click(754, 527, interval=time[1], button='right')
    pag.click(846, 666, interval=time[1])
    return


"""
the function stores the content given in a excavation query of a model in a specific csv file
"""


def store_results_in_csv(df_koordinates_mouse, name_col_df, path_fil_csv, navn_parameter, i=1, time=None):
    if time is None:
        time = mmf.get_time_increments()
    if i == 4:
        sr = pd.DataFrame([navn_parameter])
        sr.to_csv(path_or_buf=path_fil_csv, mode='w', sep=';', header=False, index=False)
        pag.rightClick(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
                       interval=time[1])  # choose boundary 1st time, right click
        i += 1
        pag.click(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
                  interval=time[1])  # velg copy data
        i += 1
        data = pd.read_clipboard()  # fetch content stored in clipboard, see full description in
        # Logg - Mastergradsoppgave_Modellering, 10.08.2021
        data.to_csv(path_or_buf=path_fil_csv, mode='a', sep=';', header=False, index=True)  # data stored in a given csv
    else:
        sr = pd.DataFrame([navn_parameter])
        sr.to_csv(path_or_buf=path_fil_csv, mode='a', sep=';', header=False, index=False)
        pag.click(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
                  interval=time[1], button='left')  # change parameter
        i += 1
        pag.click(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
                  interval=time[1], button='left')  # choose deformation
        i += 1
        pag.click(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
                  interval=time[1], button='left')  # choose total deformation
        i += 1
        pag.click(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
                  interval=time[1], button='right')  # choose boundary 2nd time, rightclick
        i += 1
        pag.click(df_koordinates_mouse[name_col_df[1]][i], df_koordinates_mouse[name_col_df[2]][i],
                  interval=time[1], button='left')  # choose copy data
        i += 1
        data = pd.read_clipboard()  # fetch content stored in clipboard, see full description in
        # Logg - Mastergradsoppgave_Modellering, 10.08.2021
        data.to_csv(path_or_buf=path_fil_csv, mode='a', sep=';', header=None, index=True)  # data stored in a given csv
    return i


"""
store_data sequences through each model and to store each query line
"""


def store_data(mappenavn_til_rs2, mappenavn_til_csv, df_stier_rs2filer, df_stier_csvfiler, path_rs2_interpret,
               df_koordinater_mus, navn_kol_df_koord_mus, ant_parametere_interpret, parameter_navn_interpret, time,
               ll_inner_points, bool_is_first_time_execute_data_store, bool_shall_execute_storedata, files_to_skip,
               storage_calculation_times):
    """

    @param mappenavn_til_rs2:
    @param mappenavn_til_csv:
    @param df_stier_rs2filer:
    @param df_stier_csvfiler:
    @param path_rs2_interpret:
    @param df_koordinater_mus:
    @param navn_kol_df_koord_mus:
    @param ant_parametere_interpret:
    @param parameter_navn_interpret:
    @param time:
    @param ll_inner_points:
    @param bool_is_first_time_execute_data_store:
    @param bool_shall_execute_storedata:
    @param files_to_skip:
    @param storage_calculation_times:
    @return:
    """
    if bool_shall_execute_storedata is False:
        return
    time_operation = time.time()
    category = 'store_data'

    i = 1
    for k, (navn_rs2, navn_csv, (colname_innerpoints, l_inner_points)) in enumerate(zip(
            mappenavn_til_rs2, mappenavn_til_csv, ll_inner_points.iteritems())):
        if k in files_to_skip:
            continue
        for j, innerpoints in enumerate(l_inner_points):
            path_fil_rs2 = df_stier_rs2filer[navn_rs2][j]
            path_fil_csv = df_stier_csvfiler[navn_csv][j]
            if isinstance(path_fil_rs2, str) and isinstance(path_fil_csv, str):
```

