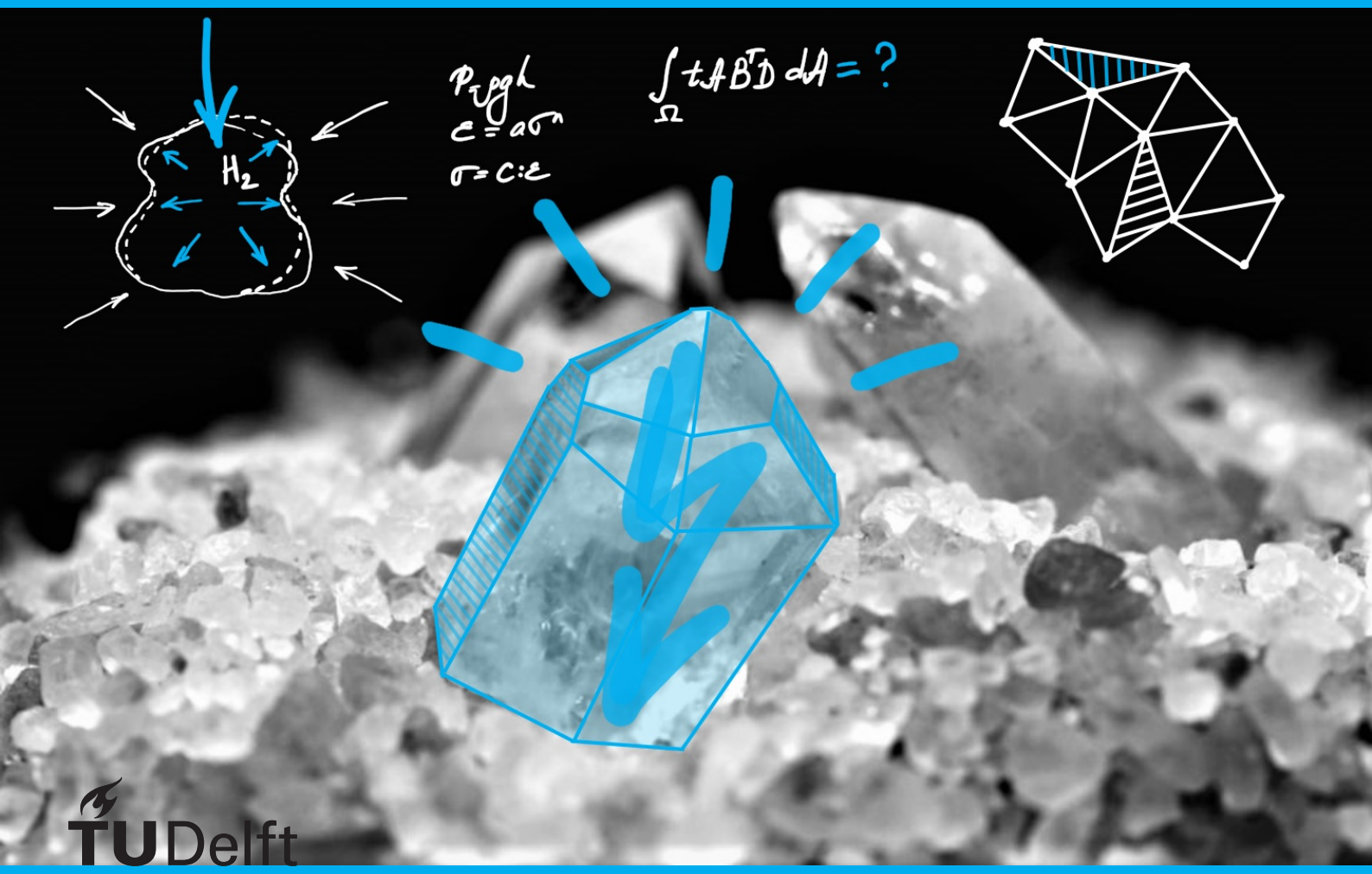
Nonlinear 2D Finite Element Modeling:

Cyclic Energy Storage in Salt Caverns with Creep Deformation Physics

Artur A. Makhmutov
Student ID: 4846311

Faculty of Civil Engineering & Geosciences, TU Delft
12 August 2020



TU Delft

Nonlinear 2D Finite Element Modeling:

Cyclic Energy Storage in Salt Caverns with Creep Deformation Physics

by

Artur A. Makhmutov

Student ID: 4846311

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday August 12, 2020 at 13:00 PM.

Student number:	4846311	
Project duration:	November 1, 2019 – July 30, 2020	
Advisor:	Associate Prof. Hadi Hajibeygi,	CITG, TU Delft
co-supervisor:	Ir. Kishan Ramesh Kumar,	CITG, TU Delft
Other committee members:	Prof. Giovanni Bertotti,	CITG, TU Delft
	Prof. Paulien Herder,	TNW, TU Delft
	Associate Prof. Karl-Heinz Wolf,	CITG, TU Delft

TU Delft

Preface

This MSc thesis is a part of ADMIRE project, which is funded by NWO-TTW ViDi grant of my supervisor Dr. Hadi Hajibeygi, who provided me with a great support throughout the entire research. It was a great honor for me to work with such an open minded and inspiring person. The knowledge and wisdom which I had an opportunity to absorb during our collaboration together extends far beyond just an academical experience.

A lot of contribution and help was received from other supportive researches as well. I would like to thank Dr. Domenico Lahaye for the time he dedicated to hone up my skills in numerical mathematics and finite elements analysis method in particular, importance of which in the current work is hard to overestimate. Special thanks to Dr. Karl-Heinz Wolf and Prof. Giovanni Bertotti, who provided me with constructive feedback and valuable literature references relevant to my topic. Thanks to Prof. Paulien Herder for accepting the invitation to be the committee member. I also greatly appreciate for checking the simulation results and thesis structure that PhD candidate Kishan Ramesh Kumar and research assistant Leila Hashemi did alongside with their feedback and comments. I am grateful for collaboration with Nordin Tippersma, who was working on similar topic, covering different material behaviour phenomenon. The struggle we have been through and the joy of overcoming it is unforgettable.

Finally, a big thanks to all of my friends, colleagues and family for support and encouragement you provided me with on every stage of my work. I would like dedicate this work to all of you.

A. A. Makhmutov
Delft, August 7, 2020

Abstract

A novel 2D finite element method (FEM) on unstructured grid for nonlinear time-dependent deformation of materials is developed. The objective is to model the complex deformation behavior of the rock salt, inside which caverns are mined to store green fuels (such as hydrogen). The analyses and the developments of the present work allow for quantification of the state of the stress of the cavern, and assess the safety and reliability of the storage structure over time. The novelty of the approach is in using minimization of the potential energy principle in conjunction with nonlinear creep deformation physics. While the available FEM-based simulators offer tools only for solving linear and non-linear elastic models, the developed simulator takes into account cyclic loading and material's damage evolution in time with possibility to predict the material's failure. Apart from that, the stored product (hydrogen) density is taken into account, which affects cavern's pressure variation with depth. Impurities, causing heterogeneous rock salt properties, are also considered in the developed model. Multivariate Gaussian distribution is utilized to generate distribution of the heterogeneous mechanical properties. Eulerian strains are introduced in the model to take into account deformation of the mesh. As such, the computational grid changes its geometry according to the deformation. Several numerical test cases are studied. Firstly, a consistency (verification) study is performed, to validate the linear elastic model. Remark that the linear elastic deformation model casts the basis of the non-linear model with creep physics. Then, several studies have been performed to analyse, quantify and approximate the deformation of the salt cavern under the gas pressure change, including the time-dependent creep physics. It is emphasised that the following modelling constitutive laws and assumptions are utilized in this project:

- Minimal total potential energy principle.
- Generalized Hooks law.
- Infinitesimal deformation theory.
- 2D Finite Element Method (FEM/FEA).
- Unstructured mesh with constant strain triangle (CST) elements.
- Euler forward (explicit) and backward (implicit) time discretization.
- Associated flow rule and the flow potential.
- Norton-Baileys power law to model the creep.

Contents

1

Introduction

1.1. Overview of the Hydrogen Salt Cavern Storage Technology

Hydrogen salt cavern storage is one of the energy storage technologies which allows to preserve energy for later use (figure 1.1). It usually represents a system of caverns, which are artificially created by the controlled dissolution of rock salt by injection of water during the solution mining process [27]. This process is called leaching and, as a result, caverns with volumes up to 300,000-500,000 m^3 and outliers up to 2,000,000 m^3 can be created [40]. Such caverns provide very high deliverability, i.e. excellent injection and production characteristics, and have (almost) perfect sealing properties [12, 13].

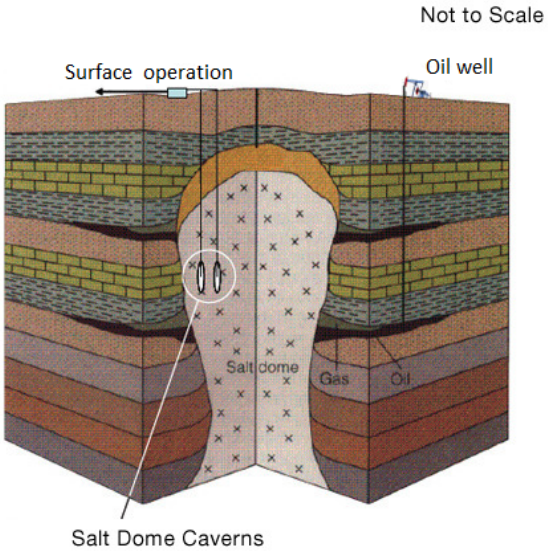


Figure 1.1: Salt cavern storage site [40].

The salt cavern construction focuses primarily on the salt diapirs - a type of structural domes, formed when a part of thick bed of salt migrates vertically into the surrounding rock strata. Primary depth target for salt cavern construction lies within 1,000-1,500 m depth range [27]. Within this depth range, the rock salt material behaves reasonably stable and permanent caverns can be constructed. The maximum depth limit is defined by the plastic behaviour of the salt at higher depths, as constructed cavities within those deep formations will eventually tend to close after the extraction. The minimum depth limit is usually defined by the cavern's operating pressure and strength characteristics of the rock salt.

The caverns usually have an elongated cylindrical shape with height from tens to a few hundred meters [12, 13, 27]. Such geometry is chosen because of its good stability. Spherical or pear-shaped

caverns usually are created in the depth range of 500 to 1,000 m [27]. If the salt layer has limited thickness then bell-shaped caverns with height up to 100 m can be formed [40].

Speaking of hydrogen, it is worth to highlight that hydrogen is an energy carrier, but not an energy source. It represents a green fuel, which can be consumed in fuel cells to produce electricity, water and heat. It can be also burned in domestic and industrial scales.

Hydrogen storage in salt caverns is considered as a large-scale storage with purpose to level down the difference between the demand and supply in the power grid. When an excessive amount of energy is produced on a windmill or solar panel power plant, electricity is used in electrolyzer to split water into hydrogen and oxygen with the use of electrolysis reaction. Afterwards, the produced hydrogen is compressed and injected into the subsurface cavern, where it is stored until it will be required to compensate the energy shortage in the power grid. In that case, the cavern is depleted and hydrogen is utilized either in a hydrogen fuel cell to generate electricity, or in the gas combustion unit, where it is usually burned in a mixture with methane. Then the generated electricity is supplied to the power grid.

1.2. Modeling the Salt Cavern Storage Deformation Behavior

When a salt cavern is excavated in a salt formation and filled with gas, it will be exposed to a load, which originates from the pressure difference between the cavern's operational pressure and the overburden (lithostatic) pressure. This pressure difference will deform the cavern and its surroundings. Therefore, modeling and predicting such deformations and corresponding stress distribution around the cavern is a crucial part of the cavern's reliability and safety analysis. The challenging part in modeling the salt rock deformation behaviour lies in its ability to creep – phenomenon when a solid material permanently deforms under the influence of persistent mechanical stresses. Moreover, the aforementioned cyclic loading conditions, to which the salt cavern is exposed in case when it is used as a hydrogen storage or buffer, bring additional engineering challenges, which need to be solved to predict the material failure and damage evolution.

1.2.1. Overview of the creep phenomenon

Essentially, after applying an external load, the material consistently goes through three stages of creep. These three creep stages are often called transient (primary or reduced) creep , steady (secondary or stationary) creep , and tertiary (accelerated) creep (figure 1.2). The primary creep stage is characterized by a monotonic decrease in the rate of the creep. Secondary creep is characterized by constant creep rate. Deformations of the secondary creep are large and of a similar nature to pure plastic deformations. The tertiary creep phase involves the formation of microscopic cracks, which subsequently lead to damage evolution and rupture of the material [2].

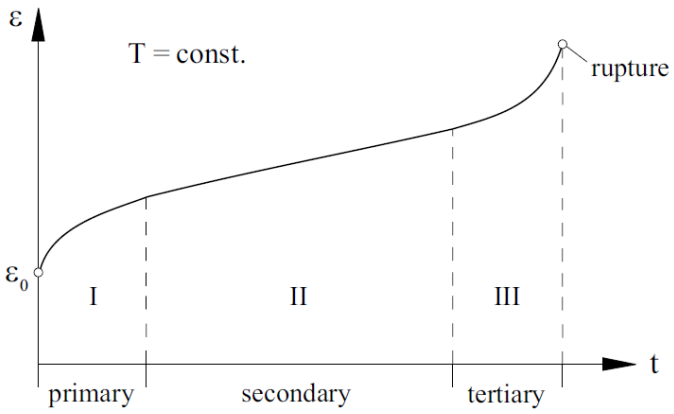


Figure 1.2: Creep Curve. ε_0 - instantaneous elastic strain [35].

1.2.2. Creep mechanisms

Creep often occurs within the temperature range of 20-200 °C [28]. The deformation mechanics are governed by dislocation within crystal and grain boundaries. There are two main deformation mechanisms that were investigated by means of the laboratory tests, displacement analysis and microstructural analysis. One is the dislocation creep mechanism, which is the primary objective in the modeling of rock salt creep behavior in the present work. It describes the dislocation effect caused by crystal defects and was studied in details in a number of laboratory experiments [1, 18, 34]. The dominant factor of the dislocation creep is temperature.

The other creep mechanism is diffusion creep or solution-precipitation creep, when a crystal grain slides along the crystal boundary. The dominant factors of this type of creep are the grains size of the rock salt crystals and temperature. The diffusion creep is considered to be dominant with decrease in the deformation rate or temperature [21]. The two aforementioned creep mechanisms as well as material failure are shown in figure 1.3.

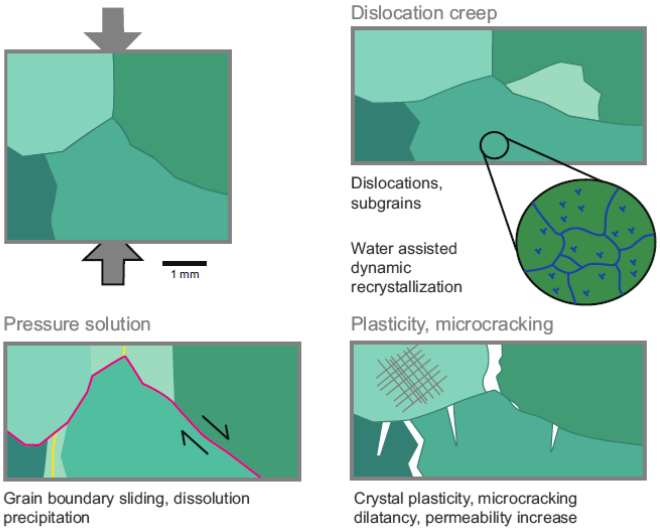


Figure 1.3: Illustration of different creep mechanisms of rock salt under confining test (top left picture). Different shades of green represent differently oriented crystals [41].

