UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

NUMERICAL RECONSTRUCTION OF INVERSE PROBLEMS FOR PARTIAL DIFFERENTIAL EQUATIONS

TESIS PARA OPTAR AL GRADO DE
DOCTOR EN CIENCIAS DE LA INGENIERÍA, MENCIÓN MODELACIÓN MATEMÁTICA

JAVIER ANÍBAL RAMÍREZ GANGA

PROFESOR GUÍA:
JAIME H. ORTEGA PALMA

PROFESOR CO-GUÍA:
GINO MONTECINOS GUZMÁN

MIEMBROS DE LA COMISIÓN:
ERIC BONNETIER
SERGIO GAETE BECERRA
ALEJANDRO JOFRÉ CÁCERES
RODRIGO LECAROS LIRA

This work was partially supported by CODELCO, CODELCOTECH by Beca Doctorado PiensaMinería, CMM ANID PIA AFB170001, CMM IRL 2907-CNRS and CODELCO-El Teniente by project Desarrollo de herramientas algorítmicas para la estimación de peligro sísmico

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR AL GRADO DE
DOCTOR EN CIENCIAS DE LA INGENIERÍA, MENCIÓN MODELACIÓN MATEMÁTICA
POR: JAVIER ANÍBAL RAMÍREZ GANGA
FECHA: 2021
PROF. GUÍA: JAIME H. ORTEGA PALMA
PROF. CO-GUÍA: GINO MONTECINOS GUZMÁN

NUMERICAL RECONSTRUCTION OF INVERSE PROBLEMS FOR PARTIAL DIFFERENTIAL EQUATIONS

Esta tesis está dedicada al estudio de problemas inversos desde un enfoque numérico. Nos centraremos principalmente en problemas aplicados a ingeniería y ciencias, especificamente estudiaremos dos problemas que enunciaremos a continuación:

En la primera parte de la tesis estudiaremos el problema de extracción de mineral en minas. Para mina subterránea, la técnica habitual para la extracción de minerales es el método de block caving, que genera e induce actividad sísmica en la mina. Comprender el método de block caving es uno de los problemas más desafiantes en la minería subterránea. Este método se basa en la gravedad para romper y transportar grandes cantidades de mineral y desechos. El estado del arte de los modelos de daños no puede representar el efecto real de la minería en la masa rocosa, ya que, por ejemplo, el daño aparece en la parte inferior del dominio en consideración y con esto no es posible recuperar la subsicendia que se ve en la mina. En esta tesis estudiamos un nuevo enfoque que recupera los efectos que ocurren en la masa rocosa a medida que la actividad minera se desarrolla. Presentamos un modelo de daño extendido y mejorado que recupera el efecto del block caving basado en el modelo de gradiente de daño propuesto por Pham y Marigo en 2010, en particular, consideraremos una formulación variacional donde los modelos de gradiente de daño aparecen como un enfoque elíptico del problema variaconal de la mecánica de fractura y el criterio de daño depende de la parte esférica y deviatoria del tensor de esfuerzo. Mostramos pruebas y simulaciones numéricas interesantes que producen comportamientos de daño realistas en el dominio mejorando todos los enfoques de daño anteriores a este problema.

Para el segundo problema estudiaremos la reconstrucción numerica de las soluciones CGO para un sistema de conductividad. Estudiaremos metodos numéricos para el cálculo de las soluciones CGO para el sistema $\text{div}(\sigma \cdot \nabla U)$ en $\mathbb{R}^2$ para funciones matriciales σ simetricas y definidas positivas. La forma de calcularlas será mediate el uso de las soluciones del sistema de Beltrami. Primero probaremos la existencia de las soluciones CGO y luego usaremos una estrategia numérica basada en el método introducido por Huhtanen y Perämäki [28] para la ecuación the Beltrami. Consideraremos experimentos numéricos para mostrar la influencia de las ecuaciones acopladas.

ABSTRACT OF MEMORY TO OBTAIN THE DEGREE OF
DOCTOR EN CIENCIAS DE LA INGENIERÍA, MENCIÓN MODELACIÓN MATEMÁTICA
BY: JAVIER ANÍBAL RAMÍREZ GANGA
DATE: 2021
PROF. GUÍA: JAIME H. ORTEGA PALMA
PROF. CO-GUÍA: GINO MONTECINOS GUZMÁN

NUMERICAL RECONSTRUCTION OF INVERSE PROBLEMS FOR PARTIAL DIFFERENTIAL EQUATIONS

This thesis is devoted to the study of inverse problems in a numerical approach. We will focus on problems applied to engineering and science, specifically we will study two problems that we will state below:

In the first part of the thesis, we will study the problem of ore extraction in mines. For underground mine, the current usual technique for ore extraction is block caving, which generates and induces seismic activity in the mine. To understand the block caving method is one of the most challenging problems in underground mining. This method relies on gravity to break and transport enormous amounts of ore and waste. The state of the art in damage models is not able to represent the real effect of the mining in the rock mass since for example the damage appears in the bottom of the domain under consideration and with this is not possible to recover the subsidence seen in the mine. In this work we study the effects that occur on rock mass as mining activity develops. In this thesis we present an extended and improved damage model that recovers the effect the block caving based on the gradient damage model proposed by Pham and Marigo in 2010 we will consider a variational formulation where gradient damage models appear as an elliptical approach to the problem of variational fracture mechanics and the damage criterion depends on the spherical and deviatoric part of the stress tensor. We show interesting numerical tests and simulations producing realistic damage behaviors in the domain, improving all previous approaches to this issue.

In the second part of this thesis, we will study the numerical reconstruction of the CGO solutions for a conductivity system. We study numerical methods for computing numerically the CGO solutions to the conductivity system $\operatorname{div}(\sigma \cdot \nabla U) = 0$ in $\mathbb{R}^2$ for symmetric, positive definite matrix functions σ. The way to compute those solutions is to use solutions to the Beltrami system. In this work, we first prove the existence of CGO solution and then use a numerical strategy based on the method introduced by Huhntanem and Perämäki in [28] for the Beltrami equation. Numerical experiments are considered to show the influence of coupled equations.

Agradecimientos

El proyecto de tesis es el término de una etapa que comencé hace varios años, un período de crecimiento académico, profesional y personal lleno de éxitos, pero también de fracasos. Por lo mismo, quisiera agradecer a las personas que me brindaron un apoyo constante desde el comienzo.

Para comenzar, debo agradecer y reconocer a mis guías de tesis Jaime Ortega y Gino Montecinos por la gran colaboración, apoyo y cada uno de los aprendizajes. En conjunto, me ayudaron e incentivaron a desarrollar y mejorar mis competencias, conocimientos y habilidades necesarias para ser un aporte significativo en la investigación. Siempre me entregaron las herramientas necesarias y la orientación para resolver las dificultades presentadas en este camino. También, agradecer a otros grandes colaboradores Rodrigo Lecaros y Ariel Pérez, quienes han compartido otras experiencias y visiones para el desarrollo de este trabajo y otros futuros, y expandido el campo de estudio y prestado su colaboración, conocimientos y apoyo en todo momento.

De igual manera, agradezco a los incentivadores de la continuación de mis estudios y su financiamiento. Al CMM IRL 2907-CNRS por dar el pie inicial para el primer período como estudiante, y a CODELCO y COLDELCOTECH quienes me brindaron la oportunidad de estudiar problemas relacionados al rubro.

A mi familia, le agradezco por acompañarme todo el tiempo, brindarme contención y cariño. De manera especial, a Daniela que decidió estar conmigo en este camino y a Santiago que llegó para disfrutar juntos del éxito final, siendo siempre ambos un apoyo incondicional. A mis amigos, quienes confiaron en mí y me dieron ánimos para seguir arduamente en este trabajo.

Finalmente, a todas las personas que directa o indirectamente pueden haber participado en mi formación tanto profesional, académica o personal y en las decisiones que he emprendido en mi vida.

Contents

Part I

A shear-compression damage model for simulation of underground mining by block caving

Introduction

Block caving is a mining method in which ore blocks are undermined, causing the rocks to cave, and thus allowing broken ore to be removed at draw-points (see [24]). This method is based on the sinking principle and mineral breakage due to the removal of a large supporting area of rock and the subsequent extraction of this by mean of tunnels or collectors. The vacuum generated by the extraction of material from the basis is filled by the material falling by the action of gravity, which added to the process of attrition given by the friction during its falling defines the size of the mineral at the extraction point. Once extracted from this point, the material is transferred to a transfer stop in charge of directing it to the next process, either reduction or transport to the processing plant for further treatment. Figure 1 summarizes the Block Caving process where it is possible to see that over the blastholes, the ore blocks are broken allowing with this that the material to pass through of the drawpoints.

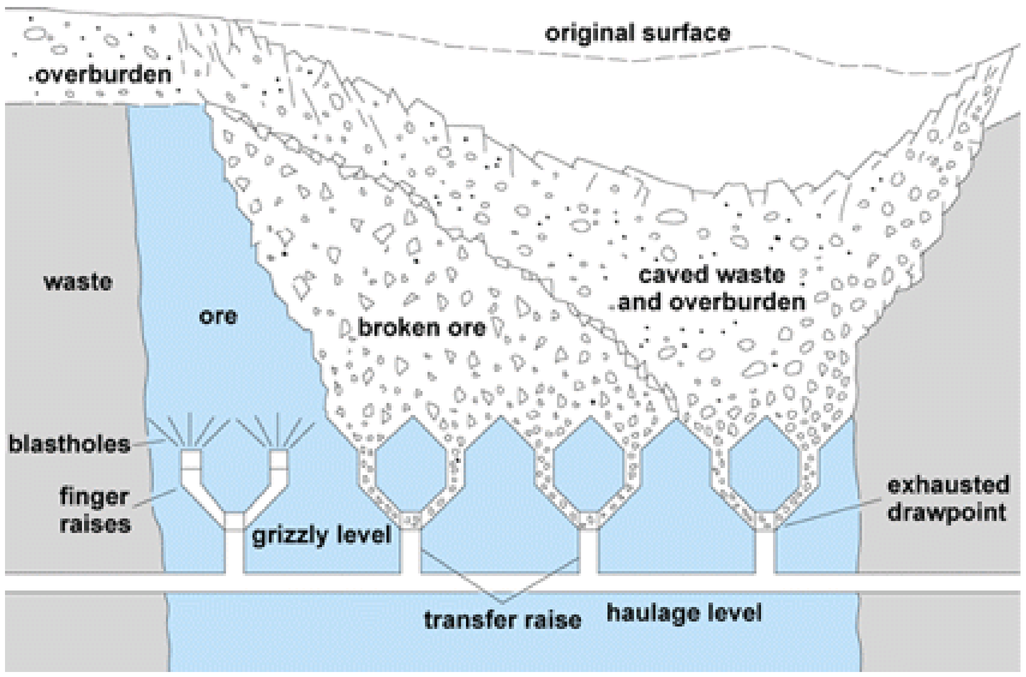


Figure 1: Block Caving process scheme.

In this work we study a mathematical model which describes the effect that occurs in the rock mass as mining activity developed by the block caving process, specifically, we seek to recover damage from underground mining in the rock mass, where the damage can be seen to appear above the cavity causing subsidence on this cavity [12]. A model of rock mechanics considering damage, in particular, we will consider a variational formulation of fracture mechanics.

The key issue of models predicting fracture is Griffith's criterion [22]. This criterion supposes that, as a crack grows, the displacement field is instantly in a new equilibrium, since the displacement may be discontinuous across the crack increment. The resulting decrease in stored elastic energy can then be balanced with the work required to create the crack increment, postulated to be proportional to the newly created area. The proportionality constant is usually known as fracture toughness. In other words, the rate of elastic energy decreases per unit area, the energy release rate, is proportional to the fracture toughness. Griffith's criterion stipulates that the crack grows only if the energy release rate is equal to the fracture toughness. Traditionally, these ideas could be formalized only for relatively simple crack typologies and often only for a pre-defined crack path. Only recently was the theory of brittle fracture freed from this restriction [1, 19]. Ambrosio and Braides [1] propose minimizing the sum of stored elastic energy and surface energy of discontinuity sets, to obtain displacements that are stable in the sense of Griffith. The first well-possed mathematical models of quasi-static fracture can be found in Francfort and Marigo [19], Dal Maso and Toader [15], and Francfort and Larsen[20]. In these references, the Dirichlet data u_D is varying in time and, at each time t, $u(t)$ is assumed to minimizes the potential energy subject to the appropiated boundary conditions, and subject to an irreversibility constraint on the crack set. In Giacomini [21], a discrete-time model of this type is proposed, based on the Ambrosio-Tortorelli approximation, which Γ-converges to the Griffith energy, see Ambrosio and Tortorelli [2]. The Ambrosio-Tortorelli approximation is particularly convenient for numerical implementation and was proposed by Bourdin, Francfort and Marigo [9] and Bourdin [10] for the simulation of the quasi-static model.

In this work, we consider the gradient damage approach for modeling fracture. In the pioneer work [46], the gradient damage approach has been used to model brittle fractures. In damage models, the failure is described by means of an internal variable, the so-called damage variable, which allows to modulate the stiffness of the material. For damage models, it is shown that the quasi-static evolution of a damaged body can be recast into a variational formulation [41] which consists of minimizing locally the total energy of the system under an irreversibility condition on the damage variable while enforcing an energy balance condition. For these models, the total energy can be read as a sum of an elastic energy and a dissipated energy.

Both gradient damage models and Griffith model of fracture relies have a variational structure and one can interpret gradient damage models as an elliptic approximation of the variational fracture mechanics problem. The variational approach of brittle fracture recasts the evolution problem for the cracked state body as a minimality principle for an energy functional sum of the elastic energy and the energy dissipated to create the crack [19]. Mathematical results based on Γ-Convergence theory show that when the internal length of gradient damage models tends to zero, the global minimum of the damage energy functional tends towards the global minima of the energy functional of Griffith brittle fracture [11].

The aim of this work is to provide a new damage model that reproduces the damage in the rock mass produced by the block caving process based on gradient damage models, that is, recover the damage above the cavity and with this obtain the subsidence. The main issue of the gradient damage models shown in [40] for underground mining is that, due of the gravity force, compression at the bottom of the domain has a great value, obtaining with this that the damage appears specifically in the bottom of the domain and then spreading throughout the domain and with this not showing the true behavior of the damage by the block caving process. For this, we will propose a new model in order to recover the true effect of the underground mining in the rock mass and the subsidence. the main idea is change the damage criterion, where, by following [35],the stress tensor is decomposed in its deviatoric and spherical part, in order to be able to control the contribution of the compression and shear forces in the damage criterion.

Through this work we will use the following summation convention on repeated indices: Vectors and second order tensors are indicated by a lowercase letter, such as u and σ for the displacement field and the stress field. Their components are denoted by u_{i} and $\sigma_{\mathrm{i}j}$. Third or fourth order tensors as well as their components are indicated by a capital letter, such as A or $A_{\mathrm{i}jkl}$ for the stiffness tensor. Such tensors are considered to be linear maps applying on vectors or second order tensors and the application is denoted without dots, like $A\varepsilon$ whose $\mathrm{i}j$-component is $A_{\mathrm{i}jkl}\varepsilon_{kl}$. The inner product between two vectors or two tensors of the same order is indicated by ":" which stands for $a_{\mathrm{i}}b_{\mathrm{i}}$ or $\sigma : \varepsilon$ for $\sigma_{\mathrm{i}j}\varepsilon_{\mathrm{i}j}$. We use the notation $A > 0$ to denote a positive definite tensor.

Chapter 1

Background and state of the art

In this chapter we will introduce the main concepts that will be used as the basis of our damage model for underground mining. In a first instance, we will introduce the variational fracture model with its respective elliptical regularization and then we will deliver the main characteristics of the gradient damage models.

1.1 Variational approach to fracture

The objective of the variational approach to fracture is to settle down a complete and unified brittle fracture theory within the Continuum Mechanics framework, which is capable of predicting the onset and the space-time evolution of the sharp-interface cracks with possible complex typologies where the previous sharp-interface fracture theories fail to deliver.

1.1.1 Brittle fracture as an energy minimization problem

Let us consider a body $\Omega \subseteq \mathbb{R}^n$, $1 \leq n \leq 3$, with a prescribed displacement load U imposed on a part of the boundary, denoted by $\partial\Omega_U$. As such Ω represents the crack-free reference configuration of an elastic body.

Throughout the work, fracture are studied within the brittle fracture theory. The fracture is represented by a family of cracks which correspond to lines in 2D and surfaces in 3D. Let Γ be the set of the cracked points within the body, where the displacement field u may be discontinuous. Here, we consider the variational approach to fracture mechanics proposed by Francfort and Marigo [19] who have introduced the following energy functional for the cracked body

$$\mathcal{P}(u,\Gamma) = \mathcal{E}(u,\Gamma) + \mathcal{S}(\Gamma) = \int_{\Omega\backslash\Gamma} \psi(\varepsilon(u))\mathrm{d}x + G_c\mathcal{H}^{n-1}(\Gamma), \tag{1.1}$$

where ψ is the elastic energy density function of the linearized strain $\varepsilon(u)$, symmetric part of the gradient of u defined by $\varepsilon(u) = \frac{\nabla u + \nabla u^T}{2}$, G_c is the fracture toughness, i.e. the energy required to create a crack of unit surface in the body Ω, and $\mathcal{H}^{n-1}$ is the Hausdorff surface measure giving the crack length, for $n = 2$ or surface, for $n = 3$. In the functional (1.1),

$\mathcal{E}(u,\Gamma)$ is the elastic energy stored in the cracked body, and $\mathcal{S}(\Gamma)$ is the energy required to create the crack according to the Griffith model [22]. For linear elastic bodies $\psi(\varepsilon(u)) = \frac{1}{2}A\varepsilon(u) : \varepsilon(u)$, where A is the fourth order elastic stiffness tensor, with

$$A(x) \text{ symmetric and such that } \alpha I \leq A(x) \leq \beta I \text{ for a.e. } x \in \Omega, \alpha, \beta > 0. \tag{1.2}$$

Remark 1.1 *The energies $\mathcal{E}(u,\Gamma)$ and $\mathcal{S}(\Gamma)$ satisfy the following elementary properties:*

1. *$\mathcal{E}(u,\Gamma)$ is monotonically decreasing in Γ for any fixed u.*
2. *$\mathcal{S}(\Gamma)$ is strictly monotically increasing in Γ.*

The variational approach to fracture sees the crack evolution as a minimization movement of the total energy under an irreversibility condition to prevent self-healing of cracks. To this effect a time-parameterized loading $U(t)$ is applied to $\partial\Omega_U$. Assume that a initial crack Γ_0 is represent in the body at the onset of the loading process. The main goal is to determine the evolution of the crack (or cracks) during the loading, i.e., to obtain the time-parameterized mapping $\Gamma(t)$. The basic idea is as follow. At a given time t, and for the corresponding loading $U(t)$ the crack $\Gamma(t)$ will be the closed subset of $\overline{\Omega}$ which minimizes $\mathcal{P}(u(t),\Gamma)$ among all cracks Γ which contain all previous $\Gamma(s)$, $s < t$.

The Francfort-Marigo model formulates the quasi-static time evolution of the displacement field u and the crack set Γ as a minimization problem in (1.1). In view of the numerical applications, we focus here on the time-discrete case, with $N+1$ time steps $\{t_0 = 0, \cdots, t_{\mathrm{i}}, \cdots, t_N = T\}$. For this, we assume that Γ_0 is given, let $U_{t_{\mathrm{i}}}$ be a sequence of loadings. Then the corresponding cracks Γ_{i} have to satisfy

$$\Gamma_{\mathrm{i}} \supseteq \Gamma_{\mathrm{i}-1}, \mathcal{E}(u,\Gamma_{\mathrm{i}}) \leq \mathcal{E}(u,\Gamma) \text{ for every } \Gamma \supseteq \Gamma_{\mathrm{i}-1}. \tag{1.3}$$

As such, the evolution is discretization-dependent. The real evolution should be construed as a limit of the discrete evolution as the time-step tends to zero.

On the basis of the knowledge of the craked at the time instant $t_{\mathrm{i}-1}$, the craked at the time step t_{i} is obtained as the solution of the following minimization problem

$$\inf\left\{\mathcal{P}(u,\Gamma) : u \in \mathcal{C}_{t_{\mathrm{i}}}(\Gamma), \Gamma \supseteq \Gamma_{\mathrm{i}-1}\right\}, \tag{1.4}$$

where the space of admissible displacements at time t is defined by

$$\mathcal{C}_t(\Gamma) := \left\{u \in H^1(\Omega \setminus \Gamma) : u = U_t \text{ on } \partial\Omega_U\right\}, \tag{1.5}$$

and the admissible crack sets have to satisfy the irreversibility condition $\Gamma \supseteq \Gamma_{\mathrm{i}-1}$. The irreversibility condition is fundamental to prevent the unphysical healing of the crack set $\Gamma_{\mathrm{i}-1}$ at the previous time step.

1.1.2 Elliptic regularization

The variational approach to fracture can be regarded as a Free Discontinuity Problem where the unknown crack set introduces displacement discontinuity somewhere in the body, see [11] for a mathematical treatment of this topic. The minimization problem for the quasi-static crack evolution as formulated in (1.4) is not prone to an immediate numerical implementation, because, for example, there is a dependency between the displacement Γ and the space of admissible displacement where u is defined. To tackle it numerically with a standard finite element discretization Bourdin et al. [9] resorts to regularization strategy proposed by Ambrosio and Tortorelli in [2] for solving similar free-discontinuity problems encountered in image segmentation [44]. To approximate the solution of the minimization problem (1.4), by Bourdin et al. [9], we consider the family of elliptic functionals defined by

$$\mathcal{P}_\ell(u,\alpha) = \mathcal{E}_\ell(u,\alpha) + G_c\mathcal{S}_\ell(\alpha), \tag{1.6}$$

with

$$\mathcal{E}_\ell(u,\alpha) = \int_\Omega \frac{1}{2}\left((1-\alpha)^2 + k_\ell\right)\psi(\varepsilon(u))\mathrm{d}x, \quad \mathcal{S}_\ell(\alpha) = \int_\Omega \left(\frac{\alpha^2}{4\ell} + \ell\nabla\alpha\cdot\nabla\alpha\right)\mathrm{d}x, \tag{1.7}$$

where ℓ and k_ℓ are positive scalar parameters, α is an additional scalar field with values in $[0,1]$.

The associated regularized version of the minimality principle (1.4) for the time discrete quasi-static evolution between the time-step $t_{\mathrm{i}-1}$ and t_i read as

$$\inf\left\{\mathcal{P}_\ell(u,\alpha)\,:\,u\in\mathcal{C}_{t_\mathrm{i}},\alpha\in\mathcal{D}_\mathrm{i}\right\}, \tag{1.8}$$

where the spaces of admissible state fields at the step i are

$$\mathcal{C}_{t_\mathrm{i}} := \left\{u\in H^1(\Omega)\,:\,u = U_{t_\mathrm{i}} \text{ on } \partial\Omega_U\right\}, \quad \mathcal{D}_\mathrm{i} := \left\{\alpha\in H^1(\Omega)\,:\,\alpha_{\mathrm{i}-1}\le\alpha\le 1\right\}. \tag{1.9}$$

The condition $\alpha\ge\alpha_{\mathrm{i}-1}$, where $\alpha_{\mathrm{i}-1}$ is the damage in the time-step $t_{\mathrm{i}-1}$, is the regularized version of the irreversibility condition $\Gamma\supseteq\Gamma_{\mathrm{i}-1}$ in (1.4). The strategy proposed in [9] avoids the dependency between the fracture and the space of admissible displacement.

Γ-convergence theorems [11] prove that, for $\ell\to 0^+$ and $0 < k_\ell \ll \ell$, the sequence $(\overline{u}_\ell,\overline{\alpha}_\ell)$, obtained as the global minimum of (1.6) for a fixed ℓ, converges, in a specific weak sense, to the global minimum of (1.1). Moreover, in [21], Giacomini proves that the time-discrete quasi-discrete evolution obtained by minimizing the regularized functional under the irreversibility condition on α converges to the quasi-static evolution of the brittle fracture model of Francfort and Marigo in [19].

1.2 Gradient damage models to approximate brittle fracture

Damage theory aims at modeling progressive degradation and failure in engineering materials such as metal, concrete, or rocks. Damage localization may be interpreted as a regularized description of cracks, that is surfaces of discontinuities of the displacement field. Its use as a genuine physical model for brittle fracture starts from the pioneering work in [46], where the properties and behaviors of the model are analyzed with respect to its aptitude to approximate fracture phenomena.

1.2.1 Variational formulation

The current formulation of the Gradient Damage Model in the evolution framework is achieved in [47, 48]. We refer the readers to [40] and references therein for a thorough review of its variational and constitutive ingredients as well its properties especially when applied to brittle fracture. This model is based on the variational approach to fracture shown in the previous chapter, where also, the surface and body forces are added into the energy functional.

Let us consider a homogeneous n-dimensional body whose reference configuration is the open connected bounded set $\Omega \subseteq \mathbb{R}^n$ and we assume that the local elastic material behavior can be characterized by the Young's modulus E and the Poisson's ratio ν. The scalar field α introduced in the elliptic regularization shown in Section 1.1.2 is now interpreted as a damage variable growing from 0 to 1, $\alpha = 0$ being the undamaged state and $\alpha = 1$ being the fully damage state.

The elastic-damage evolution is governed by several physics principles based on the definitions of a potential energy $\mathcal{P}_t(u, \alpha)$ of the body Ω. Adopting the notation used in [40] of gradient damage models and following Section 1.1.1, the potential energy $\mathcal{P}_t(u, \alpha)$ reads

$$\mathcal{P}_t(u, \alpha) = \mathcal{E}(u, \alpha) + \mathcal{S}(\alpha) - \mathcal{W}_t(u). \tag{1.10}$$

In this potential energy, the elastic energy that characterizes the elastic behavior of the material is given by

$$\mathcal{E}(u, \alpha) = \int_\Omega \psi(\varepsilon(u), \alpha)\mathrm{d}x = \int_\Omega \frac{1}{2} A(\alpha)\varepsilon(u) : \varepsilon(u)\mathrm{d}x, \tag{1.11}$$

where $A(\alpha)$ is the Hooke's elasticity tensor at a given damage state defined by

$$A(\alpha) = \mathrm{a}(\alpha) A_0, \tag{1.12}$$

with $\alpha \mapsto \mathrm{a}(\alpha)$ an adimensional real function of damage characterizing stiffness degradation from an initial undamaged state $A_0 = A(0)$. Thus, the damage-dependent stress tensor conjugates to the strain variable is given by

$$\sigma = \sigma(u, \alpha) = A(\alpha)\varepsilon(u). \tag{1.13}$$

The damage dissipation energy, that quantifies the amount of energy consumed in a damage process, is defined by

$$\mathcal{S}(\alpha) = \int_\Omega \left(w(\alpha) + \frac{1}{2} w_1 \ell^2 \nabla\alpha \cdot \nabla\alpha \right) \mathrm{d}x, \tag{1.14}$$

where $\alpha \mapsto w(\alpha)$ describes local damage dissipation during a homogeneous damage evolution and its maximal value $w(1) = w_1$, with $0 < w_1 < \infty$, is the energy completely dissipated during such process when damage attains 1.

To define the loading condition $\mathcal{W}_t(u)$ and admissible function spaces. We assume that the body Ω is submitted to a time dependent loading U_t, F_t and f_t, which consists of a imposed displacement on the boundary part $\partial\Omega_U$, the surface forces on the complementary part $\partial\Omega_F$ and the volume forces over Ω, t denoting the time parameter respectively. The potential of the given external forces at time t can read as the following linear form $\mathcal{W}_t$ defined on the set $\mathcal{C}_t$ of kinematically admissible displacement fields

$$\mathcal{W}_t(v) := \int_\Omega f_t \cdot v \mathrm{d}x + \int_{\partial\Omega_F} F_t \cdot v \mathrm{d}s, \tag{1.15}$$

with

$$\mathcal{C}_t := \{v : v = U_t \text{ on } \partial\Omega_U\}. \tag{1.16}$$

The law of evolution of the damage in the body is written in a variational form, that is, if (u, α) denotes a pair of admissible displacement and damage fields at time t, i.e., if $u \in \mathcal{C}_t$ and $\alpha \in \mathcal{D}$ with

$$\mathcal{D} := \{\beta : 0 \leq \beta \leq 1 \text{ in } \Omega\}, \tag{1.17}$$

then the evolution problem consists of finding, for every $t \geq 0$, $(u_t, \alpha_t) \in \mathcal{C}_t \times \mathcal{D}$ such that the following conditions hold (see [47, 48])

1. **Irreversibility:** $t \mapsto \alpha_t$ must be non decreasing and, at each time $t \geq 0$, $\alpha_t \in \mathcal{D}$.
2. **Stability:** At each time $t > 0$, the state (u_t, α_t) must be stable in the sense that for all $v \in \mathcal{C}_t$ and all $\beta \in \mathcal{D}$ such that $\beta \geq \alpha_t$, there exists $\overline{h} > 0$ such that for all $h \in [0, \overline{h}]$
$$\mathcal{P}_t(u_t, \alpha_t) \leq \mathcal{P}_t(u_t + h(v - u_t), \alpha_t + h(\beta - \alpha_t)). \tag{1.18}$$
3. **Energy balance:** At each time $t > 0$, the following energy balance must be hold:
$$\mathcal{P}_t(u_t, \alpha_t) = \mathcal{P}_0(u_0, \alpha_0) + \int_0^t \left(\int_\Omega \sigma_s : \varepsilon(\dot{U}_s)\mathrm{d}x - \mathcal{W}_s(\dot{U}_s) - \dot{\mathcal{W}}_s(u_s) \right) \mathrm{d}S \tag{1.19}$$

In (1.19), α_0 denotes the given damage state at the beginning of the loading process whereas u_0 is the associated displacement field obtained by solving the elastostatic problem at time 0. σ_s denotes the real stress field at time s, $\dot{U}_s$ is the rate of a given admissible displacement field at time s and $\dot{\mathcal{W}}_s$ denotes the linear form associated with the rate of the prescribed volume or surface forces at time s. The energy balance condition (1.19) characterizes the energy flow in the system: The loading condition is balanced by the mechanical energy variation of the system consisting of the elastic energy (1.11) and the energy dissipated in the process of damage production (1.14).

The necessary conditions that a solution must satisfy are obtained by follow: Dividing the stability conditions by $h > 0$ and passing to the limit when $h \to 0$, one obtains the first order conditions that (u_t, α_t) must satisfy at time t:

$$\mathcal{P}'_t(u_t, \alpha_t)(v - u_t, \beta - \alpha_t) \geq 0, \forall (v, \beta) \in \mathcal{C}_t \times \mathcal{D}(\alpha_t), \tag{1.20}$$

where $\mathcal{D}(\alpha_t) = \{\beta : \alpha_t \leq \beta \leq 1 \text{ in } \Omega\}$ and $\mathcal{P}'_t(u_t, \alpha_t)(v, \beta)$ denotes the directional derivative of $\mathcal{P}_t$ at (u_t, α_t) in the direction (v, β), i.e. the linear form defined by

$$\begin{aligned}\mathcal{P}'_t(u_t, \alpha_t)(v, \beta) = &\int_\Omega \sigma_t : \varepsilon(v)\mathrm{d}x \\ &+ \int_\Omega \left(\left(\frac{1}{2} \mathrm{a}'(\alpha_t) A_0 \varepsilon(u_t) : \varepsilon(u_t) + w'(\alpha_t) \right) \beta + w_1 \ell^2 \nabla\alpha_t \cdot \nabla\beta \right) \mathrm{d}x - \mathcal{W}_t(v),\end{aligned} \tag{1.21}$$

where, $\sigma_t = \sigma(u_t, \alpha_t)$ is the stress tensor at time t. From this global variational formulation, we can deduce the standard local formulation of the damage model by integration by parts and classical localization arguments.

The equilibrium equation for the stress σ_t is obtained by testing the variational inequality (1.20) for $\beta = \alpha_t$ and $v \in \mathcal{C}_t$. This gives the well known equilibrium equation

$$\begin{aligned} \operatorname{div}(\sigma_t) + f_t &= 0 \quad \text{in} \quad \Omega, \\ \sigma_t \cdot n &= F_t \quad \text{on} \quad \partial\Omega_F, \\ u &= U_t \quad \text{on} \quad \partial\Omega_U. \end{aligned} \tag{1.22}$$

The damage problem is obtained by testing (1.20) for arbitrary β in the convex cone $\mathcal{D}(\alpha_t)$ with $v = u_t$. That leads to the variational inequality governing the evolution of the damage, for all $\beta \in \mathcal{D}(\alpha_t)$

$$\int_\Omega \left(\left(\frac{1}{2} \mathrm{a}'(\alpha_t) A_0 \varepsilon(u_t) : \varepsilon(u_t) + w'(\alpha_t) \right) (\beta - \alpha_t) + w_1 \ell^2 \nabla\alpha_t \cdot \nabla(\beta - \alpha_t) \right) \mathrm{d}x \geq 0. \tag{1.23}$$

After integration by parts and using classical tools of calculus of variations, we find the strong formulation for the damage evolution problem in the form of the Kuhn-Tucker conditions for unilateral constrained variational problems

1. **Irreversibility:** $\dot{\alpha}_t \geq 0$ in Ω.
2. **Damage criterion:** $\frac{1}{2}\mathrm{a}'(\alpha_t) A_0 \varepsilon(u_t) : \varepsilon(u_t) + w'(\alpha_t) - w_1 \ell^2 \Delta\alpha_t \geq 0$ in Ω.
3. **Energy balance:** $\dot{\alpha}_t \left(\frac{1}{2}\mathrm{a}'(\alpha_t) A_0 \varepsilon(u_t) : \varepsilon(u_t) + w'(\alpha_t) - w_1 \ell^2 \Delta\alpha_t \right) = 0$ in Ω.
4. **Boundary Conditions:** $\frac{\partial \alpha_t}{\partial n} \geq 0$ and $\dot{\alpha}_t \frac{\partial \alpha_t}{\partial n} = 0$ on $\partial\Omega$.

The energy balance states that, at each point, the damage can increase only if the damage yield criterion is attained, that is if the damage criterion is an equality.

The equilibrium and damage problem are implicitly linked and must be satisfied simultaneously to get a solution of the evolution problem (1.20).

1.2.2 Approximation of variational brittle fracture

To show how such gradient damage models are suitable for the simulation of brittle fracture mechanics, let us first rewrite the total energy of the system (1.10) for a body Ω, assuming no external body and surface forces for brevity

$$\mathcal{P}(u, \alpha) = \int_\Omega \frac{1}{2} A(\alpha) \varepsilon(u) : \varepsilon(u) \mathrm{d}x + \frac{G_c}{c_w} \int_\Omega \left(\frac{1}{\tilde{\ell}} \frac{w(\alpha)}{w_1} + \tilde{\ell} \nabla\alpha \cdot \nabla\alpha \right) \mathrm{d}x, \tag{1.24}$$

where the fracture toughness energy G_c is defined by

$$G_c = 2\ell \int_0^1 \sqrt{2 w_1 w(\beta)} \mathrm{d}\beta = c_w \frac{\ell w_1}{\sqrt{2}}, \tag{1.25}$$

with $c_w = 4 \int_0^1 \sqrt{\dfrac{w(\beta)}{w_1}} \mathrm{d}\beta$ and $\tilde{\ell} = \dfrac{\ell}{\sqrt{2}}$.

In fracture mechanics approaches, material failure is modeled by nucleation and propagation of surfaces of discontinuity of the displacement field. Following Chapter , the quasi-static problem of fracture mechanics requires to determine the evolution of the displacement u and the crack set Γ as a function of the loading.

The fuctional (1.1) has a close analogy with the damage energy functional in the form (1.24). Both of them are the sum of an elastic energy term and a dissipated energy term, the dissipated energy being a volume integral for the damage model and a surface integral for the fracture model. Indeed, the damage functional is a regularized version of the Griffith functional. Γ -convergence results [11] show that, under some constitutive requirements, the global minimum of the damage functional (1.24) converges toward the global minimum of the Griffith functional (1.1) when the internal length ℓ goes to zero.

1.2.3 Gradient damage model for shear fracture

For case where we are interested in predicts asymmetric results in traction and in compression, the gradient damage model, shown in Section 1.2, is not able to recover this kind of fractures. An interesting idea is to consider a new variational approach to show how it might be altered tto incorporate the idea od less brittle, "deviatoric-type fracture" and apply to materials such as confine stone in underground mining.

Lancioni and Royer-Carfagni proposed in [35] a formulation reproducing the shear fracture model. They modified the energy functional (1.10) to obtain a gradient damage model developing shear bands with localized damage approximating cracks. The approach is based on the orthogonal decomposition of the linearized strain tensor in its spherical and deviatoric components given by ε^s and ε^{d}, respectively, that is:

$$\varepsilon = \varepsilon^s + \varepsilon^{\mathrm{d}}, \quad \varepsilon^s = \tfrac{1}{n}\mathrm{tr}(\varepsilon)I, \quad \varepsilon^{\mathrm{d}} = \varepsilon - \varepsilon^s, \tag{1.26}$$

where I denotes the n-dimensional identity tensor. With this decomposition, the elastic energy density function may be written as the sum of the spherical and deviatoric contribution

$$\psi(\varepsilon) = \frac{1}{2}\lambda \mathrm{tr}(\varepsilon)^2 + \mu\varepsilon : \varepsilon = k_0\frac{\mathrm{tr}(\varepsilon)^2}{2} + \mu\varepsilon^{\mathrm{d}} : \varepsilon^{\mathrm{d}}, \tag{1.27}$$

where λ and μ are the Lamé coefficients and $k_0 = \lambda + \frac{2\mu}{n}$ is the bulk modulus of the material. The formulation for shear fracture replaces the functional $\mathcal{P}_t(u,\alpha)$ of the variational statement (1.10) by

$$\tilde{\mathcal{P}}_t(u,\alpha) = \tilde{\mathcal{E}}(u,\alpha) + \mathcal{S}(\alpha) - \mathcal{W}_t(u), \tag{1.28}$$

with

$$\tilde{\mathcal{E}}(u,\alpha) = \left(k_0\frac{\mathrm{tr}(\varepsilon)^2}{2} + (\mathrm{a}(\alpha) + k_\ell)\mu\varepsilon^{\mathrm{d}} : \varepsilon^{\mathrm{d}}\right), \tag{1.29}$$

where the spherical part of the elastic energy remains unaffected by the value of the scalar field α. The modified functional implies that the creation of the surface energy may be compensated exclusively by a reduction of the deviatoric elastic energy.

1.3 Generalized standard materials

In this section, we cast the damage models we studied in the framework of generalized standard materials [23]. In an isothermal process, the Clausius-Duhem inequality that expresses the second law of thermodynamics takes the form

$$\Phi = \sigma : \dot{\varepsilon} - \dot{W} \geq 0, \tag{1.30}$$

where Φ denotes the density of dissipated power, W the free energy, σ the stress tensor and $\dot{\varepsilon}$ the strain rate tensor, defined by

$$\dot{\varepsilon} = \varepsilon(\dot{u}) = \frac{1}{2}\left(\nabla \dot{u} + \nabla \dot{u}^T\right). \tag{1.31}$$

In our case, the free energy is assumed to depend on the strain and on an internal variable α that measures damage, (1.30) can be rewritten in the form

$$\left(\sigma - \frac{\partial W}{\partial \varepsilon}\right) : \dot{\varepsilon} - \frac{\partial W}{\partial \alpha} : \dot{\alpha} \geq 0. \tag{1.32}$$

In the theory of generalized standard materials, it is postulated that there exists a non-negative, convex, lower semi-continuous function $\varphi(\dot{\varepsilon}, \dot{\alpha})$, which satisfies $\varphi(0,0) = 0$, and such that $\Phi = \dot{\varphi}$. This assumption yields the relations

$$\begin{aligned} \frac{\partial \varphi}{\partial \dot{\varepsilon}} &= \sigma - \frac{\partial W}{\partial \varepsilon}, && (1.33) \\ \frac{\partial \varphi}{\partial \dot{\alpha}} &= -\frac{\partial W}{\partial \alpha}. && (1.34) \end{aligned}$$

We next consider different choices for the free energy and the pseudopotential of dissipation to generate different damage models.

1.3.1 Gradient damage model

In a first model, we assume that the free energy and the pseudopotential of dissipation take the following form

$$\begin{aligned} W(\varepsilon, \alpha) &= \frac{1}{2}\mathrm{a}(\alpha) A_0 \varepsilon(u) : \varepsilon(u) + w(\alpha) + \frac{1}{2} w_1 \ell^2 |\nabla \alpha|^2, && (1.35) \\ \varphi(\dot{\varepsilon}, \dot{\alpha}) &= I_+(\dot{\alpha}), && (1.36) \end{aligned}$$

where, $I_+(\dot{\alpha})$ is the indicator function defined by

$$I_+(x) = \begin{cases} 0, & \text{if} \quad x \geq 0, \\ +\infty, & \text{if} \quad x < 0. \end{cases} \tag{1.37}$$

Equations (1.33)-(1.34) then yield

$$\begin{aligned} \sigma(u, \alpha) - \mathrm{a}(\alpha) A_0 \varepsilon &= 0, && (1.38) \\ \frac{1}{2}\mathrm{a}'(\alpha) A_0 \varepsilon : \varepsilon + w'(\alpha) - w_1 \ell^2 \Delta \alpha &= -R(\dot{\alpha}), && (1.39) \end{aligned}$$

where, $R(\dot{\alpha}) = 0$ if $\dot{\alpha} \geq 0$ and $R(\dot{\alpha}) \in (-\infty, 0]$ if $\dot{\alpha} = 0$.

This gradient damage model is studied in [40] and its evolution is defined as follows:

1. The stress tensor $\sigma(x,t) = \sigma(u(x,t), \alpha(x,t)) = \mathrm{a}(\alpha(x,t))A_0\varepsilon(u(x,t))$ satisfies the equilibrium equations

$$\begin{array}{rcll} \mathrm{div}(\sigma(x,t)) + f(x,t) &=& 0 & \text{in} \quad \Omega, \\ \sigma(x,t)\cdot n &=& F(x,t) & \text{on} \quad \partial\Omega_F, \\ u(x,t) &=& U(x,t) & \text{on} \quad \partial\Omega_U. \end{array} \tag{1.40}$$

2. The damage field $\alpha(x,t)$ satisfies the nonlocal damage criterion

$$\frac{1}{2}\left(\mathrm{a}'(\alpha(x,t))A_0\varepsilon(u(x,t)) : \varepsilon(u(x,t))\right) + w'(\alpha(x,t)) - w_1\ell^2\Delta\alpha(x,t) \geq 0, \quad \text{in } \Omega \tag{1.41}$$

and the nonlocal consistency condition

$$\left(\frac{1}{2}\left(\mathrm{a}'(\alpha)A_0\varepsilon(u(x,t)) : \varepsilon(u(x,t))\right) + w'(\alpha(x,t)) - w_1\ell^2\Delta\alpha(x,t)\right)\dot{\alpha}(x,t) = 0, \quad \text{in } \Omega \tag{1.42}$$

1.3.2 Gradient damage model for shear fracture

In this model, the pseudopotential of dissipation again takes the form

$$\varphi(\dot{\varepsilon}, \dot{\alpha}) = I_+(\dot{\alpha}). \tag{1.43}$$

For the free energy, we consider the approach based on the orthogonal decomposition of the linearized strain tensor in its spherical and deviatoric components defined in (1.26). With this decomposition, the free energy W is written as the sum of the spherical and deviatoric contribution of the strain tensor. Moreover, to reproduce the shear fracture, the spherical part remains unaffected by the value od the damage scalar α, that is,

$$W(\varepsilon, \alpha) = \left(\lambda + \frac{2\mu}{n}\right)\frac{\mathrm{tr}(\varepsilon(u))^2}{2} + \mathrm{a}(\alpha)\mu\varepsilon^{\mathrm{d}} : \varepsilon^{\mathrm{d}} + w(\alpha) + \frac{1}{2}w_1\ell^2|\nabla\alpha|^2, \tag{1.44}$$

where, λ and μ are the Lamé coefficients. Then, considering the above mentioned, the equations (1.33)-(1.34) yield

$$\begin{array}{rcll} \sigma(u,\alpha) - \left((2\mu + n\lambda)\varepsilon^s + 2\mathrm{a}(\alpha)\mu\varepsilon^{\mathrm{d}}\right) &=& 0, & (1.45) \\ \mathrm{a}'(\alpha)\mu\varepsilon^{\mathrm{d}} : \varepsilon^{\mathrm{d}} + w'(\alpha) - w_1\ell^2\Delta\alpha &=& -R(\dot{\alpha}). & (1.46) \end{array}$$

This damage model is studied in [35], where the creation of surface energy may be compensated exclusively by a reduction of the deviatoric elastic energy.

Accordingly, the evolution is defined as follows: at each time t,

1. The stress tensor $\sigma(x,t) = \sigma(u(x,t), \alpha(x,t)) = \left((2\mu + n\lambda)\varepsilon^s(x,t) + 2\mathrm{a}(\alpha(x,t))\mu\varepsilon^{\mathrm{d}}(x,t)\right)$ satisfies the equilibrium equations

$$\begin{array}{rcll} \mathrm{div}(\sigma(x,t)) + f(x,t) &=& 0 & \text{in} \quad \Omega, \\ \sigma(x,t)\cdot n &=& F(x,t) & \text{on} \quad \partial\Omega_F, \\ u(x,t) &=& U(x,t) & \text{on} \quad \partial\Omega_U. \end{array} \tag{1.47}$$

2. The damage field $\alpha(x,t)$ satisfies the nonlocal damage criterion

$$\mathrm{a}'(\alpha(x,t))\mu\varepsilon^{\mathrm{d}}(x,t) : \varepsilon^{\mathrm{d}}(x,t) + w'(\alpha(x,t)) - w_1\ell^2\Delta\alpha(x,t) \geq 0, \quad \text{in } \Omega, \tag{1.48}$$

and the nonlocal consistency condition

$$\left(\mathrm{a}'(\alpha(x,t))\mu\varepsilon^{\mathrm{d}}(x,t) : \varepsilon^{\mathrm{d}}(x,t) + w'(\alpha(x,t)) - w_1\ell^2\Delta\alpha(x,t)\right)\dot{\alpha}(x,t) = 0, \quad \text{in } \Omega \tag{1.49}$$

Chapter 2

Linear elasticity and differentiation with respect to the domain

In this chapter, we will consider the linear elasticity system as a first approximation of the problem, this will allow us to better understand the impact of mining on the stress behavior and with it the energy behavior of the elasticity system. Numerical simulations will be developed to study some important quantities and as these are affected by changes in geometry, for this we will use tools from the optimal design, in particular the so-called derivation with respect to the domain.

2.1 Linear Elasticity

The linear elasticity problem for static equilibrium of a homogeneous isotropic body $\Omega \subseteq \mathbb{R}^n$ under the assumption of small deformations and strains reads: find the symmetric stress tensor $\sigma = [\sigma_{\mathrm{i}j}]_1^n$ and the displacement vector $u = [u_{\mathrm{i}}]_1^n$, such that

$$\begin{array}{rcll} \operatorname{div}(\sigma) & = & f, & \text{in} \quad \Omega, \\ \sigma & = & 2\mu\varepsilon(u) + \lambda\,(\nabla \cdot u)\, I, & \text{in} \quad \Omega, \\ u & = & 0, & \text{on} \quad \Gamma_D, \\ \sigma \cdot n & = & g, & \text{on} \quad \Gamma_N. \end{array} \tag{2.1}$$

Here, f is a given body force, and g a given traction load acting along a segment Γ_N of the boundary, which has an outward unit normal n. Along the rest of the boundary Γ_D the body is clamped and can not be displaced. The elastic properties of the body are governed by the positive constants λ and μ called the Lamé parameters. Further, $\varepsilon(u) = [\varepsilon_{\mathrm{i}j}]_1^n$ is the strain tensor with components

$$\varepsilon_{\mathrm{i}j}(u) = \frac{1}{2}\left(\frac{\partial u_{\mathrm{i}}}{\partial x_j} + \frac{\partial u_j}{\partial x_{\mathrm{i}}}\right), \quad \mathrm{i}, j = 1, 2. \tag{2.2}$$

The divergence of the tensor σ and the vector u is defined by

$$\operatorname{div}(\sigma) = \left[\sum_{j=1}^{n} \frac{\partial \sigma_{\mathrm{i}j}}{\partial x_j} \right]_{\mathrm{i}=1}^{n}, \quad \operatorname{div}(u) = \textstyle\sum_{\mathrm{i}=1}^{n} \frac{\partial u_{\mathrm{i}}}{\partial x_{\mathrm{i}}}. \tag{2.3}$$

Finally, I is the $n \times n$ identity matrix.

2.2 Effect of the boundary conditions in the elasticity equation

In this section, we will study the effect of the boundary conditions in elasticity problem. To do this, we will compare the results obtained in a spherical bounded domain with the exact solution of the elasticity problem in an infinite domain and compare the results on spheres with different radii.

In the first instance, let us consider $\overline{u}$ as the solution of the following elasticity problem

$$-\operatorname{div}\left(\sigma(\overline{u})\right) = f \mathbb{1}_{\omega} \quad \text{in} \quad \mathbb{R}^n, \tag{2.4}$$

where f is a force supported in the open bounded $\omega \subseteq \mathbb{R}^n$. In particular we consider the Dirac delta function at the origin. For this problem, there is an explicit solution, defined for each i component by

$$\overline{u}_{\mathrm{i}}(x) = G_{\mathrm{i}j}(x) f_j \mathbb{1}_{\omega}, \tag{2.5}$$

with $G_{\mathrm{i}j}(x)$, the Green function for the elasticity equation defined as follow

$$G_{\mathrm{i}j}(x) = \frac{1}{16\pi\mu(1-\nu)r} \left((3-4\nu)\delta_{\mathrm{i}j} + r_{,\mathrm{i}} r_{,j} \right), \tag{2.6}$$

where $r_{\mathrm{i}} = x_{\mathrm{i}}$, $r_{,\mathrm{i}} = \frac{\partial r}{\partial x_{\mathrm{i}}}$ and $r = |x|$.

On the other hand, in our problem we are going to consider a bounded domain where we impose a Robin type boundary condition, given by

$$\begin{array}{rcll} -\operatorname{div}(\sigma(u)) & = & f \mathbb{1}_{\omega}, & \text{in} \quad \Omega, \\ \sigma(u)n + k(u \cdot n)n & = & 0, & \text{on} \quad \partial\Omega. \end{array} \tag{2.7}$$

In order to study the effect produced by delimiting the domain, let us consider as domain $\Omega = B_R$ corresponding to a ball centered on the origin and radius R and consider $w = \overline{u} - u$ that satisfies the following problem.

$$\begin{array}{rcll} -\operatorname{div}(\sigma(w)) & = & 0, & \text{in} \quad B_R, \\ \sigma(w)n + k(w \cdot n)n & = & \sigma(\overline{u})n + k(\overline{u} \cdot n)n, & \text{on} \quad \partial B_R. \end{array} \tag{2.8}$$

The objective of this analysis is to study the behavior of the solution w for different values of R, that is, we will study what happens with the difference $\overline{u} - u$ as R grows.

We will consider the following parameters

$$E = 2.9 \cdot 10^{10}, \quad \nu = 0.3, \quad k = 10^9 \quad \text{and} \quad \mu = \tfrac{E}{2(1-\nu)}. \tag{2.9}$$

We also consider that the force in ω is $f = (0, 0, -g)^T$, with $\rho = 2.7 \cdot 10^3$ and $g = 9.8$

2.2.1 Results for a fixed radius

First, we will study the results for a fixed radius $R = 15000$ and study the magnitude for each of the solutions and observe the results. To develop these examples we consider a mesh with a greater refinement in the center, this in order to better represent a concentrated force in the center of the ball, see Figure 2.1

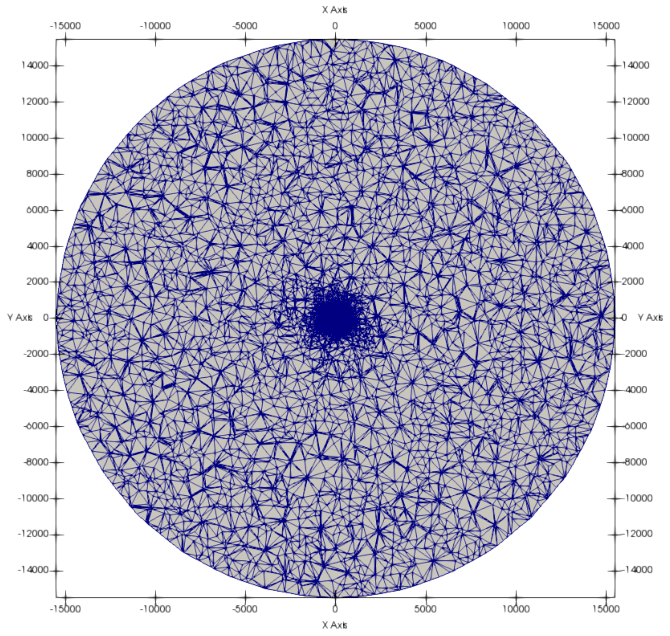


Figure 2.1: Mesh side cut.

In the Figures 2.2-2.3, is shown the magnitude of the exact solution, considering a view in the y-axis and z-axis respectively, where its highest value is in the center, this is consistent because the explicit solution has a behavior of the form $\frac{1}{R}$.

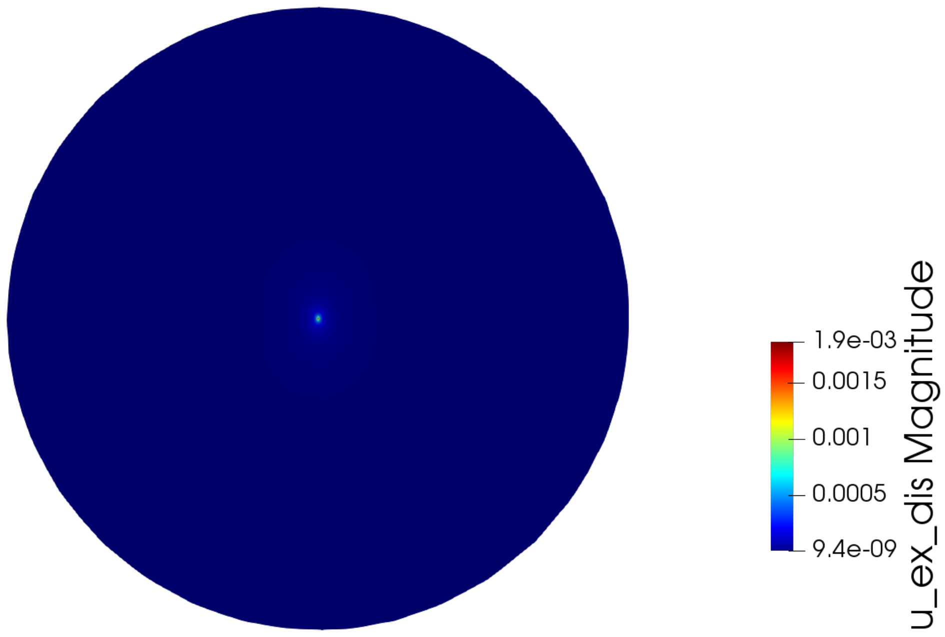


Figure 2.2: Magnitude of the exact solution (y-axis cut).

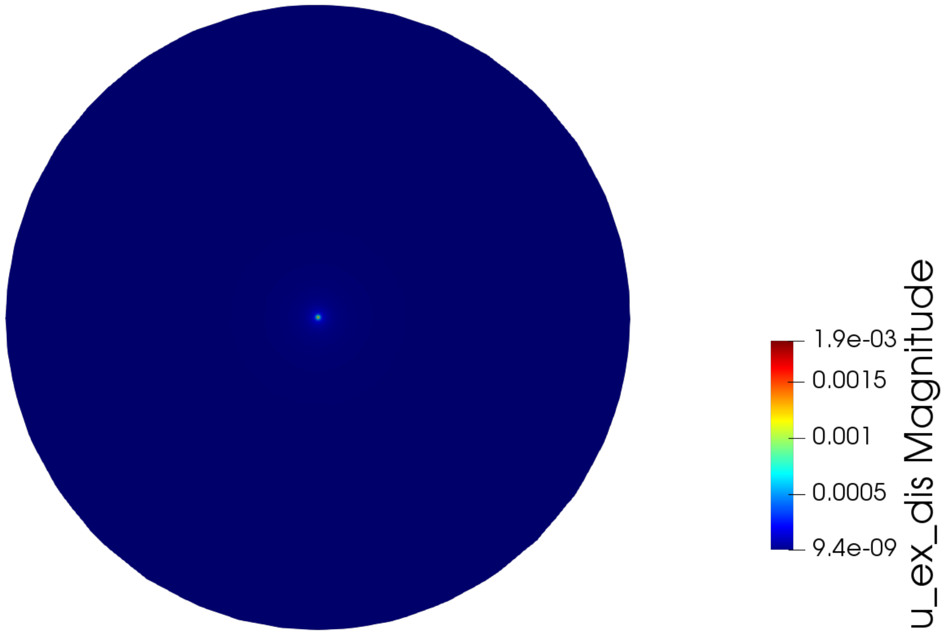


Figure 2.3: Magnitude of the exact solution (z-axis cut).

Figure 2.4 the magnitude of the error can be observed when considering a cut in the y-axis, it is observed that the magnitude of the error is greater in a stripe in the center of the sphere through the two poles. It is also possible to observe in Figure 2.5, that when making a cut in the z-axis, when considering a radius of $R = 15000$ meters the error takes its lower value at the edges and its greater value at the center axis of the sphere, so the error has a greater concentration in the center.

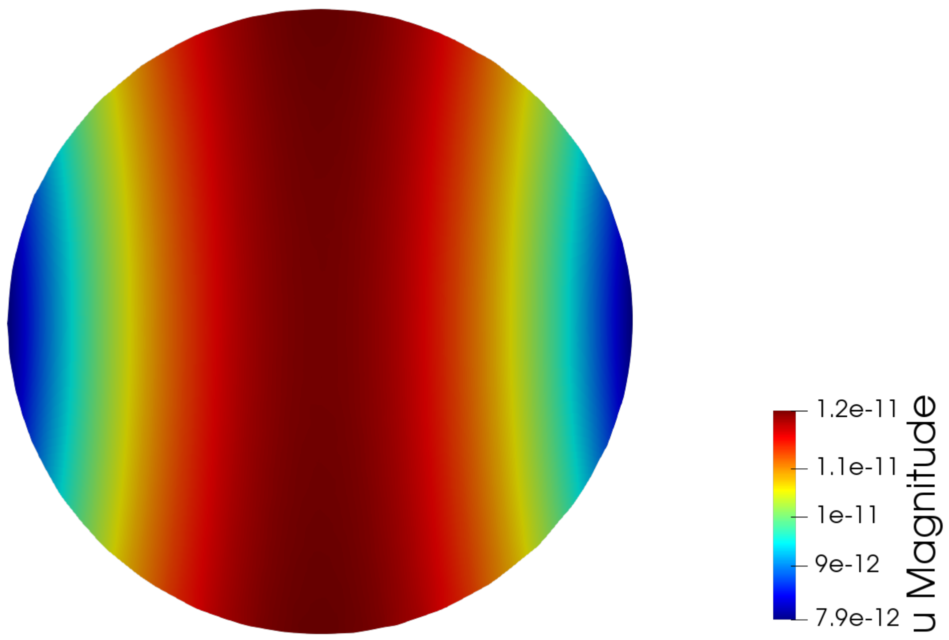


Figure 2.4: Error magnitude (y-axis cut).

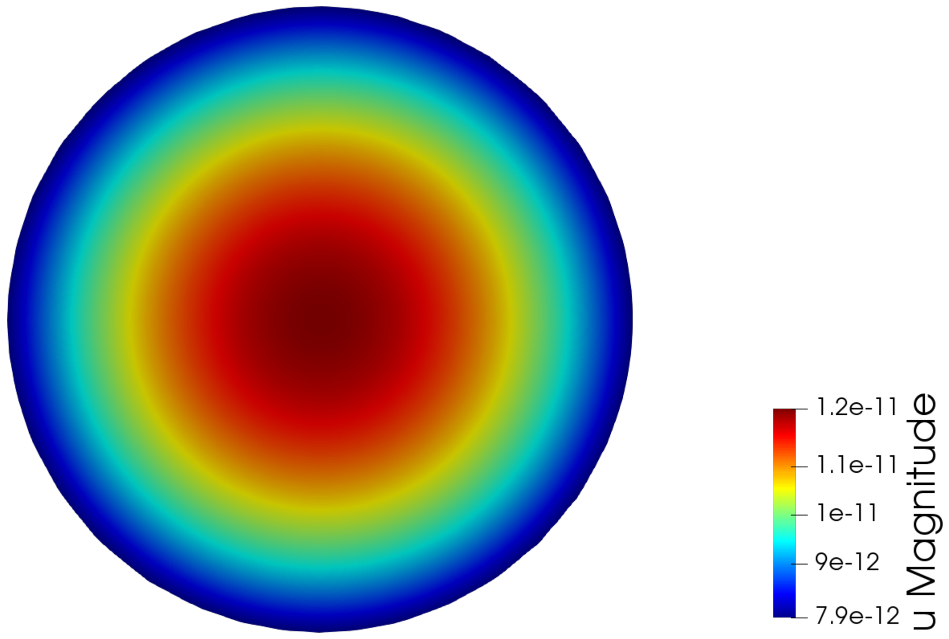


Figure 2.5: Error magnitude (z-axis cut).

Finally, if we consider the relative error, that is, the ratio between the magnitude of the error and the magnitude of the exact solution, it can be seen in Figure 2.6 the maximum value is concentrated at the poles of the sphere in the y-axis. On the other hand we can see in Figure 2.7 that the maximum value is concentrated at the boundary and the minimum value at the center in the z-axis.

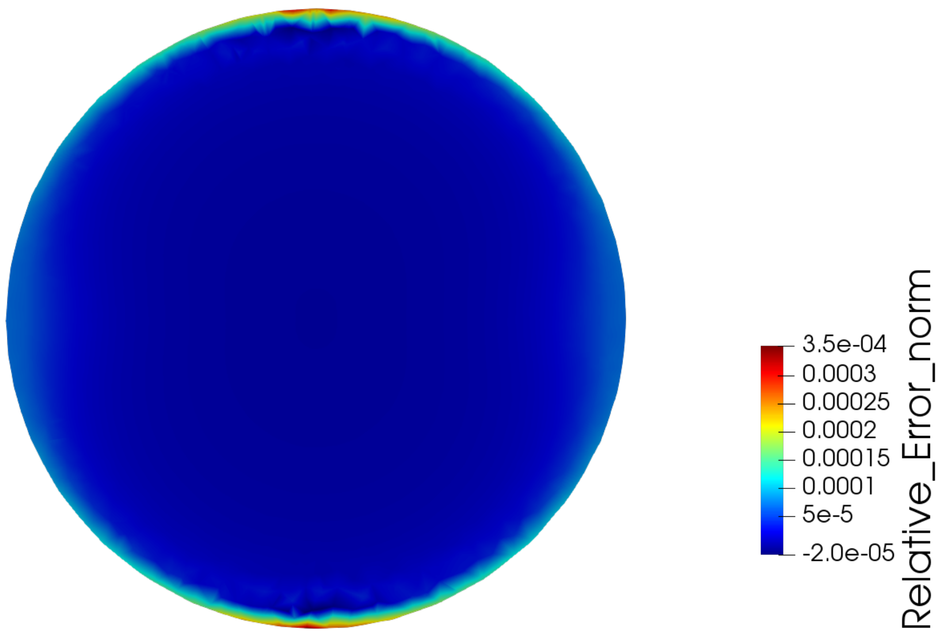


Figure 2.6: Magnitude of relative error (y-axis cut).

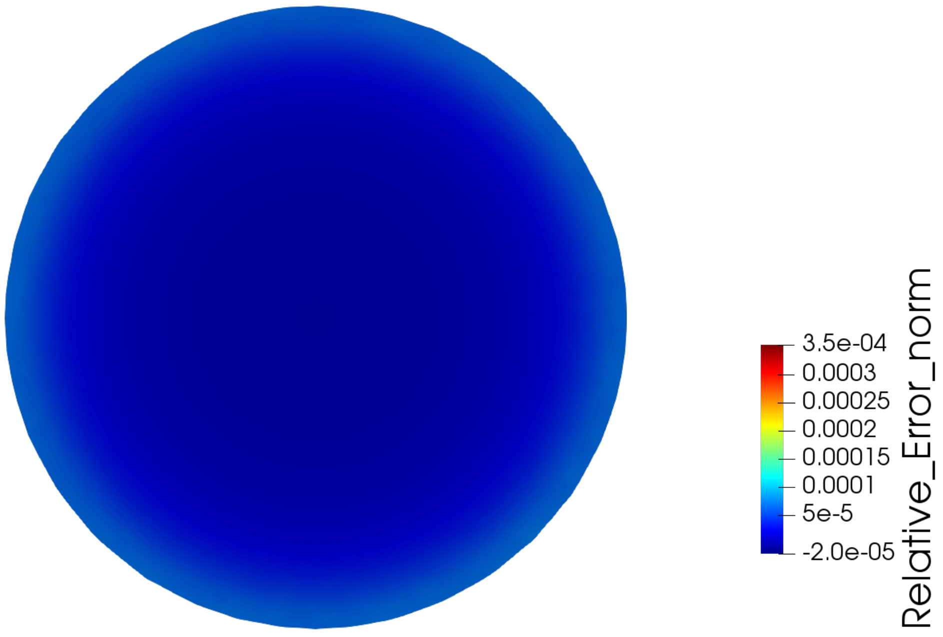


Figure 2.7: Magnitude of relative error (z-axis cut).

To study the above in a better way, we will consider the value of this magnitude in a radius located on the x-axis where the left axis will consider the magnitude on a logarithmic scale. Figure 2.8 shown that the error decreases in a slower than the exact solution, so the relative error is increasing as we approach the boundary, this is a consequence of applying the boundary condition.

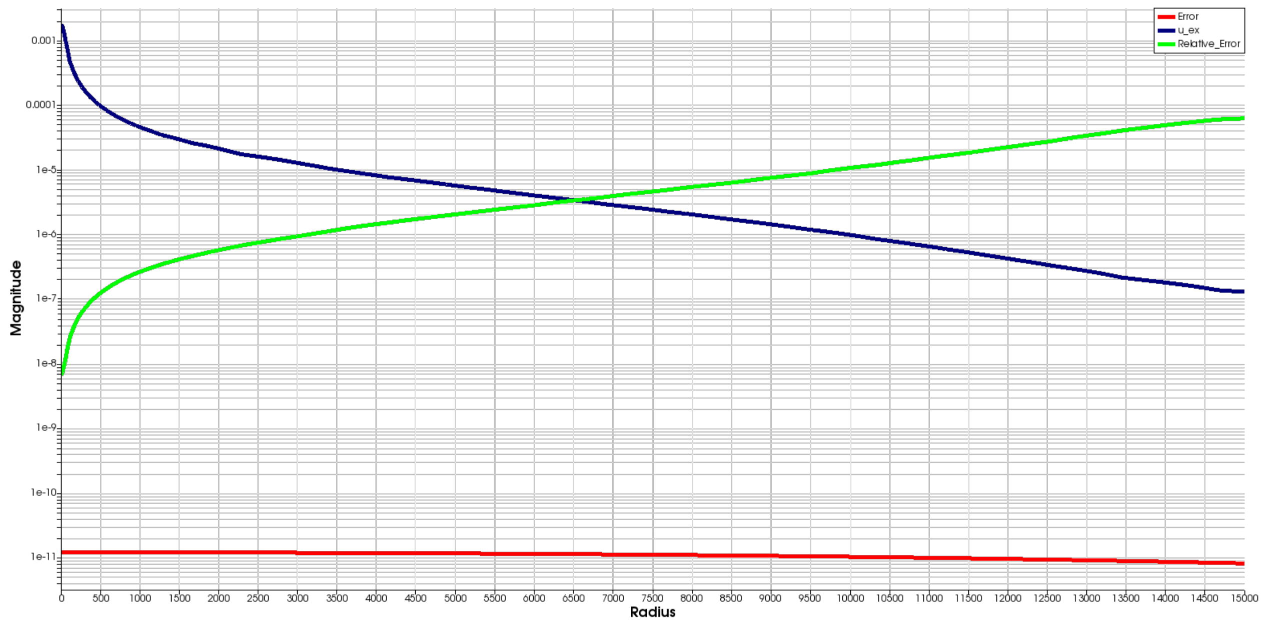


Figure 2.8: Comparison between error magnitudes, exact solution, and relative error.

The results shown above demonstrate that the exact solution magnitude behaves in the form $\frac{1}{R}$, while the magnitude of the error between both solutions is not seriously affected when considering a bounded domain. On the other hand, when considering the relative error, we can see that it takes its greatest value at the boundaries of the domain because in that place one is where it imposes the boundary conditions, affecting to the solution of the problem.

2.2.2 Results for different radius values

In this section, we will study the maximum values that are obtained in the sphere and how they vary as their radius increases. As can be seen in Figures 2.6-2.7, the relative error has its maximum value at the boundary of the domain, this because in that area it is where we impose the boundary condition, so always the value maximum we will find in that area. In order to improve this and have a better comparison for each radius variation, we are going to truncate this relative error in a radius of size 1000 meters.

For the following results we will consider the relative error, which we will define as follows:

$$\text{Relative Error} = \sup_{x} \left| \frac{\|w(x)\|_2}{\|u_{\text{ex}}(x)\|_2} \right| \tag{2.10}$$

where $\|\cdot\|_2$ is the Euclidean norm. In Table 2.1 the results can be seen, where it can be seen that the maximum of the exact solution takes a constant value, this because the maximum value is located in the center, instead the error decreases to ratio $\frac{1}{R}$ as the radius increases. It can be see nthe Relative Error for each value of radius R where it remains constant as the radius increases, but on the other hand, the truncated Relative Error decreases as the radius increases. These results tell us that as the radius grows, the error to consider bounded domains decreases.

Radius	$\|w\|_\infty$	$\|u_{ex}\|_\infty$	**Relative Error**	**Truncated relative error**
1.50E+03	1.23E-10	2.22E-03	7.62E-05	8.02E-06
2.00E+03	9.29E-11	2.25E-03	9.55E-05	3.78E-06
2.50E+03	7.46E-11	2.26E-03	1.07E-04	2.34E-06
3.00E+03	6.23E-11	2.27E-03	1.20E-04	1.74E-06
3.50E+03	5.35E-11	2.27E-03	1.30E-04	1.32E-06
4.00E+03	4.69E-11	2.28E-03	1.42E-04	1.06E-06
4.50E+03	4.17E-11	2.28E-03	1.54E-04	9.10E-07
5.00E+03	3.76E-11	2.27E-03	1.59E-04	8.05E-07
5.50E+03	3.42E-11	2.30E-03	1.71E-04	7.00E-07
6.00E+03	3.13E-11	2.29E-03	1.91E-04	6.22E-07
6.50E+03	2.89E-11	2.26E-03	1.92E-04	5.91E-07
7.00E+03	2.69E-11	2.28E-03	1.90E-04	5.47E-07
7.50E+03	2.51E-11	2.27E-03	2.16E-04	5.06E-07
8.00E+03	2.35E-11	2.29E-03	2.07E-04	4.55E-07
8.50E+03	2.22E-11	2.24E-03	2.63E-04	4.32E-07
9.00E+03	2.09E-11	2.30E-03	2.42E-04	3.84E-07
9.50E+03	1.98E-11	2.22E-03	2.46E-04	3.93E-07
1.00E+04	1.88E-11	2.34E-03	2.13E-04	3.39E-07
1.05E+04	1.80E-11	2.31E-03	2.75E-04	3.27E-07
1.10E+04	1.71E-11	2.39E-03	2.52E-04	2.91E-07
1.15E+04	1.64E-11	2.29E-03	2.71E-04	2.92E-07
1.20E+04	1.57E-11	2.23E-03	2.74E-04	2.94E-07
1.25E+04	1.51E-11	2.39E-03	2.41E-04	2.53E-07
1.30E+04	1.45E-11	2.26E-03	2.84E-04	2.59E-07
1.35E+04	1.40E-11	2.28E-03	2.83E-04	2.43E-07
1.40E+04	1.35E-11	2.19E-03	2.63E-04	2.50E-07
1.45E+04	1.30E-11	2.20E-03	2.85E-04	2.33E-07
1.50E+04	1.26E-11	2.17E-03	2.87E-04	2.30E-07
1.55E+04	1.22E-11	2.16E-03	2.77E-04	2.21E-07

Table 2.1: Maximum values of the Error and the exact solution u_{ex} for different radius R.

To see the above in a better way, we will consider the value of truncated relative error on a logarithmic scale. Figure 2.9 shown that the this magnitude is monotonically decreases attained the maximum curvature about the radius $R = 3000$, after that value the truncated relative error stabilizes.

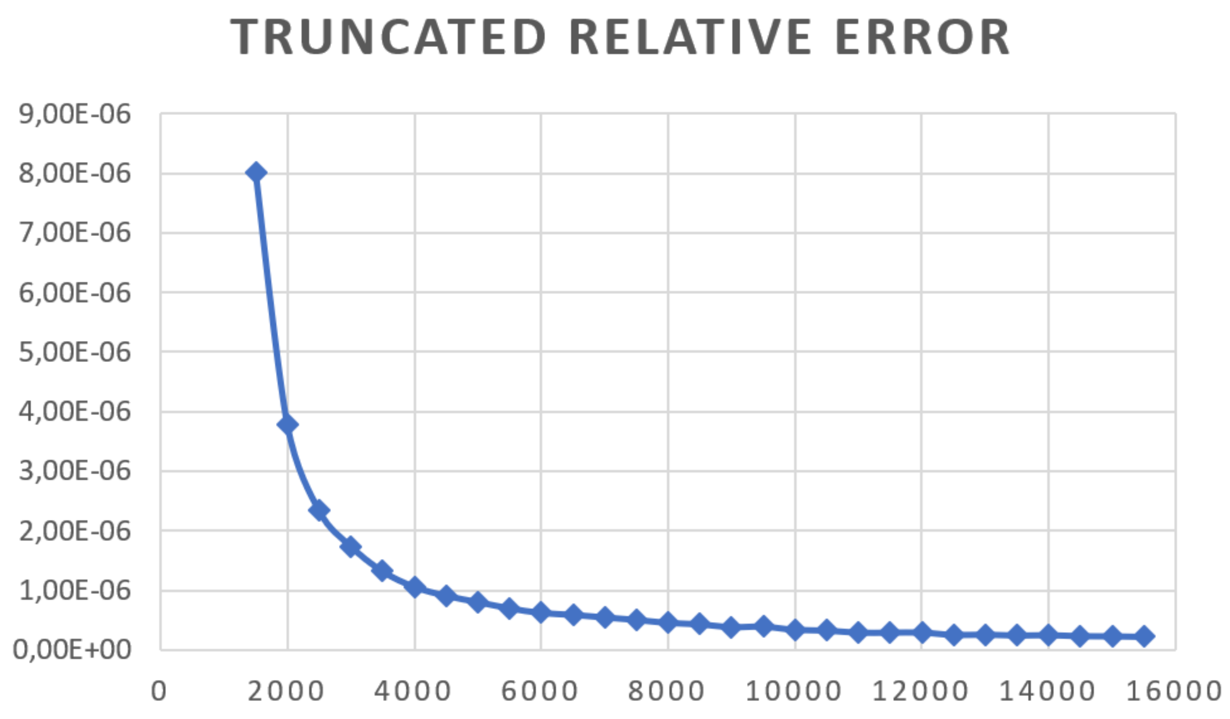


Figure 2.9: Truncated relative error for different values of R.

The results shown above exhibit that considering a radius greater than 3000, the error produces by the boundary conditions are not significant in the solution of the elasticity equation. For this reason we consider that the domains will be about 3 times bigger that the cavity in the damage problem.

2.3 Differentiation with respect to the domain

In order to study the impact of the change in the shape of the cavity in the system (2.1), the theory of differentiation with respect to the domain will be used. This approach captures the sensitivity of the solution (2.1) to the shape change of cavity.

The derivation with respect to the domain is borned with the optimal design, that is, problems in which we look for the geometry that minimizes a certain cost function. These types of problems appear naturally in areas such as the design of structures, profiles of the wings of an airplane, among others.

To define this mathematically, following [52], we consider an $\Omega \subseteq \mathbb{R}^n$ geometry and denote by $u(\Omega)$ the state of the given system as the solution of the following problem

$$\begin{array}{rclll} Au(\Omega) & = & f, & \text{in} & \Omega, \\ Bu(\Omega) & = & g, & \text{on} & \partial\Omega. \end{array} \tag{2.11}$$

Here A and B denote partial differential operators and f and g are some functions defined on $\mathbb{R}^n$.

Our objective is to know the dependence of the solution of the problem (2.11) with respect to the variations of the geometry and in particular the behavior of certain cost functionalities with respect to the geometry.

Let $\Omega \subseteq \mathbb{R}^n$ be our domain of reference, let us consider a function $\phi : \mathbb{R}^n \to \mathbb{R}^n$ regular enough, such that the domain variations can be written as

$$\Omega + \phi = \{y = x + \phi(x) \in \mathbb{R}^n \,:\, x \in \Omega\}\,. \tag{2.12}$$

In this way we can define the functional $\phi \mapsto u(\Omega + \phi)$, where $u(\Omega + \phi)$ is the solution of the problem (2.11) in the domain $\Omega + \phi$.

The central idea is to define the concept of derivative of the function (2.11) with respect to the change of geometry. To do this, let us define the local variation of u as follows. Let $\omega \subset \Omega$, we define the local variation of u in the direction ϕ as

$$u'(\phi) = \lim_{t \to 0} \frac{u(\Omega + \phi)\big|_\omega - u(\Omega)\big|_\omega}{t}, \tag{2.13}$$

and we have to

$$u(\Omega + \phi) = u(\Omega) + u'(\phi) + o(\|\phi\|). \tag{2.14}$$

In our case, we consider a domain that simulates a rock mass in three dimensions $\Omega_0 \subseteq \mathbb{R}^3$, with $\partial\Omega = \Gamma_{lat} \cup \Gamma_{up} \cup \Gamma_{down}$ denoting the lateral, upper and lower bounds of the domain respectively. We consider a interior domain $S_0 \subseteq \Omega_0$, with $\partial S_0 = \Gamma_{cav}$, representing the interior cavity of this rock mass. Finally we consider the domain $\Omega = \Omega_0 \setminus S_0$ that it will be our whole domain.

The linear elasticity equation on Ω correspond to

$$\begin{array}{rcll} -\mathrm{div}(\sigma(u)) &=& f, & \text{in} \quad \Omega, \\ \sigma(u)\cdot n &=& 0, & \text{on} \quad \Gamma_{cav}, \\ \sigma(u)\cdot n &=& 0, & \text{on} \quad \Gamma_{up}, \\ u \cdot n &=& 0, & \text{on} \quad \Gamma_{down}, \\ \sigma \cdot n &=& g, & \text{on} \quad \Gamma_{lat}, \end{array} \tag{2.15}$$

Our objective is to change the geometry of the cavity, that is, to deform Γ_{cav}, without deforming the regions exteriors $\Gamma_{lat} \cup \Gamma_{up} \cup \Gamma_{down}$ to obtain the local variations of the problem (2.15). Following the work carried out in [52], we have that the local variation u' is the solution to the problem

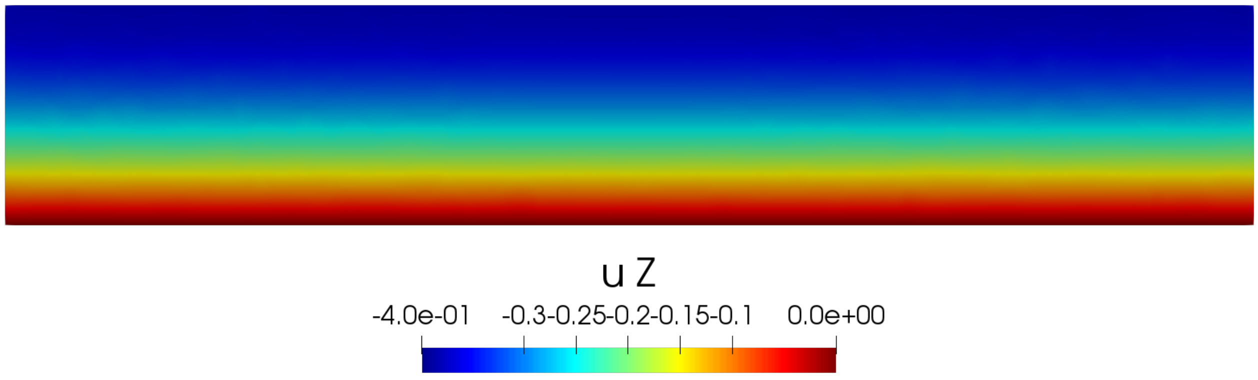
$$\begin{array}{rcll} -\mathrm{div}(\sigma(u')) &=& 0, & \text{in} \quad \Omega, \\ \sigma(u')\cdot n &=& -(\phi\cdot n)\left((\nabla\sigma(u))n\cdot n\right) + \sigma(u)\nabla_{\partial\Omega}(\phi\cdot n), & \text{on} \quad \Gamma_{cav}, \\ \sigma(u')\cdot n &=& 0, & \text{on} \quad \Gamma_{up}, \\ u'\cdot n &=& 0, & \text{on} \quad \Gamma_{down}, \\ \sigma(u')\cdot n &=& 0, & \text{on} \quad \Gamma_{lat}, \end{array} \tag{2.16}$$

where u is the solution of the problem (2.15), that is, the solution of the problem in the reference domain Ω and $\nabla_{\partial\Omega}$ is the tangential gradient defined by

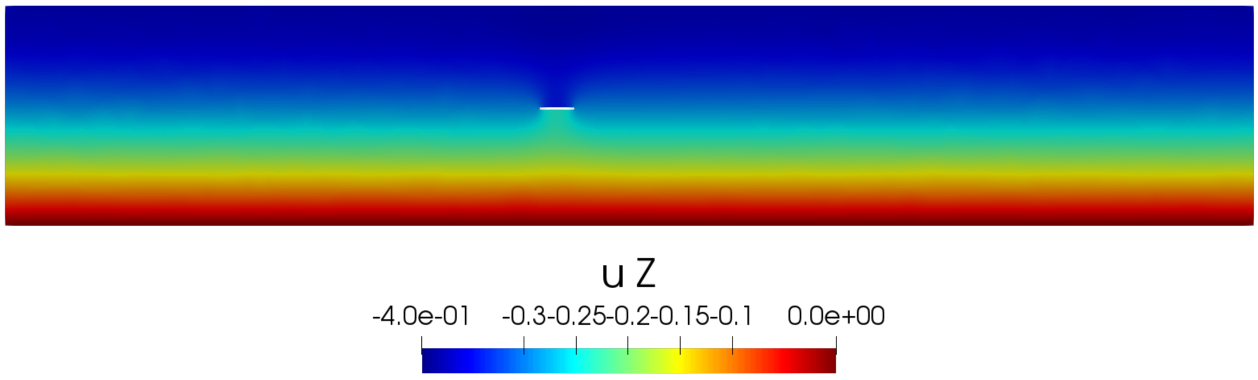
$$\nabla_{\partial\Omega} f = \nabla f - (\nabla f \cdot n)\cdot n \text{ on } \partial\Omega. \tag{2.17}$$

2.3.1 Numerical results

For the numerical tests, we consider first the linear elasticity equation (2.15). Figures 2.10-2.11 display the distribution of the z-component of the displacement in a vertical cut in the y-axis. It is possible to see that as the cavity evolves, the displacement begins to increase over the cavity attaining the maximum displacement in this area.

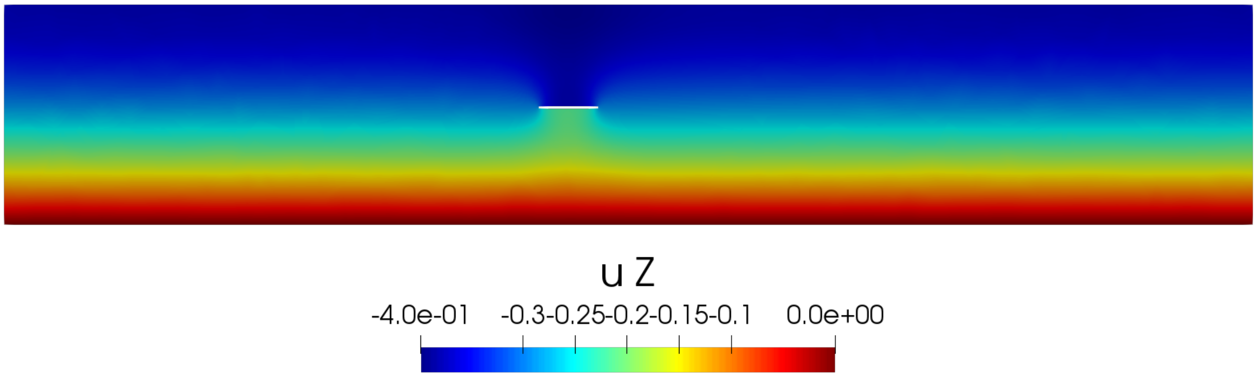


(a) $\Omega(t_0)$.

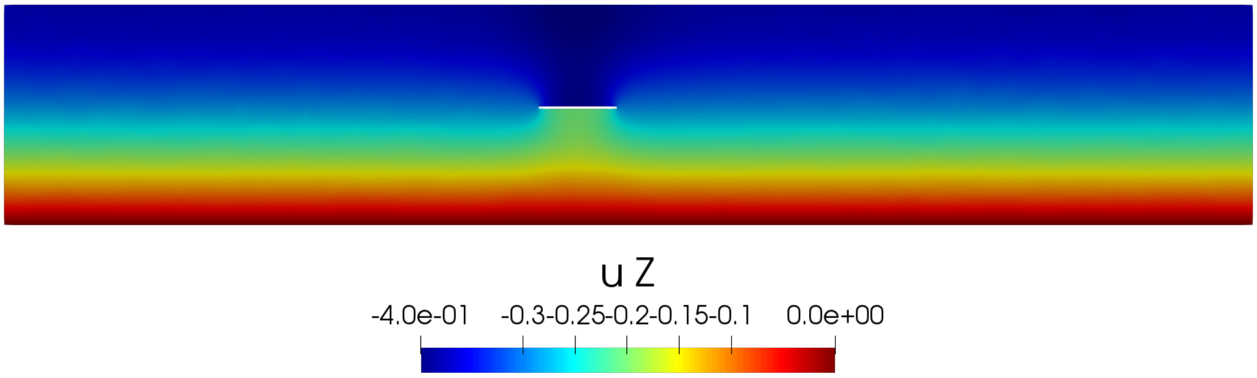


(b) $\Omega(t_7)$.

Figure 2.10: Distribution of vertical displacement in a cut on the y-axis.



(a) $\Omega(t_{15})$.

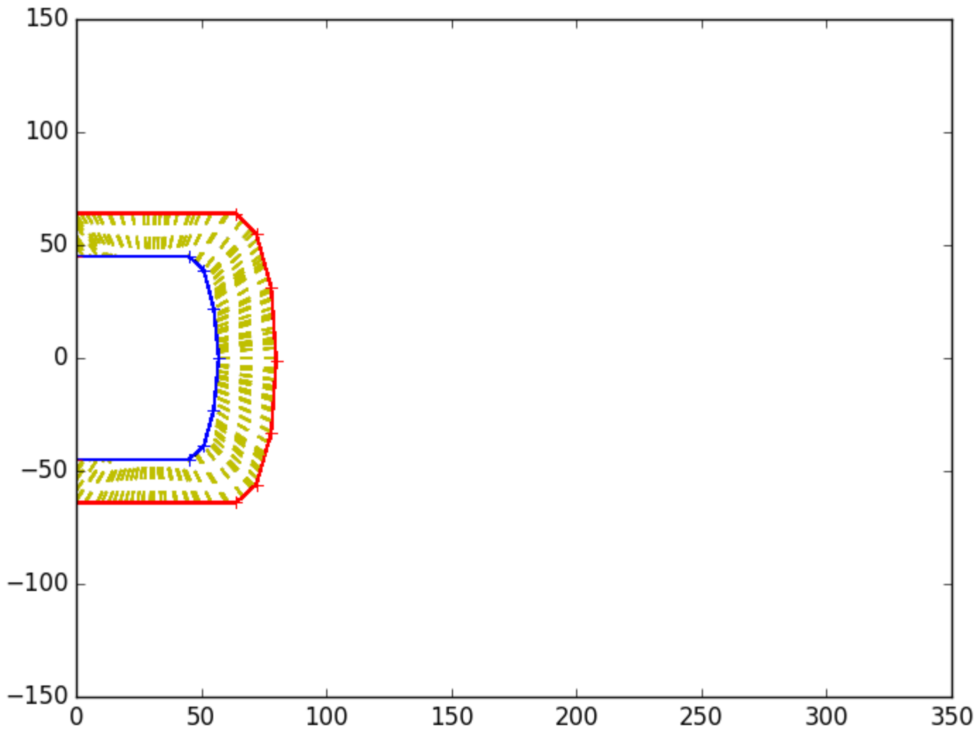


(b) $\Omega(t_{20})$.