```
            Popen([path_rs2_interpret, path_fil_rs2])
            sleep(5)
            # if there is a window open in addition to rs2 interpret, this removes it
            pag.press('tab', interval=time[1])
            pag.press('enter', interval=time[2])
            if bool_is_first_time_execute_data_store is True:
                i = store_results_csv_prep_init(df_koordinater_mus, navn_kol_df_koord_mus, i)
            else:
                i = store_results_csv_prep(df_koordinater_mus, navn_kol_df_koord_mus, i)
            for q in range(ant_parametere_interpret):
                navn_parameter = parameter_navn_interpret[q]
                i = store_results_in_csv(df_koordinater_mus, navn_kol_df_koord_mus, path_fil_csv,
                                         navn_parameter, i)
            # markere slutten pĂĽ fila
            sr = pd.DataFrame(['end'])
            sr.to_csv(path_or_buf=path_fil_csv, mode='a', sep=';', header=False, index=False)
            # lukke interpret
            pag.hotkey('ctrl', 's', interval=time[1])
            pag.hotkey('alt', 'f4', interval=time[1])
            i = 1
    mmf.calculate_computation_time(time_operation, category, storage_calculation_times)
    return
```

```
import os
import re
import time

# import numpy as np
import pandas as pd

from Automatisering_RS2 import module_main_functions as mmf

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)


def get_values_quad(to_plot, points, query_positions):
    if not (to_plot is not None and points is not None):
        return None
    value_intersect, indices_max_value = [], []
    points = [[round(point[0], 3), round(point[1], 3)] for point in points]
    for k, (data_sep, q_pos_sep) in enumerate(zip(reversed(to_plot), reversed(query_positions))):
        values, indices = [], []
        for point in points:
            for i, (value, q_pos) in enumerate(zip(data_sep, q_pos_sep)):
                if q_pos == point:
                    values.append(value[1])
                    indices.append(i)
        idx_a = indices[0]
        idx_b = indices[1]
        idx_c = indices[2]
        idx_d = indices[3]
        if k == 0:
            max_value_h, max_val_with_arc_h = get_max_totdef(idx_a, idx_b, data_sep)
            max_value_l, max_val_with_arc_l = get_max_totdef(idx_c, idx_d, data_sep)
            index_max_value_h = data_sep.index(max_val_with_arc_h)
            index_max_value_l = data_sep.index(max_val_with_arc_l)
            list_append = [values[0], values[1], max_value_h, values[3], values[2], max_value_l]
            for a in list_append:
                value_intersect.append(a)
            indices_max_value.append(index_max_value_h),
indices_max_value.append(index_max_value_l)
        else:
            max_value_h = data_sep[indices_max_value[0]][1]
            max_value_l = data_sep[indices_max_value[1]][1]
            list_append = [values[0], values[1], max_value_h, values[3], values[2], max_value_l]
            for a in list_append:
                value_intersect.append(a)
    return value_intersect


def get_max_totdef(idx_a, idx_b, data_sep):
    if idx_a < idx_b:
        indices = [i for i in range(idx_a, idx_b + 1, 1)]
    else:
        l1 = [i for i in range(0, idx_b + 1, 1)]
        l2 = [i for i in range(idx_a, len(data_sep), 1)]
        indices = l1 + l2
    data = [data_sep[idx] for idx in indices]
    data_exl_arc = [data_sep[idx][1] for idx in indices]
    max_def = max(data_exl_arc)
    idx = data_exl_arc.index(max_def)
    max_def_with_arc = data[idx]
    return max_def, max_def_with_arc


def make_container_diff(mappenavn_rs2):
    container = []
    for i in range(len(mappenavn_rs2)):
        container.append([])
    return container


def get_parameter_values(navn_allines, params_varied):
    regex_shale = r'(?<=_{})\d*\.*\d*'
    param_values = []
    for param in params_varied:
        reg_pat = regex_shale.format(param)
        param_value = float(re.findall(reg_pat, navn_allines)[0])
        param_values.append(param_value)
    return param_values


def get_corrupted_file_paths(file_paths, elements_corrupted_files):
    if elements_corrupted_files is None:
        corrupted_paths = ['']
    else:
        corrupted_paths = [path for i, path in enumerate(file_paths) if
elements_corrupted_files.count(i) > 0]
    return corrupted_paths


def get_paths_zone2lines(file_paths, elements_corrupted_files, twolines_inside):
    corrupted_paths = get_corrupted_file_paths(file_paths, elements_corrupted_files)
    paths = [iteration for iteration in twolines_inside if iteration[0] not in corrupted_paths]
    return paths


def get_paths_without_corrupted(file_paths, elements_corrupted_files):
    corrupted_paths = get_corrupted_file_paths(file_paths, elements_corrupted_files)
    paths = [iteration for iteration in file_paths if iteration not in corrupted_paths]
    return paths