1.2.3. Utilization of existing models

In the past years, several numerical studies [7–9, 24, 25, 29, 30, 37, 38, 42] were published to describe salt structures deformation behavior on a large scale, modeling salt layers deformation and diapirs shaping processes. More recently, Multiscale Finite Element Method (MSFEM) based models [5, 6, 26, 39] were developed for solving linear and non-linear elastic problems in geomechanics. Finite Element Method (FEM) based models [23, 32] were developed to simulate salt cavern storage deformation behavior, which incorporate cyclic storage loading conditions and heat transfer between the cavern and its surroundings. The latter one utilizes and compares different existing rock salt constitutive models, namely Cristescu [11] and Lux/Hou [19, 31].

Within the framework of the current study, Gunther/Salzer [16] constitutive model is adopted to describe the rock salt material behavior. It can be used to describe all three creep stages and predict material damage by utilizing internal state (damage) parameters. The model is based on the power law creep relationship derived by quantifying the dislocation creep processes, observed in the laboratory tests [17, 20, 36]. Both implicit and explicit simulation models are developed to predict linear-elastic and non-linear creep response. The governing equations are discretized by FEM approach with the use of non uniform triangular mesh with constant strain triangle (CST) elements. Local linear basis functions are solved for displacements, considering boundary conditions. Then strains are recovered from nodal displacements for every element. Stress-strain constitutive relation is used to quantify the stress distribution, which is used to calculate creep strain rate for every finite element when solving problems with creep physics.

The model is further expanded by incorporating Eulerian strains and consideration of the stored product (hydrogen) density and corresponding hydrostatic pressure. Moreover, in comparison to other numerical models, irregular shaped cavern and heterogeneous domain are considered in the present work to study their effect on the cavern's state of the stress and nonlinear (time-dependent) deformation.

2

Methodology

2.1. Problem statement

To analyse the reliability and safety of the salt cavern structure, it is required to assess and quantify the degree of deformation of the cavern's surroundings and the corresponding stress distribution. To perform this task, a proper mathematical model needs to be developed. The core of the model lies in the theory of elasticity, which relates the forces applied to an object and the resulting deformations.

To demonstrate the concept, utilized in the present work to evaluate deformations of salt cavern surroundings, figure 2.1 is given, which shows an arbitrary solid body, exposed to external distributed loads $w(x)$ and point external loads F. Under the influence of the external loads, the body will deform with displacements u and D on the surface of the solid in compliance with the supporting reactions. As a consequence, internal stresses σ and strains ε will be created throughout the volume of the body. These stresses will trigger the creep phenomenon as it was described in the previous chapters. As a result, fictitious internal creep body forces F^{cr} will appear within the body, which will cause the time-dependent creep deformation with a certain creep strain rate.

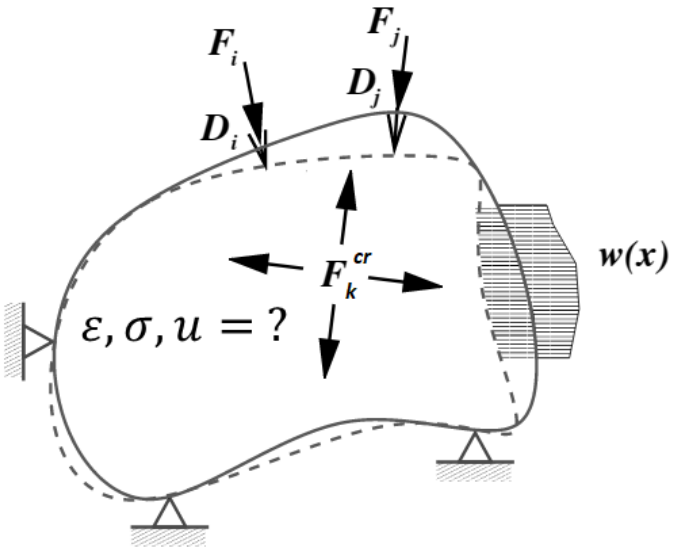


Figure 2.1: Arbitrary solid body exposed to external loads [35].

The given input of the problem is represented by the external loads and the sought solution is represented by the distribution of the displacements, strains and stresses throughout the volume of the body and their evolution in time.

2.2. Assumptions and Constitutive Relations

To develop the salt creep constitutive model, the following assumptions and constitutive relations are considered, some of which can be relaxed as will be shown in the following chapters:

- Infinitesimal deformation theory is valid.
- Generalized Hook's law is valid.
- Material is isotropic.
- Linear elastic response is instant (instant equilibrium).
- No chemical reactions between hydrogen and the rock salt occurs.
- No diffusion of hydrogen into the rock salt can happen.
- Cavern is instantly excavated and loaded at time $t = 0$. In case of cyclic load, pressure change is instantaneous.
- Adiabatic process, i.e. no heat transfer.

2.3. Potential Energy

Within the theory of elasticity, minimization of potential energy principle allows to derive the governing force balance equation. The total potential energy is represented by the sum of the strain energy stored in the deformed body in the form of stresses and the potential energy associated to the applied external and body forces:

$$\Pi = U + V, \tag{2.1}$$

where U is the internal strain energy, and V is the potential energy, associated with the applied forces.

The internal strain energy is calculated as:

$$U = \frac{1}{2}\int_V \varepsilon^T \sigma dV, \tag{2.2}$$

where σ is the stress tensor, and ε_{el} is the strain tensor.

The potential energy, associated with the applied loads is given as:

$$V = -W = -\int_S u^T f dS - \int_V u^T f^b dV, \tag{2.3}$$

where W is the work of the external forces, u is the vector of displacements, f is the vector of distributed forces acting on the part S of the surface, and f^b is the vector of body forces.

2.4. Minimization of Potential Energy

The minimum potential energy principle states that among all possible displacements and configurations of a conservative system that satisfies the equations of equilibrium, the correct state of the system is the one which minimizes the total potential energy [10], i.e.,

$$\delta\Pi = \delta(U + V) = 0. \tag{2.4}$$

2.5. Total Strain

The unknowns in the equation (2.2) are stress and strain tensors. To derive the force balance governing equation, it is necessary to express them through displacements by utilizing constitutive relations. Under the assumption of small strains (infinitesimal deformation), it is possible to decompose the total strain into elastic and inelastic strains, i.e., [35]:

$$\varepsilon = \varepsilon_{el} + \varepsilon_{ie}, \tag{2.5}$$

where inelastic strain can be represented as summation of thermal, plastic, creep and other types of strains, depending on the phenomena which governs the material behavior.

Within the framework of the current study, the main accent is put on modeling the steady state creep behavior of rock salt, as it is considered to be the most dominant one, after the elastic response. Therefore, the total strain will consist of elastic and creep components, and the elastic strain can be expressed as:

$$\varepsilon_{el} = \varepsilon - \varepsilon_{cr}, \tag{2.6}$$

or in terms of strain rates:

$$\dot{\varepsilon}_{el} = \dot{\varepsilon} - \dot{\varepsilon}_{cr}. \tag{2.7}$$

2.6. Stress-Strain Constitutive Relation and Material Parameters

The constitutive relation between the stress and strain tensors can be expressed in the form of the generalized Hook's law, i.e.,

$$\sigma = C : \varepsilon_{el}, \tag{2.8}$$

where C is the 4^{th} rank elasticity or stiffness tensor. For a 2D isotropic homogeneous material it can be expressed through the Lame's constants as

$$C_{ijkl} = \lambda\delta_{ij}\delta_{kl} + \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}), \tag{2.9}$$

or in matrix notation as

$$C = \begin{bmatrix} \lambda + 2\mu & \lambda & 0 \\ \lambda & \lambda + 2\mu & 0 \\ 0 & 0 & \mu \end{bmatrix}, \tag{2.10}$$

where δ is the Kronecker delta.

The Lame's constants can be expressed through the material parameters as

$$\mu = G = \frac{E}{2(1+\nu)}, \text{ and } \quad \lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \tag{2.11}$$

where E is the Young's modulus, G is the shear modulus and ν is the Poisson's ratio.

2.7. Creep Strain Constitutive Relation

The starting point in developing creep strain rate constitutive relation is the assumption that the creep strain is a function of stress and temperature [35], which in general notation can be written as

$$\dot{\varepsilon}_{cr} = f_\sigma(\sigma_{eq}) f_T(T). \tag{2.12}$$

Here, σ_{eq} and T are the equivalent stress and temperature respectively.

It has bee shown that for rock salts, the power law stress function gives accurate curve fitting results with the lab experiments [4]. The function can be written as

$$f_\sigma(\sigma_{eq}) = a\sigma_{eq}^n. \tag{2.13}$$

The temperature dependency is expressed by the Arrhenius law as

$$f_T(T) = \exp[-Q/RT]. \tag{2.14}$$

Assuming the Norton-Bailey type potential and von Mises type equivalent stress, the above equations can be written as

$$\dot{\varepsilon}_{cr} = \frac{3}{2} e^{-\frac{Q}{RT}} a\sigma_{vM}^{n-1} s, \tag{2.15}$$

where σ_{vM} is the von Mises equivalent stress, s is deviatoric part of the stress tensor, a and n are the material dependent constants, and finally Q, R and T denote the activation energy, Boltzmann's constant and temperature, respectively.

2.8. Strain-Displacement Relation

Under the assumption of infinitesimal deformations, it is possible to neglect the difference between the true stresses and strains and engineering stresses and strains. In terms of displacement gradient, the strain tensor can be written in general form as

$$\varepsilon = \frac{1}{2}(\nabla u + \nabla u^T + \nabla u^T \nabla u). \tag{2.16}$$

Taking into account the mentioned above assumption of infinitesimal deformations and isotropic material properties, the equation (2.16) further simplifies to

$$\varepsilon = \nabla^s u, \tag{2.17}$$

where ∇^s represents the symmetric gradient operator.

In this work both Lagrangian and Eulerian strains are considered in the model, which means that the derivatives of displacements are taken with respect to the original and deformed mesh, respectively. The developed code allows the user to flexibly choose between the two systems.

2.9. Tertiary Creep and Damage Evolution

The creep damage equations were proposed by L. Kachanov and Rabotnov [22]. Rabotnov introduced the damage variable ω. He assumed that the creep rate is not only a function of stress, but also it depends on the current damage state, i.e., the constitutive creep equation is of the form

$$\dot{\varepsilon}_{cr} = \dot{\varepsilon}_{cr}(\sigma, \omega), \tag{2.18}$$

where the damage state variable is expressed through the evolution equation as

$$\dot{\omega} = \dot{\omega}(\sigma, \omega), \quad \omega|_{t=0} = 0, \quad \omega < \omega_*. \tag{2.19}$$

Here, ω_* is the critical value of the damage, at which the given material breaks, and $\dot{\omega}$ is the damage evolution rate. The damage evolution rate is expressed as

$$\dot{\omega} = \frac{b\sigma^k}{(1-\omega)^l}. \tag{2.20}$$

Finally, the creep strain rate can be stated as

$$\dot{\varepsilon}_{cr} = \frac{a\sigma^n}{(1-\omega)^m}. \tag{2.21}$$

Variables a, b, n, m, l and k in the equations (2.20) and (2.21) above represent the material dependent constants. It is also important to note that in the case of $\omega = 0$, equations (2.21) represents the well known power law constitutive equation.

3

Simulation Results

This chapter presents numerical results of a series of 2D test cases to demonstrate the capabilities of the developed simulator: (1) Linear elastic model under constant load, (2) Validation of the linear elastic model results, (3) Creep model under constant load, (4) Creep model under cyclic load, (5) Complex heterogeneous model, (6) Irregular cavern shape model with and without heterogeneity, (7) Tertiary creep and material failure, (8) Multi-caern system. The geometry and the mesh (as shown in figure 3.1) were generated using the open source software Gmsh [15], which allows to refine the elements where it is necessary, e.g. near the cavern's wall. The values of the input parameters used in the simulator are summarized in the table 3.1 with corresponding references.

Table 3.1: Input parameters for simulation

Parameter	Value
Overburden rock density, [kg/m^3]	2200
Rock salt density, [kg/m^3]	2200 [33]
Rock salt bulk modulus, [GPa]	24.3 [33]
Rock salt shear modulus, [GPa]	7.5 [33]
Depth of the top of the salt layer, [m]	500
Hanging wall thickness, [m]	200 [13]
Foot wall thickness, [m]	200 [13]
Cavern's volume, [1000 m^3]	500 [13]
NL Temperature gradient, [°C/km]	31.3 [3]
Creep constant a, [Pa^n]	8.10E-26 [4]
Creep exponent n, [-]	3.5 [4]
Creep activation energy Q, [J/mol]	51600 [4]

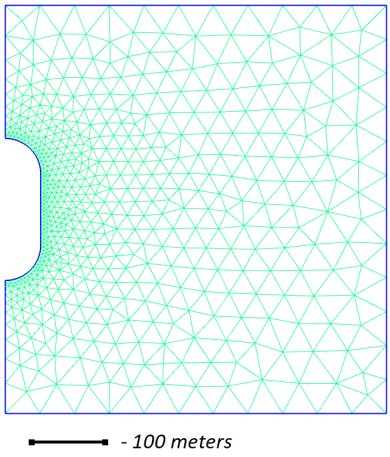


Figure 3.1: Vertical cross-sections of the geometry of the model and the generated mesh using Gmsh.

3.1. Defining the Load for the Boundary Conditions

To reflect the loading conditions as close to the reality as possible, the cavern's pressure and lithostatic pressure are considered. According to [13], the cavern's pressure should not exceed 24%-80% of lithostatic pressure. As such, the minimum allowable pressure difference between the cavern's pressure and lithostatic pressure will be at the cavern's roof location when it is charged and the maximum will be at the bottom of it when it is depleted (points 1 and 2 in the left figure 3.2, respectively). This makes the cavern's pressure a function of overburden rock density, rock salt density and depth of the roof and the bottom of the cavern.

Once minimum and maximum allowable cavern pressure and corresponding pressure difference are estimated, their values are used to calculate equivalent surface forces acting on the cavern's wall (figure 3.2 right). These surface forces then converted into equivalent nodal forces to be used in the numerical model as an input.

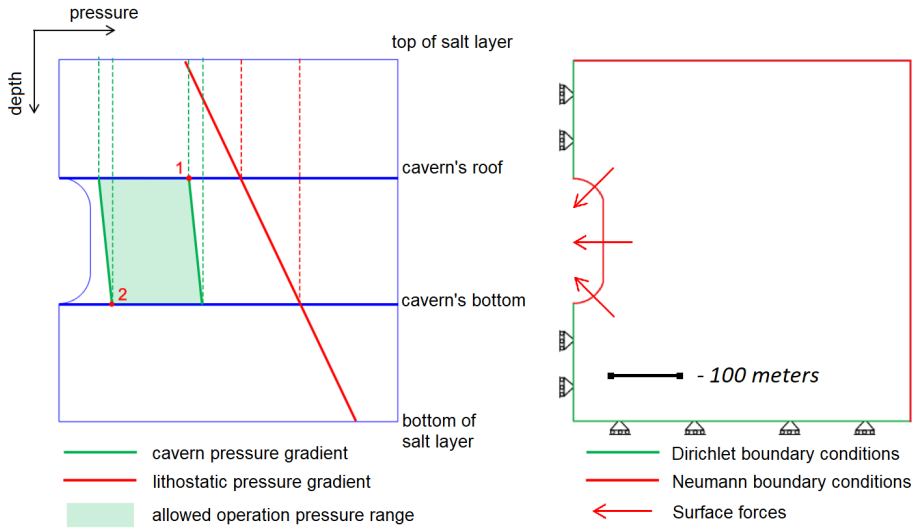
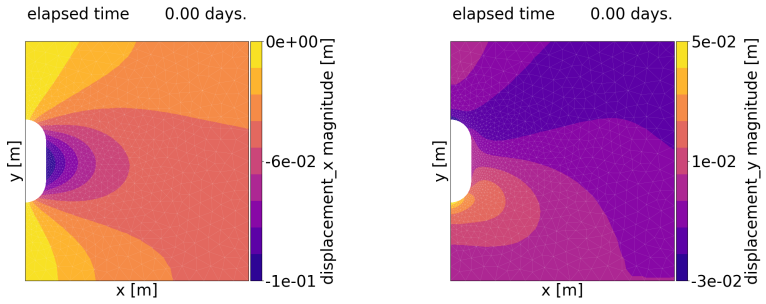


Figure 3.2: Pressure gradients and safe pressure range within the salt domain (left) and Implied numerical model boundary conditions (right).