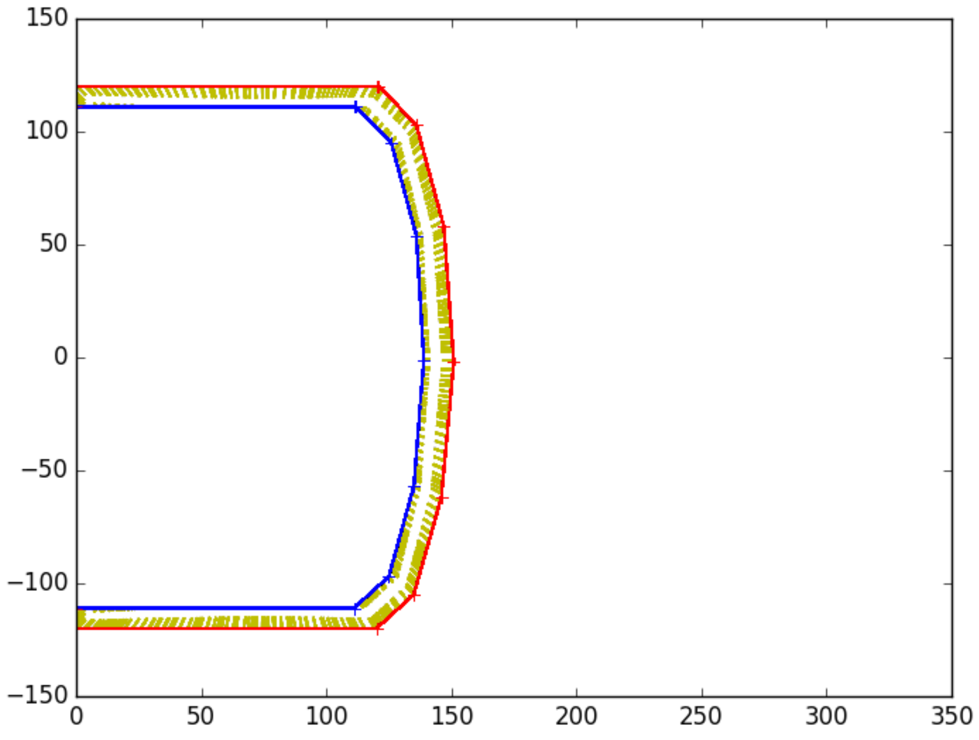
Figure 2.11: Distribution of vertical displacement in a cut on the y-axis.

The results shown above exhibit the behavior of the displacement as the cavity advances, where it is possible to see that the behavior is observed in the block caving process. It is possible to see that considering only the elasticity problem the model can recover the subsidence seen in underground mining.

Now when considering the differentiation with respect to the domain, first we need the deformation vectors in each cavity configuration. Figures 2.12-2.13 show the deformation vectors in a cut in the z-axis for some scenarios, where the blue line represents the cavity a fixed instant, the red line represents the cavity in the following state and the yellow lines represent the deformation vectors in these states..

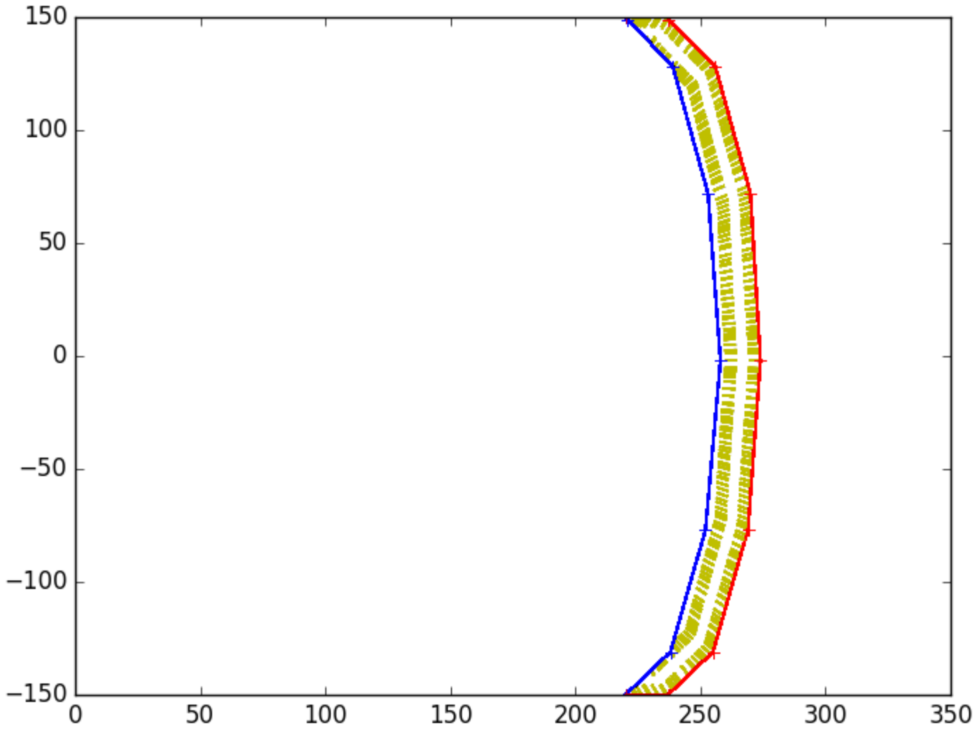


(a) Deformation vectors between $\Omega(t_1)$ and $\Omega(t_2)$.

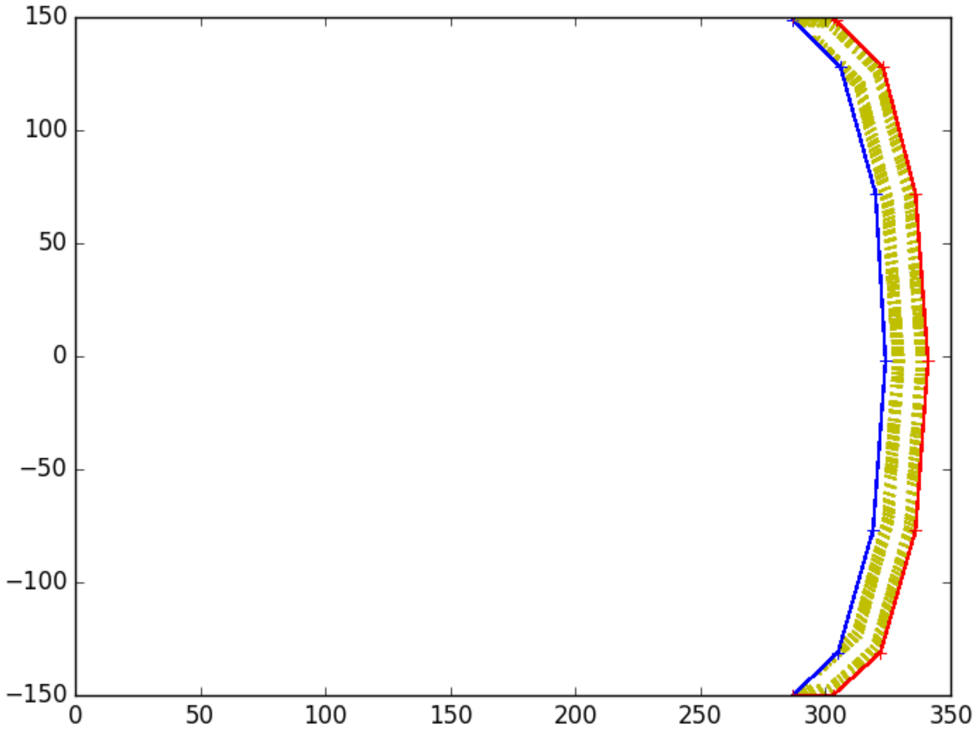


(b) Deformation vectors between $\Omega(t_6)$ and $\Omega(t_7)$.

Figure 2.12: Horizontal view of the deformation vector.



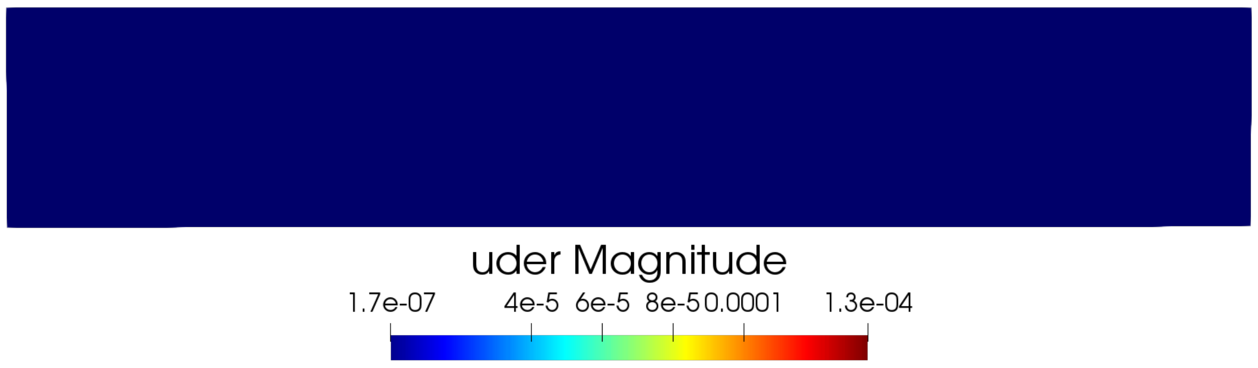
(a) Deformation vectors between $\Omega(t_{14})$ and $\Omega(t_{15})$.



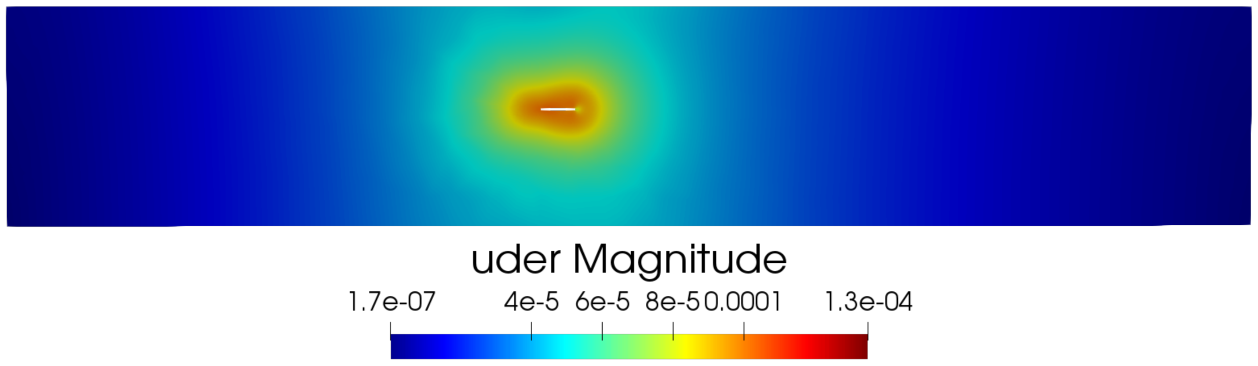
(b) Deformation vectors between $\Omega(t_{19})$ and $\Omega(t_{20})$.

Figure 2.13: Horizontal view of the deformation vector.

Figure 2.14-2.15 display the distribution of the magnitude of the derivative with respect to the domain of the displacement. It is possible to see that as the cavity begins to advance this magnitude spreads around the cavity because the cavity advances both on the x-axis and on the y-axis. On the other hand in the last cavity advance the magnitude of the derivative is distributed around the right cavity end because the cavity advances only in the x-axis.

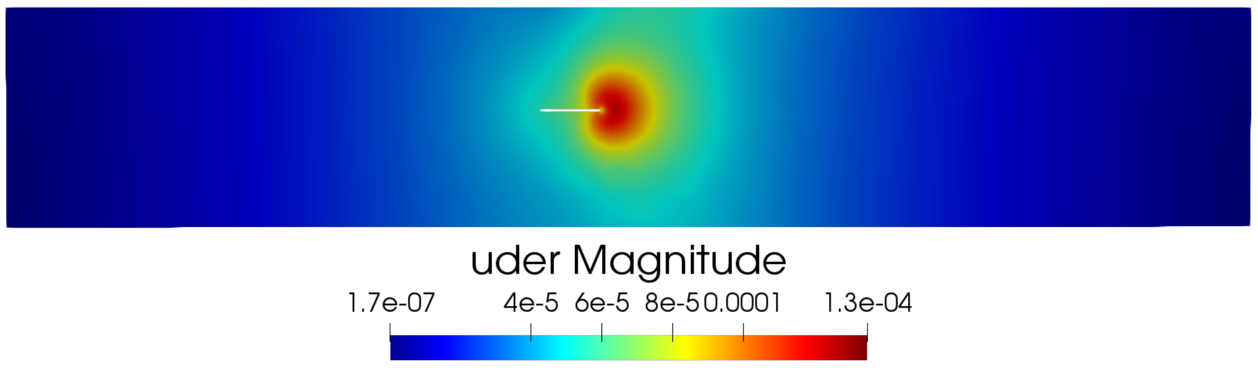


(a) $\Omega(t_0)$.

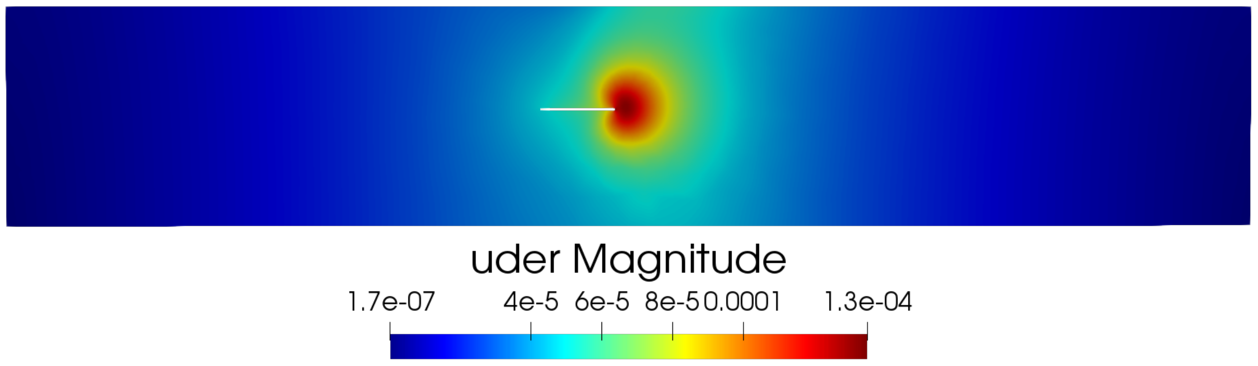


(b) $\Omega(t_7)$.

Figure 2.14: Distribution of magnitude of the derivative with respect to the domain of the displacement in a cut on the y-axis.



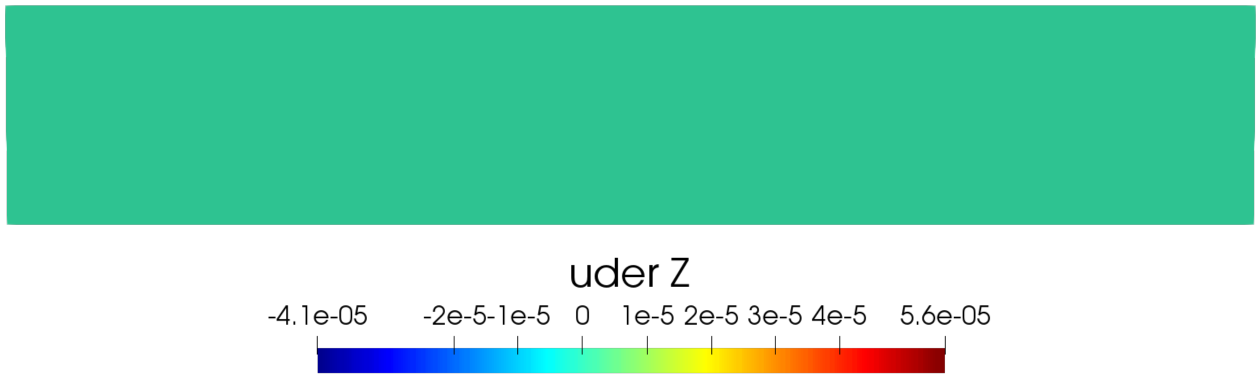
(a) $\Omega(t_{15})$.



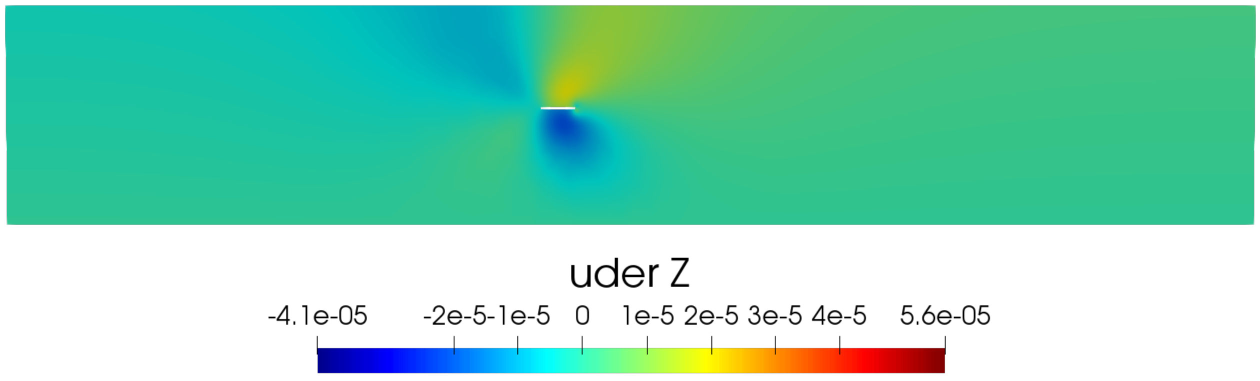
(b) $\Omega(t_{20})$.

Figure 2.15: Distribution of magnitude of the derivative with respect to the domain of the displacement in a cut on the y-axis.

Figure 2.16-2.17 display the distribution of the z-component of the derivative with respect to the domain of the displacement. As in the previous result, it is possible to see that as the cavity begins to advance this magnitude spreads around the cavity because the cavity advances both on the x-axis and on the y-axis. On the other hand in the last cavity advance the magnitude of the derivative is distributed around the right cavity end because the cavity advances only in the x-axis. It is also possible to see how the growth rate of the z-component of the displacement will be as the cavity advances.

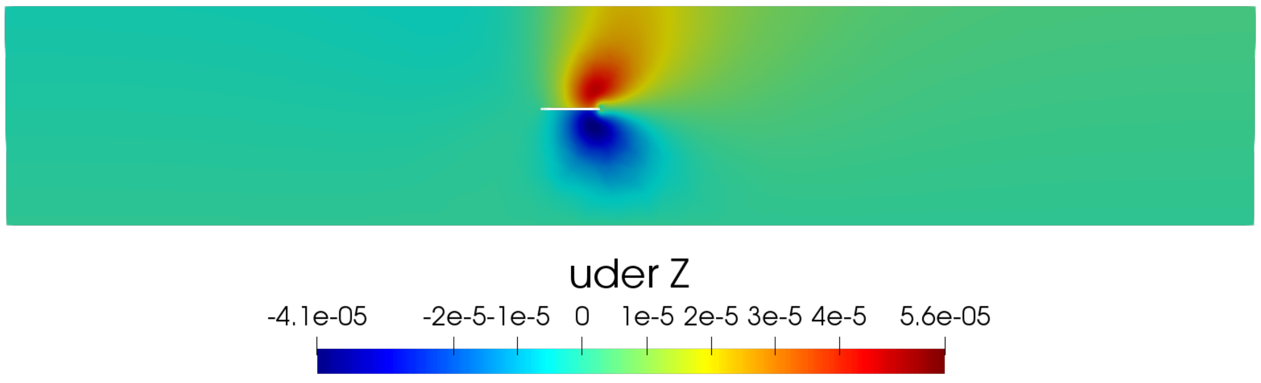


(a) $\Omega(t_0)$.

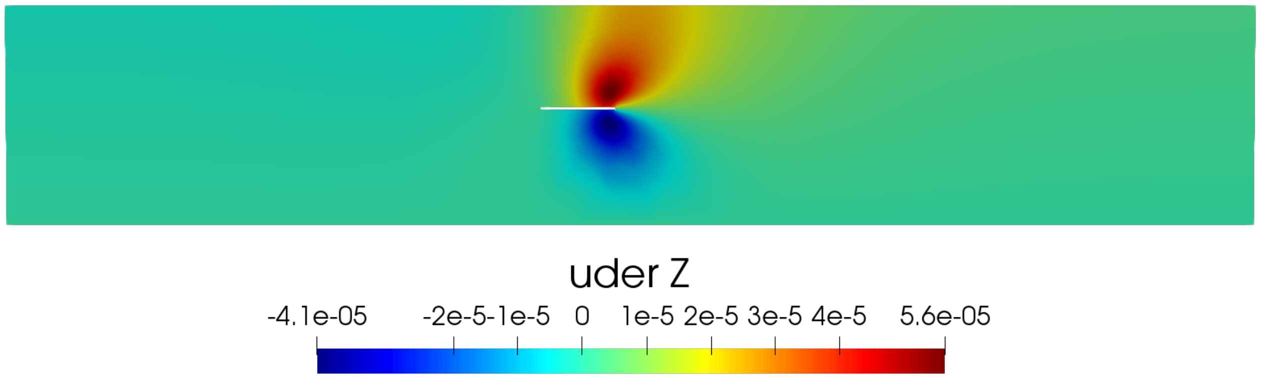


(b) $\Omega(t_7)$.

Figure 2.16: Distribution z-component of the derivative with respect to the domain of the displacement in a cut on the y-axis.



(a) $\Omega(t_{15})$



(b) $\Omega(t_{20})$

Figure 2.17: Distribution z-component of the derivative with respect to the domain of the displacement in a cut on the y-axis.

The results shown above give us a first approximation of how the rock mass will behave in the block caving process. It can be observed that the displacement behaves in the expected way since the subsidence seen in underground mining can be observed. On the other hand, the derivative with respect to the domain gives us information about how the displacement variation rate will be as the block caving process develops.

Chapter 3

A damage model for the simulation of underground mining

In this chapter we introduce the shear-compression damage model, we will use this model to approximate the Block Caving process and compare the results with the classical Gradient Damage Model. The main characteristic of this model is that we can control the contribution in the damage process given by shearing and compression and, as in the Block Caving process the main damage is by the effect of the shear, this model is suitable for modeling underground mining.

3.1 Shear-compression damage model

The main issue of gradient damage models in underground mining is that, due to the gravity force, the damage due to the block caving process is negligible with respect to the damage associated to large compression in the bottom of the domain. For this, we will proposed a new model in order to recover the true effect of the underground mining in the rock mass, the principal idea for this is change the damage criterion, where we will decompose the stress tensor in its deviatoric and spherical part, in order to be able to control the damage produced by the deviatoric and spherical component of the stress.

Following the gradient damage models shown in Section 1.2, the stress-strain relation is given by

$$\sigma(u, \alpha) = \mathrm{a}(\alpha) A_0 \varepsilon(u). \tag{3.1}$$

For the equation that governs the evolution of damage, we consider the following relationship

$$\frac{1}{2} H(\varepsilon, \alpha) + w'(\alpha) - w_1 \ell^2 \Delta\alpha = -R(\dot{\alpha}), \tag{3.2}$$

where $H(\varepsilon, \alpha)$ is a source term for damage.

We define the expression $H(\varepsilon, \alpha)$ in terms of the principal stress of the stress tensor. For a given stress tensor $\sigma = \sigma(u, \alpha) = \mathrm{a}(\alpha) A_0 \varepsilon(u)$, we can solve the characteristic equation to explicitly determine the principal values and directions. In three-dimensional case, following

[53], if we consider

$$m = \frac{1}{3}\text{tr}(\sigma), \tag{3.3}$$

$$q = \frac{1}{2}\det(\sigma - mI), \tag{3.4}$$

$$p = \frac{1}{6}\sum_{\text{i}j}(\sigma - mI)^2_{\text{ij}}, \tag{3.5}$$

then, from Cardano's trigonometric solutions of $\det[(\sigma - mI) - \lambda I]$ as a cubic polynomial in λ, the eigenvalues of σ are defined by

$$\lambda_1 = m + 2\sqrt{p}\cos(\theta), \tag{3.6}$$

$$\lambda_2 = m - \sqrt{p}\left(\cos(\theta) + \sqrt{3}\sin(\theta)\right), \tag{3.7}$$

$$\lambda_3 = m - \sqrt{p}\left(\cos(\theta) - \sqrt{3}\sin(\theta)\right), \tag{3.8}$$

where $\theta = \frac{1}{3}\tan^{-1}\left(\frac{\sqrt{p^3-q^2}}{q}\right)$ and $0 \leq \theta \leq \pi$. Thus, if we rewrite the eigenvalues, we have

$$\lambda_1 = m + 2\sqrt{p}\cos(\theta), \tag{3.9}$$

$$\lambda_2 = m - 2\sqrt{p}\cos\left(\theta - \frac{\pi}{3}\right), \tag{3.10}$$

$$\lambda_3 = m - 2\sqrt{p}\cos\left(\theta + \frac{\pi}{3}\right), \tag{3.11}$$

subtracting the eigenvalues, we obtain

$$\lambda_1 - \lambda_2 = 2\sqrt{3p}\cos\left(\theta - \frac{\pi}{6}\right), \tag{3.12}$$

$$\lambda_1 - \lambda_3 = 2\sqrt{3p}\cos\left(\theta + \frac{\pi}{6}\right), \tag{3.13}$$

$$\lambda_2 - \lambda_3 = 2\sqrt{3p}\sin(\theta). \tag{3.14}$$

Then, considering the trigonometric part of the previous equations, we can assert, following the Mohr's circle of stress, that the largest shear component is obtained when $\frac{\lambda_1 - \lambda_2}{2} = \sqrt{3p}$.

We assume that the material gets damaged when the magnitude of the shear component S is greater than the factor of the magnitude of the normal component N, i.e., $S > \kappa N$. As mentioned earlier, the maximun shear is $\sqrt{3p}$, our damage criterion will be $\sqrt{3p} > \kappa N$. On the other hand, considering $\sigma = \sigma^s + \sigma^{\text{d}}$, where σ^s and σ^{d} are the spherical and deviatoric part of σ respectively defined by $\sigma^s = mI$ and $\sigma^{\text{d}} = \sigma - mI$, we have

$$|m| = \sqrt{m^2} = \sqrt{\frac{1}{3}\sigma^s : \sigma^s}, \quad \sqrt{3p} = \sqrt{\tfrac{1}{2}\textstyle\sum_{\text{i}j}(\sigma - mI)^2_{\text{i}j}} = \sqrt{\tfrac{1}{2}\sigma^{\text{d}} : \sigma^{\text{d}}}, \tag{3.15}$$

so that damage takes place when

$$\sigma^{\text{d}} : \sigma^{\text{d}} - \frac{2}{3}\kappa\sigma^s : \sigma^s > 0. \tag{3.16}$$

In order to compare our model with the models presented in Section 1.3, we rewrite the above quantity using the stress-strain relation $\sigma = \mathrm{a}(\alpha)\,(2\mu\varepsilon + \lambda\mathrm{tr}(\varepsilon)I)$, which yields

$$\sigma^{\mathrm{d}} = \mathrm{a}(\alpha)2\mu\varepsilon^{\mathrm{d}}, \tag{3.17}$$

$$\sigma^{s} = \mathrm{a}(\alpha)(2\mu + 3\lambda)\varepsilon^{s}, \tag{3.18}$$

so that

$$\sigma^{\mathrm{d}} : \sigma^{\mathrm{d}} = \mathrm{a}(\alpha)2\mu\sigma^{\mathrm{d}} : \varepsilon^{\mathrm{d}}, \tag{3.19}$$

$$\sigma^{s} : \sigma^{s} = \mathrm{a}(\alpha)(2\mu + 3\lambda)\sigma^{s} : \varepsilon^{s}. \tag{3.20}$$

These quantities can also expressed in terms of Young's modulus and Poisson's ratio according to the relations

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}. \tag{3.21}$$

The conditions for damage can thus be written, up to normalizing by a factor $\frac{1}{E}$ to compare with the models of Section 1.2.

$$\frac{1}{E}\left(\sigma^{\mathrm{d}} : \sigma^{\mathrm{d}} - \frac{2}{3}\kappa\sigma^{s} : \sigma^{s}\right) > 0. \tag{3.22}$$

We finally arrive at the form of the function $H(\varepsilon, \alpha)$ as the variation of the damage condition (3.22) with respect to α

$$H(\varepsilon, \alpha) = \frac{\partial}{\partial\alpha}\left(\frac{1}{E}\left(\sigma^{\mathrm{d}} : \sigma^{\mathrm{d}} - \frac{2}{3}\kappa\sigma^{s} : \sigma^{s}\right)\right). \tag{3.23}$$

For the case of two dimensions, the characteristic equation takes the following form

$$\det(\sigma - \lambda I) = \lambda^2 - \mathrm{tr}(\sigma) + \det(\sigma), \tag{3.24}$$

where the eigenvalues of σ are defined by

$$\lambda_1 = \frac{\mathrm{tr}(\sigma) + \sqrt{\mathrm{tr}(\sigma)^2 - 4\det(\sigma)}}{2}, \quad \lambda_2 = \frac{\mathrm{tr}(\sigma) - \sqrt{\mathrm{tr}(\sigma)^2 - 4\det(\sigma)}}{2}. \tag{3.25}$$

Then, following the Mohr's circle, we can assert that the largest shear component is obtained when

$$\frac{\lambda_1 - \lambda_2}{2} = \frac{1}{2}\sqrt{\mathrm{tr}(\sigma)^2 - 4\det(\sigma)}. \tag{3.26}$$

Considering $m = \frac{1}{2}\mathrm{tr}(\sigma)$, we have

$$|m| = \sqrt{m^2} = \sqrt{\frac{1}{2}\sigma^{s} : \sigma^{s}}, \quad \tfrac{1}{2}\sqrt{\mathrm{tr}(\sigma)^2 - 4\det(\sigma)} = \sqrt{\tfrac{1}{2}\textstyle\sum_{\mathrm{ij}}(\sigma - mI)^2_{\mathrm{ij}}} = \sqrt{\tfrac{1}{2}\sigma^{\mathrm{d}} : \sigma^{\mathrm{d}}}, \tag{3.27}$$

so, in two-dimensional case, damage takes place when

$$\sigma^{\mathrm{d}} : \sigma^{\mathrm{d}} - \kappa\sigma^{s} : \sigma^{s} > 0, \tag{3.28}$$

and the function $H(\varepsilon, \alpha)$ takes the following form

$$H(\varepsilon, \alpha) = \frac{\partial}{\partial\alpha}\left(\frac{1}{E}\left(\sigma^{\mathrm{d}} : \sigma^{\mathrm{d}} - \kappa\sigma^{s} : \sigma^{s}\right)\right). \tag{3.29}$$

3.2 New model to underground mining

In order to model the damage in the underground mining process and considering the pseudopotential of dissipation defined above, we propose the following evolution model for block caving that satisfy the following conditions

1. The stress tensor $\sigma(x,t) = \sigma(u(x,t), \alpha(x,t)) = \mathrm{a}(\alpha(x,t))A_0\varepsilon(u(x,t))$ satisfies the equilibrium equations

$$\begin{array}{rcllr} \operatorname{div}(\sigma(x,t)) + f(x,t) & = & 0 & \text{in} & \Omega, \\ \sigma(x,t)\cdot n & = & F(x,t) & \text{on} & \partial\Omega_F, \\ u(x,t) & = & U(x,t) & \text{on} & \partial\Omega_U. \end{array} \tag{3.30}$$

2. The damage field $\alpha(x,t)$ satisfies the nonlocal damage criterion in Ω

$$\frac{(\mathrm{a}^2(\alpha(x,t)))'}{2E}\left((A_0\varepsilon)^{\mathrm{d}} : (A_0\varepsilon)^{\mathrm{d}} - c(A_0\varepsilon)^s : (A_0\varepsilon)^s\right) + w'(\alpha(x,t)) - w_1\ell^2\Delta\alpha(x,t) \geq 0, \tag{3.31}$$

and the nonlocal consistency condition in Ω

$$\left(\frac{(\mathrm{a}^2(\alpha(x,t)))'}{2E}\left((A_0\varepsilon)^{\mathrm{d}} : (A_0\varepsilon)^{\mathrm{d}} - c(A_0\varepsilon)^s : (A_0\varepsilon)^s\right) + w'(\alpha(x,t)) - w_1\ell^2\Delta\alpha(x,t)\right)\dot{\alpha}(x,t) = 0, \tag{3.32}$$

where $c = \kappa$ in the two-dimensional case and $c = \frac{2}{3}\kappa$ in the three-dimensional case.

3.3 Numerical results

In this section we describe the numerical strategy to attack the underground mining problem and show synthetic 2D test cases where our new damage model is compared with other existing models describes in Section 1.2, and present different simulations in the sense of the underground mining.

3.3.1 Discretization and solution algorithm

The three models have been numerically implemented in two dimension, following [40]. Our numerical scheme uses an alternate minimization algorithm, which consists in solving a series of subproblems at each time step, to determine u when α is fixed, then to determine α at fixed u, until convergence.

The evolution is discretized in time. Given the displacement and the damage field $(u_{\mathrm{i}-1}, \alpha_{\mathrm{i}-1})$ at time step $t_{\mathrm{i}-1}$, the displacement u_{i} at time t_{i} is first obtained by solving the variational problem: Find u such that

$$\forall v \in \mathcal{C}_{t_{\mathrm{i}}}, \quad \int_\Omega \sigma(u, \alpha_{\mathrm{i}-1}) : \varepsilon(v)\mathrm{d}x = \int_\Omega f_{\mathrm{i}} \cdot v\mathrm{d}x + \int_{\partial\Omega_F} F_{\mathrm{i}} \cdot v\mathrm{d}S, \tag{3.33}$$

Subsequently, the field α_{i} is determined as a solution to the following bound-constrained minimization problem

$$\inf\left\{\overline{\mathcal{P}}(u_{\mathrm{i}}, \alpha) : \alpha \in \mathcal{D}_t(\alpha_{\mathrm{i}-1})\right\}, \tag{3.34}$$

where the functional $\overline{\mathcal{P}}(u,\alpha)$ is defined for the three different models by
Gradient damage model:

$$\overline{\mathcal{P}}(u,\alpha) = \int_\Omega \frac{1}{2}\mathrm{a}(\alpha)A_0\varepsilon(u) : \varepsilon(u) + w(\alpha) + \frac{1}{2}w_1\ell^2|\nabla\alpha|^2, \tag{3.35}$$

Gradient damage model for shear fracture:

$$\overline{\mathcal{P}}(u,\alpha) = \int_\Omega \left(\lambda + \frac{2\mu}{n}\right)\frac{\mathrm{tr}(\varepsilon(u))^2}{2} + \mathrm{a}(\alpha)\mu\varepsilon^{\mathrm{d}} : \varepsilon^{\mathrm{d}} + w(\alpha) + \frac{1}{2}w_1\ell^2|\nabla\alpha|^2, \tag{3.36}$$

Shear-compression damage model:

$$\overline{\mathcal{P}}(u,\alpha) = \int_\Omega \frac{\mathrm{a}^2(\alpha)}{2E}\Big((A_0\varepsilon(u))^{\mathrm{d}} : (A_0\varepsilon(u))^{\mathrm{d}} - \kappa\,(A_0\varepsilon(u))^{s} : (A_0\varepsilon(u))^{s}\Big) + w(\alpha) + w_1\ell^2|\nabla\alpha|^2. \tag{3.37}$$

The unilateral constraint $\alpha(x) \geq \alpha_{\mathrm{i}-1}$ is the time-discrete version of the irreversibility of damage Considering this way of calculating the damage, we can include the reversibility of the problem in the equations. The solution strategy is summarized in Algorithm 3 and implemented in FEniCS, where the codes can be seen in the Appendix B.1.

Algorithm 1 Numerical algorithm to solve the damage problem

```
1: **return** Solution of time step $t_{\mathrm{i}}$.
2: Given $(u_{\mathrm{i}-1}, \alpha_{\mathrm{i}-1})$, the sate at the previous loading step.
3: Set $(u^{(0)}, \alpha^{(0)}) := (u_{\mathrm{i}-1}, \alpha_{\mathrm{i}-1})$ and $\mathrm{error}^{(0)} = 1.0$
4: **while** $\mathrm{error}^{(p)} >$ tolerance **do**
5:     Solve $u^{(p)}$ from (3.33) with $\alpha^{(p-1)}$.
6:     Find $\alpha^{(p)} := \underset{\alpha\in\mathcal{D}(\alpha_{\mathrm{i}-1})}{\arg\min}\,\overline{\mathcal{P}}(u^{(p)},\alpha)$.
7:     $\mathrm{error}^{(p)} = \|\alpha^{(p-1)} - \alpha^{(p)}\|_\infty$.
8: **end while**
9: Set $(\mathbf{u}_{\mathrm{i}}, \alpha_{\mathrm{i}}) = (u^p, \alpha^p)$.
```

3.3.2 Influence of the cavity in the damage model

For modeling the block caving process, we consider a domain that simulates a rock mass in two dimensions $\Omega_0 \subseteq \mathbb{R}^2$ with $\partial\Omega_0 = \Gamma_{lat} \cup \Gamma_{up} \cup \Gamma_{down}$ denoting the lateral, upper and lower bounded of the domain respectively. We consider a time depending interior domain $S(t_{\mathrm{i}}) \subseteq \Omega_0$, with $\partial S(t_{\mathrm{i}}) = \Gamma_{cav}(t_{\mathrm{i}})$ representing the interior cavity of this rock mass, where we assume free boundary conditions in $\Gamma_{cav}(t_{\mathrm{i}})$, that is, $\sigma \cdot n = 0$ in $\Gamma_{cav}(t_{\mathrm{i}})$. Finally, the domain to consider the block caving process is $\Omega(t_{\mathrm{i}}) = \Omega_0 \setminus S(t_{\mathrm{i}})$. Figure 3.1 shows a sketch of the domain defined above with the boundary conditions used in these problems. The loading is given by the gravity defined by $f_t = \rho g$, with $\rho = 2.7 \cdot 10^3$ and $g = (0.0, -9.8)$.

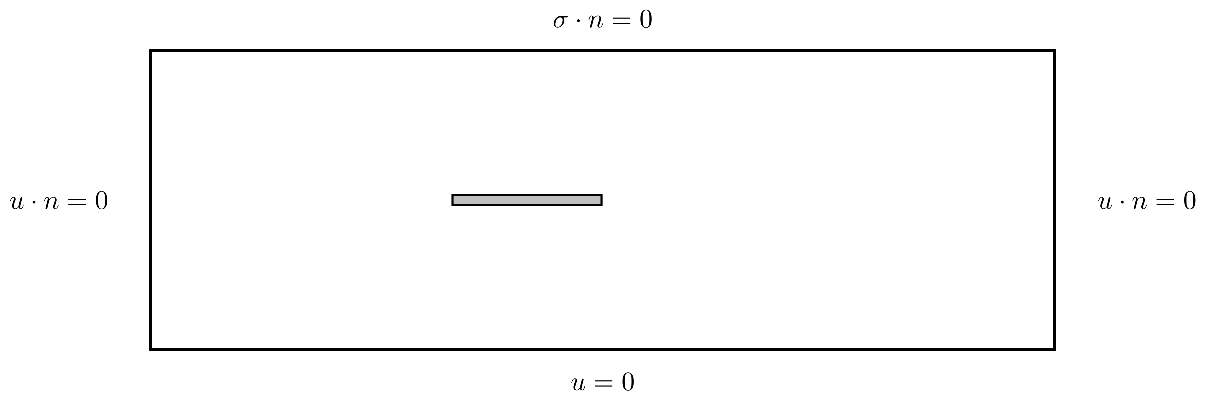


Figure 3.1: Geometry and boundary condition for the cavity problem.

In our tests, we consider that the rock mass is defined by $\Omega_0 = (-1500, 1500) \times (-500, 500)$ and the cavities are represented by $S(t) = (-500, -500 + 40t_\mathrm{i}) \times (-20, 20)$. All simulations presented here use the following values for material parameters

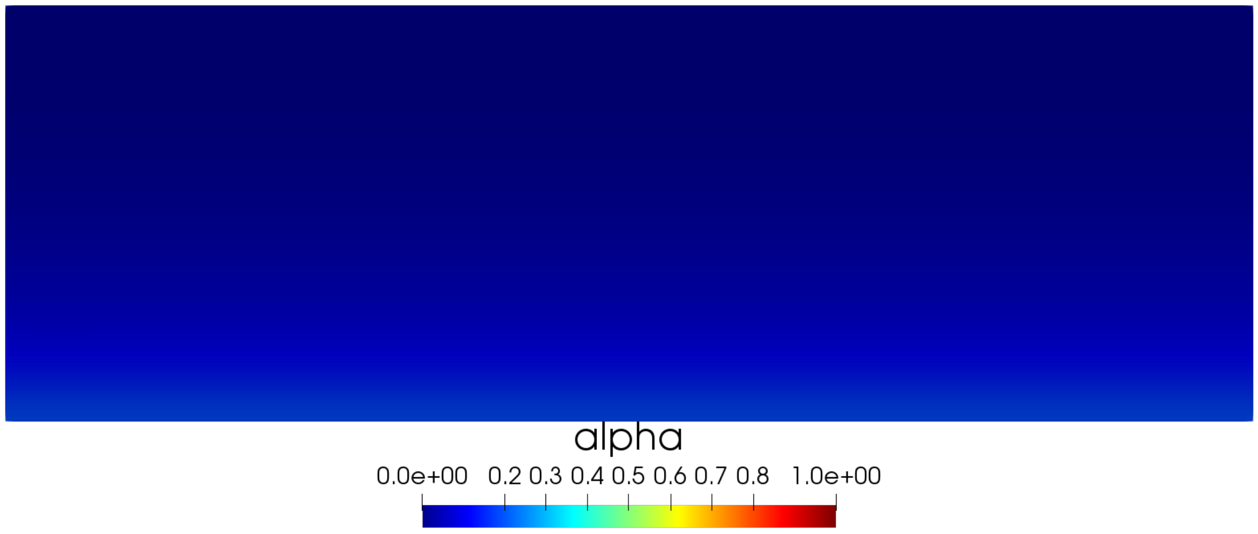
$$E = 2.9 \cdot 10^{10}, \quad \kappa = 1.0, \quad \text{and} \quad \nu = 0.3, \tag{3.38}$$

where we use a quadratic damage model defined by

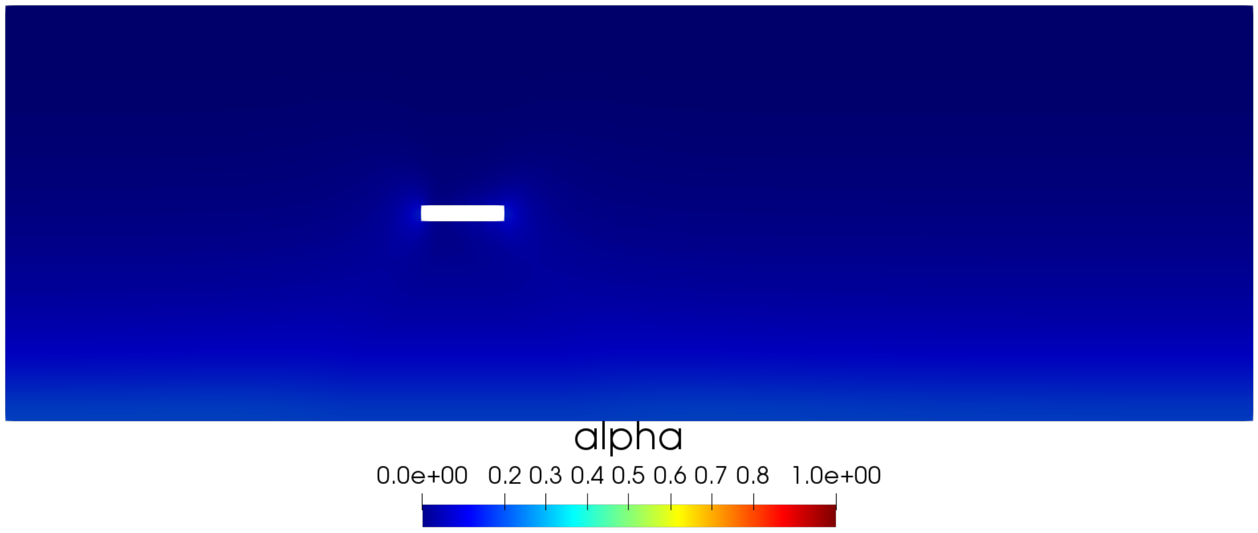
$$\mathrm{a}(\alpha) = (1-\alpha)^2, \quad w(\alpha) = w_1 \alpha^2. \tag{3.39}$$

Gradient damage model

In the first test case, we consider the gradient damage model described the Section 1.3.1. Figures 3.2 3.3 summarize the results obtained with this model. The images display the evolution of damage when the cavity advances in time. We chose $w_1 = 10^5$. We observe that the damage appears in whole the domain and it is distributed in an instant of time (the same phenomenon can be seen when considering values smaller of w_1). This indicates that in this example, damage is essentially triggered by compression due to the gravity forces, which is the main forcing term in the model and which is uniformly distributed in the domain. Damage does not seem to be very sensitive to the presence of the cavity.

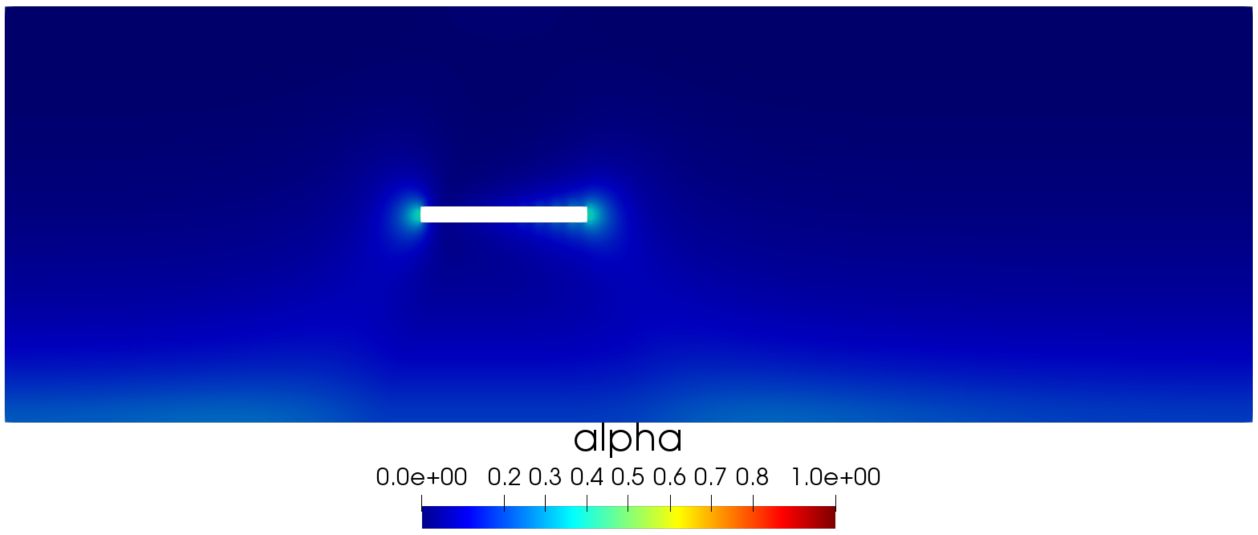


(a) $t_{\mathrm{i}} = 0$

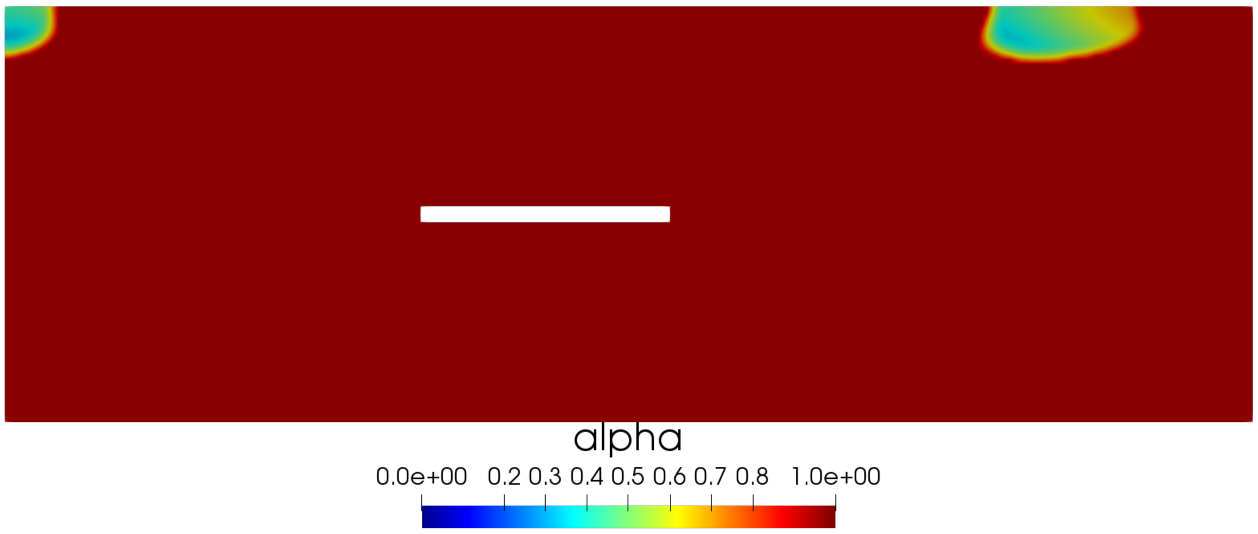


(b) $t_{\mathrm{i}} = 5$

Figure 3.2: Damage field distribution in the rock mass for $w_1 = 10^5$.



(a) $t_{\mathrm{i}} = 10$



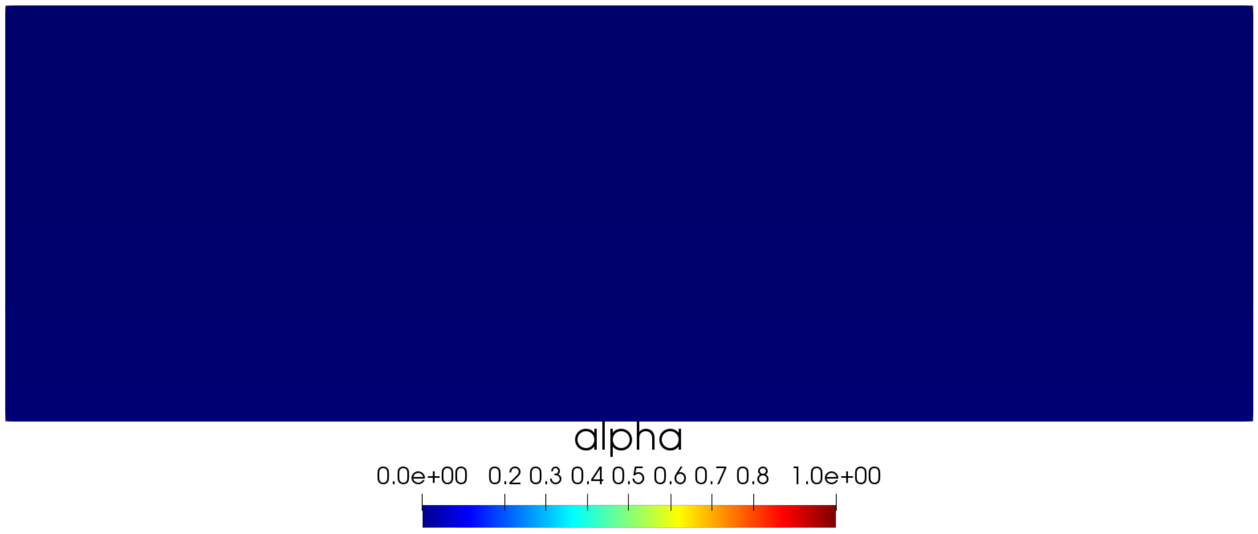
(b) $t_{\mathrm{i}} = 15$

Figure 3.3: Damage field distribution in the rock mass for $w_1 = 10^5$.

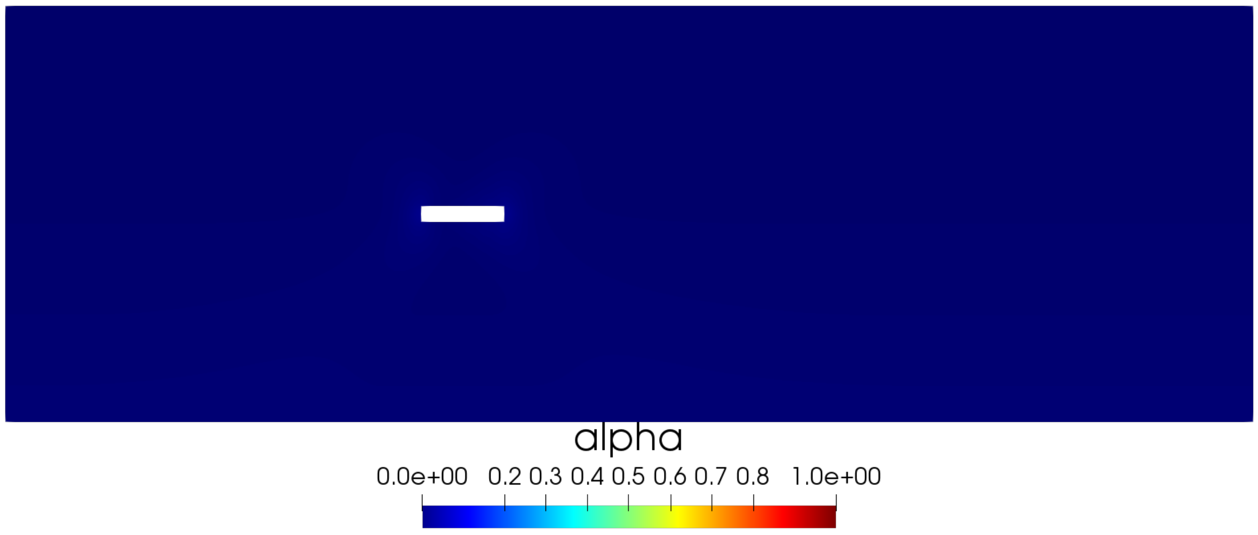
We observe that on the first time steps, the damage begins to appear in the lower area of the domain, below the extraction cavity, and later distributes throughout the domain. Such qualitative behavior is not consistent with the expected mechanism of fracking produced by block caving.

Gradient damage model for shear fracture

In the second test case, we consider the gradient damage model for shear fracture given in Section 1.3.2. Figures 3.4-3.5 display the evolution of the damage when the cavity advances in time. Again, $w_1 = 10^5$. Here, damage hardly appears, and takes values close to zero around the cavity.

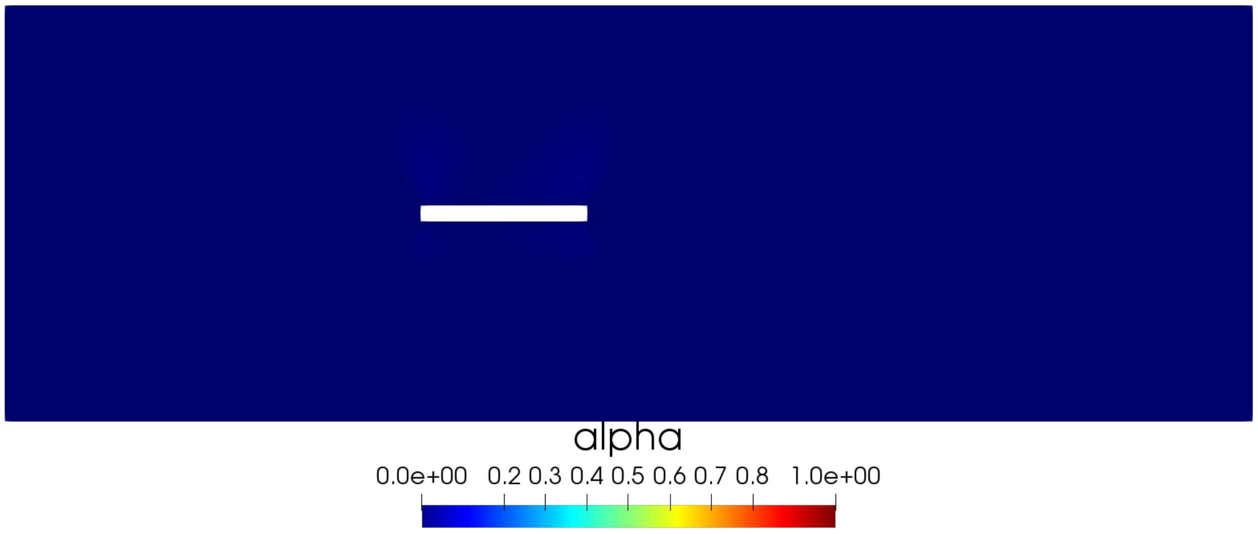


(a) $t_\mathrm{i} = 0$

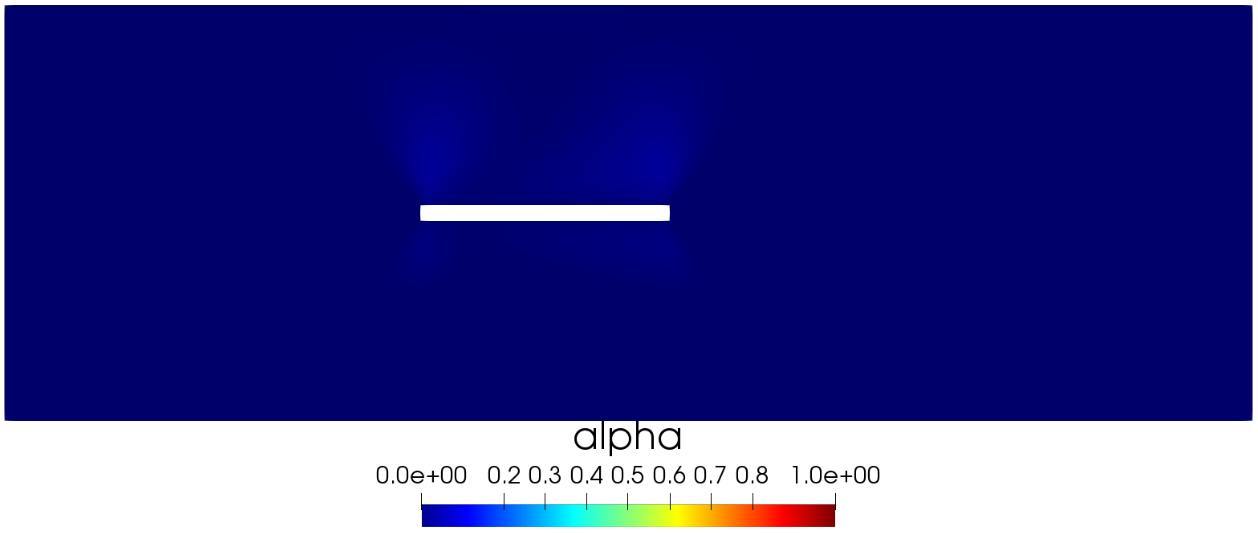


(b) $t_\mathrm{i} = 5$

Figure 3.4: Damage field distribution in the rock mass for $w_1 = 10^5$.



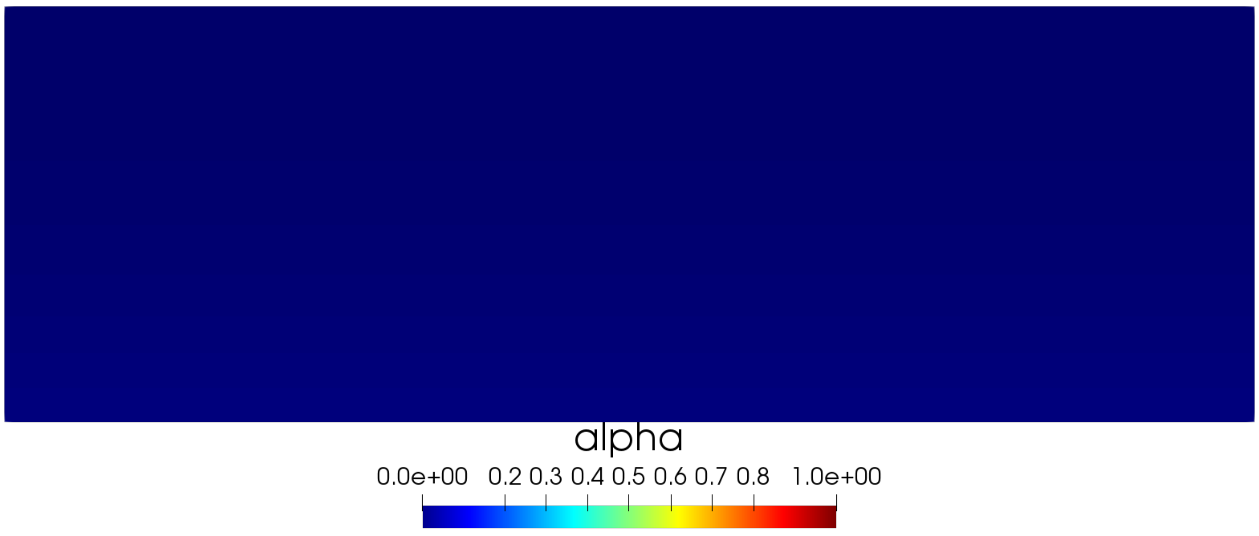
(a) $t_{\mathrm{i}} = 10$



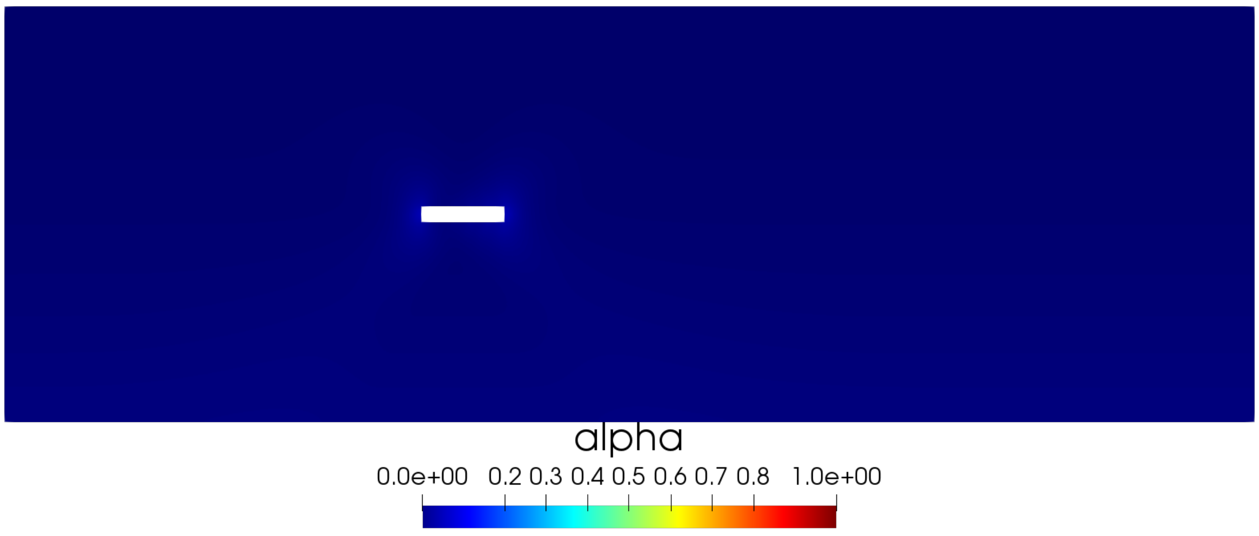
(b) $t_{\mathrm{i}} = 15$

Figure 3.5: Damage field distribution in the rock mass for $w_1 = 10^5$.

For smaller values of w_1 (see Figures 3.6-3.7 and Figures 3.8-3.9 for $w_1 = 5 \cdot 10^4$ and $w_1 = 10^4$) damage remains distributed around the cavity. When $w_1 = 5 \cdot 10^4$ begins to appear around the end corners of the cavity (where stresses are expected to blow up, due to corner singularities). On the other hand when $w_1 = 10^4$ damage begins to appear in the bottom of the rock mass. Again these results do not represent the expected effect of block caving.

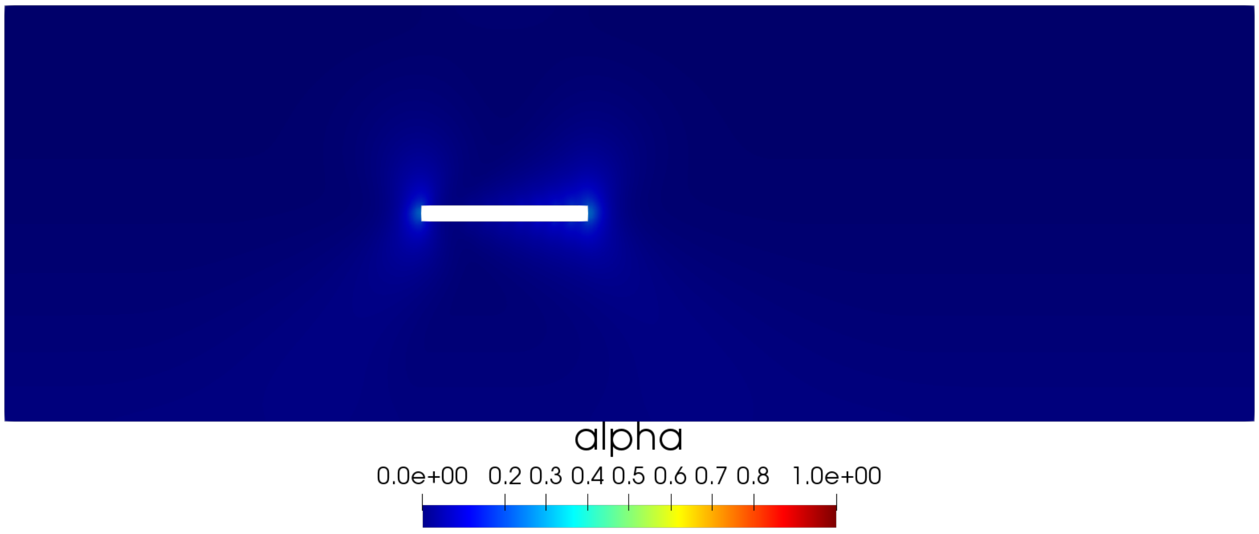


(a) $t_{\mathrm{i}} = 0$

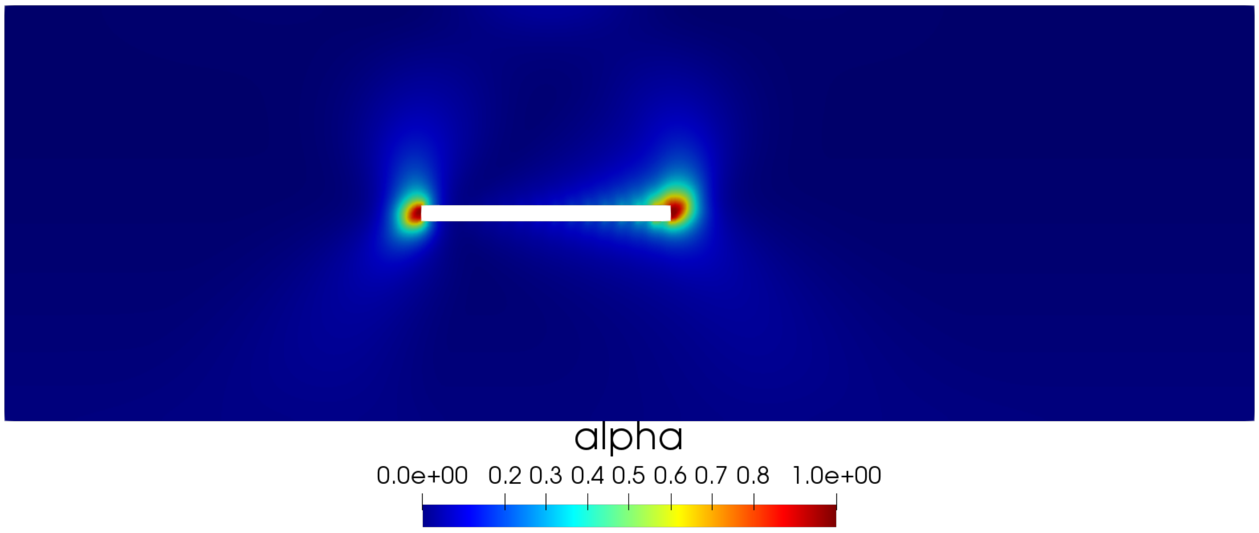


(b) $t_{\mathrm{i}} = 5$

Figure 3.6: Damage field distribution in the rock mass for $w_1 = 5 \cdot 10^4$

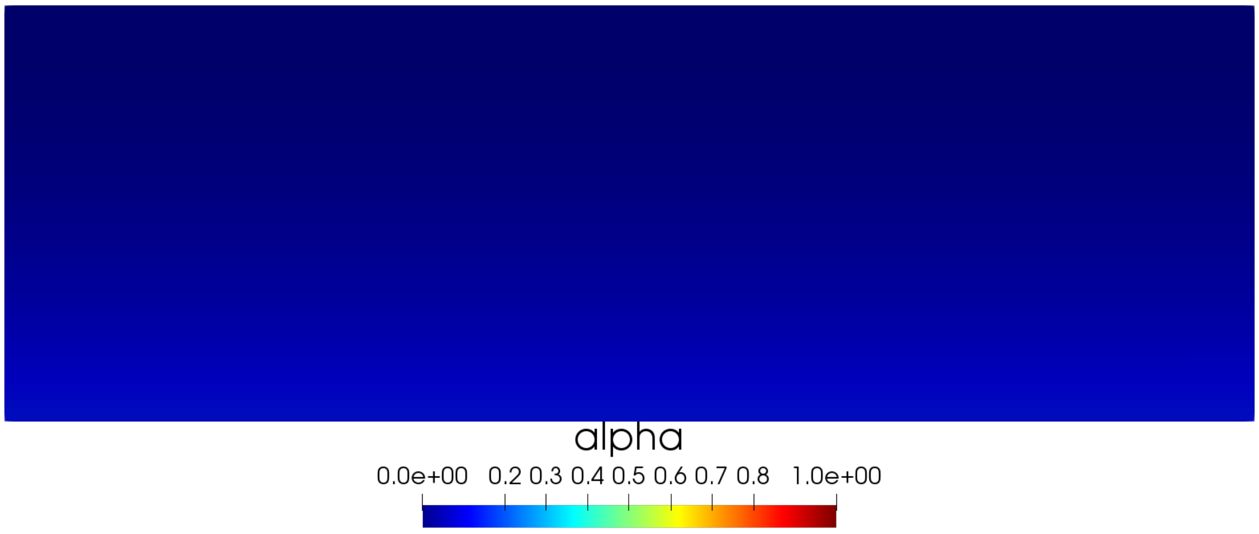


(a) $t_\mathrm{i} = 10$

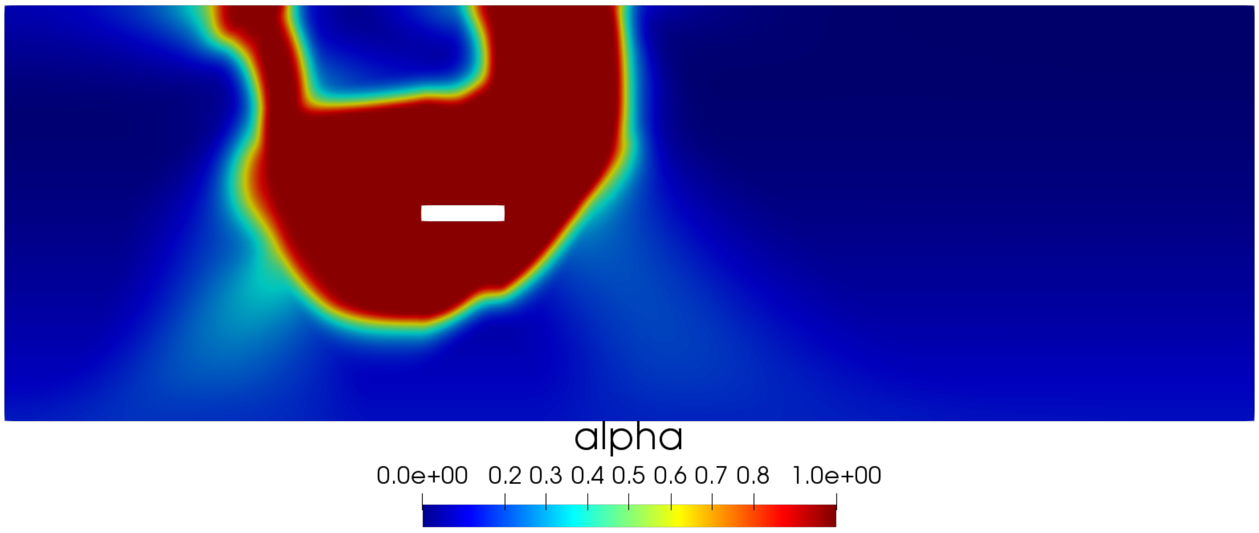


(b) $t_\mathrm{i} = 15$

Figure 3.7: Damage field distribution in the rock mass for $w_1 = 5 \cdot 10^4$

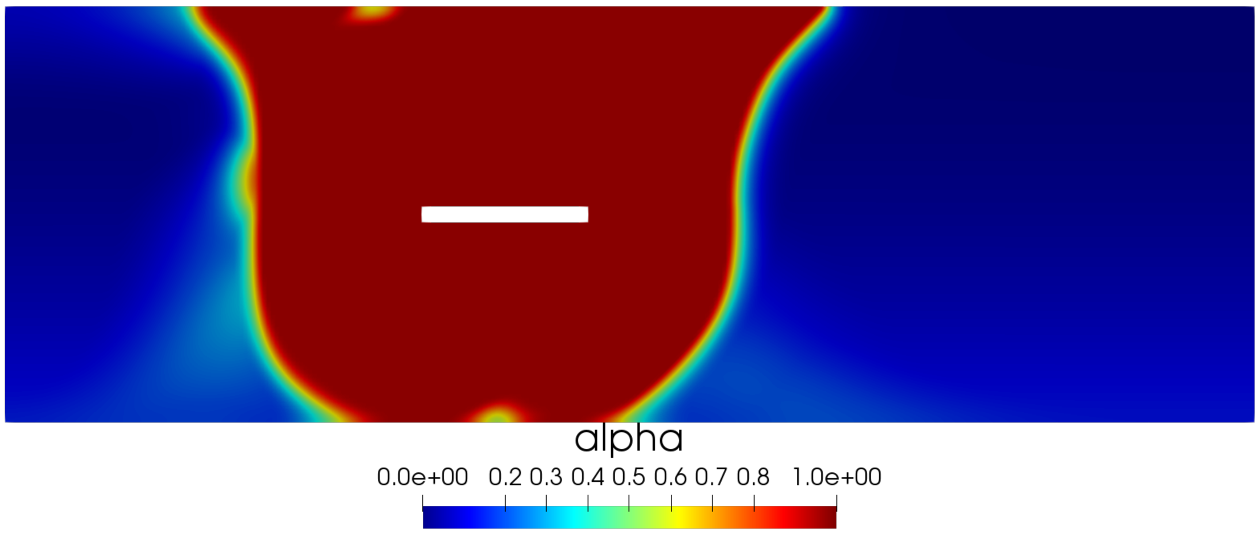


(a) $t_{\mathrm{i}} = 0$

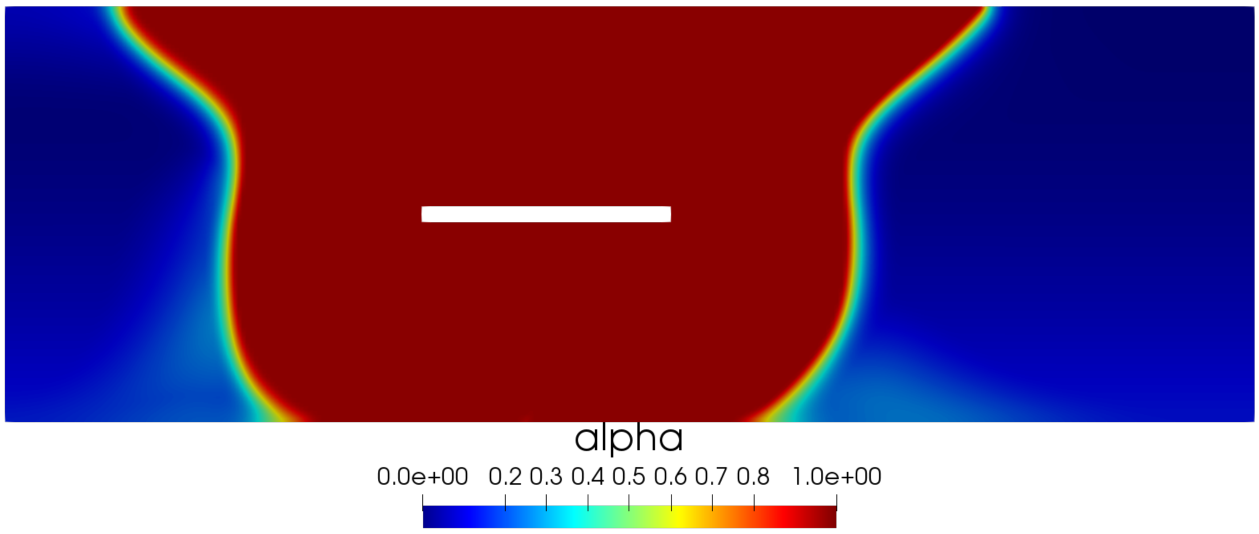


(b) $t_{\mathrm{i}} = 5$

Figure 3.8: Damage field distribution in the rock mass for $w_1 = 10^4$



(a) $t_{\mathrm{i}} = 10$

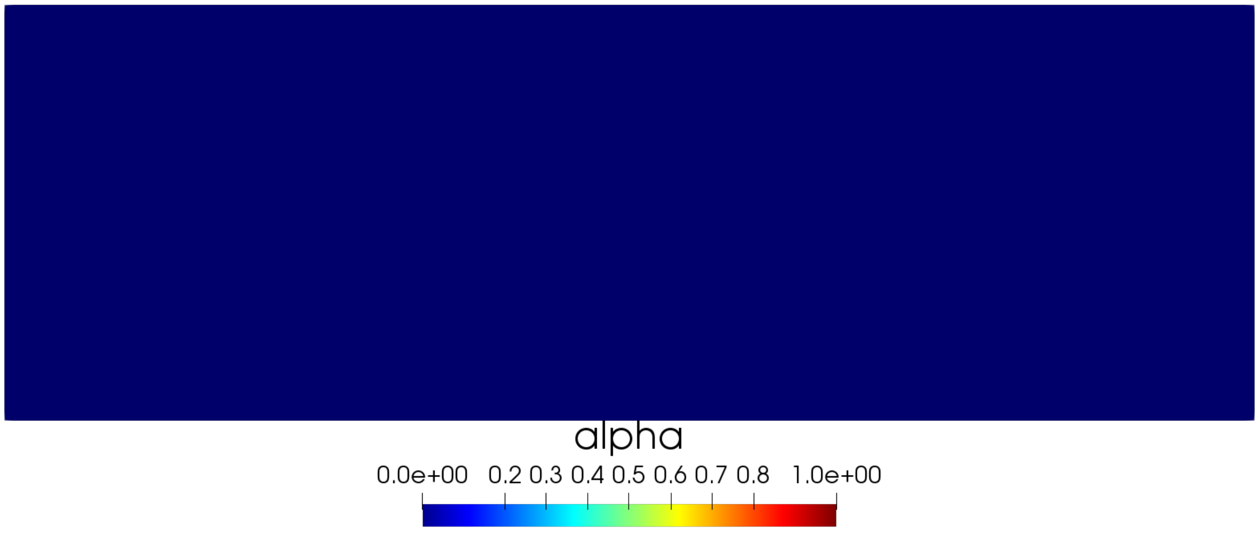


(b) $t_{\mathrm{i}} = 15$

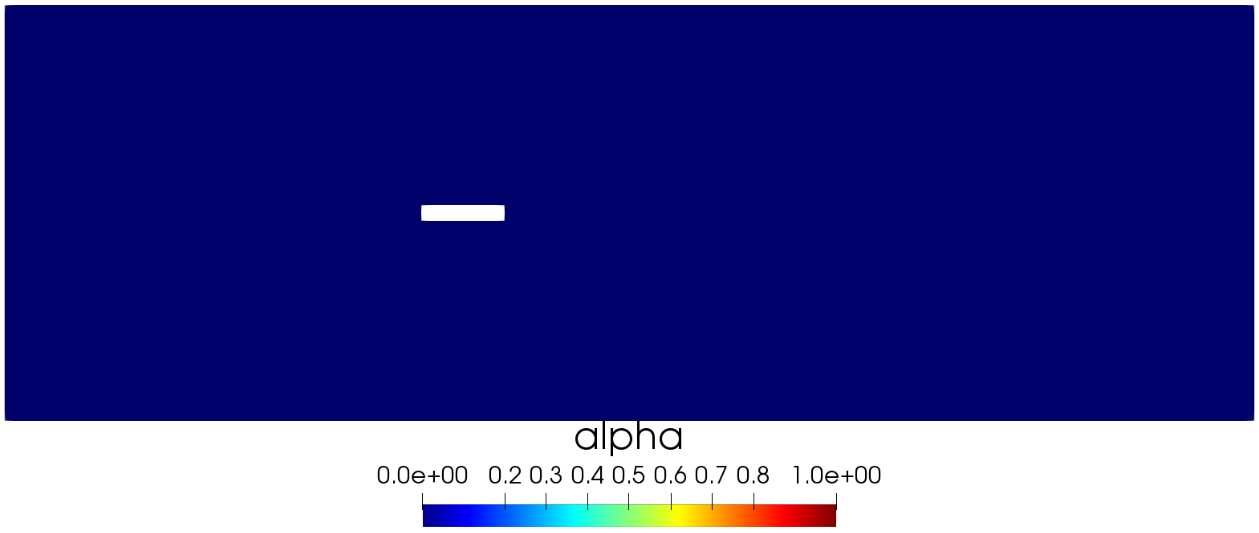
Figure 3.9: Damage field distribution in the rock mass for $w_1 = 10^4$

Shear-compression damage model

In this test case, we consider our model for the damage problem presented in Section 3.1. In Figures 3.10-3.11, the evolution of damage is displayed as the cavity advances in time, for the choice $w_1 = 10^5$. We observe that the level of damage is lower that produced by the gradient damage model, and it is mostly localized around the cavity.

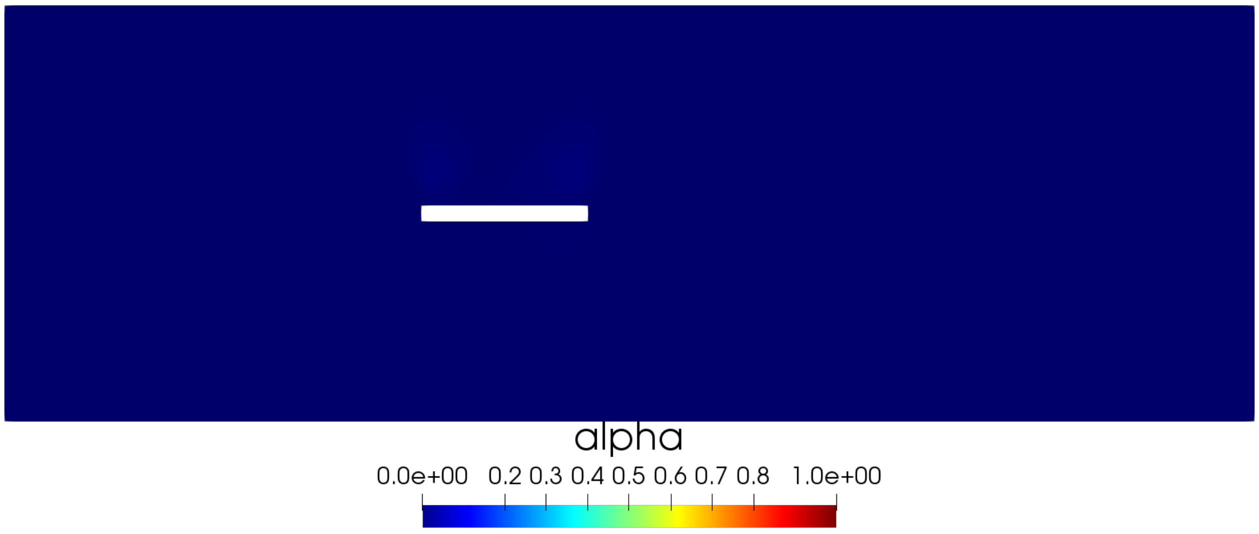


(a) $t_{\mathrm{i}} = 0$

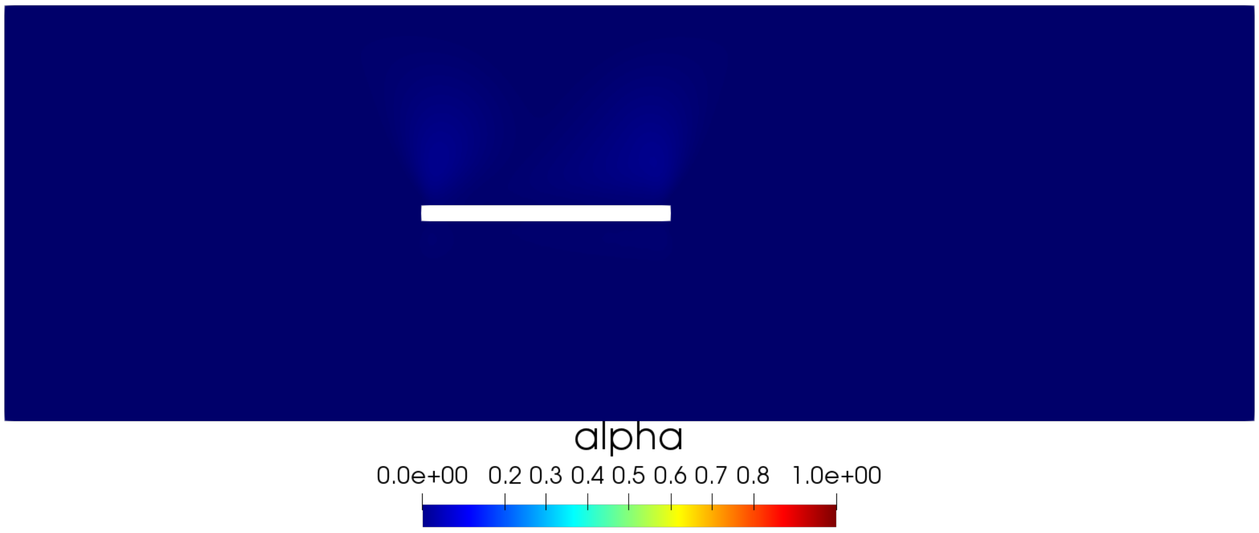


(b) $t_{\mathrm{i}} = 5$

Figure 3.10: Damage field distribution in the rock mass for $w_1 = 10^5$



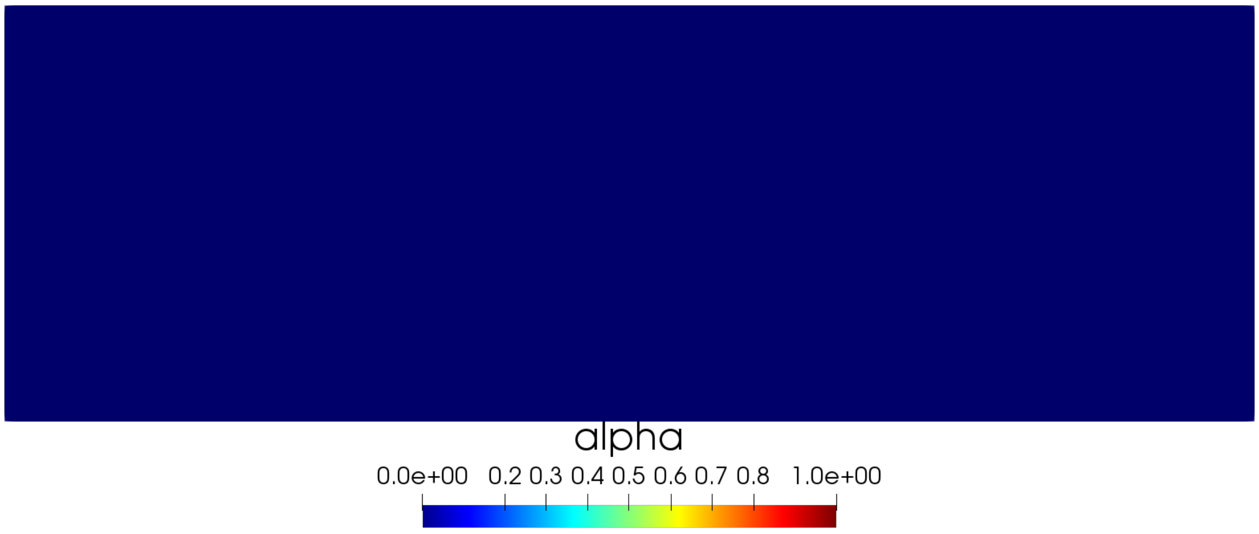
(a) $t_{\text{i}} = 10$



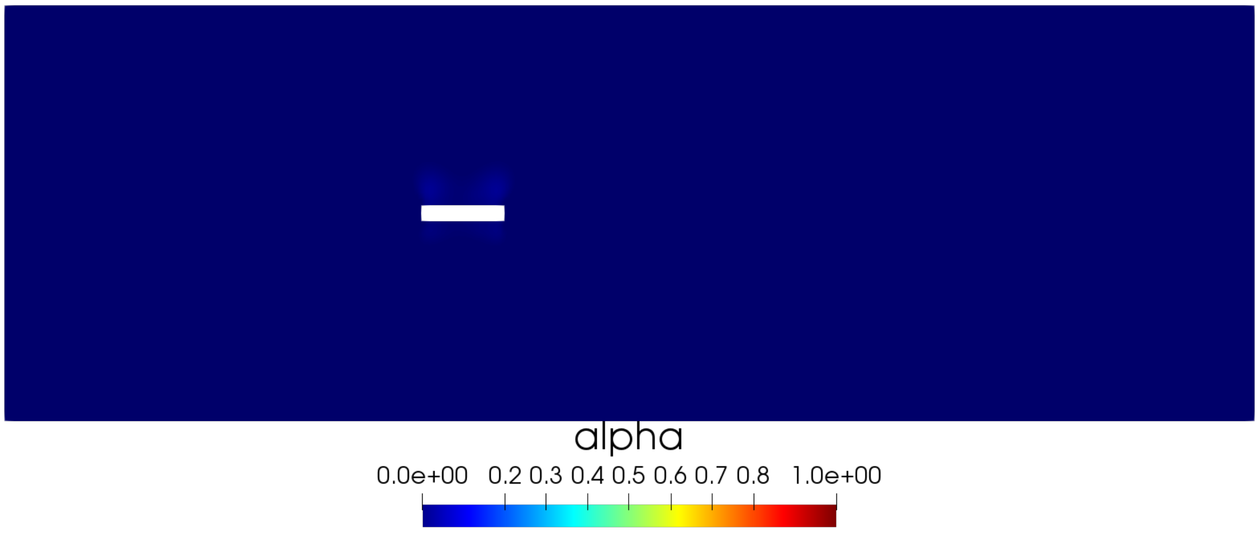
(b) $t_{\text{i}} = 15$

Figure 3.11: Damage field distribution in the rock mass for $w_1 = 10^5$

Figures 3.12-3.13 and 3.14-3.15 show the evolution of damage for $w_1 = 10^4$ and $w_1 = 10^3$ respectively. Damage remains localized around the cavity. As the latter advances in time, damage gets distributed above the ceiling of the cavity, consistently with what is expected in block caving.