 """

"""


def create_csv_max_values(foldername_csv, list_values, parameternavn_interpret, paths_fil_csv,
                          path_data_storage, elements_corrupted_files, val_navn,
                          sti_values_toplot, parameters_varied, list_true_lengths):
    if all(path is None for path in paths_fil_csv):
        return None, None
    t = path_data_storage
    t = mmf.alternate_slash([t])[0]
    p = parameternavn_interpret.copy()
    p.pop()
    # fjerner de paths som er korruperte
    paths = get_paths_without_corrupted(paths_fil_csv, elements_corrupted_files)
    list_navn_allines = [name.replace('.csv', '') for name in paths]
    list_varied_param_values = [get_parameter_values(navn, parameters_varied) for navn in
list_navn_allines]
    # lager paths til der hvor verdiene skal lagres
    path_all = t + '/' + foldername_csv + 'max_values.csv'
    list_to_df = []
    for navn, values, varied_param_values, true_len in zip(list_navn_allines, list_values,
list_varied_param_values,
                                                          list_true_lengths):
        list_to_df.append([navn] + [true_len] + varied_param_values + values)
    df_values = pd.DataFrame(list_to_df, columns=val_navn)
    df_values.to_csv(path_or_buf=path_all, sep=';', mode='w', index=False)
    df_values.to_csv(path_or_buf=sti_values_toplot, sep=';', mode='a', index=False)
    return


"""
these functions
"""


def get_size_file(filepath):
    size = os.path.getsize(filepath) / 1000  # gives answear in kilobytes
    return size


def get_list_size_files(filepaths):
    file_sizes = []
    for path in filepaths:
        size = get_size_file(path)
        file_sizes.append(size)
    return file_sizes


def get_average(list0):
    return sum(list0) / len(list0)


def get_elements_small_files(file_sizes):
    mean = get_average(file_sizes)
    elements = [i for i, file_size in enumerate(file_sizes) if file_size < (mean - mean / 4)]
    return elements


def get_elements_corrupted_files(file_paths):
    if all(path is None for path in file_paths):
        return [None for _ in file_paths]
    file_sizes = get_list_size_files(file_paths)
    elements = get_elements_small_files(file_sizes)
    if not elements:
        elements = [None for _ in file_paths]
    return elements
```

```python
"""
get_elements_corrupted_files_2lines
"""


def get_elements_corrupted_files_2lines(file_paths):
    if all(path is None for path in file_paths):
        return None
    file_sizes = get_list_size_files(file_paths)
    elements = get_elements_small_files(file_sizes)
    if not elements:
        elements = None
    return elements


"""
skewness
"""


# def calculate_skewness(_listoflists_values, tolerance, _listoflists_archlength):
#    totdef_list = _listoflists_values[1]
#    arclength_list = _listoflists_archlength[0]
#    list_index = [index for val, index in enumerate(totdef_list)]
#    list_index = list_index[90::] + list_index[0:90]
#    list_bool = []
#    indices_momentum = []
#    # find indices to define ranges for momentum calculation
#    for i in list_index:
#        if i == list_index[-1]:
#            delta = np.abs(totdef_list[i] - totdef_list[0])
#        else:
#            delta = np.abs(totdef_list[i] - totdef_list[i + 1])
#        if delta > tolerance:
#            _bool = True
#        else:
#            _bool = False
#        list_bool.append(_bool)
#    for i in range(len(list_index)):
#        if i == len(list_index) - 1:
#            if (list_bool[0] is True and list_bool[i] is False) or (list_bool[0] is False and list_bool[i] is
True):
#                indices_momentum.append(list_index[i])
#        else:
#            if (list_bool[i] is True and list_bool[i + 1] is False) or (
#                    list_bool[i] is False and list_bool[i + 1] is True):
#                indices_momentum.append(list_index[i])
#    # defining the ranges, both counterclockwise and clockwise for both intersections
#    range_top_med_klokka = [indices_momentum[0], indices_momentum[1]]
#    range_top_mot_klokka = [indices_momentum[1], indices_momentum[0]]
#    range_bot_med_klokka = [indices_momentum[2], indices_momentum[3]]
#    range_bot_mot_klokka = [indices_momentum[3], indices_momentum[2]]
#
#    totdef_top_cclockwise = totdef_list[range_top_mot_klokka[0]:range_top_mot_klokka[1]:1]
#    totdef_top_clockwise = totdef_list[range_top_med_klokka[0]:range_top_med_klokka[1]:-1]
#    totdef_bot_cclockwise = totdef_list[range_bot_mot_klokka[0]:range_bot_mot_klokka[1]:1]
#    totdef_bot_clockwise = totdef_list[range_bot_med_klokka[0]:range_bot_med_klokka[1]:-1]
#    iter_totdef = [totdef_top_cclockwise, totdef_top_clockwise, totdef_bot_cclockwise,
totdef_bot_clockwise]
#    momentum = []
#    for list_def, ist_arc in zip(iter_totdef, iter_arc):
#
#    # \n mĂĽ legges til hver rad der hver rad lagres som streng
#    return momentum