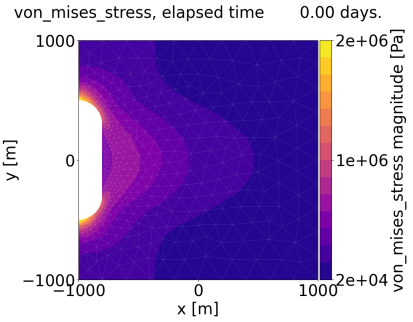
3.2. Numerical Test Cases

3.2.1. Linear elastic model under constant load

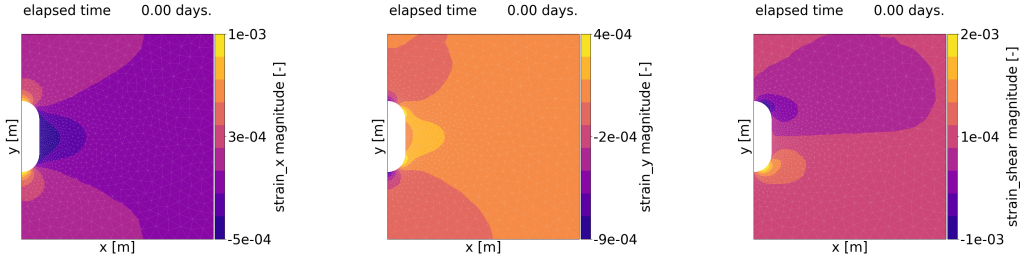
First of all, a solution for linear elastic model ($F_{cr} = 0$) under constant load is calculated. The results of the numerical simulation are given in figure 3.3. Figures show the instantaneous system response to the external load acting on the cavern's wall. As a result, the system is deformed (a, b), developed strains $(d - f)$ and the external forces are balanced with internal stresses $(g - i)$.



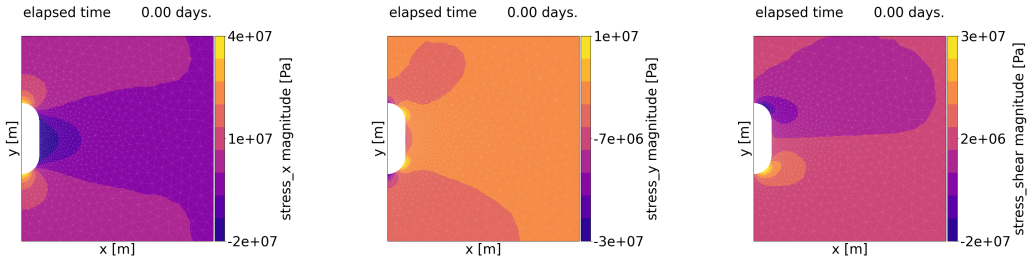
(a) Displacement in x direction. (b) Displacement in y direction.



(c) Von Mises Stress.



(d) X component of strain. (e) Y component of strain. (f) Shear component of strain.



(g) X component of stress. (h) Y component of stress. (i) Shear component of stress.

Figure 3.3: Results of solving the linear elastic model under constant load.

3.2.2. Validation of the linear elastic model results

The validation of the numerical solution from the previous section 3.2.1 is done by comparing it with synthetic analytical results as it was done in [39]. For this purpose, displacements in x and y directions are defined as arbitrary functions of coordinates:

$$u(x,y) = \sin\left(\frac{\pi x}{L}\right)\sin\left(\frac{\pi y}{W}\right)$$
$$v(x,y) = \sin\left(\frac{\pi x}{L}\right)\sin\left(\frac{\pi y}{W}\right) \tag{3.1}$$

where $L = W = 1000$ m - are the domain's length and width respectively.

With known displacement functions, strains, stresses and nodal forces can be easily calculated. These analytically retrieved nodal forces then used as an input to calculate numerical solution for the mentioned parameters. Comparison of the analytical and numerical solutions (figure 3.4) is done by taking the absolute value of their difference as demonstrated in figure 3.5.

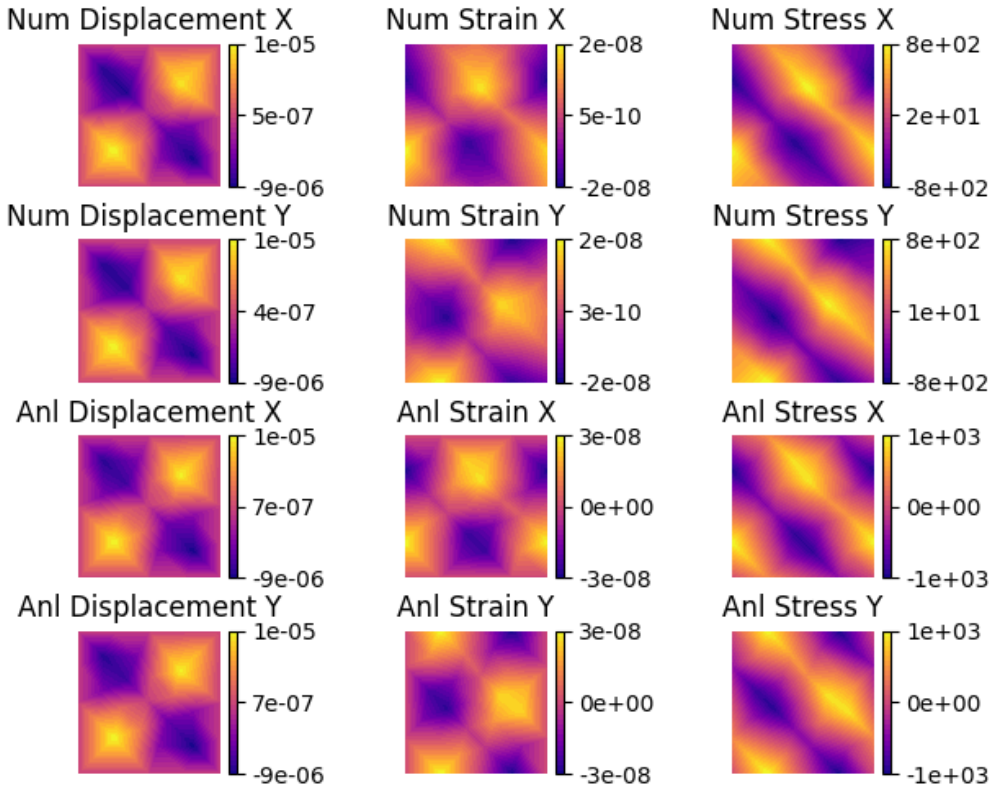


Figure 3.4: Synthetic analytical and numerical solutions. 56 elements.

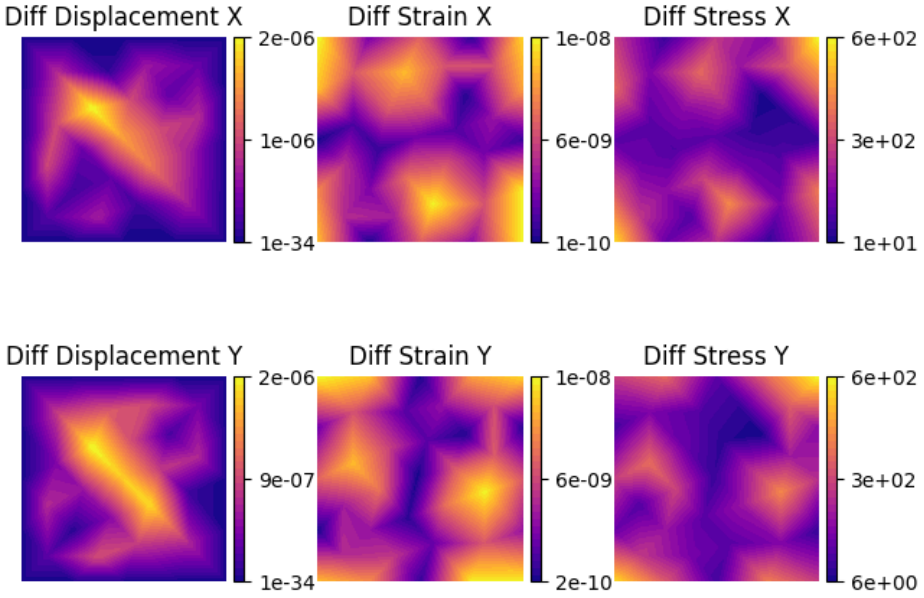
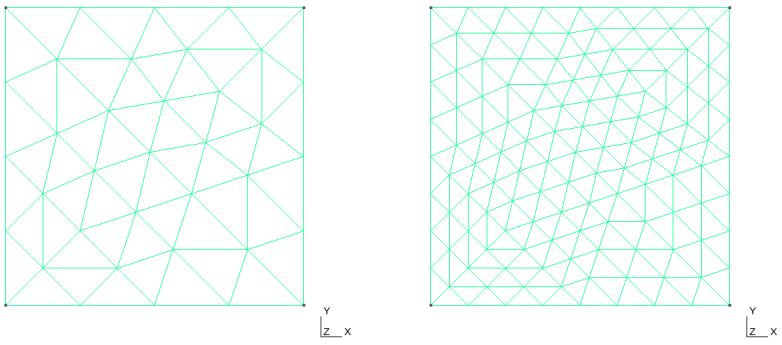


Figure 3.5: Difference between synthetic analytical and numerical solutions. 56 elements.

The mentioned above procedure was repeated three more times with the consequent mesh refinement at every step, in such a way that an arbitrary finite element is split into smaller elements by reducing each edge of the original element by half as it is shown in figure 3.6. The finest mesh grid results are given in the appendix (figures A.3-A.4).



(a) Original mesh, 56 elements. (b) 1 times refined mesh, 224 elements.

Figure 3.6: Grid refinement.

After every refinement, an error of the numerical approximation is calculated as the two norm of the difference between the analytical and numerical solutions:

$$e = ||g(x) - \hat{g}(x)||_2 \tag{3.2}$$

where $g(x)$ is the analytical synthetic (exact) solution of a certain parameter (displacement, strain, stress), and $g(\hat{x})$ is the numerical approximation. Then the order of numerical approximation is estimated according to

$$P = \frac{\ln(e_{n+1}) - \ln(e_n)}{\ln(dx_{n+1}) - \ln(dx_n)}. \tag{3.3}$$

Here, dx is representative finite element size, with subscripts $n+1$ and n denoting two subsequent refinements of the mesh.

The rate at which the error of the numerical solution decreases is represented by a slope of the line $e = f(dx)$ in the loglog scale as shown in figure 3.7. As it can be seen, the resulting solution for the displacement is 2nd order accurate in space, while solution for the strain and stress is 1st order accurate.

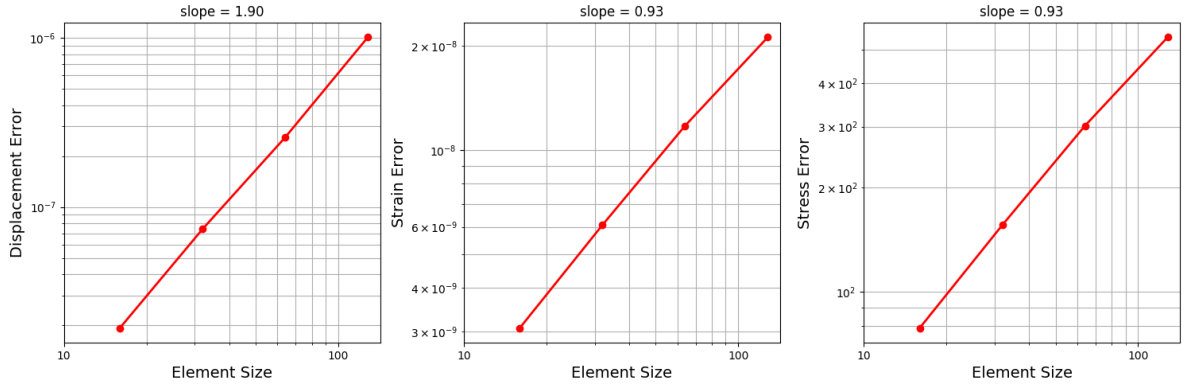
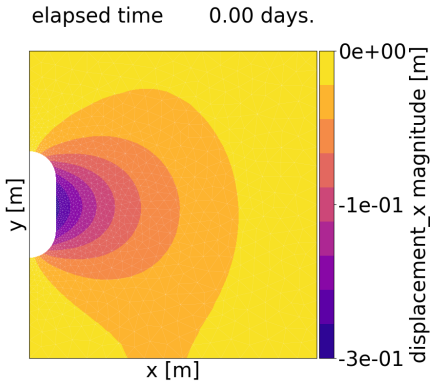


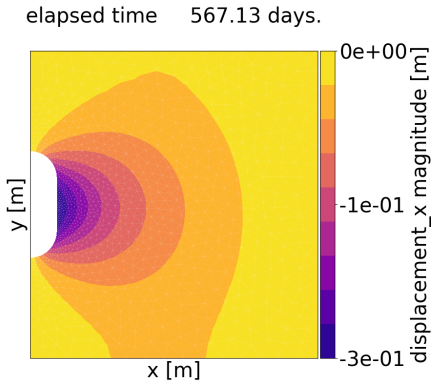
Figure 3.7: Error of the numerical solution vs finite element size.

3.2.3. Creep model under constant load

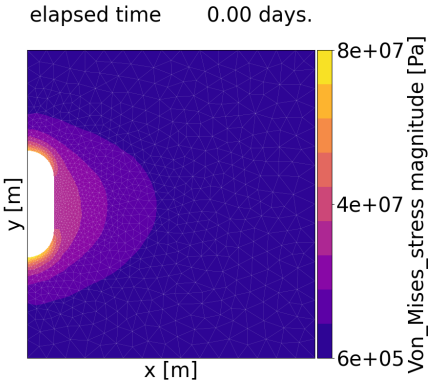
After solving the problem for linear elastic model, fictitious creep forces are introduced to the governing equations. This makes the output solution time dependent. The rest of the parameters are kept the same as in the model in the section 3.2.1. Figures 3.8 $a-b$ show displacement evolution over time in x direction, figure 3.8 c shows von Mises stress distribution. Figure 3.9 shows strain evolution (a) under the constant stress (b) for an arbitrary point of the domain.



(a) Displacement in x at the first time step.

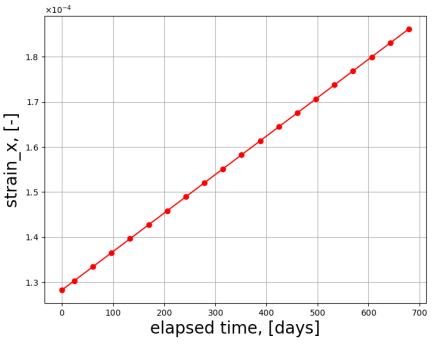


(b) Displacement in x at the last time step.

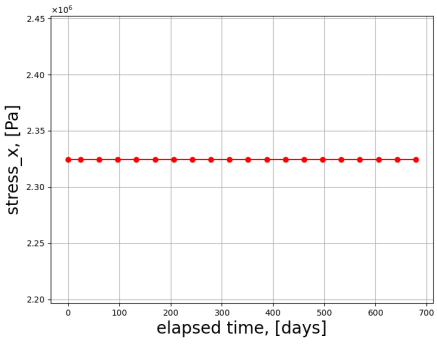


(c) Von Mises stress.