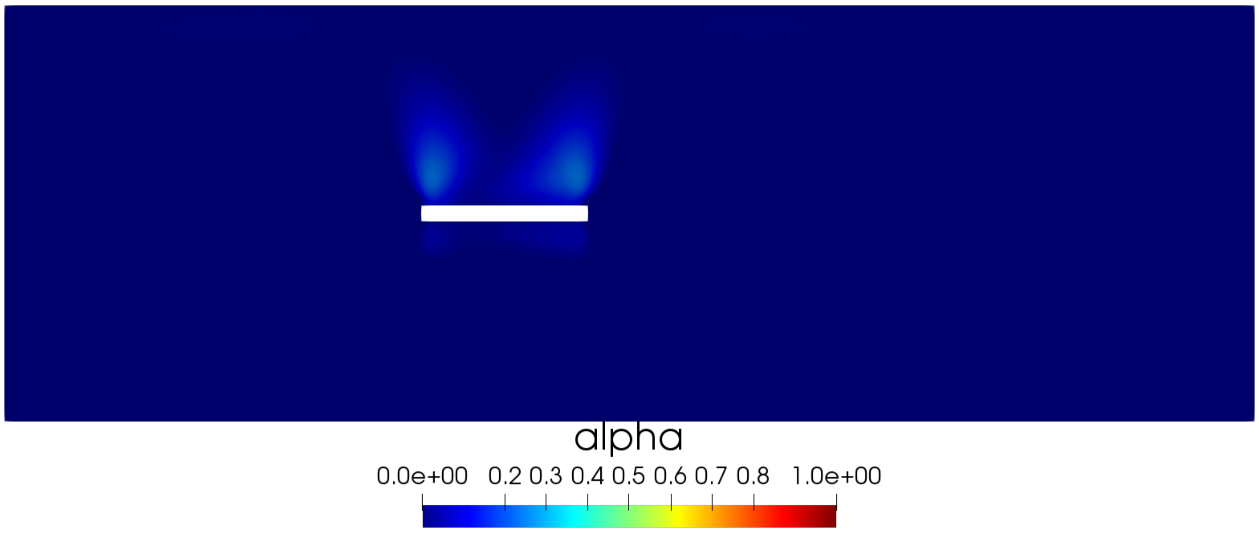


(a) $t_{\mathrm{i}} = 0$

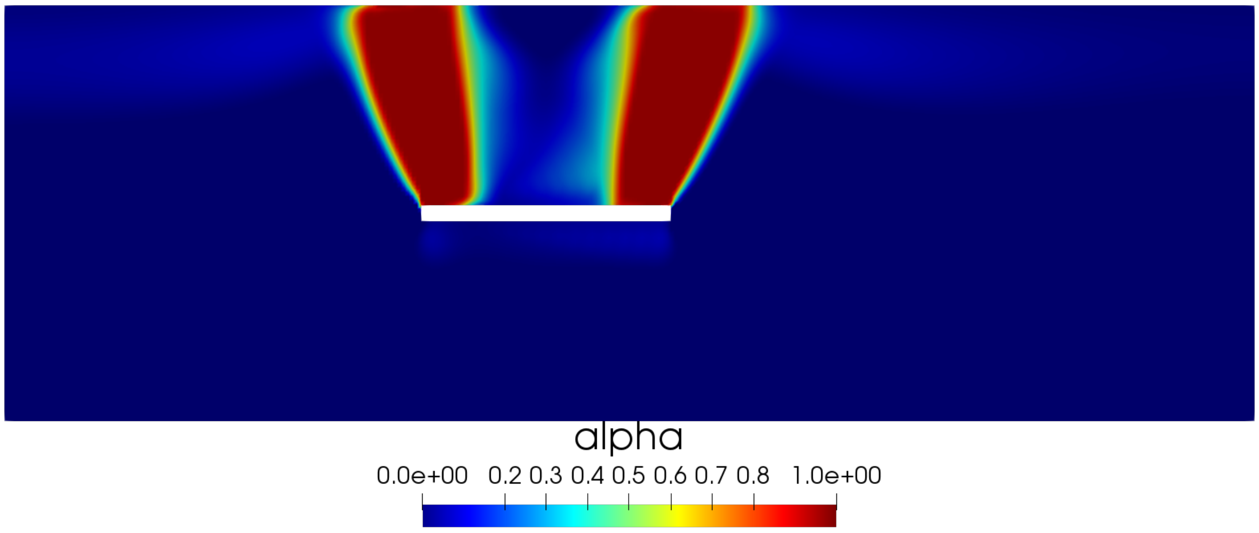


(b) $t_{\mathrm{i}} = 5$

Figure 3.12: Damage field distribution in the rock mass for $w_1 = 10^4$.

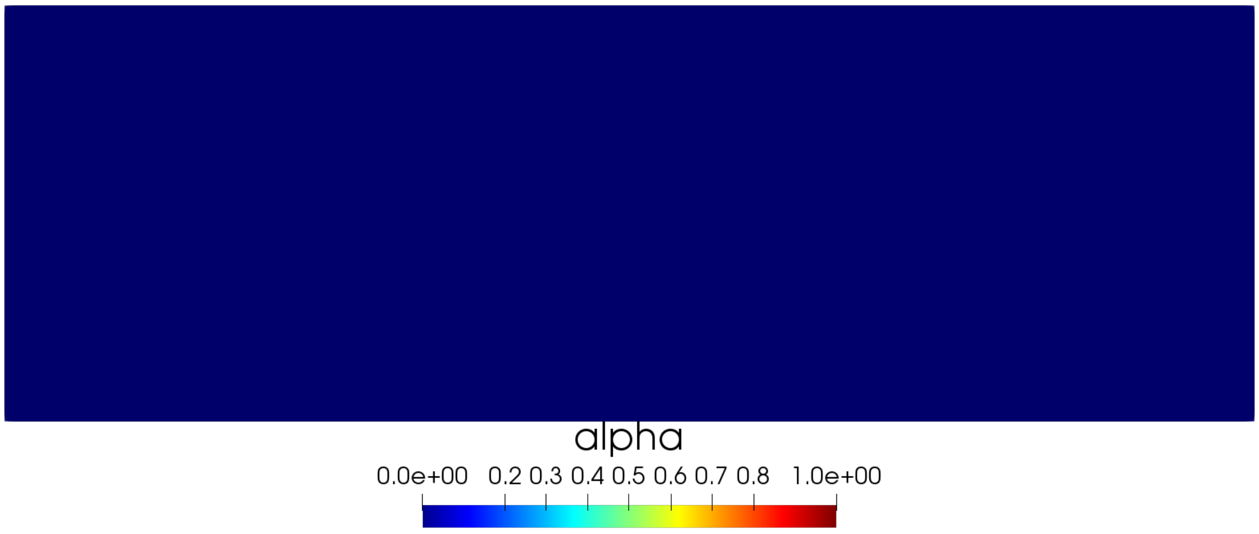


(a) $t_{\mathrm{i}} = 10$

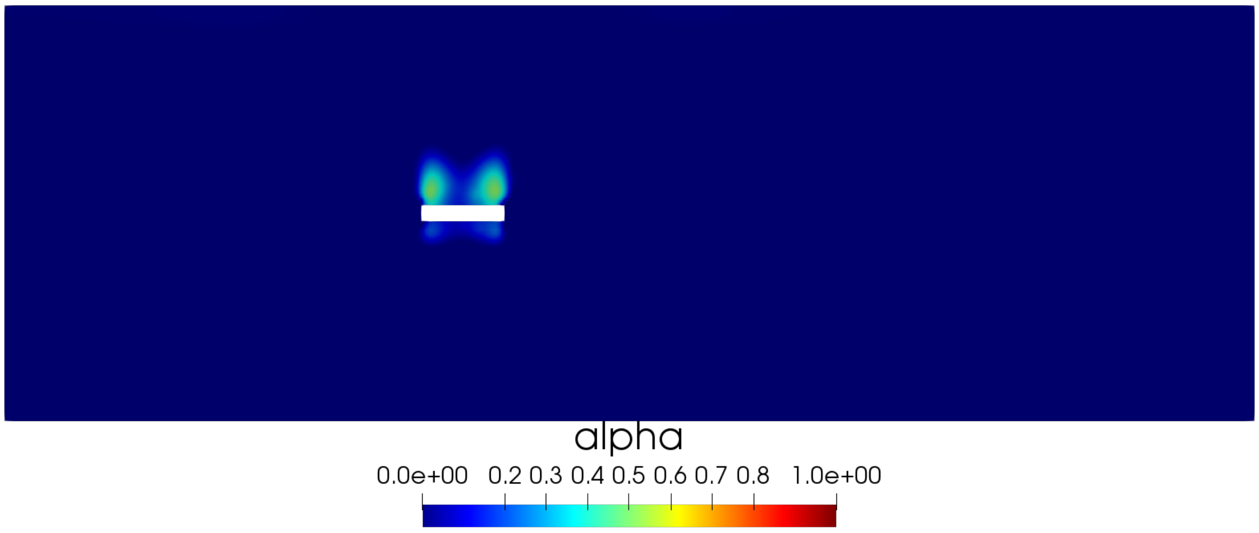


(b) $t_{\mathrm{i}} = 15$

Figure 3.13: Damage field distribution in the rock mass for $w_1 = 10^4$.

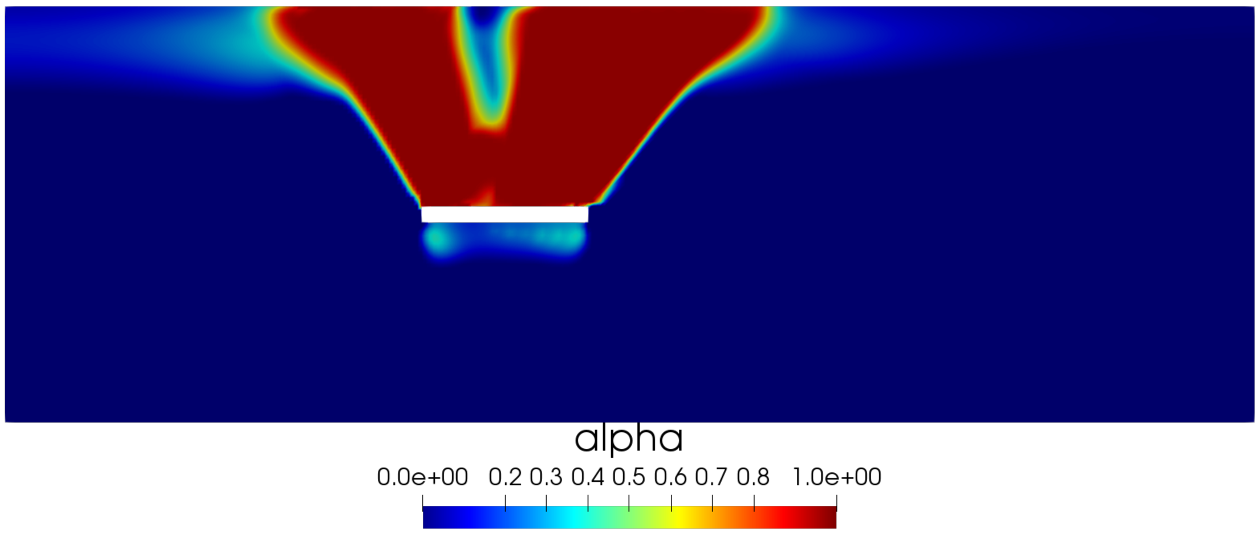


(a) $t_{\mathrm{i}} = 0$

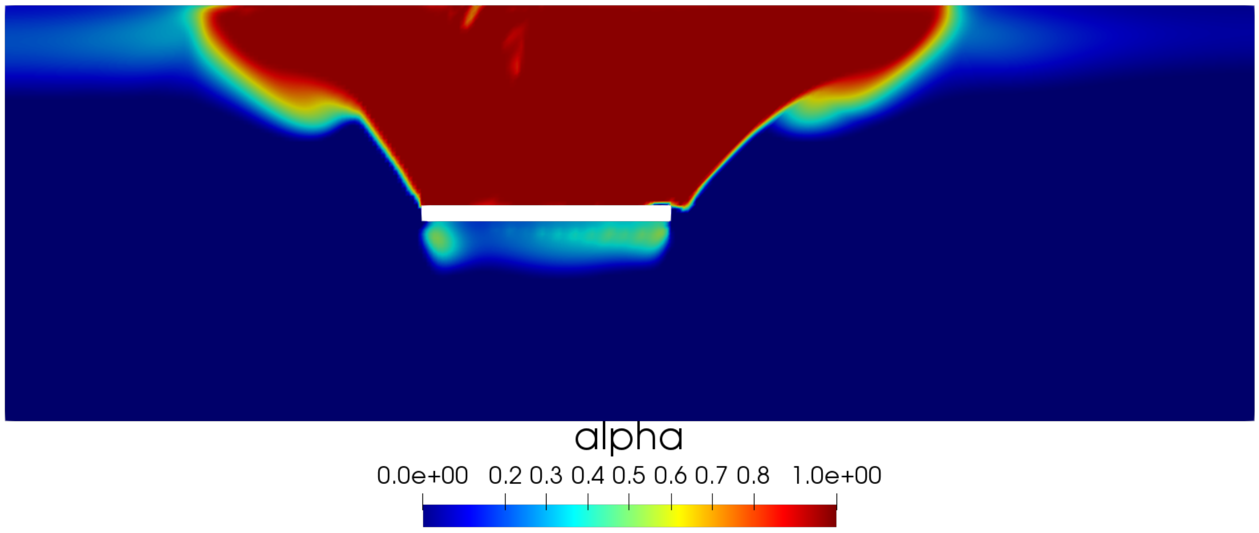


(b) $t_{\mathrm{i}} = 5$

Figure 3.14: Damage field distribution in the rock mass for $w_1 = 10^3$.



(a) $t_{\mathrm{i}} = 10$



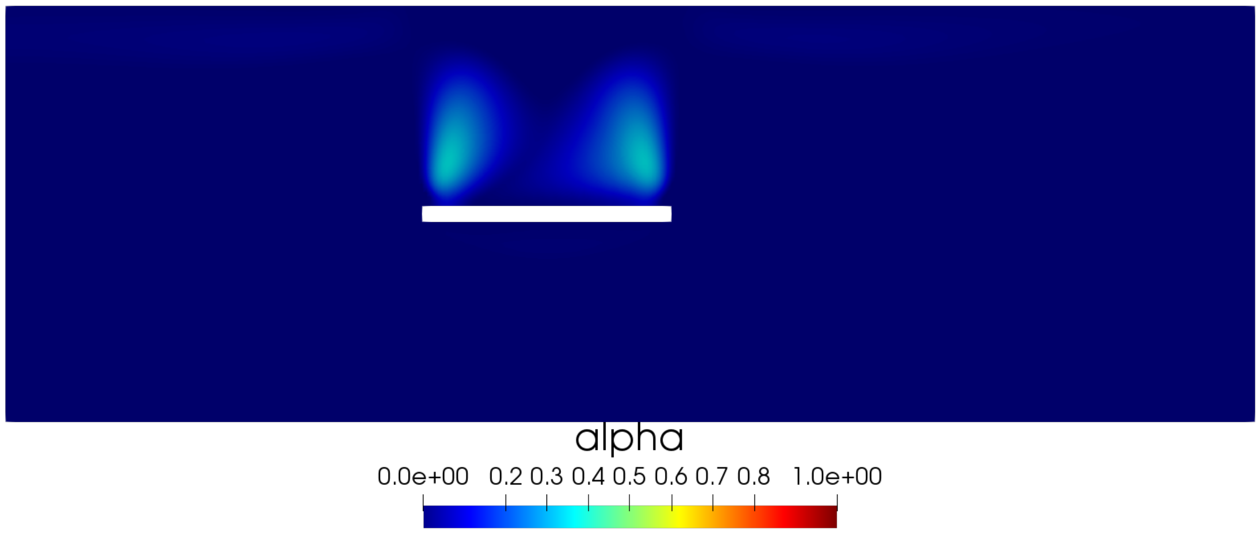
(b) $t_{\mathrm{i}} = 15$

Figure 3.15: Damage field distribution in the rock mass for $w_1 = 10^3$.

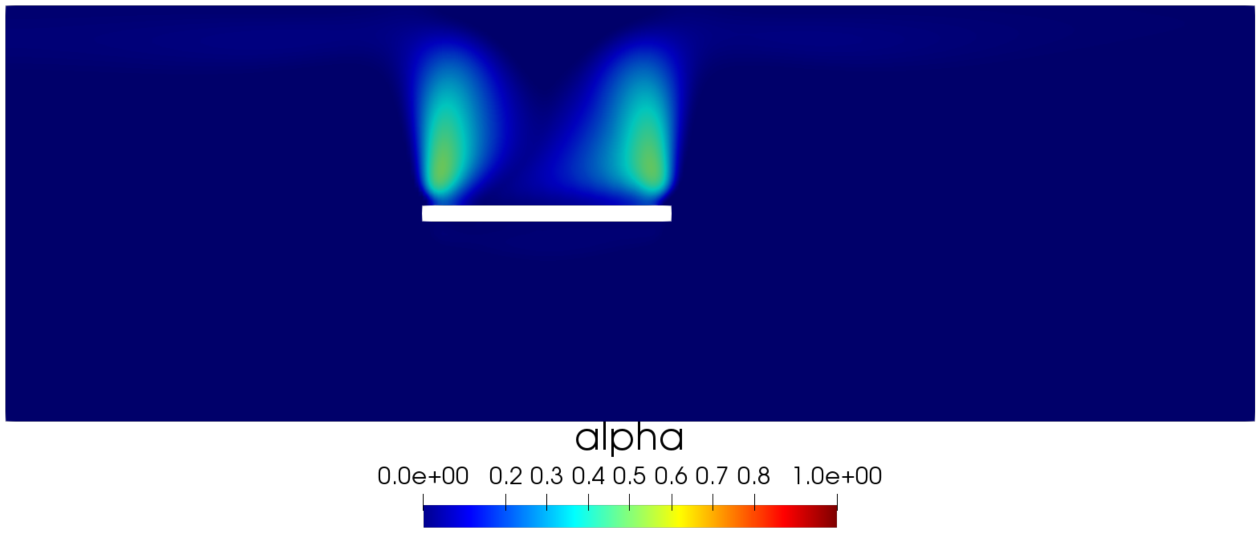
Sensitivity analysis

As seen in the damage criterion (3.31), the parameter κ plays a fundamental role in the propagation of damage in the rock mass, as it controls the contribution of the spherical and deviatoric part of the stress tensor in the damage criterion.

Figures 3.16-3.17 display results for several values of κ. As this parameter decreases, damage gets distributed throughout the whole domain, starting from the ceiling of the cavity to the upper surface of the rock mass. When κ becomes smaller than 1, damage appears in the bottom of the rock mass and invades the whole domain. This is consistent with the fact that for such values of κ, the damage criterion is less to the deviatoric part of the stresses, whereas the main forcing term is compressive. On the other hand, larger values of κ emphasize the deviatoric component of the stress in the damage criterion and privilege shear forces.

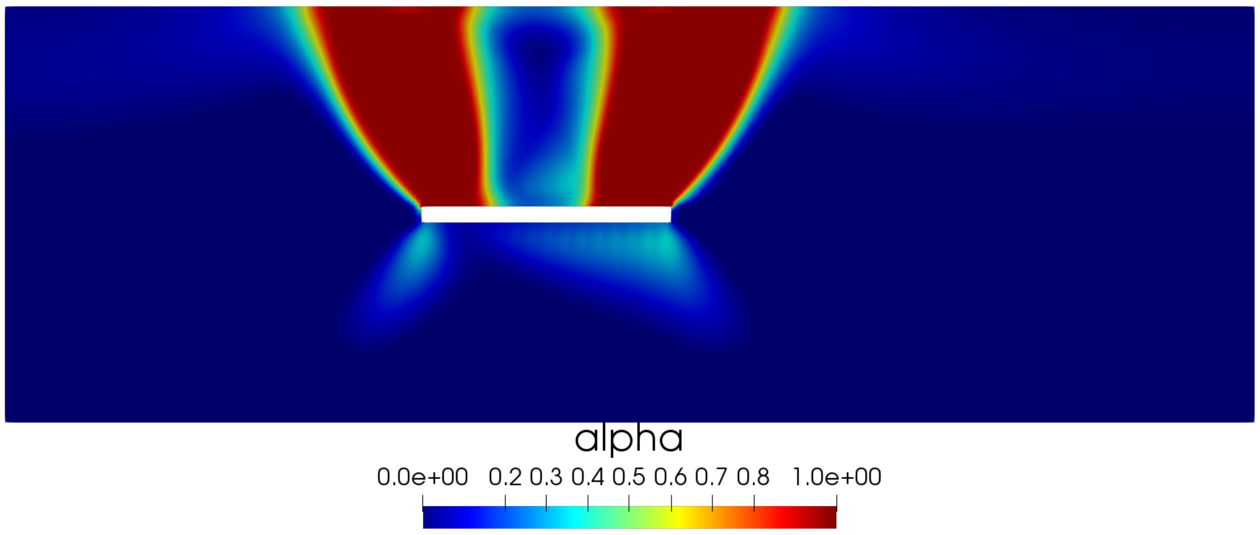


(a) $\kappa = 2.0$

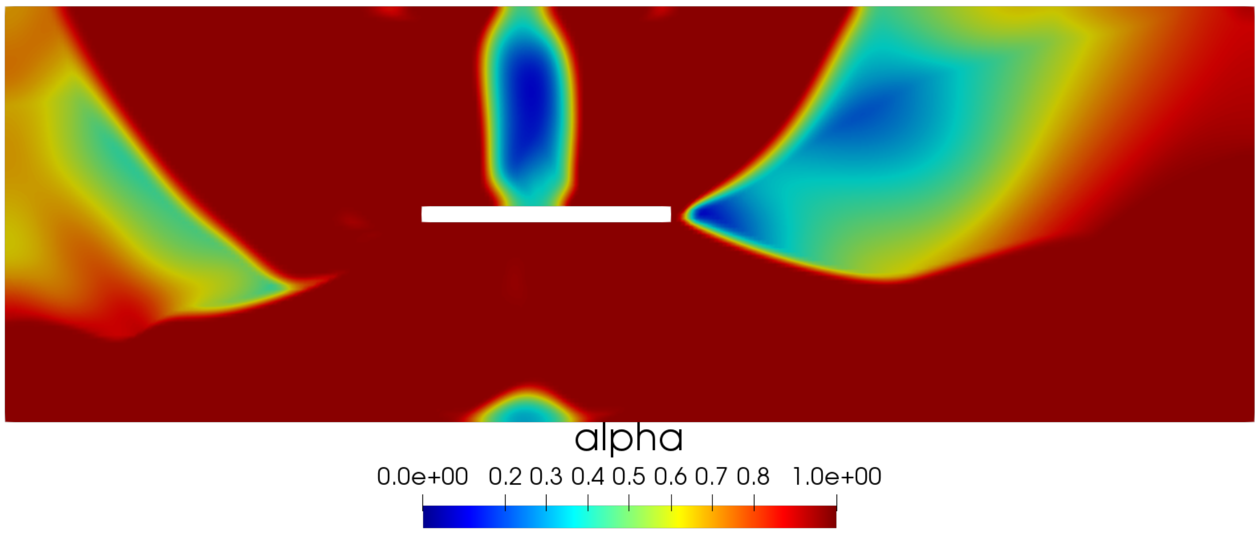


(b) $\kappa = 1.5$

Figure 3.16: Damage field distribution in the rock mass for $w_1 = 10^4$, with $t_\mathrm{i} = 15$.



(a) $\kappa = 0.5$



(b) $\kappa = 0.2$

Figure 3.17: Damage field distribution in the rock mass for $w_1 = 10^4$, with $t_i = 15$.

The results seen above show the importance of parameter κ in our model. In a these examples, one sees that, for a smaller than 1 values of κ, the spherical component of the stress tensor predominates over the deviatoric component and with this the damage is damage is mainly caused by compression causing damage to appear in the bottom of the domain. On the other hand, for a bigger values of the parameter κ the deviatoric component of the stress tensor predominates over the spherical part and with this the damage is mainly produced by the shear forces effect.

Chapter 4

Numerical improve for block caving process

In this chapter we proposed a fast numerical algorithm to solve the damage problem in underground mining.

4.1 Hardening properties analysis

In order to have a better approximation of impact of the underground mining in the rock mass, it is necessary to model different materials that can represent the different types of rocks found in mining deposits. In this section we will study different numerical results to analyze the different combinations of $A(\alpha)$ and $w(\alpha)$. To define the different models, we will study the Strain Hardening and Stress Softening conditions.

4.1.1 Hardening properties

For a homogeneous damage distribution α, let us define the elastic domains in the strain and stress spaces by

$$\mathcal{R}(\alpha) = \left\{\varepsilon \in \mathbb{M}_s : \frac{1}{2}\mathrm{a}'(\alpha)A_0\varepsilon : \varepsilon \leq w'(\alpha)\right\}, \tag{4.1}$$

$$\mathcal{R}^*(\alpha) = \left\{\sigma \in \mathbb{M}_s : \frac{1}{2}S'(\alpha)\sigma : \sigma \leq w'(\alpha)\right\}, \tag{4.2}$$

where $\mathbb{M}_s$ is the space of symmetric tensors and $S(\alpha) = A^{-1}(\alpha)$.

The evolution of the sizes of (4.1) play a important role in the qualitative properties in the evolution damage problems. To see this, first we introduce the following definition

Definition 4.1 (Hardening properties) *We say that the material behavior is:* **Strain Hardening** *if $\alpha \mapsto \mathcal{R}(\alpha)$ is increasing,* **Stress Hardening** *if $\alpha \mapsto \mathcal{R}^*(\alpha)$ is increasing and* **Stress Softening** *if $\alpha \mapsto \mathcal{R}^*(\alpha)$ is decreasing.*

Remark 4.2 *The monoticity properties mut be understood in the sense of the set inclusion. For example, the Strain Hardening propertie means that*

$$\alpha_1 < \alpha_2 \Rightarrow \mathcal{R}(\alpha_1) \subseteq \mathcal{R}(\alpha_2). \tag{4.3}$$

Depending on the behavior of the quadratic forms defining these domains, the material is said to be

- *Strain Hardening* when $\alpha \mapsto (-A'(\alpha)/w'(\alpha))$ is decreasing with respect to α, i.e.,
$$w'(\alpha)A''(\alpha) - w''(\alpha)A'(\alpha) > 0. \tag{4.4}$$
- *Stress Hardening* (resp. Softening) when $\alpha \mapsto (S'(\alpha)/w'(\alpha))$ is decreasing (resp. increasing) with respect to α, i.e.,
$$w'(\alpha)S''(\alpha) - w''(\alpha)S'(\alpha) < 0 \text{ (resp. } > 0 \text{)}. \tag{4.5}$$

The importance of these definitions will appear in the study of the solutions of the damage problems as we will show in the next section.

4.1.2 Models for damage laws

For the study of the different behavior, following [40], we consider four different models, with its damage laws and hardening properties respectively. This models are defined as follows:

- **Model 1:** We consider a model with an elastic phase. For this, we assume a linear function $w(\alpha)$ and quadratic function $\mathrm{a}(\alpha)$, that is, we take the following damage law
$$w(\alpha) = w_1\alpha, \quad \mathrm{a}(\alpha) = (1-\alpha)^2. \tag{4.6}$$
This law satisfies both the strain hardening and stresss softening conditions for $\alpha \in [0, 1)$.
- **Model 2:** We consider the original Ambrosio-Tortorelli regularization model. For this, we assume a quadratic function $w(\alpha)$ and quadratic function $\mathrm{a}(\alpha)$, that is, we take the following damage law
$$w(\alpha) = w_1\alpha^2, \quad \mathrm{a}(\alpha) = (1-\alpha)^2. \tag{4.7}$$
This law satisfies the condition strain hardening for $\alpha \in [0, 1)$ and stress softening only for $\alpha \geq 1/4$.
- **Model 3:** We consider a family of models with the same homogeneous strain-stress response. For this, we consider the following family of damage models indexed by parameter $p > 0$
$$w(\alpha) = w_1\left(1 - (1-\alpha)^{p/2}\right), \quad \mathrm{a}(\alpha) = (1-\alpha)^p. \tag{4.8}$$
This case is a generalization of the Model 1 which is recovered for $p = 2$. It satisfies both the strain hardening and stress softening conditions for $\alpha \in [0, 1)$ and any $p > 0$.
- **Model 4:** We consider a model where ultimate fracture occurs at finite strain. For this, we assume that it is defined by the following material functions parametrized by a scalar parameter $k > 1$
$$w(\alpha) = w_1\alpha, \quad \mathrm{a}(\alpha) = \frac{1-\alpha}{1+(k-1)\alpha}. \tag{4.9}$$

4.1.3 Numerical results

In this section we will validate the model proposed in Section 3. For this, we will replicate the results seen in a cylindrical "testigo" that undergoes a vertical compression and that also has an initial vertical failure, as seen in Figure 4.1a. In the experiments it can be seen that, after applying the compression, the "testigo" breaks diagonally following the line of the initial fracture (Figure 4.1b).

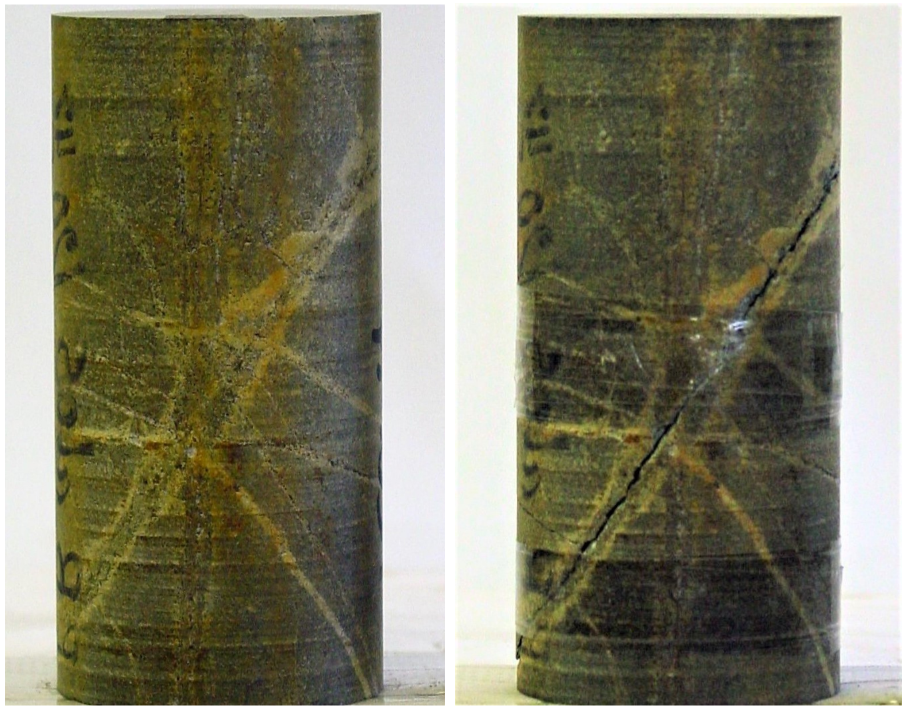
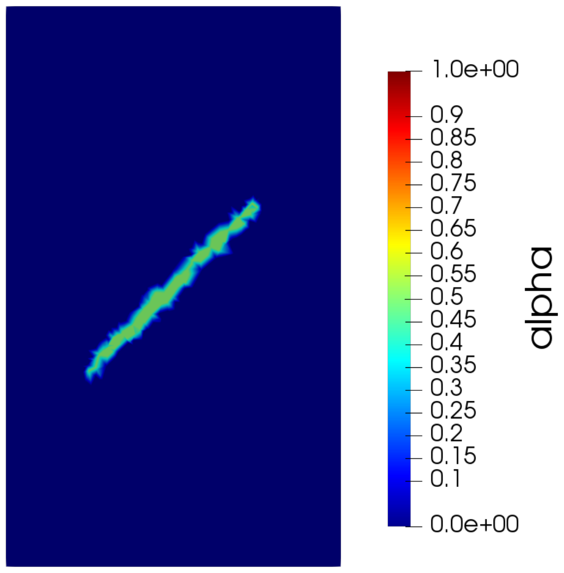

(a) Before compression. (b) After compression.

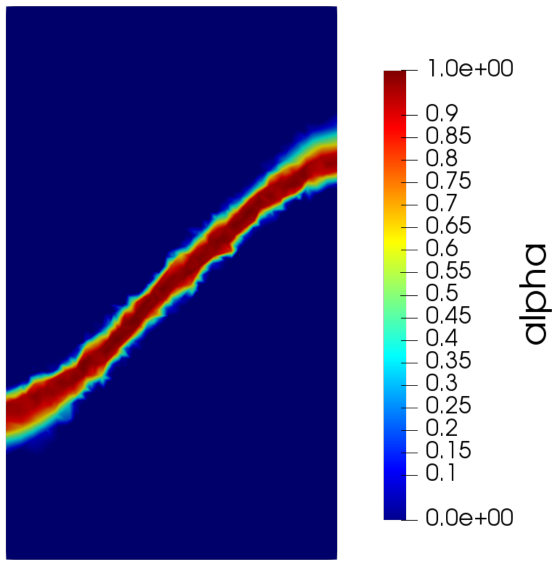
Figure 4.1: Laboratory test of a "testigo" subjected to a vertical compression.

To model the experiment seen in the Figure 4.1, let us consider a homogeneous cylindrical domain of $L = 0.2$ meters length and radius $R = 0.06$ meters. For the lower base of the cylinder we will consider that the displacement is null in all directions, while in the upper base we will consider controlled displacement in the axis z, that is, $u_z = -0.005t$ meters. In all these examples we will consider free condition for the displacement in the lateral boundary, that is, $\sigma \cdot n = 0$. For the case of damage, the boundary conditions will be considered homogeneous Neumann type, that is, $\frac{\partial \alpha}{\partial n} = 0$.

The Figures 4.2-4.5 display the distribution of the damage for the different models proposed previously and for the times $t = 0.0$ seconds and $t = 1.0$ seconds with an initial diagonal fracture in the center of the cylinder with magnitude $\alpha = 0.5$. Is possible see that, the different models have different behaviour for the distribution of the damage. In the Models 1 and 3 the damage only appear in the path of the diagonal fracture, where the the values of the damage grows to $\alpha = 1.0$ and extends following this path. In the Models 2 and 4 is possible see that the distribution of the damage does not consider the initial fracture and the damage is distributed more homogeneously throughout the domain.

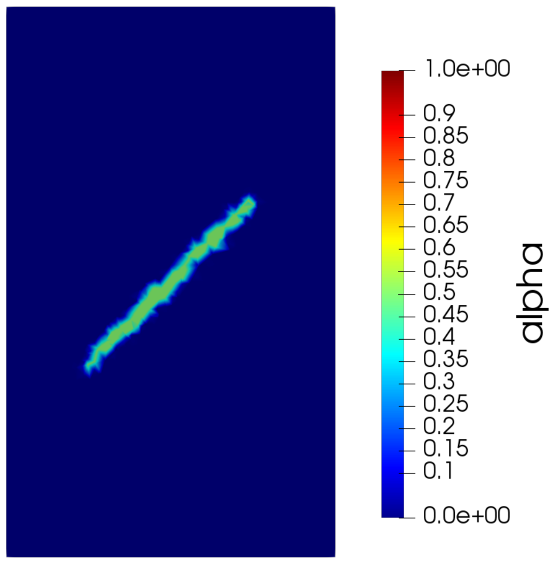


(a) t=0.0 seg.

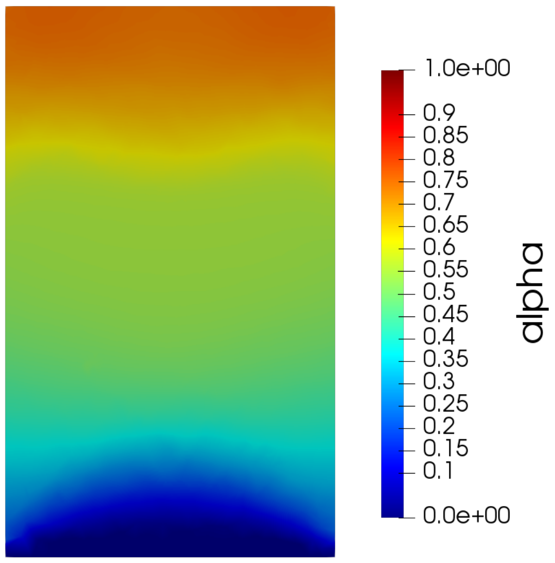


(b) t=1.0 seg.

Figure 4.2: Damage field distribution in the cylinder for the model 1.

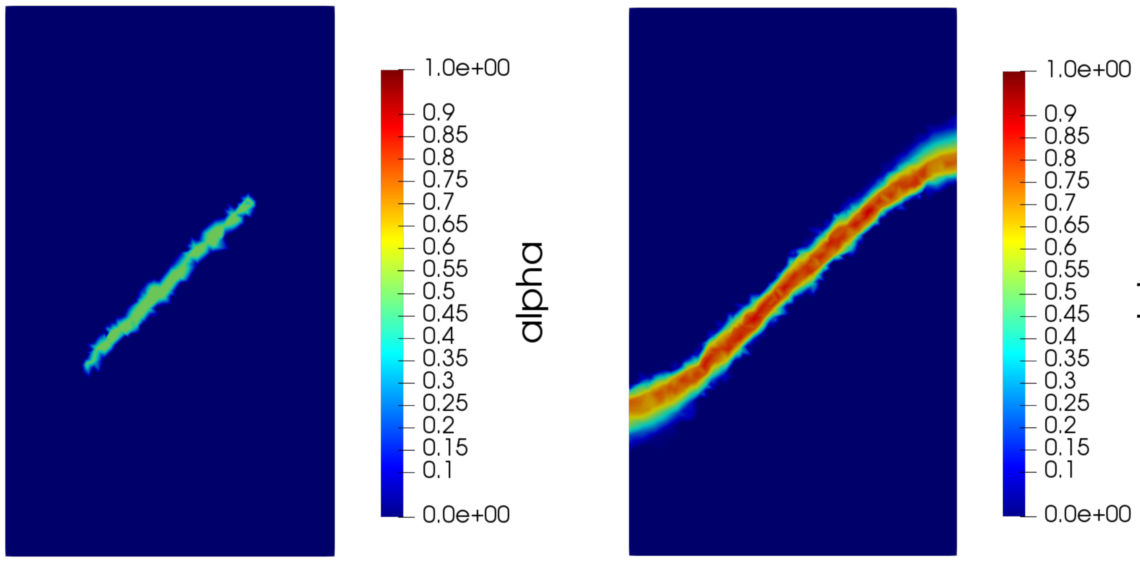


(a) t=0.0 seg.



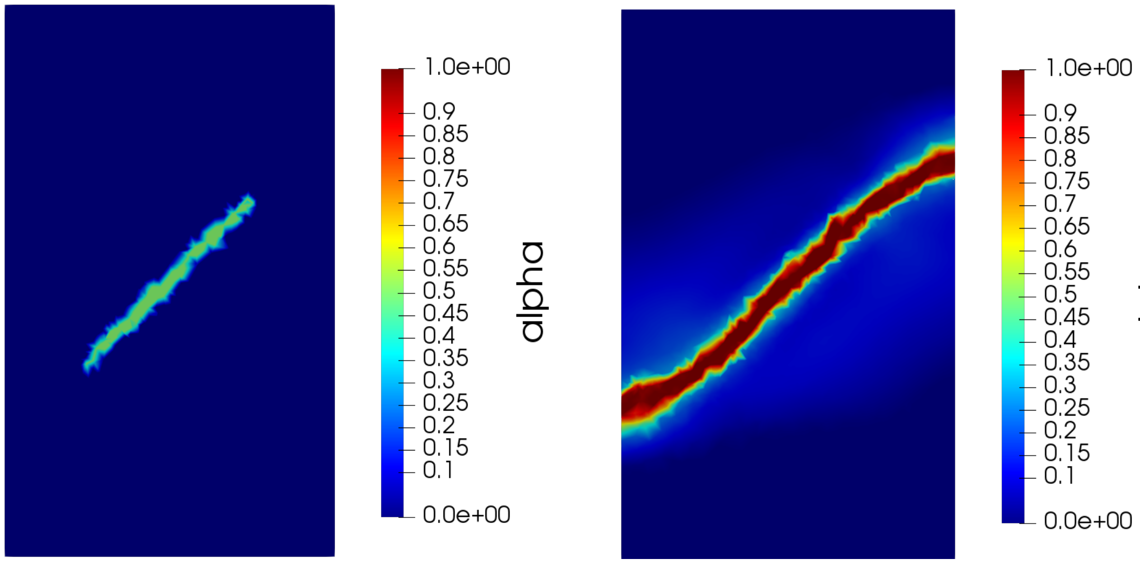
(b) t=1.0 seg.

Figure 4.3: Damage field distribution in the cylinder for the model 2.



(a) t=0.0 seg. (b) t=1.0 seg.

Figure 4.4: Damage field distribution in the cylinder for the model 3.



(a) t=0.0 seg. (b) t=1.0 seg.

Figure 4.5: Damage field distribution in the cylinder for the model 4.

Through these results, we can differentiate two kinds of materials: The first kind of materials are those that, from a fracture (damaged area), follow the path of the fracture getting the material to be damaged in a particular area, for example the Models 1, 3 and 4, following the laboratory results seen in the Figure 4.1. The second kind of materials, are those that do not consider the fracture and are damaged homogeneously (Model 2), that is, do not show the results seen in laboratory tests. After seeing the results, we can conclude that our model is valid to represent the behavior of fractures, since it is capable of replicating the laboratory test shown in Figure 4.1.

4.2 Damage model in the block caving process and boundary conditions

For simulate the block caving process, we consider a domain that simulates a rock mass in three dimensions $\Omega_0 \subseteq \mathbb{R}^3$ with $\partial\Omega = \Gamma_{lat} \cup \Gamma_{up} \cup \Gamma_{down}$ denoting the lateral, upper and lower bounded of the domain respectively. We consider a time depending interior domain $S(t_{\mathrm{i}}) \subseteq \Omega_0$, with $\partial S(t_{\mathrm{i}}) = \Gamma_{cav}(t_{\mathrm{i}})$ and $S(t_{\mathrm{i}}) \subseteq S(t_{\mathrm{i}+1})$, representing the interior cavity of this rock mass. Then in each time step, we consider the evolution domain $\Omega(t_{\mathrm{i}}) = \Omega_0 \setminus S(t_{\mathrm{i}})$, that is, our domain where we calculate the damage model will be the initial domain Ω_0 minus the internal cavity in each time step.

Finally, to define the boundary conditions for our problems, let us consider the equilibrium equation for fixed value of α that, no loss of generality, we assume that $\alpha = 0$

$$\begin{array}{rclcl} -\mathrm{div}(\sigma) &=& f, & \text{in} & \Omega, \\ \sigma \cdot n &=& 0, & \text{on} & \Gamma_{cav}, \\ \sigma \cdot n &=& 0, & \text{on} & \Gamma_{up}, \\ u \cdot n &=& 0, & \text{on} & \Gamma_{down}, \\ \sigma \cdot n &=& g, & \text{on} & \Gamma_{lat}, \end{array} \tag{4.10}$$

where $\sigma = 2\mu\varepsilon(u) + \lambda \mathrm{tr}(\varepsilon(u))I$ and $f = (0, 0, -\rho g)^T$. We will assume that, in the rock mass, we do not have lateral displacement, that is,

$$u = \begin{pmatrix} 0 \\ 0 \\ u_z(z) \end{pmatrix} \text{ and } \varepsilon(u) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & u_z'(z) \end{pmatrix}, \tag{4.11}$$

so we will have

$$\sigma = \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda + 2\mu \end{pmatrix} u_z'(z) \text{ and } \mathrm{div}(\sigma) = \begin{pmatrix} 0 \\ 0 \\ (\lambda + 2\mu)u_z''(z) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \rho g \end{pmatrix}. \tag{4.12}$$

Then

$$u_z'(z) = \frac{\rho g}{\lambda + 2\mu} z + C. \tag{4.13}$$

Now, denoting by H_{max}, the maximum height of the domain and considering that $\sigma_{zz}(H_{max}) = 0$, we have

$$C = -\frac{\rho g H_{max}}{\lambda + 2\mu}. \tag{4.14}$$

Then

$$u_z'(z) = \frac{\rho g}{\lambda + 2\mu}(z - H_{max}), \tag{4.15}$$

and, thus

$$\sigma = \begin{pmatrix} \frac{\lambda}{\lambda+2\mu} & 0 & 0 \\ 0 & \frac{\lambda}{\lambda+2\mu} & 0 \\ 0 & 0 & 1 \end{pmatrix} \rho g(z - H_{max}). \tag{4.16}$$

This gives us a condition that depends on the height z and the maximum height of the domain H_{max}, so for non-homogeneous domains, for example a real mine, this condition gives us problems, because the border conditions have different values for each side face, since these would not have the same height, causing that the damage appears in the corners of the domain. To solve this problem we will include a Robin boundary condition of the form $\sigma \cdot n + \overline{k}(u \cdot n) \cdot n = 0$, with $\overline{k} > 0$ a real constant. Finally the boundary condition is of the form

$$g = \frac{\lambda}{\lambda + 2\mu} \rho g (z - H_{max}) - \overline{k}(u \cdot n) \cdot n. \tag{4.17}$$

4.2.1 Numerical results

In all our tests, we consider Ω_0 such that $x \in [-1540, 2060]$, $y \in [-1050, 1050]$ and $z \in [-500, 450]$. Here the meshes are defined following the Appendix A and we use the synthetic cavities defined in the Figures A.1, where it is possible to see the different configurations of each cavity. All simulations presented here use the following values for the parameters in the damage model

$$E = 2.9 \cdot 10^{10}, \quad \nu = 0.3, \quad \kappa = 1.0 \quad \text{and} \quad \overline{k} = 10^9, \tag{4.18}$$

For the numerical test, first, we consider that the constant in the function $w(\alpha)$ is independent of the constant in the elliptic regularization term in (3.34), this in order to independently control the variable the energy dissipated without affecting the term of regularization. Thus, for the models shown in Section 4.1.2, we take

$$w(\alpha) = w_{11}\alpha, \quad w(\alpha) = w_{11}\alpha^2, \quad w(\alpha) = w_{11}\left(1 - (1-\alpha)^{p/2}\right), \quad \text{with} \quad w_{11} \neq w_1. \tag{4.19}$$

The Figures 4.6-4.9 summarize the maximum values of α in each mesh iteration. The images display the evolution of this maximum value and we can be see that for little values w_{11} the damage grows faster than bigger values of this constant. For large w_{11} values the damage does not achieve the maximum value to fully damage state.

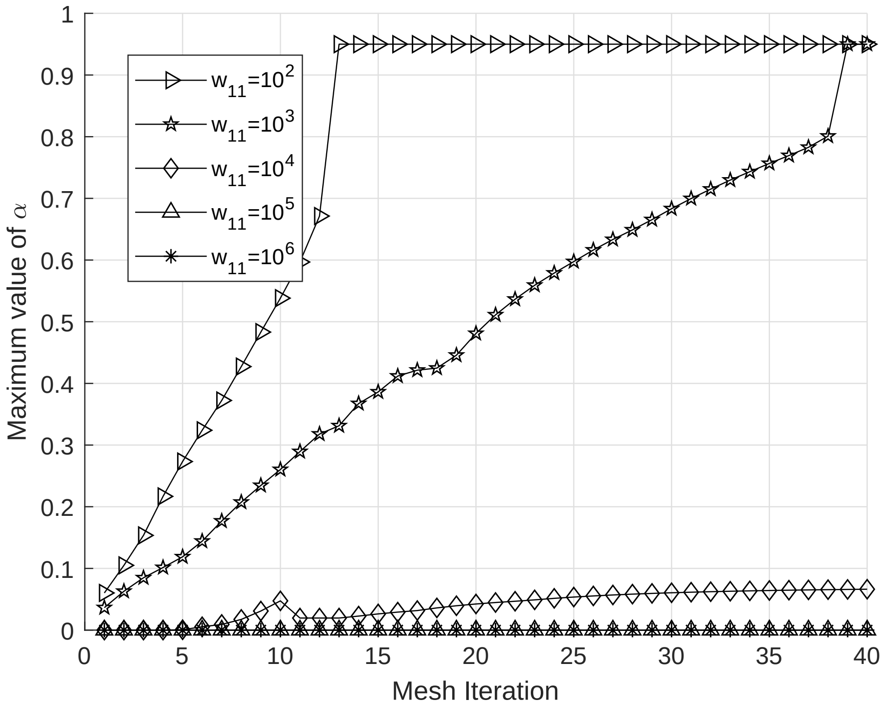


Figure 4.6: Maximum damage value for Model 1 and for each cavity advance considering different values of w_{11}.

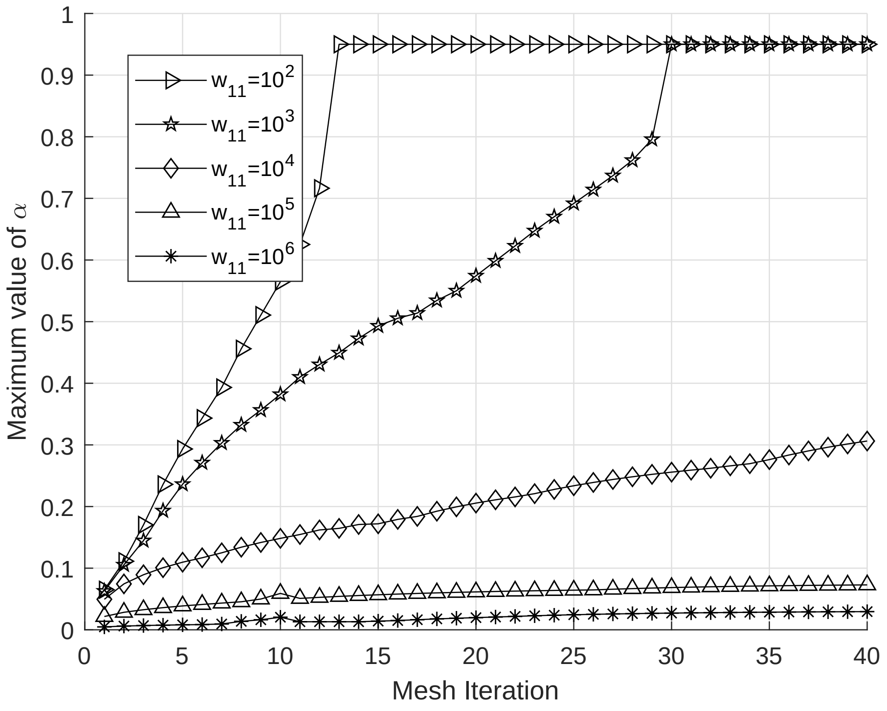


Figure 4.7: Maximum damage value for Model 2 and for each cavity advance considering different values of w_{11}.

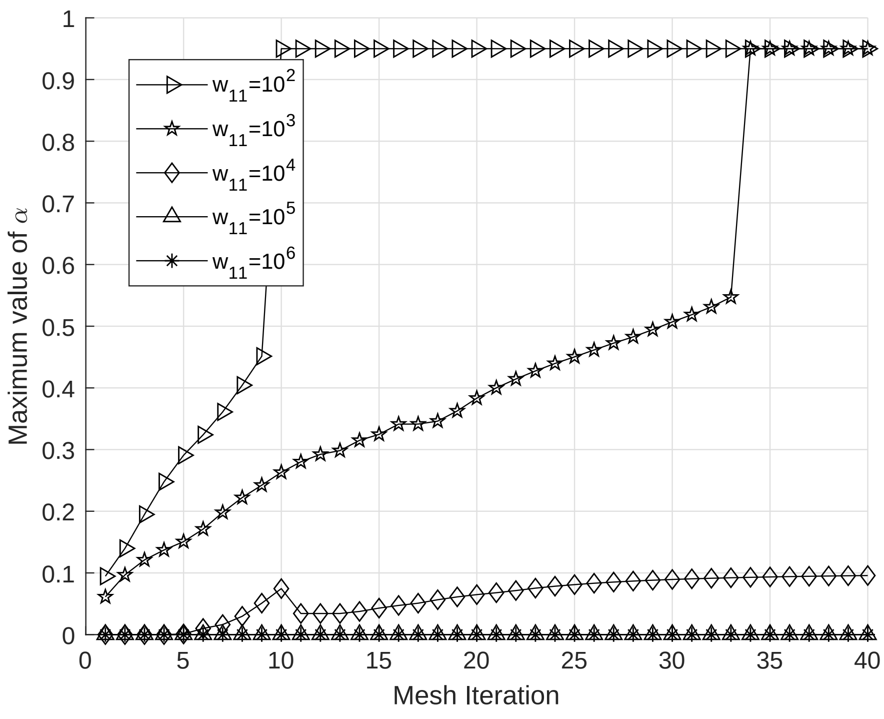


Figure 4.8: Maximum damage value for Model 3 and for each cavity advance considering different values of w_{11}.

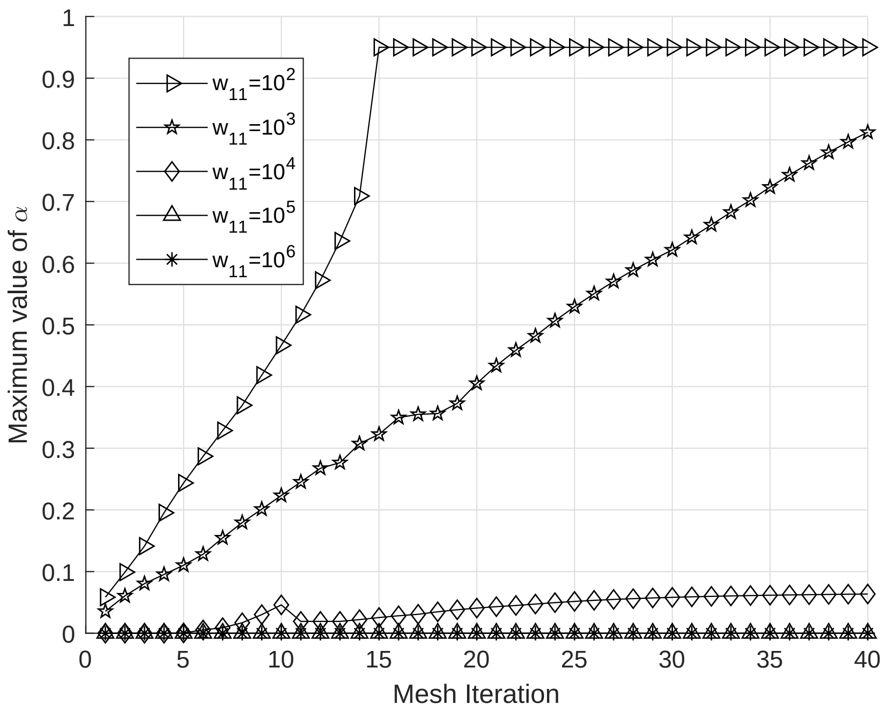
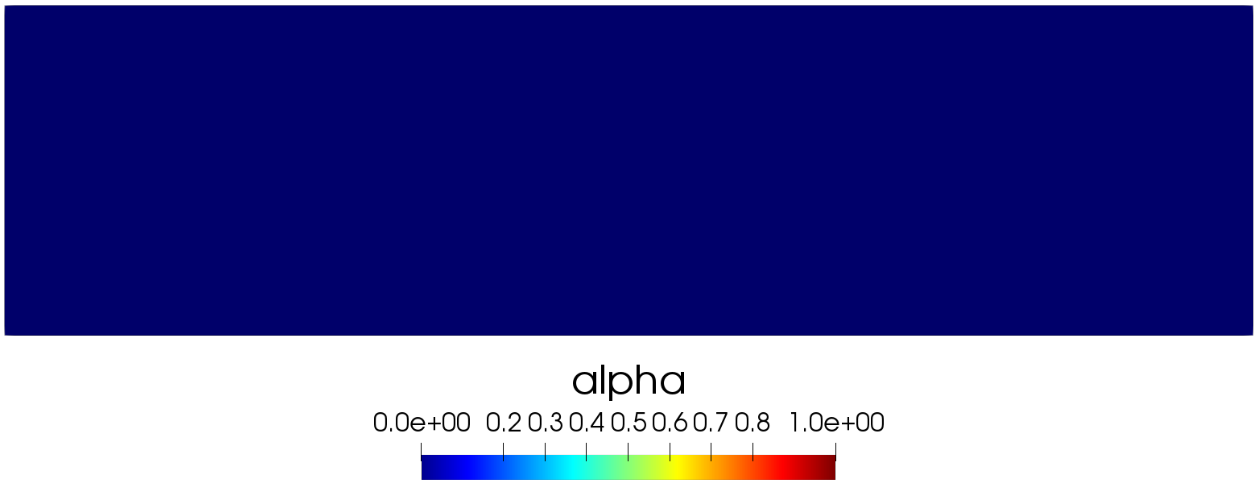
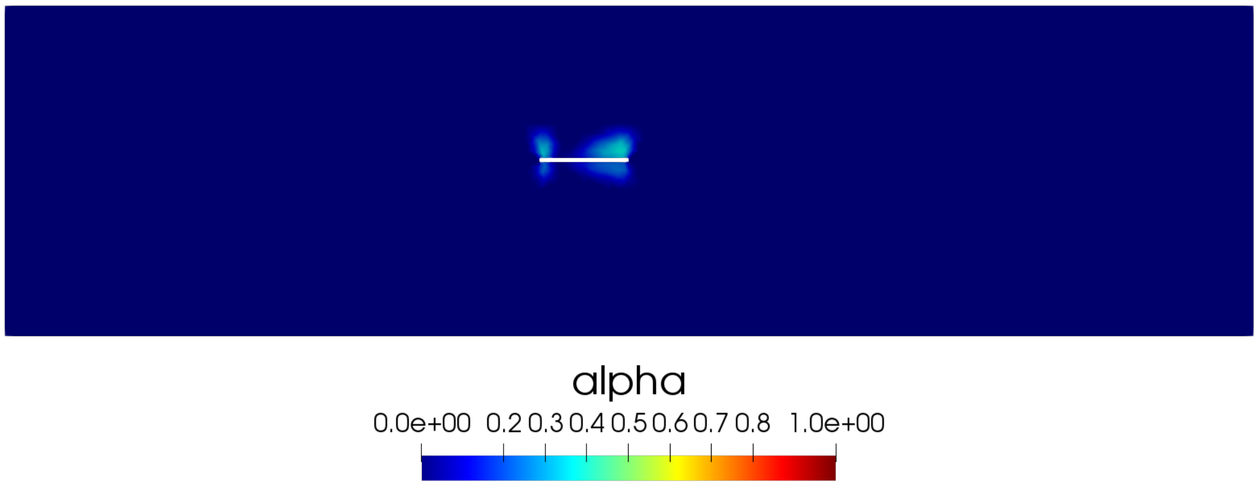


Figure 4.9: Maximum damage value for Model 4 and for each cavity advance considering different values of K_w.

Following the previous results, we consider that, for the Model 1, 2 and 3, $w_{11} = 10^3$ and, for Model 4, $w_{11} = 10^2$. Figures 4.10-4.17 display the evolution of the damage when the cavity advance in time, for the different models. As it showed in the Chapter 3, the damage is close to zero almost everywhere except around the cavity, it is possible see that the distribution of the damage reaching the upper boundary of the domain. For the all models, the material is fully damaged over the cavity.

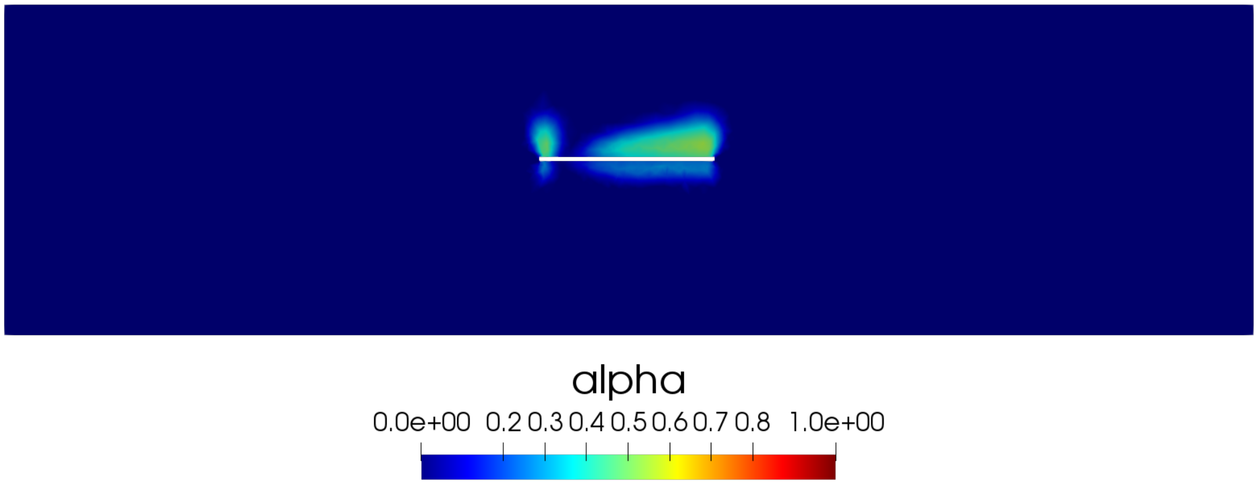


(a) $\Omega(t_0)$.

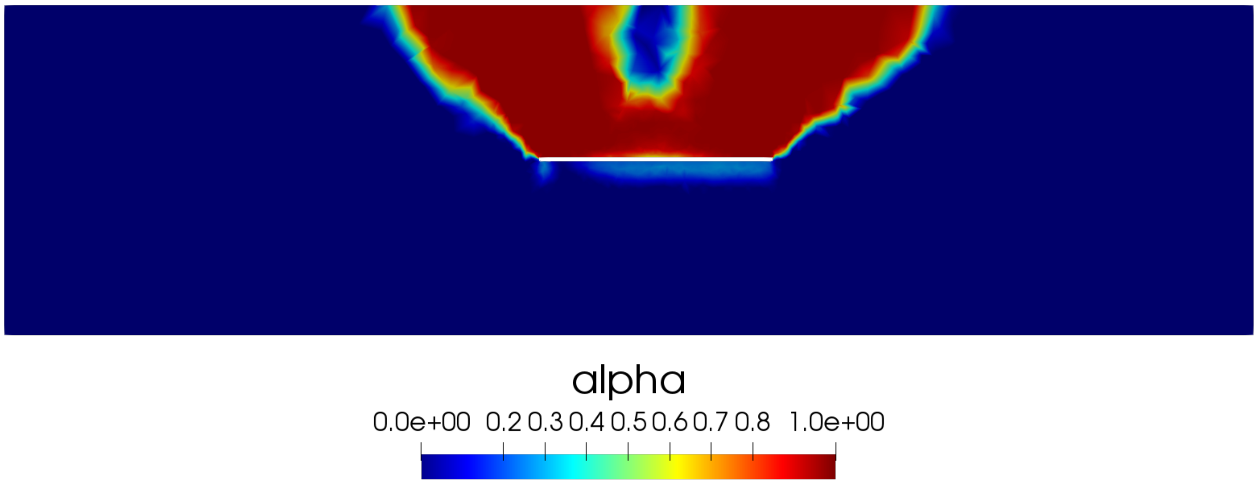


(b) $\Omega(t_{15})$.

Figure 4.10: Damage field distribution in the rock mass for Model 1.

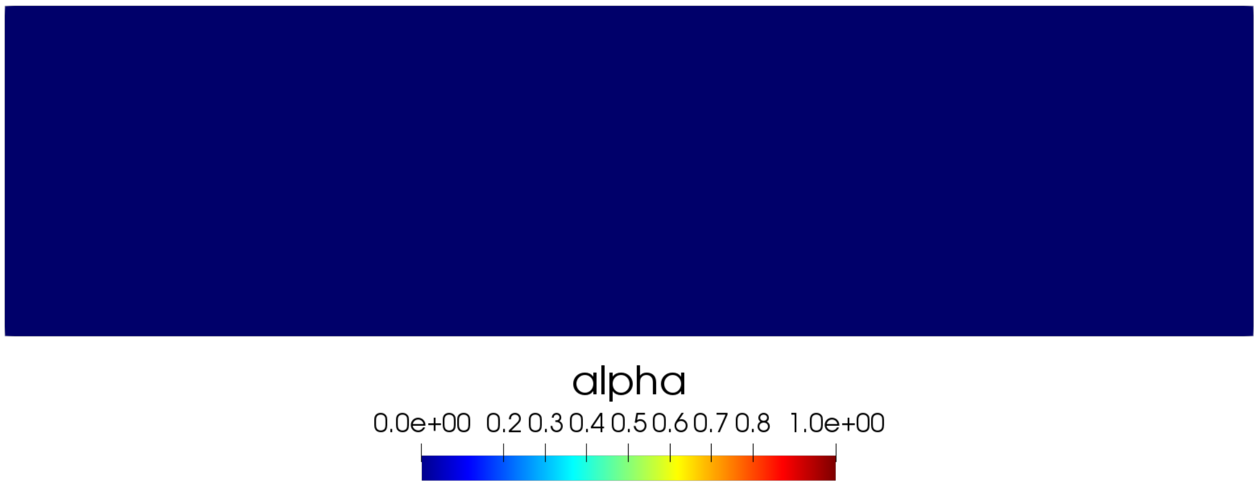


(a) $\Omega(t_{30})$.

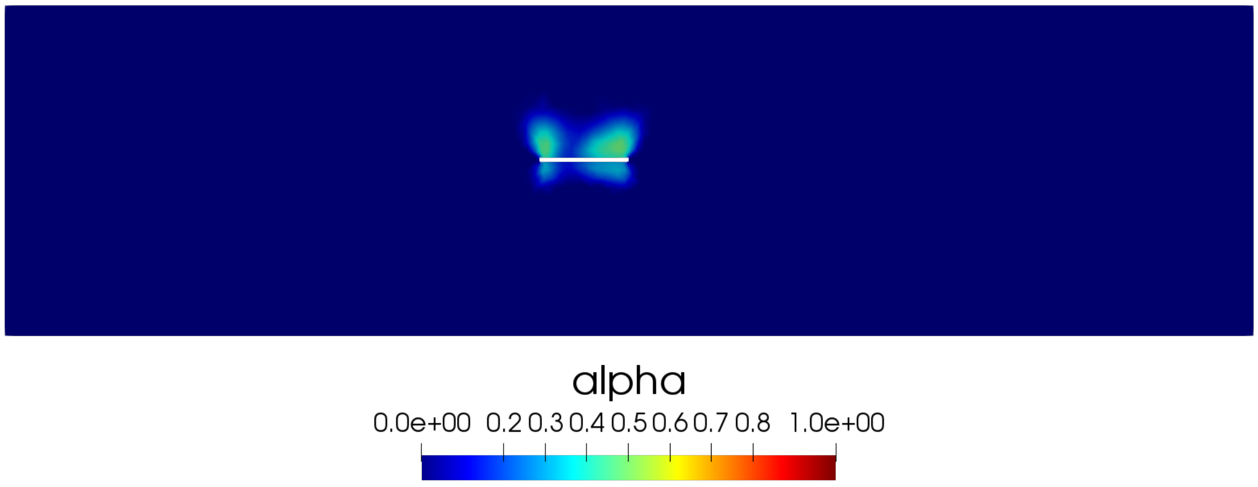


(b) $\Omega(t_{40})$.

Figure 4.11: Damage field distribution in the rock mass for Model 1.

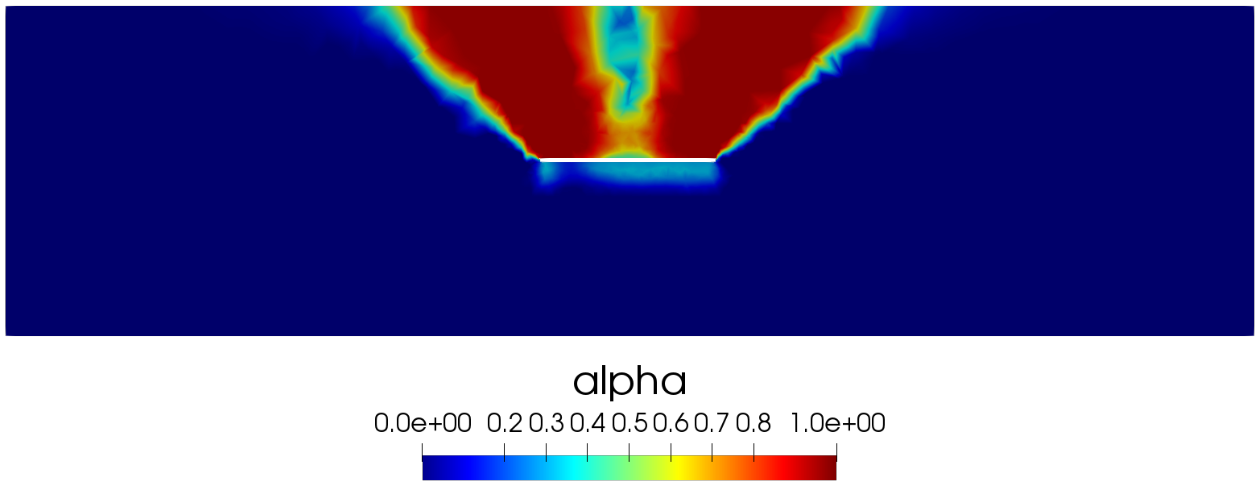


(a) $\Omega(t_0)$.

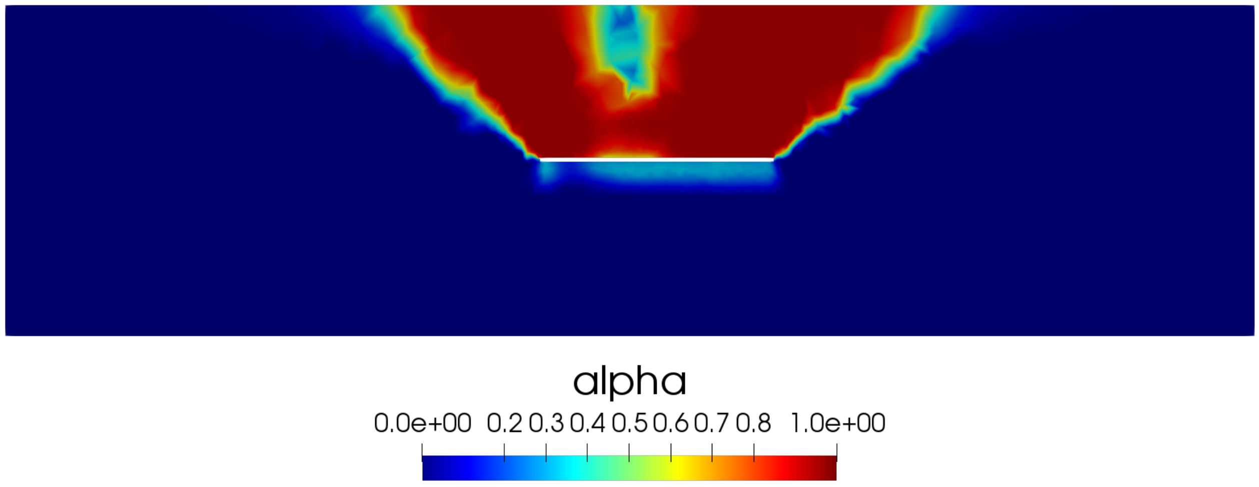


(b) $\Omega(t_{15})$.

Figure 4.12: Damage field distribution in the rock mass for Model 2.

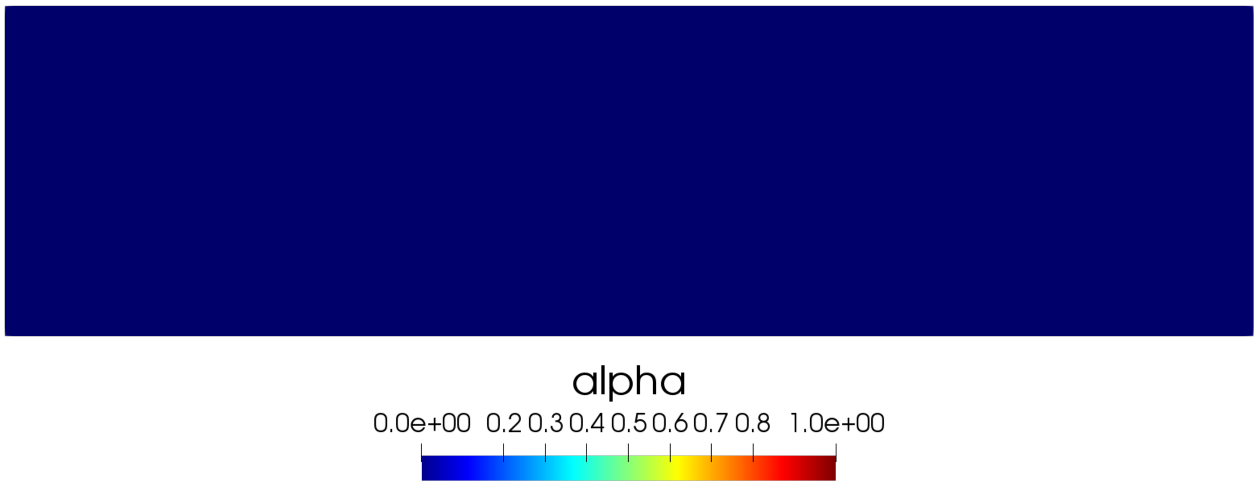


(a) $\Omega(t_{30})$.

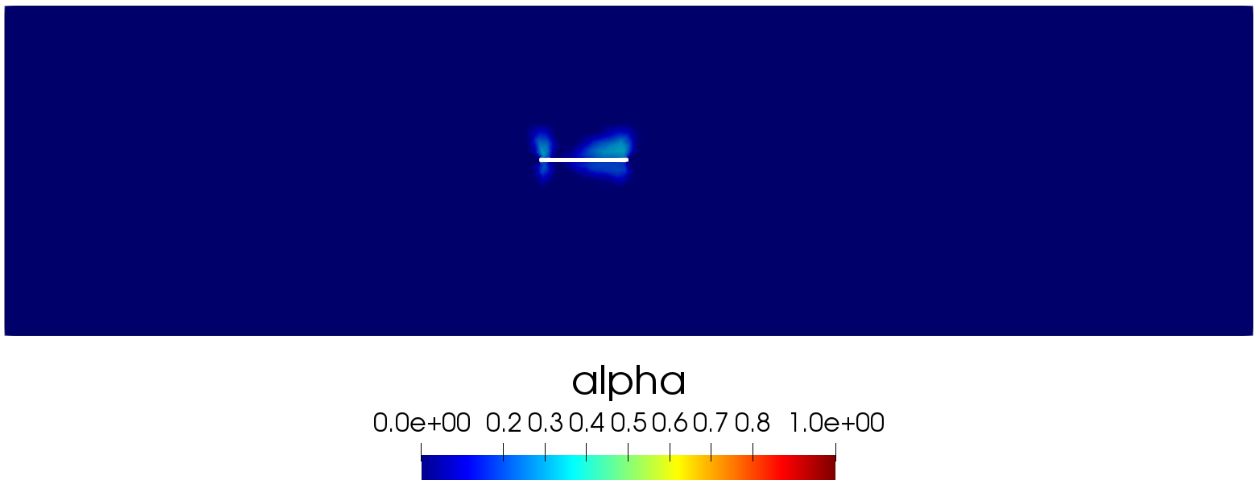


(b) $\Omega(t_{40})$.

Figure 4.13: Damage field distribution in the rock mass for Model 2.

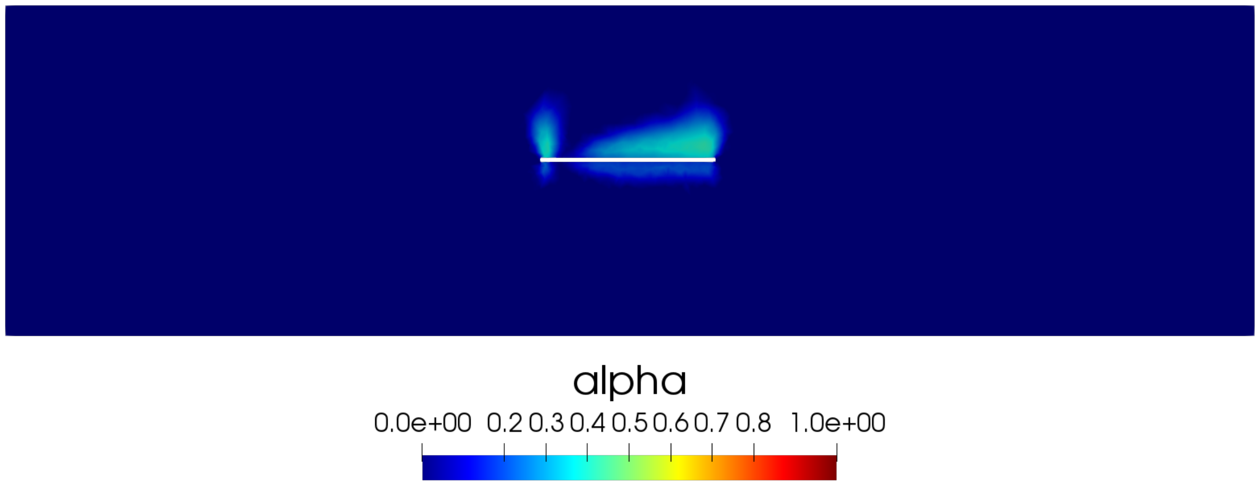


(a) $\Omega(t_0)$.

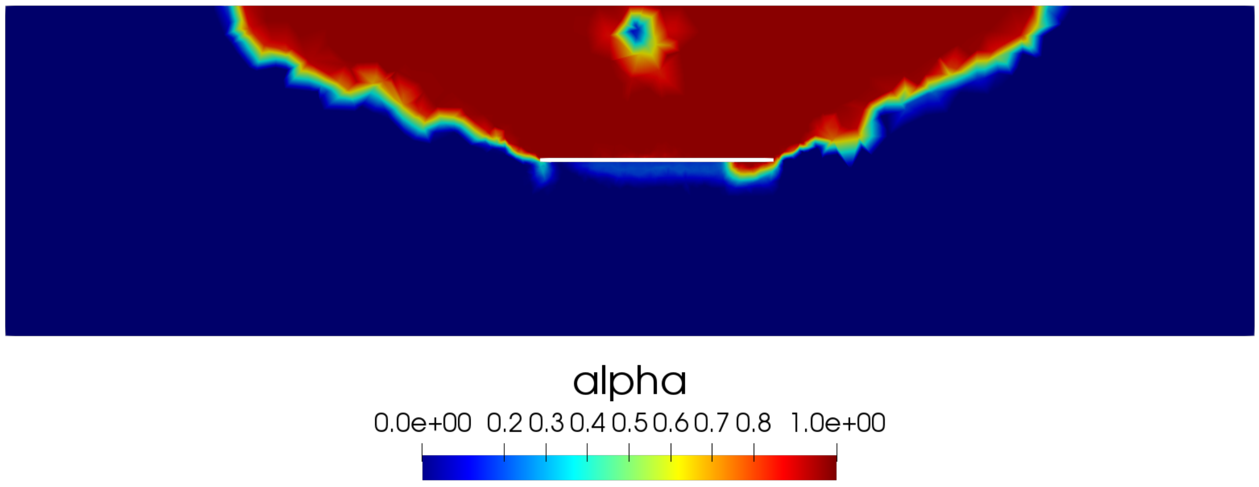


(b) $\Omega(t_{15})$.

Figure 4.14: Damage field distribution in the rock mass for Model 3.

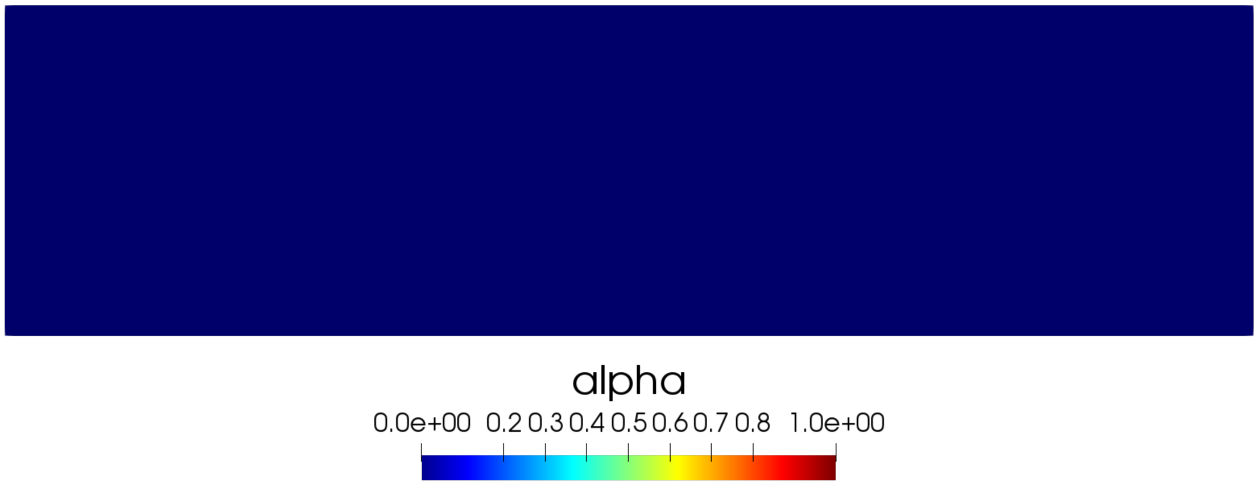


(a) $\Omega(t_{30})$.

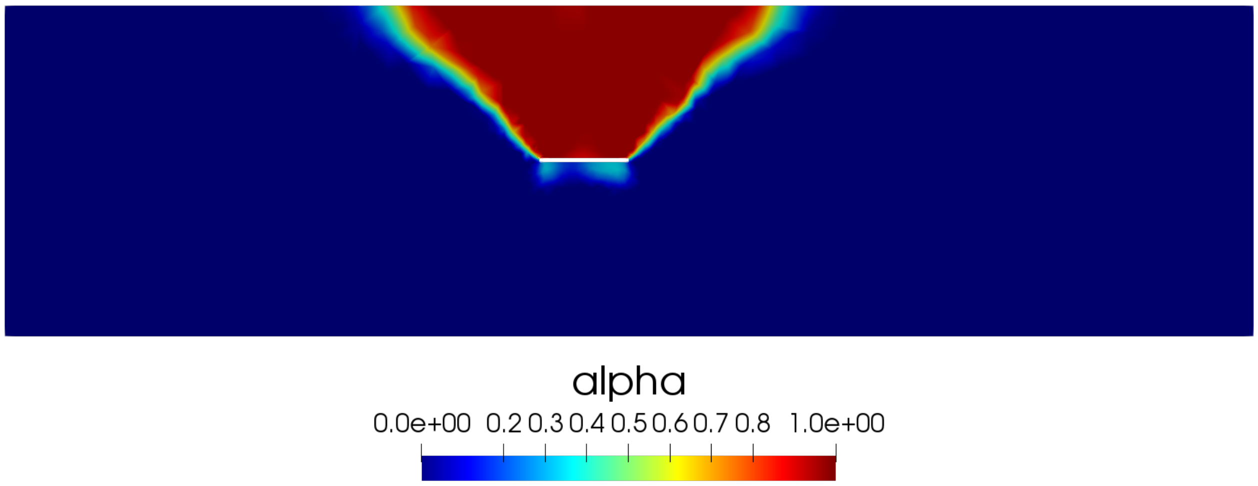


(b) $\Omega(t_{40})$.

Figure 4.15: Damage field distribution in the rock mass for Model 3.

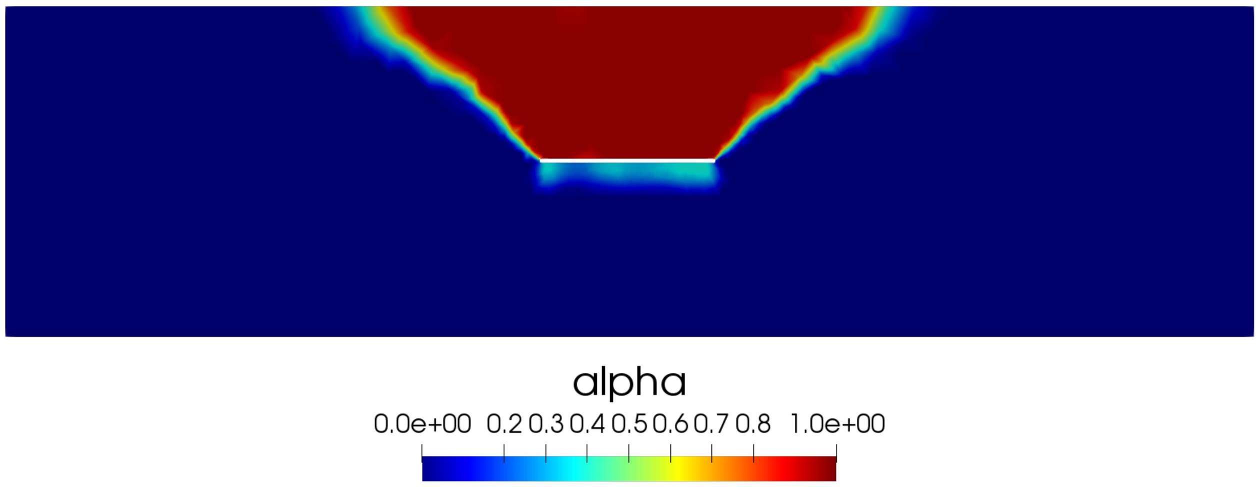


(a) $\Omega(t_0)$.

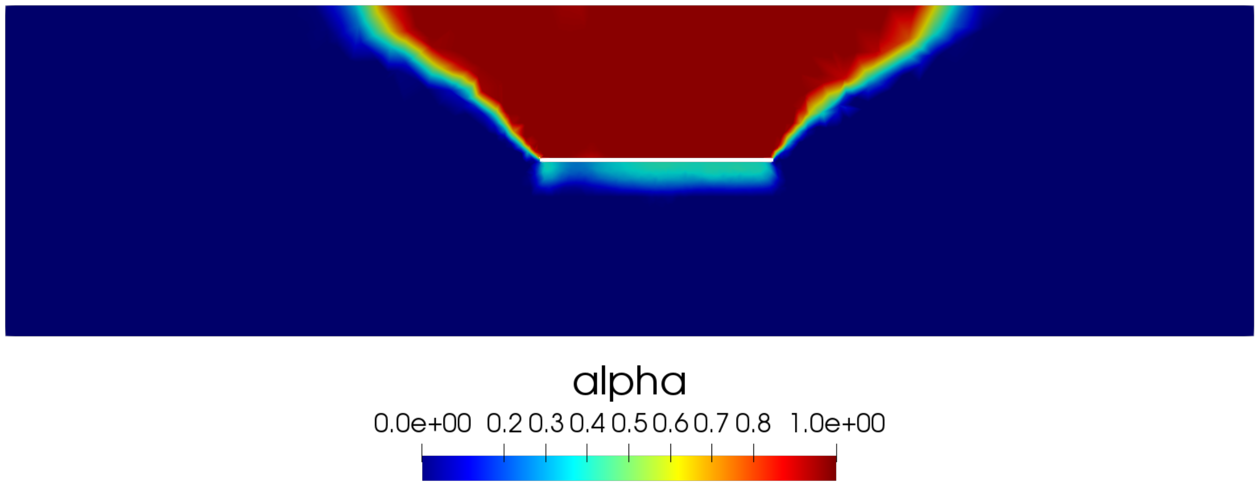


(b) $\Omega(t_{15})$.

Figure 4.16: Damage field distribution in the rock mass for Model 4.