def numerical_momentum_per_meter(list_def, list_arc, radius, youngs_modulus):
    momentum = 0
    for i in range(len(list_def) - 1):
        strain0, strain1 = list_def[i] / radius, list_def[i + 1] / radius
        arc0, arc1 = list_arc[i], list_arc[i + 1]
        momentum = momentum + calculate_momentum_i_per_meter(youngs_modulus, strain0,
strain1, arc0, arc1)
    return


def calculate_momentum_i_per_meter(youngs_modulus, strain0, strain1, arc0, arc1):
    delta_strain = strain1 - strain0
    delta_arc = arc1 - arc0
    delta_arc2 = arc1 ** 2 - arc0 ** 2
    delta_arc3 = arc1 ** 3 - arc0 ** 3

    momentum_i_per_meter = (youngs_modulus / 2) * ((delta_arc2 - arc0 * delta_arc) * strain0 +
                                              (1 / 12) * ((2 * delta_arc3 - 3 * arc0 * delta_arc2) /
                                                    delta_arc) * delta_strain)
    return momentum_i_per_meter


def prep_parameter_navn(parameter_navn):
    p = []
    for navn in parameter_navn:
        navn = navn + '\n'
        p.append(navn)
    return p


def prep_strings_to_float(data):
    prepped_data = []
    for index, points in enumerate(data):
        if index < 2:
            continue
        points_string = re.findall(r"[-+]?(?:\d*\.\d+|\d+\b(?!:))", points)
        points = [float(point) for point in points_string]
        prepped_data.append(points)
    return prepped_data


def get_query_values(path_to_csv, parameternavn):
    if path_to_csv is None:
        data_split = None
        return data_split
    with open(path_to_csv, 'r') as file:
        data = file.readlines()
    data_split = []
    for i in range(len(parameternavn) - 1):
        index_start = data.index(parameternavn[i])
        index_slutt = data.index(parameternavn[i + 1])
        p = data[index_start:index_slutt].copy()
        p = [_list for _list in p if 'UNDEFINED' not in _list]
        p = prep_strings_to_float(p)
        data_split.append(p)
    return data_split


def get_query_positions(query_values):
    positions = []
    for i, query in enumerate(query_values):
        positions.append([])
        for value in query:
            positions[i].append(value[3:5])
    return positions


def get_values_to_plot(query_values):
    arclengths = []
    values = []
    for i, query in enumerate(query_values):
        arclengths.append([])
        values.append([])
        for value in query:
            arclengths[i].append(value[5])
            values[i].append(value[6])
    return arclengths, values


def get_values_to_plot_arc_and_val(query_values):
    to_plot = []
    for i, query in enumerate(query_values):
        to_plot.append([])
        for value in query:
            to_plot[i].append(value[5:])
    return to_plot


"""

"""


def execute_data_processing(parameter_navn_interpret, mappenavn_til_rs2, mappenavn_til_csv,
                            df_stier_csvfiler, list_points_to_check, sti_til_mapper_endelige_filer,
                            list_excluded_files_2linescalc, list_valnavn, sti_values_toplot,
                            list_0lines_inside, list_1line_inside, parameters_varied, true_lengths,
                            bool_shall_execute_data_processing, files_to_skip,
                            storage_calculation_times):
    """