Figure 3.8: Solution obtained by solving the creep model.



(a) Strain evolution.



(b) Stress evolution.

Figure 3.9: Constant stress, secondary creep model.

3.2.4. Creep model under cyclic load

To take into account cyclic loading, the cavern's pressure is assumed to be a function of time as a discrete periodic time signal that can take two values: maximum cavern's pressure and minimum cavern's pressure over a certain time period (figure 3.10). Figures 3.11 a, d show the displacement, b, e - strain and c, f - fictitious creep forces evolution over time for an arbitrary point of the domain under the cyclic loading conditions. The full cycle of the cavern's loading unloading in the given results is set to 6 days.

The high peak values are related to instantaneous elastic response of the rock salt material, after which there is a short period of creep development with slope of the line representing the magnitude of the creep rate, just like in the previous case with constant load. The higher the load, the steeper creep strain line is observed (higher creep rate).

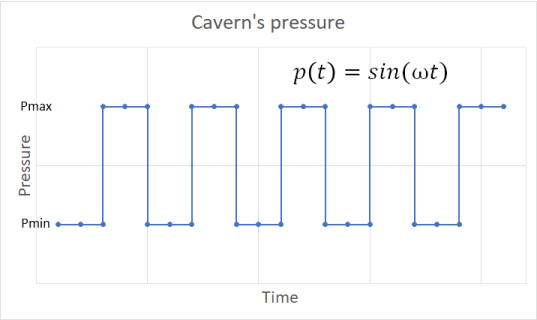
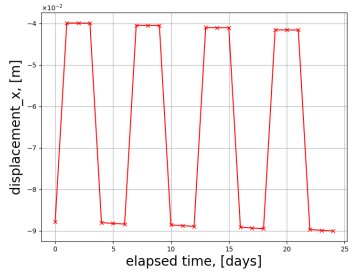
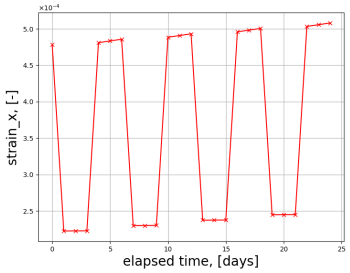


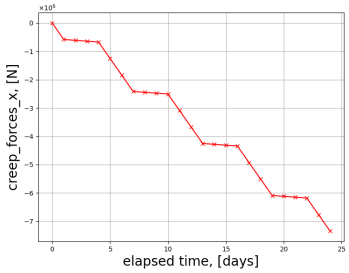
Figure 3.10: Pressure variation over time as discrete periodic signal.



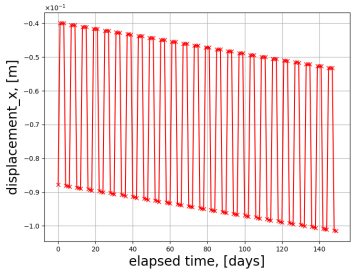
(a) X component of displacement evolution in time. 25 time steps.



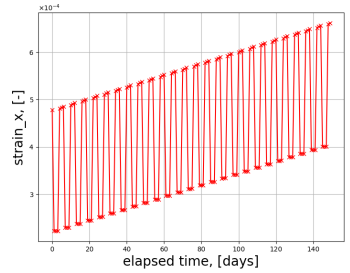
(b) X component of strain evolution in time. 25 time steps.



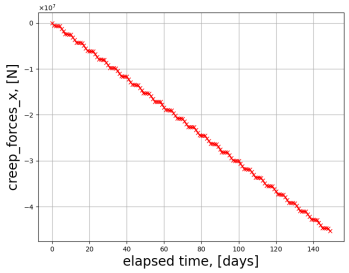
(c) X component of creep forces evolution in time. 25 time steps.



(d) X component of displacement evolution in time. 150 time steps.



(e) X component of strain evolution in time. 150 time steps.

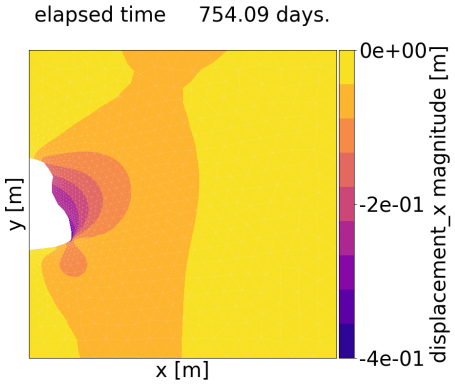


(f) X component of creep forces evolution in time. 150 time steps.

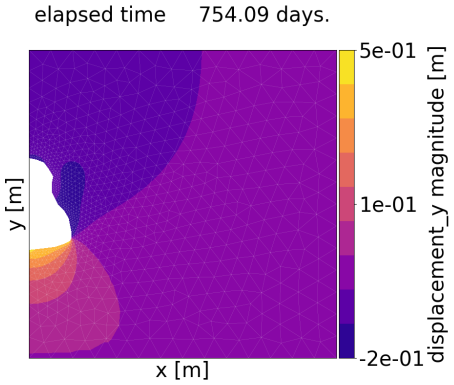
Figure 3.11: Solution obtained by solving the cyclic load model.

3.2.5. Irregular cavern shape model

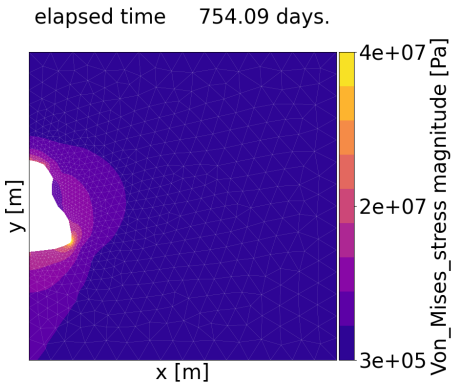
Heterogeneity of the salt rock may affect the solution mining process, which in its turn will affect the shape of the constructed cavern. Figures 3.12 represents results simulation model with such irregular cavern shape. From the figures it can be seen, that the maximum amplitude of displacement is increased in comparison to the regular shaped cavern, which can be explained by increased equivalent surface forces due to increased surface of the cavern. Also the pattern of the displacements is changed, which can be explained by the increased stiffness of the arch-shaped convex parts of the cavern – displacements are less in the magnitude at the top and bottom of the cavern.



(a) Displacement in x at the last time step.



(b) Displacement in y at the last time step.



(c) Von Mises stress.

Figure 3.12: Solution obtained by solving the creep model for irregular bell shaped cavern.

3.2.6. Heterogeneity implementation for irregular cavern shape case

A multivariate Gaussian distribution was used to generate a variation of mechanical properties, namely Young modulus and Poisson ratio as shown in the left figure 3.13 for halite. The range for Poisson ratio is taken 0.10-0.43 with mean value of 0.3 and Young modulus 27.2-58.7 GPa with mean value of 44.4 GPa [14].

In a similar approach more minerals with different mechanical properties were introduced, such as anhydrite, potassium salt and shale rock, with corresponding Young modulus and Poisson ratio correlation as shown in the left figure 3.13. The right figure 3.13 shows the resulting test case with potassium salt lens and a shale layer.

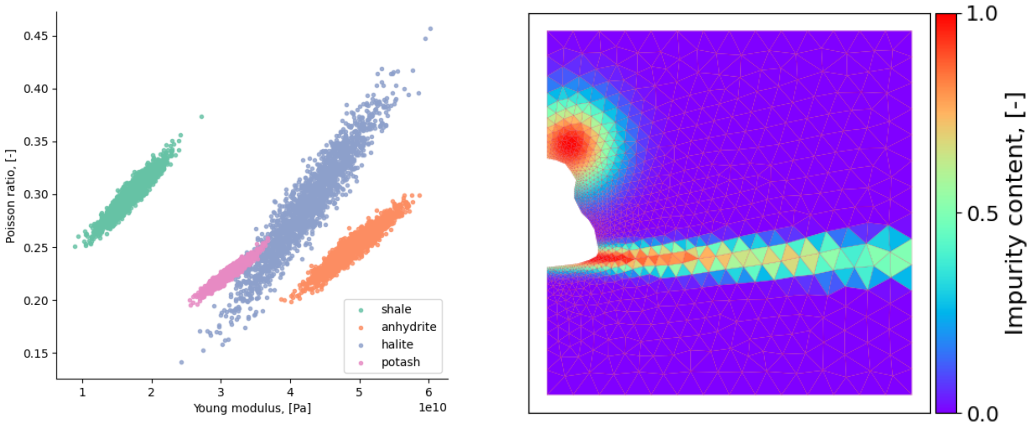
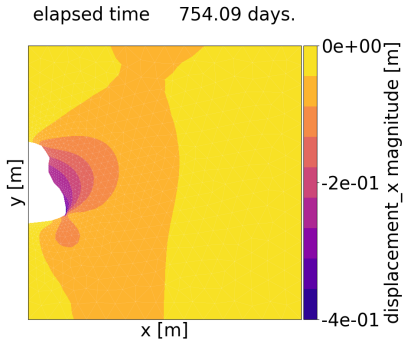
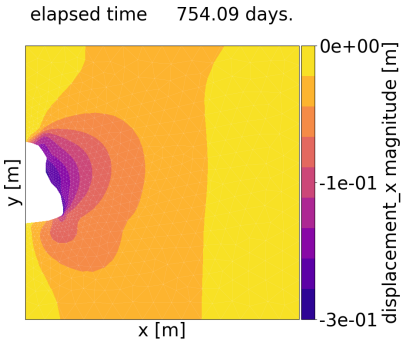


Figure 3.13: Rock properties distribution (left) and numerical model with potash lens and shale layer impurities (right).

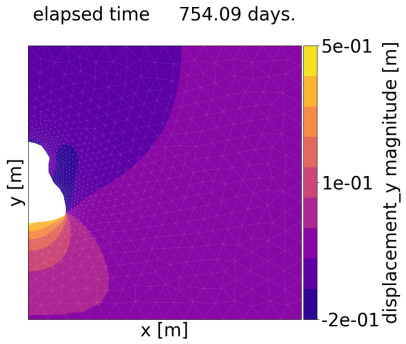
Figures 3.14 $b - e$ show comparison of the simulation results of the two cases – with and without heterogeneity. The effect of the more plastic potash salt is obvious in both directions: the displacement both in the x and y directions is now bigger in magnitude in the upper part of the cavern. The shale layer, on the other hand, does not show any significant effect on mechanical response of the cavern's surroundings. The rest parameters of the two simulations are given in the appendix figures A.5 and A.6.



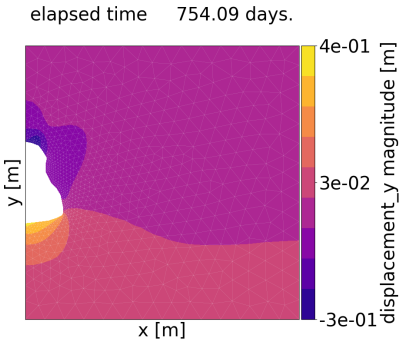
(a) Displacement in x at the last time step. Homogeneous Domain.



(b) Displacement in x at the last time step. Heterogeneous domain.



(c) Displacement in y at the last time step. Homogeneous Domain.

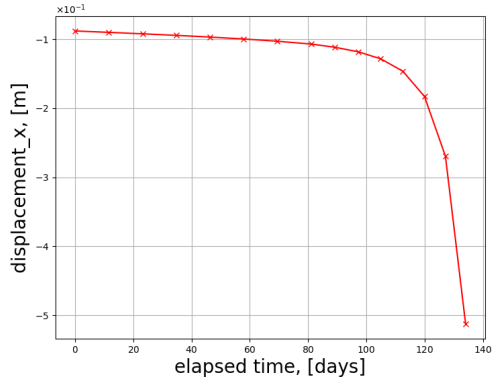


(d) Displacement in y at the last time step. Heterogeneous domain.

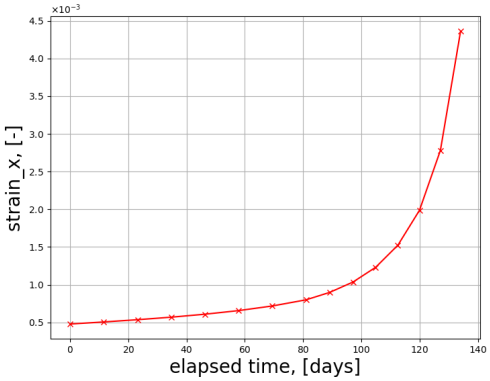
Figure 3.14: Solution obtained by solving the creep model for irregular bell shaped cavern with impurities.

3.2.7. Tertiary creep and material failure

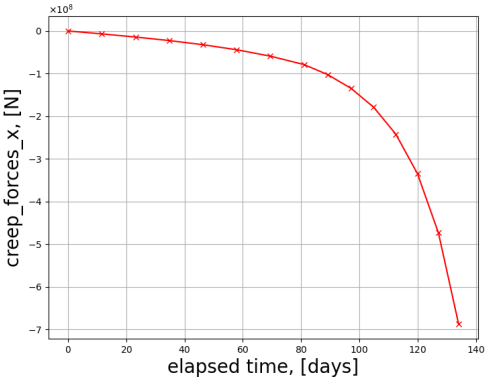
Last scenario represents a test case with damage evolution parameters. Figure 3.15 shows transition from secondary creep to tertiary state and subsequent material failure. The given example shows the solution for a point of a maximum strain, i.e. the first nodal point, where the failure of the material will begin. For a simple case, excluding heterogeneity of the rock salt, such point will be always on a cavern's wall, but in real life it will not be always the case.



(a) Displacement in x direction evolution.



(b) X component of creep strain.



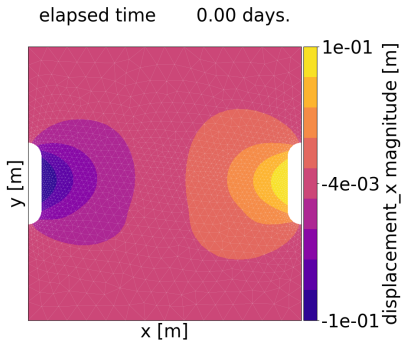
(c) X component of creep forces.

Figure 3.15: Solution obtained by solving the creep model with damage evolution.

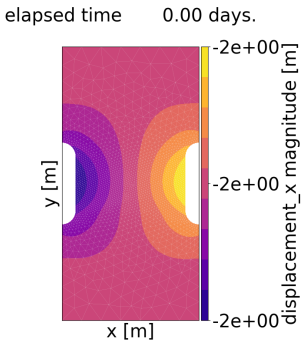
3.2.8. Multi-cavern system

Figure 3.16 shows the results for a multi-cavern system, which are provided to further demonstrate the capabilities of the developed simulator. As in the previous cases, deformation, strains and corresponding stresses are calculated, which can be used to further calculations to estimate the safe distance between the caverns (cavern wall). In the given example, homogeneous domain is considered. In the first case the distance between the caverns is 500 m, in the second case 150 m.

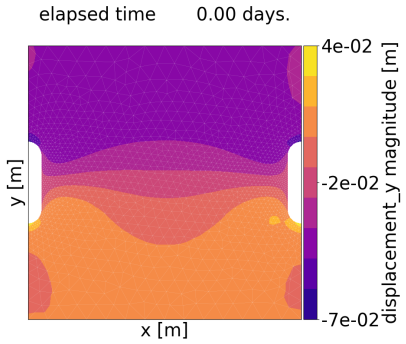
From the results it can be seen, that the second case with reduced cavern wall undergoes more significant deformation and the corresponding stress has higher magnitude due to the reduced stiffness of the system. It confirms the fact, that in the real life, caverns significantly affect each other and the overall stability of the system [27].



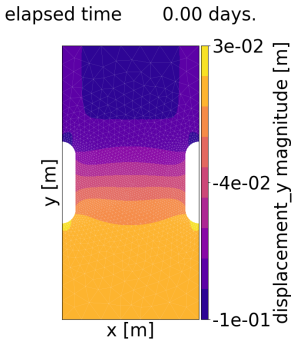
(a) Displacement in x direction, case 1.