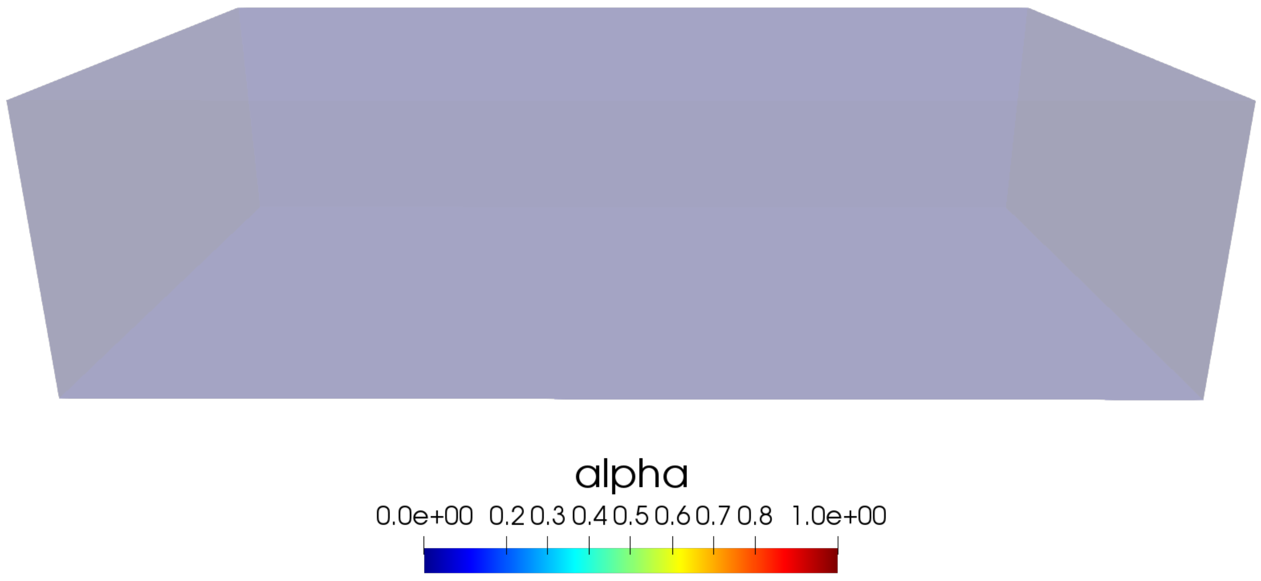
(a) $\Omega(t_{30})$.



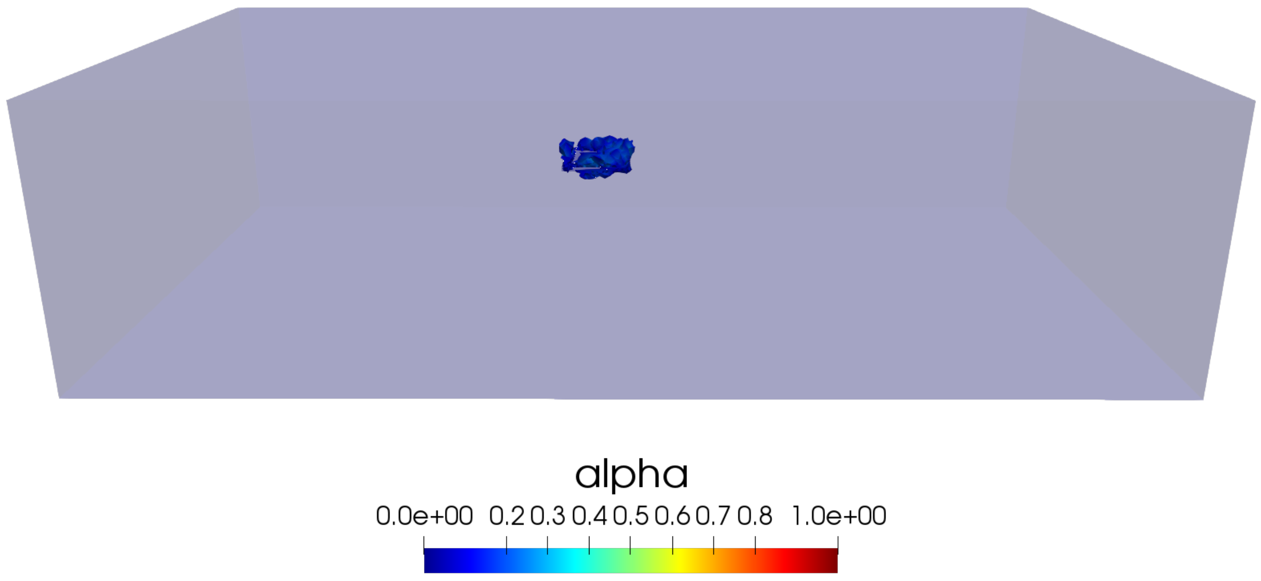
(b) $\Omega(t_{40})$.

Figure 4.17: Damage field distribution in the rock mass for Model 4.

For 3D visualization, the figures 4.18-4.25 display the distribution of the damage. In this view is possible see that the damage is close to zero almost everywhere except around the cavity and in all models the damage reaches the upper boundary with different magnitudes and distributions.

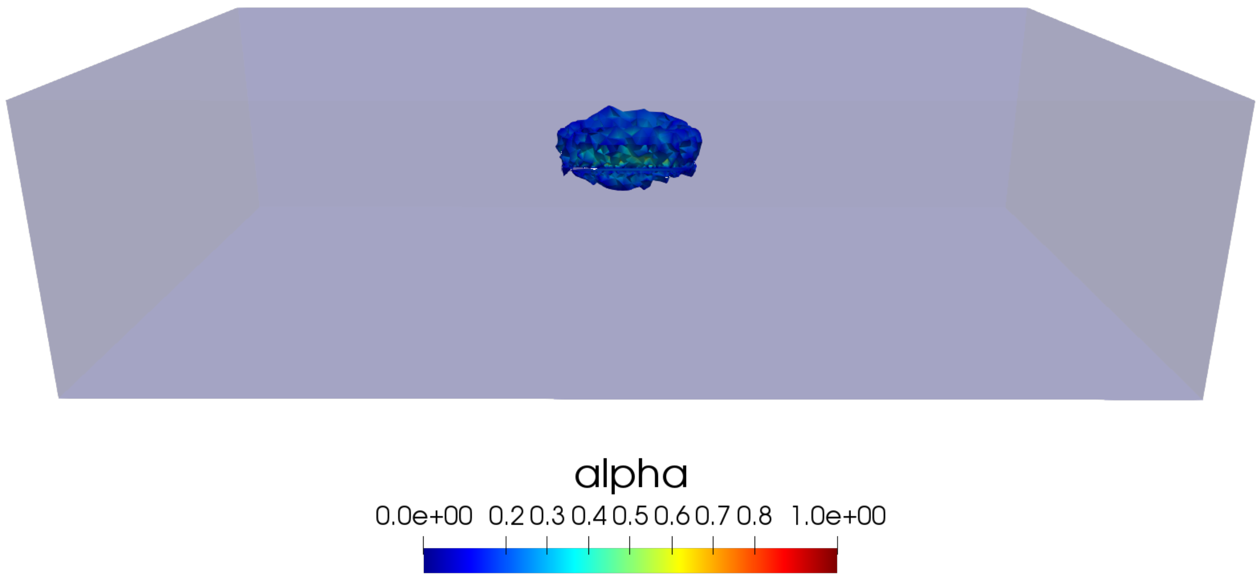


(a) $\Omega(t_0)$.

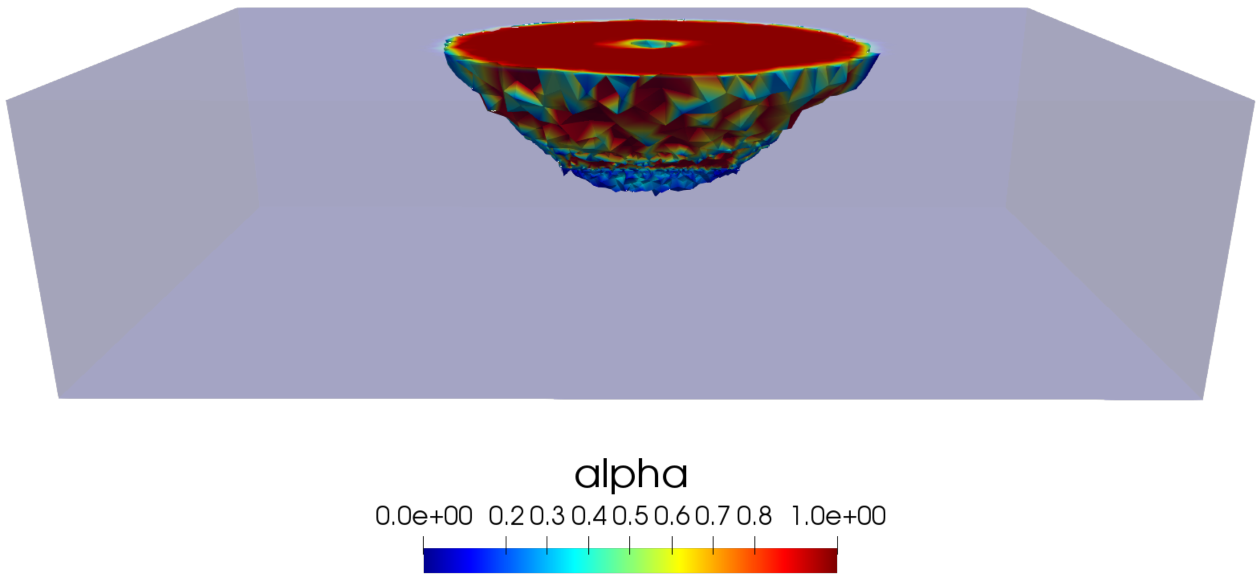


(b) $\Omega(t_{15})$.

Figure 4.18: Damage field distribution in the rock mass for Model 1 in 3D.

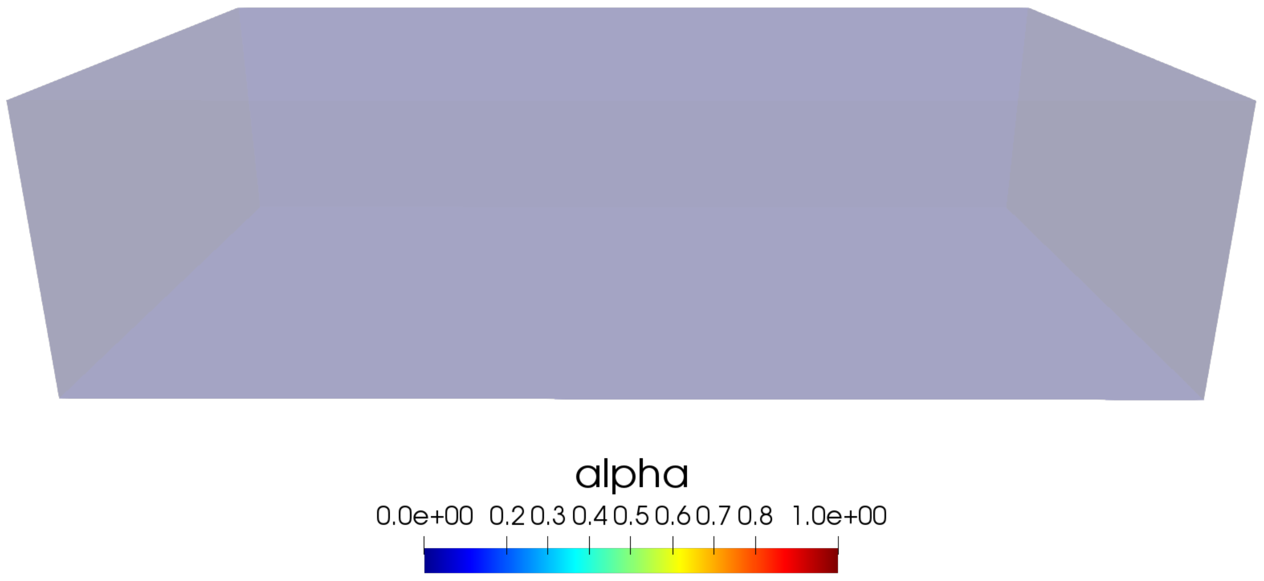


(a) $\Omega(t_{30})$.

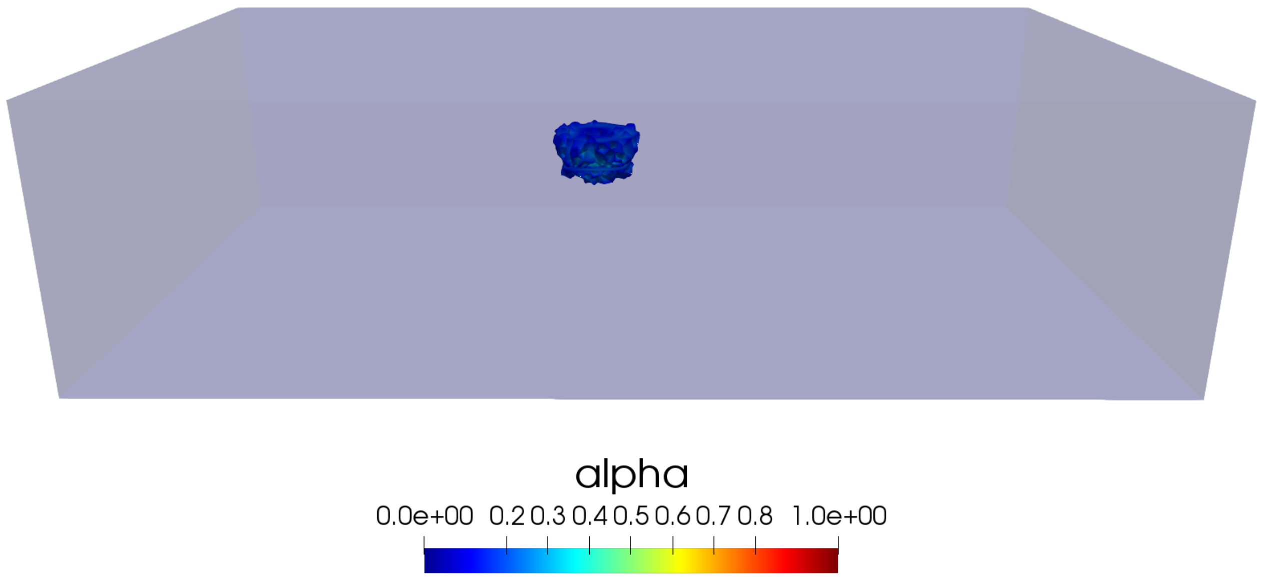


(b) $\Omega(t_{40})$.

Figure 4.19: Damage field distribution in the rock mass for Model 1 in 3D.

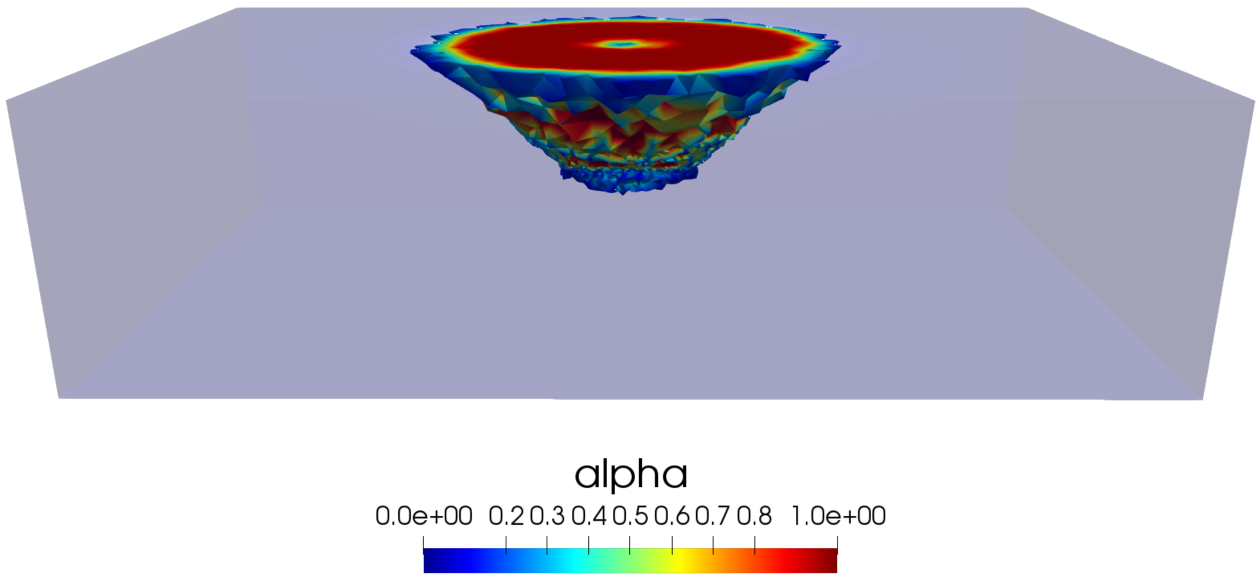


(a) $\Omega(t_0)$.

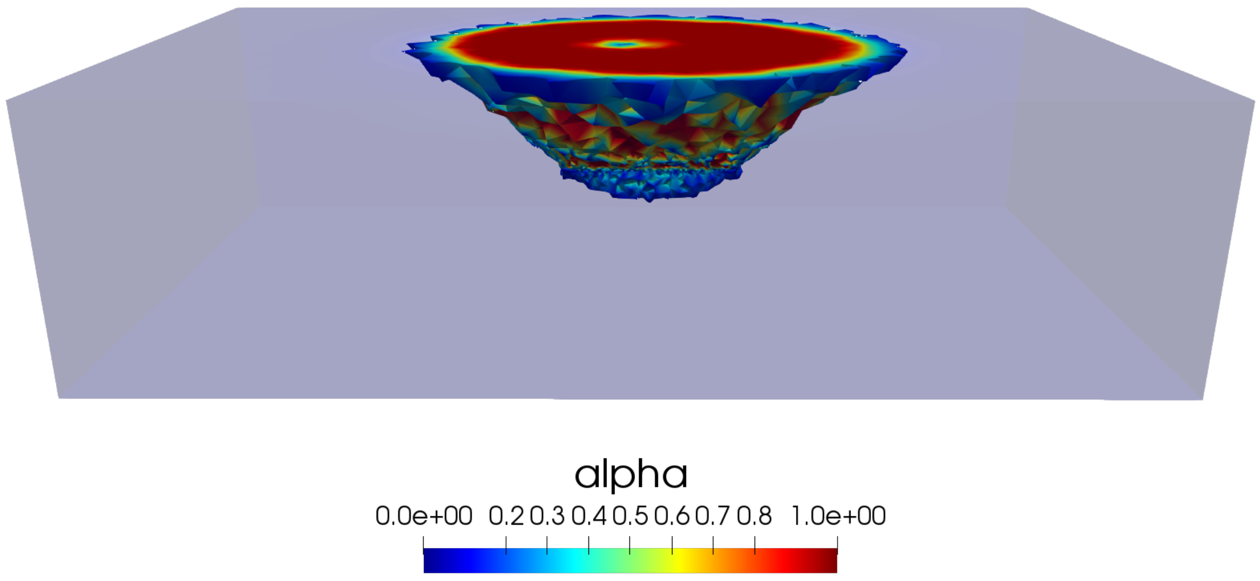


(b) $\Omega(t_{15})$.

Figure 4.20: Damage field distribution in the rock mass for Model 2 in 3D.

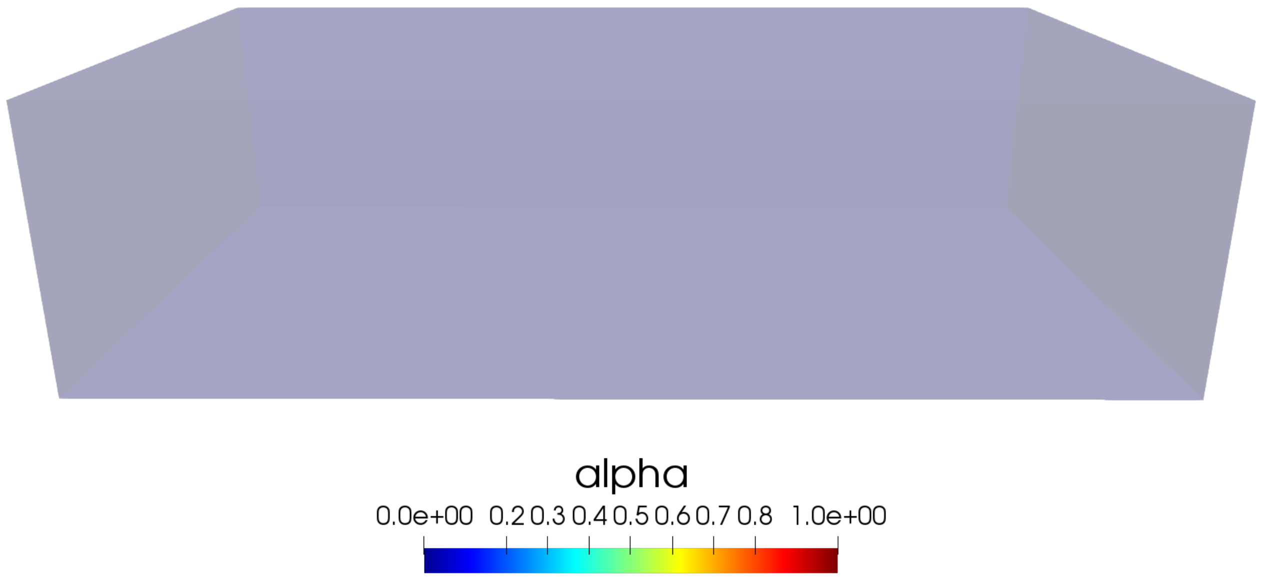


(a) $\Omega(t_{30})$.

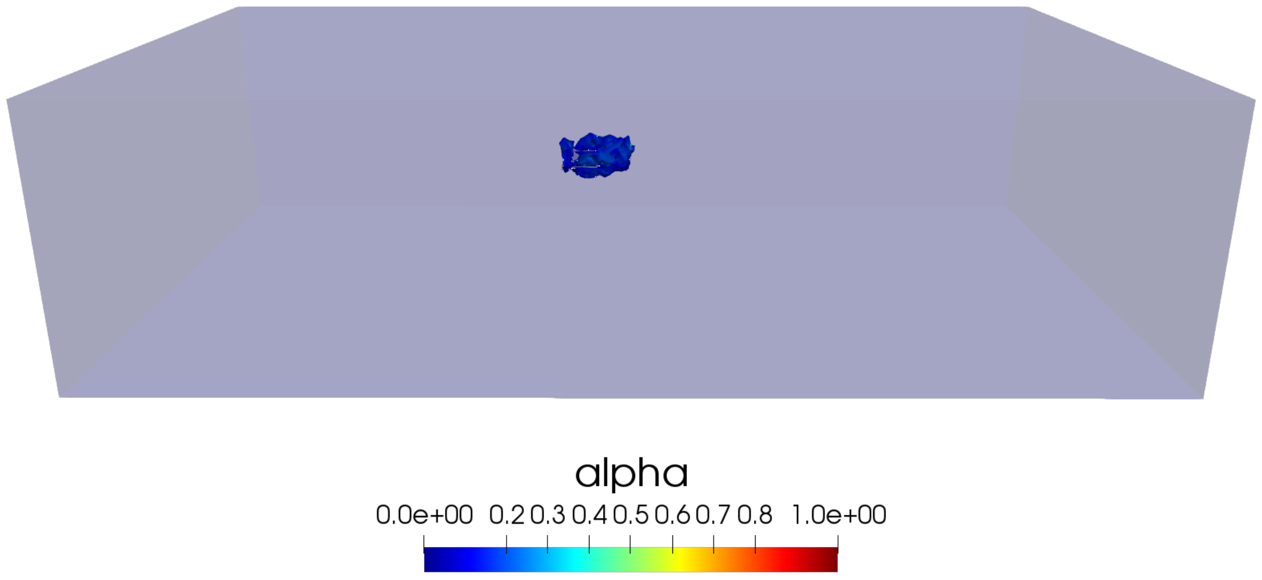


(b) $\Omega(t_{40})$.

Figure 4.21: Damage field distribution in the rock mass for Model 2 in 3D.

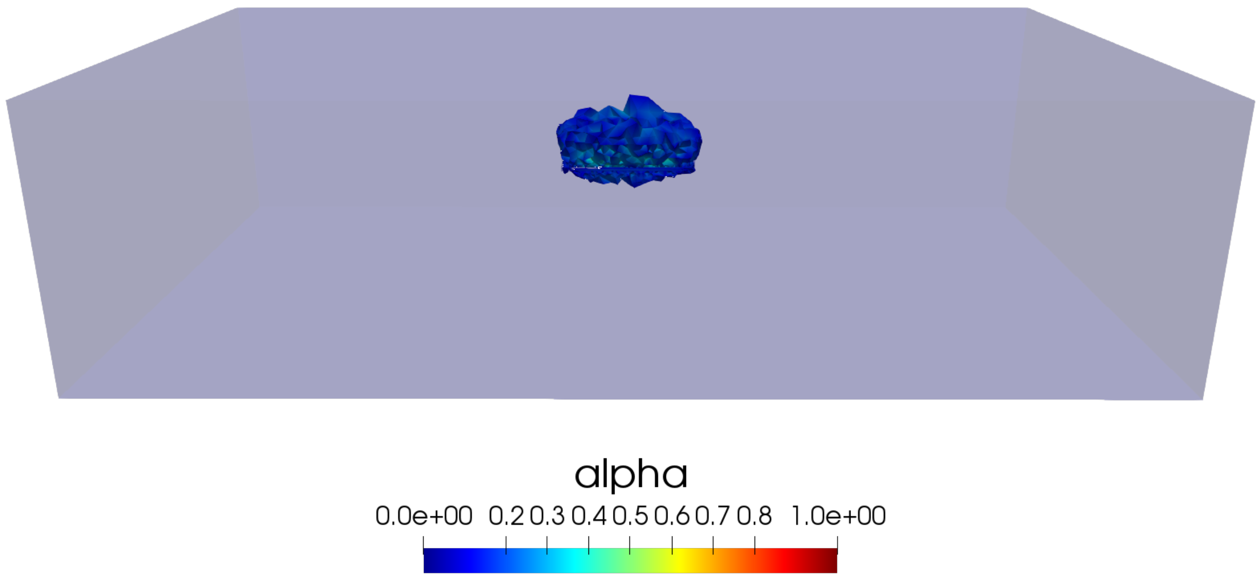


(a) $\Omega(t_0)$.

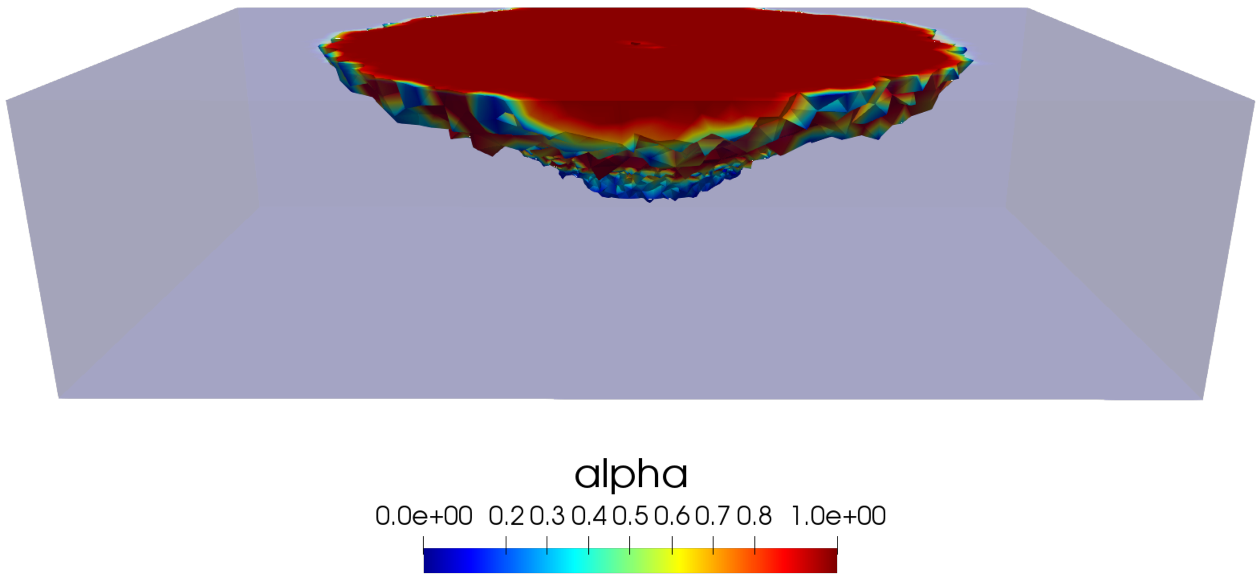


(b) $\Omega(t_{15})$.

Figure 4.22: Damage field distribution in the rock mass for Model 3 in 3D.

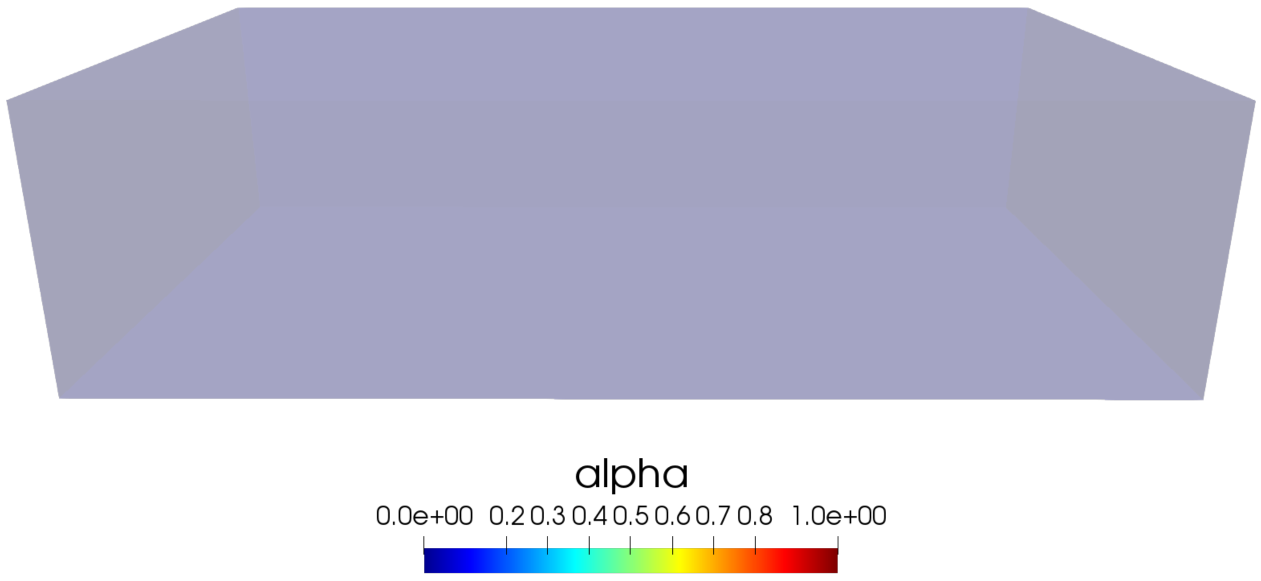


(a) $\Omega(t_{30})$.

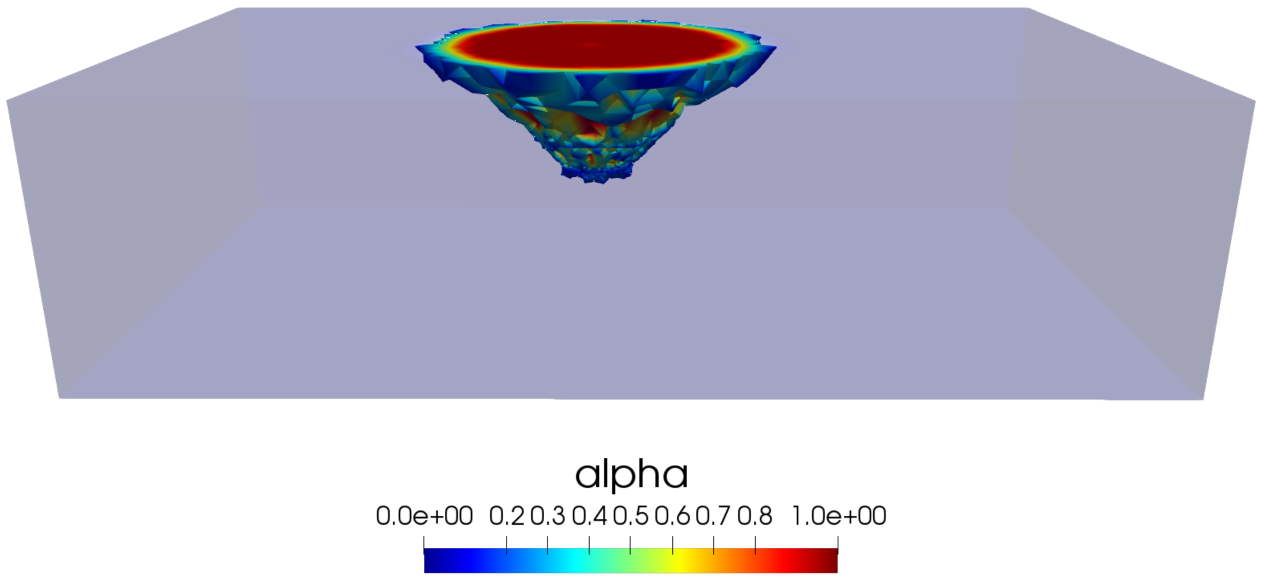


(b) $\Omega(t_{40})$.

Figure 4.23: Damage field distribution in the rock mass for Model 3 in 3D.

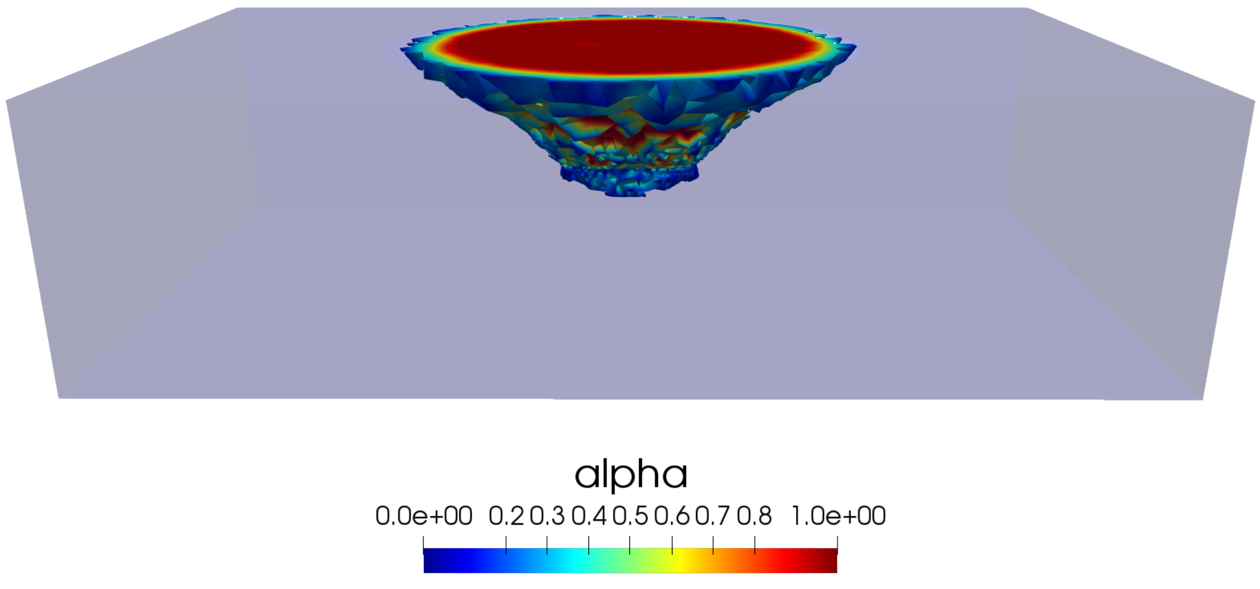


(a) $\Omega(t_0)$.

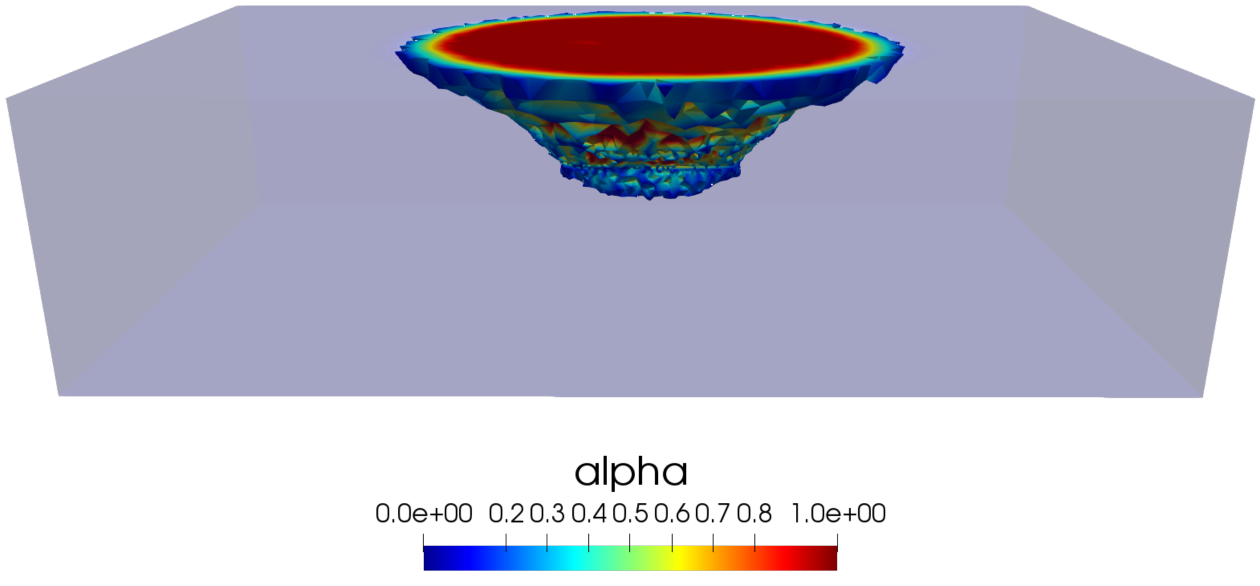


(b) $\Omega(t_{15})$.

Figure 4.24: Damage field distribution in the rock mass for Model 4 in 3D.



(a) $\Omega(t_{30})$.



(b) $\Omega(t_{40})$.

Figure 4.25: Damage field distribution in the rock mass for Model 4 in 3D.

4.3 Fast algorithm to solve the block caving process

For large-scale models like those that model a real mine, calculation times can be very large, causing the results to take a long time to be to solve this, we will proposed a new algorithm that will lower calculation times without greatly varying the solution to the damage problem.

4.3.1 Errors and new algorithm

The results, obtained by Agorithm 3 in the previous section, suffer oscillations in the errors to achieve the convergence. This causes felays in the solving tho problem, causing more time to find the solution to this problem.

Figure 4.26 shows the logarithm of the errors in each mesh iteration fot the four models. It is possible to see that, For the Model 1, the oscillations are obtained in the last time step. In the Model 2 and 3, the oscillations appear around the mesh step 30 and 31 respectively. Finally, in the Model 4, the are no oscillations.

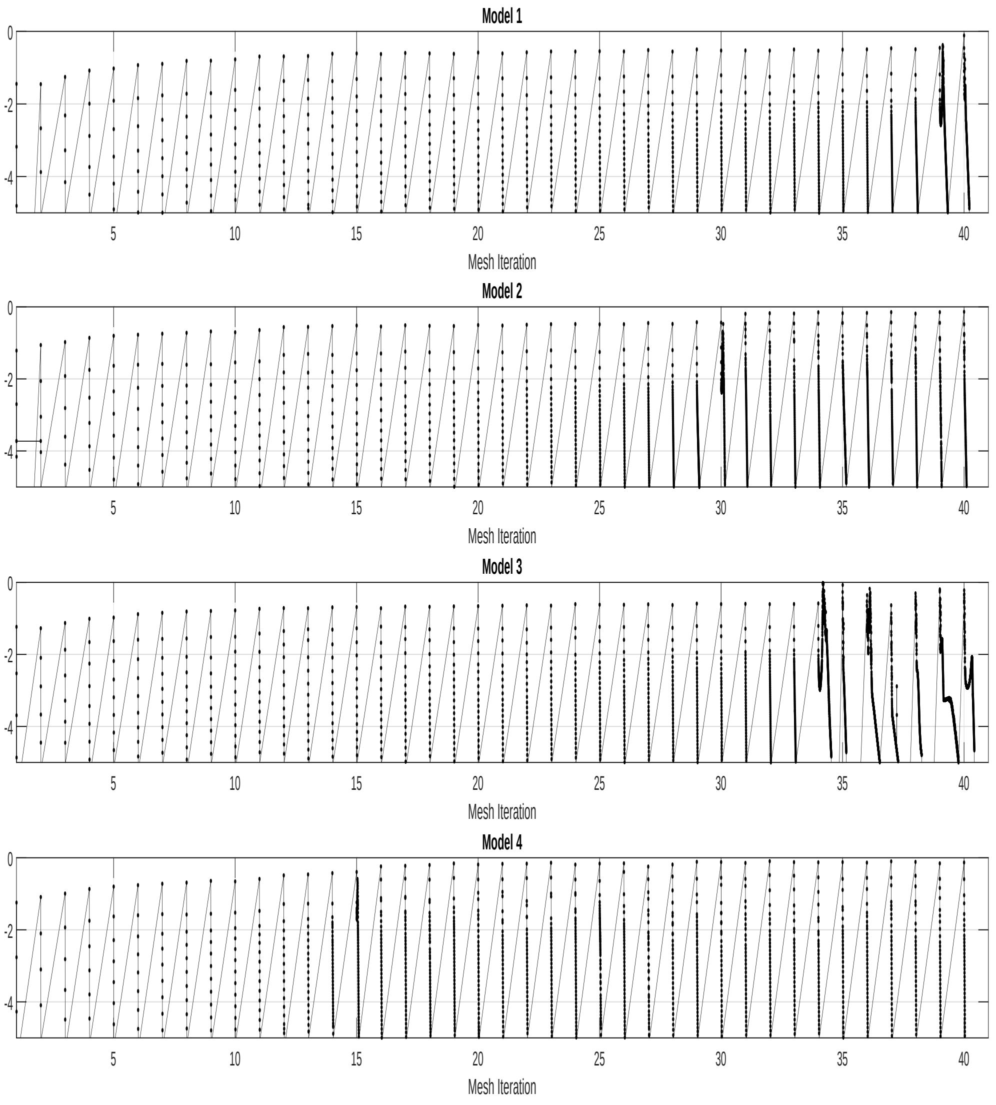


Figure 4.26: Logarithm of the error for each cavity advance.

Figure 4.27-4.30 shows a zoom of the oscillations in the mesh step of these oscillations occur. It can be seen that the oscillations appear in the same mesh step where attained its maximum value (see Figures 4.6-4.9).

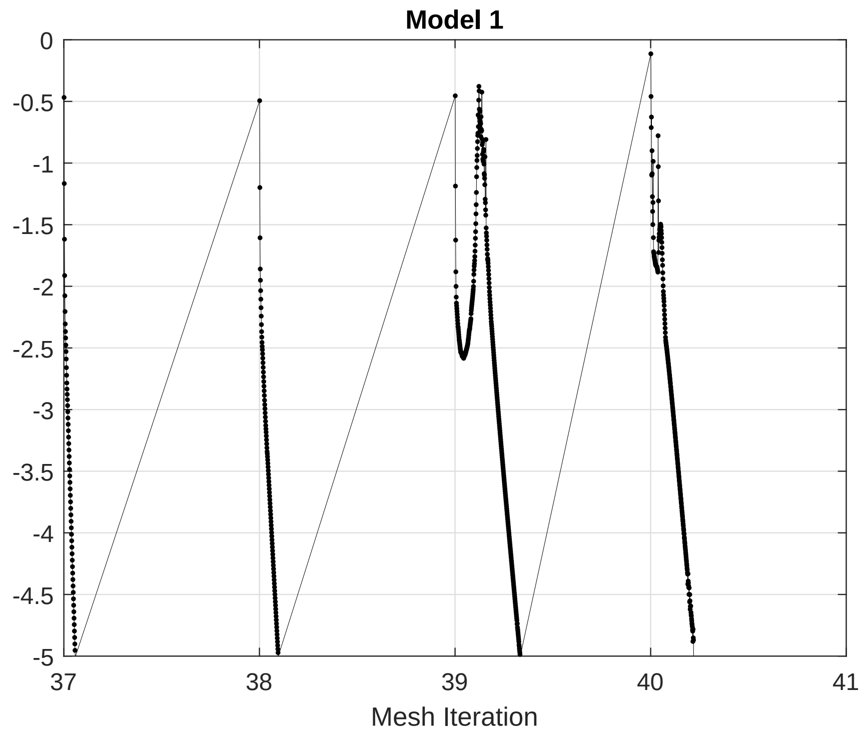


Figure 4.27: Zoom of the logarithm of the errors for Model 1.

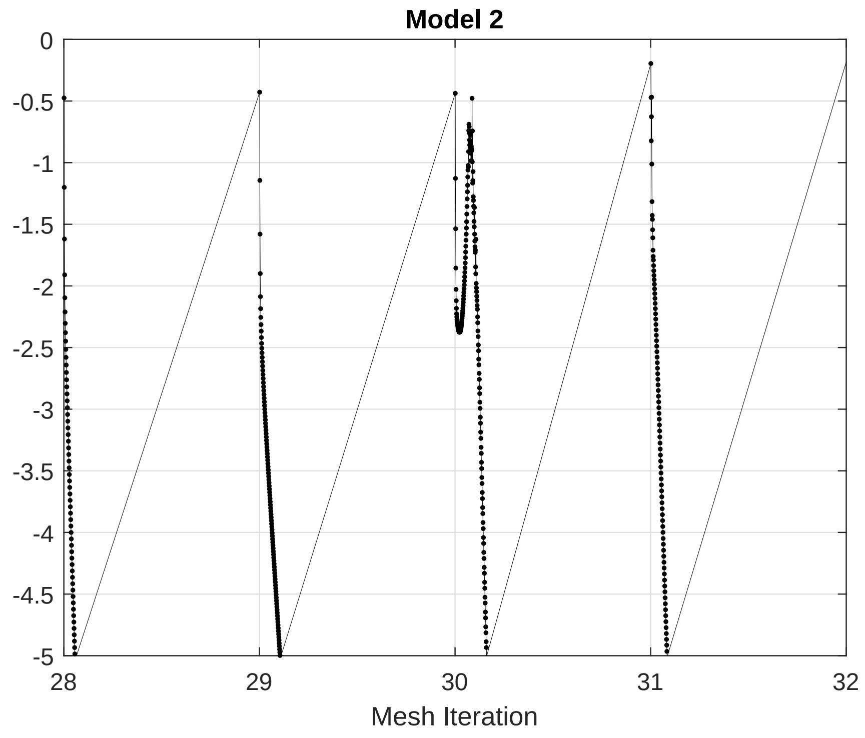


Figure 4.28: Zoom of the logarithm of the errors for Model 2.

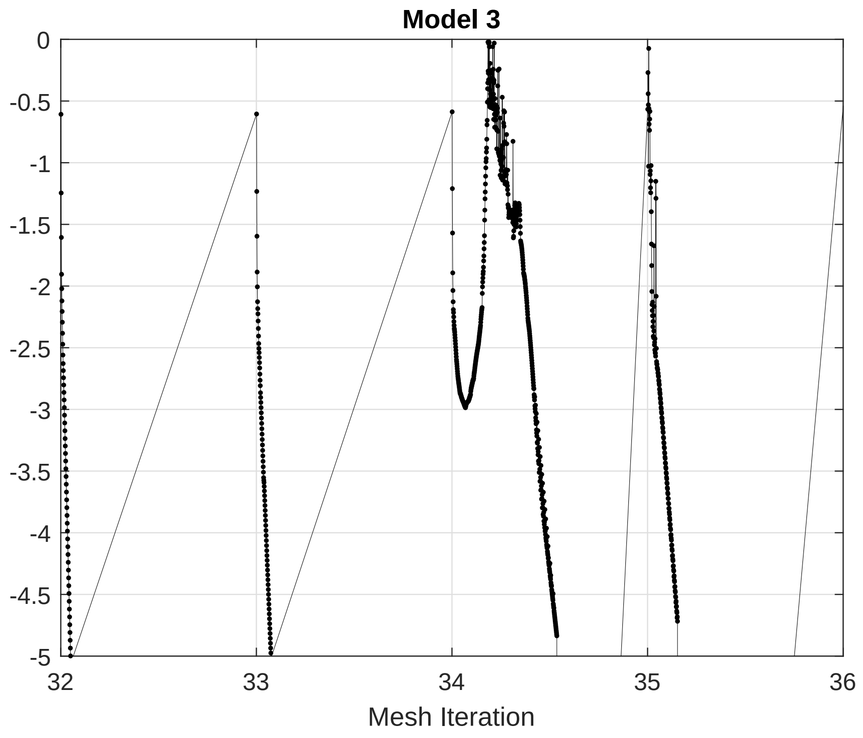


Figure 4.29: Zoom of the logarithm of the errors for Model 3.

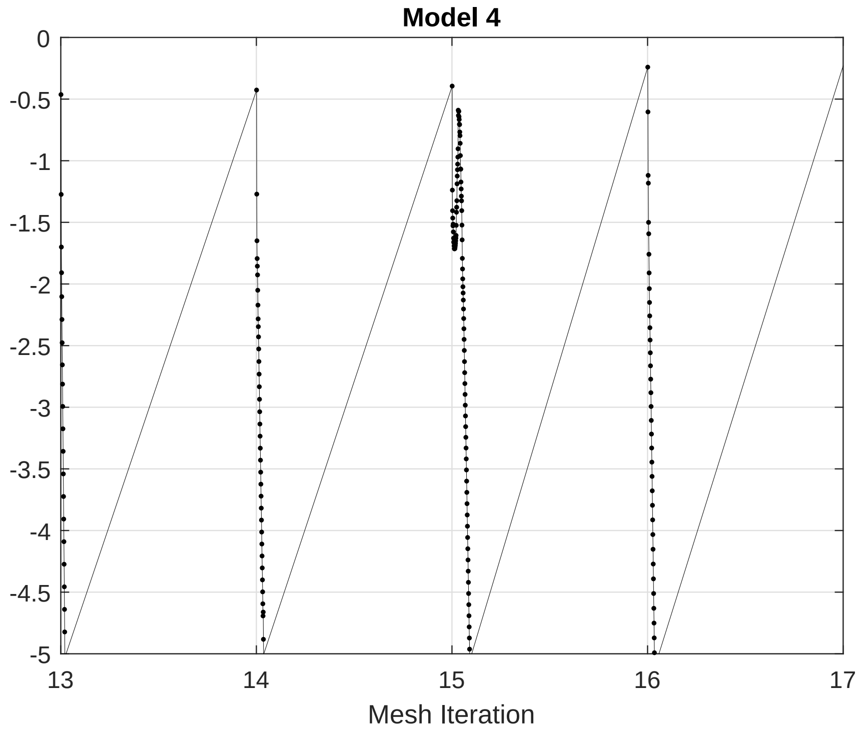


Figure 4.30: Zoom of the logarithm of the errors for Model 4.

To avoid the oscillations in the solution of the problem and with this improve the time in which the solution is obtained, we propose a modification in the Algorithm 3. This modification consists in considering, when the error grows, and arrangement of the form

$$\alpha^{(p)} = C_L\alpha^{(p-1)} + (1 - C_L)\alpha^{(p)}, \tag{4.20}$$

where this arrangement is a combination between the solution in iteration (p) and the previous solution in the iteration $(p-1)$, with $C_L \in (0,1)$ a constant. This new approach to solve the approach is summarized in Algorithm 2, where the FEniCS implementation can be seen in Appendix B.2.

Algorithm 2 Fast algorithm to solve the damage problem

```
 1: return Solution at time step $t_\mathrm{i}$.
 2: Given $(u_{\mathrm{i}-1}, \alpha_{\mathrm{i}-1})$, the sate at the previous loading step.
 3: Set $(u^{(0)}, \alpha^{(0)}) := (u_{\mathrm{i}-1}, \alpha_{\mathrm{i}-1})$ and $\mathrm{error}^{(0)} = 1.0$
 4: while $\mathrm{error}^{(p)}$ >tolerance do
 5:    Solve $u^{(p)}$ from (3.33) with $\alpha^{(p-1)}$.
 6:    Find $\alpha^{(p)} := \underset{\alpha\in\mathcal{D}(\alpha_{\mathrm{i}-1})}{\arg\min}\ \overline{\mathcal{P}}(u^{(p)}, \alpha)$.
 7:    $\mathrm{error}^{(p)} = \|\alpha^{(p-1)} - \alpha^{(p)}\|_\infty$.
 8:    while $\mathrm{error}^{(p)}$ >$\mathrm{error}^{(p-1)}$ do
 9:       $\overline{\alpha}^{(p)} = C_L\alpha^{(p-1)} + (1 - C_L)\alpha^{(p)}$.
10:       $\alpha^{(p)} = \overline{\alpha}^{(p)}$.
11:       $\mathrm{error}^{(p)} = \|\alpha^{(p-1)} - \alpha^{(p)}\|_\infty$.
12:    end while
13: end while
14: Set $(\mathbf{u}_\mathrm{i}, \alpha_\mathrm{i}) = (u^p, \alpha^p)$.
```

4.3.2 numerical results

For this new algorithm, we consider the same previous test, where we apply the Algorithm 2 to all models. The Figures 4.31-4.34 display the logarithm of the error for $C_L = 0.5$ and $C_L = 0.9$ respectively and for the different models. It is possible to see that the oscillations of the error disappear that with a smaller amount of iterations the convergence of the algorithm is attained, causing with this that the solution is obtained in a lower time.

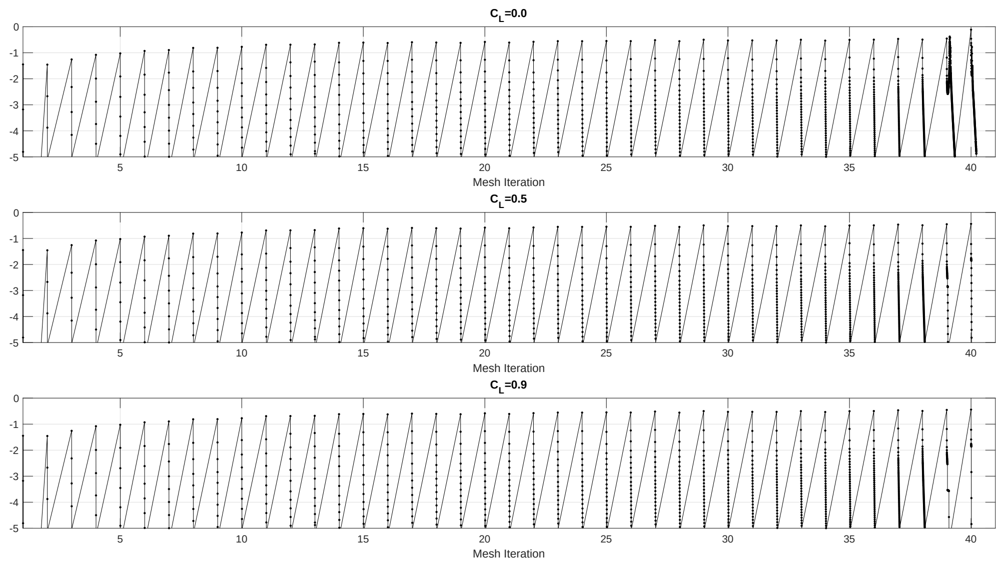


Figure 4.31: Logarithm of the error in the Model 1 for different values of C_L.

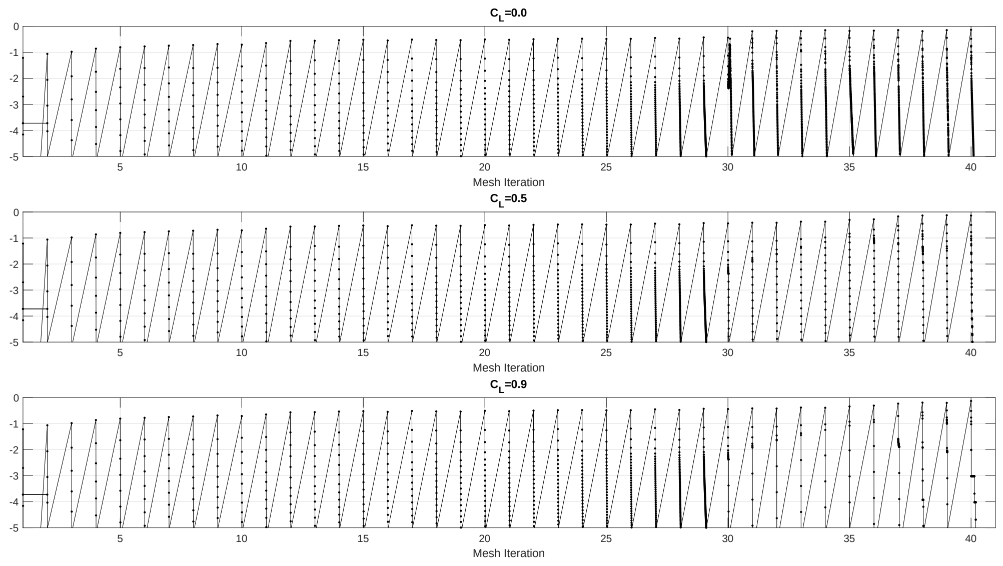


Figure 4.32: Logarithm of the error in the Model 2 for different values of C_L.

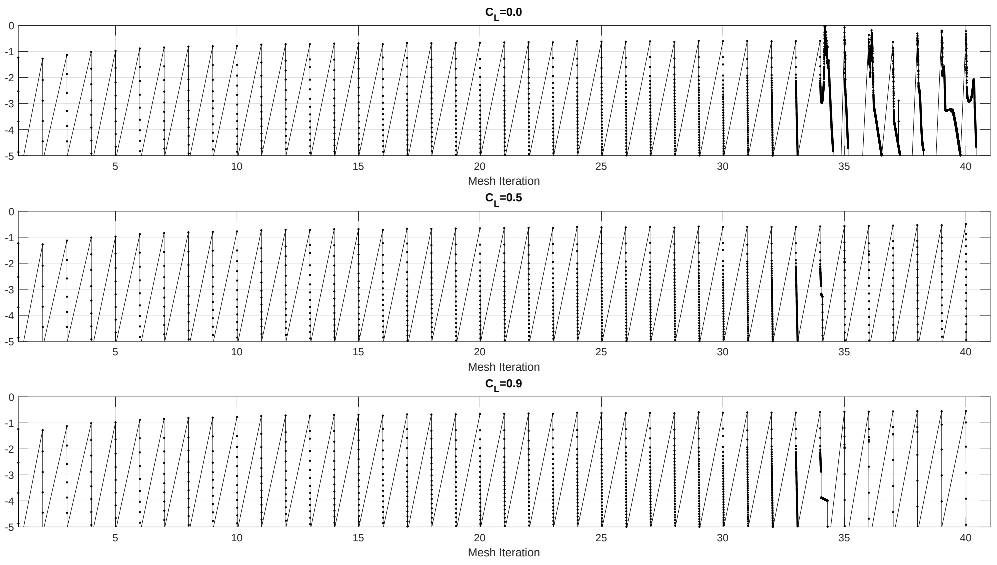


Figure 4.33: Logarithm of the error in the Model 3 for different values of C_L.

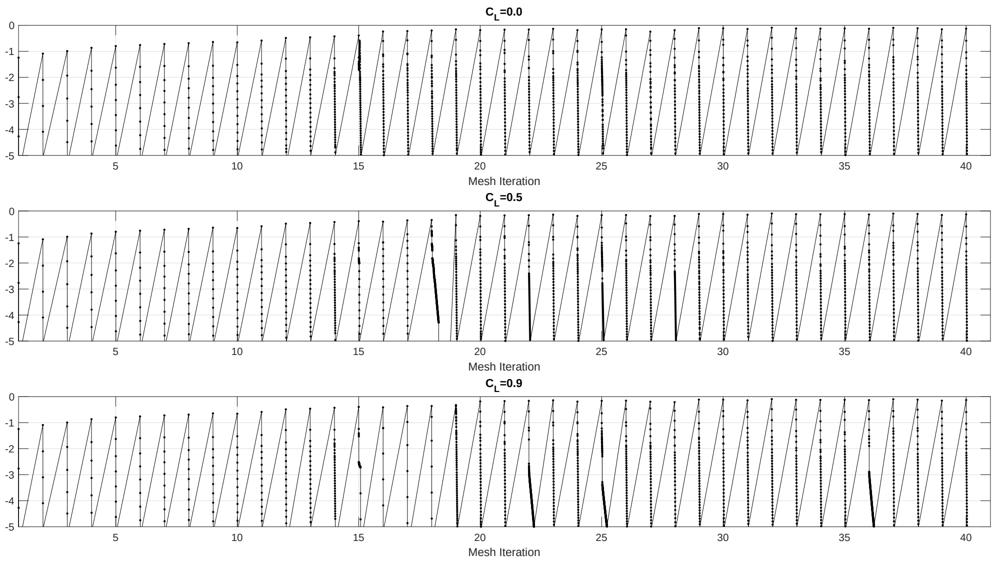


Figure 4.34: Logarithm of the error in the Model 4 for different values of C_L.

Figures 4.35-4.38 summarized the maximum values of α for the different values of C_L and different models in each mesh iteration. The figure displays the evolution of this maximum value and it can be see that the value of the constant C_L is higher. It is possible see that the value of the maximum damage is the same for the different values of C_L in the final mesh step, except in Model 1 and 3, that the maximum value is lower, for this reason we will consider $C_L = 0.9$ since this value require less iterations to achieve the convergence.

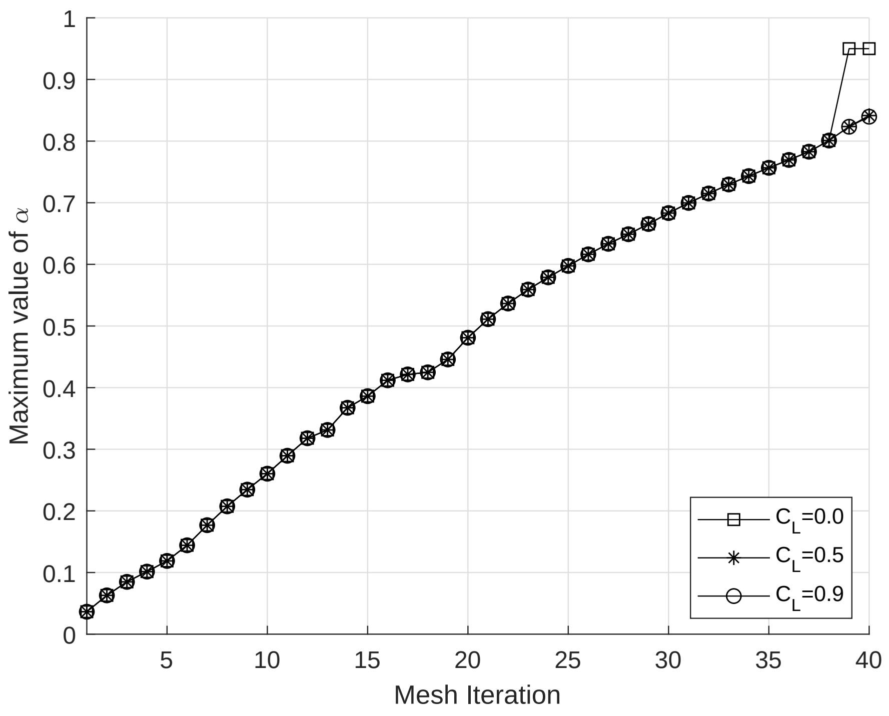


Figure 4.35: Maximum damage value in the Model 1 for each mesh step for different values of C_L.

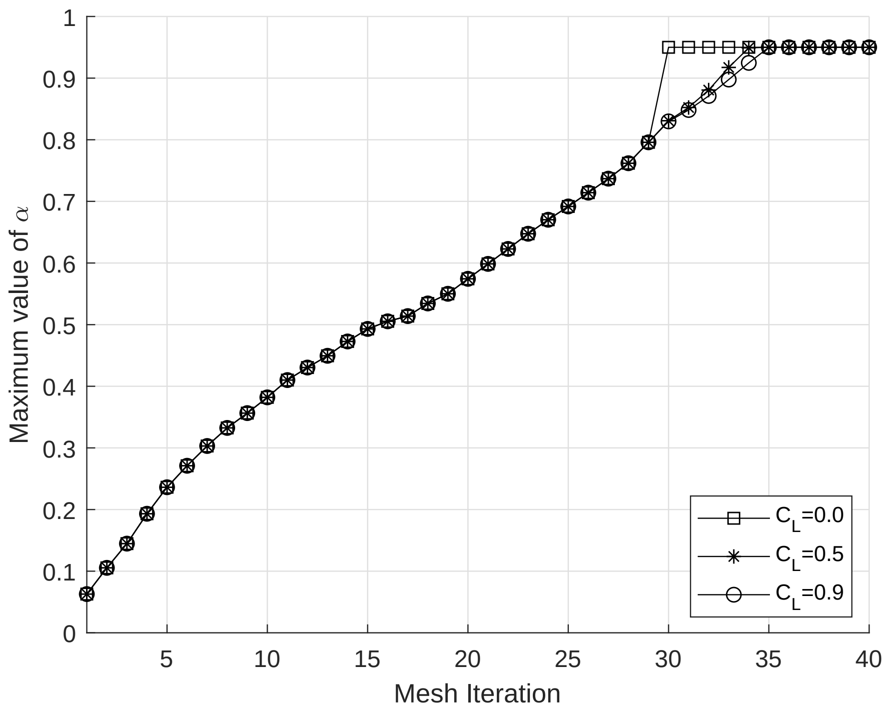


Figure 4.36: Maximum damage value in the Model 2 for each mesh step for different values of C_L.

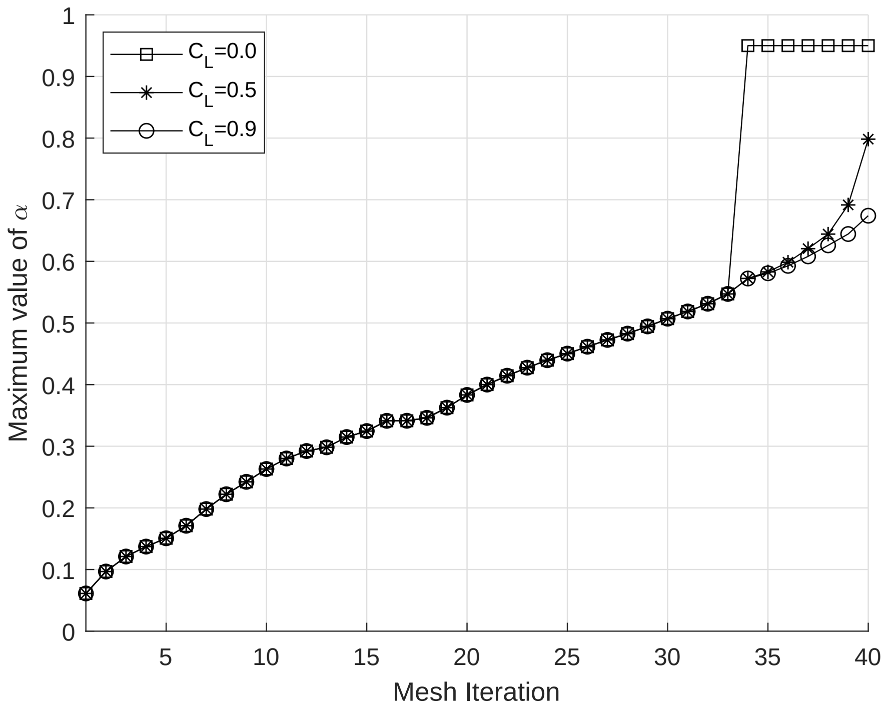


Figure 4.37: Maximum damage value in the Model 3 for each mesh step for different values of C_L.

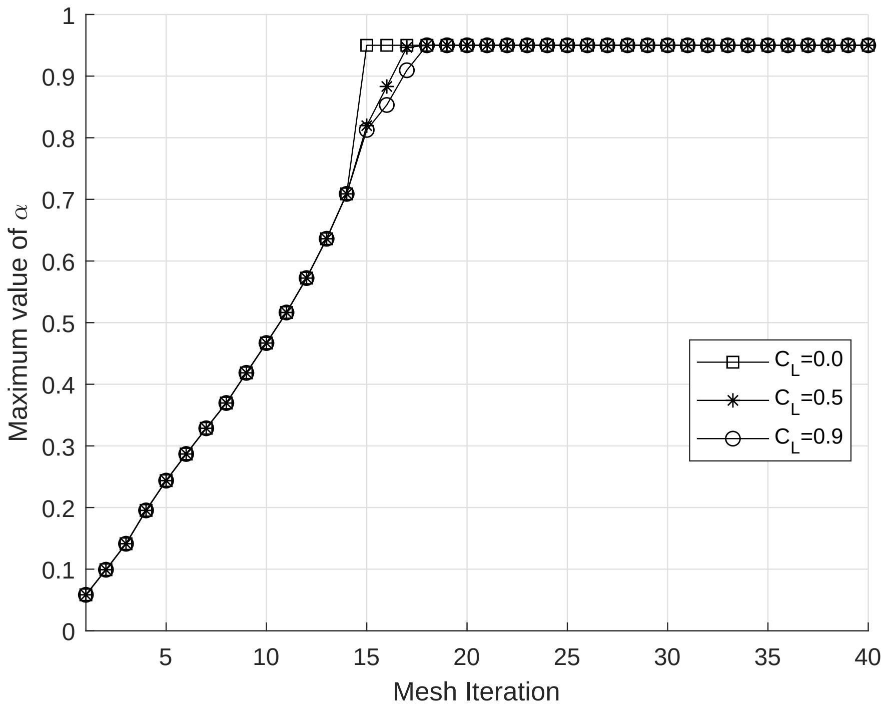
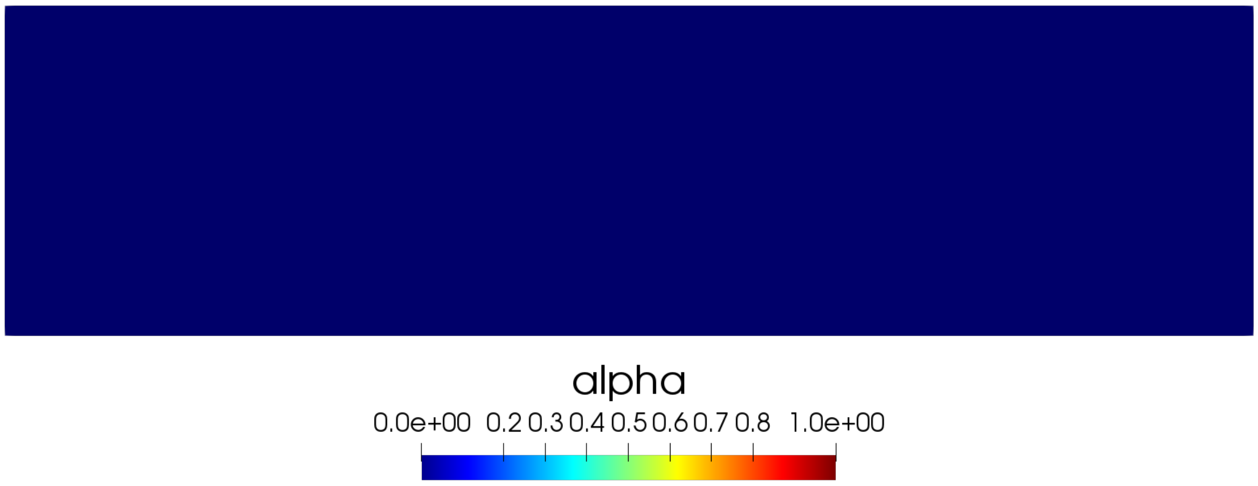
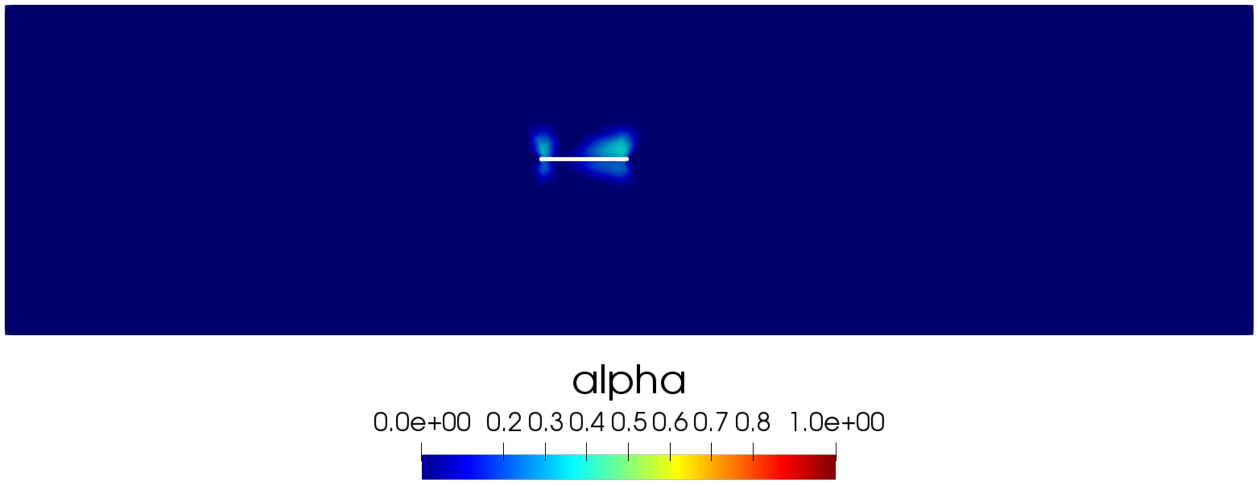


Figure 4.38: Maximum damage value in the Model 4 for each mesh step for different values of C_L.

Figures 4.39-4.46 display the distribution of the damage when the cavity advances in time using the Algorithm 2. In the same way as using the Algorithm 3, the damage is closed to zero almost everywhere but with a lower value and not reaching the upper boundary of the domain in the Model 1. In the Models 2 and 3 the material is fully damaged over the cavity in the same way as the Algorithm 3. The Figures 4.47-4.54 display the distribution of the damage in 3D.

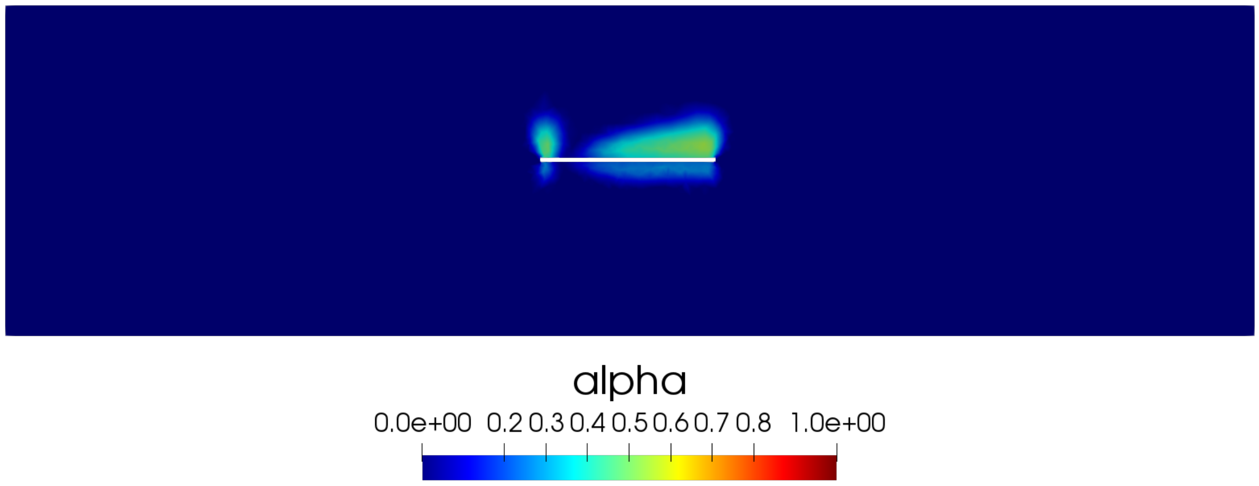


(a) $\Omega(t_0)$.

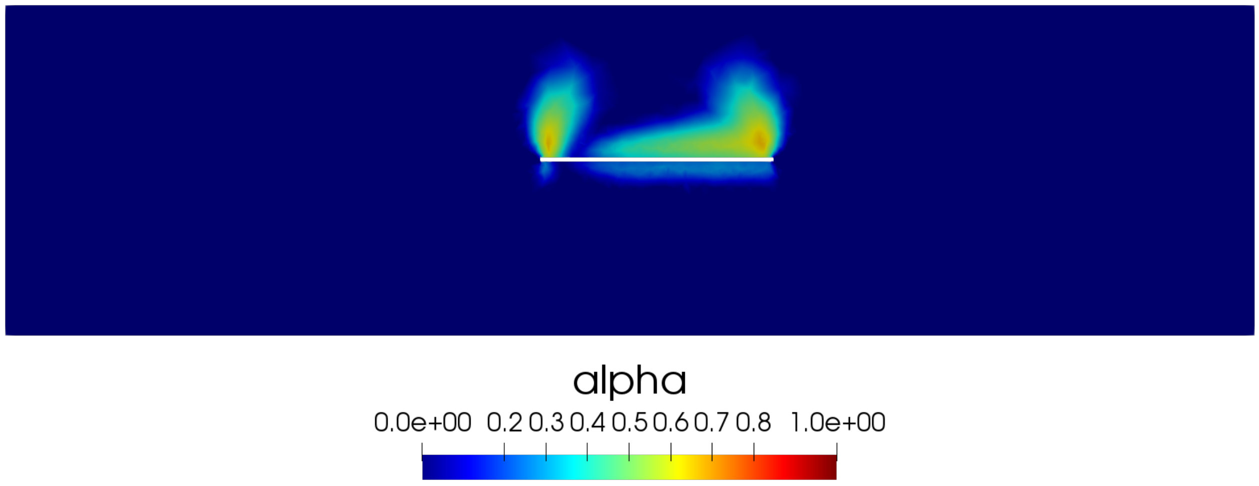


(b) $\Omega(t_{15})$.

Figure 4.39: Damage field distribution in the rock mass for Model 1 and $C_L = 0.9$.

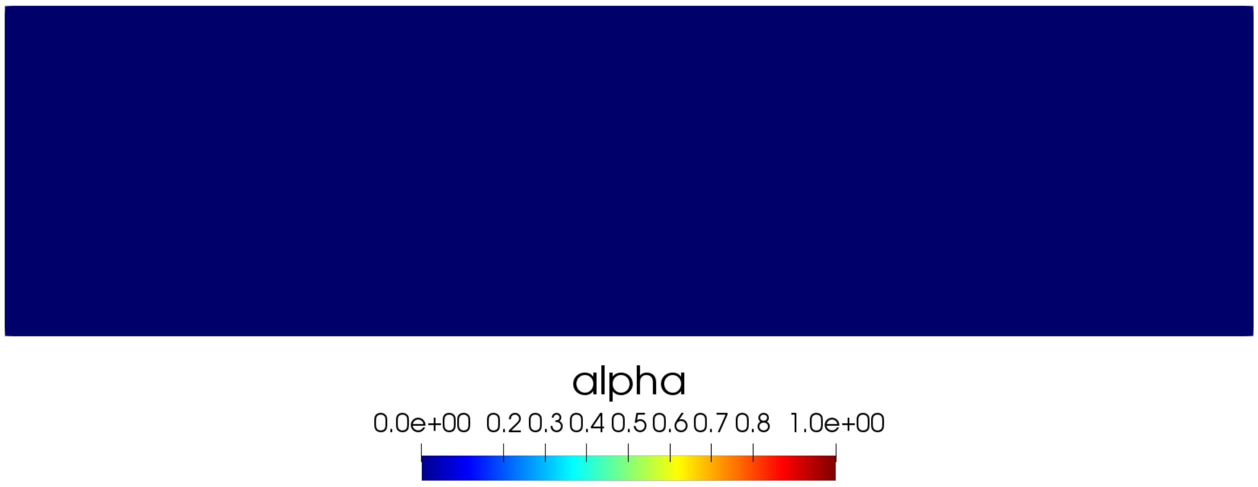


(a) $\Omega(t_{30})$.

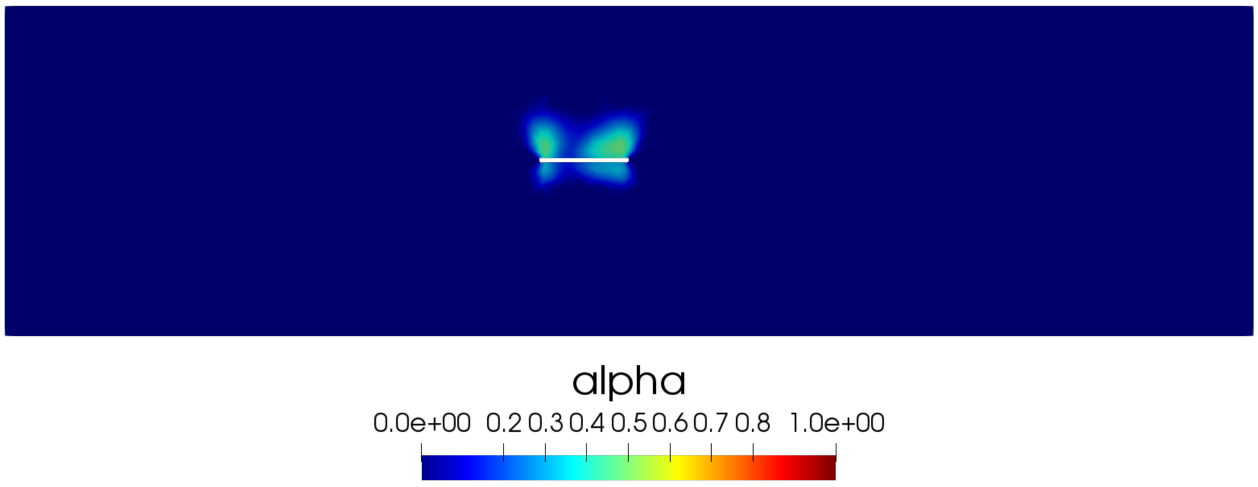


(b) $\Omega(t_{40})$.

Figure 4.40: Damage field distribution in the rock mass for Model 1 and $C_L = 0.9$.

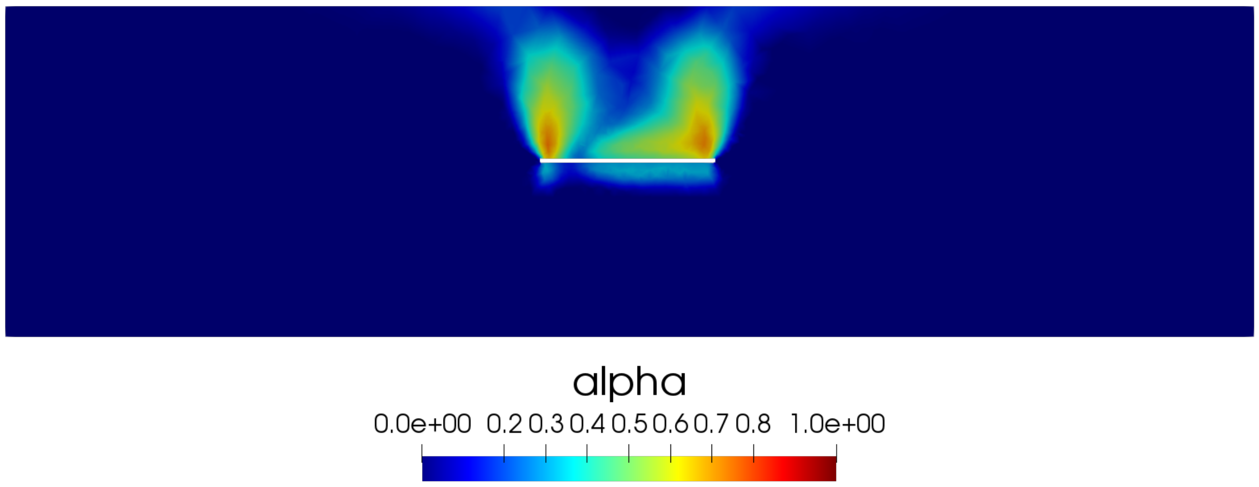


(a) $\Omega(t_0)$.

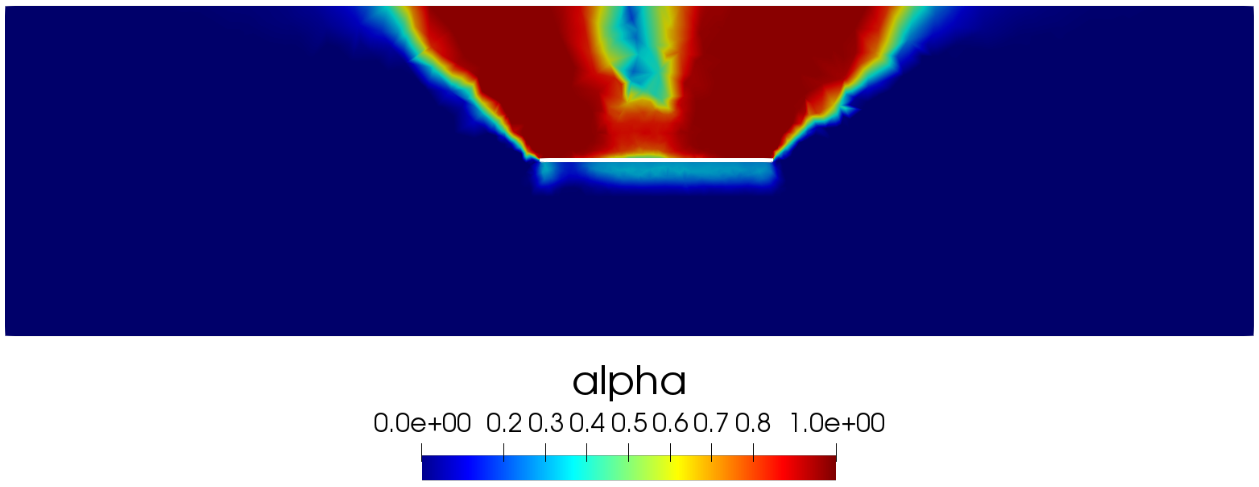


(b) $\Omega(t_{15})$.

Figure 4.41: Damage field distribution in the rock mass for Model 2 and $C_L = 0.9$.

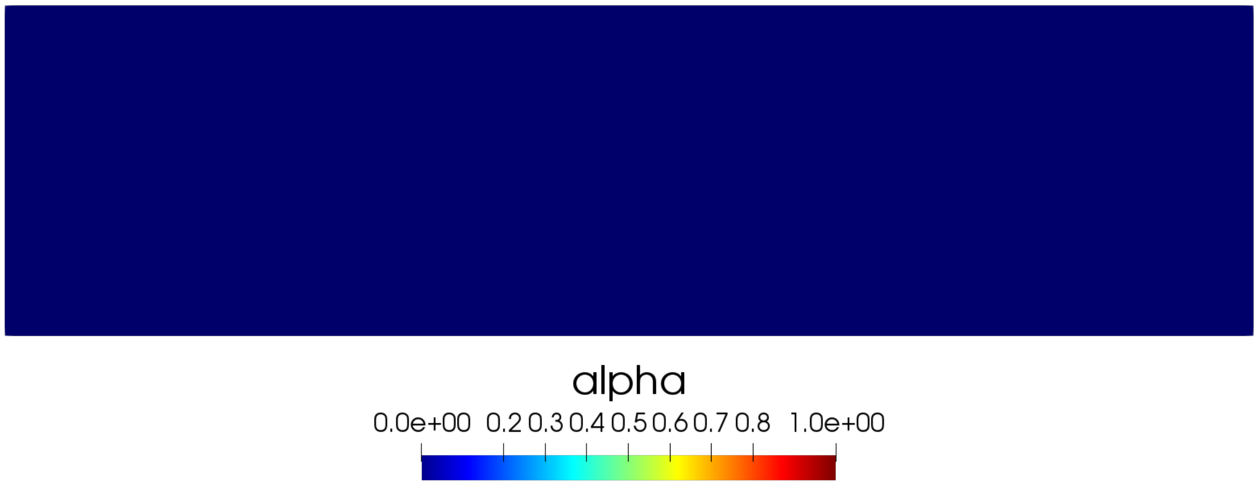


(a) $\Omega(t_{30})$.

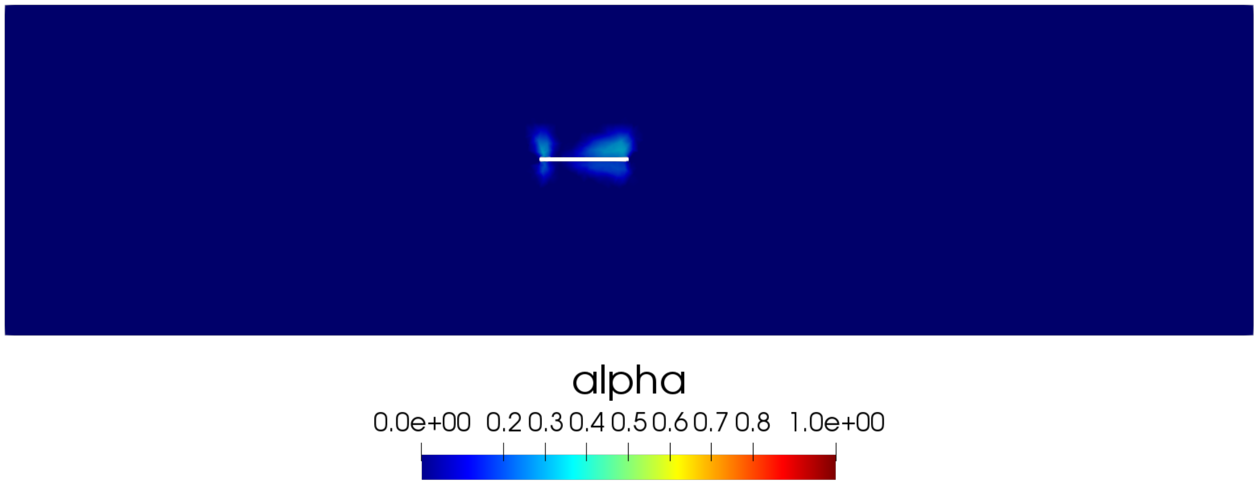


(b) $\Omega(t_{40})$.