    @param parameter_navn_interpret:
    @param mappenavn_til_rs2:
```

```
    @param mappenavn_til_csv:
    @param df_stier_csvfiler:
    @param list_points_to_check:
    @param sti_til_mapper_endelige_filer:
    @param list_excluded_files_2linescalc:
    @param list_valnavn:
    @param sti_values_toplot:
    @param list_0lines_inside:
    @param list_1line_inside:
    @param parameters_varied:
    @param true_lengths:
    @param bool_shall_execute_data_processing:
    @param files_to_skip:
    @param storage_calculation_times:
    @return:
    """
    if bool_shall_execute_data_processing is False:
        return
    time_operation = time.time()
    category = 'data_processing'

    list_values = make_container_diff(mappenavn_til_rs2)

    # list_momentum_values = make_container_diff(mappenavn_til_rs2)
    parameter_navn_interpret0 = prep_parameter_navn(parameter_navn_interpret)

    for k, (navn_rs2, navn_csv, excluded_files,
            (points_check_colname, points_to_check), valnavn, (zerolines_colname, _0lines_inside),
            (oneline_colname, _1line_inside)) in \
            enumerate(zip(mappenavn_til_rs2, mappenavn_til_csv, list_excluded_files_2linescalc,
                          list_points_to_check.iteritems(),
                          list_valnavn, list_0lines_inside.iteritems(), list_1line_inside.iteritems())):
        if k in files_to_skip:
            continue
        # element_files_corrupted er definert mtp en treshold for filstĂ¸rrelse, da de filer hvor
feilklikking inntreffer
        # viser ca 1/3 mindre stĂ¸rrelse.
        elements_files_corrupted = get_elements_corrupted_files(df_stier_csvfiler[navn_csv])
        idx0, idx1, idx2 = 0, 0, 0
        true_lengths_copy = true_lengths.copy()
        list_true_lengths = []
        for z, (csv_sti, corrupted) in enumerate(zip(df_stier_csvfiler[navn_csv],
elements_files_corrupted)):
            if true_lengths_copy:
                true_len = true_lengths_copy.pop(0)
            else:
                true_lengths_copy = true_lengths.copy()
                true_len = true_lengths_copy.pop(0)
            if corrupted is not None:
                continue
            list_true_lengths.append(true_len)
            query_values = get_query_values(csv_sti, parameter_navn_interpret0)
            # her kan jeg legge inn for mentberegninger
            arclengths_to_plot, values_to_plot = get_values_to_plot(query_values)
            to_plot = get_values_to_plot_arc_and_val(query_values)
            query_positions = get_query_positions(query_values)
            if csv_sti == _0lines_inside[idx0][1]:
                max_val_def = max(values_to_plot[1])
                idx_max_val_def = values_to_plot[1].index(max_val_def)
                # arclen_max_val_def = arclengths_to_plot[1][idx_max_val_def]
                sig1_max_val_def = values_to_plot[0][idx_max_val_def]
                values = [max_val_def, sig1_max_val_def, None, None, None, None]
                list_values[k].append(values)
                _0lines_inside.pop(idx0)
                idx0 += 1
            elif csv_sti == _1line_inside[idx1][1]:
                max_val_def = max(values_to_plot[1])
                idx_max_val_def = values_to_plot[1].index(max_val_def)
                # arclen_max_val_def = arclengths_to_plot[1][idx_max_val_def]
                sig1_max_val_def = values_to_plot[0][idx_max_val_def]
                values = [max_val_def, sig1_max_val_def, None, None, None, None]
                list_values[k].append(values)
                _1line_inside.pop(idx1)
                idx1 += 1
            else:
                points = points_to_check.pop(idx2)
                idx2 += 1
                values_to_plot_2lines = get_values_quad(to_plot, points, query_positions)
                values = [None, None, values_to_plot_2lines[2], values_to_plot_2lines[5],
values_to_plot_2lines[8],
                        values_to_plot_2lines[11]]
                if all(value is not None for value in values_to_plot_2lines):
                    list_values[k].append(values)
        paths_fil_csv = df_stier_csvfiler[navn_csv]
        create_csv_max_values(navn_csv, list_values[k], parameter_navn_interpret, paths_fil_csv,
                          sti_til_mapper_endelige_filer,
                          elements_files_corrupted, valnavn,
                          sti_values_toplot, parameters_varied, list_true_lengths)

        mmf.calculate_computation_time(time_operation, category, storage_calculation_times)
    return
```

Eirik Kaasbøll Andresen

Sensitivity analysis using Finitie Element Method

NTNU
Norwegian University of
Science and Technology