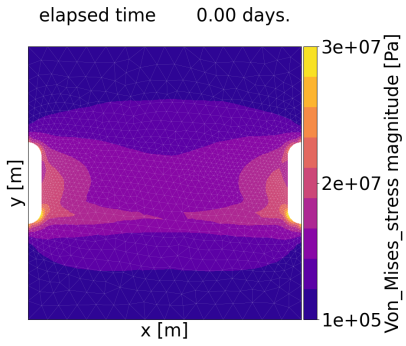
(b) Displacement in x direction, case 2.



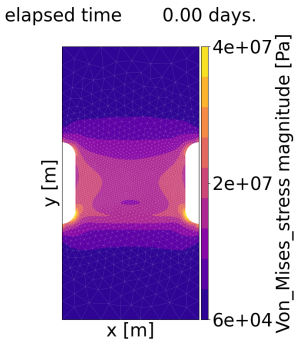
(c) Displacement in y direction, case 1.



(d) Displacement in y direction, case 2.



(e) Von Mises stress, case 1.



(f) Von Mises stress, case 2.

Figure 3.16: Results of solving the multi-cavern system model.

3.3. Sensitivity Analysis

To test the sensitivity of the model, the response of the system was studied by changing the input parameters of the base case from table 3.1. The system's response is studied on displacement results for an arbitrary point A of the domain (figure 3.17 left).

As it can be seen from the results, the closer the analysed finite element is to the cavern's boundary the more it is subjected to deformation (figure 3.17 right). The depth of the roof of the cavern (figure 3.18 (a)) and shear modulus (figure 3.19 (b)) have noticeable significant impact both on elastic and creep response of the system. Temperature has impact only on the creep rate, which exponentially increases with increasing temperature (figure 3.18 (b)). Bulk modulus variations are almost negligible (figure 3.19 (a)). Sensitivity results of the heterogeneous model (figure 3.19 (c)) show that the more plastic potash lens has an effect on the cavern's roof behaviour: there is noticeable difference between points A and A2, which are located close to the cavern's roof, where the potash lens is placed. Shale layer on the other hand does not show any significant effect: peripheral points B and B2 does not show significant difference in their behaviour. However important thing to note here that only mechanical behaviour is considered in this test. In real life other than mechanical unfavourable effects can be related to shale layer, such as leakage for example.

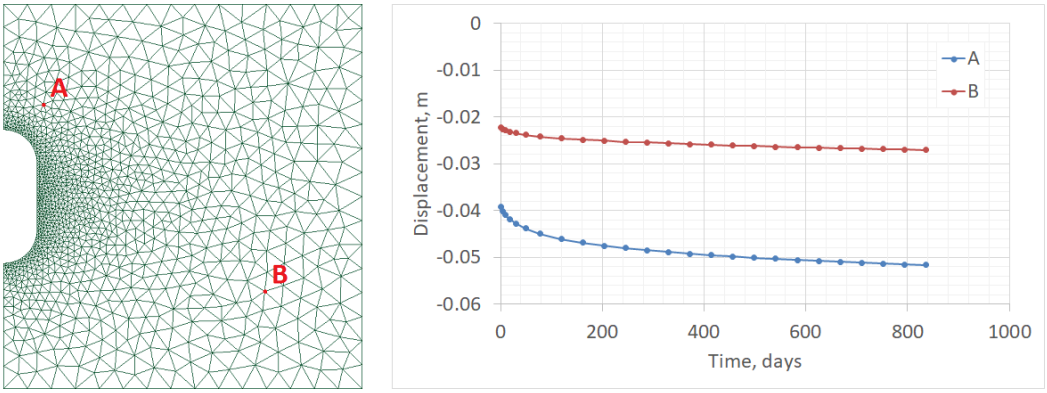
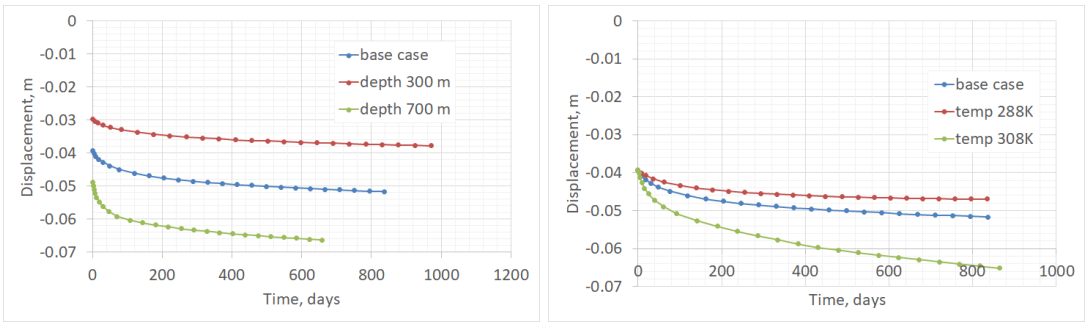


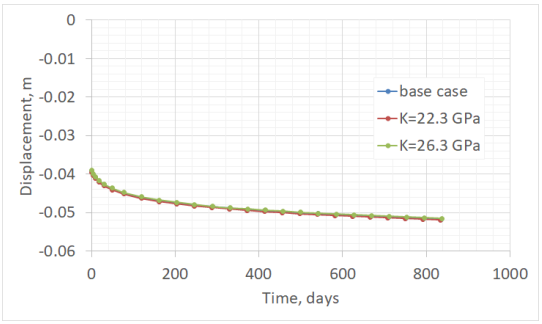
Figure 3.17: Analysed nodes (left) and Displacement results (right).



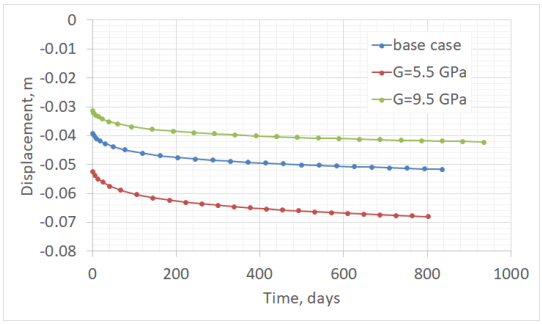
(a) Depth of the cavern.

(b) Temperature of the surrounding rock salt.

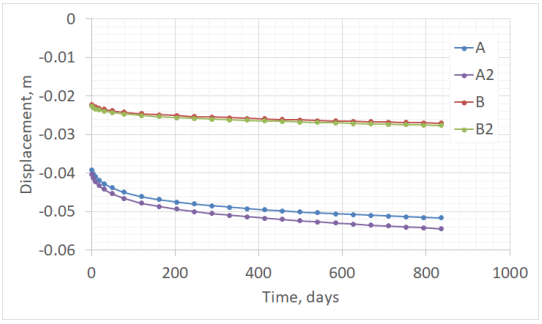
Figure 3.18: Sensitivity results of the model.



(a) Bulk modulus.



(b) Shear modulus.



(c) Heterogeneity sensitivity.

Figure 3.19: Sensitivity results of the model.

To emphasise the effect of temperature on creep, the figure 3.20 is provided. It shows how the creep strain rate is changing with changing temperature for different pressure regimes. In a rough estimation, the variation in temperature of 10 °C gives 10 times increase in creep strain rate. The difference in creep strain rate at temperatures of 20 °C and 120 °C is more than 100 times. The same temperature effect was observed in many laboratory experiments [1, 18, 34]. Based on these experiments a database was build by Shi-Yuan Li in [28], who observed that for every 50 °C temperature increase, the corresponding increase of the strain rate is around 1.5-2 orders of magnitude, which is in compliance with the results of the current work.

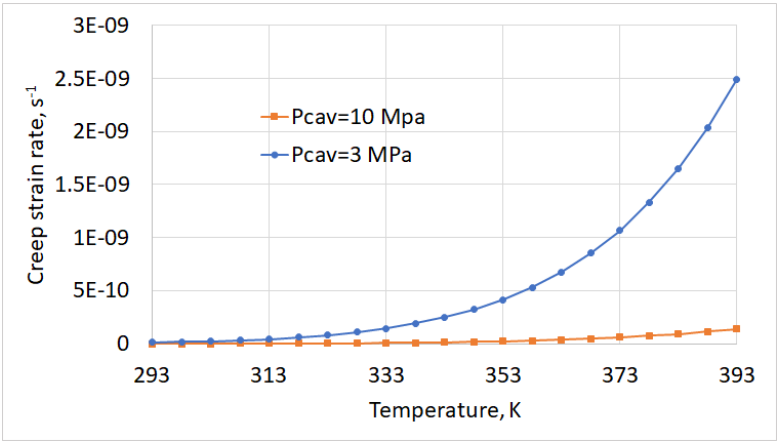


Figure 3.20: Temperature effect on the creep strain rate.

4

Conclusions

In this work, first 2D finite element method for rock salt creep behaviour modeling on unstructured grid was developed. The unique characteristics of the present work are following: 1) the secondary creep behaviour is introduced into the initial linear elastic model by using the vector of fictitious creep forces in the mathematical model, which is based on minimization of the potential energy principle, 2) the tertiary creep behaviour is introduced into the model by utilizing the damage evolution parameters, allowing to predict failure of the material 3) cyclic loading is taken into account by using a discrete time signal to model the pressure variations in the cavern, 4) various types of the rock salt heterogeneity is taken into account, 5) both Lagrangian and Eulerian strains are utilized in the simulation.

A consistency check was done, showing the validity of the numerical model solution. Two time discretization methods are proposed, namely both Euler forward and backwards, which allows to 1) use explicit method with high computational performance for simple scenarios and 2) use implicit method, when it is required to get rid off the CFL constraints and increase the time step size.

The developed simulator was extensively tested on various models: linear elastic and non-linear creep, homogeneous and heterogeneous, varying loading conditions, with and without damage evolution, also different boundary conditions were tested. Test cases with irregular cavern shapes demonstrated, that the simulator is also capable to take into account the geometry and its effect on the stiffness of the structure. Moreover, the simulator can be used for deformation and corresponding stress evaluation in multi-cavern system, which can be used as a reference during design of the salt cavern energy storage, caverns dimensions and distances between them.

Speaking about creep, all test cases showed that it is a very slow process and on a short time scale it is insignificant, sometimes almost negligible in comparison to the linear elastic response of the system. However, on a scale of several years the effect of creep becomes obvious, especially on the stage when tertiary creep starts to develop.

4.1. Future Work

The developed simulator showed descent stability and absence of convergence issues on a broad variety of different test case scenarios. In this section, topics for further improvements of the different aspects of the developed simulator are highlighted.

4.1.1. 3D FEM

With 2D simulator showing reliable results, it is absolutely necessary to have a 3D expansion before drawing accurate conclusions about real life rock salt behaviour. With complex geometry and material heterogeneity and anisotropy, expanding the simulator to three dimensional space will definitely show more complex material behaviour, which may be missing in 2D version and which may significantly change the material fatigue or damage evolution physics.

4.1.2. Optimization and efficiency

With descent performance of the developed 2D simulator, even without moving to the three dimensional space, there are plenty of aspects where the performance and efficiency can be improved.

Amount of elements of the mesh grid plays a significant role on the performance. However there is no algorithm which would control the elements size, it was arbitrary chosen beforehand. However, using adaptive mesh refinement it could be possible to always keep the optimum number of elements, which would still give results of the desired accuracy and without convergence issues and at maximum performance.

The slowest performance was shown when using Eulerian strains in combination with implicit time integration as coordinates were updated at every time step and every iteration. As a result, all of the finite elements with respective properties, e.g. mechanical properties, elasticity tensor etc. needed to be initialized again with every coordinate update which in the end heavily affected the performance of the algorithm. One of the proposed solutions could be transfer to more suitable computational environment such as C++ and using parallel thread programming algorithms such as Compute Unified Device Architecture (CUDA by nvidia), which will significantly increase the performance, as all finite elements can be initialized simultaneously using the parallel programming, instead of using the 'for' loop.

4.1.3. Improved Physics

The simulator is written in object oriented programming finite elements methods (OOPFEM) structure, which allows for wide possibilities of easy modular physics expansion. As such, for example viscoplasticity behaviour or diffusion of hydrogen into the rock salt effect on mechanical properties can be added as expansion modules.

As it was shown in the sensitivity analysis, temperature plays a crucial role in the creep deformation mechanism. However, at this point only adiabatic process is considered in the present work. Therefore, one can think about further physics improvement by taking into account heat transfer between the stored gas and the cavern surroundings.

4.1.4. Tertiary Creep

Tertiary creep and damage evolution were presented in the current work by utilizing Gunther/Salzer [16] constitutive model. However it is important to note here, that at this point it was not possible to validate the results of the tertiary creep model due to absence of the material parameters for the rock salt that are used in the governing equation - parameters b, k, l, m in the equations (2.20, 2.21). Therefore, it is recommended to perform relevant multi-axial laboratory tests to derive these parameters. In this work arbitrary parameters were used just to demonstrate the simulator capabilities.

A

Finite Element Method and Code Implementation

A.1. Finite Element Method

A.1.1. 2D Triangular Elements

To solve the problem formulated in the previous chapter numerically, it is necessary to discretize the domain with finite elements. Numerical solution for the displacements will be sought at nodal locations of each element, whereas strains and stresses will be recovered for the entire element from the Gauss points. Such elements are called Constant Strain Triangle (CST) elements (figure A.1) and widely used in the structural mechanics.

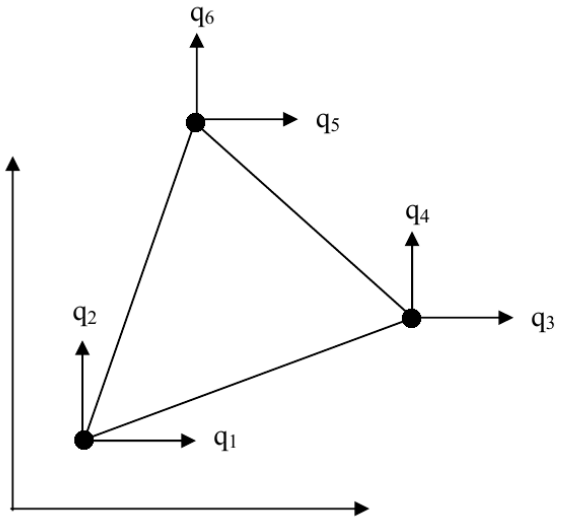


Figure A.1: 2D finite element with nodal displacements.

Let us write the local displacement vector for arbitrary finite element as:

$$q = \{q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6\}^T \tag{A.1}$$

where odd indexes denote the displacements in the x direction and even indexes - in the y direction.

For the whole domain, the global displacement vector can be written as:

$$Q = \{Q_1 \quad Q_2 \cdots \quad Q_{2i-1} \quad Q_{2i} \cdots \quad Q_{2n-1} \quad Q_{2n}\}^T \tag{A.2}$$

where n - is the total number of nodes.

A.1.2. Shape Functions

To compute the displacements for an arbitrary point inside the triangle (figure A.2) linear shape functions are utilized to interpolate the nodal displacement values. For the triangular element in plane

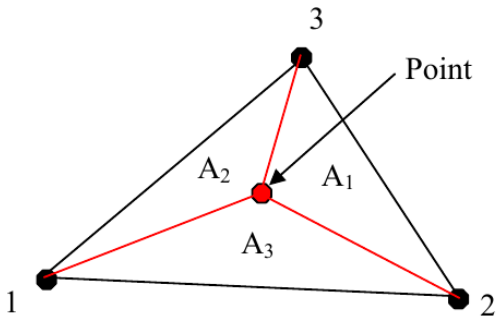


Figure A.2: Interior point interpolation.

stress, the discretized displacement vector field can be written as follows:

$$\begin{aligned} u(x,y) &= N_1(x,y)q_1 + N_2(x,y)q_3 + N_3(x,y)q_5 = [N]\{u\} \\ v(x,y) &= N_1(x,y)q_2 + N_2(x,y)q_4 + N_3(x,y)q_6 = [N]\{v\} \end{aligned} \tag{A.3}$$

where N_i - are the interpolation functions, u and v - displacements of an arbitrary interior point in the x and y directions respectively.