Figure 4.42: Damage field distribution in the rock mass for Model 2 and $C_L = 0.9$.

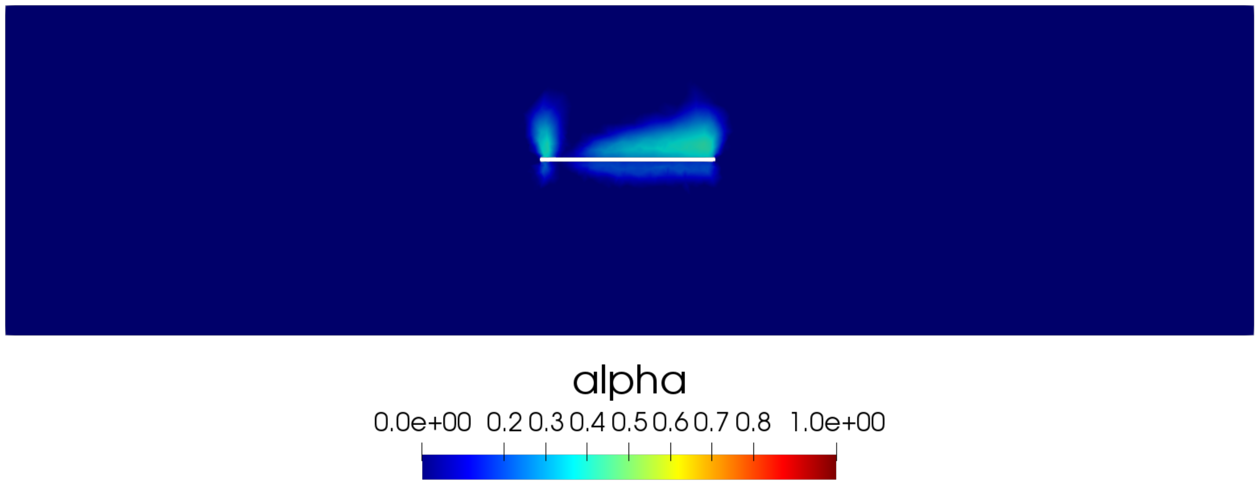


(a) $\Omega(t_0)$.

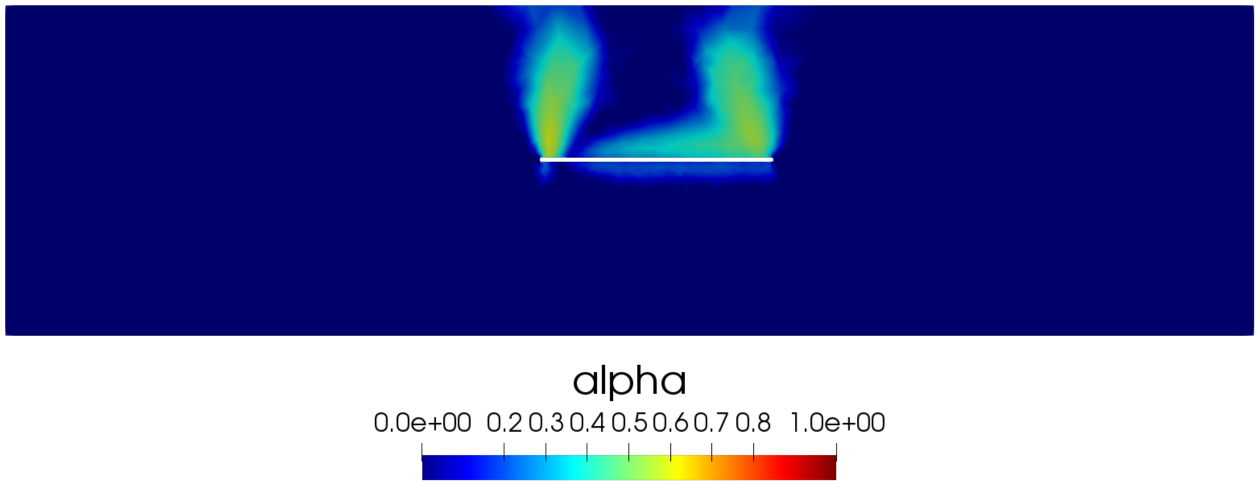


(b) $\Omega(t_{15})$.

Figure 4.43: Damage field distribution in the rock mass for Model 3 and $C_L = 0.9$.

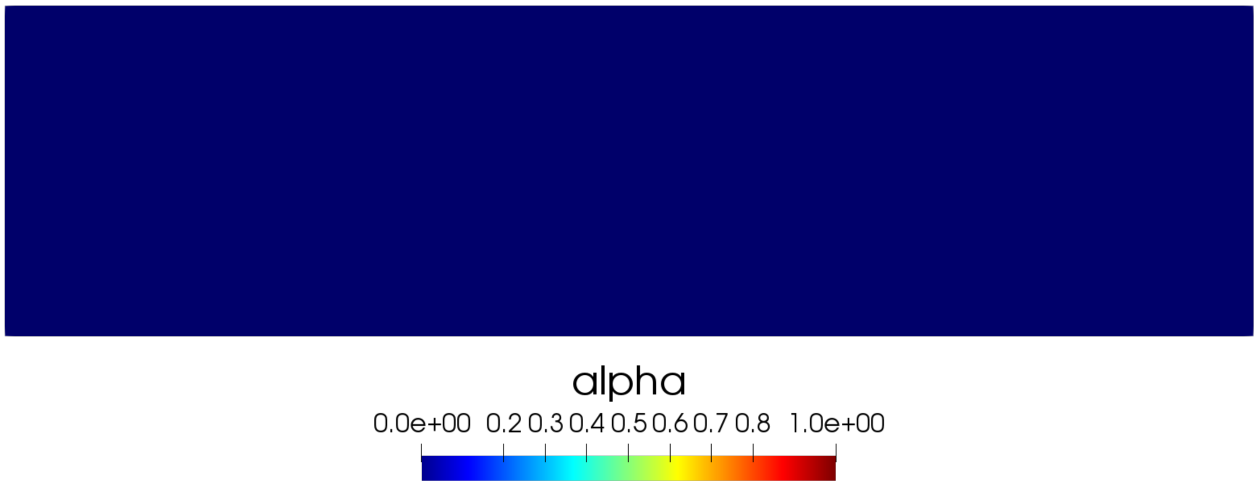


(a) $\Omega(t_{30})$.

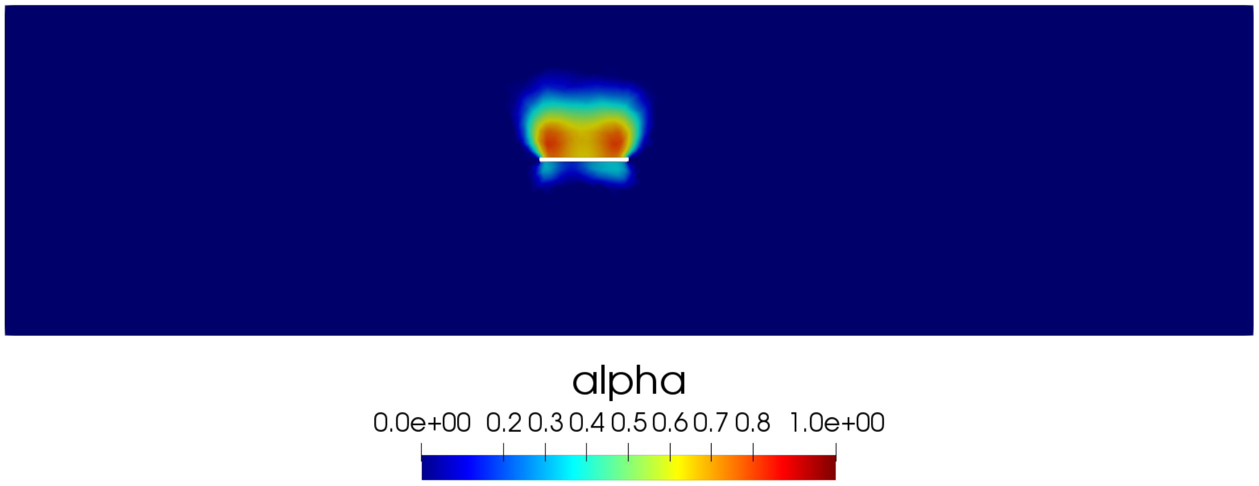


(b) $\Omega(t_{40})$.

Figure 4.44: Damage field distribution in the rock mass for Model 3 and $C_L = 0.9$.

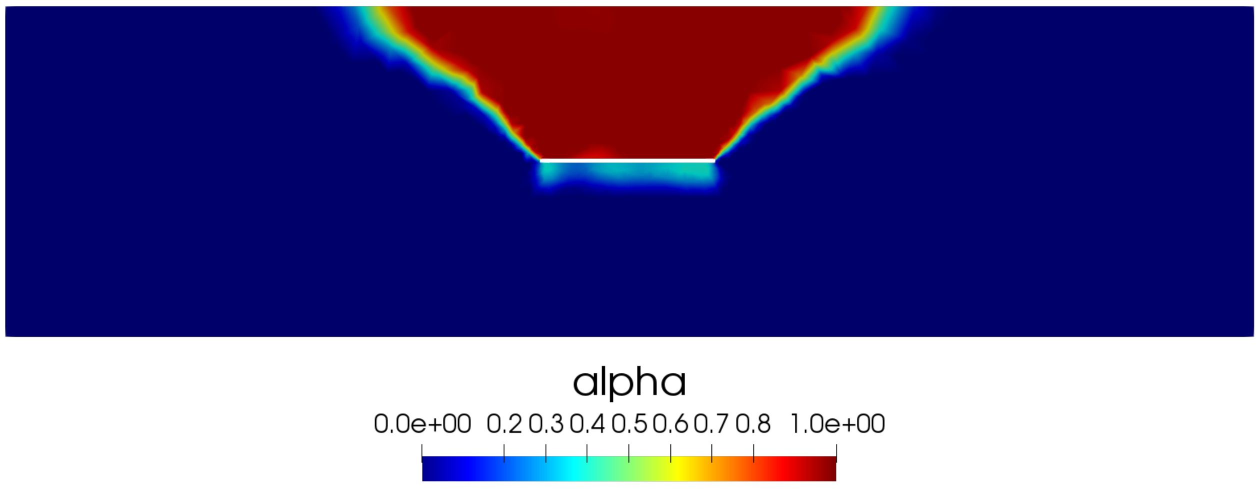


(a) $\Omega(t_0)$.

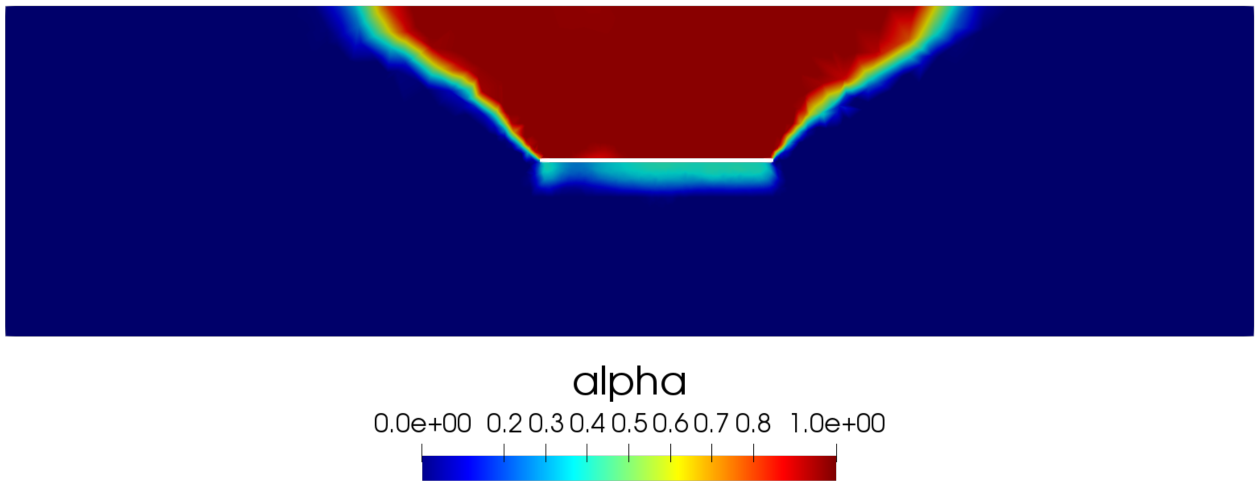


(b) $\Omega(t_{15})$.

Figure 4.45: Damage field distribution in the rock mass for Model 4 and $C_L = 0.9$.

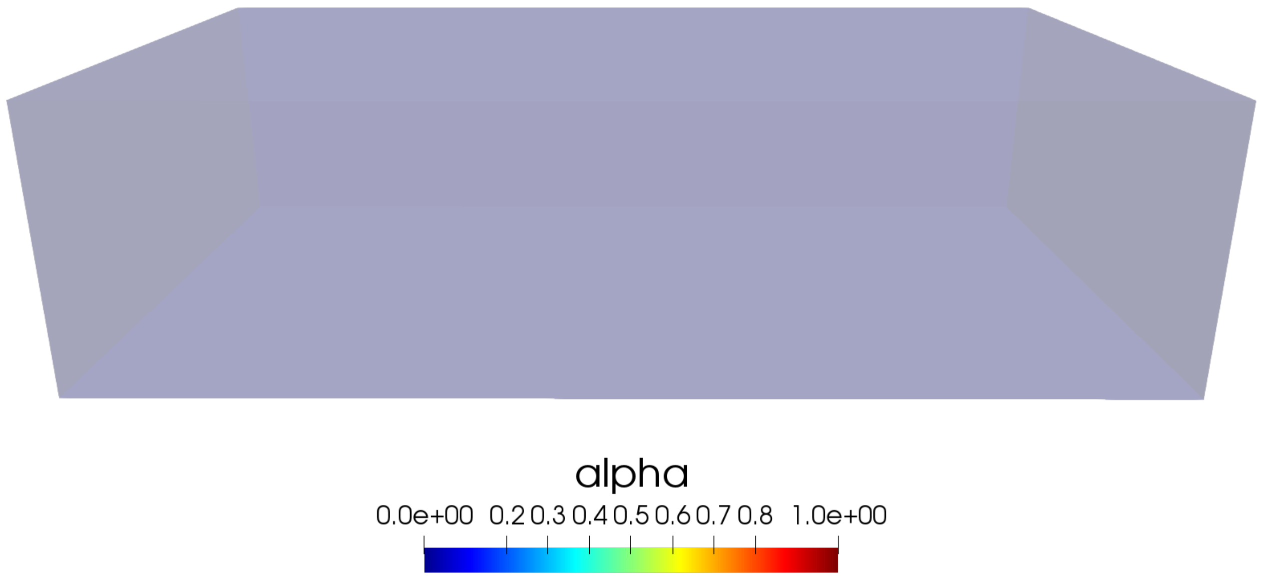


(a) $\Omega(t_{30})$.

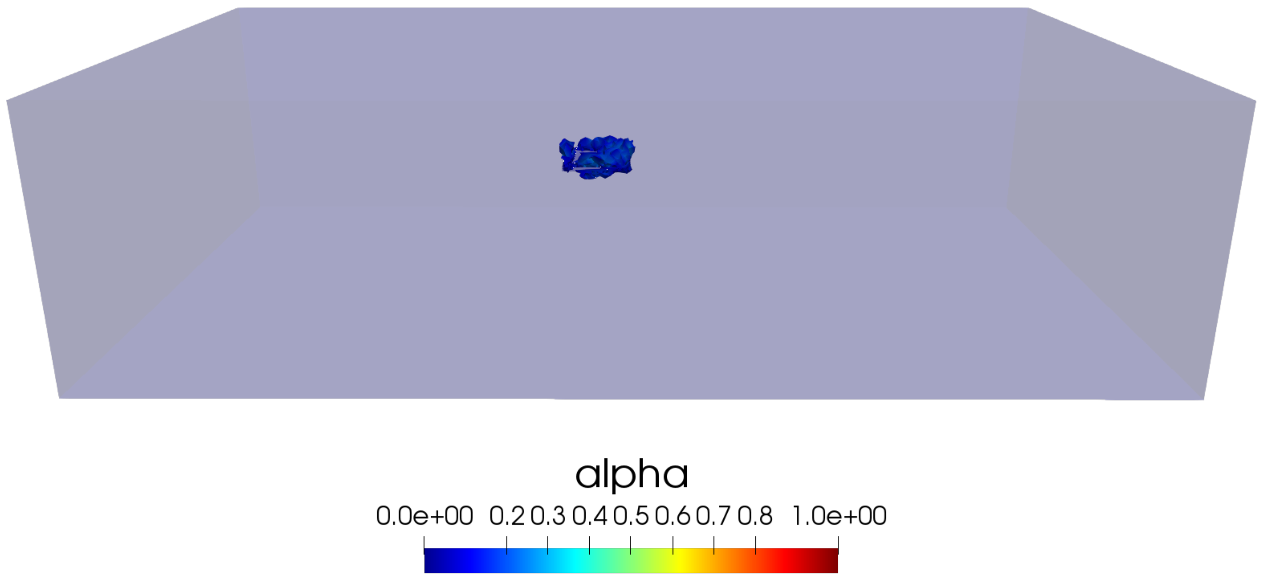


(b) $\Omega(t_{40})$.

Figure 4.46: Damage field distribution in the rock mass for Model 4 and $C_L = 0.9$.

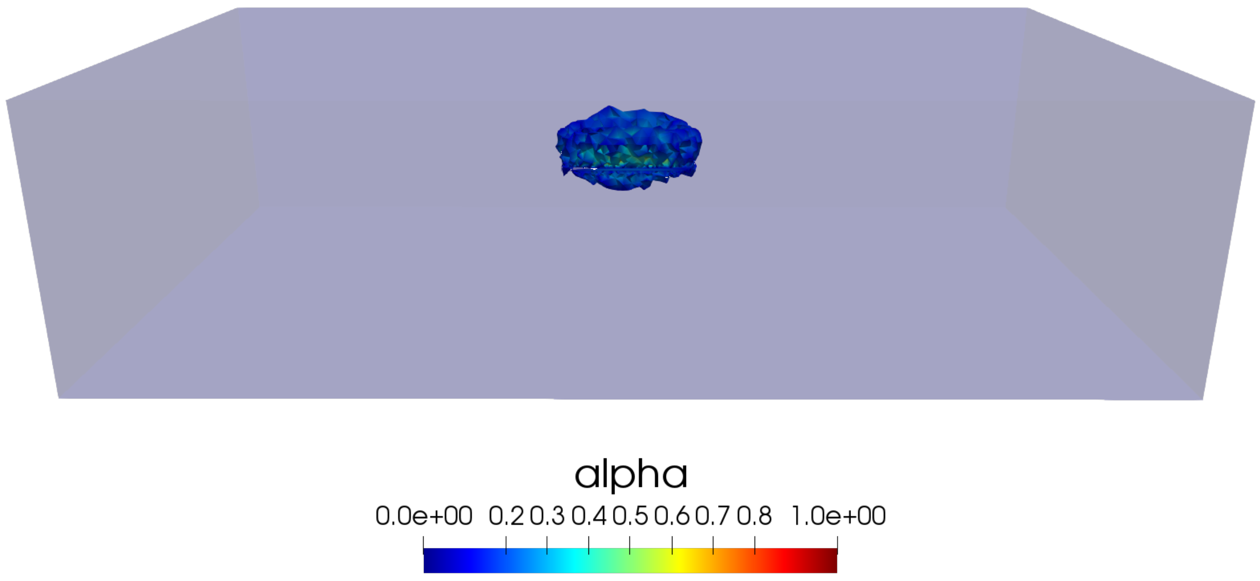


(a) $\Omega(t_0)$.

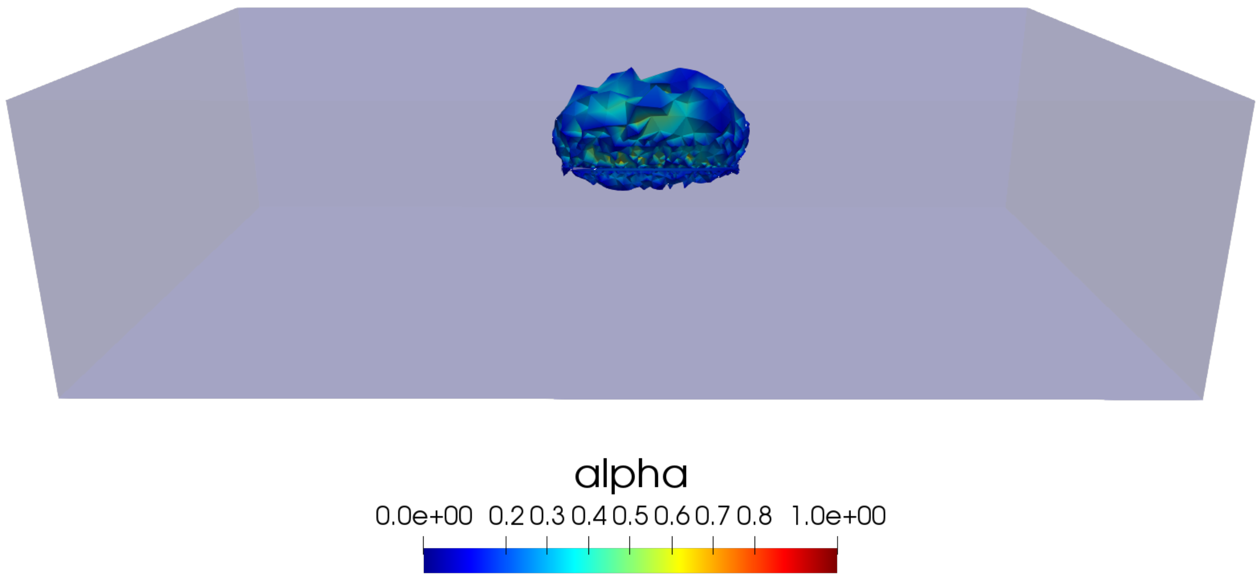


(b) $\Omega(t_{15})$.

Figure 4.47: Damage field distribution in the rock mass for Model 1 in 3D considering $C_L = 0.9$.

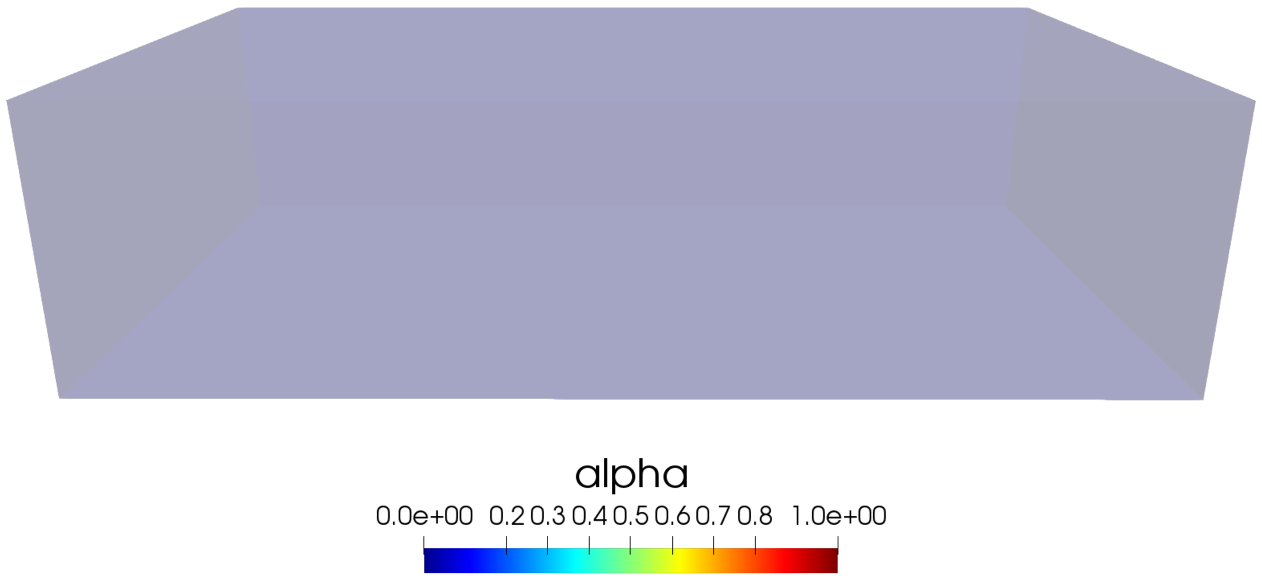


(a) $\Omega(t_{30})$.

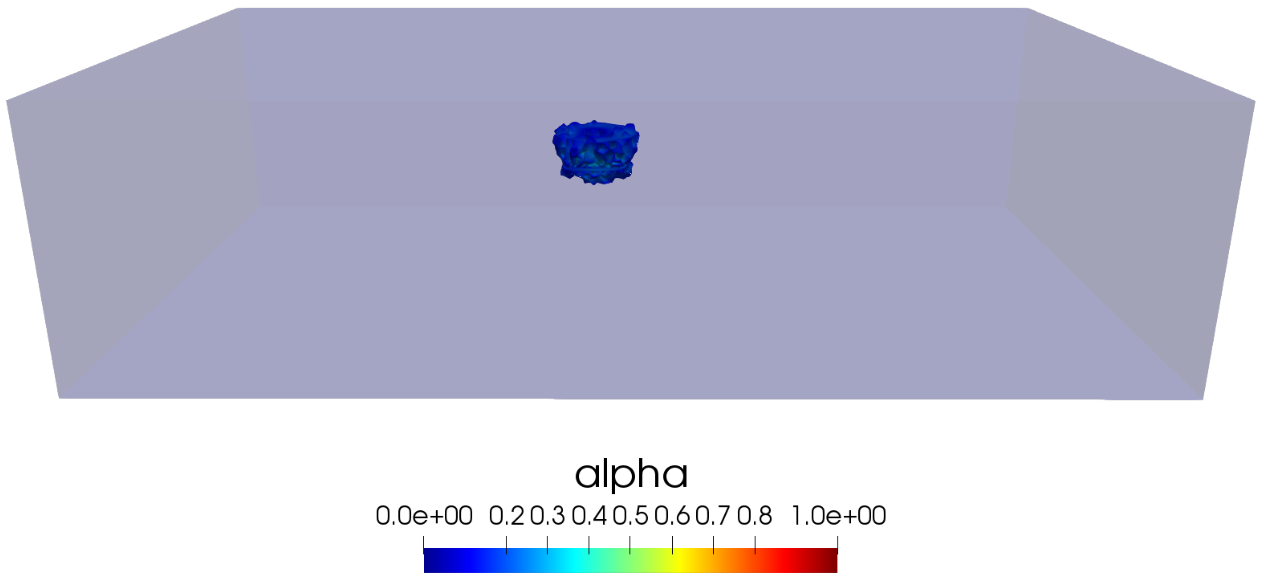


(b) $\Omega(t_{40})$.

Figure 4.48: Damage field distribution in the rock mass for Model 1 in 3D considering $C_L = 0.9$.

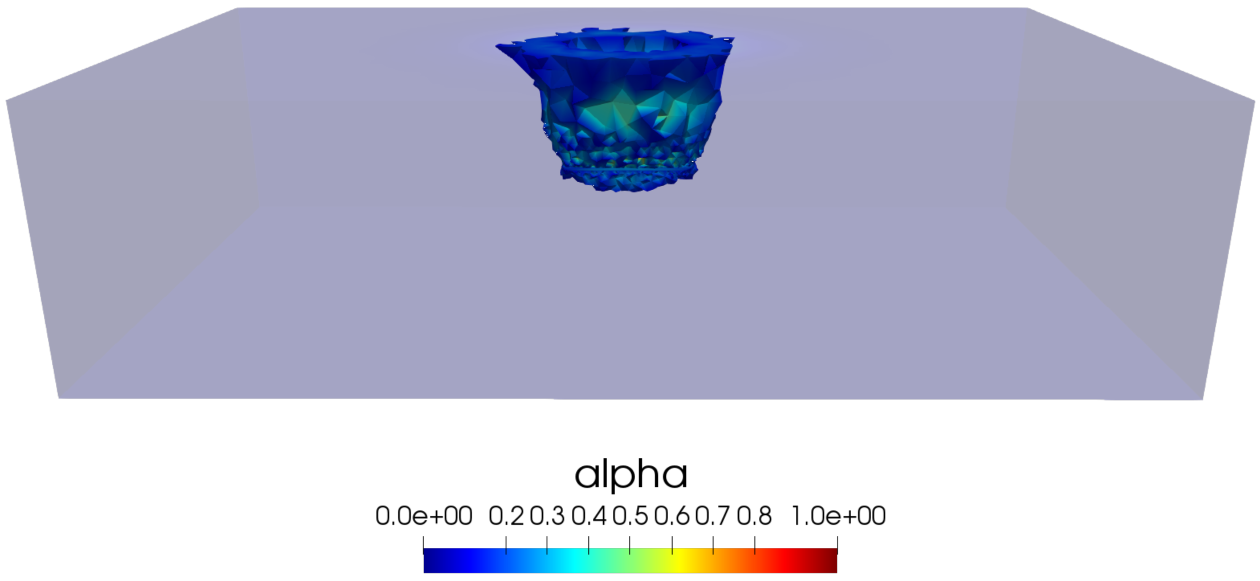


(a) $\Omega(t_0)$.

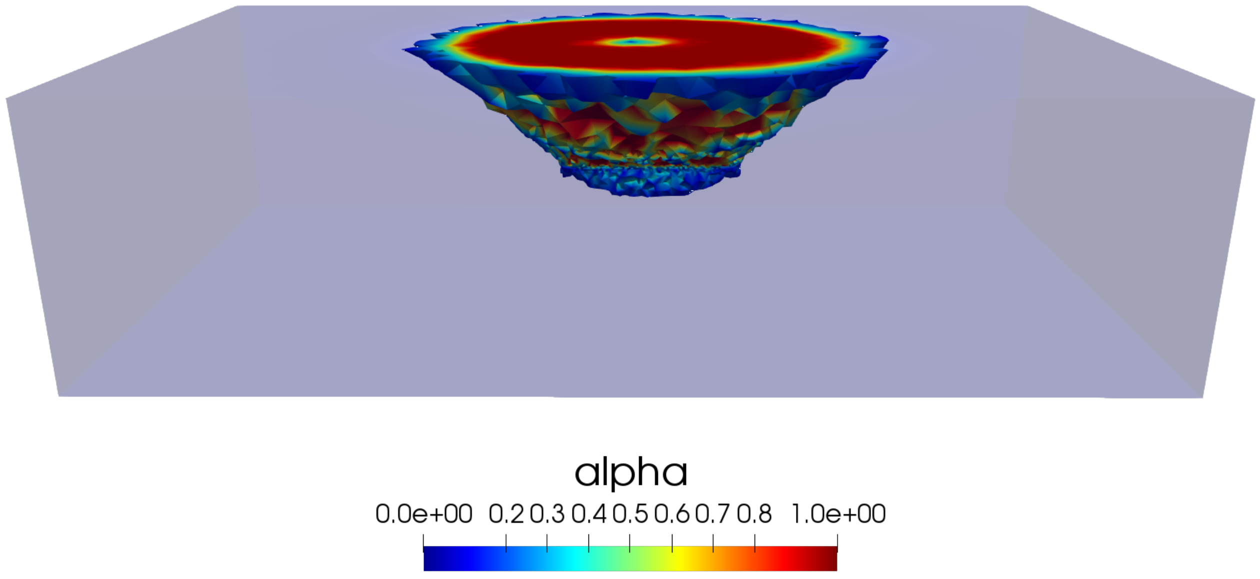


(b) $\Omega(t_{15})$.

Figure 4.49: Damage field distribution in the rock mass for Model 2 in 3D considering $C_L = 0.9$.

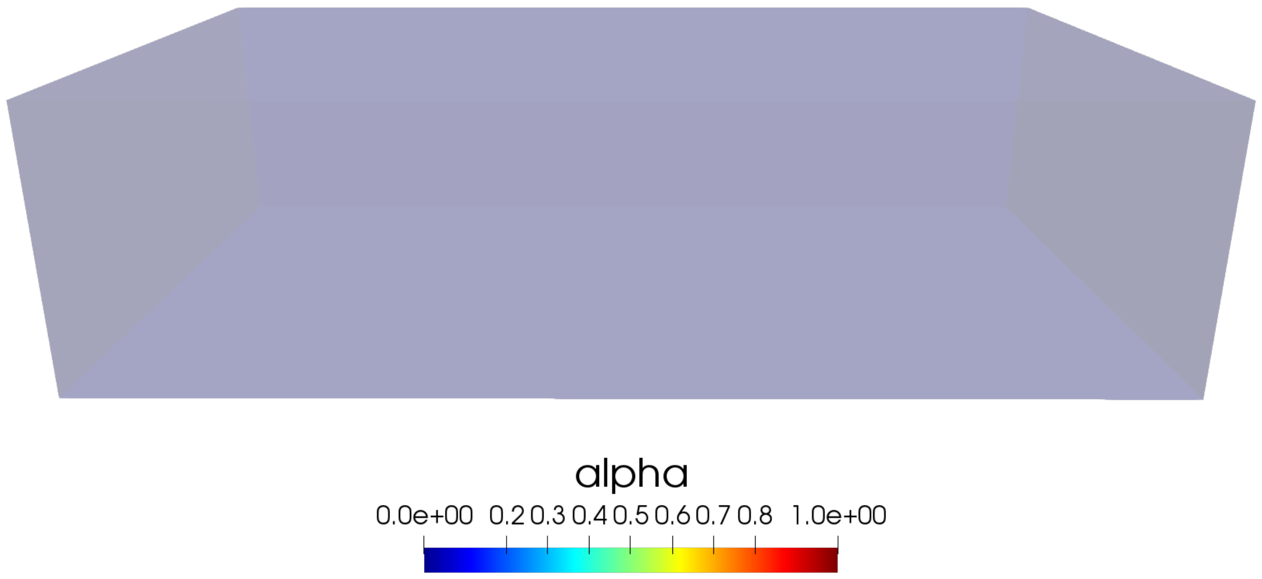


(a) $\Omega(t_{30})$.

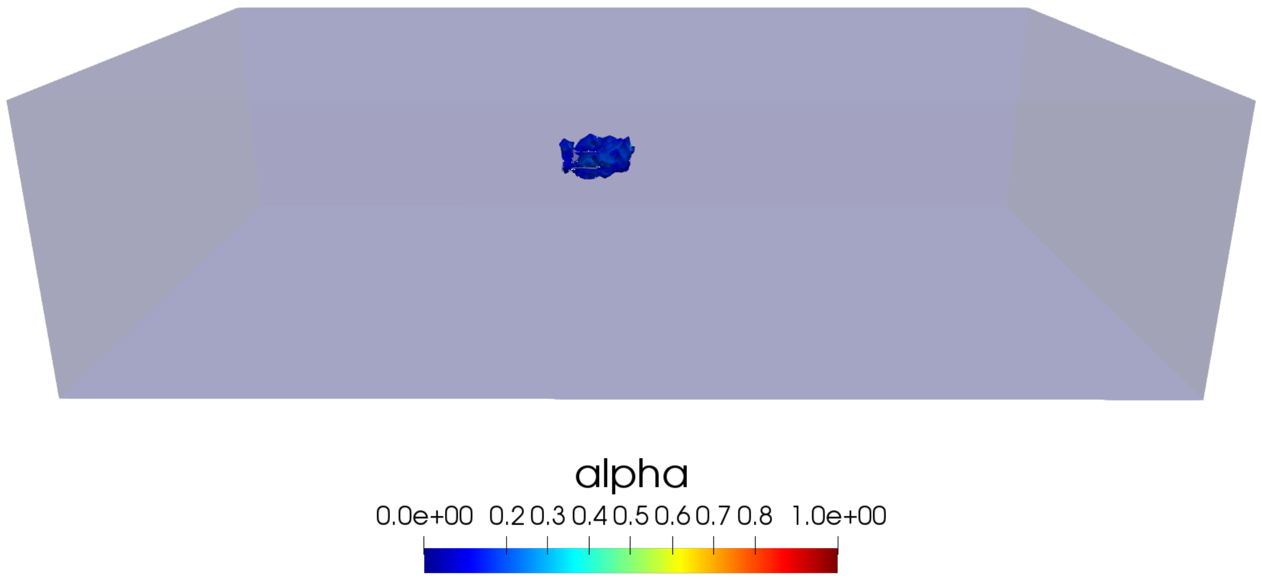


(b) $\Omega(t_{40})$.

Figure 4.50: Damage field distribution in the rock mass for Model 2 in 3D considering $C_L = 0.9$.

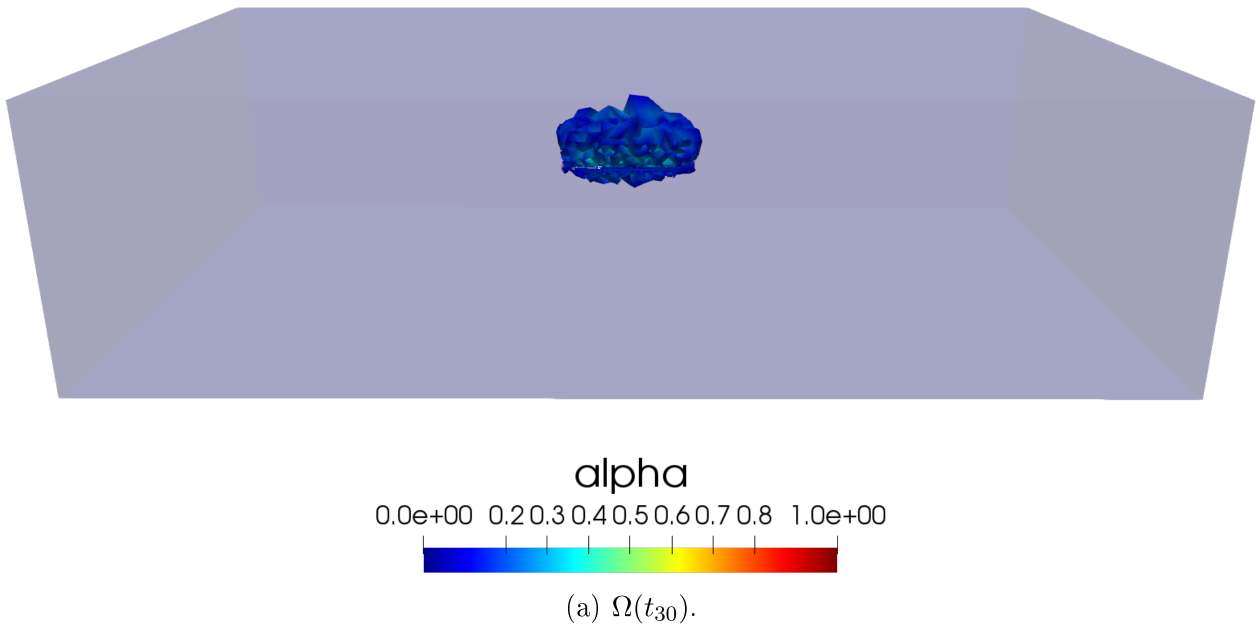


(a) $\Omega(t_0)$.

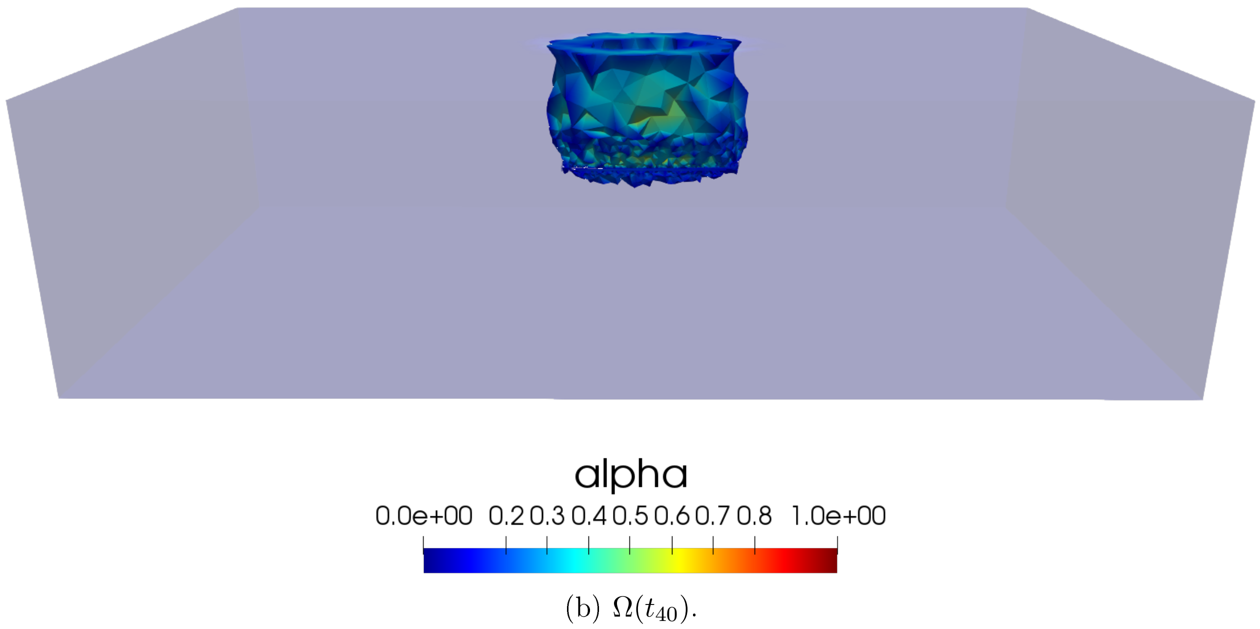


(b) $\Omega(t_{15})$.

Figure 4.51: Damage field distribution in the rock mass for Model 3 in 3D considering $C_L = 0.9$.

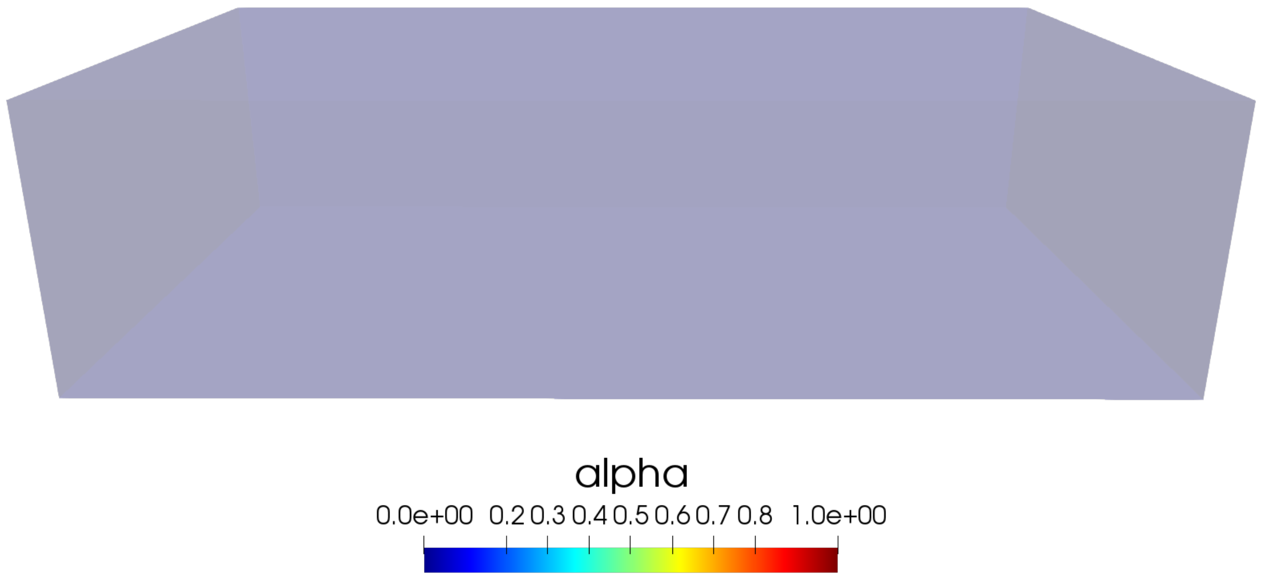


(a) $\Omega(t_{30})$.

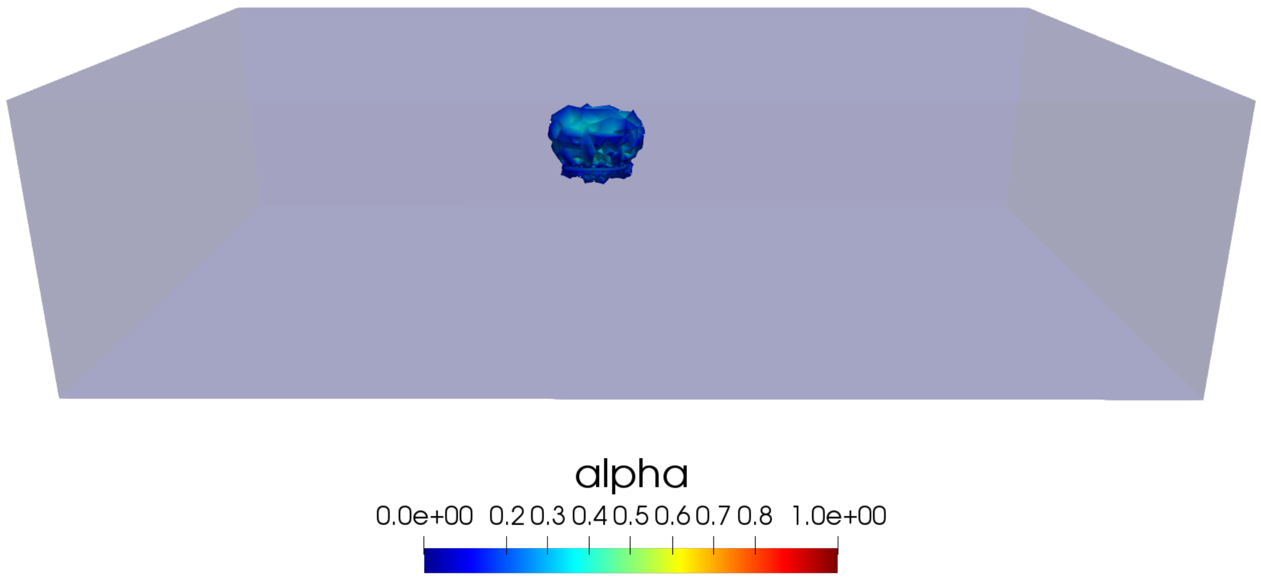


(b) $\Omega(t_{40})$.

Figure 4.52: Damage field distribution in the rock mass for Model 3 in 3D considering $C_L = 0.9$.

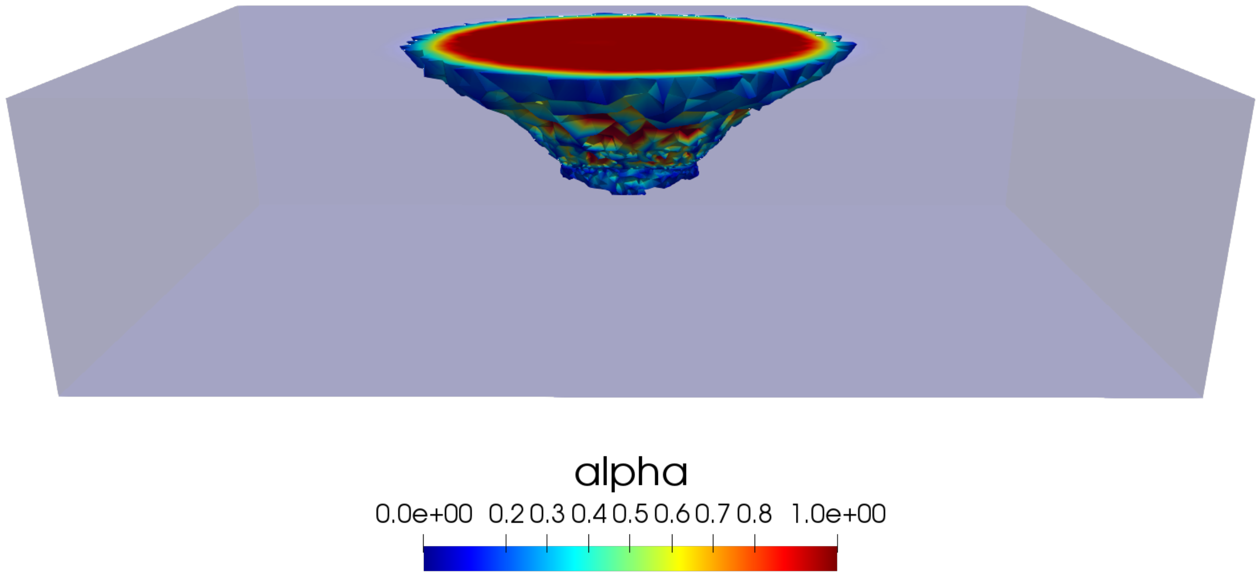


(a) $\Omega(t_0)$.

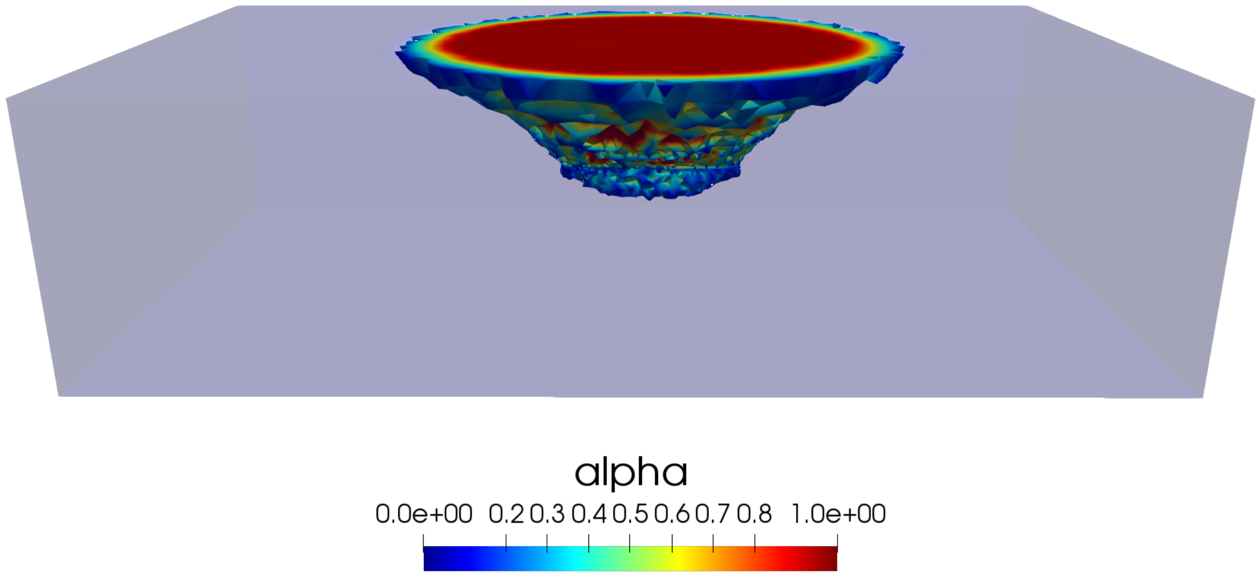


(b) $\Omega(t_{15})$.

Figure 4.53: Damage field distribution in the rock mass for Model 4 in 3D considering $C_L = 0.9$.



(a) $\Omega(t_{30})$.



(b) $\Omega(t_{40})$.

Figure 4.54: Damage field distribution in the rock mass for Model 4 in 3D considering $C_L = 0.9$.

The results seen above show that our proposed algorithm, in these synthetic examples, is able to model the damage produce in the rock mass as the Block Caving process develops and with this we can use this fast algorithm to predict the damage using lower calculation time.

Chapter 5

Hydraulic fracturing in the damage model

In this chapter, we will consider a first view of hydraulic fracturing to underground mining. Hydraulic fracturing has been investigated as an alternative to conventional fracturing methods, where the main motivation is to mitigate the seismic response of the rock mass. We will study if it is possible with our model to be able to replicate the hydraulic fracturing used in underground mining.

5.1 hydraulic fracturing modeling

To model the hydraulic fracturing, we consider that the domain has an initial fracture on the cavity, that is $\alpha_0 \neq 0$. The main objective is to study the effect produced by the initial hydraulic fracturing in our underground mining problem. For this we will analyze the damage caused by the cavity and the elastic and dissipated energies, and compared the results with those obtained without using hydraulic fracturing.

Considering the definition of the total energy $\mathcal{P}(u, \alpha)$ defined in Chapter 1.2 by

$$\mathcal{P}_t(u, \alpha) = \mathcal{E}(u, \alpha) + \mathcal{S}(\alpha) - \mathcal{W}_t(u), \tag{5.1}$$

where

$$\begin{aligned}
\mathcal{E}(u, \alpha) &= \int_\Omega \frac{1}{2} A(\alpha)\varepsilon(u) : \varepsilon(u)\mathrm{d}x, &(5.2)\\
\mathcal{S}(\alpha) &= \int_\Omega \left(w(\alpha) + \frac{1}{2} w_1 \ell^2 \nabla\alpha \cdot \nabla\alpha \right) \mathrm{d}x, &(5.3)\\
\mathcal{W}_t(u) &= \int_\Omega f_t \cdot u\mathrm{d}x + \int_{\partial\Omega_F} F_t \cdot u\mathrm{d}s. &(5.4)
\end{aligned}$$

To model the impact of mining activity, let us consider a homogeneous body $\Omega_0 \subset \mathbb{R}^n$, with $n = 2, 3$, that represents the reference rock mass, and consider subdomains $S_\mathrm{i} \subseteq \Omega_0$,

$\mathrm{i} = 1, \cdots, M$, that represent the interior cavities with $S_{\mathrm{i}} \subseteq S_{\mathrm{i}+1}$, where our domain to consider the block caving process will be $\Omega_{\mathrm{i}} = \Omega_0 \setminus S_{\mathrm{i}}$, with $k = 1, \cdots, M$. These domains represent the cavity advance in mining activity. Thus, we consider, for $k \geq 1$

$$\begin{aligned} u_k &= \operatorname*{arg\,min}_{u \in \mathcal{C}_{t_{\mathrm{i}}}} \mathcal{P}_t(u, \alpha_{k-1}), & (5.5) \\ \alpha_k &= \operatorname*{arg\,min}_{\alpha \geq \alpha_{k-1}} \mathcal{P}_t(u_k, \alpha). & (5.6) \end{aligned}$$

Previously, the solutions to this problem were obtained considering the initial damage $\alpha_0 = 0$, but to study the influence of hydraulic fracturing in underground mining, we consider $\alpha_0 \neq 0$. In this way, the results obtained in each step of cavity advance are associated to this initial damage, where we will denote these solutions by $\{u_k(\alpha_0), \alpha_k(\alpha_0)\}$, $k = 1, \cdots, M$.

As $\alpha_0 \geq 0$, $\alpha_k(\alpha_0) \geq \alpha_k(0)$, for $k \geq 1$. Therefore it is not evident that the dissipated energy associated with the damage can be compared with the case of hydraulic fracturing of the one that does not have it, this is because the solutions in each cavity advance come from different initial damages.

On the other hadn, if we consider the elastic energy with $A(\alpha) = (1-\alpha)^2 A_0$

$$\mathcal{E}(u_k, \alpha_k) = \int_\Omega \frac{1}{2}(1-\alpha_k)^2 A_0 \varepsilon(u_k) : \varepsilon(u_k) \mathrm{d}x, \tag{5.7}$$

and taking $\alpha_0 = 1 - \lambda$, with $0 \leq \lambda \leq 1$ constant, we have

$$\mathcal{E}(u_k(\alpha_0), \alpha_0) = \frac{1}{\lambda^2} \mathcal{E}(u_k(0), 0) > \mathcal{E}(u_k(0), 0). \tag{5.8}$$

In this way, the elastic energy associated with hydraulic fracturing will always be greater than the elastic energy without considering hydraulic fracturing. This leads us to find a better way to compare the associated energies. So, we define the following sequences

$$\begin{aligned} \Lambda_1(\alpha_0) &:= \{\mathcal{E}(u_k(\alpha_0), \alpha_k(\alpha_0)) - \mathcal{E}(u_{k-1}(\alpha_0), \alpha_{k-1}(\alpha_0))\}_{k=1}^M, & (5.9) \\ \Lambda_2(\alpha_0) &:= \{\mathcal{S}(\alpha_l(\alpha_0)) - \mathcal{S}(\alpha_{k-1}(\alpha_0))\}_{k=1}^M, & (5.10) \end{aligned}$$

which represent the change in the elastic and dissipated energies respectively by the mining process. In this way we would be interested in comparing $\Lambda_{\mathrm{i}}(0)$ with $\Lambda_{\mathrm{i}}(\alpha_0)$, $\mathrm{i} = 1, 2$.

5.2 Numerical results

We present some numerical results for the analysis of the hydraulic fracturing effect considering the cavity evolution in a synthetic example that models the rock mass in underground mining. Figure 5.1 displays the lateral cut of the rock mass considering the hydraulic fracturing.

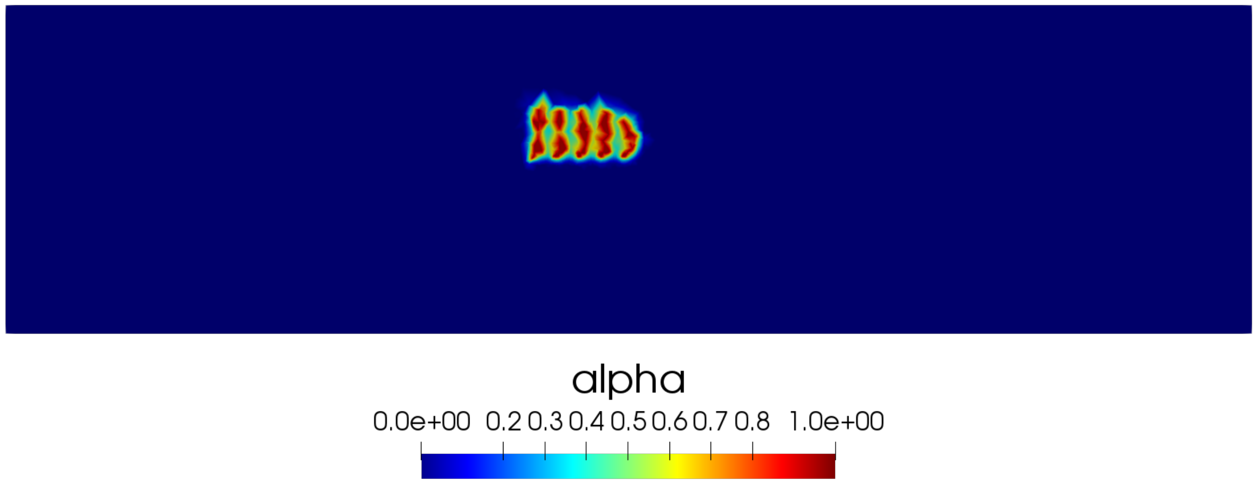
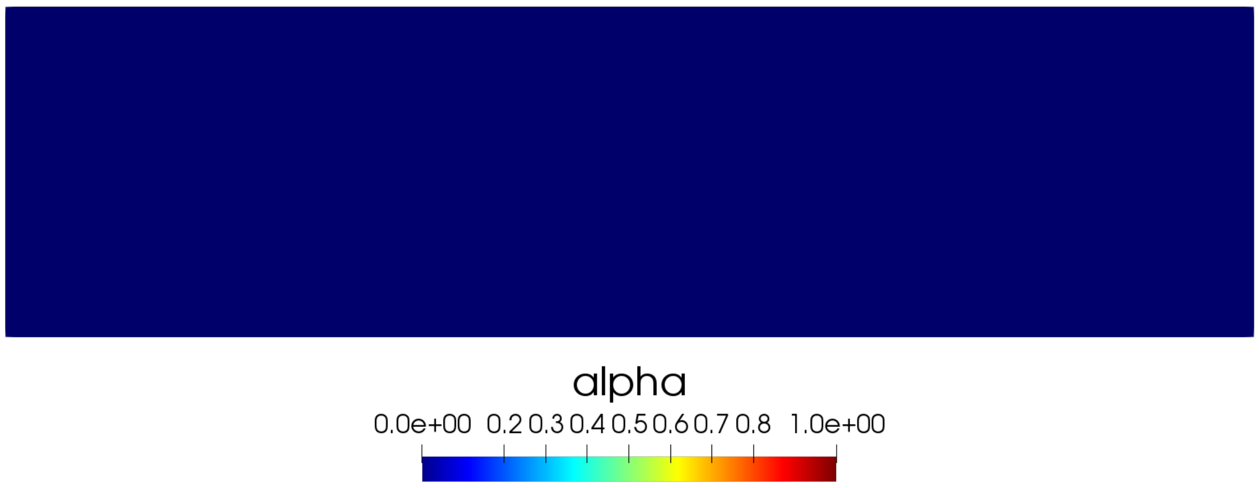
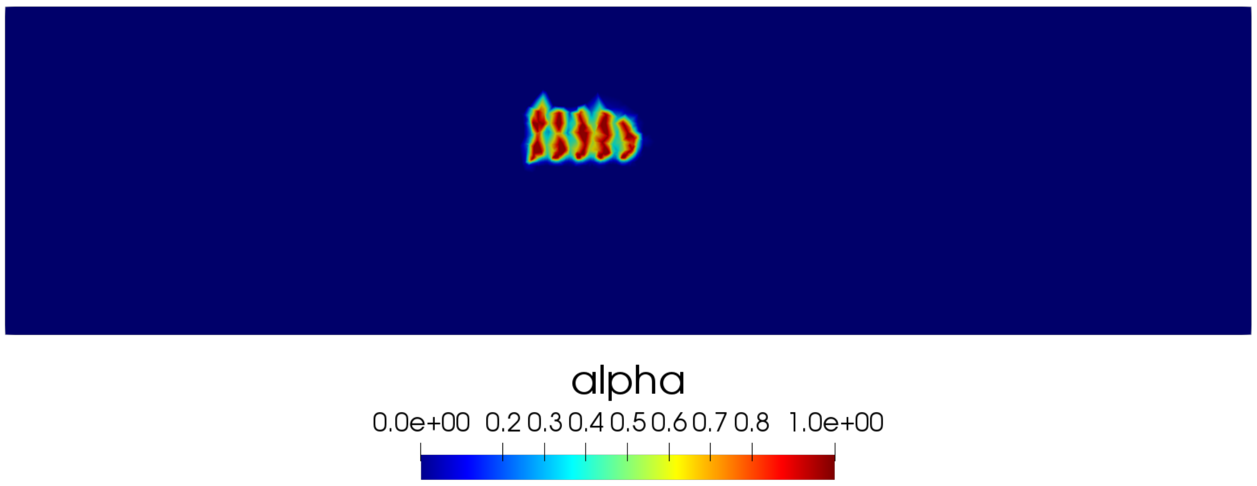


Figure 5.1: Damage distribution in initial time.

Figures 5.2-5.13 display a comparison between the block caving process without hydraulic fracturing and the block caving process considering this. It is possible to see that the damage distribution does not vary perceptibly when considering hydraulic fracturing, so we must see what happens with other quantities when hydraulic fracturing is taken into account.

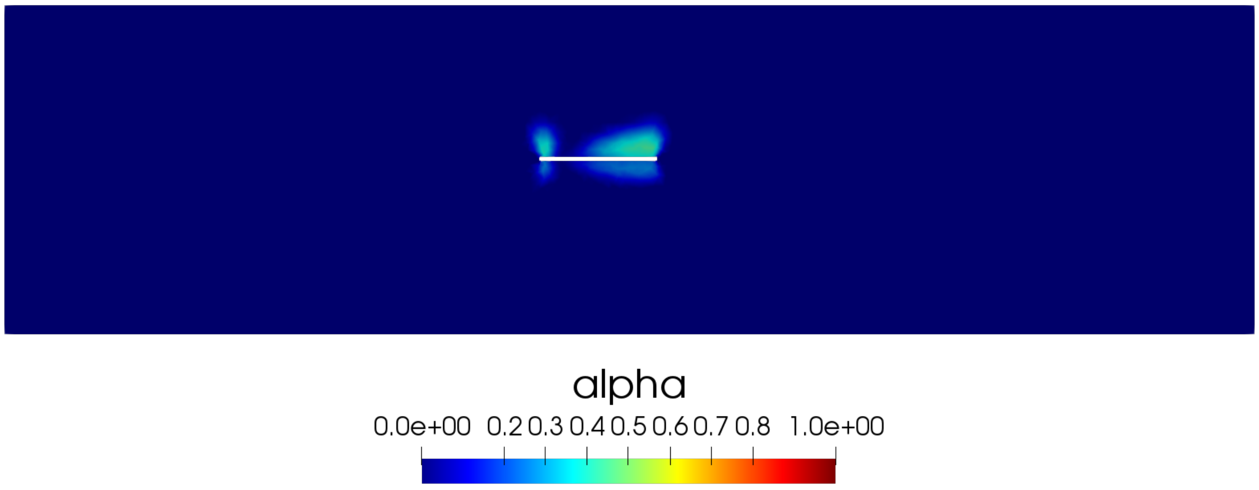


(a) Domain without initial damage in $\Omega(t_0)$.

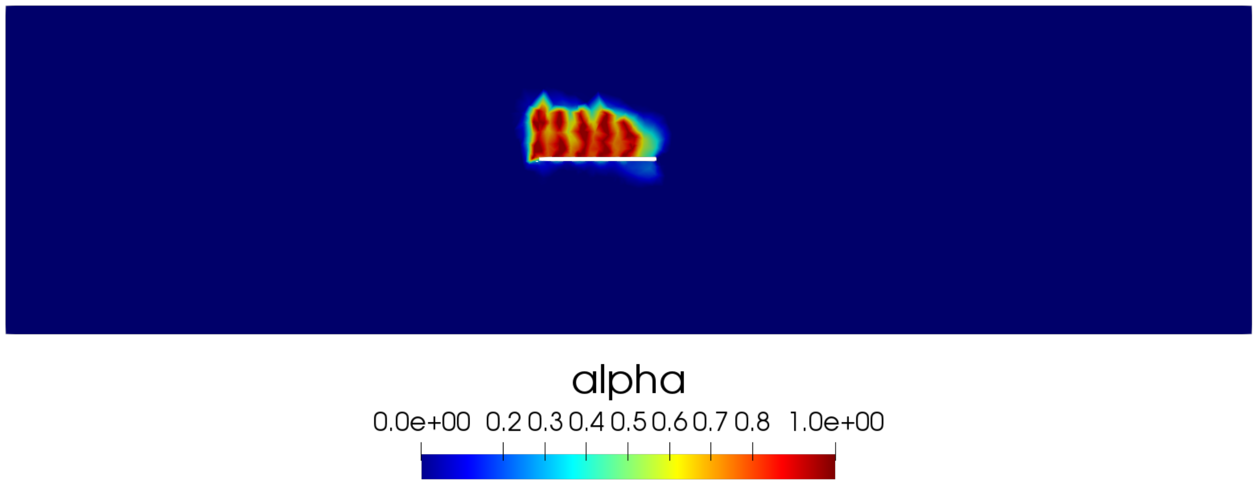


(b) Domain with initial damage in $\Omega(t_0)$.

Figure 5.2: Lateral cut of the mesh for different time steps for the Model 1.

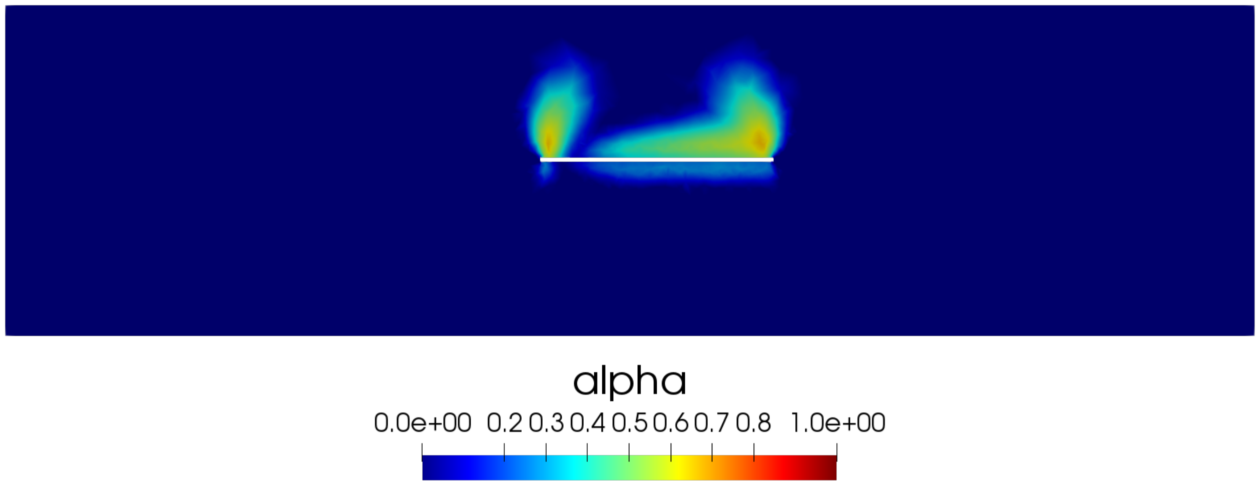


(a) Domain without initial damage in $\Omega(t_20)$.

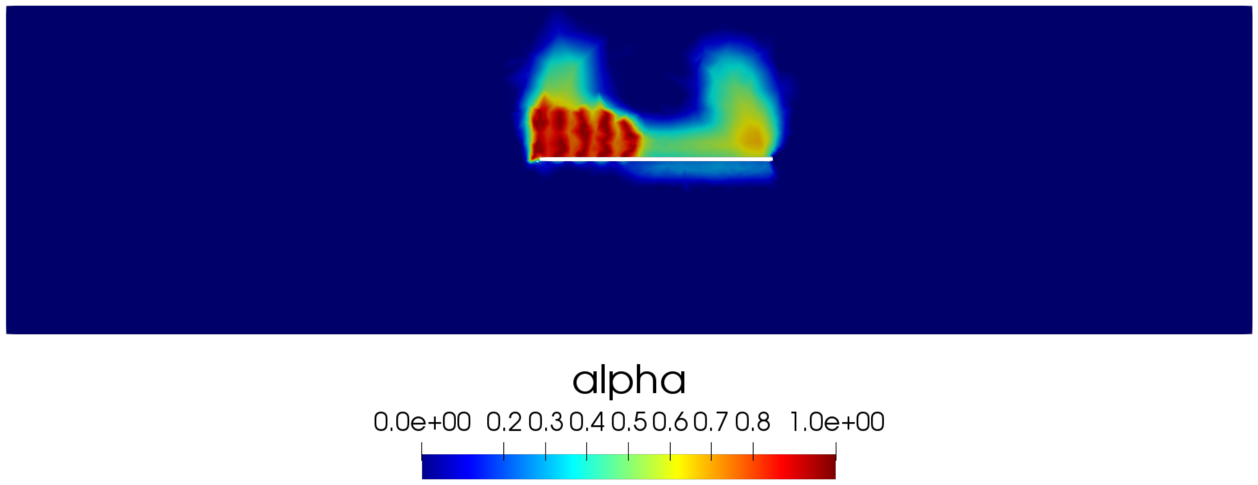


(b) Domain with initial damage in $\Omega(t_20)$.

Figure 5.3: Lateral cut of the mesh for different time steps for the Model 1.

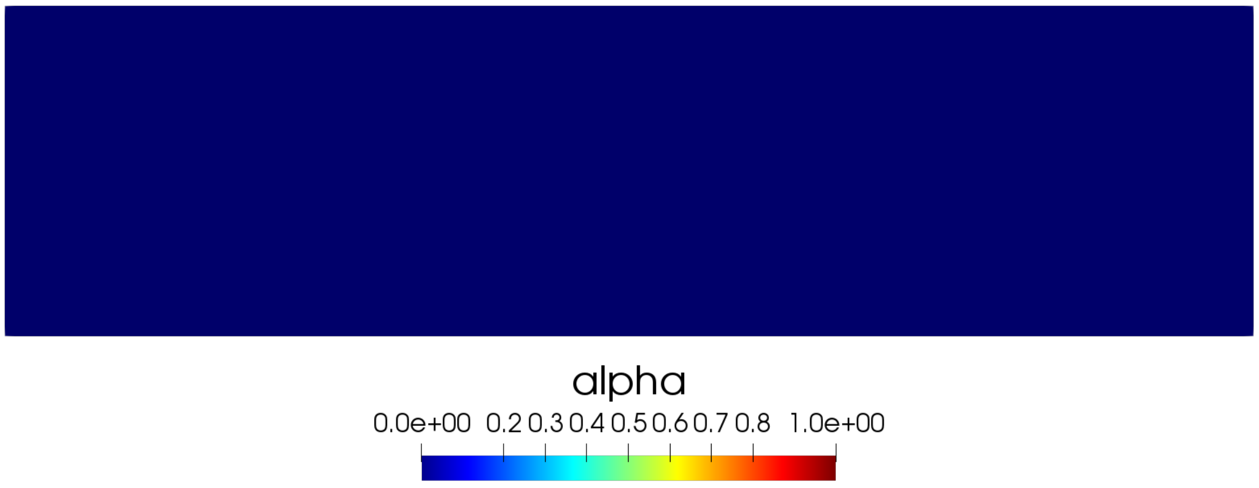


(a) Domain without initial damage in $\Omega(t_40)$.

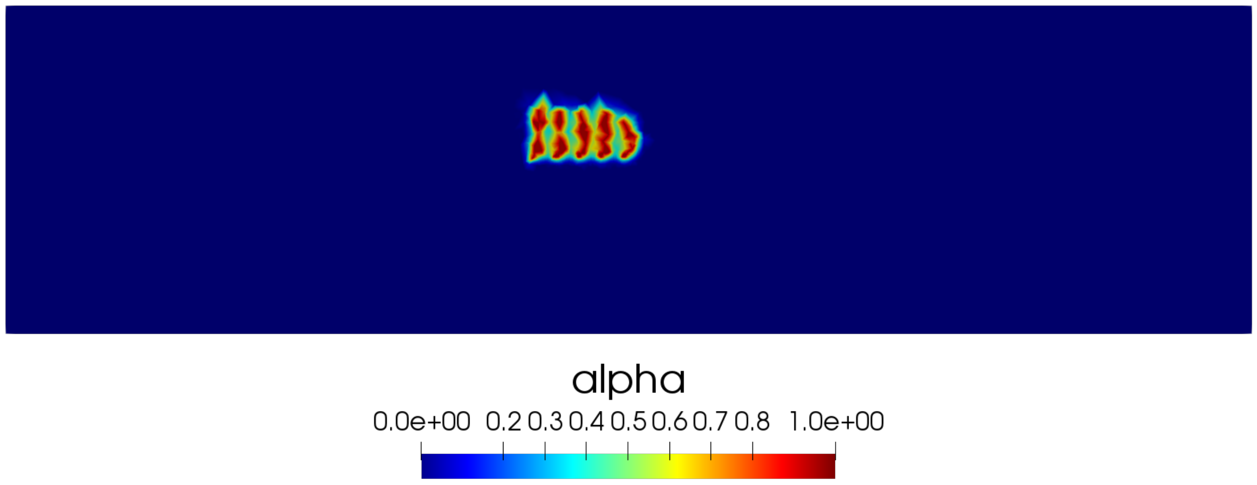


(b) Domain with initial damage in $\Omega(t_40)$.

Figure 5.4: Lateral cut of the mesh for different time steps for the Model 1.

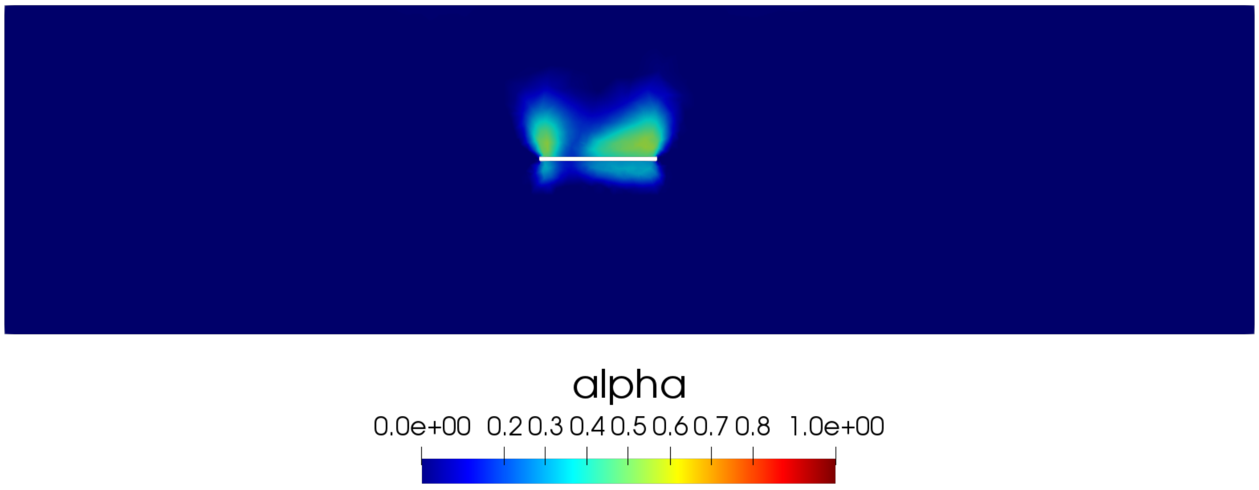


(a) Domain without initial damage.

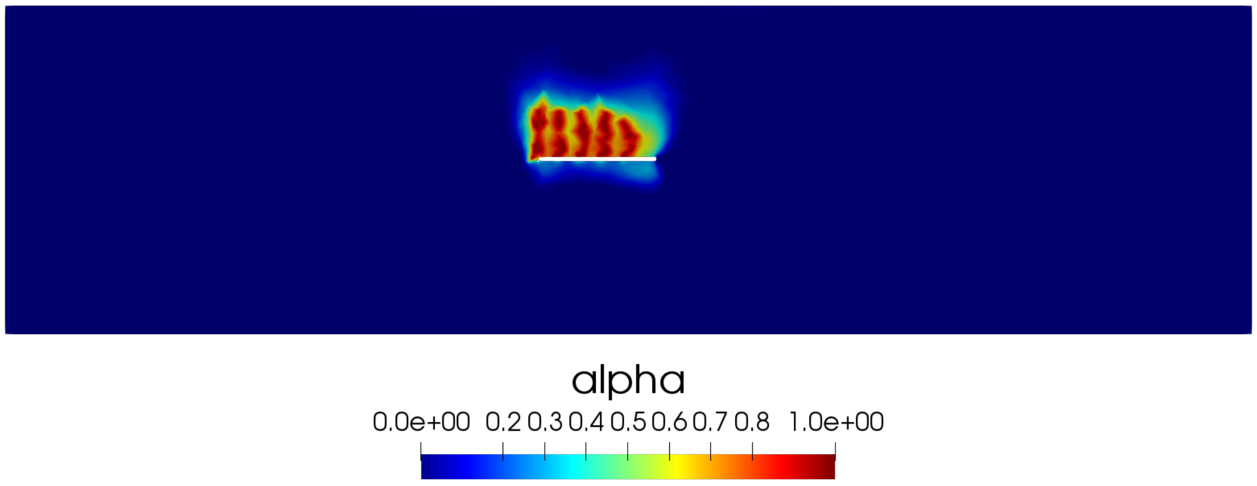


(b) Domain with initial damage.

Figure 5.5: Lateral cut of the mesh for different time steps for the Model 2.

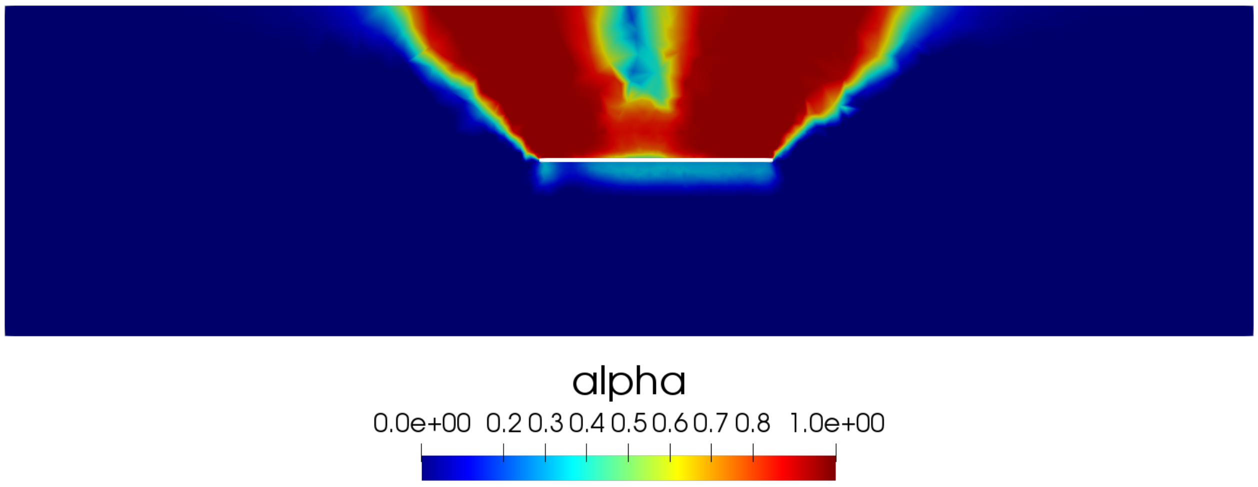


(a) Domain without initial damage.

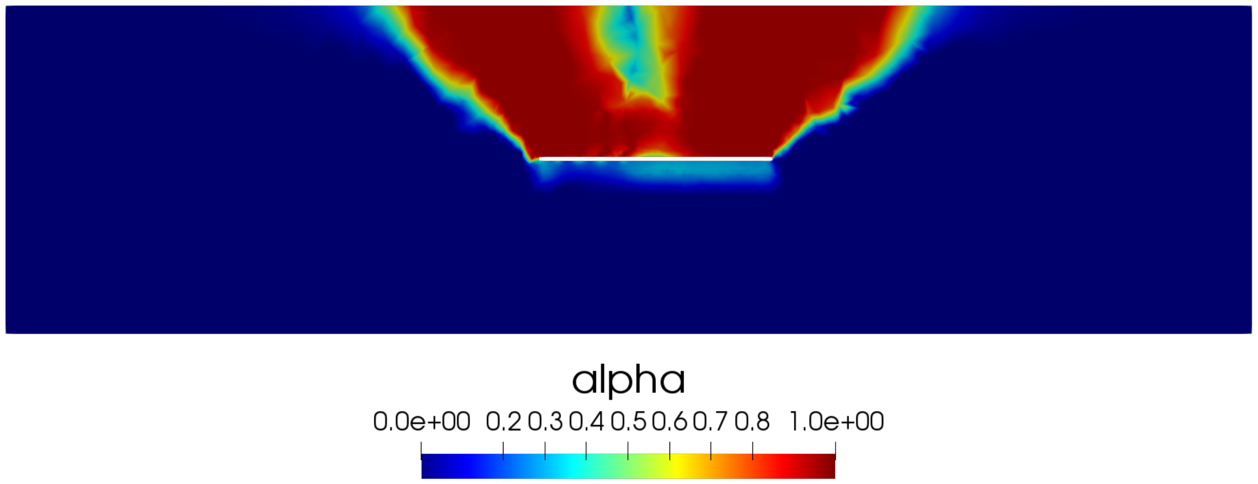


(b) Domain with initial damage.

Figure 5.6: Lateral cut of the mesh for different time steps for the Model 2.

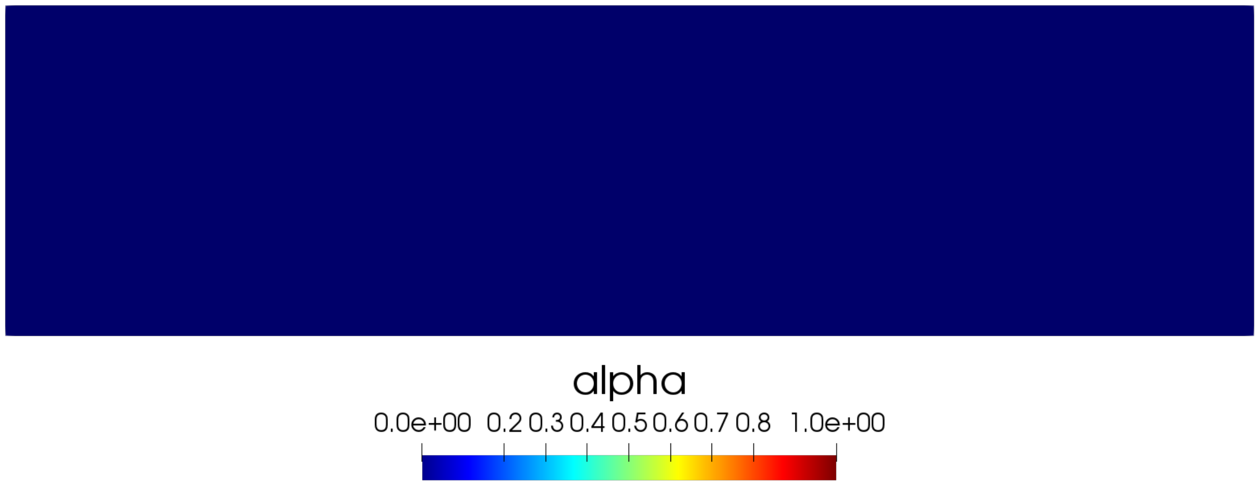


(a) Domain without initial damage.

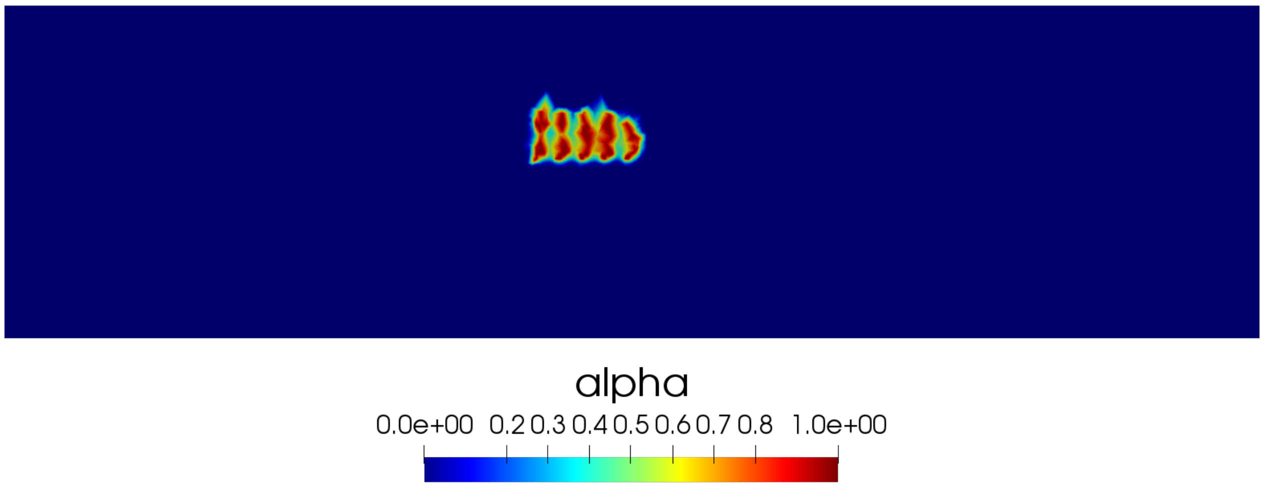


(b) Domain with initial damage.

Figure 5.7: Lateral cut of the mesh for different time steps for the Model 2.

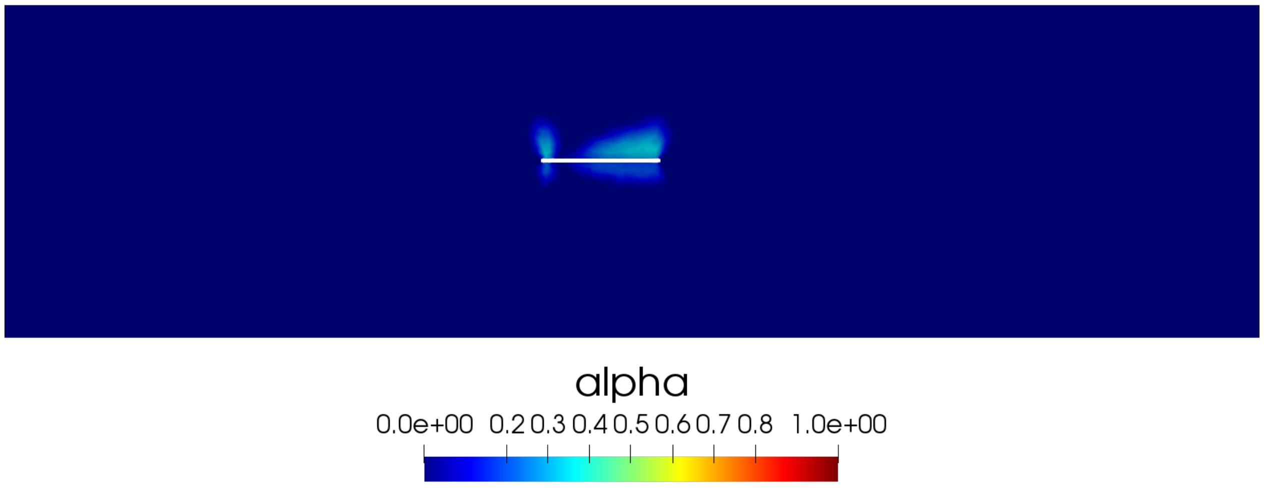


(a) Domain without initial damage.

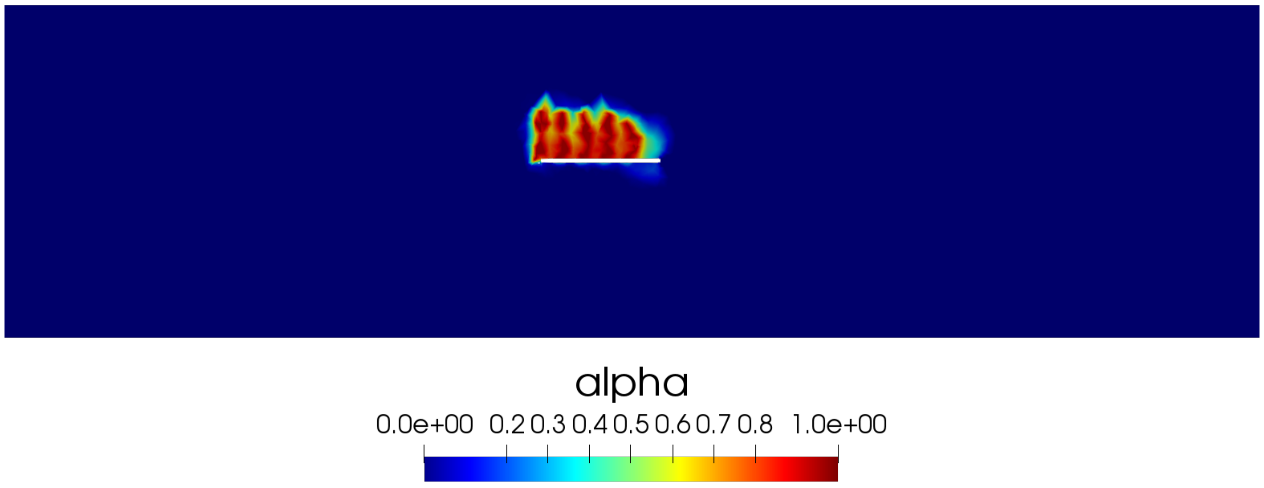


(b) Domain with initial damage.

Figure 5.8: Lateral cut of the mesh for different time steps for the Model 3.

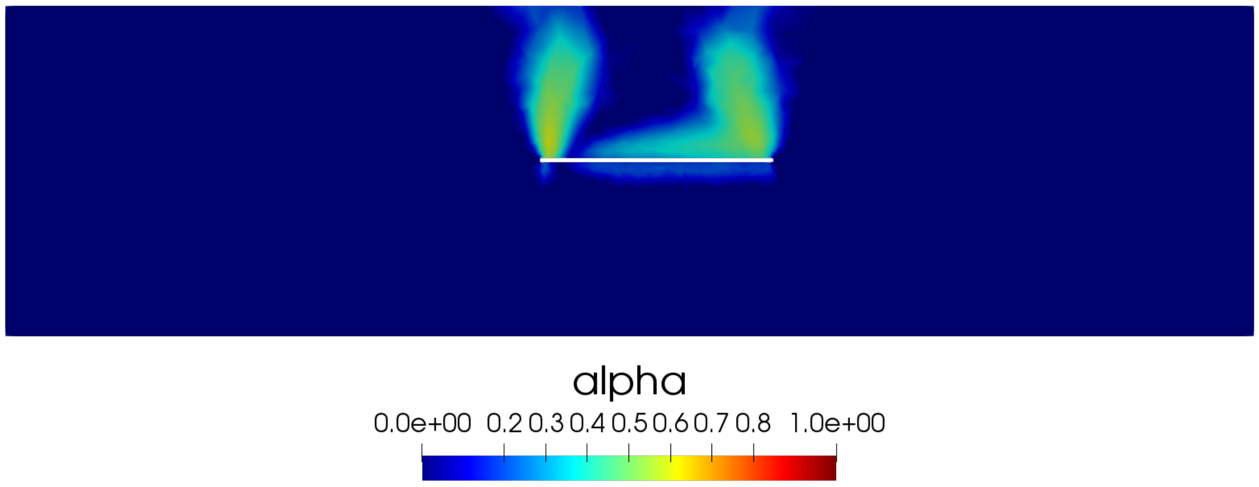


(a) Domain without initial damage.

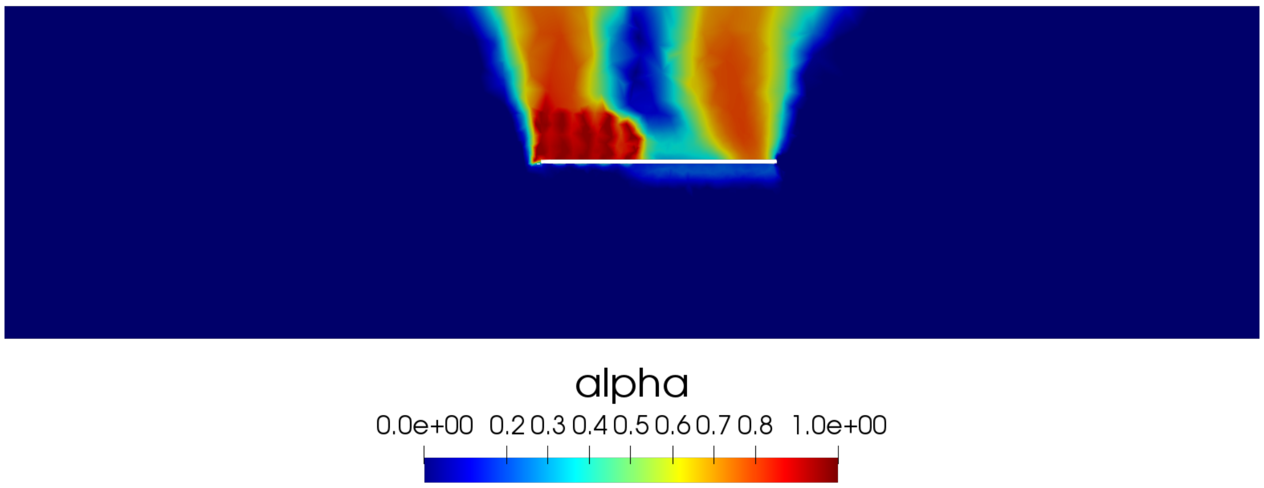


(b) Domain with initial damage.

Figure 5.9: Lateral cut of the mesh for different time steps for the Model 3.

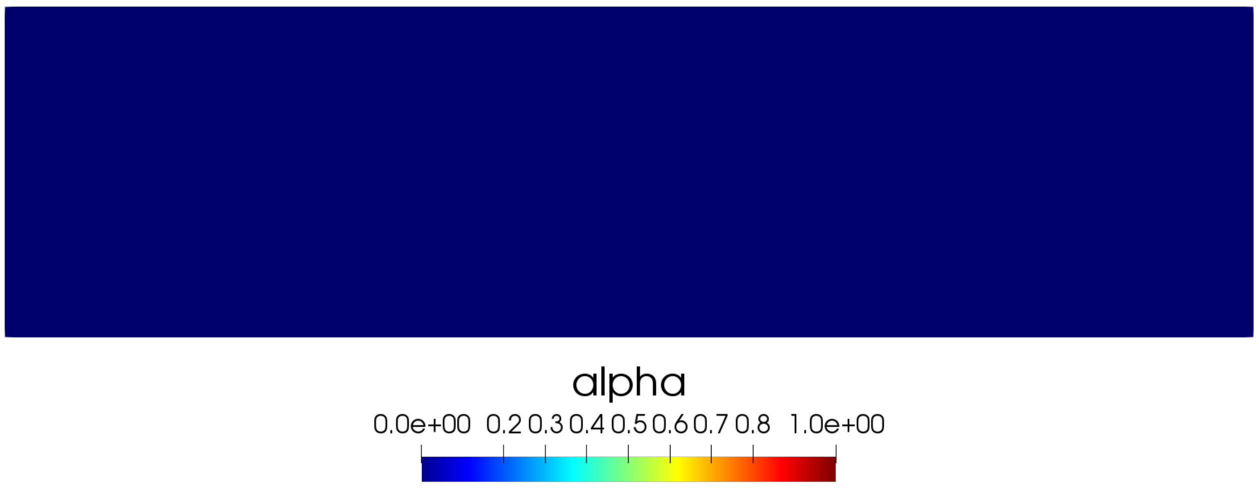


(a) Domain without initial damage.

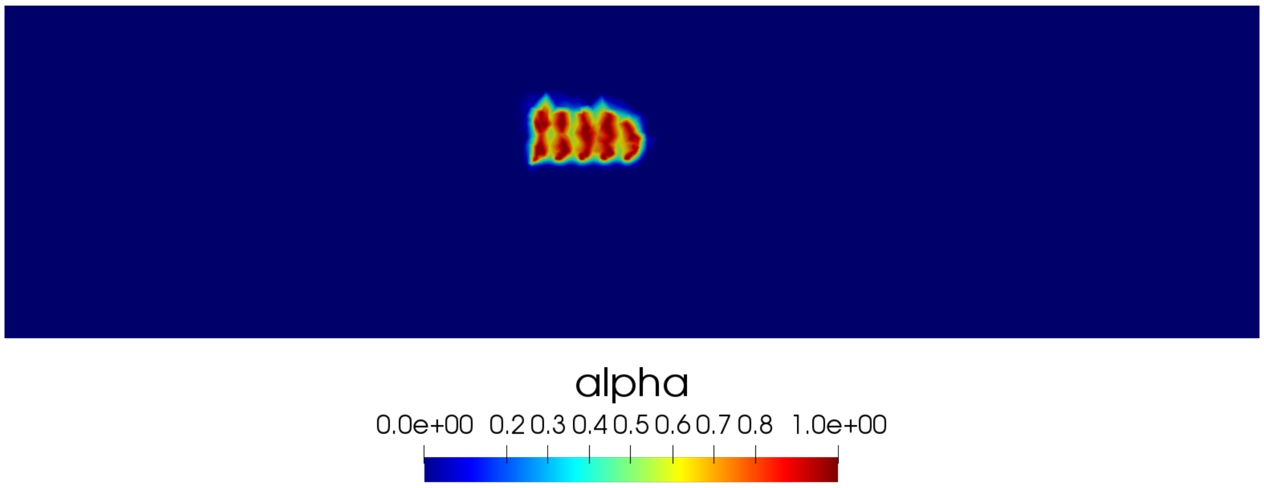


(b) Domain with initial damage.

Figure 5.10: Lateral cut of the mesh for different time steps for the Model 3.

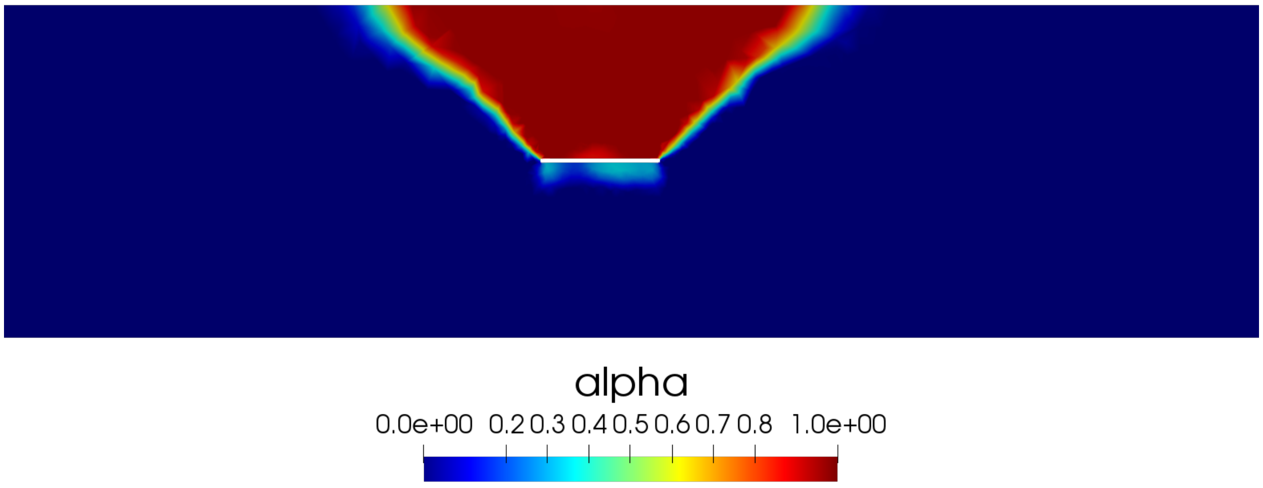


(a) Domain without initial damage.

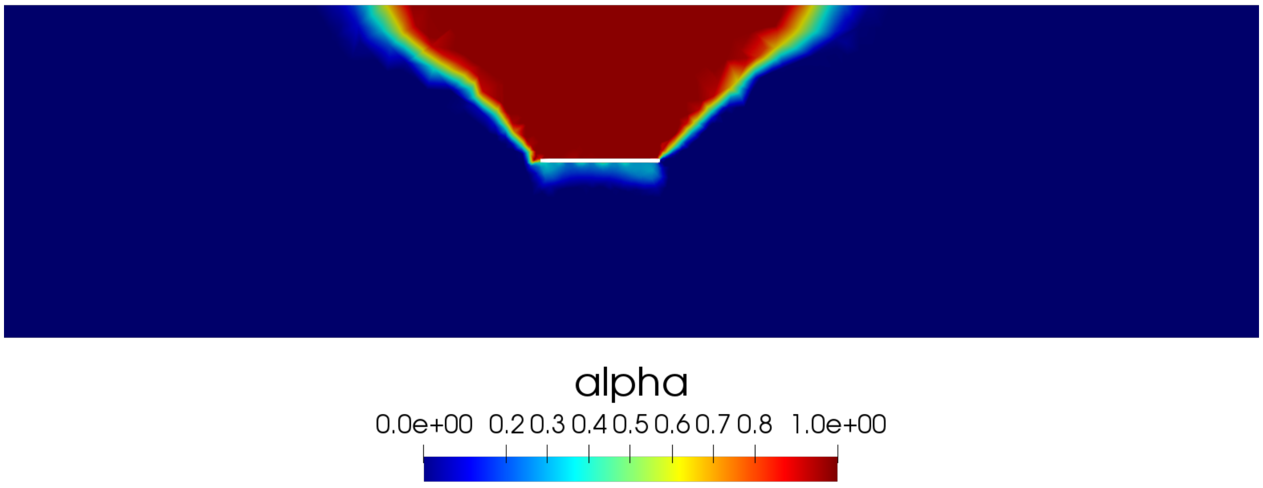


(b) Domain with initial damage.

Figure 5.11: Lateral cut of the mesh for different time steps for the Model 4.

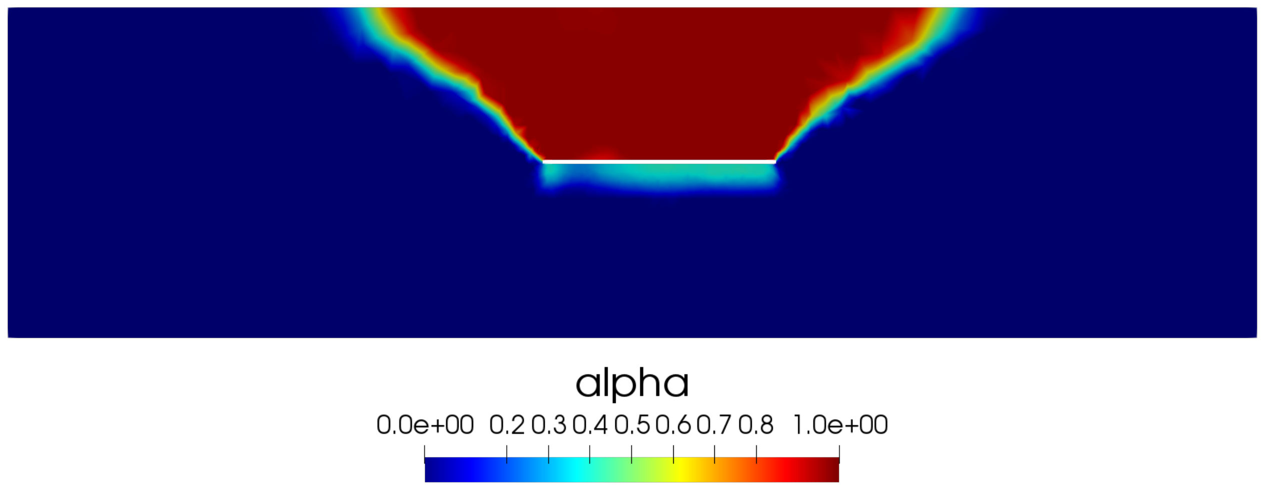


(a) Domain without initial damage.

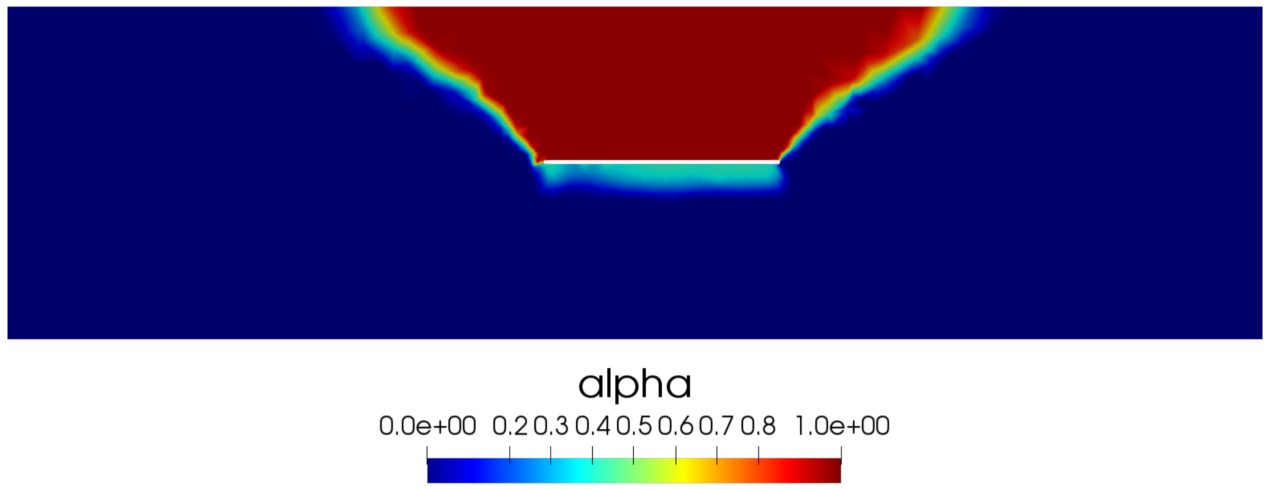


(b) Domain with initial damage.

Figure 5.12: Lateral cut of the mesh for different time steps for the Model 4.



(a) Domain without initial damage.



(b) Domain with initial damage.

Figure 5.13: Lateral cut of the mesh for different time steps for the Model 4.

Considering now the volume of the dissipated energy generated by the damage. Figures 5.14 -5.17 show the comparison of the volume of the dissipated energy in a log-log scale for the final cavity advance (that is, $\Omega(t_{40})$), where it is possible to see that the values are not affected mainly by hydraulic fracturing, obtaining similar results.

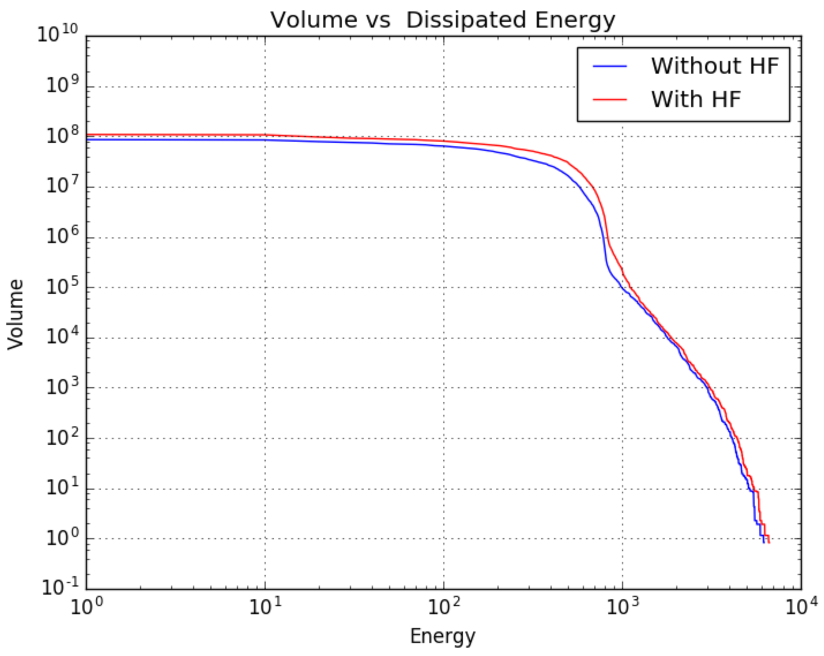


Figure 5.14: Comparison of the dissipated energy in the final cavity advance for the Model 1.

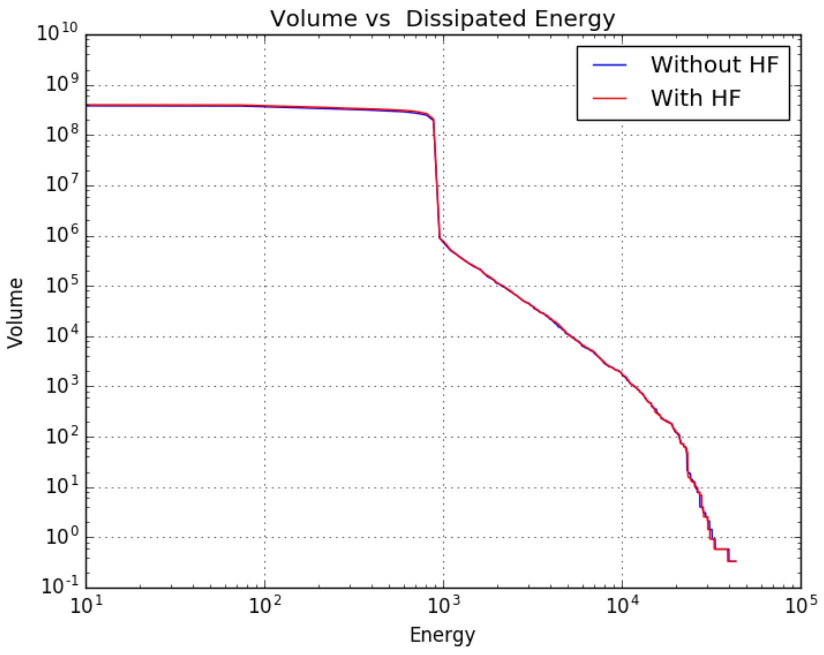


Figure 5.15: Comparison of the dissipated energy in the final cavity advance for the Model 2.

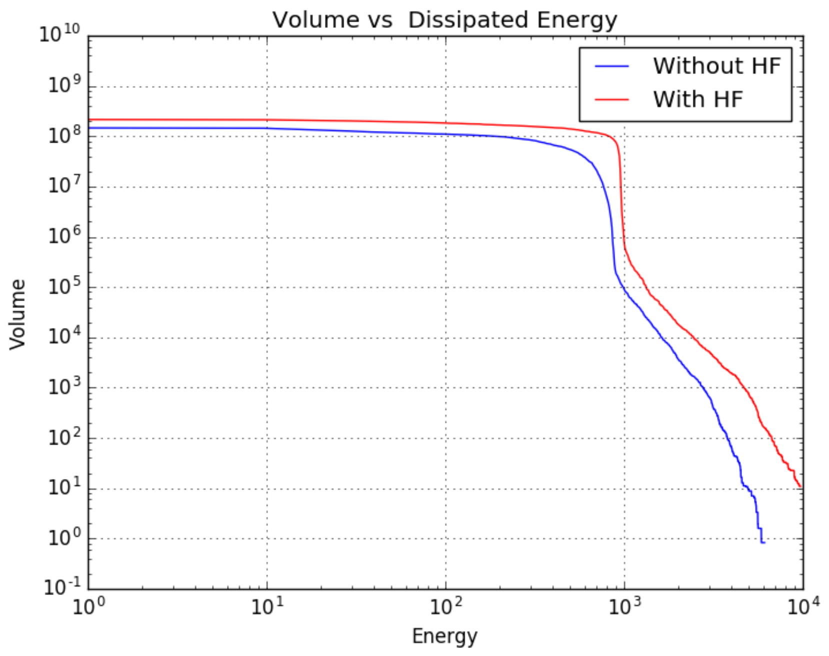


Figure 5.16: Comparison of the dissipated energy in the final cavity advance for the Model 3.

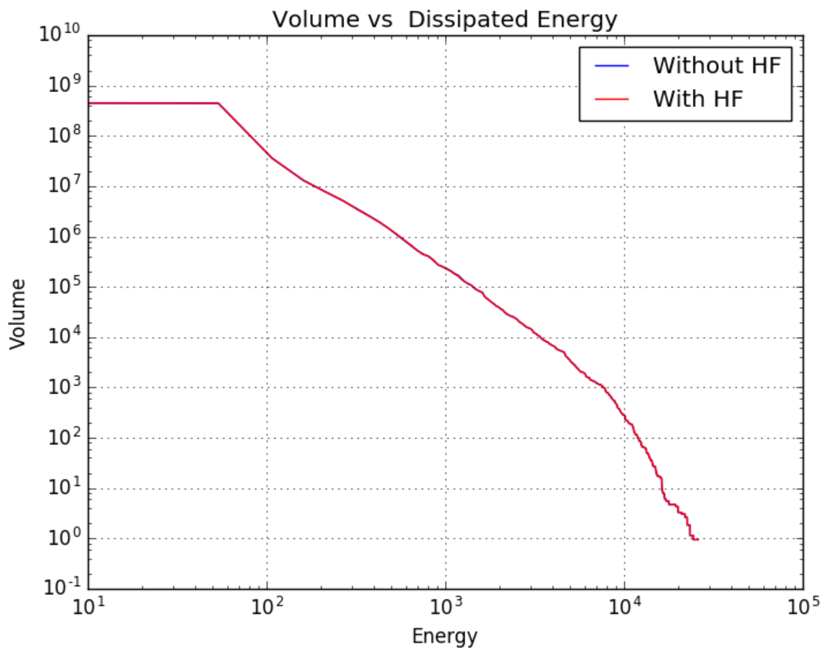


Figure 5.17: Comparison of the dissipated energy in the final cavity advance for the Model 4.

Figures 5.18-5.21 display the comparison of the volume of the elastic energy in a log-log scale for the final cavity advance where, it is possible to see, unlike previous results, that the values for Model 1 and Model 3 have a more significant difference, while for Model 2 and Model 4, the results are practically the same.

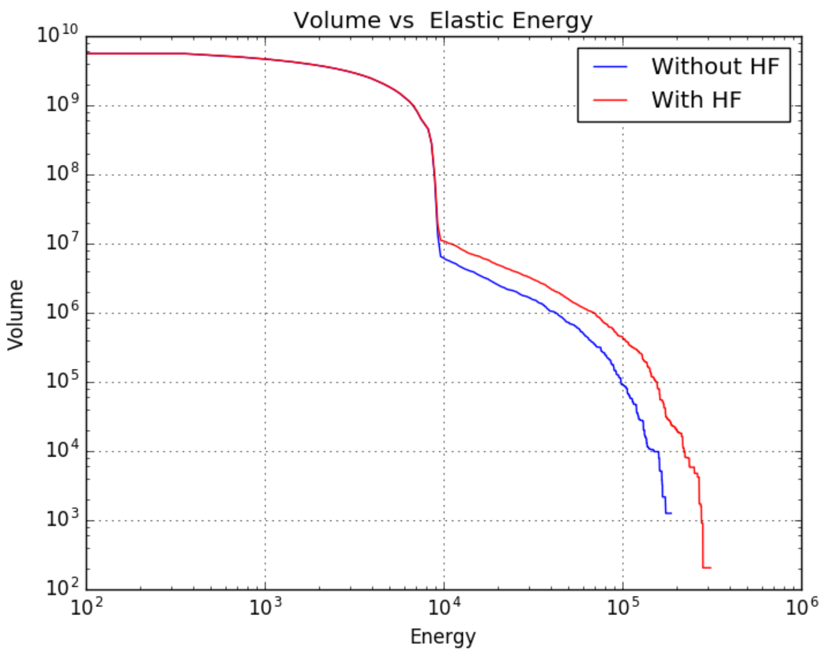


Figure 5.18: Comparison of the elastic energy in the final cavity advance for the Model 1.

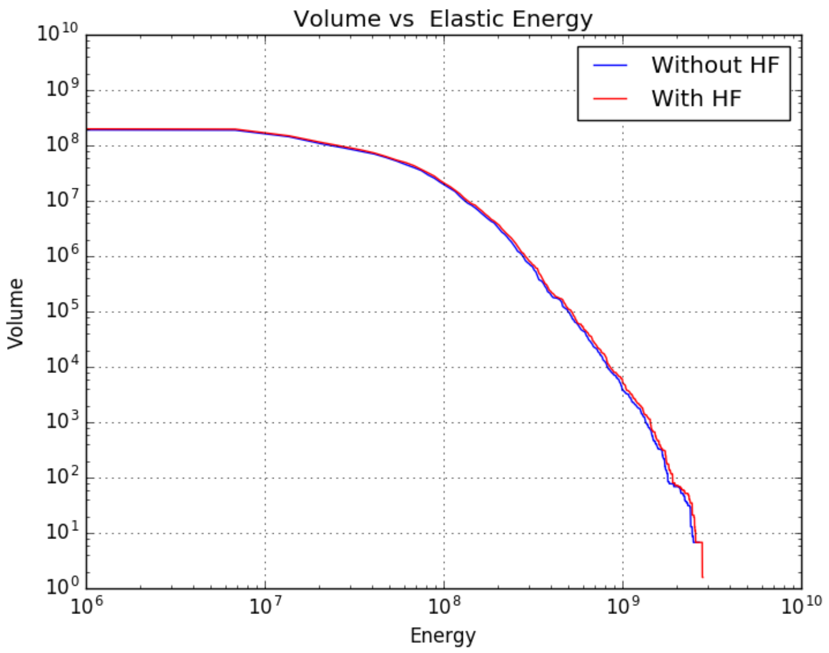


Figure 5.19: Comparison of the elastic energy in the final cavity advance for the Model 2.

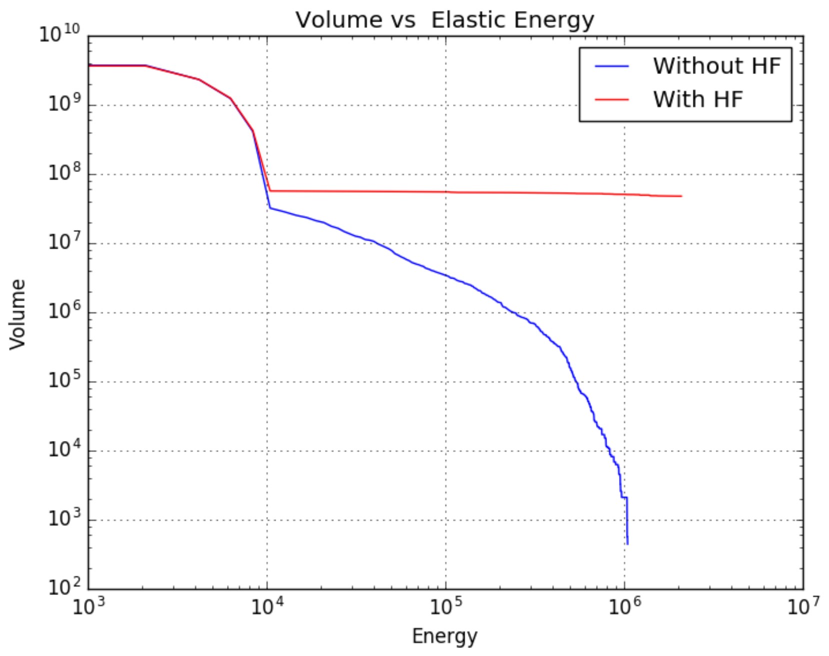


Figure 5.20: Comparison of the elastic energy in the final cavity advance for the Model 3.

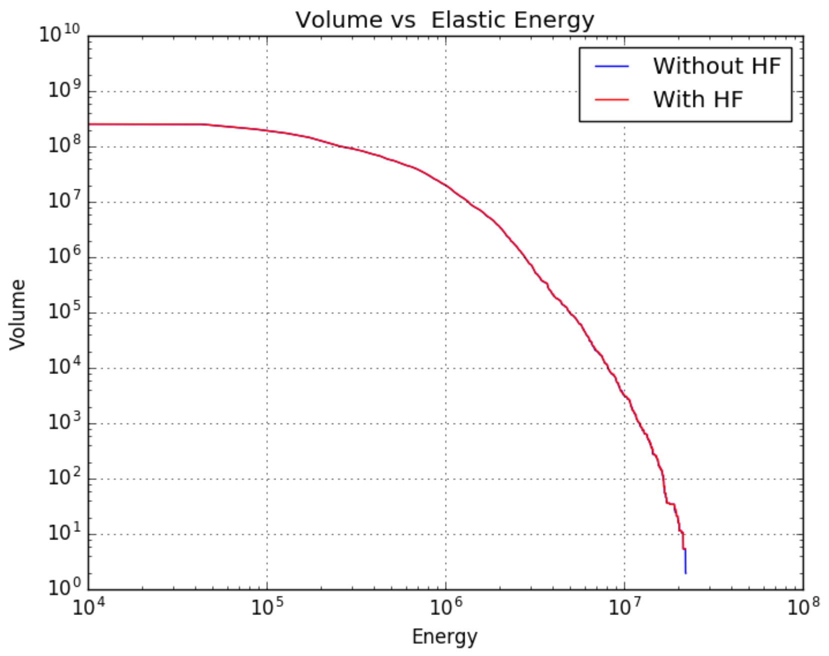


Figure 5.21: Comparison of the elastic energy in the final cavity advance for the Model 4.

Finally, we consider the difference between of the dissipated energy with the previous step, that is $\Lambda_2(\alpha_0)$, for $\alpha_0 = 0$ and $\alpha_0 \neq$. Figures 5.22-5.25 display a comparison between the of $\Lambda_2(\alpha_0)$ without hydraulic fracturing and the block caving process considering hydraulic fracturing. It is possible to see that for the Models 1,2 and 3 there is a difference between the value considering hydraulic fracturing or not. On the other hand, for the Model 4 there is no difference between the two cases.

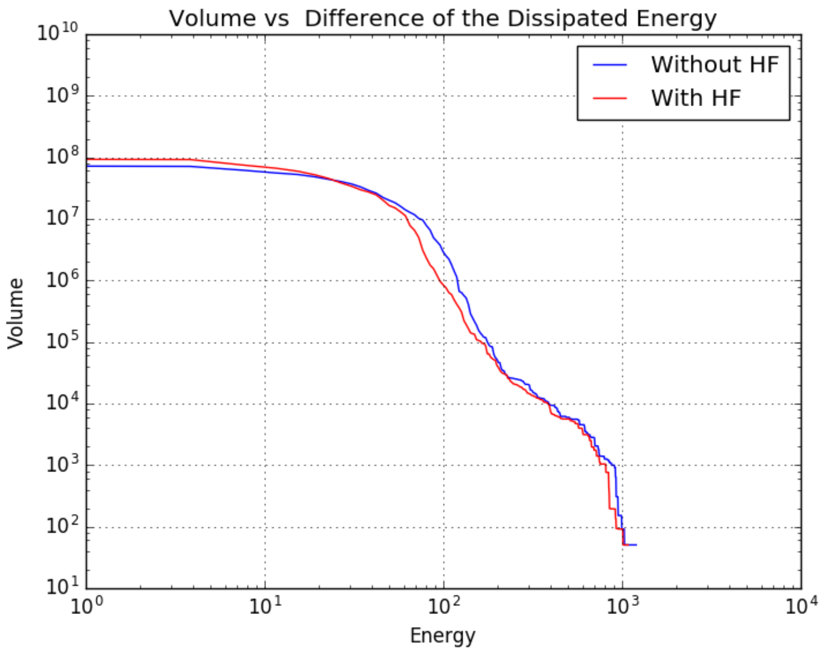


Figure 5.22: Comparison of the elastic energy in the final cavity advance for the Model 1.

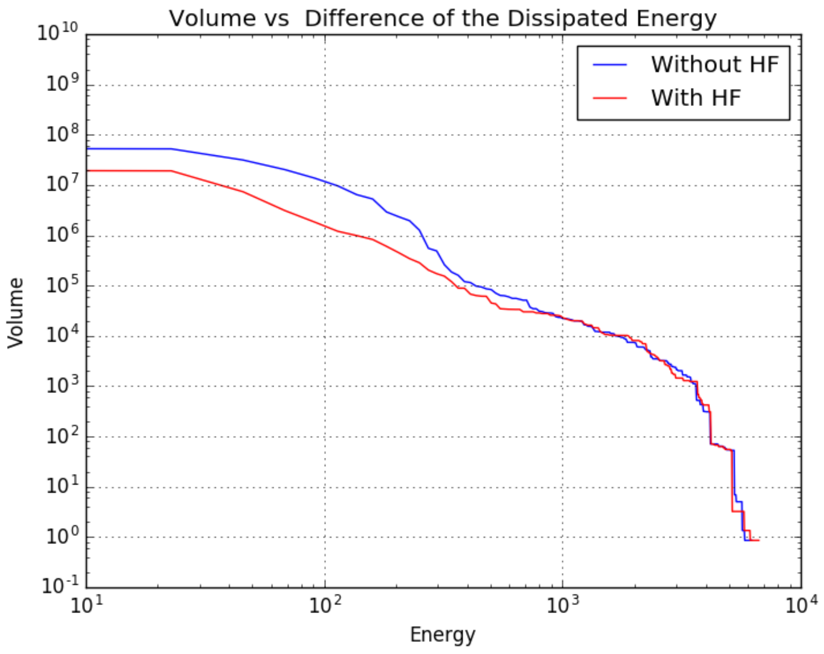


Figure 5.23: Comparison of the elastic energy in the final cavity advance for the Model 2.

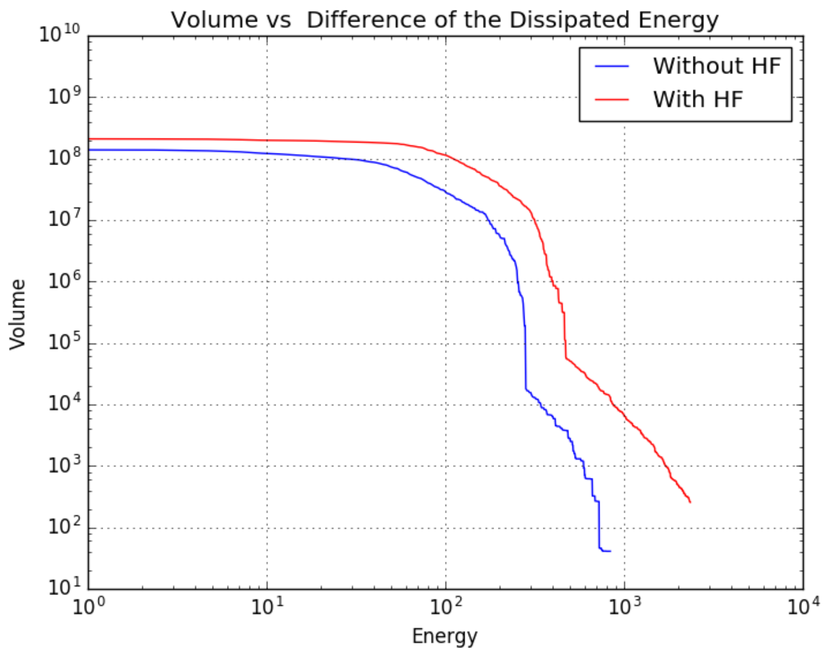


Figure 5.24: Comparison of the elastic energy in the final cavity advance for the Model 3.

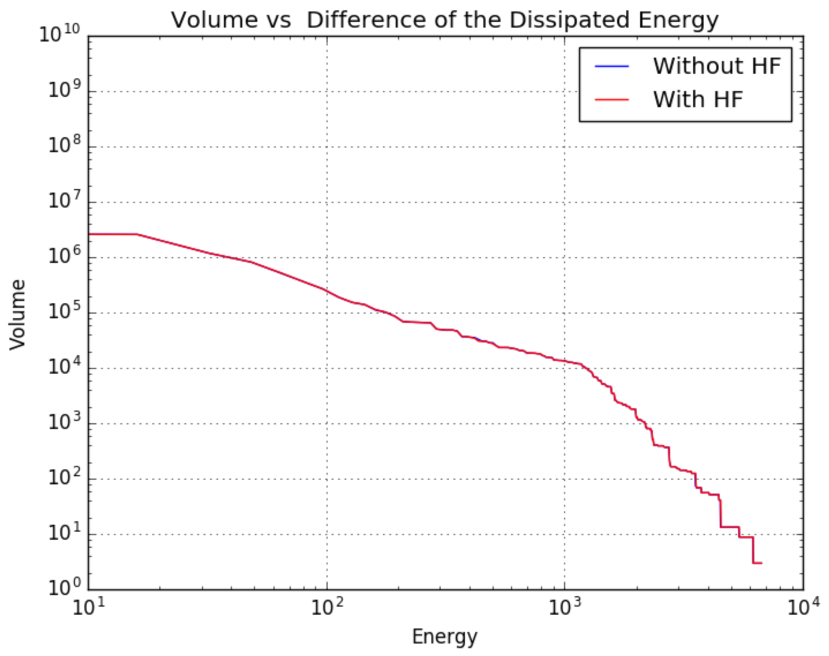


Figure 5.25: Comparison of the elastic energy in the final cavity advance for the Model 4.

In the test shown above, considering for underground mining that the hydraulic fracturing is characterized by an initial damage and comparing it with the case without damage, it is possible to see that the hydraulic fracturing does not represent a significant effect in the Block Caving process and does not affect the damage generation process, on the other hand, it can also be observed that the energies dissipated also do not undergo changes when considering $\alpha_0 \neq 0$. It should be noted that the real case of hydraulic fracturing requires additional work to implement it, this due to the size of the initial fractures must be of a considerably small size in comparison to the considered domain. Finally, a future study will be a better way to consider hydraulic fracturing and how to implement it numerically in our model.

Chapter 6

The damage model in a real mine

In this chapter, we consider our damage model and the fast algorithm proposed in the Chapters 3-4 respectively for a realistic aplication, that is, we consider a domain that represent the real topography of the El Teniente mine.

6.1 El Teniente mine

In this section we consider the previous results and apply in a mesh that corresponds to the real topography of the El Teniente mine (see Figure 6.1), where the meshes are defined following the Appendix A. For this test, we consider the Model 1 and $C_L = 0.9$, for avoiding errors oscillations, and we will show the behavior of these models for this domain, in addition to the mesh step, we will consider the real progress of the mine through the polylines of the cavity, data delivered by the El Teniente Division. This type of more complex topography brings numerical difficulties and increased calculation times, so it is necessary to consider distributed algorithms in order to overcome them.

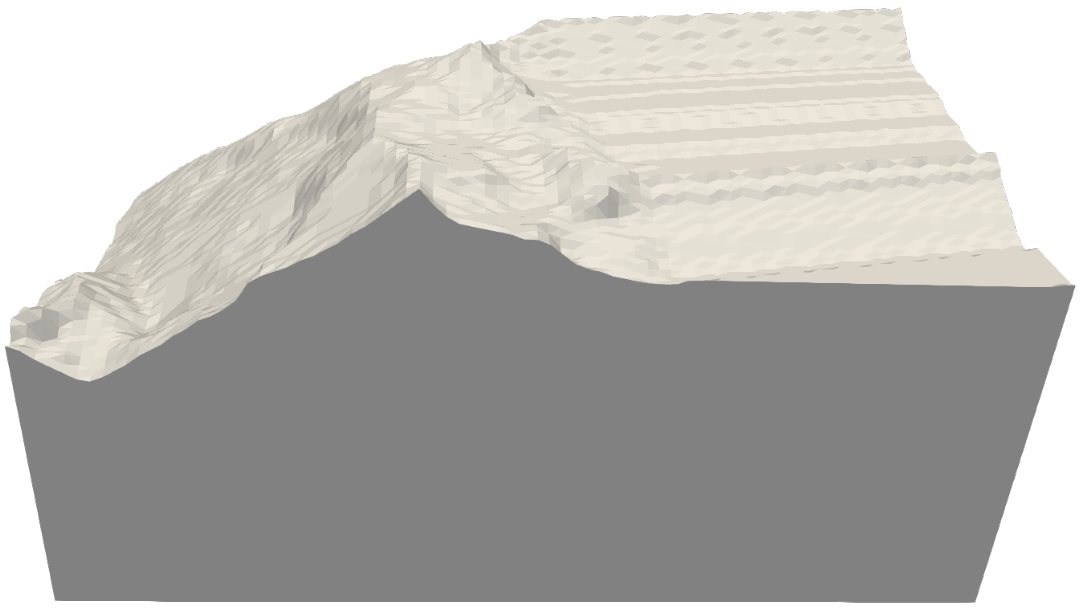


Figure 6.1: 3D view of the topography that represent El Teniente mine.

6.2 Numerical results

In this first test, we consider the Esmeralda sector in El Teniente mine. For this we will define the cavity advances as shown in Figure 6.2, where it can be observed that these advances are not regular as in the previous tests.

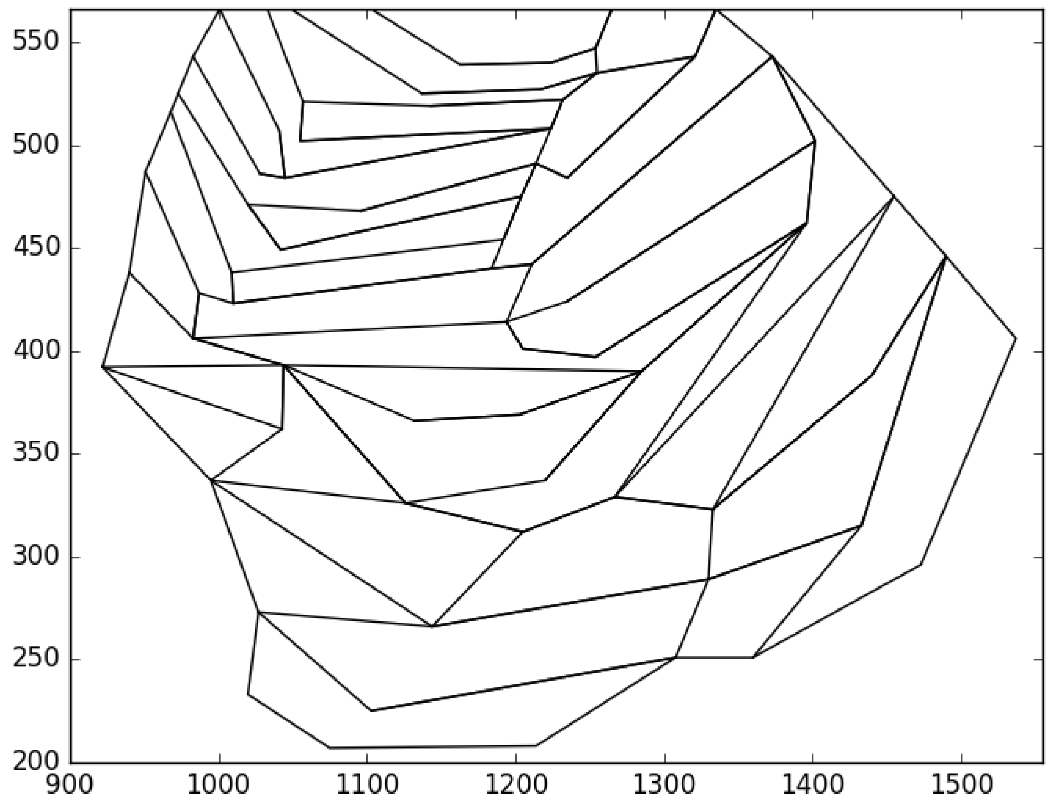
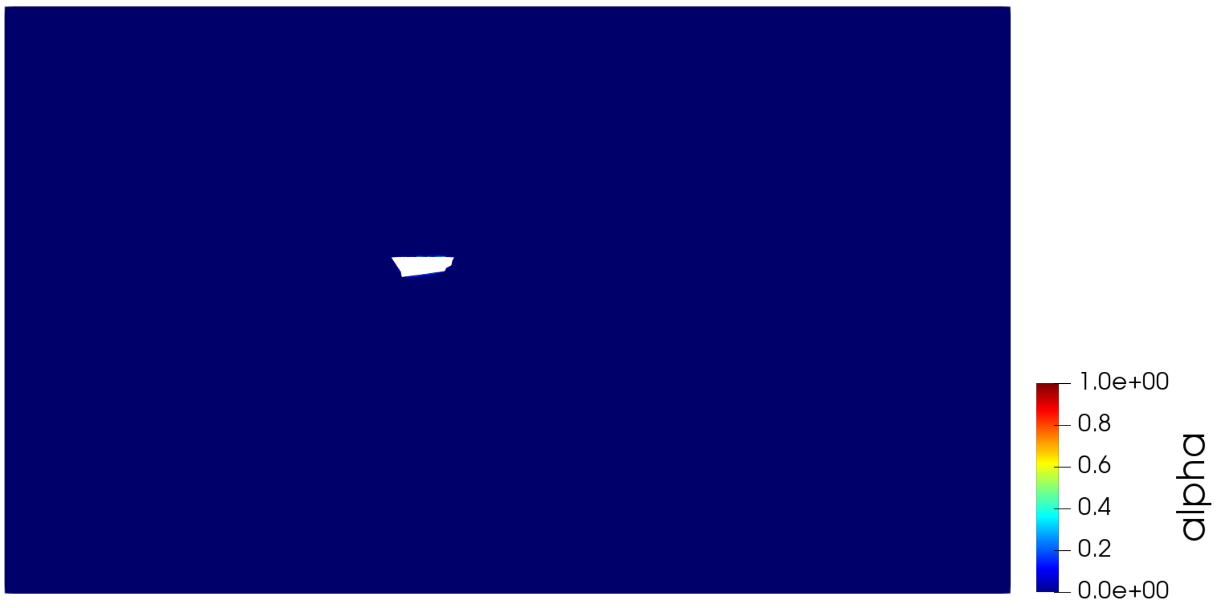


Figure 6.2: Cavity advances for the Esmeralda sector in El Teniente mine.

Figures 6.3-6.5 display the damage distribution in a plane parallel to the xy-plane. It is possible to see that the damage appears in the lateral boundary of the cavity and the damage is close to zero almost everywhere except in a circumscribed area of the cavity where the value of α never achieves the maximum value $\alpha = 1$. It is observed that the advance of the cavity also eliminates areas damaged in the previous steps as the cavity advances and a larger and more irregular cavity is created.

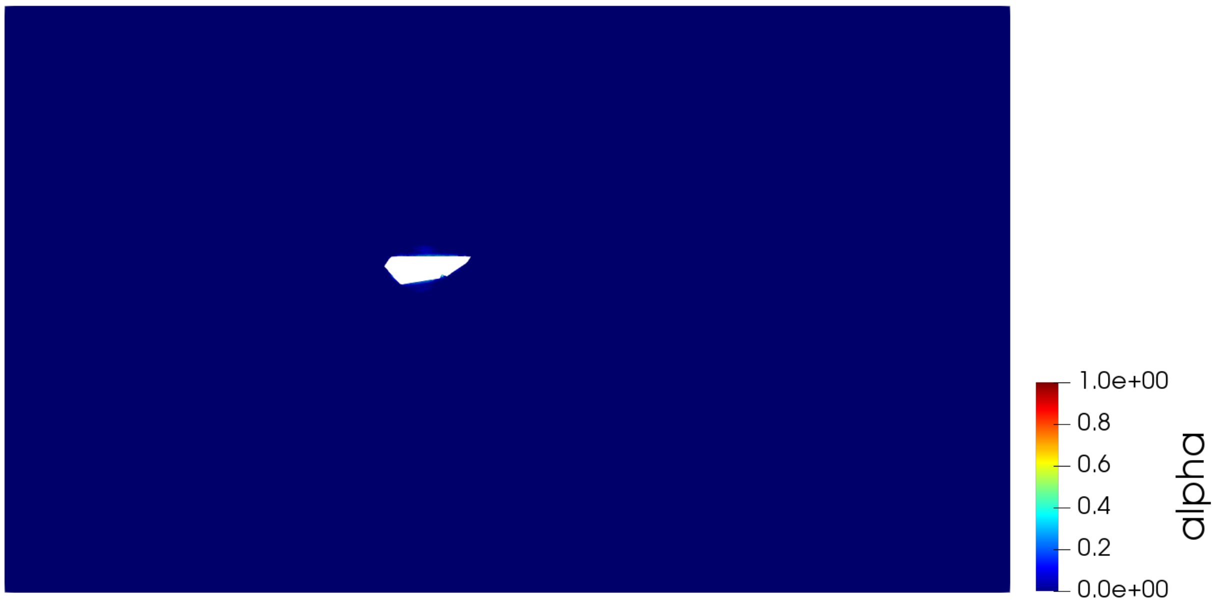
1.0e+00
0.8
0.6
0.4
0.2
0.0e+00
alpha

(a) $\Omega(t_0)$

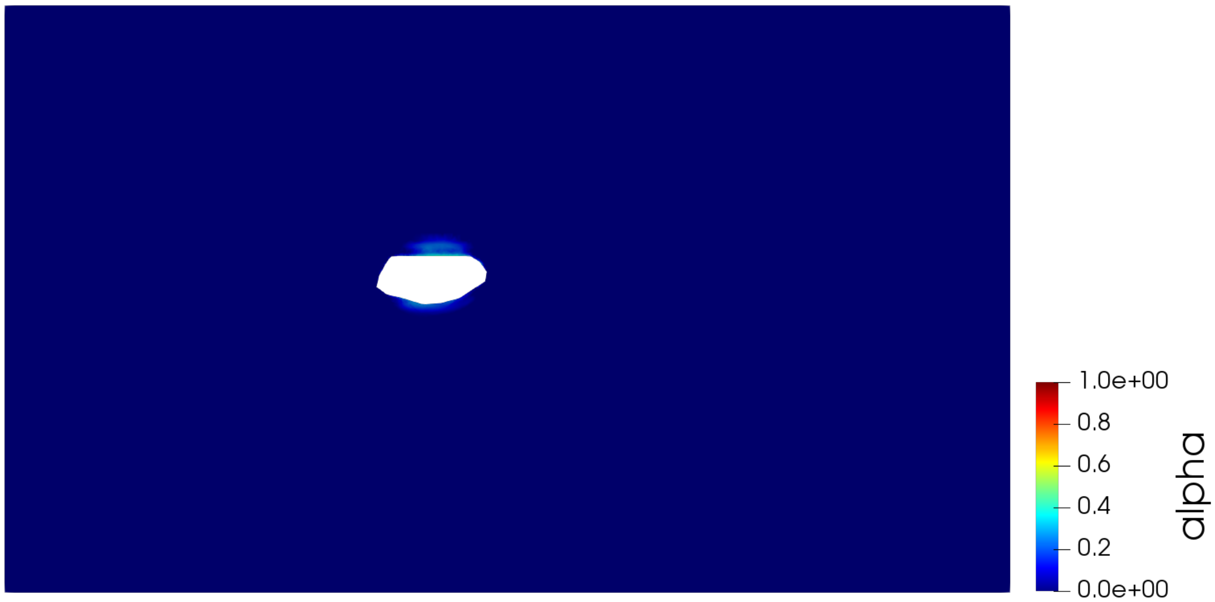


(b) $\Omega(t_5)$

Figure 6.3: Horizontal cut of the mesh for different time step.

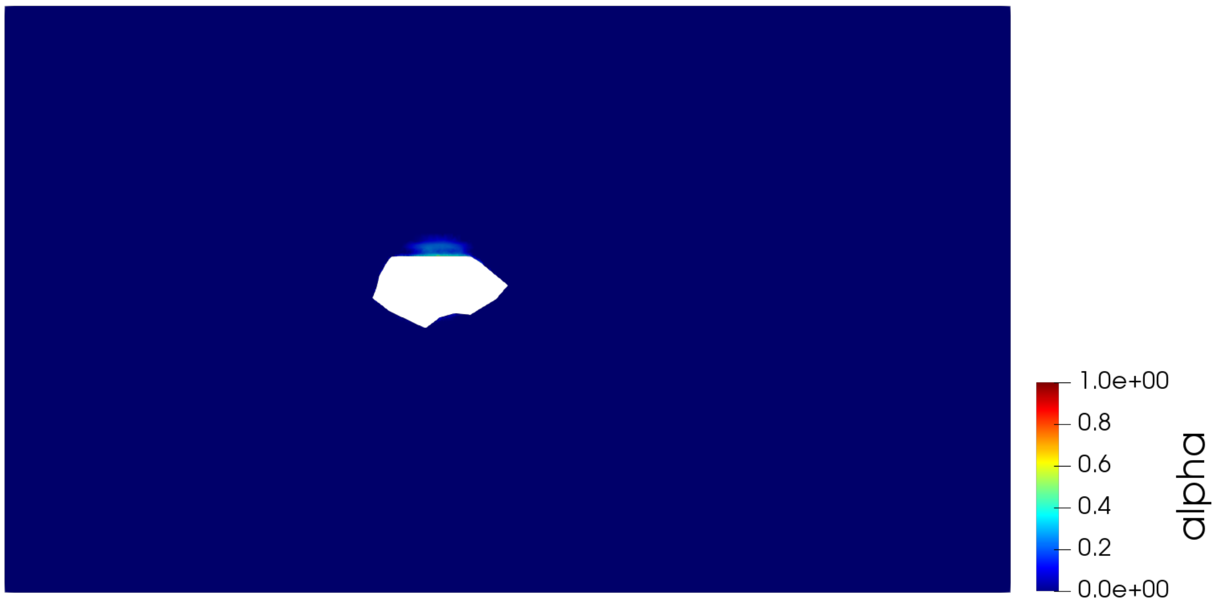


(a) $\Omega(t_{10})$

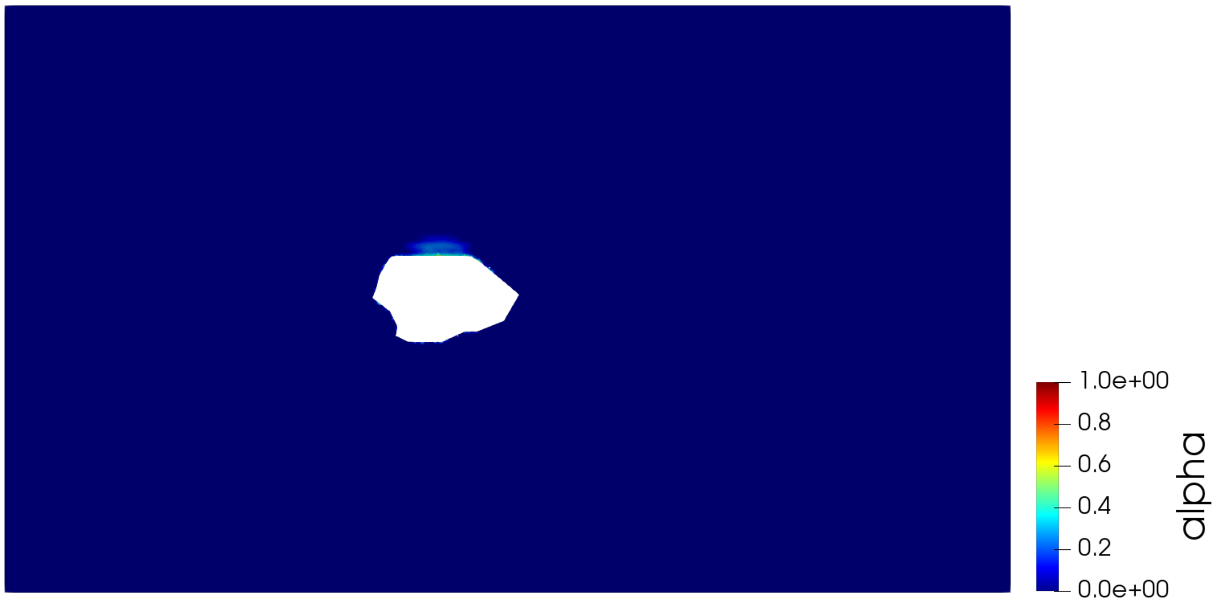


(b) $\Omega(t_{20})$

Figure 6.4: Horizontal cut of the mesh for different time step.



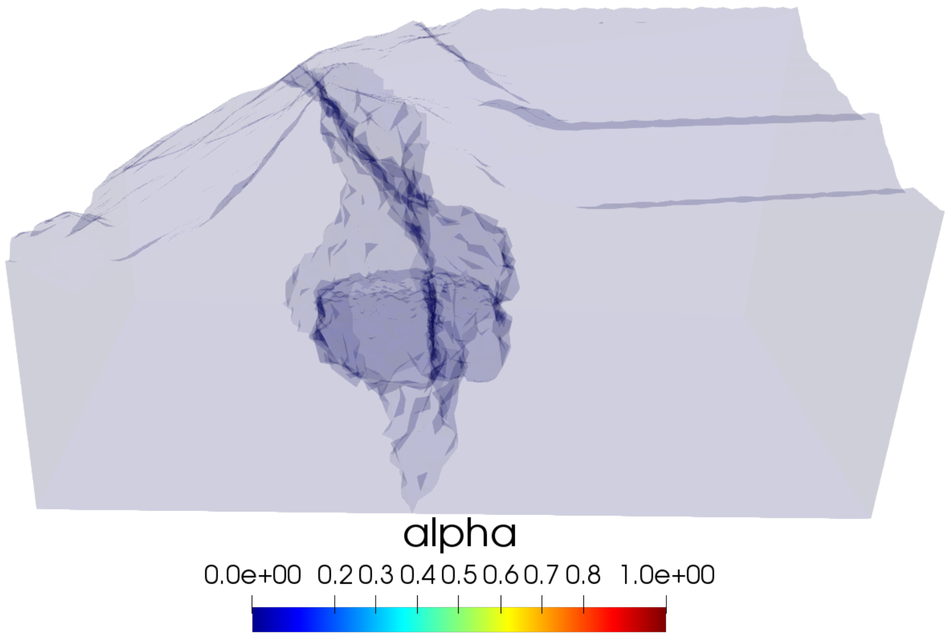
(a) $\Omega(t_{30})$



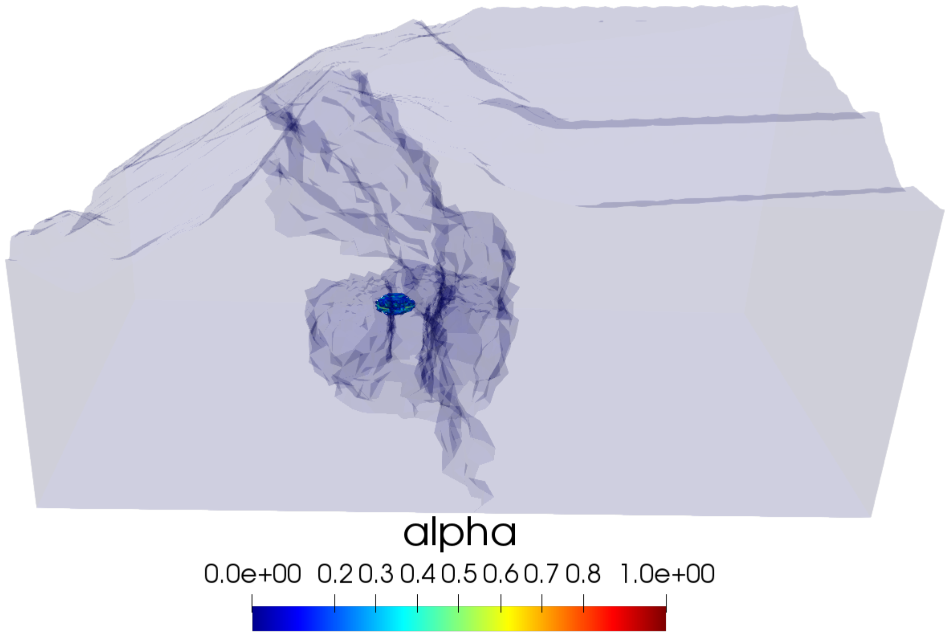
(b) $\Omega(t_{38})$

Figure 6.5: Horizontal cut of the mesh for different time step.

Figures 6.6-6.8 display the damage distribution produce by the cavity advance for different time steps in 3D. It is observed that as the advance of the cavities occurs, the damage begins to spread throughout the cavity increasing the value of the α parameter. It is possible to see that the damage is close to zero almost everywhere, but the material is fully damaged ($\alpha = 1$) at the upper boundary of the domain reaching the surface.

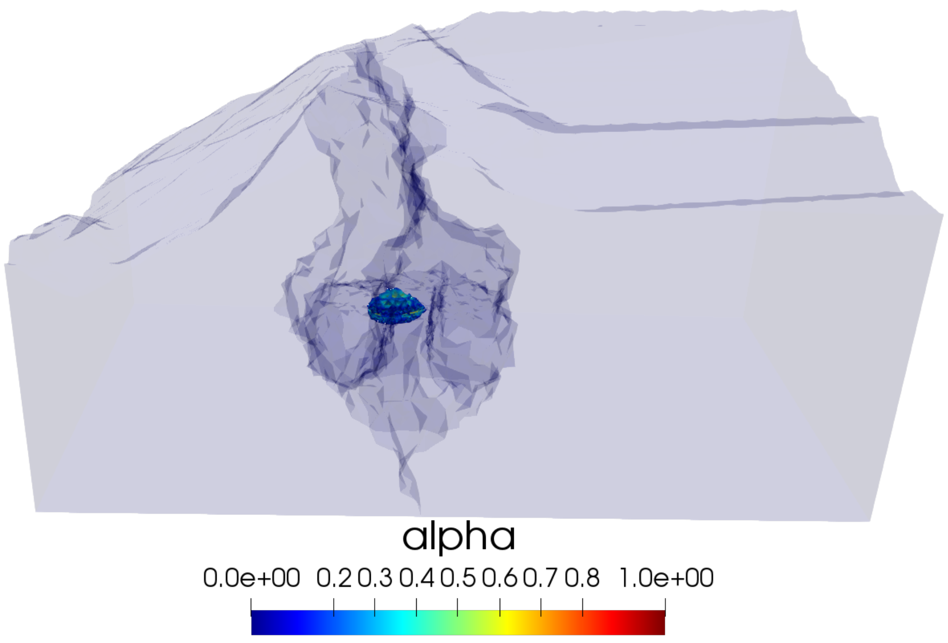


(a) $\Omega(t_0)$

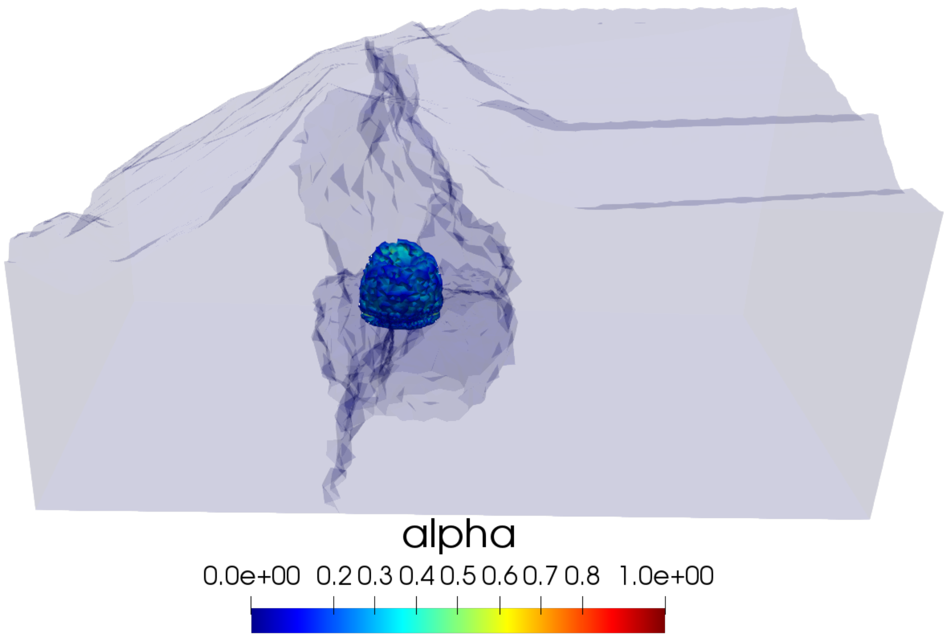


(b) $\Omega(t_5)$

Figure 6.6: Damage distribution in the rock mass in 3D.

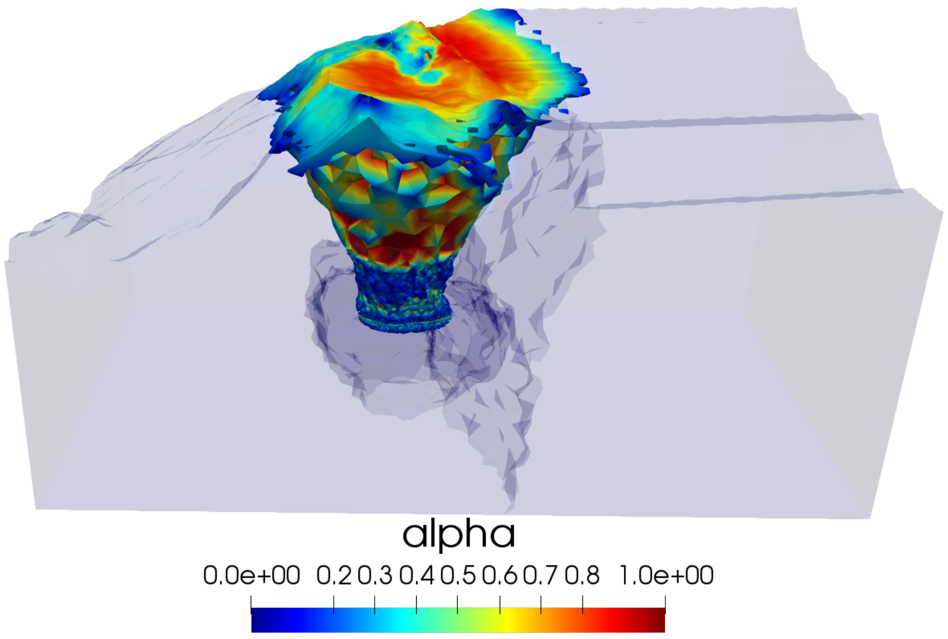


(a) $\Omega(t_{10})$

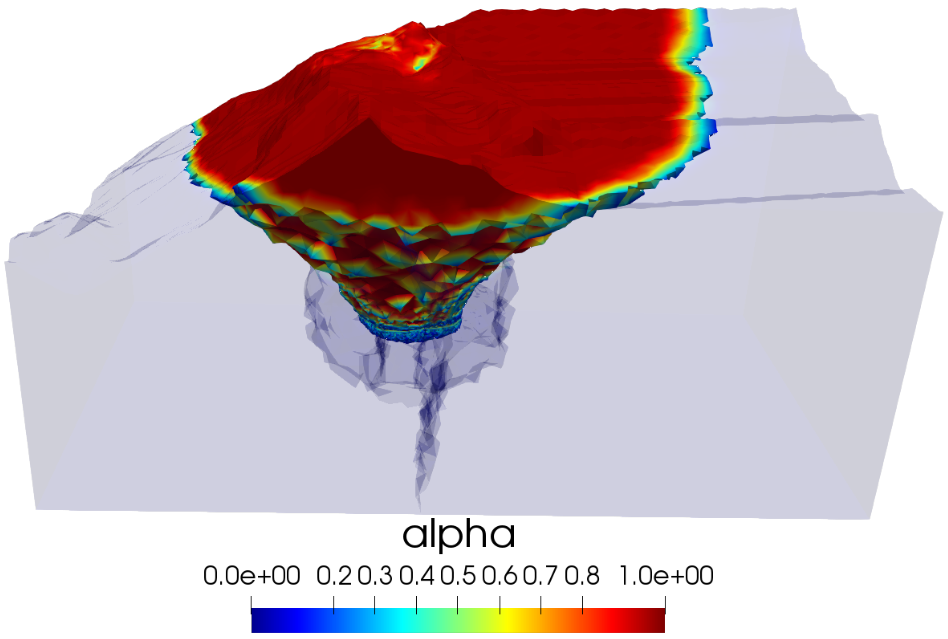


(b) $\Omega(t_{20})$

Figure 6.7: Damage distribution in the rock mass in 3D.



(a) $\Omega(t_{30})$



(b) $\Omega(t_{38})$

Figure 6.8: Damage distribution in the rock mass in 3D.

In this second test, we consider the Sub6 sector of the El Teniente mine. For this, we will define the cavity advance as shown in Figure 6.9, where we again consider an irregular cavity.

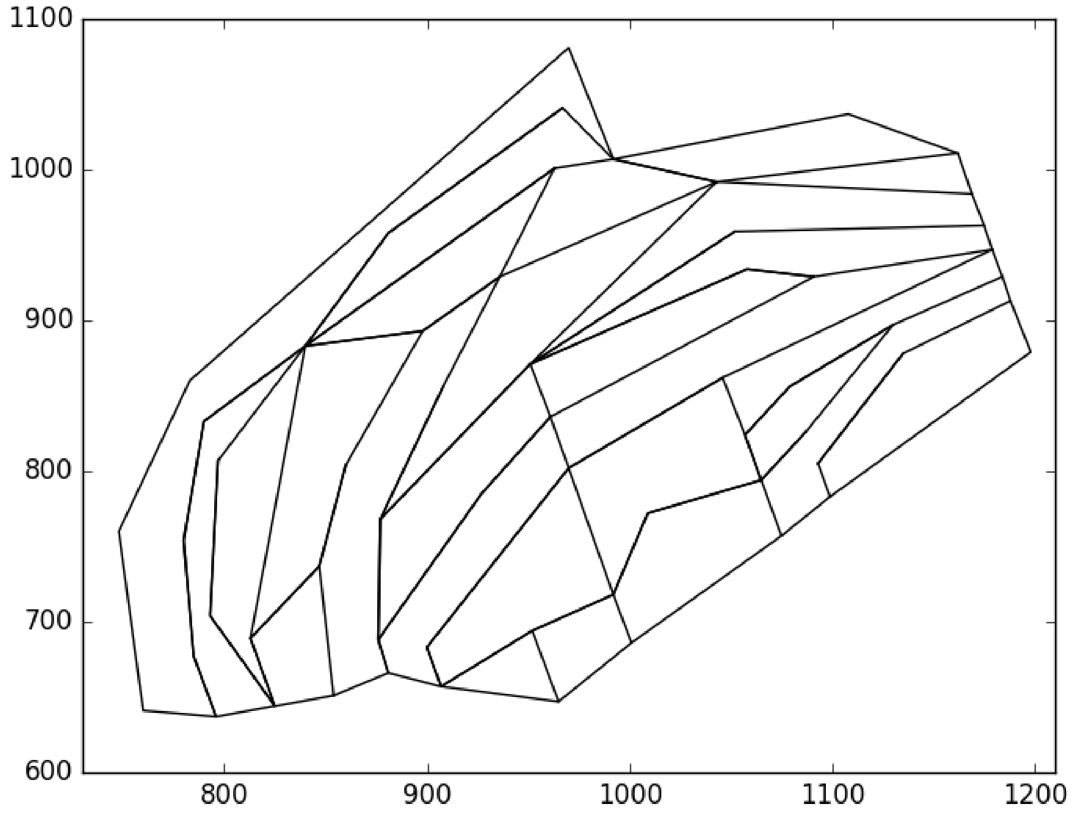
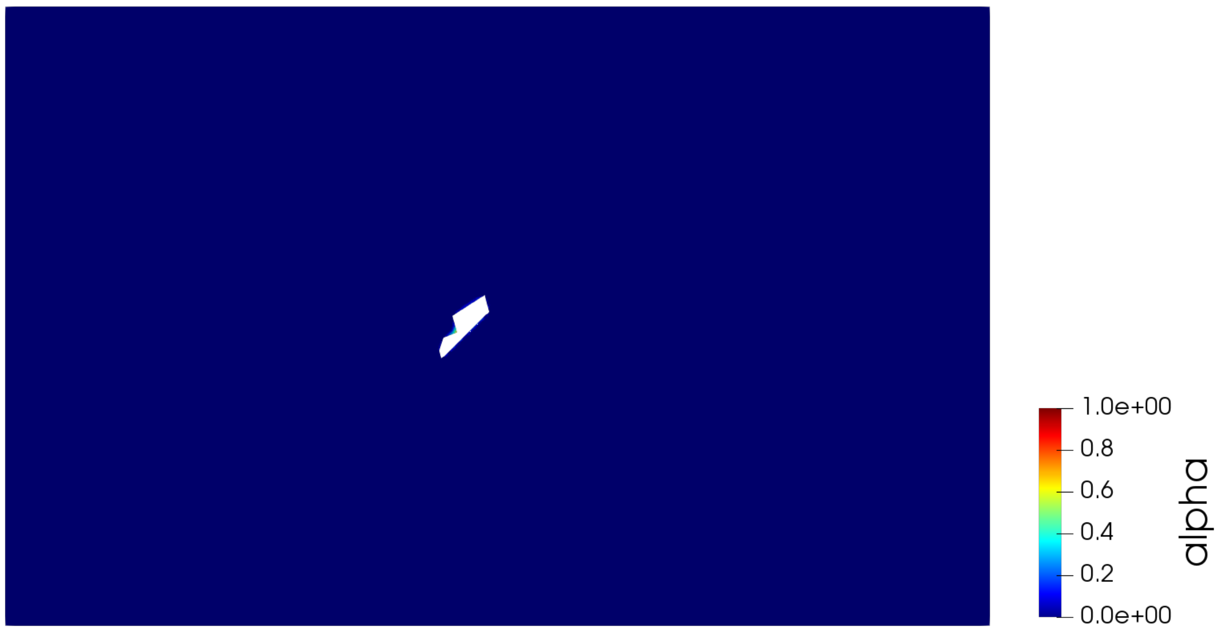


Figure 6.9: Cavity advances for the Sub6 sector in El Teniente mine..

Figures 6.10-6.12 display the damage distribution in a plane parallel to xy-plane. It is possible to see that the damage appears in the lateral boundary of the cavity and the damage is close to zero almost everywhere except in a circumscribed area of the cavity where the value of α never achieves the maximum value $\alpha = 1$. Again, it is observed that the advance of the cavity eliminates areas damaged in the previous steps as the cavity advances and a larger and more irregular cavity is created.

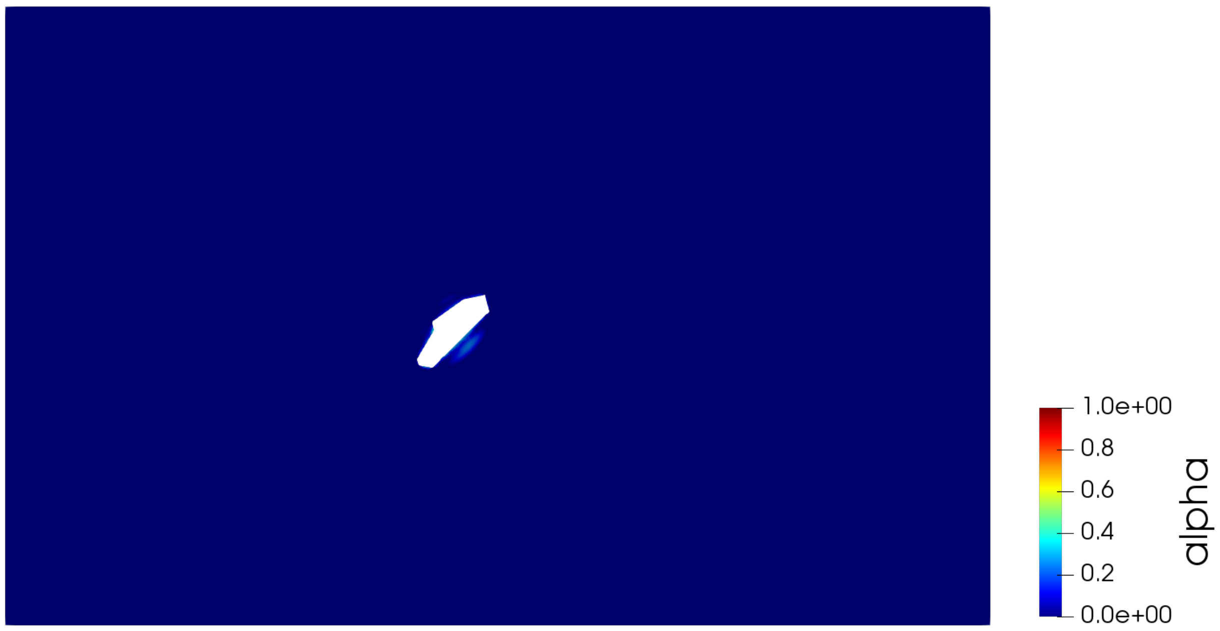
1.0e+00
0.8
0.6
0.4
0.2
0.0e+00
alpha

(a) $\Omega(t_0)$

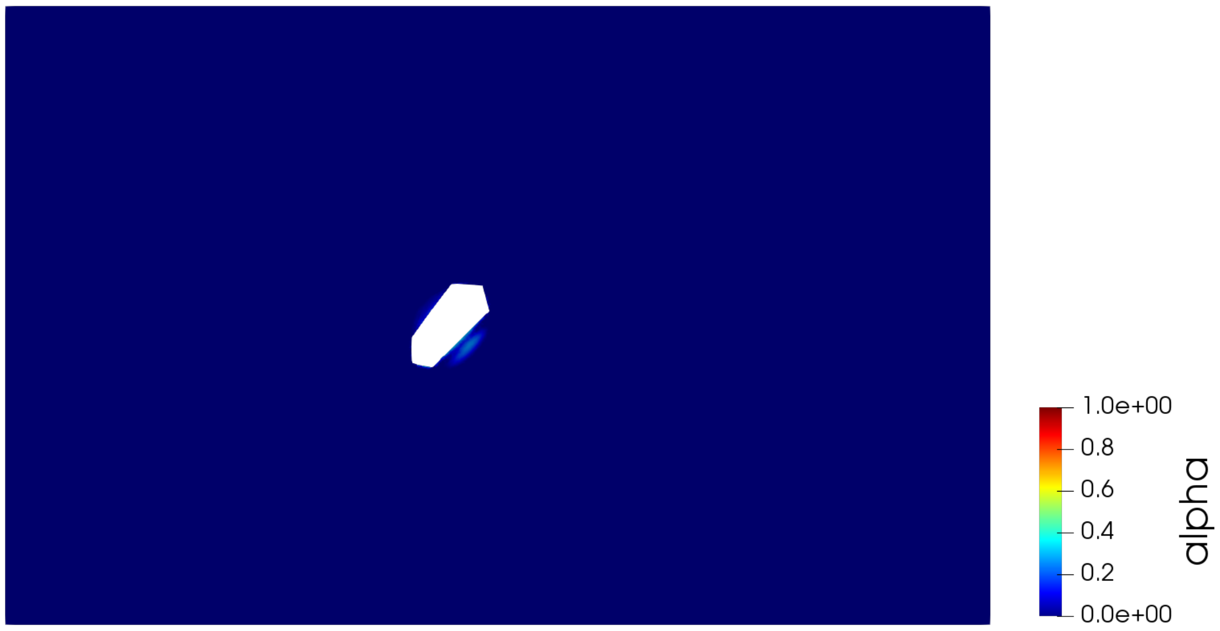


(b) $\Omega(t_5)$

Figure 6.10: Horizontal cut of the mesh for different time step.

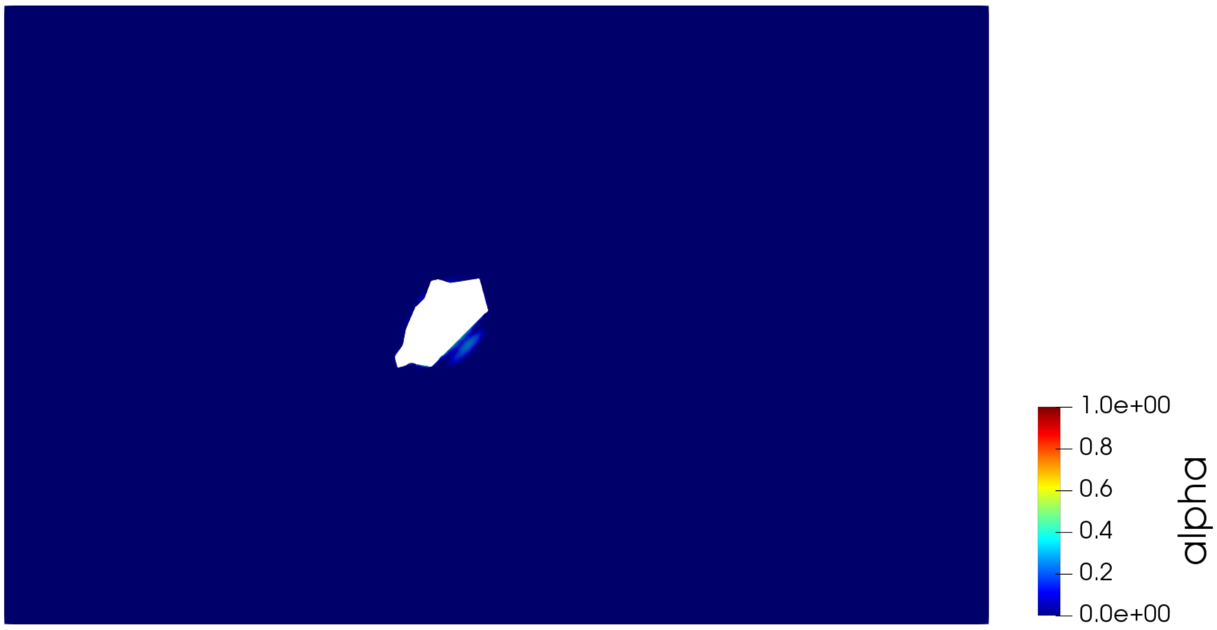


(a) $\Omega(t_{10})$

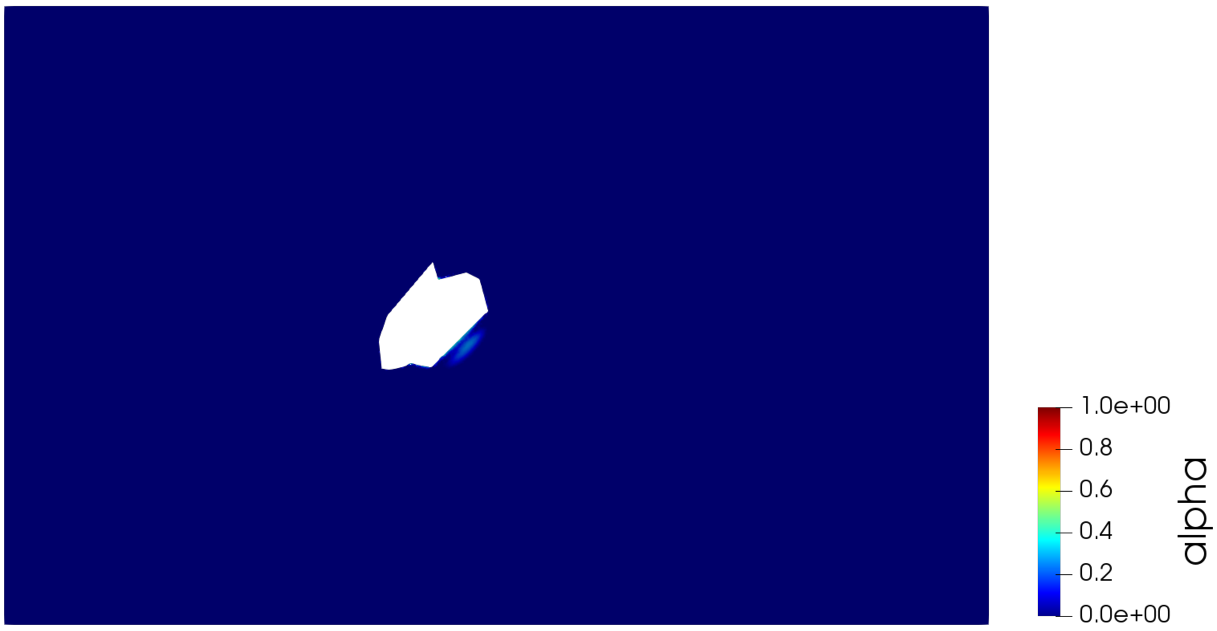


(b) $\Omega(t_{15})$

Figure 6.11: Horizontal cut of the mesh for different time step.