The shape functions can be derived as follows:

$$N_i = \frac{A_i}{A} \tag{A.4}$$

where A_i - is the area of the triangular element, opposed to the node i. Code implementation of the shape functions class is shown in A.1.

```
class Shapefns(object):
    """
    Define shape functions
    These will be defined on the local coordinates xi and tau
    Shapefns()
    eval(n,xi): phi[n](xi, tau)
    ddxi(n):  dphi[n](xi, tau)
    ddtau(n):  dphi[n](xi, tau)
    size(): number of nodes for these shape functions
    """

    def __init__(self):
        """
        an array of functions for phi and deriv phi
        """
        # linear shape functions
        self.__phi = [lambda xi, tau: xi,
                      lambda xi, tau: tau,
                      lambda xi, tau: 1 - xi - tau]
        # and derivatives of phi w.r.t. xi and tau
        self.__dphidxi = [1, 0, -1]
        self.__dphidtau = [0, 1, -1]
        self.__N = 3  # number of nodes in element

    def eval(self, n, xi, tau):
        """
        the function phi[n](xi, tau), for any xi and tau
        """
        return self.__phi[n](xi, tau)

    def ddxi(self, n):
```

```
        """
        the function dphidxi[n]
        """
        return self.__dphidxi[n]

    def ddtau(self, n):
        """
        the function dphidtau[n]
        """
        return self.__dphidtau[n]

    def size(self):
        """
        the  number of points
        """
        return self.__N
```

Listing A.1: Shape functions and their derivatives

As it can be seen from figure A.2 and equation (A.4) the sum of all shape functions in the element equals to one, which means, that they are not independent and by knowing two shape functions it is possible to define the other one. As such, let us express the shape functions through local coordinates as follows:

$$N_1 = \xi, \quad N_2 = \eta, \quad N_3 = 1 - \xi - \eta \tag{A.5}$$

Substituting (A.5) into (A.3) yields:

$$\begin{aligned} u(x,y) &= (q_1 - q_5)\xi + (q_3 - q_5)\eta + q_5 \\ v(x,y) &= (q_2 - q_6)\xi + (q_4 - q_6)\eta + q_6 \end{aligned} \tag{A.6}$$

The same shape functions can be utilized to compute the global coordinates of an arbitrary interior point as follows:

$$\begin{aligned} x &= N_1 x_1 + N_2 x_2 + N_3 x_3 = (x_1 - x_3)\xi + (x_2 - x_3)\eta + x_3 \\ y &= N_1 y_1 + N_2 y_2 + N_3 y_3 = (y_1 - y_3)\xi + (y_2 - y_3)\eta + y_3 \end{aligned} \tag{A.7}$$

where x_i, y_i - coordinates of i^{th} node of the element.

A.1.3. Strain-Displacement Relation

Given u and v as displacements in the x and y directions respectively, the strain can be written as follows:

$$\varepsilon = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_\tau \end{bmatrix} = \begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{bmatrix} \tag{A.8}$$

Let us use the chain rule and write derivatives of displacement u with respect to local coordinates as follows:

$$\begin{aligned} \frac{\partial u}{\partial \xi} &= \frac{\partial u}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial \xi} \\ \frac{\partial u}{\partial \eta} &= \frac{\partial u}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial \eta} \end{aligned} \tag{A.9}$$

In the matrix form it can be written as:

$$\begin{bmatrix} \dfrac{\partial u}{\partial \xi} \\ \dfrac{\partial u}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial u}{\partial y} \end{bmatrix} \tag{A.10}$$

or

$$\begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \end{bmatrix} \tag{A.11}$$

The above matrix is called Jacobian matrix. Its components can be defined from the equations (A.7):

$$\frac{\partial x}{\partial \xi} = x_1 - x_3 \tag{A.12}$$

$$\frac{\partial x}{\partial \eta} = x_2 - x_3 \tag{A.13}$$

$$\frac{\partial y}{\partial \xi} = y_1 - y_3 \tag{A.14}$$

$$\frac{\partial y}{\partial \eta} = y_2 - y_3 \tag{A.15}$$

Code implementation of the jacobian and jacobian invariant calculations are shown in A.2.

```
    def jacobi(self, inv=False):
        """calculate J to perform local to global coordinates transformation
        x,y: coordinates
        jacobi(inv=False): jacobian
        jacobi(inv=True): invariant of the jacobian
        """

        x, y = self.__endpts
        xc = np.zeros((3, 3))
        yc = np.zeros((3, 3))

        for i in range(3):
            for j in range(3):
                xc[i, j] = x[i] - x[j]
                yc[i, j] = y[i] - y[j]

        j = [[xc[0, 2], yc[0, 2]],
             [xc[1, 2], yc[1, 2]]]

        if inv:
            return np.linalg.inv(j)
        else:
            return j
```

Listing A.2: Jacobian and its invariant

For compaction we can write:

$$x_{ij} = x_i - x_j \tag{A.16}$$

$$y_{ij} = y_i - y_j \tag{A.17}$$

Equation (A.11) can be written:

$$\begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} = J^{-1} \begin{bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \end{bmatrix} \tag{A.18}$$

From the linear algebra we know that if

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{A.19}$$

then

$$A^{-1} = \frac{1}{\det A}\begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \tag{A.20}$$

Therefore (A.18) can be written as:

$$\begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} = \frac{1}{\det J}\begin{bmatrix} y_{23} & -y_{13} \\ -x_{23} & x_{13} \end{bmatrix}\begin{bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \end{bmatrix} \tag{A.21}$$

After multiplying the terms, the expression above will have the following form:

$$\frac{\partial u}{\partial x} = \frac{1}{\det J}(y_{23}(q_1 - q_5) - y_{13}(q_3 - q_5)) \tag{A.22}$$

$$\frac{\partial u}{\partial y} = \frac{1}{\det J}(-x_{23}(q_1 - q_5) + x_{13}(q_3 - q_5)) \tag{A.23}$$

Using a similar process, corresponding equations for v are derived resulting in:

$$\frac{\partial v}{\partial x} = \frac{1}{\det J}(y_{23}(q_2 - q_6) - y_{13}(q_4 - q_6)) \tag{A.24}$$

$$\frac{\partial v}{\partial y} = \frac{1}{\det J}(-x_{23}(q_2 - q_6) + x_{13}(q_4 - q_6)) \tag{A.25}$$

Finally the equation (A.8) can be written as follows:

$$\varepsilon = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_\tau \end{bmatrix} = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{bmatrix} = \frac{1}{\det J}\begin{bmatrix} y_{23}(q_1 - q_5) - y_{13}(q_3 - q_5) \\ -x_{23}(q_2 - q_6) + x_{13}(q_4 - q_6) \\ -x_{23}(q_1 - q_5) + x_{13}(q_3 - q_5) + y_{23}(q_2 - q_6) - y_{13}(q_4 - q_6) \end{bmatrix} \tag{A.26}$$

Which after simplifying can be written as:

$$\varepsilon = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_\tau \end{bmatrix} = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{bmatrix} = \frac{1}{\det J}\begin{bmatrix} y_{23}q_1 + y_{31}q_3 + y_{12}q_5 \\ x_{32}q_2 + x_{13}q_4 + x_{21}q_5 \\ x_{32}q_1 + y_{23}q_2 + x_{13}q_3 + y_{31}q_4 + x_{21}q_5 + y_{12}q_6 \end{bmatrix} \tag{A.27}$$

In matrix notation it will be:

$$\varepsilon = Bq \tag{A.28}$$

Where B is a matrix relating 3 strains over an element to 6 nodal displacements:

$$B = \frac{1}{\det J}\begin{bmatrix} y_{23} & 0 & y_{31} & 0 & y_{12} & 0 \\ 0 & x_{32} & 0 & x_{13} & 0 & x_{21} \\ x_{32} & y_{23} & x_{13} & y_{31} & x_{21} & y_{12} \end{bmatrix} \tag{A.29}$$

Code implementation of the element wise strain-displacement matrix is shown in A.3.

```
    def strain_disp_matrix(self, eltno):
        """
        Assemble strain-displacement matrix
        Nl: shape functions of the current element
        dNl: derivatives of the shape functions of the current element
        B: strain-displacement matrix
        """
        B = np.zeros((3, 2))
        for i in range(3):
            # access shape functions derivatives
            dNl = self.__elts[eltno].derivative(i)
            # transfer shape functions from local to global coordinates
            dN = np.dot(self.__elts[eltno].jacobi(inv=True), dNl)
            Bi = np.array([[dN[0], 0],
                           [0, dN[1]],
                           [dN[1], dN[0]]])
            # assemble strain-displacement matrix
            B = np.append(B, Bi, axis=1)
        B = np.delete(B, [0, 1], axis=1)
        return B
```

Listing A.3: Strain-displacement matrix

Code implementation of the element wise calculation of strains and stresses for every CST element is shown in A.4.

```
    # calculate strain at gaussian point within a given CST element
    def gauss_strain(self, u):
        """
        Element wise calculated strain.
        nele: total number of elements of the mesh grid
        """

        strain = np.zeros((3, self.__nele))
        for elt in self.__elts:
            # create an array of nodes indexes of the current element
            node = elt.nodes()
            # assemble strain-displacement matrix for the current element
            B = self.strain_disp_matrix(elt.eltno())
            # access nodal displacement values of the current element
            q = np.array([u[node[0] * 2], u[node[0] * 2 + 1],
                          u[node[1] * 2], u[node[1] * 2 + 1],
                          u[node[2] * 2], u[node[2] * 2 + 1], ])
            strain[:, [elt.eltno()]] = np.dot(B, q)

        return strain

    # calculate strain at Gaussian point within a given CST element
    def gauss_stress(self, straing):

    stressg = np.zeros((3, self.__nele))
    for elt in self.__elts:
        stressg[:, elt.eltno()] = np.dot(elt.el_tenz(), straing[:, elt.eltno()])

    return stressg
```

Listing A.4: Strain and stress calculation per element

A.1.4. Governing Equations

Substituting the above strain-displacement relation (A.28) into (2.6) then to (2.8) and subsequently to (2.2) the expression for strain energy can be written as:

$$U = \frac{1}{2}\int_V q^T B^T D(Bq - \varepsilon_{cr})dV \tag{A.30}$$

Replacing the integral over the volume V with integral over area A by introducing thickness t, it is possible to write the last expression for a single triangular element as:

$$U_e = \frac{1}{2}\int_e q^T B^T D(Bq - \varepsilon_{cr})t_e dA \tag{A.31}$$

It can be seen that all terms inside the integral in the equation above are constants, which gives the possibility to take them out:

$$U_e = \frac{1}{2} q^T B^T D (Bq - \varepsilon_{cr}) t_e \int_e dA \tag{A.32}$$

where $\int_e dA$ - simply the area of the triangular element:

$$U_e = \frac{1}{2} q^T B^T D (Bq - \varepsilon_{cr}) t_e A_e = \frac{1}{2} q^T B^T D B q t_e A_e - \frac{1}{2} q^T B^T D \varepsilon_{cr} t_e A_e \tag{A.33}$$

For the sake of compaction, let us introduce stiffness matrix k and creep forces vector f_{cr} as:

$$\begin{aligned} k_e &= t_e A_e B^T D B \\ f_{(cr)e} &= \frac{1}{2} t_e A_e B^T D \varepsilon_{cr} \end{aligned} \tag{A.34}$$

Now the expression for an arbitrary element's potential energy can be written as:

$$U_e = \frac{1}{2} q^T k_e q - q^T f_{(cr)e} \tag{A.35}$$

The total potential energy of the whole domain will be a summation of potential energies of the finite elements it consists of:

$$U = \sum_e \left(\frac{1}{2} q^T k_e q - q^T f_{(cr)e} \right) \tag{A.36}$$

or

$$U = \frac{1}{2} Q^T K Q - Q^T F_{cr} \tag{A.37}$$

where Q and K represent the global displacement vector and global stiffness matrix respectively.

By omitting the body forces, and converting all distributed external loads to nodal forces, the expression for the external work done by the external forces (2.3) can be written as follows:

$$W = Q^T F \tag{A.38}$$

The total potential energy:

$$\Pi = U - W = \frac{1}{2} Q^T K Q - Q^T F_{cr} - Q^T F \tag{A.39}$$

Applying the minimization of the potential energy principle $\delta \Pi = 0$ to (A.39) yields:

$$KQ = F + F_{cr}, \quad x \in \Omega \tag{A.40}$$

As it can be seen from the expression above, in absence of creep, i.e. $F_{cr} = 0$, the model simplifies to linear elastic model. Code routine for assembling the stiffness matrix, load vector and fictitious forces vector are shown in A.5, A.6 and A.7 respectively.

```
    def stiff_matrix(self, th=1):
        """
        assemble stiffness matrix
        nDofs: total number of degrees of freedom
        D: elasticity tenzor
        B: strain-displacement matrix
        ind: indexes of the degrees of freedom of the current element
        area: area of the current element
        ke: current element stiffness matrix
        ixgrid: indexes of the dofs of the current element stiffness matrix in the global
    stiffness matrix
        k: global stiffness matrix
        """
```

```
        nDofs = self.__nDOFs
        k = np.zeros((nDofs, nDofs))
        for elt in self.__elts:
            D = elt.el_tenz()
            B = self.strain_disp_matrix(elt.eltno())
            ind = elt.dofnos()
            area = elt.area()
            ke = th * area * np.dot(B.transpose(), (np.dot(D, B)))
            ixgrid = np.ix_(ind, ind)
            k[ixgrid] += ke
        return k
```

Listing A.5: Stiffness matrix

```
    def load_vector(self, p, temp, g, depth, th, et, i, sign, pressure, boundary):
        """
        assemble load vector
        x,y: nodal coordinates
        """
        x, y = self.__mesh.coordinates()

        if boundary == 'cavern':
            # d - vector of lengths between nodes on the cavern's wall
            d, alpha = self.nodal_forces()
            # indexes of the nodes of the cavern(s)
            nind_c, nind_c1, nind_c2 = self.cavern_nodes_ind()
            # depths of the roof of the cavern(s)
            d_cav_top = depth + max(y) - max(y[nind_c])
            d_cav_bot = depth + max(y) - min(y[nind_c])
            # minimum and maximum allowable cavern pressures
            pc_min = 0.2 * p * d_cav_bot
            pc_max = 0.8 * p * d_cav_top

            if pressure == 'max':
                pc = pc_max
            elif pressure == 'min':
                pc = pc_min

        # hydrogen density calculated using the coolprop library
        rho_h2 = PropsSI('D', 'T', temp, 'P', pc, 'hydrogen')

        # initialize the forces vector
        f = np.zeros((2 * self.__nnodes, 1))

        # assemble the forces vector
        for elt in self.__elts:
            node = elt.nodes()
            ind = elt.dofnos()
            fe = np.zeros(6)

            for i in range(3):
                # Applying Neumann's B.C. on cavern's wall
                if boundary == 'cavern':
                    if node[i] in nind_c:
                        # calculate the lithostatic and cavern pressure difference
                        dp = (pc + rho_h2 * g * (max(y[nind_c] - y[node[i]])) - p * (depth +
    max(y) - y[node[i]])) * \
                             d[np.where(nind_c == node[i])] * th
                        fe[2 * i] += dp * np.cos(alpha[np.where(nind_c == node[i])])
                        fe[2 * i + 1] += dp * np.sin(alpha[np.where(nind_c == node[i])])

            f[ind] = fe.reshape((6, 1))

        return f, sign
```

Listing A.6: Load vector

```
    def creep_load_vector(self, dt, a, n, q, r, temp, stress, strain_crg, arrhenius=None, th
    =1):
        """
        assemble creep load vector
        :return:
        """

        def deviatoric_stress():
            """
            calculate deviatoric stress
            """
            dstressx = stress[0] - 0.5 * (stress[0] + stress[1])
            dstressy = stress[1] - 0.5 * (stress[0] + stress[1])

            return np.array([dstressx, dstressy, stress[2]])

        def assemble_creep_forces_vector():
            """
            assemble the global fictitious creep forces vector
            """
            fcr = np.zeros((self.__nDOFs, 1))

            for elt in self.__elts:
                area = elt.area()
                ind = elt.dofnos()
                B = self.strain_disp_matrix(elt.eltno())
                D = elt.el_tenz()
                fcre = 1 / 2 * th * area * np.dot(np.transpose(B), np.dot(D, strain_crg[:,
    elt.eltno()]))
                fcr[ind] += fcre.reshape((6, 1))

            return fcr

        # deviatoric stress at gauss points
        dstressg = deviatoric_stress()
        # von mises stress at gauss points
        svmg = von_mises_stress(stress)
        # take into account Arrhenius term if required
        if arrhenius is not None:
            arr = np.exp(- q / (r * temp))
        else:
            arr = 1

        # calculate the creep strain
        g_crg = 3 / 2 * a * abs(np.power(svmg, n - 2)) * svmg * dstressg * arr
        strain_crg = strain_crg + g_crg * dt

        # assemble the global fictitious creep forces vector
        f_cr = assemble_creep_forces_vector()

        return f_cr, strain_crg
```

Listing A.7: Fictitious forces vector

A.1.5. Boundary Conditions

In order to make the described above problem well posed, it is necessary that the solution of (A.40) would satisfy boundary conditions:

$$\begin{aligned} q &= 0, \quad x \in \Gamma^h \\ \sigma \cdot n &= f, \quad x \in \Gamma^s \end{aligned} \tag{A.41}$$

where n is an outward unit normal vector to the surface Γ^s, f - applied surface traction on the boundary Γ^s.