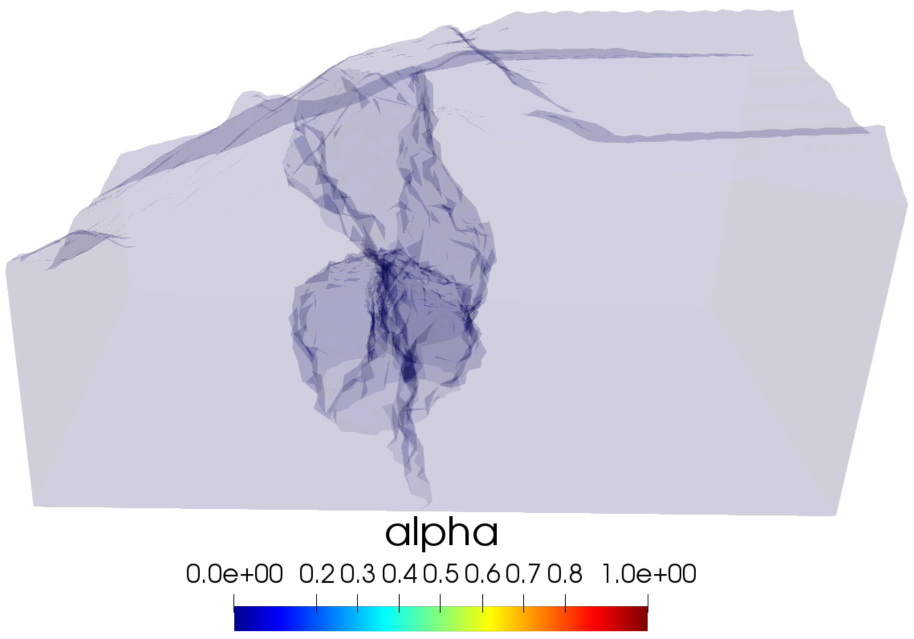
(a) $\Omega(t_{20})$



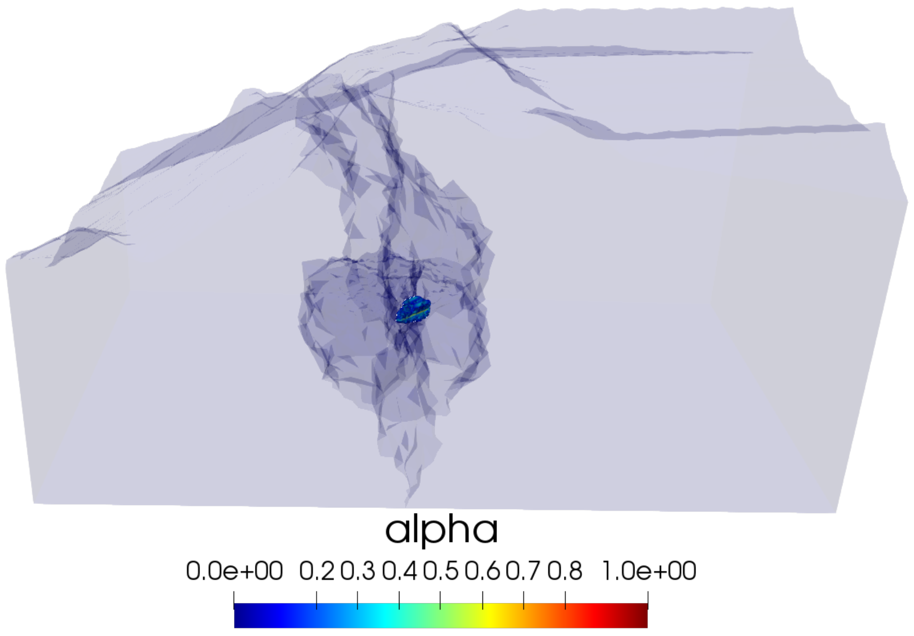
(b) $\Omega(t_{28})$

Figure 6.12: Horizontal cut of the mesh for different time step.

Figures 6.13-6.15 display the damage distribution produced by the cavity advance for different time steps. It is observed that as the advance of the cavities the damage begins to spread over the cavity increasing the value of the α parameter. It is possible see that the damage is close to zero almost everywhere, but the material is fully damaged ($\alpha = 1$) at the upper boundary of the domain reaching the surface.

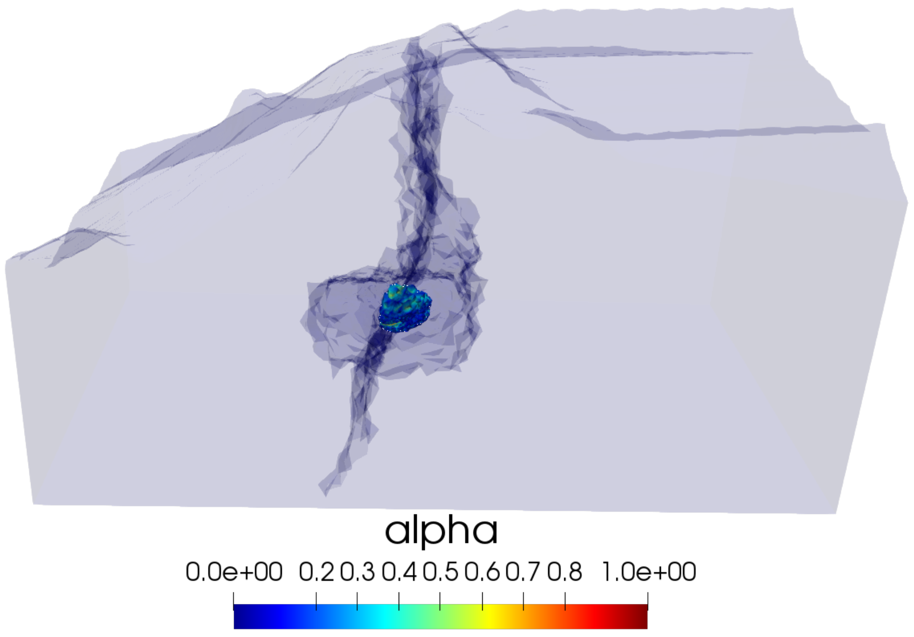


(a) $\Omega(t_0)$

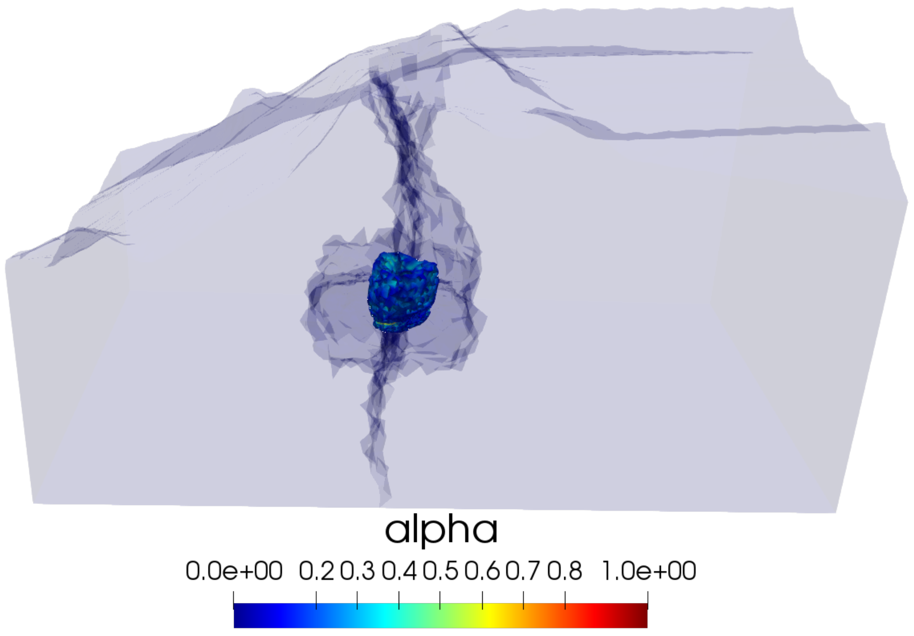


(b) $\Omega(t_5)$

Figure 6.13: Damage distribution in the rock mass in 3D.

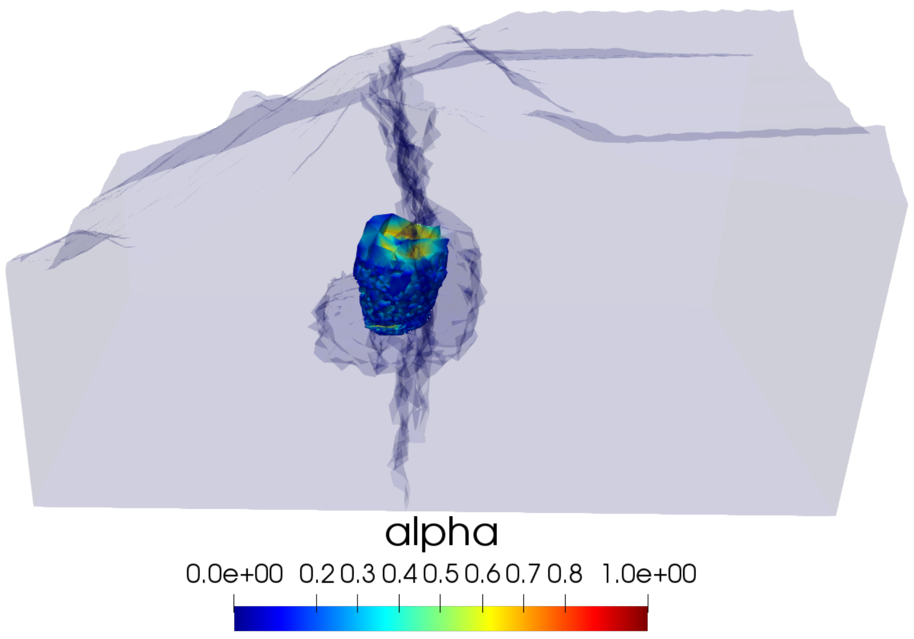


(a) $\Omega(t_{10})$

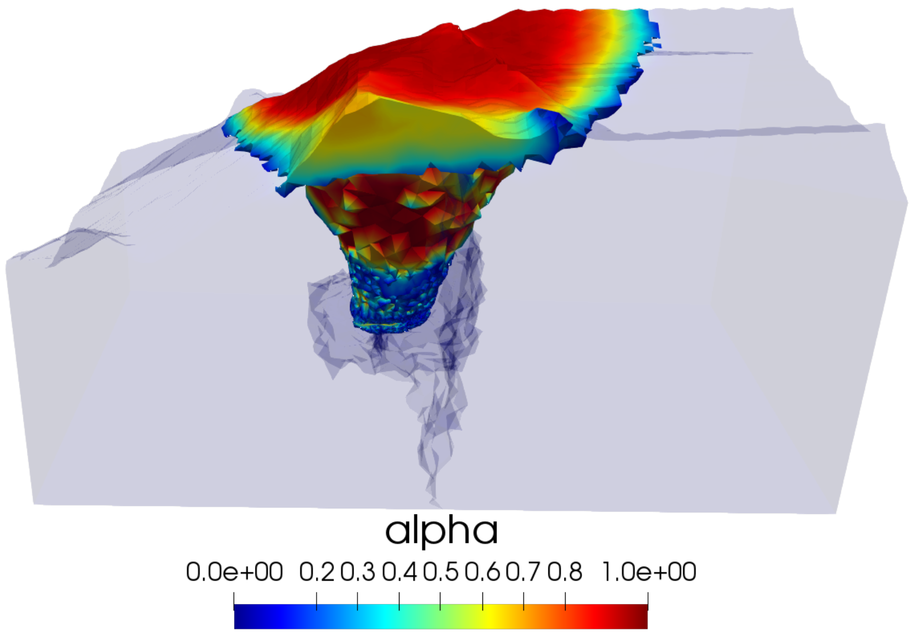


(b) $\Omega(t_{15})$

Figure 6.14: Damage distribution in the rock mass in 3D.



(a) $\Omega(t_{20})$



(b) $\Omega(t_{28})$

Figure 6.15: Damage distribution in the rock mass in 3D.

In the last test, we consider the Nuevo Nivel Mina sector of the El Teniente mine, where we define the irregular advances of the cavity as shown in Figure 6.16

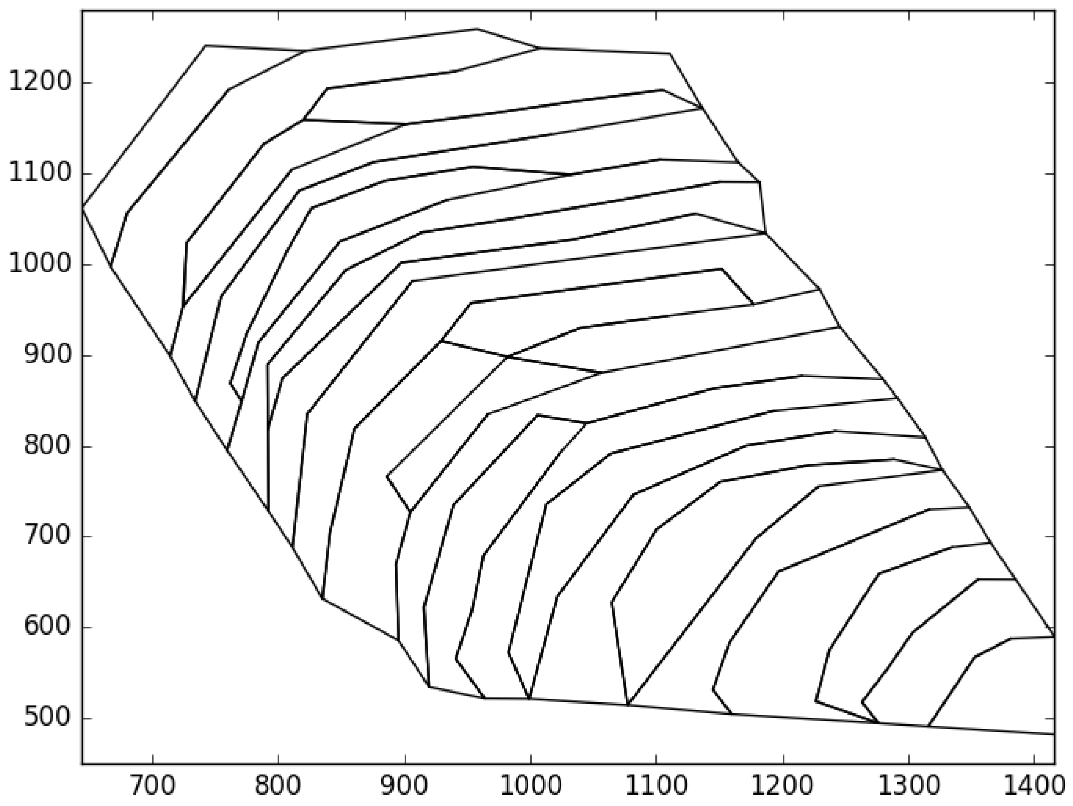
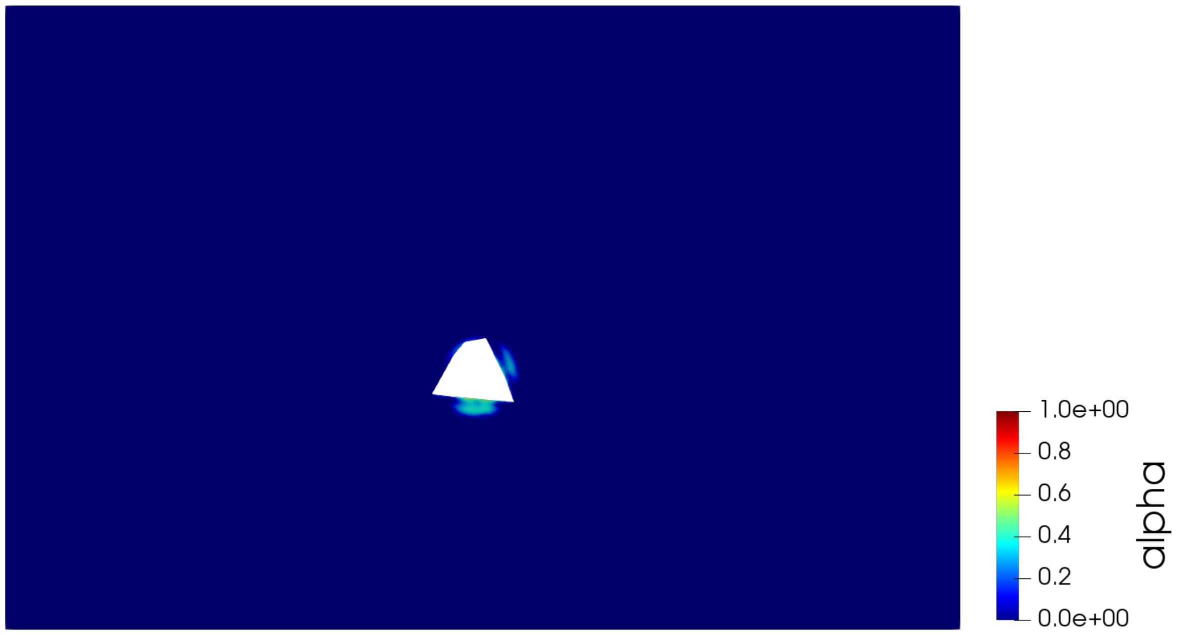


Figure 6.16: Cavity advances fot ge Nuevo Nivel Mina sector in El Teniente mine.

Figures 6.17-6.19 display the damage distribution in a plane parallel to xy-plane. It is possible to see that the damage appears in the lateral boundary of the cavity and it is close to zero almost everywhere close to zero almost everywhere except in a circumscribed area of the cavity where the value of α never achieves the maximum value $\alpha = 1$. It is also observed that the damage attains the full damage state around the first cavity advance.

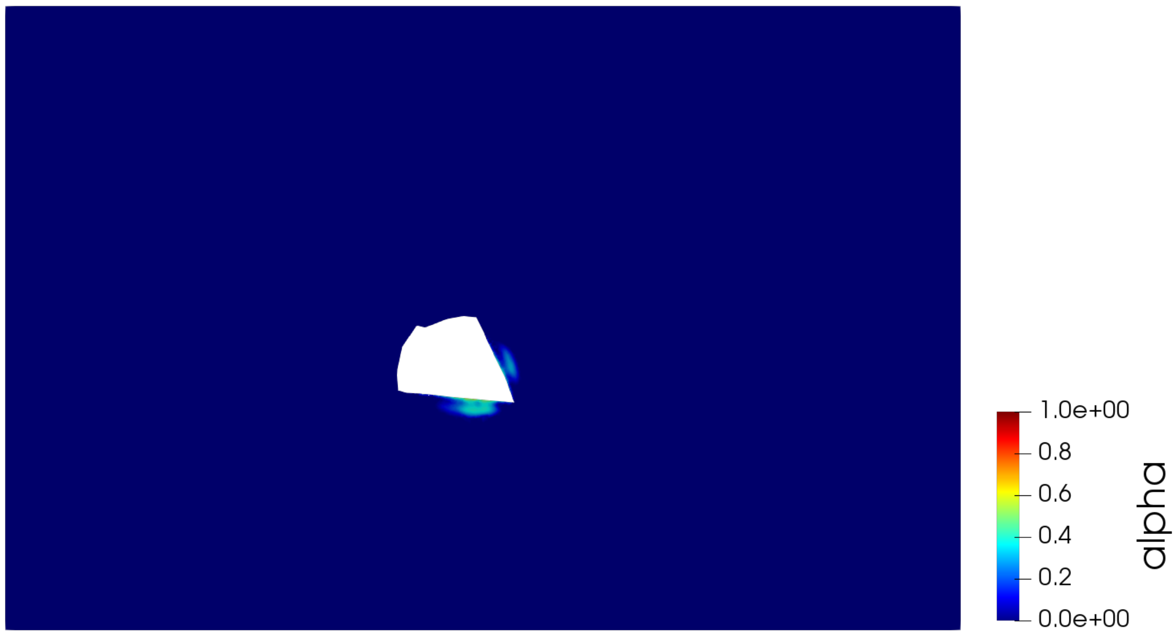
1.0e+00
0.8
0.6
0.4
0.2
0.0e+00
alpha

(a) $\Omega(t_0)$

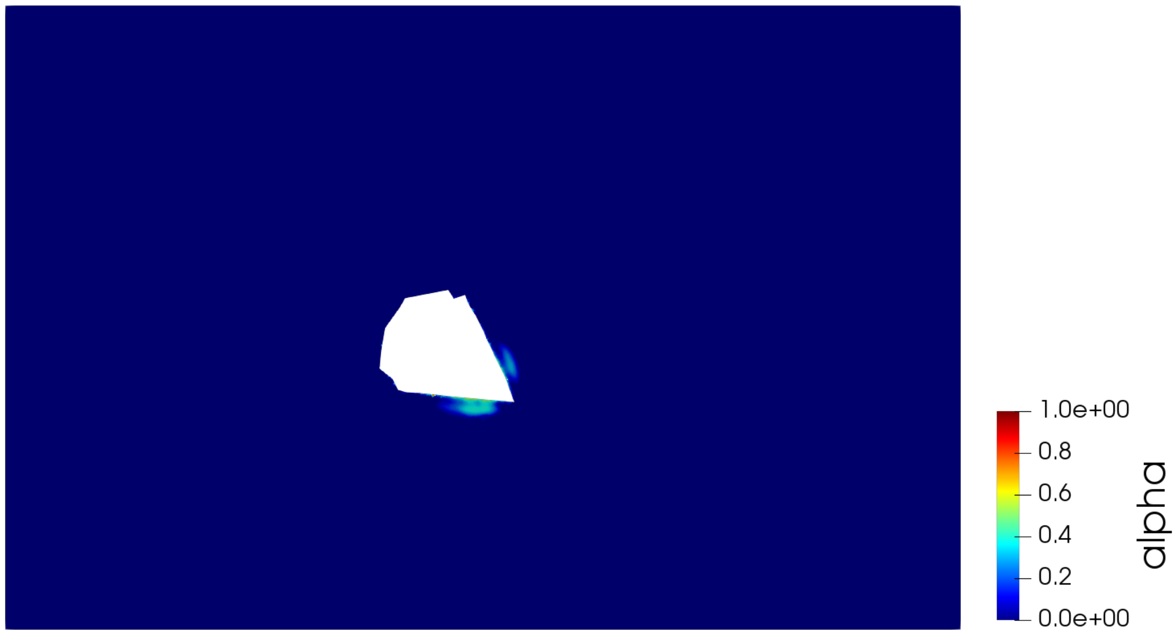


(b) $\Omega(t_5)$

Figure 6.17: Horizontal cut of the mesh for different time step.

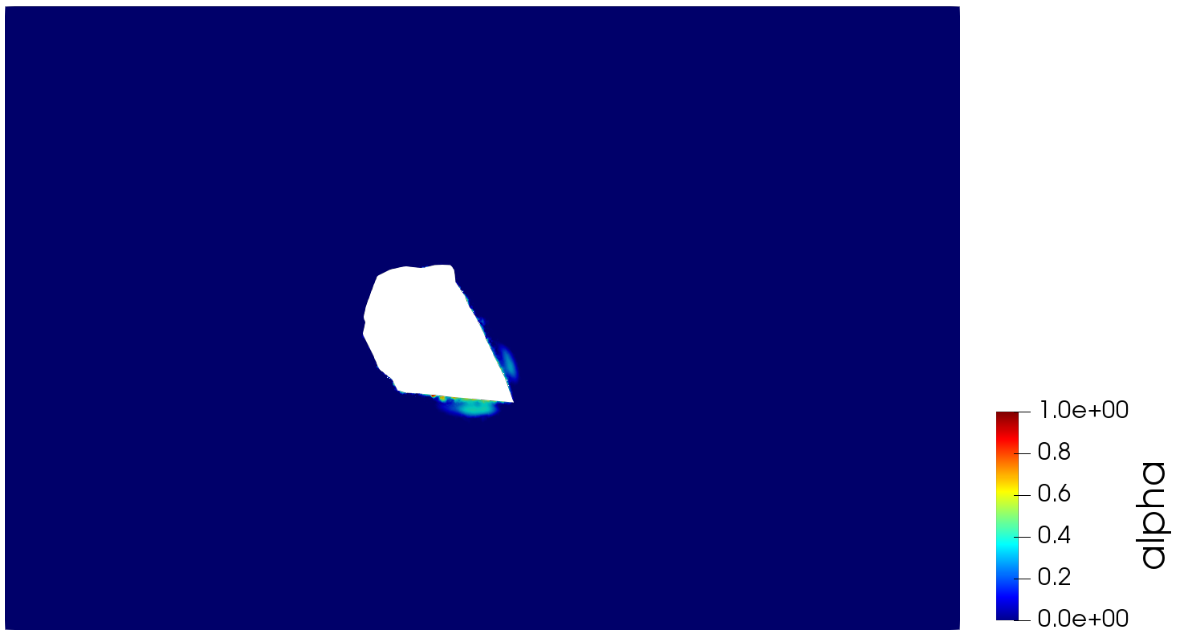


(a) $\Omega(t_{10})$

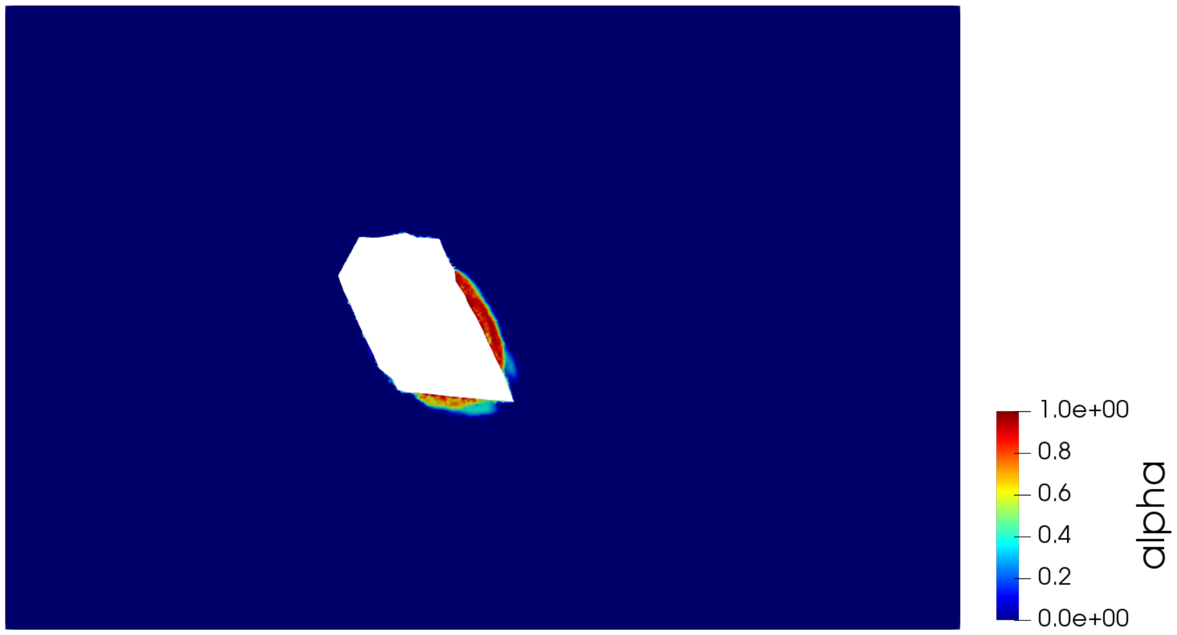


(b) $\Omega(t_{15})$

Figure 6.18: Horizontal cut of the mesh for different time step.



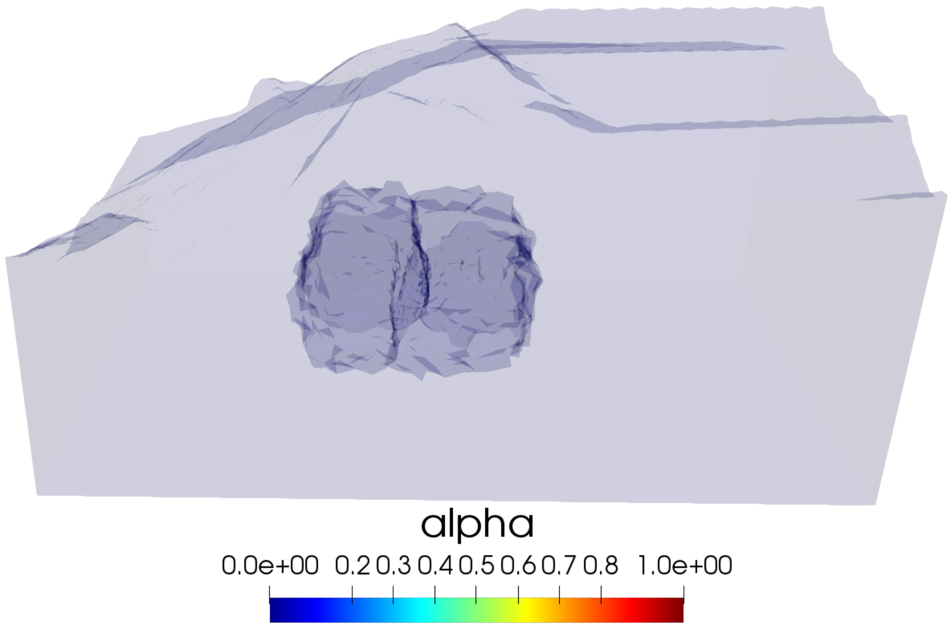
(a) $\Omega(t_{20})$



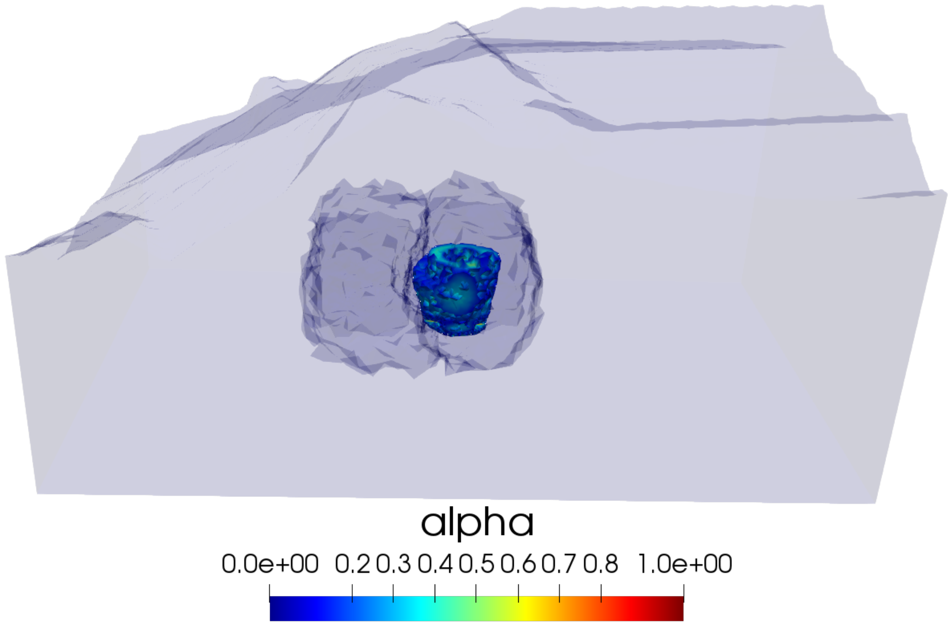
(b) $\Omega(t_{27})$

Figure 6.19: Horizontal cut of the mesh for different time step.

Figures 6.20-6.22 display the damage distribution produce by the cavity advance for different time steps. It is observed that as the cavity advances, the damage begins to spread throughout the cavity increasing the value of the α parameter faster than in the previous tests. It is possible see that the damage is close zero almost everywhere, but the material is fully damaged ($\alpha = 1$) at the upper boundary of the domain, reaching the surface.

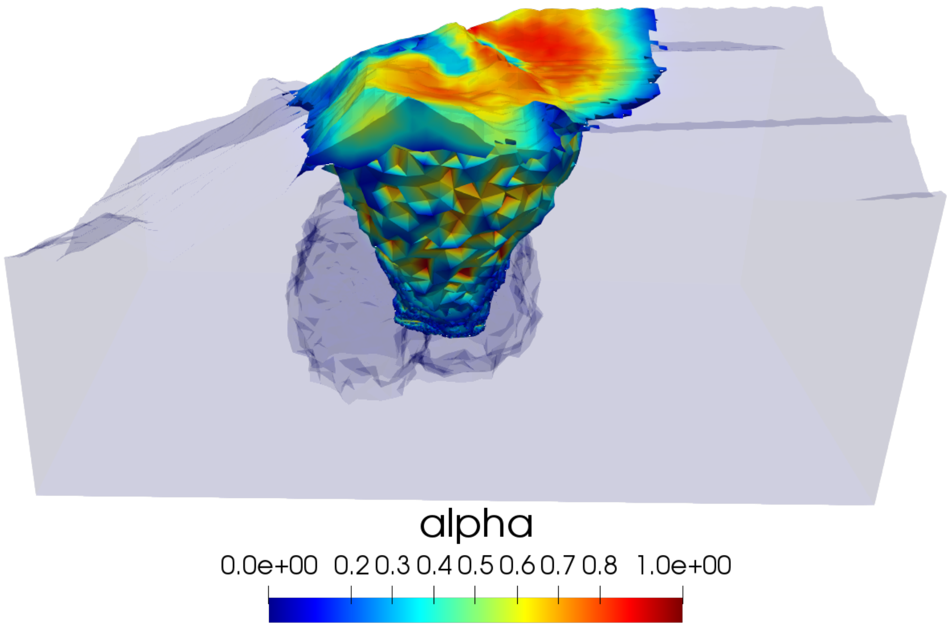


(a) $\Omega(t_0)$

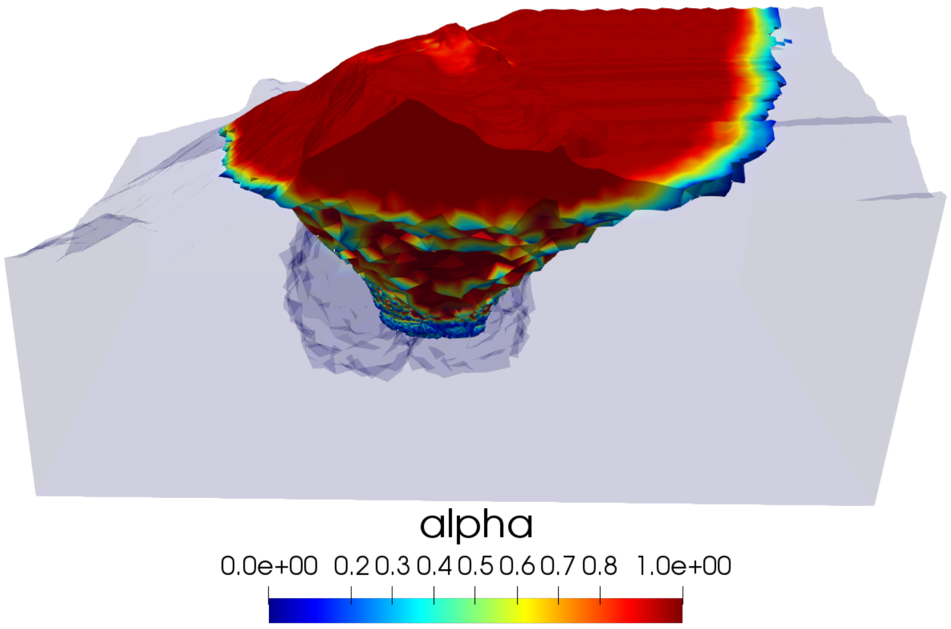


(b) $\Omega(t_5)$

Figure 6.20: Damage distribution in the rock mass in 3D.

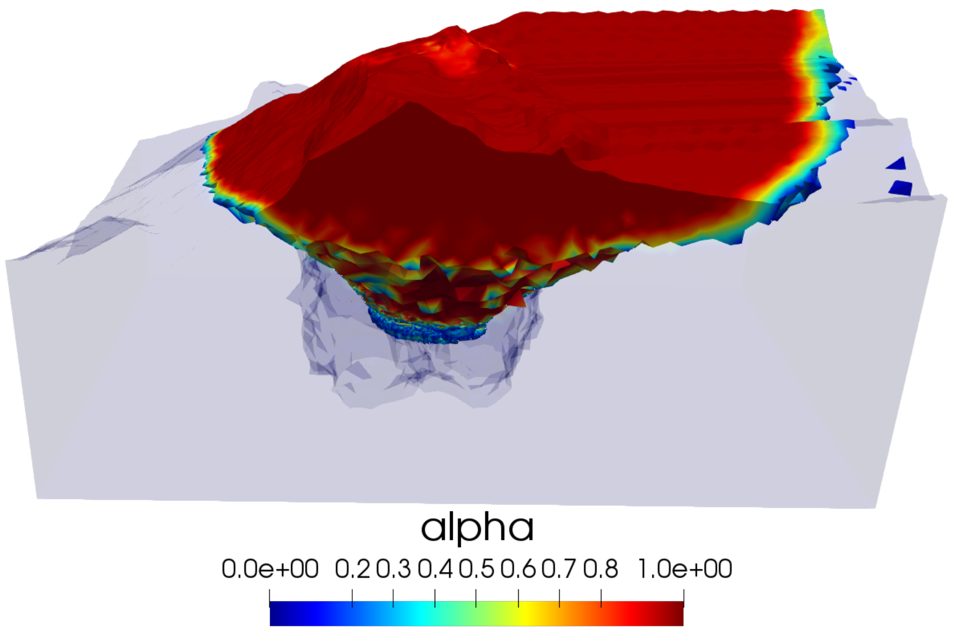


(a) $\Omega(t_{10})$

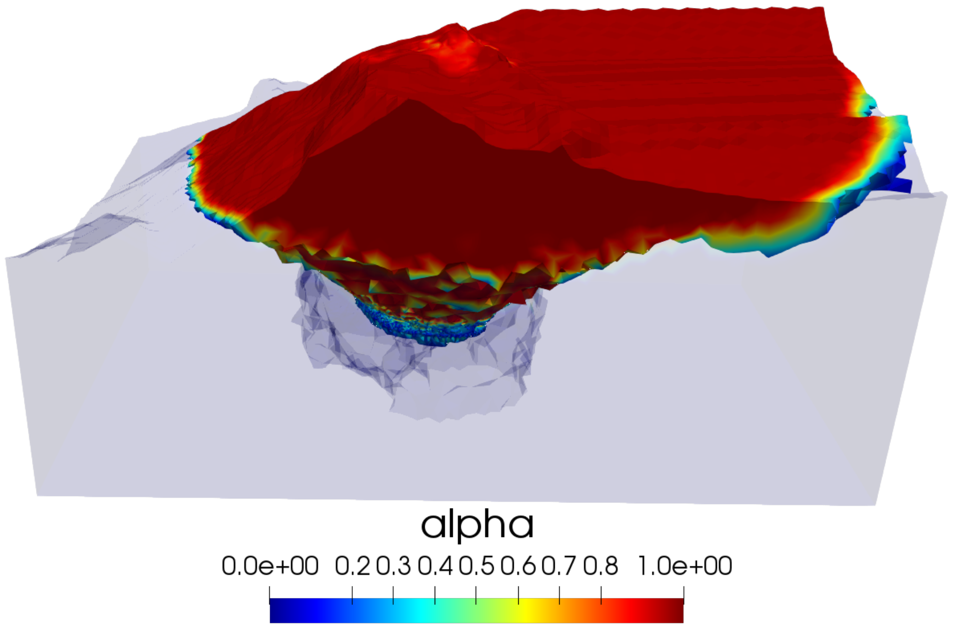


(b) $\Omega(t_{15})$

Figure 6.21: Damage distribution in the rock mass in 3D.



(a) $\Omega(t_{20})$



(b) $\Omega(t_{27})$

Figure 6.22: Damage distribution in the rock mass in 3D.

The results show that our proposed model captures the damage produced by the block caving process not only for a synthetic example but for more complex geometries and cavities that represent a real mine. Accordingly, this model can be used to predict the damage caused by the advance of different cavities in underground mining and thus be a good approximation for understanding the behavior of the rock as the Block Caving process is developed.

Conclusion

We defined a new approach to modeling the block caving process based in gradient damage model shown in [40], we propose a modified model separating the spherical part and the deviatoric part of the stress tensor σ in the damage criterion. We have shown in this work that our new damage model is a good candidate to approximate this underground mining process. In this work, we consider 2D examples to compare our results with the classical gradient damage model [40] and gradient damage model for shear fracture [35], where these models do not recover the damage produced by the mining effect since the damage appears in the bottom of the rock mass or almost in the whole domain. We also consider 3D examples where, in the first instance, we validate our model with a laboratory example in the "testigo" and then with the validated model carry out different tests for synthetic examples that model the block caving process. Then, we proposed a new algorithm that, using fewer iterations, is able to recover the effect of the underground mining in the rocky mass. Lastly, we use the results obtained previously to be able to model the effect of hydraulic fracturing and perform simulations on topographies that simulate a real mine.

In order to extend the results and obtain a more realistic model, we have identified some strategies listed below:

1. To consider the numerical experiments in a 3D case, taking into account the edge conditions and surface forces for this case.
2. To consider the temporal effect following the dynamic approach in [37].
3. Introduce a temporary dissipation in the model, which allows the study of the variation of the damage as a function of time in order to extend the results of the quasi-static model to a more realistic environment.
4. Incorporate into the model the effects of the granular phase produced when the material is totally damaged combined with plasticity models (see [39]).
5. Incorporate into the model the effects of density variation due to the mass loss caused by the material extracted during the block caving process.

Part II

Numerical reconstruction of CGO of conductivity systems

Introduction

Electrical impedance tomography (EIT) is an imaging technique in which electrodes are placed on the surfae of the body, and low-frequacy current is applied on rhe electrodes which can then be measured. The measurement is repeated for a specifed set of current patterns, or choices of current amplitudes at each electrode. The resulting current-to-voltage map serves as data for the inverse problem.

The mathematical model of EIT is called the inverse conductivity problem: recover the conductivity distribution inside the body given electric boundary measurements performed on the surface of the body. It is a nonlinear and secerely ill-posed problem.

Two questions were posed by Calderón in [14] which is often pointed to as the mathematical beginnings of the inverse conductivity problem. The first queston is, Is it possible to uniquely determine the conductivity of an unknown object from boundary measurements? The other question is, How can this conductivity be reconstructed? Calderón shows that the linearized problem has an affirmative answer too the uniqueness question, and he proposed a linearized reconstruction scheme. His methods have inspired a multitude of research on the problem, including the use of complex geometrical optics (CGO) solutions for answering both of his quetions and for designing a regularized inversion method for practical EIT.

In this work, following [5, 4], we study the numerical computation of the CGO solution to the conductivity system

$$\mathrm{di}v\,(\sigma \cdot \nabla U) \;=\; 0, \quad \text{in} \quad \mathbb{R}^2, \tag{6.1}$$

where, these solutions are specified by their asymptotics

$$U(z,k) = \mathrm{e}^{\mathrm{i}kz}\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix} + \mathcal{W}(z)\right), \tag{6.2}$$

with $\mathcal{W}(z) = \begin{pmatrix} \mathcal{W}_1(z) \\ \mathcal{W}_2(z) \end{pmatrix}$ that satisfies

$$\mathcal{W}_{\mathrm{i}}(z) = \mathcal{O}\left(\frac{1}{z}\right), \text{ as } |z| \to \infty, \text{ for i} = 1, 2.$$

Here k is a complex parameter, i is the imaginary unit, the conductivity $\sigma \in \mathbb{M}_{2\times 2}(\mathbb{R})$ is a given symmetric, positive definite matrix function, with $\sigma_{\ell m}(z) = 1$, for $\ell = m$, and

$\sigma_{\ell m}(z) = 0$, for $\ell \neq m$, outside a compact set Ω. For simplicity let us take Ω as the unit disk, this is not a significant loss of generality, as a large class of more general setting can be reduced to this case.

This problem can be considered as a vectorial extension of the scalar case of Calderón's inverse conductivity problem. In 1980 Alberto Calderón published a short paper entitled "On an inverse boundary value problem" [14]. This pioneer contribution has motivated many developments in inverse problem, in particular in the construction of "Complex Geometrical Optics" solutions of partial differential equations to solve several inverse problems. The problem that Calderón considered was wheter one can determinate the electrical conductivity of a medium by making voltage and current measurements at the boundary of the medium. This inverse method is know as Electrical Impedance Tomography (EIT). EIT also arises in medical imaging given that human organs and tissues have quite diferent conductivities [32, 60, 29]. This inverse problem has also been used to detect leaks from buried pipes [31].

Let us describe the details of this problem as follows. Let $\Omega \subset \mathbb{R}^N$ be a bounded domain with smooth boundary. The electrical conductivity of Ω is represented by a bounded and positive function $q(x)$. In the absence of sink or sources of current, the equation for potential is given by

$$\mathrm{div}\,(q\nabla u) = 0, \text{ in } \Omega,$$

where $q\nabla u$ represents the current flux.

Given a potential $\phi \in H^{1/2}(\partial\Omega)$ on the boundary, the induced potential $u \in H^1(\Omega)$ solves the Dirichlet problem

$$\begin{array}{rcll} \mathrm{div}\,(q\nabla u) & = & 0, & \text{in} \quad \Omega, \\ u & = & \phi, & \text{on} \quad \partial\Omega. \end{array} \tag{6.3}$$

The Dicichlet to Neumann map, or vortage to current map, is given by

$$\begin{array}{rcl} \Lambda_q : H^{1/2}(\partial\Omega) & \to & H^{-1/2}(\partial\Omega), \\ \phi & \mapsto & q\, \frac{\partial u}{\partial n}\Big|_{\partial\Omega}, \end{array}$$

where n denotes the unit outer normal to $\partial\Omega$.

The inverse problem is to determine q knowing Λ_q. It is difficult to find a systematic way of prescribing voltage measurements at the boundary to be able to find the conductivity. Calderón took instead a different route. Using the divergence theorem we have

$$Q_q(\phi) := \int_\Omega q|\nabla u|^2 \mathrm{d}x = \int_{\partial\Omega} \Lambda_q(\phi) = \phi \mathrm{d}S, \tag{6.4}$$

where $\mathrm{d}S$ denotes surface measure and u is the solution of (6.3). $Q_q(\phi)$ is the quadratic form associated to the linear map $\Lambda_q(\phi)$, and know $\Lambda_q(\phi)$ or $Q_q(\phi)$ for all $\phi \in H^{1/2}(\partial\Omega)$ is equivalent. $Q_q(\phi)$ measures the energy needed to maintain the potential ϕ at the boundary. Calderón's point of view is that if one looks at $Q_q(\phi)$ the problem is changed to finding enough solutions $u \in H^1(\Omega)$ of the equation (6.3) in order to find q in the interior.

In [14], Calderón used complex exponentials harmonics functions $u = \mathrm{e}^{x\cdot\rho}$ and $v = \mathrm{e}^{-x\cdot\rho}$, where $\rho \in \mathbb{C}^N$ and $\rho \cdot \rho = 0$, to prove that the linearization of (6.4) is injective at constant conductivities. He also gave an approximation formula to reconstruct a conductivity which is, a priori, close to a constant conductivity.

Several uniqueness results have been obteined for the inverse conductivity problem, for example, in dimension higher than two for smooth conductivities by Sylvester and Uhlmann in 1987 [56]. In dimension two, Nachman [45] produced in 1995 a uniqueness result for conductivities with two derivatives. Earlier, the problem was solved for piecewise analytic conductivities by Koh and Vogelius in [33] and [34] and the generic uniqueness was established by Sun and Uhlmann [54]. In particular, in dimension two, Astala and Päivärinta [5] proved that the Dirichlet-to-Neumann map, Λ_σ, uniquely determines the conductivity $\sigma \in L^\infty(\Omega)$, $0 < c \leq \sigma$. In [51], Santacesaria propose a first step to tackle the Calderon's problem in $\mathbb{R}^n$, based in the Astala and Päivärinta method [5] and Clifford algebras.

The crucial technical tools for the uniqueness results are Complex Geometrical Optics (CGO) solutions, sometimes also called exponentially growing solutions. These solutions have their origin in optics, and the complex-valued CGO solutions have exponential growth in certain directions and exponential decay in others. They were first time introduced by Faddeev in 1966 [18] and later rediscovered in the context of inverse problems. CGO solutions are a valuable tool both theoretically and computationally points of view since many proofs involving them are constructive and lend themselves well to computational algorithms. For a thorough survey see [57].

In dimension 2, Astala and Päivärinta [5] use the CGO solutions to solve the Calderón's problem. In this case (with L^∞-conductivities) the CGO solutions need to be constructed via the Beltrami equation

$$\overline{\partial} f_\mu = \mu \overline{\partial f_\mu}, \tag{6.5}$$

where μ is a compactly supported L^∞ functions, connected to q by the identity

$$\mu = \frac{1-q}{1+q}. \tag{6.6}$$

Indeed, the respective complex CGO solution are related by the equation

$$2u(z,k) = f_\mu(z,k) + f_{-\mu}(z,k) + \overline{f_\mu(z,k)} - \overline{f_{-\mu}(z,k)}. \tag{6.7}$$

The simple reason behind these identities is that the real part $u(z,k)$ of $f_\mu(x,k)$ solves the equation (6.3) while the imaginary part solves the same equation with q replaced by $1/q$.

Then, an asymptotical condition is required as well

$$f(z,k) = \mathrm{e}^{\mathrm{i}kz}(1 + \omega(z,k)),\ \omega(z,k) = \mathcal{O}\left(\frac{1}{z}\right), |z| \to \infty. \tag{6.8}$$

The numerical computation of the CGO solutions of the equation (6.5) was first time introduced in [4], the authors proposed a complicated method to compute $\omega(z,k)$ in (6.8) via the solution of a $\mathbb{R}$-linear integral equation based on periodization, truncation of a Neumann series, discretization, Fast Fourier Transform (FFT), and the GMRES method [50]. In [17], a simpler numerical method for solving the same $\mathbb{R}$-linear integral equation was proposed, which solves the $\mathbb{R}$-linear integral equation in the unit disk directly, based on the fast algorithm in [16]. In [28] Huhntanen and Perämäki introduced an efficient method for the computation of the CGO, where they considered a new way to discretize the $\mathbb{R}$-linear integral equation.

For our problem, let us consider the set of all symmetric 2-matrices, $\mathcal{S}_2$, equipped with the inner product $M \cdot N = trace(MN)$ and the norm

$$\|M\|_{\mathcal{S}_2} = (M \cdot M)^{1/2} = \left(\sum_{\mathrm{i},j=1}^{2} m_{\mathrm{i}j}^2 \right)^{1/2},$$

where $M \in \mathcal{S}_2$. Let M and N be in $\mathcal{S}_2$, let us consider the order relation $M \leq N$ by

$$M\xi \cdot \xi \leq N\xi \cdot \xi, \text{ for all } \xi \in \mathbb{R}^2.$$

Finally, in the space $L^\infty(\Omega; \mathbb{R}^{2\times 2})$ we use the norm

$$\|H\|_{L^\infty(\Omega;\mathbb{R}^{2\times 2})} := \max_{1\leq \mathrm{i},j\leq 2} \|h_{\mathrm{i}j}\|_{L^\infty(\Omega)},$$

where $H \in L^\infty(\Omega; \mathbb{R}^{2\times 2})$.

Suppose that $\Omega \subset \mathbb{R}^2$ is the unit disc and $\sigma = [\sigma_{jk}]_{j,k=1}^2$ such that

$$[\sigma_{jk}] \in L^\infty(\Omega; \mathbb{R}^{2\times 2}), \quad [\sigma_{jk}]^t = [\sigma_{jk}], \quad C^{-1}I \leq [\sigma_{jk}] \leq CI$$

where $C > 0$ is a real number and the super-cript ^{t} denotes the non-conjugate matrix transpose. Let $U \in [H^1(\Omega)]^2$ be the unique solution to

$$\begin{array}{rcll} \mathrm{div}\,(\sigma \cdot \nabla U) & = & 0, & \text{in} \quad \Omega, \\ U & = & \Phi, & \text{on} \quad \partial\Omega. \end{array} \tag{6.9}$$

If σ and $\partial\Omega$ are smooth, we can define the Dirichlet-to-Neumann, or voltage-to-current, map by

$$\begin{array}{rcl} \Lambda_\sigma : [H^{1/2}(\partial\Omega)]^2 & \to & [H^{-1/2}(\partial)]^2, \\ \Phi & \mapsto & n \cdot \sigma \nabla U. \end{array}$$

Compared to the identification σ in (6.3), the problem of identifying the matrix σ not only has not been demonstrated but also it has received less attention than scalar problems. However, there are some contributions treating the following problem:

$$\begin{array}{rcll} \mathrm{div}\,(\sigma \nabla u) = \displaystyle\sum_{\mathrm{i},j=1}^{2} \frac{\partial}{\partial z_\mathrm{i}} (\sigma_{\mathrm{i}j}(z)) \frac{\partial}{\partial z_j} u & = & 0, & \text{in} \quad \Omega, \\ u & = & \phi, & \text{on} \quad \partial\Omega. \end{array} \tag{6.10}$$

Hoffmann and Sprekels in [27] proposed a dynamical system approach to reconstruct the matrix σ in equation (6.10). In [49], Rannacher and Vexler employed the finite element method and showed error estimates for a matrix identification problem from pointwise measurements of the state variable, provided that the sought matrix is constant and the exact data is smooth enough. Astala et al. [6] showed that it is possible to determine a L^∞ smooth anisotropic conductivity up to a $W^{1,2}$ diffeomorphism ϕ.

In [26], an alternative method for reconstruction to matrix coefficient is proposed, based on convex energy functional method with Tikhonov regularization.

We can consider this work as a natural extension to Calderón's problem proposed in [5] for coupled conductivity systems and for non-symmetric matrix conductivity. This paper gives the first step to extend the Calderon problem to coupled conductivity systems by Astala Päivärinta method for L^∞ non-symmetric matrix coefficients and can be extended for more complex problems, for example elasticity equation and Stoke equation. The principal idea of this problem is to reconstruct numerically the CGO solutions for the matrix conductivity case where we consider different types of conductivities which represent different types of materials, for example anisotropic conductivities. There are other works in the context to vectorial equations. In [58], Uhlmann and Wang construct CGO solutions for the isotropic elasticity system concentrated near spheres, where the domain is modeled as an inhomogeneous, isotropic, elastic medium characterized by the Lamé parameters $\lambda(x) \in C^2(\overline{\Omega})$ and $\mu \in C^4(\overline{\Omega})$. In [25] Heck et al., transform the Stoke equations to the decoupled system which is a matrix-valued Schrödinger equation.

Chapter 7

Background and state of the art

7.1 Calderón's paper

In [14], Caldeón proved that the map Q is analytic. The Fréchet derivative of Q at $q = q_0$ in the direction h is given by

$$\mathrm{d}Q\big|_{q=q_0}(h)(f,g) = \int_\Omega h\nabla u \cdot \nabla v \mathrm{d}x, \tag{7.1}$$

where $u, v \in H^1(\Omega)$ solve

$$\begin{cases} \operatorname{div}(q_0\nabla u) = \operatorname{div}(q_0\nabla v) = 0 \text{ in } \Omega \\ u\big|_{\partial\Omega} = f \in H^{\frac{1}{2}}(\partial\Omega), \quad v\big|_{\partial\Omega} = g \in H^{\frac{1}{2}}(\partial\Omega). \end{cases} \tag{7.2}$$

So the linearized map is injective if the product of $H^1(\Omega)$ solutios of $\operatorname{div}(q_0\nabla u) = 0$ is dense in $L^2(\Omega)$.

Calderón proved injectivity of the linearized map in the case where q_0 is a constant, which we assume for simplicity to be the constant function 1. The question is reduced to whether the product of gradients of harmonic functions is dense in $L^2(\Omega)$.

The harmonic functions used by Calderón was the following

$$u = \mathrm{e}^{x\cdot\rho}, \quad v = \mathrm{e}^{-x\cdot\bar{\rho}}, \tag{7.3}$$

where $\rho \in \mathbb{C}^n$ with

$$\rho \cdot \rho = 0. \tag{7.4}$$

The condition (7.4) is equivalent to the following

$$\begin{array}{l} \rho = \frac{\eta + \mathrm{i}k}{2}, \quad \eta, k \in \mathbb{R}^n, \\ |\eta| = |k|, \quad \eta \cdot k = 0. \end{array} \tag{7.5}$$

Then by plugging the solutios (7.3) into (7.1) we obtain if $\mathrm{d}Q\big|_{q_0=1}(h) = 0$

$$|k|^2(\chi_\Omega h)^\wedge(k) = 0, \forall k \in \mathbb{R}^n, \tag{7.6}$$

where χ_Ω denotes the characteristic function of Ω and $\wedge$ denotes Furier transform. Then we conclude by the Fourier inversion that $h = 0$ on Ω. However, one cannot apply the implicit function theorem to conclude that q is invertible near a constant since conditions on the range of Q that would allow use of the implicit function theorem are either false or not known.

Calderón also observed that using the solutions (7.3) one can find an approximation for the conductivity q if

$$q = 1 + h \tag{7.7}$$

and h is small enough in the L^∞ norm. Considering

$$G_q = Q_q\left(\mathrm{e}^{x\cdot\rho}\big|_{\partial\Omega}, \mathrm{e}^{x\cdot\overline{\rho}}\big|_{\partial\Omega}\right), \tag{7.8}$$

with $\rho \in \mathbb{C}^n$ as in (7.4). Now

$$G_q = \int_\Omega (1+h)\nabla u\cdot\nabla v\mathrm{d}x + \int_\Gamma h(\nabla\delta u\cdot\nabla v + \nabla u\cdot\nabla\delta v)\mathrm{d}x + \int_\Omega (1+h)\nabla\delta y\cdot\nabla\delta v\mathrm{d}x, \tag{7.9}$$

with u, v as in (7.3) and

$$\begin{gathered}\operatorname{div}(q\nabla(u+\delta v)) = \operatorname{div}(q\nabla(v+\delta v)) = 0 \text{ in } \Omega,\\ \delta u\big|_{\partial\Omega} = \delta v\big|_{\partial\Omega} = 0.\end{gathered} \tag{7.10}$$

Now standar elliptic estimates applied to (7.10) show that

$$\|\nabla\delta u\|_{L^2(\Omega)}, \|\nabla\delta v\|_{L^2(\Omega)} \le C\|h\|_{L^\infty(\Omega)}|k|\mathrm{e}^{\frac{1}{2}r|k||} \tag{7.11}$$

for fome $C > 0$ where r denotes the radius of the smallest ball containing Ω.

Plugging u, v into (7.8) we obtain

$$\widehat{\chi_\Omega q}(k) = -2\frac{G_q}{|k|^2} + R(k) = \widehat{F}(k) + R(k), \tag{7.12}$$

where F is determined by G_q and therefore this is known. Using (7.11), we can show that $R(k)$ satisfies the estimate

$$|R(k)| \le C\|h\|^2_{L^\infty(\Omega)}\mathrm{e}^{r|k|}. \tag{7.13}$$

In other words we know $\widehat{\chi_\Omega q}(k)$ up to a term that is small for k, where k is small enough. More precisely, let $1 < \alpha < 2$. Then for

$$|k| \le \frac{2-\alpha}{t}log\left(\frac{1}{\|h\|^\infty_L}\right) = \beta \tag{7.14}$$

we have

$$|R(k)| \le C\|h\|^\alpha_{L^\infty(\Omega)} \tag{7.15}$$

for some $C > 0$.

We take $\widehat{\eta}$ a C^∞ cut-off so that $\widehat{\eta}(0) = 1$, $\operatorname{supp}\widehat{\eta}(k) \subset \{k \in \mathbb{R}^n, |k| \le 1\}$ and $\eta_\beta(x) = \beta^n\eta(\beta x)$. Then we obtain

$$\widehat{\chi_\Omega q}(k)\widehat{\eta}\left(\frac{k}{\beta}\right) = \frac{-2G_q q}{|k|^2}\widehat{\eta}\left(\frac{k}{\beta}\right) + R(k)\widehat{k}\left(\frac{k}{\beta}\right). \tag{7.16}$$

Using this we get the following estimate

$$|l(x)\| \leq C\|h\|^{\alpha}_{L^\infty(\Omega)} \left(log \left(\frac{1}{\|h\|_{L^\infty}} \right) \right)^n , \tag{7.17}$$

where $l(x) = (\chi_\Omega q * \eta_\beta)(x) - (F * \eta_\beta)(x)$. Formula (7.17) gives then an approximation to the smoothed out conductivity, $\chi_\Omega q * \eta_\beta$, for h sufficiently small.

7.2 Complex geometrical optics solutions with a linear phase

Motivated by Calderón exponential solutions, Sylvester and Uhlmann [55, 56] constructed in dimension $n \geq 2$ CGO solutions of the conductivity equation for C^2 conductivities that behave like Calderón exponential solutions for large frequencies. This can be reduced to constructing solutions in the whole space for Schrödinger equation with potential.

Let $q \in C^2(\mathbb{R}^n)$, q strictly positive in $\mathbb{R}^n$ and $q = 1$ for $|x| \geq R$, $R > 0$. Let $L_q u = \operatorname{div}(q\nabla u)$. Then we have

$$q^{-\frac{1}{2}} L_q (q^{-\frac{1}{2}}) = \Delta - \gamma \tag{7.18}$$

where

$$\gamma = \frac{\Delta\sqrt{q}}{\sqrt{q}}. \tag{7.19}$$

Therefore, to construct solutions of $L_q u = 0$ in $\mathbb{R}^n$ it is enough to construct solutions of the Schödinger equation $(\Delta - \gamma)u = 0$ with γ of the form (7.19). The next result proved in [55, 56] states the existence of CGO solutions for the Schödinger equation associated to any bounded and compactly supported potential.

Theorem 7.1 *Let $\gamma \in L^\infty(\mathbb{R}^n)$, $n \geq 2$ with $q(x) = 0$ for $|x\| \geq R > 0$. Let $-1 < \delta < 0$. There exists $\varepsilon(\delta)$ and such that for every $\rho \in \mathbb{C}^n$ satisfying*

$$\rho \cdot \rho = 0 \tag{7.20}$$

and

$$\frac{\|(1 + |x|^2)^{1/2}\gamma\|_{L^\infty(\mathbb{R}^n)} + 1}{|\rho|} \leq \varepsilon \tag{7.21}$$

there exists a unique solution to

$$(\Delta - \gamma)u = 0 \tag{7.22}$$

of the form

$$u = \mathrm{e}^{x\cdot\rho}(1 + \psi_\gamma(x, \rho)) \tag{7.23}$$

with $\phi_\gamma(\cdot, \rho) \in L^2_\delta(\mathbb{R}^m)$. Moreover $\psi_\gamma(\cdot, \rho) \in H^2_\delta(\mathbb{R}^n)$ and for $0 \leq s \leq 2$ there exists $C = C(n, s, \delta) > 0$ such that

$$\|\psi_\gamma(\cdot, \rho)\|_{H^s_{delta}} \leq \frac{C}{|\rho|^{1-s}}. \tag{7.24}$$

Here

$$L^2_\delta(\mathbb{R}^n) = \left\{ f : \int (1+|x|^2)^\delta |f(x)|^2 \mathrm{d}x < \infty \right\}, \tag{7.25}$$

with the norm given by $\|f\|^2_{L^2_\delta} = \int (1+|x|^2)^\delta |f(x)|^2 \mathrm{d}x$ and $H^m_\delta(\mathbb{R}^n$ denotes the corresponding Sobolev space. Note that for large $|\rho|$ these solutions behave like Calderón's exponential solutions $\mathrm{e}^{x\cdot\rho}$.

7.3 The Calderón Problem in two dimensions

Astala and Päivärinta, in [5], have extended significantly the uniqueness result of [45] for conductivities having two derivatives in an appropiate sense and the result of [13] for conductivities having one derivatives in appropiate sense, by proving that any L^∞ conductvity in two dimensions can be determined uniquely form the Dicichlet to Neumann map. The $\overline{\partial}$ method, introduced in [8], has been used in numerical reconstruction procedures in two dimensions in [30, 43].

The proof of [5] relies also on contruction of CGO solutions for the conductivity equation with L^∞ coefficients and the $\overline{\partial}$ method. This is done by transforming the conductivity equation to a quasi-regular map. let $\mathcal{D}$ be the unit disk in the plane. Then we have

Lemma 7.2 *Assume $u \in H^1(\mathcal{D})$ is a real valued and satisfies the conductivity equation on $\mathcal{D}$. Then there exists a function $v \in H^1(\mathcal{D})$, unique up to a constant, such that $f = u + \mathrm{i}v$ satisfies the Beltrami equation*

$$\overline{\partial} f = \mu \overline{\partial f}, \tag{7.26}$$

where $\mu = (1-q)/(1+q)$.

Conversely, if $f \in H^1(\mathcal{D})$ satisfies (7.26) with a real-valued μ, then $u = Re(f)$ and $v = Im(f)$ satisfy

$$\mathrm{div}(q\nabla u) = 0 \textit{ and } \mathrm{div}\left(\frac{1}{q}\nabla v\right) = 0, \tag{7.27}$$

respectively, where $q = (1-\mu)((1+\mu)$.

Let us denote $\kappa = \|\mu\|_{L^\infty} < 1$. Then (7.26) means that f is a quasi-regular map. The function v is called the q-harmonic conjugate of u and it is unique up to a constant.

Astala and Päivärinta consider the μ-Hilbert transform $\mathcal{H}_\mu : H^{1/2}(\partial\Omega) \to H^{1/2}(\partial\Omega)$ that is defined by

$$\mathcal{H}_\mu : u\big|_{\partial\Omega} \mapsto v\big|_{\partial\Omega} \tag{7.28}$$

and show that the DN map Λ_q determines $\mathcal{H}_\mu$ and vice versa.

Below we use the complex notation $z = x_1 + \mathrm{i}x_2$. Moreover, for the equation (7.26), it is shown that for every $k \in \mathbb{C}$ there are CGO solutions of the Beltrami equation that have the form

$$f_\mu(z,k) = \mathrm{e}^{\mathrm{i}kz} M_\mu(z,k), \tag{7.29}$$

where

$$M_\mu(z,k) = 1 + \mathcal{O}\left(\frac{1}{z}\right) \quad \text{as } |z| \to \infty. \tag{7.30}$$

More precisely, they prove that

Theorem 7.3 *For each $k \in \mathbb{C}$ and for each $2 < p < 1 + 1/\kappa$ the equation (7.26) admits a unique solution $f \in W^{1,p}_{loc}(\mathbb{C})$ of the form (7.29) such that the asymptotic formula (7.30) holds true.*

In the case of non-smooth coefficients the function M_μ grows sub-exponentially in k. Astala and Päivärinta introduce the transport matrix to deal with this problem and they show that this matrix is determined by the Hilbert transform H_μ and therefore by the Dicichlet to Neumann map. Then they use the transport matrix to show that Λ_q determines uniquely q.

Chapter 8

Construction of CGO of conductivity systems

8.1 Existence of CGO solution

In this work we identify $\mathbb{R}^2$ and $\mathbb{C}$ by the map $(x_1, x_2) \mapsto x_1 + \mathrm{i}x_2$ and denote $z = x_1 + \mathrm{i}x_2$, with i satisfying $\mathrm{i}^2 = -1$. We use the standar notations:

$$\begin{aligned} \partial = \partial_z = \tfrac{1}{2}\left(\partial_1 - \mathrm{i}\partial_2\right) \\ \overline{\partial} = \partial_{\overline{z}} = \tfrac{1}{2}\left(\partial_1 + \mathrm{i}\partial_2\right) \end{aligned}$$

where $\partial_{\mathrm{i}} = \dfrac{\partial}{\partial x_j}$, $j = 1, 2$. We will consider $\Omega \subset \mathbb{R}^2$ to be the unit disc and $\sigma \in L^\infty(\Omega; \mathbb{R}^{2\times 2})$ a symmetric, positive definite matrix.

The aim of this work is, following [5], for $k \in \mathbb{C}$, compute numerically a unique solution U_1 and U_2 to

$$\begin{aligned} \mathrm{di}v\left(\sigma(z) \cdot \nabla U_1(z,k)\right) &= 0, \\ \mathrm{di}v\left(\sigma(z)^{-1} \cdot \nabla U_2(z,k)\right) &= 0, \end{aligned} \tag{8.1}$$

where U_1 and U_2 have asymptotic behavier similar to (6.2). The way to get U_1 and U_2 is considering solutions to the Beltramy system

$$\overline{\partial} F = \mu \overline{\partial F}, \tag{8.2}$$

where

$$\mu(z) = \left(I - \sigma(z)\right)\left(I + \sigma(z)\right)^{-1}. \tag{8.3}$$

Here, the connection between the equations (6.9) and (8.2) is given by the following Lemma:

Lemma 8.1 *Suppose $U \in [H^1(\Omega)]^2$ satisfies the equation (6.9). Then there exists a function $V \in [H^1(\Omega)]^2$ such that $F = U + \mathrm{i}V$ satisfies the Beltrami system (8.2), where μ is defined* *by (8.3). Conversely, if $F \in [H^1(\Omega)]^2$ satisfies (8.2) with $\mu \in L^\infty(\Omega;\mathbb{R}^{2\times 2})$, then $U = Re(F)$ and $V = Im(F)$ satisfy*

$$\begin{aligned} \mathrm{div}(\sigma \cdot \nabla U) = 0 \\ \mathrm{div}(\sigma^{-1} \cdot \nabla V) = 0, \end{aligned}$$

respectively, where

$$\sigma = (I + \mu)^{-1}(I - \mu).$$

Proof. Let us denote $W = (-\sigma\partial_2 U, \sigma\partial_1 U)$. By (6.9),

$$\begin{aligned} 0 &= \mathrm{div}(\sigma \cdot \nabla U), \\ &= \mathrm{div}(\sigma(\partial_1 U, \partial_2 U)), \\ &= \partial_1(\sigma\partial_1 U) + \partial_2(\sigma\partial_2 U), \end{aligned}$$

then $\partial_2 W_1 = \partial_1 W_2$.

Therefore there exists $V \in [H^1(\mathbb{D})]^2$ such that

$$\left.\begin{array}{rcl} \partial_1 V &=& -\sigma\partial_2 U, \\ \partial_2 V &=& \sigma\partial_1 U. \end{array}\right\}$$

Then a straightforward calculation prove the equivalence of the two equations. $\square$

Remark 8.2 *We note that the condition for σ and μ implies the existence of a constant $0 \le \kappa < 1$ such that*

$$\|\mu\|_{L^\infty(\Omega;\mathbb{R}^{2\times 2})} \le \kappa,$$

holds for almost very $z \in \mathbb{C}$ and for $\sigma \in L^\infty(\Omega;\mathbb{R}^{2\times 2})$.

In the same way that for the scalar case, the Beltrami system (8.2) and its solutios are governed and controlled by the extension of two basic linear operators, the Cauchy transform and the Beurling transform.

The Cauchy transform is extended for the vectorial case by

$$\mathbb{P}G(z) = \left(\begin{array}{c} PG_1(z) \\ PG_2(z) \end{array} \right),$$

where the scalar Cauchy transform is defined by

$$Pg(z) = -\frac{1}{\pi}\int_{\mathbb{C}} \frac{g(\omega)}{\omega - z} \mathrm{d}m(\omega).$$

Remark 8.3 *The vectorial Cauchy transform $\mathbb{P}$ acts as the inverse operator to $\overline{\partial}$, i.e., $\mathbb{P}\overline{\partial}G = \overline{\partial}\mathbb{P}G = G$ for $G \in [C_0^\infty(\mathbb{C})]^2$.*

The operator P has some properties in an appropiate Lebesgue, Sobolev and Lipschitz space (see [59]) and can be easily extended to the operator $\mathbb{P}$. If we denote

$$L^p(\Omega) = \left\{ G \in [L^p(\mathbb{C})]^2 \mid G\big|_{\mathbb{C}\setminus\Omega} \equiv 0 \right\},$$

we have the following properties.

Proposition 8.4 *Let $\Omega \subset \mathbb{C}$ be a bounded domain and let $1 < q < 2$ and $2 < p < \infty$. Then*

1. $\mathbb{P} : [L^p(\mathbb{C})]^2 \to [Lip_\alpha(\mathbb{C})]^2$, *where* $\alpha = 1 - \frac{2}{p}$.
2. $\mathbb{P} : [L^p(\Omega)]^2 \to [W^{1,p}(\mathbb{C})]^2$ *is bounded.*
3. $\mathbb{P} : [L^p(\Omega)]^2 \to [L^p(\mathbb{C})]^2$ *is compact.*
4. $\mathbb{P} : [L^p(\mathbb{C}) \cap L^q(\mathbb{C})]^2 \to [C_0(\mathbb{C})]^2$ *is bounded, where C_0 is the closure of C_0^∞ in L^∞.*

On the other hand, the Beurling transform is also extended for the vectorial case by

$$\mathbb{S}G(z) = \begin{pmatrix} SG_1(z) \\ SG_2(z) \end{pmatrix} = \begin{pmatrix} \partial PG_1(z) \\ \partial PG_2(z) \end{pmatrix} = \partial \mathbb{P} G(z),$$

where S is determined as a principal-value integral

$$Sg(z) = -\frac{1}{\pi} \int_{\mathbb{C}} \frac{g(\omega)}{(\omega - z)^2} \mathrm{d}m(\omega).$$

To undertand the mapping properties of $\mathbb{P}$ and the invertibility of the operator $I - \mu\overline{\mathbb{S}}$ on some appropriate spaces, the following proposition is very usefull, where $\overline{\mathbb{S}}$ denotes the operator $\overline{\mathbb{S}}(G) = \overline{\mathbb{S}(G)}$.

Proposition 8.5 *Let $M, N \in L^\infty(\mathbb{C}; \mathbb{C}^{2\times 2})$ such that*

$$2 \left(\|M\|_{L^\infty(\mathbb{C};\mathbb{C}^{2\times 2})} + \|N\|_{L^\infty(\mathbb{C};\mathbb{C}^{2\times 2})} \right) \leq \kappa,$$

holds for almost every $z \in \mathbb{C}$ with a constant $0 \leq \kappa < 1$. Suppose that $1 + \kappa < p < 1 + 1/\kappa$. Then the operator

$$\mathcal{B} = I - M\mathbb{S} - N\overline{\mathbb{S}}$$

is bounden and invertible in $[L^2(\mathbb{C})]^2$, where the norm of $\mathcal{B}$ and $\mathcal{B}^{-1}$ are bounded by constants depending only on κ and p.

Moreover, the bound in p is sharp; for each $p \leq 1 + \kappa$ and for each $p \geq 1 + 1/\kappa$ there are M_1 and M_2, as above, such that $\mathcal{B}$ is not invertible in $[L^p(\mathbb{C})]^2$.

Proof. If we consider

$$\begin{aligned}
\mathcal{B} &= I - M\mathbb{S} - N\overline{\mathbb{S}} \\
&= I - M\begin{pmatrix} S \\ S \end{pmatrix} - N\begin{pmatrix} \overline{S} \\ \overline{S} \end{pmatrix} \\
&= I - \begin{pmatrix} (m_{11}+m_{12})S \\ (m_{21}+m_{22})S \end{pmatrix} - \begin{pmatrix} (n_{11}+n_{12})\overline{S} \\ (n_{21}+n_{22})\overline{S} \end{pmatrix} \\
&= \begin{pmatrix} I - (m_{11}+m_{12})S - (n_{11}+n_{12})\overline{S} \\ I - (m_{21}+m_{22})S - (n_{21}+n_{22})\overline{S} \end{pmatrix},
\end{aligned}$$

where every component is bounded and invertible in $L^p(\mathbb{C})$ (see [3]) and then the result holds. $\square$

We need the following useful proposition as well, which is a natural extension of the result shown in [5] and [59].

Proposition 8.6 *Let $\mathcal{F} = (F_1, F_2) \in [W^{1,p}_{loc}]^2$ and $\Gamma \in L^p_{loc}(\mathbb{C})$ for some $p > 2$. Suppose that for some constant $0 \le \kappa < 1$,*

$$|\overline{\partial} F_{\mathrm{i}}(z)| \le \kappa |\partial F_{\mathrm{i}}(z)| + \Gamma(z)|F_{\mathrm{i}}|, \quad \mathrm{i} = 1, 2,$$

holds for almost every $z \in \mathbb{C}$. Then, if $\mathcal{F}(z) \to 0$ as $|z| \to \infty$ and Γ has a compact support then

$$\mathcal{F}(z) \equiv 0.$$

Now, we can establish the existence of the CGO solutions to (8.2) of the form

$$F_\mu(z,k) = \mathrm{e}^{\mathrm{i}kz}\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix} + \mathcal{N}(z)\right) \tag{8.4}$$

where $\mathcal{N} = \begin{pmatrix} \mathcal{N}_1 \\ \mathcal{N}_2 \end{pmatrix}$ and

$$\mathcal{N}_{\mathrm{i}}(z,k) = \mathcal{O}\left(\frac{1}{z}\right), \text{ as } |z| \to \infty, \text{ for } \mathrm{i} = 1, 2. \tag{8.5}$$

In order to establish the existence and shape (8.4) of CGO solutions, we begin with the following proposition.

Proposition 8.7 *Suppose that $2 < p < 1 + 1/\kappa$, $\alpha \in L^\infty(\mathbb{C}; \mathbb{R}^{2\times 2})$ with $\mathrm{supp}(\alpha) \subset \Omega$ and $\|\nu\|_{L^\infty(\mathbb{C};\mathbb{R}^{2\times 2})} \le \kappa \chi_\Omega(z)$ for almost every $z \in \Omega$. Define the operator $\mathcal{K} : [L^p(\mathbb{C})]^2 \to [L^p(\mathbb{C})]^2$ by*

$$\mathcal{K}G = \mathbb{P}\left(I - \nu\mathbb{S}\right)^{-1}\left(\alpha\overline{G}\right).$$

Then $\mathcal{K} : [L^p(\mathbb{C})]^2 \to [W^{1,p}(\mathbb{C})]^2$ and $I - \mathcal{K}$ is invertible in $[L^p(\mathbb{C})]^2$.

Proof. First, since $\|\nu\|_{L^\infty(\mathbb{C};\mathbb{R}^{2\times 2})} \leq \kappa\chi_\Omega(z)$, by Proposition 8.5, we have that $I - \nu\mathcal{S}$ is invertible in L^p and, by Proposition 8.4, the operator $\mathcal{K} : [L^p(\mathbb{C})]^2 \to [L^p(\mathbb{C})]^2$ is well-defined and compact. We also have supp $\left(I - \nu\mathbb{S}\right)\left(\alpha\overline{G}\right) \subset \Omega$.

Finally, to prove the invertibility of $I - \mathcal{K}$ in $[L^p(\mathbb{C})]^2$, we use the Fredholm's alternative. For this, let us prove that $I - \mathcal{K}$ is injective in $[L^p(\mathbb{C})]^2$.

Let us suppose that $G = \begin{pmatrix} G_1 \\ G_2 \end{pmatrix} \in [L^p(\mathbb{C})]^2$ satisfying

$$G = \mathbb{P}\left(\left(I - \nu\mathbb{S}\right)^{-1}\left(\alpha\overline{G}\right)\right),$$

by Proposition 8.4, we have that $G \in [W^{1,p}(\mathbb{C})]^2$ and thus

$$\overline{\partial}G = \left(I - \nu\mathbb{S}\right)^{-1}\left(\alpha\overline{G}\right),$$

which is equivalent to

$$\overline{\partial}G - \nu\overline{\partial G} = \alpha\overline{G}. \tag{8.6}$$

Finally, from (8.6) we can conclude that $\overline{\partial}G = 0$ outside Ω, and therefore G is analytic. Then this combined with $G \in [L^p(\mathbb{C})]^2$ implies that

$$G_{\mathrm{i}}(z) = \mathcal{O}\left(\frac{1}{z}\right), \text{ for } |z| \to \infty, \text{ for i} = 1, 2.$$

Thus, the assumptions of Proposition 8.6 are fulfilled and we must have $G \equiv 0$. □

Finally, the following theorem establishes the existence of the Complex Geometric Optics solutions to the Beltrami system (8.2).

Theorem 8.8 *For each $k \in \mathbb{C}$ and for each $2 < p < 1 + 1/\kappa$ the system (8.2) admits a unique solution $F \in [W^{1,p}_{loc}(\mathbb{C})]^2$ of the form (8.4) such that the asymtotic formula (8.5) holds true.*

Proof. If we write

$$F_\mu(z,k) = \mathrm{e}^{\mathrm{i}kz}\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix} + \mathcal{N}(z)\right), \tag{8.7}$$

and plug this into the Beltrami system (8.2) we obtain

$$\overline{\partial}\mathcal{N} - \mathrm{e}_{-k}\mu\overline{\partial\mathcal{N}} \;=\; \alpha\overline{\mathcal{N}} + \alpha\begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

where

$$\begin{aligned} \mathrm{e}_{-k}(z) &= \mathrm{e}^{-\mathrm{i}\left(kz+\overline{k}\overline{z}\right)}. \\ \alpha(z) &= -\mathrm{i}\overline{k}\mathrm{e}_{-k}(z)\mu(z). \end{aligned} \tag{8.8}$$

Since $\overline{\mathbb{S}}\left(\overline{\partial} G\right) = \overline{\partial G}$, we obtain

$$\overline{\partial}\mathcal{N} = \left(I - \mathrm{e}_{-k}\mu\overline{\mathbb{S}}\right)^{-1}\left(\alpha\overline{\mathcal{N}} + \alpha\begin{pmatrix}1\\1\end{pmatrix}\right).$$

If now, $\mathcal{K}$ is defined, as in the Proposition 8.7, with $\nu = \mathrm{e}_{-k}\mu$ we get

$$\mathcal{N} - \mathcal{K}\mathcal{N} = \mathcal{K}\left(\chi_\Omega\right) \in [L^p(\mathbb{C})]^2. \tag{8.9}$$

Since $I - \mathcal{K}$ is invertible in $[L^p(\mathbb{C})]^2$, and $\mathcal{N}$ is analytic in $\mathbb{C} \setminus \Omega$ the result holds. □

Finally, by the Theorem 8.8, the Complex Geometrical Optics solutions F_μ are given by substituting the unique solution of equation (8.9) by the formula (8.7).

8.2 Compute the CGO solutions

Once the existence of the CGO solution of (8.2) has been proved, we want to calculate this solution F_μ numerically. For the scalar case, the numerical computation of CGO was introduced in [4], based on the original contruction in [5]. In [4], the difficulty of the numerical computation of CGO solutions is the lack of complex-linearity in the equation that was compensated by keeping the real and imaginary parts of the solution separately in a real-linear solution process. To amend this Huhntanen and Perämäki introduced in [28] an efficient method for the computation of the CGO solutions. Let us describe this method below:

First, by Theorem 8.8 we know that the function $\mathcal{N}$ satisfies the equation

$$\overline{\partial}\mathcal{N} - \nu\overline{\partial\mathcal{N}} - \alpha\overline{\mathcal{N}} - \alpha\begin{pmatrix}1\\1\end{pmatrix} = 0. \tag{8.10}$$

Define $U \in [L^p(\Omega)]^2$ by $\overline{U} = -\overline{\partial}\mathcal{N}$. Then $\mathcal{N} = -\mathbb{P}\overline{U}$ and $\partial\mathcal{N} = -\mathbb{S}U$. Substituting U into (8.10) leads to the real-linear integral equation

$$-\overline{U} - \nu\overline{\left(-\mathbb{S}\overline{U}\right)} - \alpha\overline{\left(-\mathbb{P}\overline{U}\right)} = \alpha\begin{pmatrix}1\\1\end{pmatrix}.$$

Then

$$U + \left(-\overline{\nu}\mathbb{S} - \overline{\alpha}\mathbb{P}\right)\overline{U} = -\overline{\alpha}\begin{pmatrix}1\\1\end{pmatrix}. \tag{8.11}$$

Let us denote the complex conjugate $\overline{G}$ of an operator G as $\overline{G} = \rho(G)$, then (8.11) takes the form

$$\left(I + A\rho\right)U = -\overline{\alpha}\begin{pmatrix}1\\1\end{pmatrix}, \tag{8.12}$$

where $A := \left(-\overline{\nu}\mathbb{S} - \overline{\alpha}\mathbb{P}\right)$.

The operator $I+A$ is invertible in $[L^p(\Omega)]^2$. Indeed, note that, by Proposition 8.5, $I-\overline{\nu}\mathbb{S}\rho$ is invertible. Hence the equation

$$\left(I-(I-\overline{\nu}\mathbb{S}\rho)^{-1}(\overline{\alpha}\mathbb{P}\rho)\right)U=(I-\overline{\nu}\mathbb{S}\rho)^{-1}\left(-\overline{\alpha}\begin{pmatrix}1\\1\end{pmatrix}\right)$$

is equivalent to (8.12). The operator on the left-hand side is invertible by the fact that its null space is trivial and $(I-\overline{\nu}\mathbb{S}\rho)^{-1}(\overline{\alpha}\mathbb{P}\rho)$ is compact. Therefore, $I+A$ is invertible as well.

A special precoditioning step is first time introduced in [28], it consists of the transformation of the real-linear equation (8.12) into a complex-linear equation allowing standar iterative solution by GMRES. Consider the following equation in the space $[L^p(\Omega)]^2$:

$$\left(I-A\overline{A}\right)V=-\overline{\alpha}\begin{pmatrix}1\\1\end{pmatrix}, \tag{8.13}$$

where $\overline{A}V=\overline{A\overline{V}}$. Now (8.13) is complex-linear, and the solution U of (8.12) can be written as $U=(I-A\rho)V$.

Summarizing, the computation of the function $\mathcal{N}(z,k)$ defined in (8.9) for a given $k\in\mathbb{C}$ proceeds as follows:

Algorithm 3 Solution of $\mathcal{N}_\mu$

```
1: return  Solution of the function $\mathcal{N}(z,k)$.
2: Given $k, \alpha, \mu$ ,Where $k \in \mathbb{C}$, $\alpha$ is compute from (8.8) , $\mu$ is compute from(8.3).
3: Solve for $V$ from (8.13). Note that $V$ is supported in $\Omega$.
4: Calculate $U = (I - A\rho) V$. Note that $U$ is supported in $\Omega$.
5: Compute $\mathcal{N} = -\mathcal{P}\overline{U}$.
6: $\mathcal{N}_\mu = \mathcal{N}$.
```

8.3 Reduction to a periodic integral equation and discretization

As shown in [28] and discussed in the previous section, the computation of CGO solution to the real-linear Beltrami equation can be reduced to the solution of the complex-linear equation (8.13). Furthermore, one can use the iterative GMRES method for the solution of periodized and discretized version of (8.13). To that end, followong [4], we need to introduce a periodic version of the operator $A:=(-\overline{\nu}\mathcal{S}-\overline{\alpha}\mathcal{P})$.

Take $s>2$ and define a square $Q\subset\mathbb{R}^2$ by

$$Q:=\left\{(x,y)\in\mathbb{R}^2\middle|-s\le x<s,-s\le y<s\right\}.$$

We consider tiling of the plane by translated copies of Q and work with 2s-periodic functions $f:\mathbb{R}^2\to\mathbb{C}$ satisfying

$$\tilde{f}(x+2j_1s,y+2j_2s)=\tilde{f}(x,y),\ \text{for } j_1,j_2\in\mathbb{Z},$$

where we indicate 2s-periodic functions adding $\tilde{\cdot}$ on top of symbols.

Choose a smooth cutoff function η satisfying

$$\eta(z) = \begin{cases} 1, & \text{for} \quad |z| \leq 2, \\ 0, & \text{for} \quad |z| \geq 2 + (s-2)/2, \end{cases} \tag{8.14}$$

and $0 \leq \eta(z) \leq 1$ for all $z \in \mathbb{C}$. Define a $2s$-periodic approximate Green's function $\tilde{g}$ for the D-bar operator by setting it to $\eta(z)/(\pi z)$ inside Q and extending periodically by

$$\tilde{g}(z + 2j_1 s + \mathrm{i}2j_2 s) = \frac{\eta(z)}{\pi z}$$

for $z \in Q \setminus 0$ and $j_1, j_2 \in \mathbb{Z}$. Define a periodic approximate Cauchy transform by

$$\tilde{P}f(z) := (\tilde{g} \tilde{*} f)(z) = \int_Q \tilde{g}(z-w) f(w) \mathrm{d}w_1 \mathrm{d}w_2, \tag{8.15}$$

where $\tilde{*}$ denotes convolution on the torus.

The Beurling transform is approximated in the periodic context by writing

$$\tilde{\beta}(z + 2j_1 s + \mathrm{i}2j_2 s) = \frac{\eta(z)}{\pi z^2}$$

for $z \in Q \setminus 0$ and $j_1, j_2 \in \mathbb{Z}$, and defining

$$\tilde{S}g(z) := \left(\tilde{\beta} \tilde{*} g\right)(z) = \int_Q \tilde{\beta}(z-w) g(w) \mathrm{d}w_1 \mathrm{d}w_2. \tag{8.16}$$

Then the extentions of the periodic Cauchy transform $\tilde{P}$ and periodic Beurling transform $\tilde{S}$ is defined by

$$\tilde{\mathbb{P}} = \begin{pmatrix} \tilde{P} \\ \tilde{P} \end{pmatrix}, \quad \tilde{\mathbb{S}} = \begin{pmatrix} \tilde{S} \\ \tilde{S} \end{pmatrix}.$$

Set $\tilde{A} := \left(-\overline{\tilde{\nu}}\tilde{\mathbb{S}} - \overline{\tilde{\alpha}}\tilde{\mathbb{P}}\right)$ with the functions $\tilde{\alpha}$ and $\tilde{\nu}$ being trivial periodic extensions of functions defined in (8.8), α and ν, which are both supported in the unit disc. The periodic version of (8.13) takes the form

$$\left(I - \tilde{A}\overline{\tilde{A}}\right)\tilde{V} = \overline{\tilde{\alpha}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{8.17}$$

Now, for discretization, choose a positive integer m, denote $M = 2^m$, and set $h = 2s/M$. Define a grid $\mathcal{G}_m \subset Q$ by

$$\mathcal{G}_m = \left\{ jh \middle| j \in \mathbb{Z}_m^2 \right\},$$
$$\mathbb{Z}_m^2 = \left\{ j = (j_1, j_2) \in \mathbb{Z}^2 \middle| -2^{m-1} \leq j_l < 2^{m-1}, l = 1, 2 \right\}.$$

Note that the number of points in $\mathcal{G}_m$ is M^2. Define the grid approximation $\varphi_h : \mathbb{Z}_m^2 \to \mathbb{C}$ of a function $\varphi : Q \to \mathbb{C}$ by

$$\varphi_h(j) = \varphi(jh).$$

Our strategy is to use the iterative GMRES method for the solution of the discretized version of the periodic equation (8.17). To that end, we need to discretize the periodic Cauchy and Beurling transforms defined in (8.15) abd (8.16), respectively.