The boundary condition on Γ^h is called the displacement or Dirichlet essential boundary condition, whereas the boundary condition on Γ^s is called traction or Neumann natural boundary condition. The latter one is already "naturally" introduced in the problem in the form of the vector of the external loads

F incorporated in the governing equation (A.40). The Dirichlet boundary condition is implemented by modifying the corresponding elements of the stiffness matrix K.

In the code implementation, first, indexes of the required degrees of freedom, where Dirichlet boundary conditions needs to be applied, are accessed as shown in A.8.

```
    def extract_bnd(self, lx=None, ly=None, rx=None, ry=None, tx=None, ty=None, bx=None, by=
    None):
        """
        Extracts indices of dof on the domain's boundary, such that L_bnd and R_bnd contain x
    -dof indices and B_bnd and T_bnd contain y-dof indices.
        d_bnd: array of dof indexes, where dbc will be implied
        x,y: nodal coordinates
        """

        l_bnd_x = np.array([], dtype='i')
        l_bnd_y = np.array([], dtype='i')
        r_bnd_x = np.array([], dtype='i')
        r_bnd_y = np.array([], dtype='i')
        b_bnd_x = np.array([], dtype='i')
        b_bnd_y = np.array([], dtype='i')
        t_bnd_x = np.array([], dtype='i')
        t_bnd_y = np.array([], dtype='i')
        d_bnd = np.array([], dtype='i')
        x, y = self.__nodes

        for i in range(self.__Nnodes):
            # dofs of the left edge
            if x[i] == np.min(x):
                l_bnd_x = np.append(l_bnd_x, i * 2)
                l_bnd_y = np.append(l_bnd_y, i * 2 + 1)
            # dofs of the bottom edge
            if y[i] == np.min(y):
                b_bnd_x = np.append(b_bnd_x, i * 2)
                b_bnd_y = np.append(b_bnd_y, i * 2 + 1)
            # dofs of the right edge
            if x[i] == np.max(x):
                r_bnd_x = np.append(r_bnd_x, i * 2)
                r_bnd_y = np.append(r_bnd_y, i * 2 + 1)
            # dofs of the top edge
            if y[i] == np.max(y):
                t_bnd_x = np.append(t_bnd_x, i * 2)
                t_bnd_y = np.append(t_bnd_y, i * 2 + 1)

        # after dofs of the left, right, bottom and top edges are accessed, the function will
     return only the array of the dofs of the edges, where dbc are chosen to be implemented
    by the user's choice.
        if not (lx == False):
            d_bnd = np.concatenate((d_bnd, l_bnd_x))
        if not (ly == False):
            d_bnd = np.concatenate((d_bnd, l_bnd_y))
        if not (rx == False):
            d_bnd = np.concatenate((d_bnd, r_bnd_x))
        if not (ry == False):
            d_bnd = np.concatenate((d_bnd, r_bnd_y))
        if not (bx == False):
            d_bnd = np.concatenate((d_bnd, b_bnd_x))
        if not (by == False):
            d_bnd = np.concatenate((d_bnd, b_bnd_y))
        if not (tx == False):
            d_bnd = np.concatenate((d_bnd, t_bnd_x))
        if not (ty == False):
            d_bnd = np.concatenate((d_bnd, t_bnd_y))

        return d_bnd
```

Listing A.8: Indexes of the degrees of freedom where DBC will be implemented

After the degrees of freedom subjected to Dirichlet boundary conditions are defined, the implementation is simply done using the code routine shown in A.9.

```
def impose_dirichlet(k, f, d_bnd):
    """Impose Dirichlet boundary conditions.
    k: stiffness matrix
    f: load vector
    """

    k[d_bnd, :] = 0
    k[:, d_bnd] = 0
    k[d_bnd, d_bnd] = 1
    f[d_bnd] = 0

    return k, f
```

Listing A.9: Implying DBC

A.1.6. Time Discretization

Let us write down the time dependent term (2.15) in the following form:

$$\frac{\partial \varepsilon_{cr}}{\partial t} = \frac{3}{2} e^{-\frac{Q}{RT}} a \sigma_{vM}^{n-1} s \tag{A.42}$$

or for simplicity one can write:

$$\begin{aligned} \frac{\partial \varepsilon_{cr}}{\partial t} &= f(T)g(\sigma) \\ \frac{\varepsilon_{cr}^{n+1} - \varepsilon_{cr}^{n}}{\Delta t} &= f(T)g(\sigma) \\ \varepsilon_{cr}^{n+1} &= \varepsilon_{cr}^{n} + f(T)g(\sigma)\Delta t \end{aligned} \tag{A.43}$$

where superscripts $(n+1)$ and n denote the value of property at the next and current time step respectively.

As it can be seen, in order to incorporate the creep strain in the mathematical model and solve the problem numerically, it is necessary to discretize the creep term in time domain. In the current study it is done in two ways: using the Euler forward time discretization scheme, i.e. explicit method and using the Euler backward time discretization scheme, i.e. implicit method. The first method yields the following expression of (A.43):

$$\varepsilon_{cr}^{n+1} = \varepsilon_{cr}^{n} + f(T)g(\sigma^{n})\Delta t \tag{A.44}$$

After substituting the obtained expression (A.44) into (A.40) the system of linear algebraic equations can be solved for every time step directly using the Gaussian elimination:

$$KQ^{n+1} = F + F_{cr}^{n} \tag{A.45}$$

The code routine for the mentioned above procedure with time dependent solution is shown in A.10.

```
    # check number of timesteps > 1
    if nt > 1:
        # iterate through each time step
        for i in tqdm(range(nt - 1)):
            # assemble the load vector
            fo, sign = V.load_vector()
            # assemble the fictitious creep forces vector
            f_cr, strain_crg = V.creep_load_vector()
            # extrapolate gauss points creep strain to nodal points
            strain_cr = V.nodal_extrapolation(strain_crg)
            f = fo + f_cr
            # imply DBC
            f[d_bnd] = 0
            # find solution for the displacement
            u = solve_disp(k, f)
            # evaluate gauss strains
```

```
            straing = V.gauss_strain(u)
            # interpolate gauss point strain values to nodal points
            strain = V.nodal_extrapolation(straing)
            if (i % freq == 0) and i != 0:
                # if cyclic load is defined, recalculate stresses every freq_th timestep,
                # freq=1, 2, 3... etc
                stressg = V.gauss_stress(straing - strain_crg)
                # interpolate gauss point stress values to nodal points
                stress = V.nodal_extrapolation(stressg)
```

Listing A.10: Creep time dependent model, explicit method

Using the implicit time discretizaton scheme, the following expression is obtained from (A.43):

$$\varepsilon_{cr}^{n+1} = \varepsilon_{cr}^{n} + f(T)g(\sigma^{n+1})\Delta t \tag{A.46}$$

There are now two unknown terms in the governing equation and it is not possible to solve it directly:

$$KQ^{n+1} = F + F_{cr}^{n+1} \tag{A.47}$$

In this case the Newton-Raphson iterative approach is proposed to solve the problem. The code routine for the approach is shown in A.11

```
    # set jacobian to be equal stiffness matrix
    J = k
    # check number of timesteps > 1
    if nt > 1:
        # iterate through each time step
        for i in tqdm(range(nt - 1)):
            # print('\nTime step {}, dt = {} s:'.format(i + 1, dt))
            converged = 0
            iter = 0
            # set maximum number of iterations
            max_iter = 5
            # set convergence criteria
            conv = 1e-4

            while converged == 0:
                # update mesh coordinates
                mesh.update_mesh(u)
                FunctionSpace(mesh, sfns, mu, kb)
                # assemble the fictitious creep forces vector
                f_cr, strain_crg = V.creep_load_vector()
                # extrapolate gauss points creep strain to nodal points
                strain_cr = V.nodal_extrapolation(strain_crg)
                f = fo + f_cr
                # imply DBC
                f[d_bnd] = 0
                # calculate residual
                residual = np.dot(k, u) - f
                # calculate displacement increment
                delta_u = - np.linalg.solve(J, residual)
                # update the displacement
                u = u + delta_u

                # evaluate gauss strains
                straing = V.gauss_strain(u)
                # interpolate gauss point strain values to nodal points
                strain = V.nodal_extrapolation(straing)
                if (i % freq == 0) and i != 0:
                    # if cyclic load is defined, recalculate stresses every freq_th timestep,
                    # freq=1, 2, 3... etc
                    stressg = V.gauss_stress(straing - strain_crg)
                    # interpolate gauss point stress values to nodal points
                    stress = V.nodal_extrapolation(stressg)

                # elapsed time
                et = np.append(et, et[-1] + dt)

                # check the residual
```

```
48                  residual = np.dot(k, u) - f
49                  res = np.linalg.norm(residual)
50                  iter += 1
51
52                  # print("\nIteration {}, norm(residual) = {}.".format(iter, res))
53                  # check the maximum iterations and convergence threshold criteria
54                  if iter == max_iter and res >= conv:
55                      print("\nMaximum iterations reached.")
56                  if res < conv or iter >= max_iter:
57                      converged = 1
```

Listing A.11: Creep time dependent model, implicit method

A.2. Figures

A.2.1. Simulation results: Consistency analysis

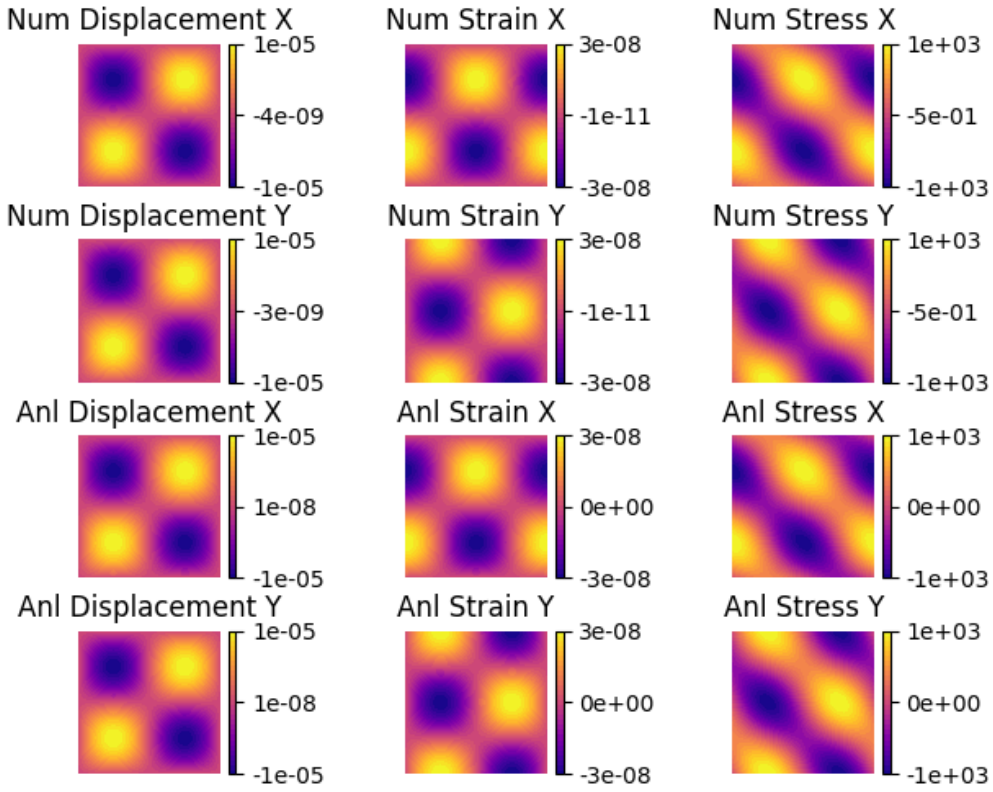


Figure A.3: Synthetic analytical and numerical solutions. 3584 elements.

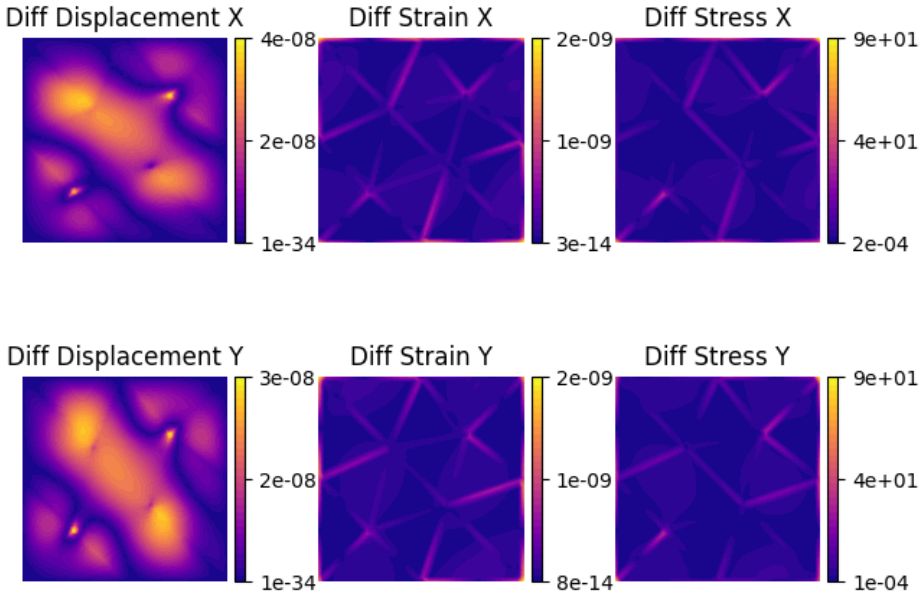
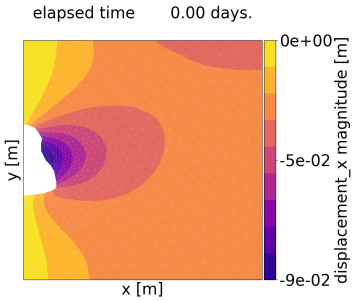
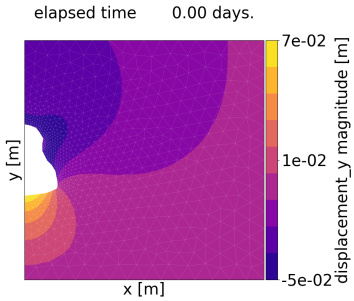


Figure A.4: Difference between synthetic analytical and numerical solutions. 3584 elements.

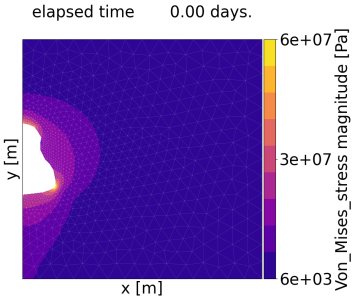
A.2.2. Simulation results: Irregular cavern



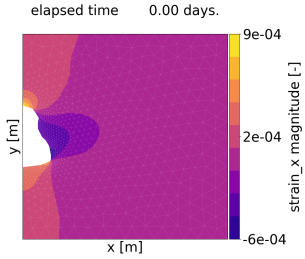
(a) Displacement in x direction.



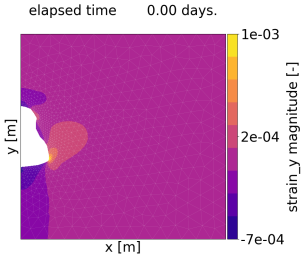
(b) Displacement in y direction.



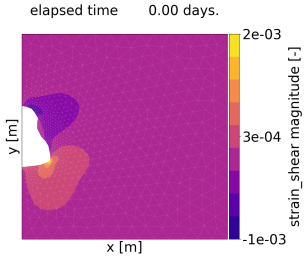
(c) Von Mises Stress.



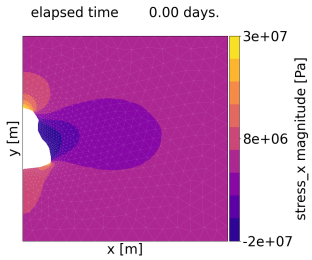
(d) X component of strain.



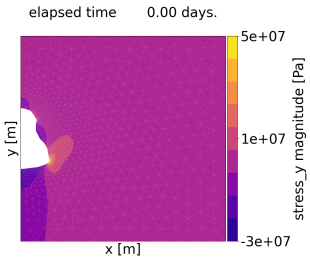
(e) Y component of strain.



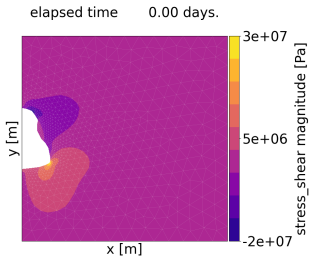
(f) Shear component of strain.



(g) X component of stress.



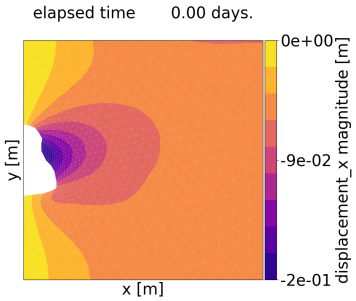
(h) Y component of stress.



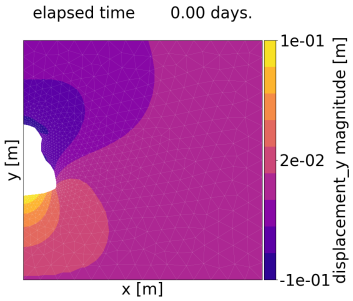
(i) Shear component of stress.

Figure A.5: Results of solving the irregular cavern model.

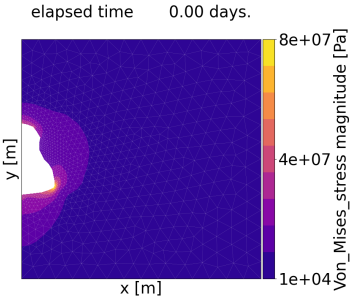
A.2.3. Simulation results: Irregular cavern with heterogeneity



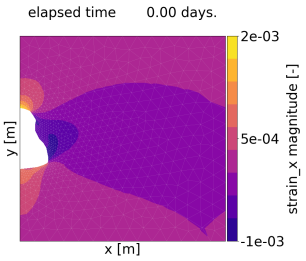
(a) Displacement in x direction.



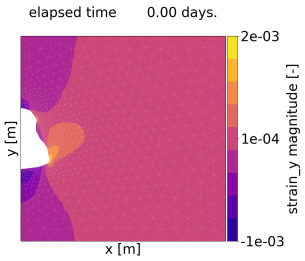
(b) Displacement in y direction.



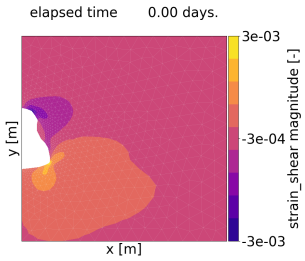
(c) Von Mises Stress.



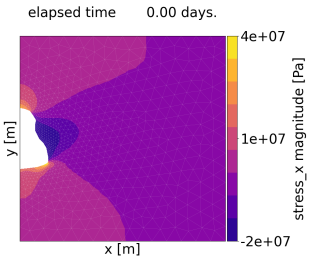
(d) X component of strain.



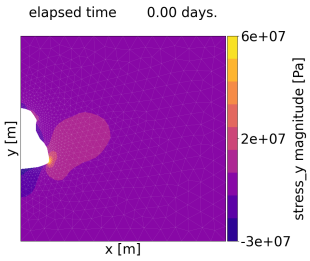
(e) Y component of strain.



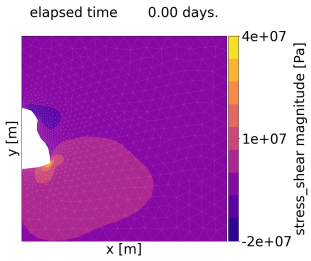
(f) Shear component of strain.



(g) X component of stress.



(h) Y component of stress.



(i) Shear component of stress.

Figure A.6: Results of solving the irregular cavern with heterogeneity model.

Bibliography

[1] P Bérest, J Béraud, M Bourcier, A Dimanov, H Gharbi, B Brouard, K DeVries, and D Tribout. Very slow creep tests on rock samples. In Mechanical Behaviour of Salt VII. CRC Press, mar 2012. doi: 10.1201/b12041-12.

[2] J. Betten. Creep mechanics. Springer, 2014.

[3] D. Bonté, J.-D. van Wees, and J.M. Verweij. Subsurface temperature of the onshore netherlands: new temperature dataset and modelling. Netherlands Journal of Geosciences - Geologie en Mijnbouw, 91(4):491–515, dec 2012. doi: 10.1017/s0016774600000354.

[4] N.L. Carter, S.T. Horseman, J.E. Russell, and J. Handin. Rheology of rocksalt. Journal of Structural Geology, 15(9-10):1257–1271, sep 1993. doi: 10.1016/0191-8141(93)90168-a.

[5] N. Castelletto, H. Hajibeygi, and H. A. Tchelepi. Multiscale finite-element method for linear elastic geomechanics. Journal of Computational Physics, 331:337–356, feb 2017. doi: 10.1016/j.jcp.2016.11.044.

[6] N. Castelletto, S. Klevtsov, H. Hajibeygi, and H. A. Tchelepi. Multiscale two-stage solver for biot's poroelasticity equations in subsurface media. Computational Geosciences, 23(2):207–224, 2019. doi: 10.1007/s10596-018-9791-z. URL `https://doi.org/10.1007/s10596-018-9791-z`.

[7] Z. Chemia and H. Koyi. The control of salt supply on entrainment of an anhydrite layer within a salt diapir. Journal of Structural Geology, 30(9):1192–1200, sep 2008. doi: 10.1016/j.jsg.2008.06.004.

[8] Z. Chemia, H. Koyi, and H. Schmeling. Numerical modelling of rise and fall of a dense layer in salt diapirs. Geophysical Journal International, 172(2):798–816, feb 2008. doi: 10.1111/j.1365-246x.2007.03661.x.

[9] Z. Chemia, H. Schmeling, and H. Koyi. The effect of the salt viscosity on future evolution of the gorleben salt diapir, germany. Tectonophysics, 473(3-4):446–456, aug 2009. doi: 10.1016/j.tecto.2009.03.027.

[10] R.D. Cook. Concepts and applications of finite element analysis. John Wiley Sons, 2002.

[11] N. Cristescu. General constitutive equation for transient and stationary creep of rock salt. International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts, 30(6):344, dec 1993. doi: 10.1016/0148-9062(93)91323-b.

[12] D.G. Caglayan et al. The impact of temporal complexity reduction on a 100% renewable european energy system with hydrogen infrastructure. oct 2019. doi: 10.20944/preprints201910.0150.v1.

[13] D.G. Caglayan et al. Technical potential of salt caverns for hydrogen storage in europe. International Journal of Hydrogen Energy, 45(11):6793–6805, feb 2020. doi: 10.1016/j.ijhydene.2019.12.161.

[14] W. Liang et al. Experiments on mechanical properties of salt rocks under cyclic loading. Journal of Rock Mechanics and Geotechnical Engineering, 4(1):54–61, apr 2012. doi: 10.3724/sp.j.1235.2012.00054.

[15] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering, 2009.

[16] R.-M. Günther and K. Salzer. A model for rock salt, describing transient, stationary, and accelerated creep and dilatancy. In The Mechanical Behavior of Salt – Understanding of THMC Processes in Salt, pages 109–117. CRC Press, dec 2017. doi: 10.1201/9781315106502-13.

[17] A. Hampel and O. Schulze. The composite dilatancy model: A constitutive model for the mechanical behavior of rock salt. In The Mechanical Behavior of Salt – Understanding of THMC Processes in Salt, pages 99–107. CRC Press, dec 2017. doi: 10.1201/9781315106502-12.

[18] J.H. Ter Heege, J.H.P. De Bresser, and C.J. Spiers. Dynamic recrystallization of wet synthetic polycrystalline halite: dependence of grain size distribution on flow stress, temperature and strain. Tectonophysics, 396(1-2):35–57, feb 2005. doi: 10.1016/j.tecto.2004.10.002.

[19] Z. Hou. Mechanical and hydraulic behavior of rock salt in the excavation disturbed zone around underground facilities. International Journal of Rock Mechanics and Mining Sciences, 40(5):725–738, jul 2003. doi: 10.1016/s1365-1609(03)00064-9.

[20] U. Hunsche and A. Hampel. Rock salt — the mechanical properties of the host rock material for a radioactive waste repository. Engineering Geology, 52(3-4):271–291, apr 1999. doi: 10.1016/s0013-7952(99)00011-3.

[21] M.L. Jeremic. Rock Mechanics in Salt Mining. 1994.

[22] L.M. Kachanov. The Theory of Creep. Nauka, Moscow, 1960.

[23] K. Khaledi, E. Mahmoudi, T. Schanz, and M. Datcheva. Finite element modeling of the behavior of salt caverns under cyclic loading. In Geomechanics from Micro to Macro, pages 945–950. CRC Press, aug 2014. doi: 10.1201/b17395-169.

[24] H. Koyi. Salt flow by aggrading and prograding overburdens. Geological Society, London, Special Publications, 100(1):243–258, 1996. doi: 10.1144/gsl.sp.1996.100.01.15.

[25] H. Koyi. The shaping of salt diapirs. Journal of Structural Geology, 20(4):321–338, apr 1998. doi: 10.1016/s0191-8141(97)00092-8.

[26] K. R. Kumar. Multi-scale nonlinear modeling of subsurface energy storage: cyclic loading with inelastic creep deformation. In 17th European Conference on the Mathematics of Oil Recovery, ECMOR 2020, volume 2020. European Association of Geoscientists and Engineers, EAGE, sep 2020.

[27] T. M. Letcher. Storing energy: with special reference to renewable energy sources. Elsevier, 2016.

[28] S. Li and J. L. Urai. Rheology of rock salt for salt tectonics modeling. Petroleum Science, 13(4):712–724, oct 2016. doi: 10.1007/s12182-016-0121-6.

[29] S. Li, L. Feng, P. Tang, G. Rao, and Y. Bao. Calculation of depth to detachment and its significance in the kuqa depression: A discussion. Petroleum Science, 6(1):17–20, feb 2009. doi: 10.1007/s12182-009-0003-2.

[30] S. Li, S. Abe, L. Reuning, S. Becker, J. L. Urai, and P. A. Kukla. Numerical modelling of the displacement and deformation of embedded rock bodies during salt tectonics: A case study from the south oman salt basin. Geological Society, London, Special Publications, 363(1):503–520, 2012. doi: 10.1144/sp363.24.

[31] K.-H. Lux and S. Eberth. Fundamentals and first application of a new healing model for rock salt. In The Mechanical Behavior of Salt – Understanding of THMC Processes in Salt, pages 129–138. CRC Press, dec 2017. doi: 10.1201/9781315106502-15.

[32] H. Ma, C. Yang, J. Liu, and J. Chen. The influence of cyclic loading on deformation of rock salt. In Rock Characterisation, Modelling and Engineering Design Methods, pages 63–68. CRC Press, may 2013. doi: 10.1201/b14917-10.

[33] G. Marketos, C. J. Spiers, and R. Govers. Impact of rock salt creep law choice on subsidence calculations for hydrocarbon reservoirs overlain by evaporite caprocks. Journal of Geophysical Research: Solid Earth, 121(6):4249–4267, jun 2016. doi: 10.1002/2016jb012892.

[34] S. Meer, C. J. Spiers, C. J. Peach, and T. Watanabe. Diffusive properties of fluid-filled grain boundaries measured electrically during active pressure solution. Earth and Planetary Science Letters, 200(1-2):147–157, jun 2002. doi: 10.1016/s0012-821x(02)00585-x.

[35] K. Naumenko and H. Altenbach. Modeling of Creep for Structural Analysis. Springer-Verlag GmbH, 2007. ISBN 3540708340.

[36] C. J. Peach, C. J. Spiers, and P. W. Trimby. Effect of confining pressure on dilatation, recrystallization, and flow of rock salt at 150c. Journal of Geophysical Research: Solid Earth, 106(B7): 13315–13328, jul 2001. doi: 10.1029/2000jb900300.

[37] A.N.B. Poliakov, Yu. Podladchikov, and C. Talbot. Initiation of salt diapirs with frictional overburdens: numerical experiments. Tectonophysics, 228(3-4):199–210, dec 1993. doi: 10.1016/0040-1951(93)90341-g.

[38] D. Schultz-Ela. Evolution of extensional fault systems linked with salt diapirism modeled with finite elements. AAPG Bulletin, 77, 1993. doi: 10.1306/d9cb6353-1715-11d7-8645000102c1865d.

[39] I. Sokolova, M. G. Bastisya, and H. Hajibeygi. Multiscale finite volume method for finite-volume-based simulation of poroelasticity. Journal of Computational Physics, 379:309–324, feb 2019. doi: 10.1016/j.jcp.2018.11.039.

[40] TNO. Salt extraction data sheets. Technical report, Apr 2012. URL `https://www.tno.nl/en/`.

[41] J.L. Urai and C.J. Spiers. The effect of grain boundary water on deformation mechanisms and rheology of rocksalt during long-term deformation. In The Mechanical Behavior of Salt – Understanding of THMC Processes in Salt, pages 149–158. CRC Press, dec 2017. doi: 10.1201/9781315106502-17.

[42] P.E. van Keken, C.J. Spiers, A.P. van den Berg, and E.J. Muyzert. The effective viscosity of rocksalt: implementation of steady-state creep laws in numerical models of salt diapirism. Tectonophysics, 225(4):457–476, oct 1993. doi: 10.1016/0040-1951(93)90310-g.