Set

$$\tilde{g}_h(j) = \begin{cases} \tilde{g}(jh) & \text{for} \quad j \in \mathbb{Z}_m^2 \setminus 0, \\ 0 & \text{for} \quad j = 0, \end{cases}$$

and

$$\tilde{\beta}_h(j) = \begin{cases} \tilde{\beta}(jh) & \text{for} \quad j \in \mathbb{Z}_m^2 \setminus 0, \\ 0 & \text{for} \quad j = 0, \end{cases}$$

where the point $jh \in \mathbb{R}^2$ is interpreted as the complex number $hj_1 + \mathrm{i}hj_2$. Now $\tilde{g}_h$ and $\tilde{\beta}_h$ are $M \times M$ matrices with complex entries. Given a periodic function φ, the discrete transforms $\tilde{P}\varphi$ are defined by

$$\begin{aligned} \left(\tilde{P}\varphi_h\right)_h &= h^2 \texttt{IFFT}\left(\texttt{FFT}(\tilde{g}_h) \cdot \texttt{FFT}(\varphi_h)\right), \\ \left(\tilde{S}\varphi_h\right)_h &= h^2 \texttt{IFFT}\left(\texttt{FFT}(\tilde{\beta}_h) \cdot \texttt{FFT}(\varphi_h)\right), \end{aligned}$$

and all the ingredients for the numerical solution are in place.

Chapter 9

Numerical results

In this section we assess the ability of the proposed approach for obtaining numerical approximation on several contexts. We will take strictly positive conductivities $\sigma_{lm} : \Omega \to \mathbb{R}$ that models an idealized cross-section of human chest. Since the Astala-Päivärinta theory is developed for nonsmooth conductivities $\sigma \in L^{\infty}(\Omega)$ we will consider the background conductivity has the value one and the conductive heart and resistive lungs are separated from the background by a discontinuity.

For the first test, based on the example in [4, 42], let us consider $\sigma_{lm}(z) = \sigma(z)$, when the conductivity of the heart is 2 an the conductivity of the lungs is 0.7, see Figure 9.1 for a graphical representation of the conductivity. In this example we consider that the two equations act separately, that is, they are not coupled, the equation is the following

$$\mathrm{div}\left(\sigma(z)\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\cdot\nabla U\right) \quad = \quad 0, \quad \text{in} \quad \Omega,$$

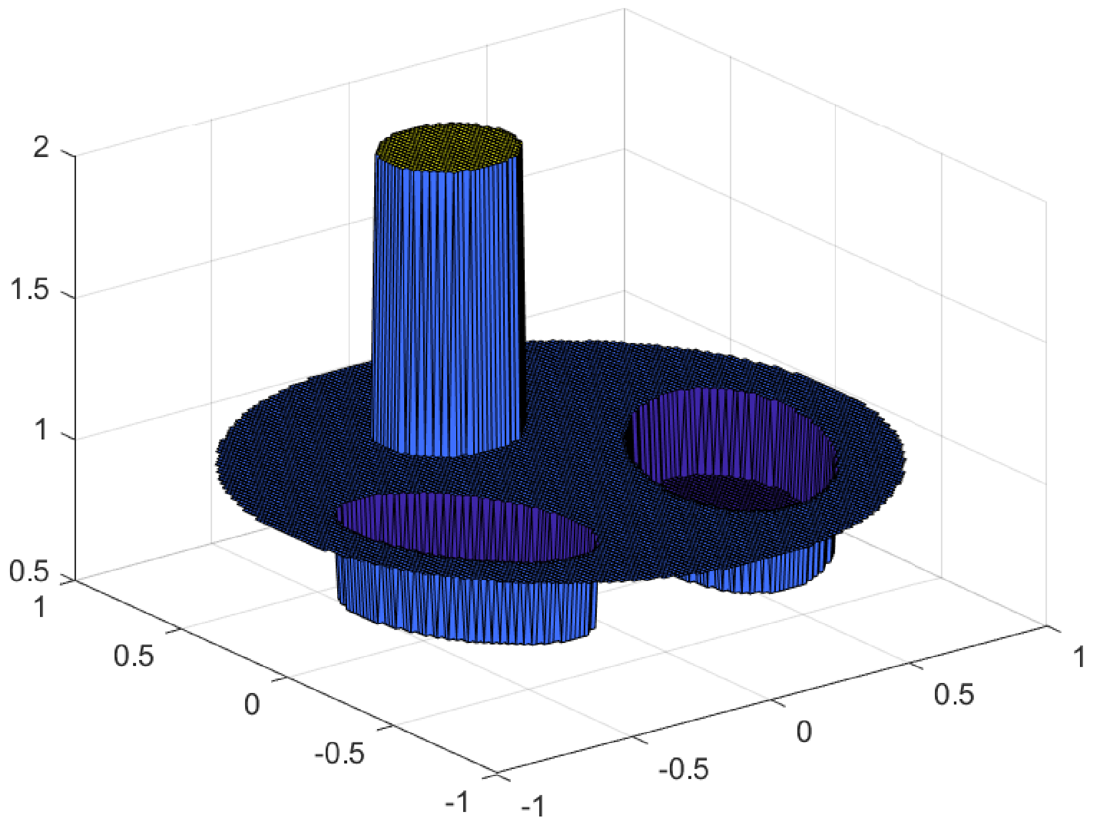
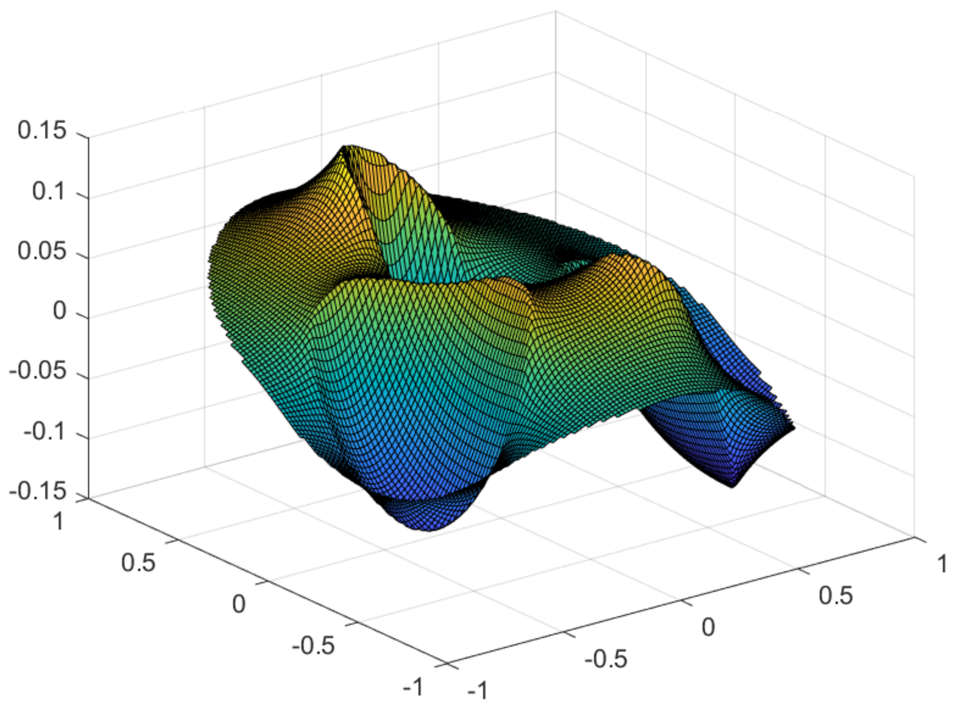
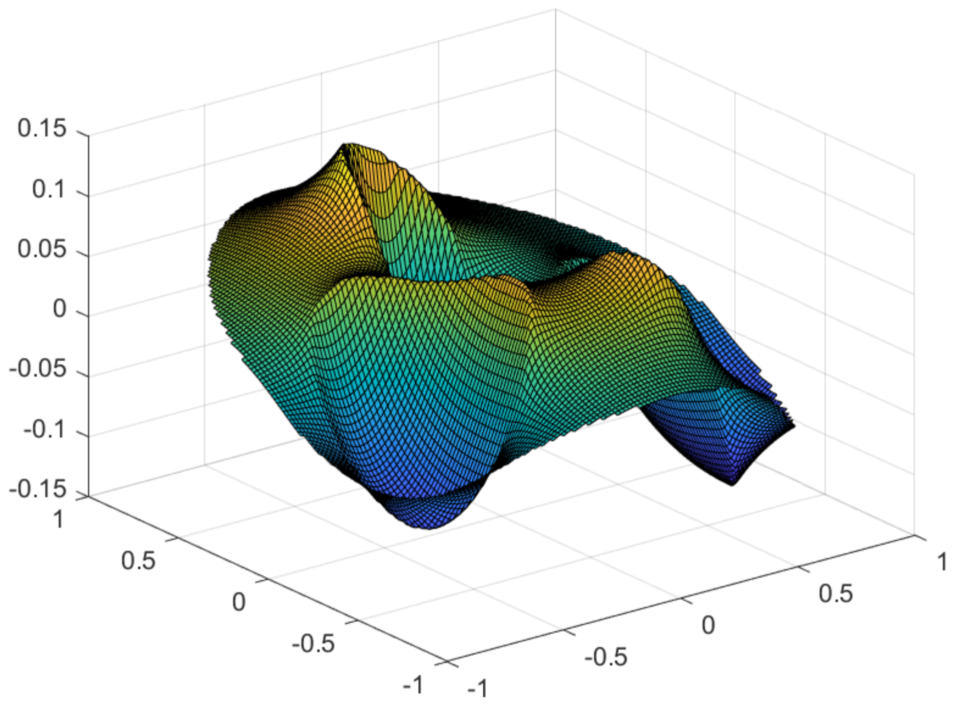


Figure 9.1: Three-dimensional mesh plot of the discontinuous conductivity σ.

In the Figure 9.2 it can be seen that the two equations act independently with the same conductivity, obtaining the same result for the CGO solution for $\mathcal{N}_1$ and $\mathcal{N}_2$. It is possible to observe that the solutions in the Figure 9.2 are equivalent to the solutions shown in the Figure 14.12 in [42]. It's possible to observe that our procedure solves the fully vectorial expression and recover the scalar case for the diagonal matrices, then our procedure is an extension of the scalar algorithm.



(a) Real part of $\mathcal{N}_1(z, 2)$.



(b) Real part of $\mathcal{N}_2(z, 2)$.

Figure 9.2: Three-dimensional mesh plot of the CGO solutions corresponding to the nonsmooth conductivity σ. Here $k = 2$ and $m = 8$.

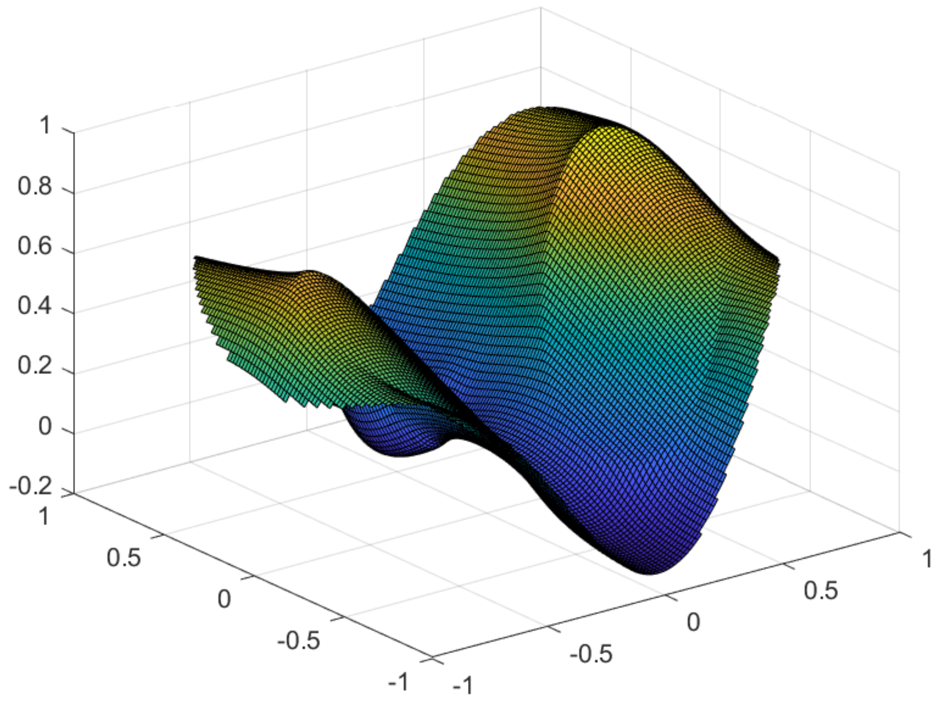
For the second example we consider, as in the previous example, the conductivity $\sigma_{lm}(z) = \sigma(z)$ and as a diagonal matrix as well, but in this case the conductivity is different in the diagonal, the equation that represent this is the following

$$\mathrm{di}v\left(\sigma(z)\begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}\cdot\nabla U\right) = 0, \quad \text{in} \quad \Omega,$$

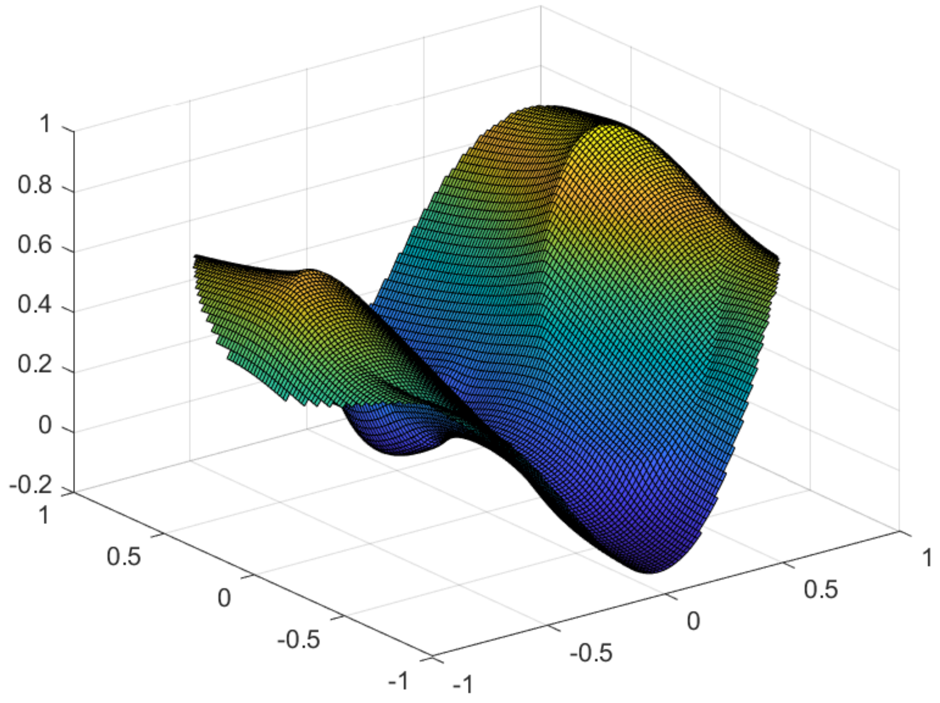
For this example we will compare the results obtained by the vectorial algorithm and the results obtained by the scalar algorithm presented in [28]. For this cases, the scalar algorithm solves the folowings problems:

$$\mathrm{di}v\left(\lambda\sigma(z)\nabla u\right) = 0, \quad \text{in} \quad \Omega,$$

where $\lambda = 2, 3$ and $\overline{\mathcal{N}}_1$, $\overline{\mathcal{N}}_2$ represent the scalar solution for $\lambda = 2$ and $\lambda = 3$, respectively. In the Figures 9.3-9.4 it can be seen that the two equations act independently, but since the factor that multiplies σ is different in each equation, a different solution is observer for $\mathcal{N}_1$ and $\mathcal{N}_2$. Also in the Figures 9.3-9.4, it is possible to see that, the solutions have the same form.

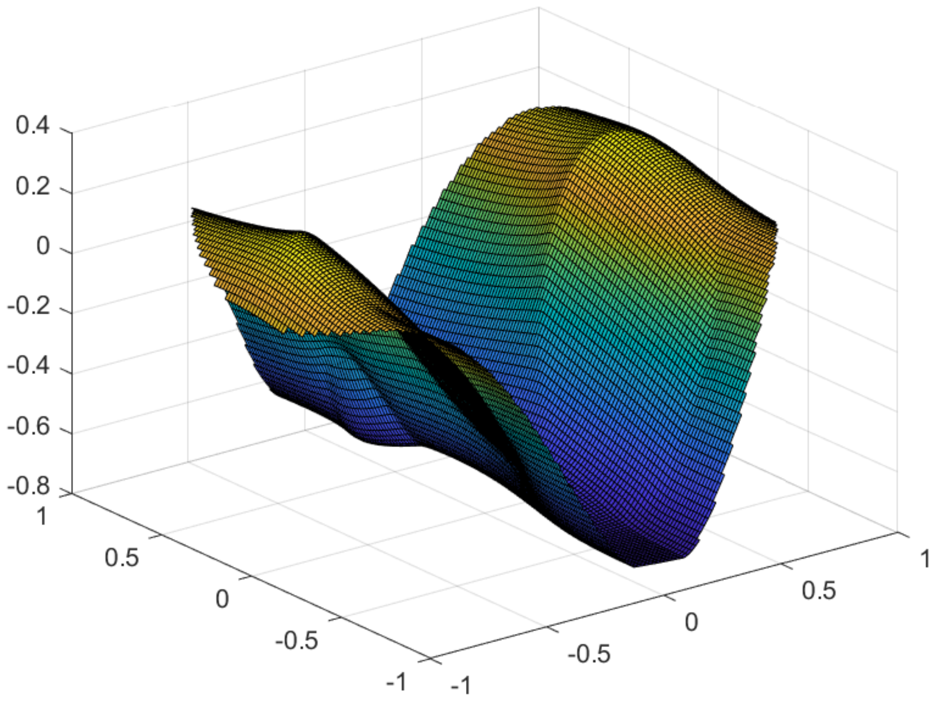


(a) Real part of $\mathcal{N}_1(z, 2)$.

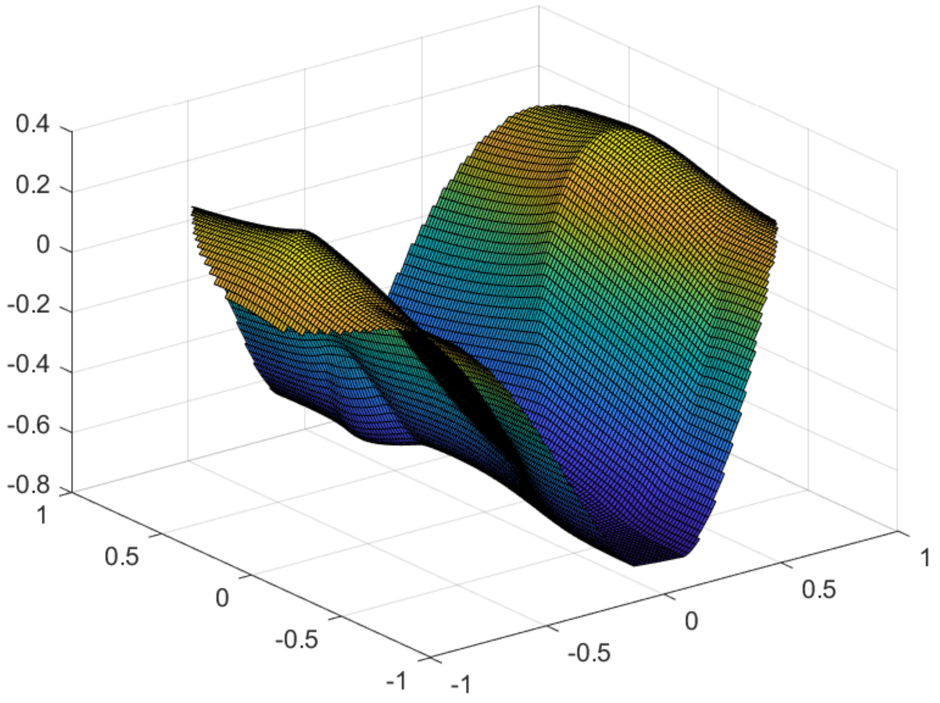


(b) Real part of $\overline{\mathcal{N}}_1(z, 2)$.

Figure 9.3: Three-dimensional mesh plot of the CGO solutions. Here $k = 2$ and $m = 8$.



(a) Real part of $\mathcal{N}_2(z, 2)$.



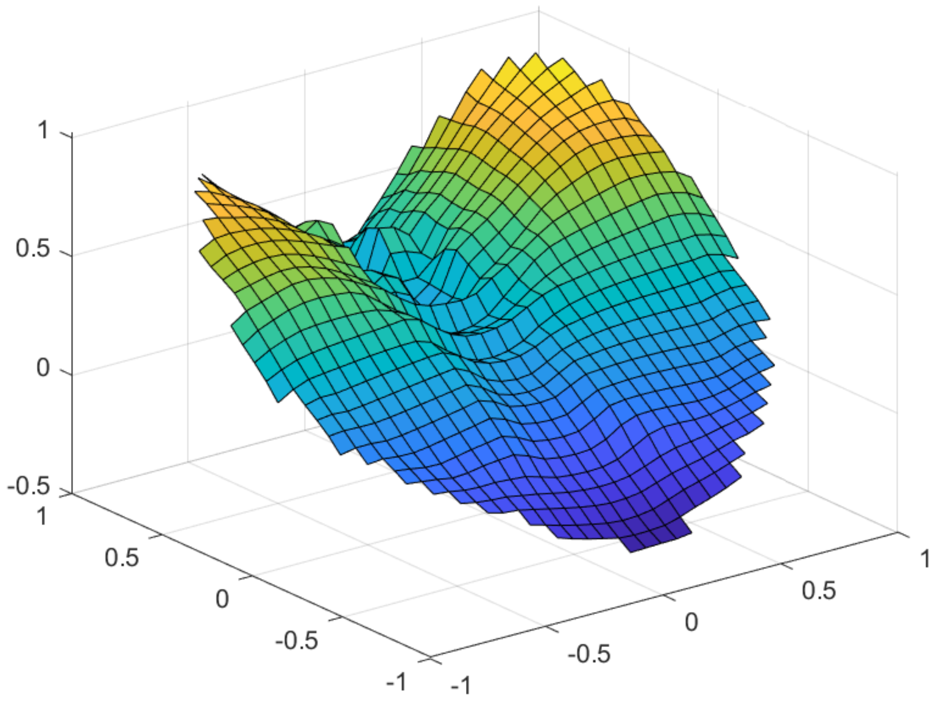
(b) Real part of $\overline{\mathcal{N}}_2(z, 2)$.

Figure 9.4: Three-dimensional mesh plot of the CGO solutions. Here $k = 2$ and $m = 8$.

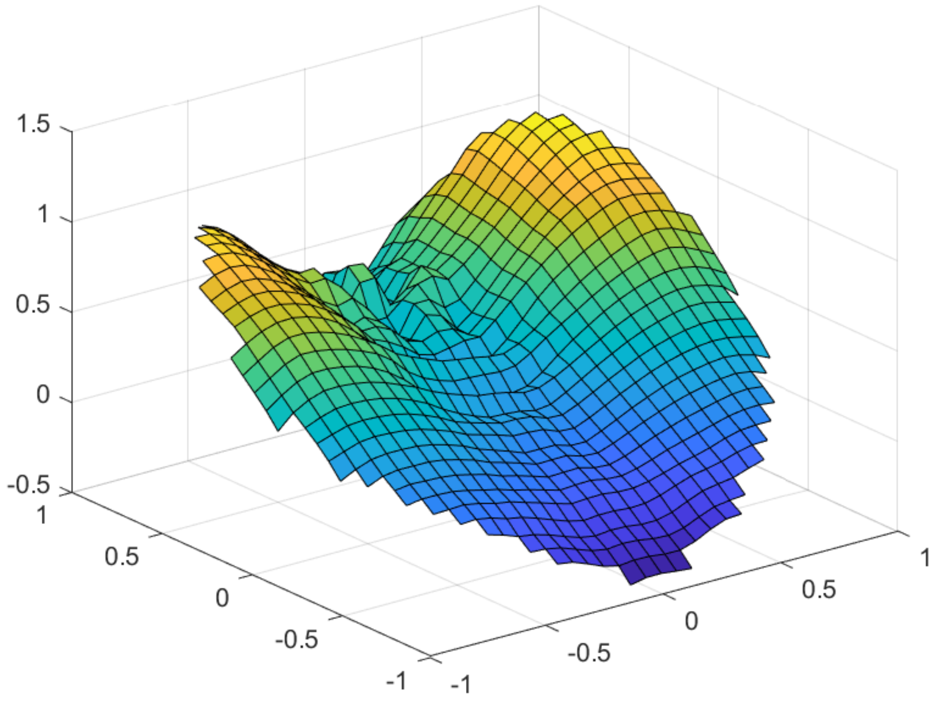
For the third example, we also take $\sigma_{lm}(z) = \sigma(z)$, but in this case, we consider a non-diagonal matrix, with this choice we have that the equations are coupled. This will be represented by the following equation

$$\mathrm{di}v \left(\sigma(z) \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix} \cdot \nabla U \right) \quad = \quad 0, \quad \text{in} \quad \Omega,$$

In order to assess the consistency and mesh independence of the method in this example, we have three different mesh refinements, $m = 6, 7, 8$, and compare their differents results. In the Figures 9.5-9.7 can be seen that the coupled equation gives us solutions that differ both in their magnitude and their shape, so it can be seen that there is an interaction between the conductivities of each of the solutions. From Figures 9.5-9.7 it is possible to see that, for differents refinements, the solution is the same. In the Tables 9.1-9.2 it can be see the error between the discretization m and $m - 1$, where it can be observed that as the discretization increases, the error compared to the previous discretization is lower. In order to compare both solutions, we consider only the common nodes that contain both discretizations.

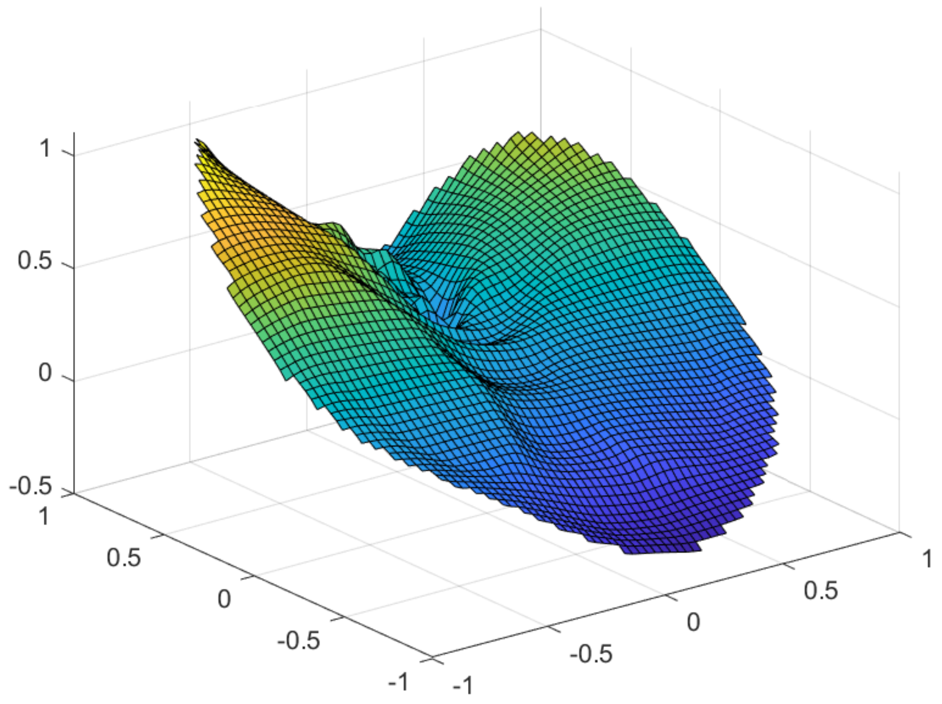


(a) Real part of $\mathcal{N}_1(z, 2)$.

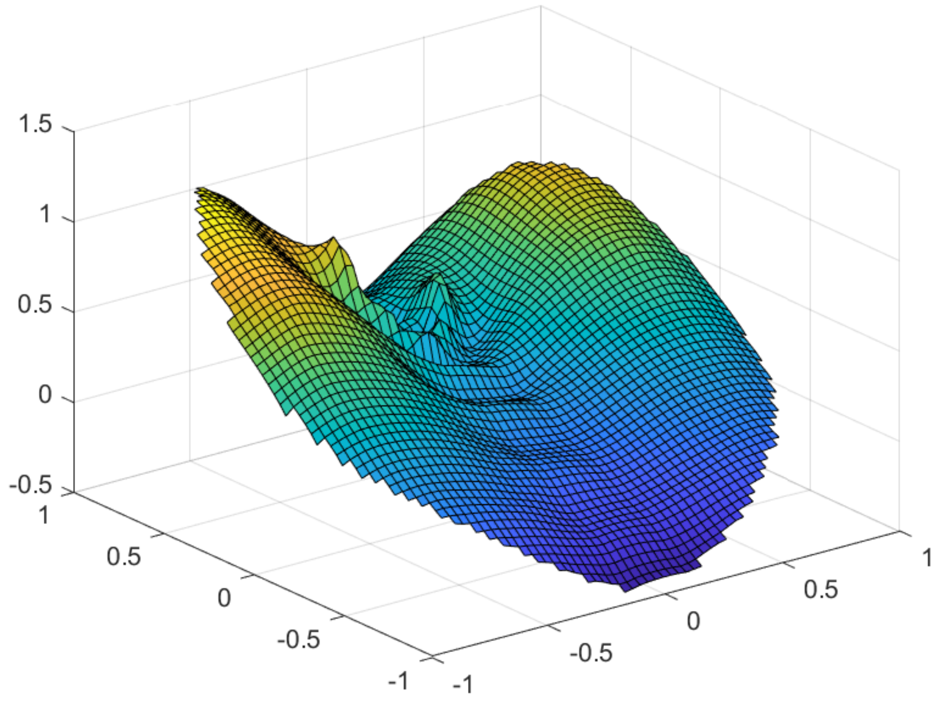


(b) Real part of $\mathcal{N}_2(z, 2)$.

Figure 9.5: Three-dimensional mesh plot of the CGO solutions. Here $k = 2$ and $m = 6$.

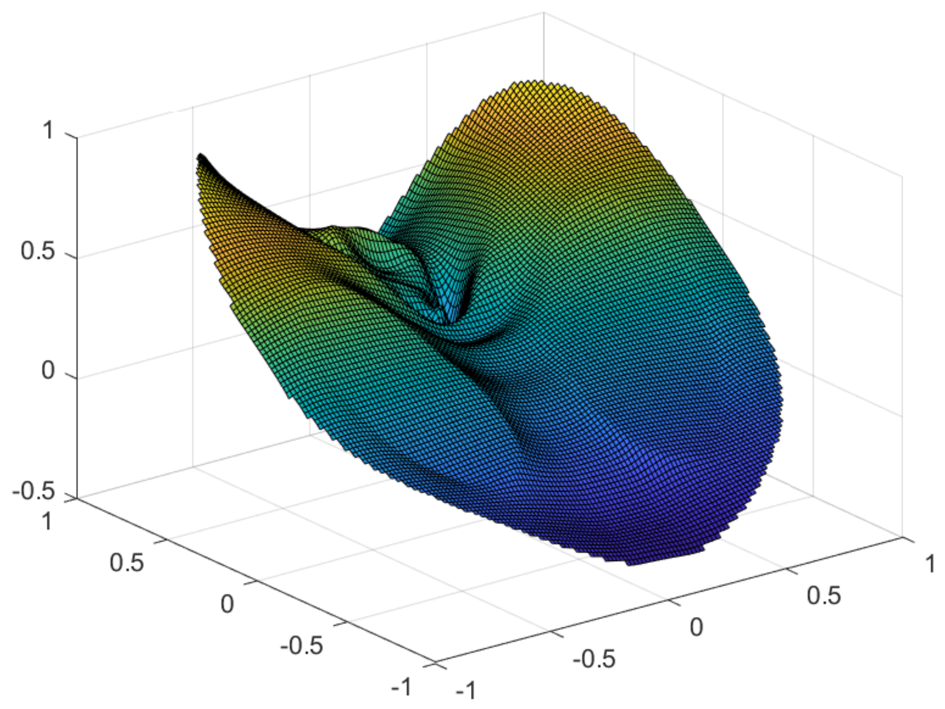


(a) Real part of $\mathcal{N}_1(z,2)$.

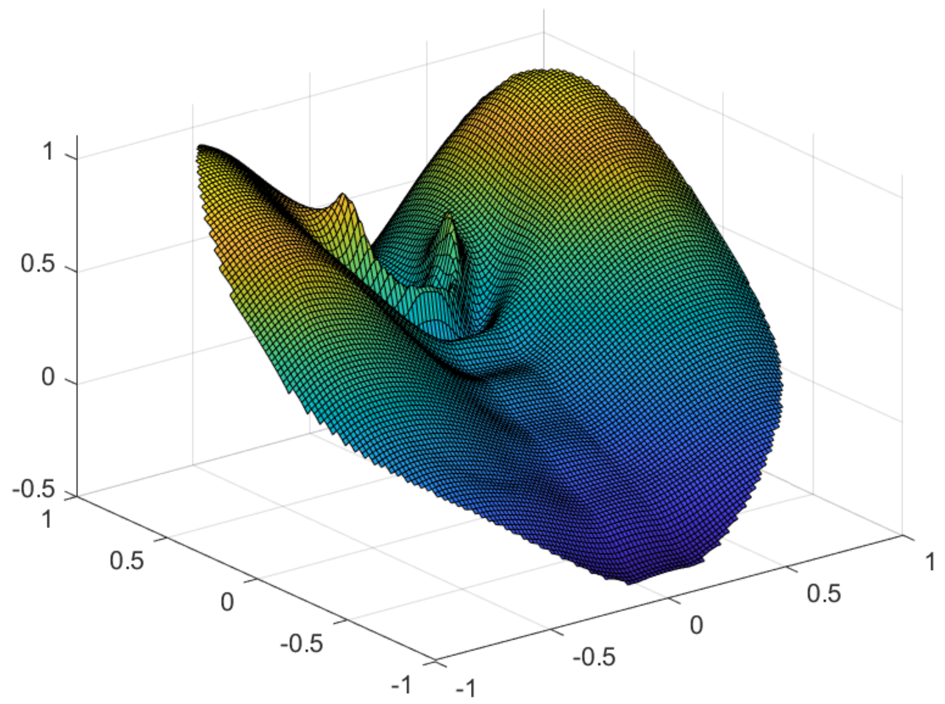


(b) Real part of $\mathcal{N}_2(z,2)$.

Figure 9.6: Three-dimensional mesh plot of the CGO solutions. Here $k = 2$ and $m = 7$.



(a) Real part of $\mathcal{N}_1(z, 2)$.



(b) Real part of $\mathcal{N}_2(z, 2)$.

Figure 9.7: Three-dimensional mesh plot of the CGO solutions. Here $k = 2$ and $m = 8$.

m	$\|\mathcal{N}_1^m\|_\infty$	$\|\mathcal{N}_1^m - \mathcal{N}_1^{m-1}\|_\infty$
7	1.0990	0.2955
8	0.9603	0.1619
9	0.9837	0.0809
10	0.9833	0.0394

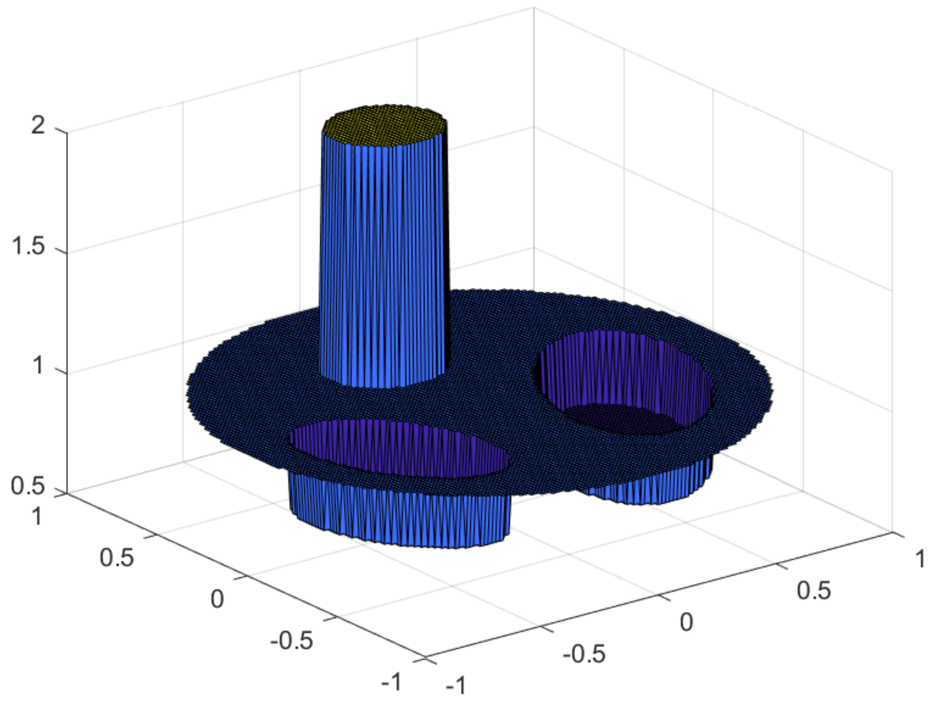
Table 9.1: Comparison of $\mathcal{N}_1$ for different refinements m.

m	$\|\mathcal{N}_2^m\|_\infty$	$\|\mathcal{N}_2^m - \mathcal{N}_2^{m-1}\|_\infty$
7	1.2320	0.3385
8	1.1026	0.2319
9	1.1264	0.1241
10	1.1264	0.0480

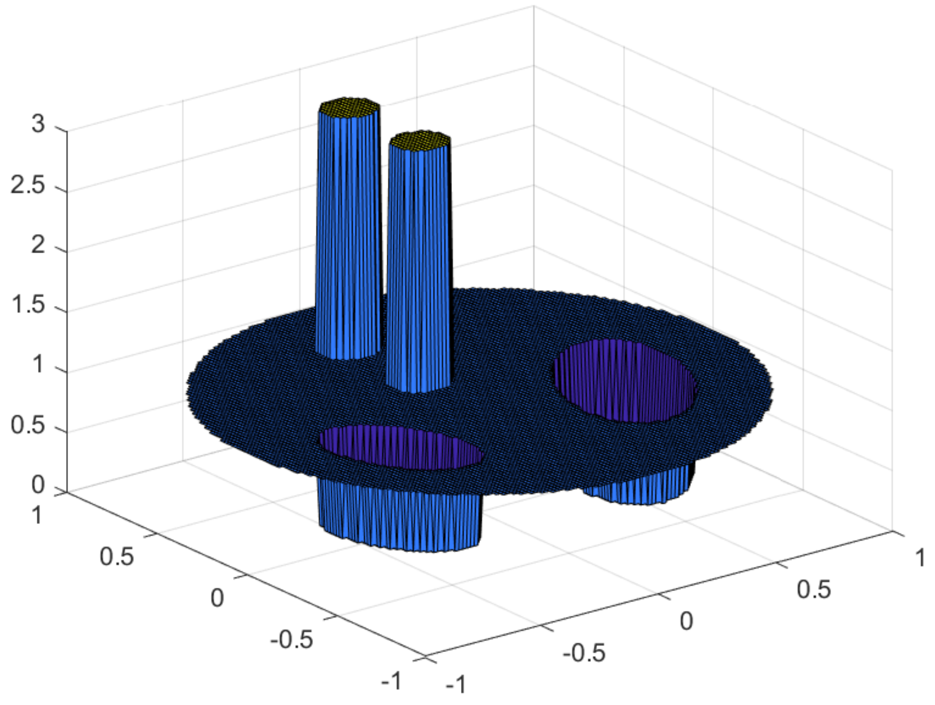
Table 9.2: Comparison of $\mathcal{N}_2$ for different refinements m.

Now, for the last example, we will consider different conductivities for the matrix but with the condition that the matrix remains symmetric, see Figures 9.8-9.9 for plot of the conductivities. The equation that represent this is the following

$$\mathrm{di}v\left(\begin{pmatrix} \sigma_{11}(z) & \sigma_{12}(z) \\ \sigma_{21}(z) & \sigma_{22}(z) \end{pmatrix} \cdot \nabla U\right) \quad = \quad 0, \quad \text{in} \quad \Omega,$$

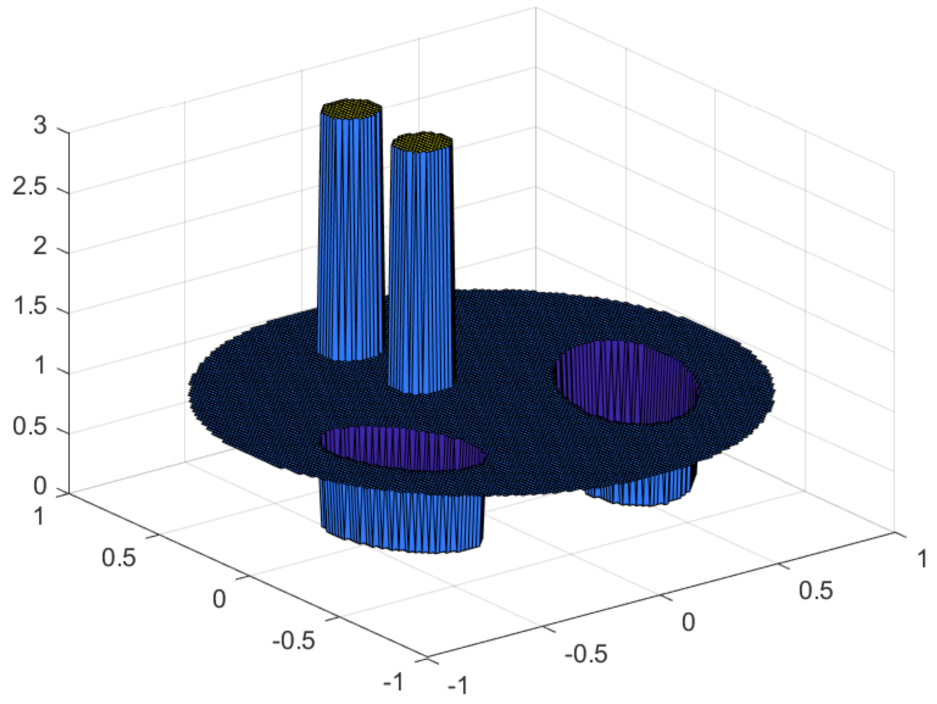


(a) $\sigma_{11}(z)$.

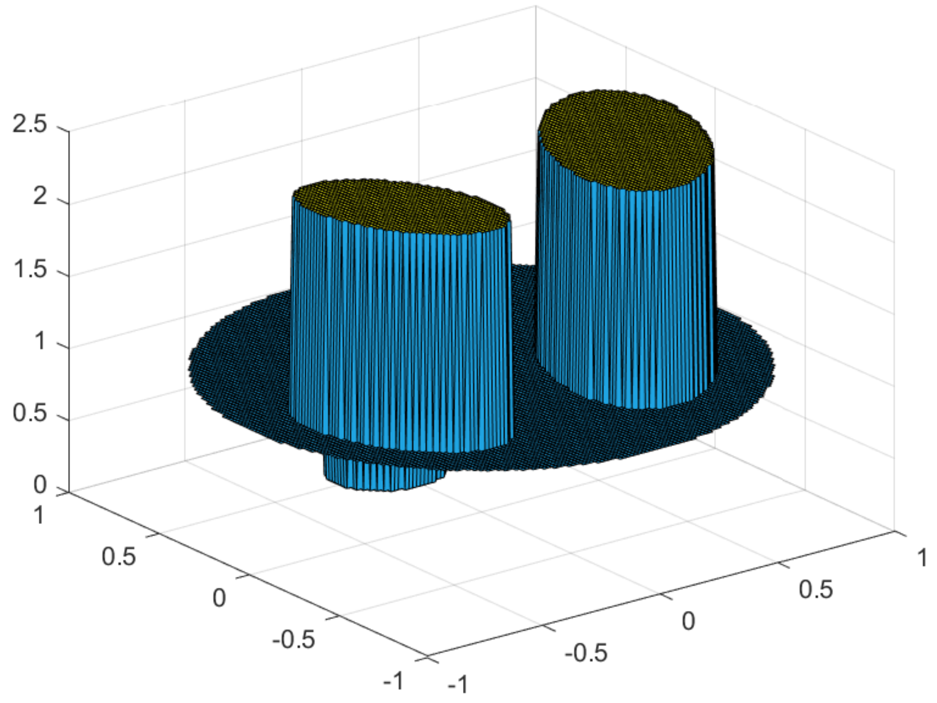


(b) $\sigma_{12}(z)$.

Figure 9.8: Three-dimensional mesh plot of the discontinuous conductivity σ_{lm}.



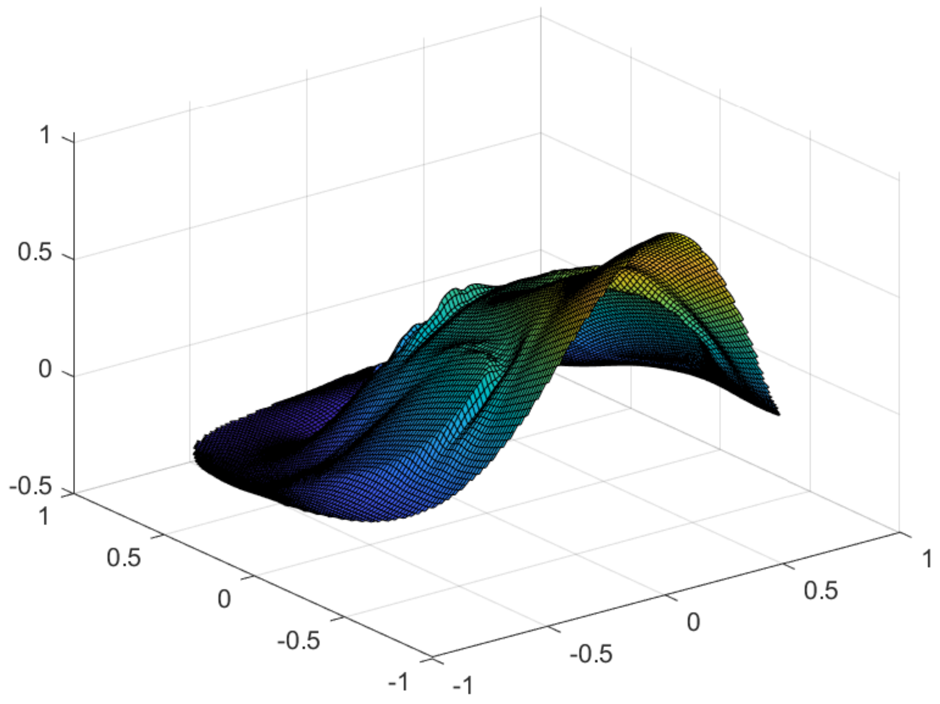
(a) $\sigma_{21}(z)$.



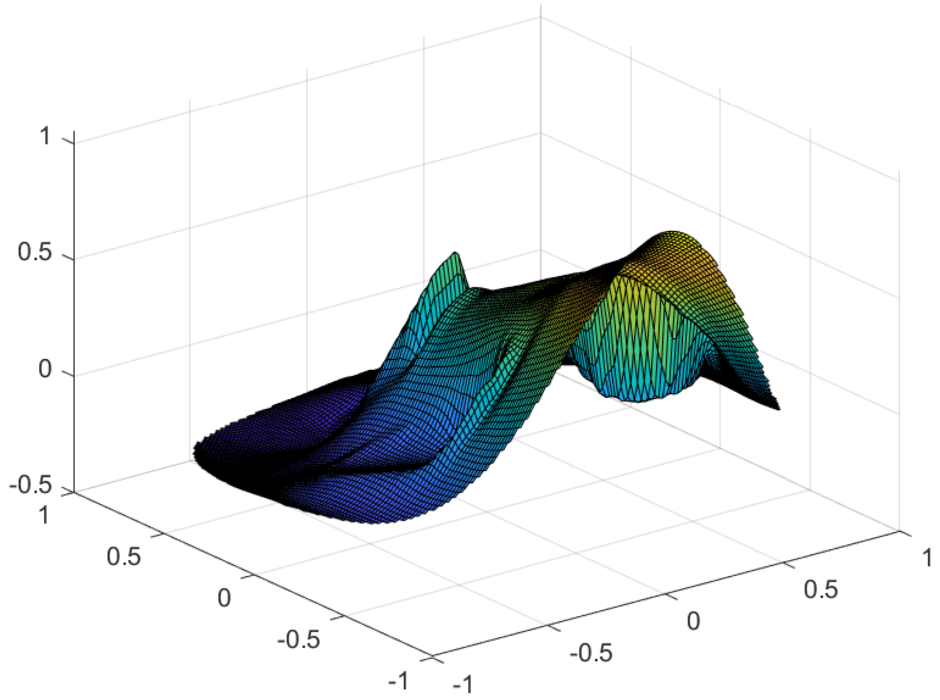
(b) $\sigma_{22}(z)$.

Figure 9.9: Three-dimensional mesh plot of the discontinuous conductivity σ_{lm}.

In the Figure 9.10 it can be seen that the coupled equation gives us solutions that differ both in their magnitude and their shape, so it can be seen that there is an interaction between the conductivities of each of the solutions. In this example it is possible to consider different conductivities that represent deformities of organs and tumors that may have these organs.



(a) Real part of$\mathcal{N}_1(z, 2)$.

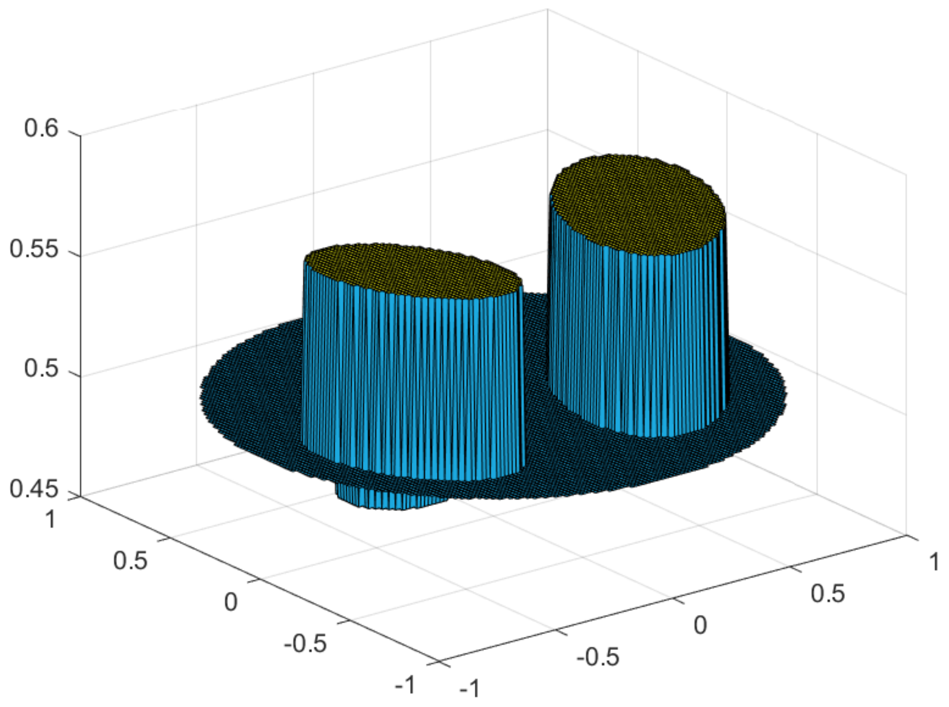


(b) Real part of $\mathcal{N}_2(z, 2)$.

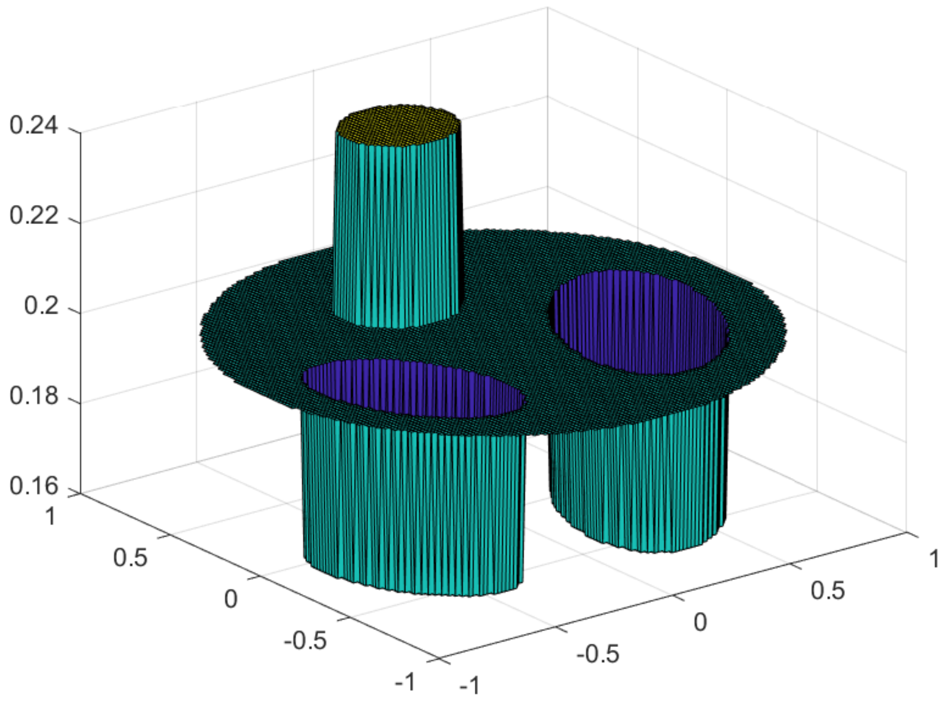
Figure 9.10: Three-dimensional mesh plot of the CGO solutions corresponding to the nonsmooth conductivity σ_{lm}. Here $k = 2$ and $m = 8$.

Finally, for the fifth example, we will consider the non-symmetric matrix, i.e., each component of the matrix is a different conducticity, see Figure 9.11-9.12 for plot of the conductivities. The equation that represent this is the following

$$\mathrm{di}v\left(\left(\begin{array}{cc}\sigma_{11}(z) & \sigma_{12}(z)\\ \sigma_{21}(z) & \sigma_{22}(z)\end{array}\right)\cdot\nabla U\right) = 0, \quad \text{in} \quad \Omega, \tag{9.1}$$

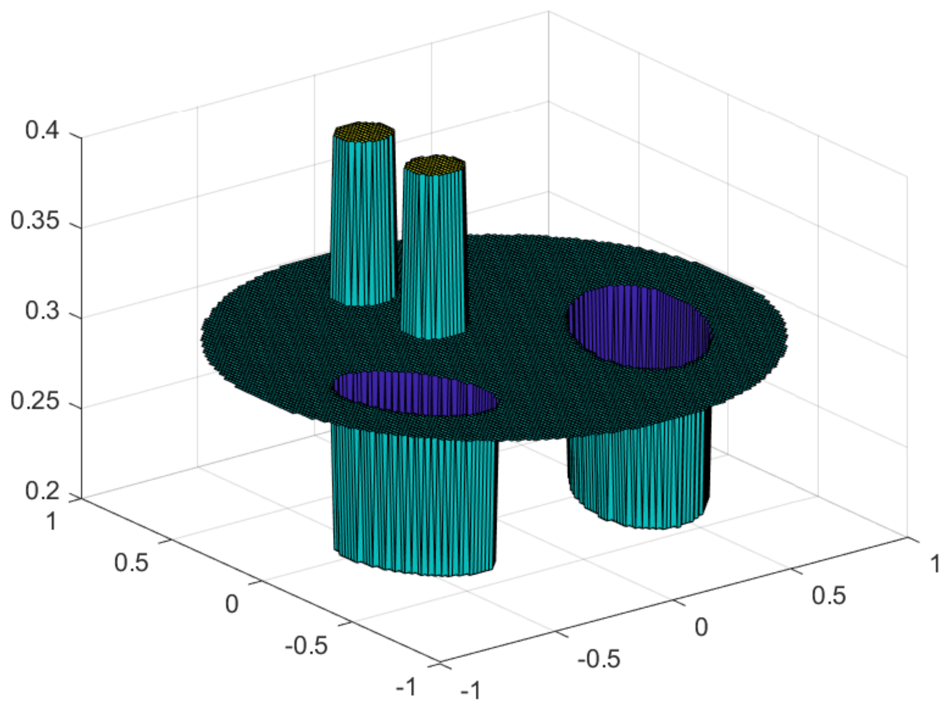


(a) $\sigma_{11}(z)$.

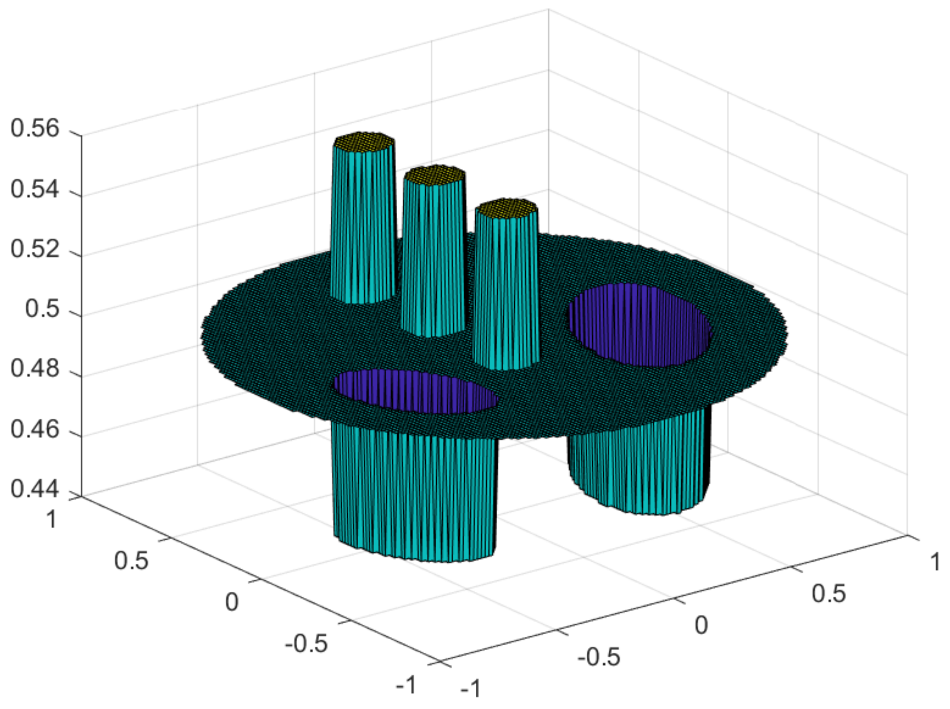


(b) $\sigma_{12}(z)$.

Figure 9.11: Three-dimensional mesh plot of the discontinuous conductivity σ_{lm}.



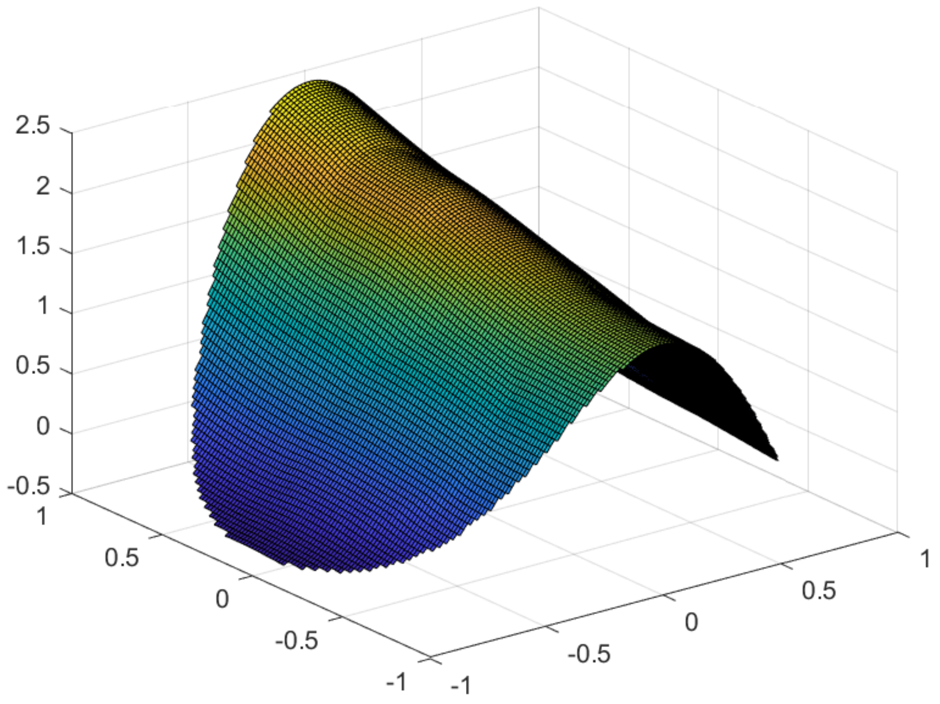
(a) $\sigma_{21}(z)$.



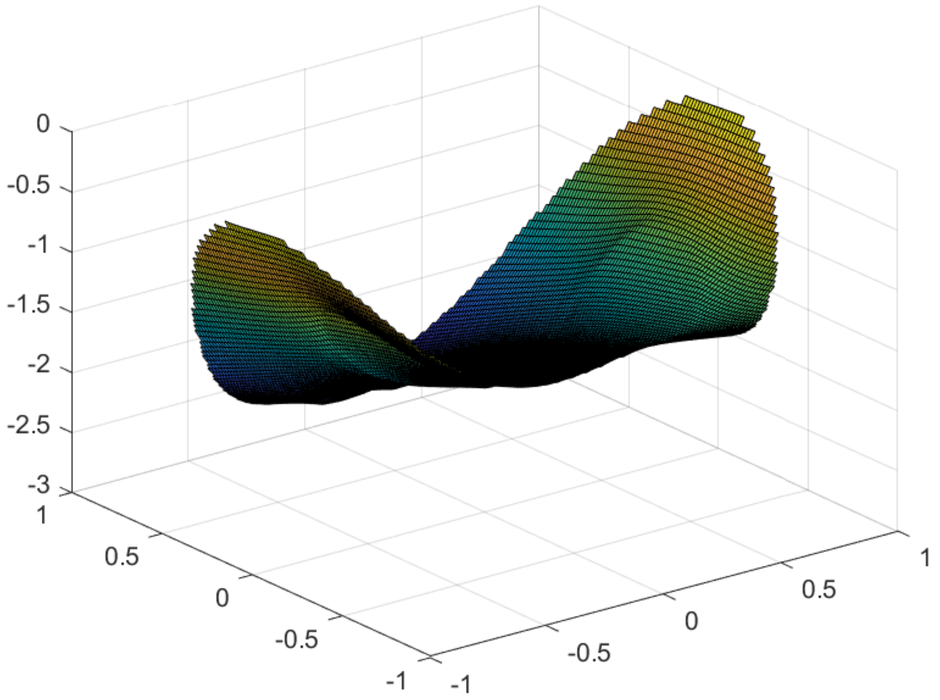
(b) $\sigma_{22}(z)$.

Figure 9.12: Three-dimensional mesh plot of the discontinuous conductivity σ_{lm}.

In the Figure 9.13, it can be seen that, for non-symmetric matrix, the coupled system give us different solutions in shape and magnitude. In the example is possible to see that for non-symmetric matrix σ, our algorithm can reconstruct the CGO solutions of the system.



(a) Real part of$\mathcal{N}_1(z,2)$.



(b) Real part of $\mathcal{N}_2(z,2)$.

Figure 9.13: Three-dimensional mesh plot of the CGO solutions corresponding to the nonsmooth conductivity σ_{lm}. Here $k = 2$ and $m = 8$.

Conclusion

We present a numerical reconstruction of the CGO solutions for the conductivity systems, this reconstruction is based on solving the problem for a Beltrami system equivalent to the conductivity system. For this, we prove the existence of the CGO solutions of the Beltrami system and then reconstruct the solution of this system. Several examples were presented to represent all the possible cases of conductivity systems, where we consider a simple case of a diagonal matrix to a more complex example of anisotropic conductivity and non-symmetric matrix. In the examples are considered different ways to validate the method where we can see that we can replicate the solution obtained for the scalar case, it can bee seen that the method converges under mesh refinements.

In order to extend the results, we have identified some future works listed below:

1. To solve Calderón's problem (Uniqueness, Stability, Reconstruction) for conductivity systems, considering the CGO solutions.
2. To consider numerical reconstruction of the CGO solution for other elliptic vectorial equations, for instance the elasticity equation or Stoke equation, using the same strategy.
3. To solve Calderón's problem for the conductivity equation in $\mathbb{R}^n$, following the ideas of Santacesaria [51], where it is necessary the Astala and Päivärinta method [5] and Clifford algebras.

Appendix A

Meshes Creation

In this appendix we will show the methodology used to create the meshes for the different numerical results shown in this work. To create the meshes, in the first instance, to build the mesh surfaces using FEniCS, and then fill each surface with tetrahedron using TetGen.

To construct the mesh that represents our domain, first we need to define the interior cavity in each time step t_{i}. For this, we consider a flat surface that represents the base of the cavity, which is projected on the z axis. More specifically, the base is repeated and scaling at different levels of z in the range $z \in [0, z_{max}]$ to build the cavity surface in 3D. For example, Figure A.1 displays the forty interior flat surface that represent the cavity sequences that we use in our synthetic tests.

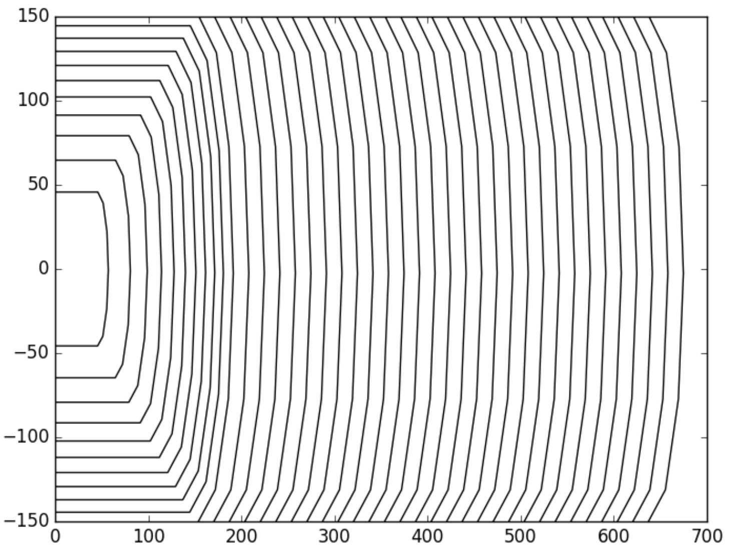


Figure A.1: Interior flat surface that represent the cavity sequences.

Once the cavity in each time step is known, for each cavity, a mesh is built around it where near the boundary of the cavity a more refined mesh is defined in order to better reflect the effects produced by each advance of the cavity in the damage process. Figure A.2 shows a lateral view of the mesh in the last cavity advance for the synthetic case, where, we can observe the greater refinement around the cavity.

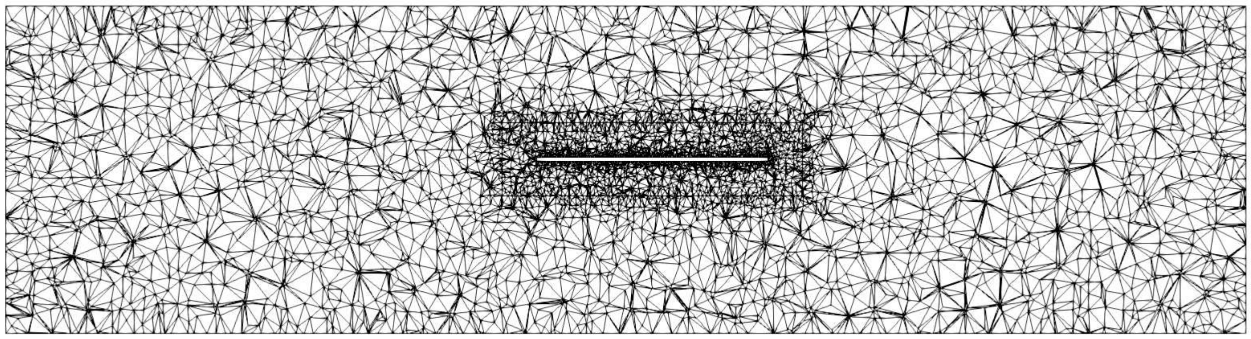
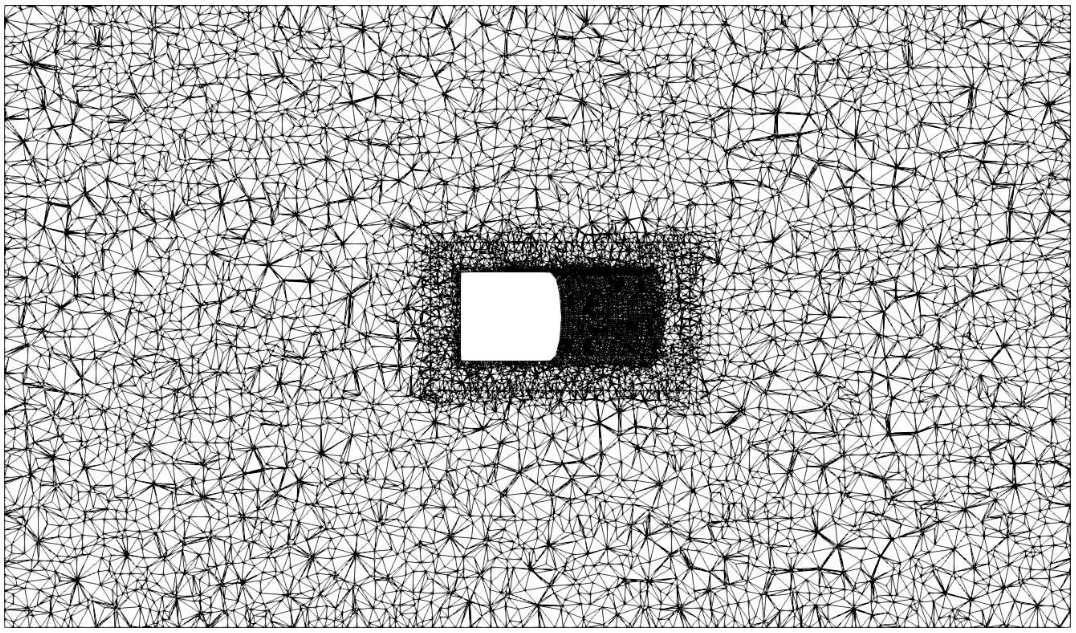


Figure A.2: Lateral cut of the mesh in the last time step.

To define the mesh sequences and taking into account the irreversibility of the problem, it is necessary that as the cavity progresses, the mesh should not change its form between one advance to another, that is, the nodes should be kept in the same place in each advance of this. To make this possible, we will consider a series of meshes in which we will only eliminate the internal domain of the cavity, keeping the rest of the nodes in the same position. In the Figure A.3, the mesh can be observed for different advances of this in a horizontal cut, where the location of the nodes for each advance is not affected. On the other hand, in Figure A.4 it possible see the mesh for the same advances but now in a Lateral cut.


(a) $\Omega(t_{20})$

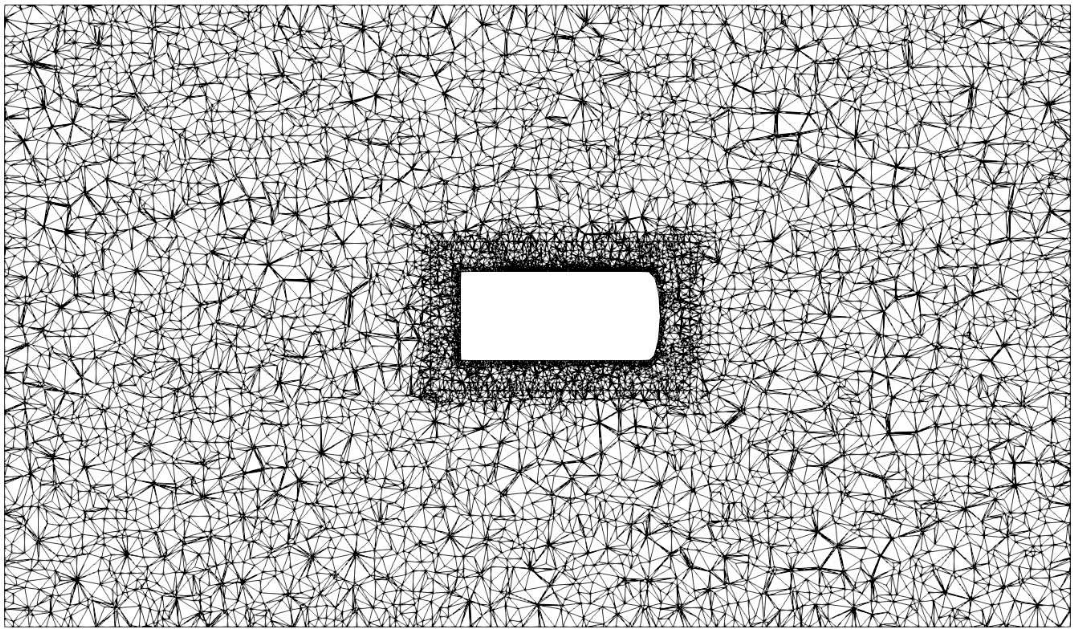
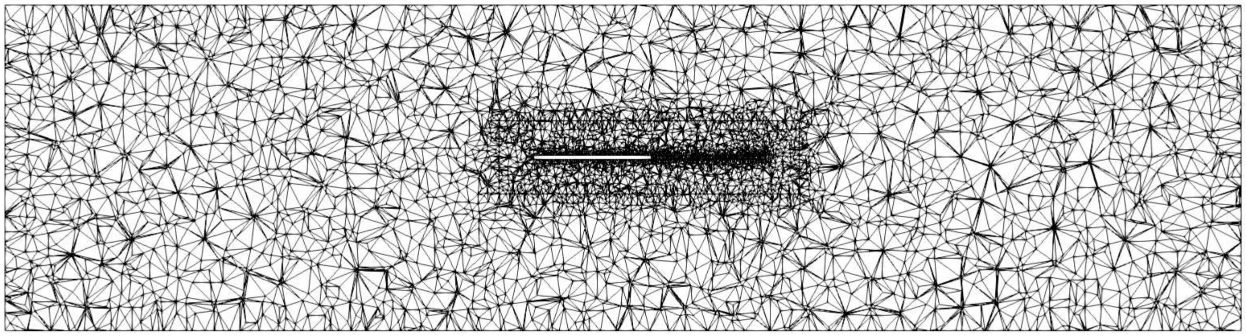
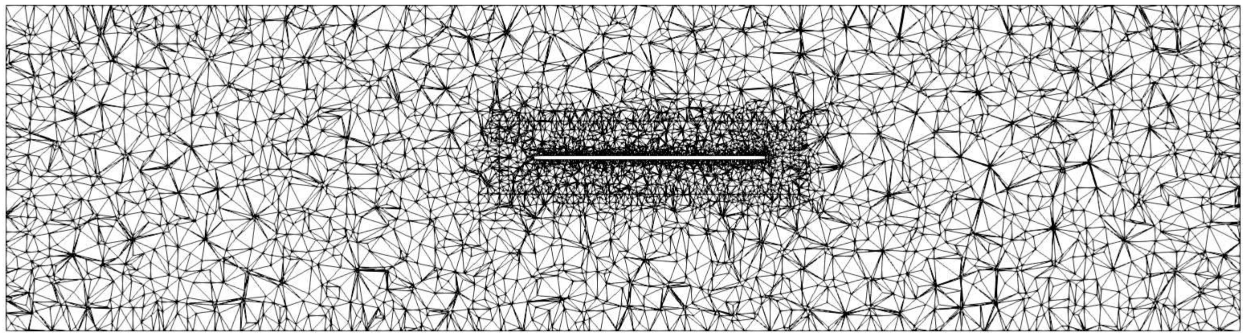

(b) $\Omega(t_{40})$

Figure A.3: Horizontal cut of the mesh for different time step.



(a) $\Omega(t_{20})$



(b) $\Omega(t_{40})$

Figure A.4: Lateral cut of the mesh for different time step.

Appendix B

Codes in Python

This appendix presents the computer codes used in the Part I, which have been developed in Python and with the use of FEniCs software. FEniCS is a popular open-source computing platform for solving partial differential equations (PDEs). FEniCS enables users to quickly translate scientific models into an efficient finite element code. With the high-level Python and C++ interfaces to FEniCS, it is easy to get started, but FEniCS offers also powerful capabilities for more experienced programmers. FEniCS runs on a multitude of platforms ranging from laptops to high-performance clusters. For more information on FEniCS and the latest FEniCS software updates, visit the FEniCS website: http://fenicsproject.org.

The unilateral constraint included in the damage problem requires the use of variational inequalities solvers, here we use the open source library PETSc [7], where this capability is available. The problem is discretized in space with standard triangular finite elements with piece-wise linear approximation for u and α. The mesh size is selected to have a suffiecient number of elements in the localization band (we tipically use 5-10 elements in a localization band [40]). A finite element implementation based on the open-source FEniCS library are used for this problems [36, 38].

B.1 Code for Cavity in 2D

B.1.1 Cavity2D.py

```
#------------------------------------------------------------------------
#  IMPORT ALL PARAMETERS FROM parameters.py.
#------------------------------------------------------------------------
from parameters import *
# read paramaters from command line
parameters.parse()
# set some dolfin specific parameters
parameters["form_compiler"]["optimize"]     = True
parameters["form_compiler"]["cpp_optimize"] = True
parameters["form_compiler"]["representation"] = "uflacs"
parameters["allow_extrapolation"] = True
# The minimization procedure requires parameters to get a suitable performance
    . The following is a suitable set of arrangements.
```

```
solver_minimization_parameters =  {"method" : "gpcg",
                                    "linear_solver" : "gmres",
                               #--------------------------------
                               # These are parameters for optimization
                               #--------------------------------
                               "line_search": "armijo",
                               "preconditioner" : "hypre_euclid",
                               "maximum_iterations" :200,
                               "error_on_nonconvergence": False,
                               #--------------------------------
                               # These are parameters for linear solver
                               #--------------------------------
                               "krylov_solver" : {
                                   "maximum_iterations" : 200,
                                   "nonzero_initial_guess" : True,
                                   "report" : True,
                                   "monitor_convergence" : False,
                                   "relative_tolerance" : 1e-8
                               }
                              }
# The linear solver requires parameters to get a suitable performance. The
    following is a suitable set of arrangements.
solver_LS_parameters =  {"linear_solver" : "cg",
                               "symmetric" : True,
                               "preconditioner" : "jacobi",
                               "krylov_solver" : {
                                   "report" : True,
                                   "monitor_convergence" : False,
                                   "relative_tolerance" : 1e-8
                                   }
                               }
#----------------------------------------------------------------
# DEFINE THE OPERATORS.
#----------------------------------------------------------------
# Constitutive functions of the damage model
def w(alpha):
    if model=="lin":
        return w11*alpha
    if model =="quad":
        return w11*alpha**2
def A(alpha):
    return (1-alpha)**2
# Define th Deviator and Spheric Tensors
def Dev(Tensor):
    return Tensor-Sph(Tensor)
def Sph(Tensor):
    Tensor2=as_matrix([[1,0],[0,1]])
    return (inner(Tensor,Tensor2)/inner(Tensor2,Tensor2))*Tensor2
# Strain and stress in free damage regime.
def eps(v):
    return sym(grad(v))
def sigma_0(eps):
    return 2.0*mu*(eps)+lmbda*tr(eps)*Identity(ndim)
# modification of the Young modulus.
def sigma(eps,alpha):
    return (A(alpha)+k_ell)*sigma_0(eps)
```

```
#------------------------------------------------------------------------------
#  MESH WITHOUT CAVITY.
#------------------------------------------------------------------------------
# Define the function, test and trial fields.
# "u, du, v" are vectorial expressions.
u    =    Function(V_vector,name="u")
du   =    TrialFunction(V_vector)
v    =    TestFunction(V_vector)
# "alpha, dalpha, beta" are scalar.
alpha    =    Function(V_scalar,name="alpha")
alpha0   =    Function(V_scalar,name = "alpha")
dalpha   =    TrialFunction(V_scalar)
beta     =    TestFunction(V_scalar)
# Define the energies functions.
W_energy     =    Function(V_scalar,name="energy_w")
dev_energy   =    Function(V_scalar,name="energy_dev")
div_energy   =    Function(V_scalar,name="energy_sph")
E_energy     =    Function(V_scalar,name="energy_E")
p_energy     =    Function(V_scalar,name="energy_p")
dis_energy   =    Function(V_scalar,name="energy_dis")
# Define the stress and strain functions.
stressG =    Function(V_tensor,name="sigma")
strainG =    Function(V_tensor,name="epsilon")
# Define the alpha auxiliar function.
alphaAux     =    Function(V_scalar,name="alpha")
# Define the function "alpha_error" to measure relative error.
alpha_error =    Function(V_scalar)
# Interpolate the initial condition for the damage variable "alpha".
alpha_0      =    interpolate ( Expression("0.",degree = 2), V_scalar)
# Define ds and dx.
ds   =    Measure('ds',domain=mesh,subdomain_data=boundaries)
dx   =    Measure('dx',domain=mesh)
# Let us define the total energy of the system as the sum of elastic energy,
    dissipated energy due to the damage and external work due to body forces.
# Elastic energy for differents cases.
if case=="Marigo":
     elastic_energy1 =    0.5*inner(sigma(eps(u),alpha),eps(u))*dx
     elastic_energy2 =    0.5*inner(sigma(eps(u),alpha),eps(u))*dx
if case=="DMSF":
     Id_Tensor        =    as_matrix([[1,0],[0,1]])
     elastic_energy1 =    (0.5*((lmbda+mu)*(tr(eps(u))**2)*inner(Id_Tensor,
    Id_Tensor)))*dx+((A(alpha)+k_ell)*mu*inner(Dev(eps(u)),Dev(eps(u))))*dx
     elastic_energy2 =    (0.5*((lmbda+mu)*(tr(eps(u))**2)*inner(Id_Tensor,
    Id_Tensor)))*dx+((A(alpha)+k_ell)*mu*inner(Dev(eps(u)),Dev(eps(u))))*dx
if case=="ShearComp":
     elastic_energy1 =    0.5*inner(sigma(eps(u),alpha),eps(u))*dx
     elastic_energy2 =    0.5/E*(inner(Dev(sigma(eps(u),alpha)),Dev(sigma(eps(u)
    ,alpha)))\
                              -kappa*inner(Sph(sigma(eps(u),alpha)),Sph(sigma(
    eps(u),alpha))))*dx
# External work.
external_work    =    dot(body_force,u)*dx
# Dissipated energy.
dissipated_energy    =    (w(alpha)+ellv**2*w1*dot(grad(alpha),grad(alpha)))*dx
# Definition of the total energy
total_energy1    =    elastic_energy1+dissipated_energy-external_work
```

```
total_energy2   =   elastic_energy2+dissipated_energy-external_work
# Weak form of damage problem. This is the formal expression for the tangent
    problem which gives us the equilibrium equations.
E_u     =   derivative(total_energy1,u,v)
E_alpha =   derivative(total_energy2,alpha,beta)
# Hessian matrix.
E_alpha_alpha   =   derivative(E_alpha,alpha,dalpha)
# Writing tangent problems in term of test and trial functions for matrix
    assembly.
E_du     =  replace(E_u,{u:du})
E_dalpha =  replace(E_alpha,{alpha:dalpha})
# Once the tangent problems are formulated in terms of trial and text
    functions, we define the variatonal problems.
# Variational problem for the displacement.
problem_u   =   LinearVariationalProblem(lhs(E_du),rhs(E_du),u,bc_u)
# Define the classs Optimization Problem for then define the damage.
# Variational problem for the damage (non-linear to use variational inequality
     solvers of petsc).
class DamageProblem(OptimisationProblem):
    def __init__(self):
        OptimisationProblem.__init__(self)
    # Objective vector
    def f(self,x):
        alpha.vector()[:]=x
        return assemble(total_energy2)
    # Gradient of the objective function
    def F(self,b,x):
        alpha.vector()[:]=x
        assemble(E_alpha,tensor=b)
    # Hessian of the objective function
    def J(self,A,x):
        alpha.vector()[:]=x
        assemble(E_alpha_alpha, tensor=A)
# Define the minimization problem using the class.
problem_alpha   =   DamageProblem()
# Set up the solvers. Define the object for solving the displacement problem,
    "solver_u".
solver_u    =   LinearVariationalSolver(problem_u)
# Get the set of paramters for the class "solver_u". This only requires the
    solution of linear system solver.
solver_u.parameters.update(solver_LS_parameters)
# Define the corresponding object, "solver_alpha".
# The object associated to minimization is created.
solver_alpha    =   PETScTAOSolver()
# Get the set of paramters for the class "solver_alpha". This requires the
    solution of a minimization problem.
solver_alpha.parameters.update(solver_minimization_parameters)
# As the optimization is a constrained type we need to provide the
    corresponding lower and upper  bounds.
lb = interpolate(Expression("0.0",degree = 0),V_scalar)
ub = interpolate(Expression ("0.95", degree = 0),V_scalar)
lb.vector()[:] = alpha.vector()
# Crete the files to store the solution of damage and displacements.
file_alpha  =   File(savedir+"/alpha.pvd")
file_u      =   File(savedir+"/u.pvd")
#---------------------------------------------------------------------------
```

```
# ALTERNATE MINIIZATION.
#------------------------------------------------------------
# Initialization
iter = 1; err_alpha = 1; err_alpha_aux=1

a0 = Vector(MPI.COMM_SELF)
a1 = Vector(MPI.COMM_SELF)
# Iterations of the alternate minimization stop if an error limit is reached
    or a maximim number of iterations have been done.
while True and err_alpha > toll and iter < maxiter :
     alpha.vector().gather(a0, np.array(range(V_scalar.dim()), "intc"))
     amin=alpha.vector().min()
     amax=alpha.vector().max()
     if MPI.COMM_WORLD.Get_rank() == 0:
          print("Job %d: Iteration:  %2d, a0:[%.8g,%.8g]"%(MPI.COMM_WORLD.
    Get_rank(),iter,a0.min(),a0.max()))
     # solve elastic problem
     solver_u.solve()
     # solve damage problem via a constrained minimization algorithm.
     solver_alpha.solve(problem_alpha,alpha.vector(),lb.vector(),ub.vector())
     alpha.vector().get_local()[alpha.vector().get_local()>0.95]=0.95
     alpha.vector().gather(a1,np.array(range(V_scalar.dim()),"intc"))
     # Compute the norm of the the error vector.
     err_alpha = np.linalg.norm(a1-a0,ord=np.Inf)
     # Numerical Improve.
     if C_L != 0.0:
          while err_alpha > err_alpha_aux:
               alpha_2 = C_L*alpha_0+(1.0-C_L)*alpha
               alpha.assign(alpha_2)
               alpha.vector().gather(a1,np.array(range(V_scalar.dim()),"intc"))
               err_alpha = np.linalg.norm(a1-a0,ord=np.Inf)
     # Monitor the results
     if MPI.COMM_WORLD.Get_rank() >= 0:
          print ("Iteration:  %2d, Error: %2.8g, alpha_max: %.8g" % (iter,
    err_alpha,alpha.vector().max()))
     # Update the solution for the current alternate minimization iteration.
     err_alpha_aux   =   err_alpha
     alpha_0.assign(alpha)
     iter = iter + 1
# updating the lower bound with the solution of the solution corresponding to
    the current global iteration, it is for accounting for the irreversibility.
lb.vector ( ) [ :] = alpha.vector ( )
print ("-------------------------------------------------------")
print (" End of the alternate minimization without cavity. ")
print ("-------------------------------------------------------")
#------------------------------------------------------------
#  END ALTERNATE MINIMIZATION.
#------------------------------------------------------------
# Store u,alpha.
if MPI.COMM_WORLD.Get_rank()>=0:
     file_u      <<  (u, 0.)
     file_alpha  <<  (alpha,0.)
# Store the damage for this geometry
alphaAux.assign(alpha)
alpha0.assign(alpha)
print ("------------------------------------------------------------------------")
print ("                Geometry without cavity is finished.                ")
print ("--------------------------------------------------------------------")
# Remove previous integrating factors "dx, ds"
del ds, dx

#----------------------------------------------------------------------
#  MESH WITH CAVITY.
#----------------------------------------------------------------------
# Start loop over new geometries. These are obtained from a sequence of
    geometries which are obtained from an external folder. The number of
    external files is "NstepW" and the call is driven by the counter "itmesh".
#  Starting the loop of the mesh sequence. It is driven by the index "itmesh".
while itmesh <= NstepW :
    a0 = Vector(MPI.COMM_SELF)
    a1 = Vector(MPI.COMM_SELF)
    # Define Subdomain Cavity.
    class Structure(SubDomain):
        def inside(self, x,on_boundary):
            return between(x[0],(-500.0,-500+40*itmesh))andbetween(x[1]
    ,(-20.0,20.0))
    # Create sub domain markers and mark everaything as 0
    sub_domains = MeshFunction("size_t",mesh,mesh.topology().dim())
    sub_domains.set_all(0)
    # Mark structure domain as 1
    structure = Structure()
    structure.mark(sub_domains,1)
    # Extract sub meshes
    domain_new = SubMesh(mesh,sub_domains,0)
    tunel_new = SubMesh(mesh, sub_domains,1)
    # Define boundary sets for boundary conditions
    class Left_new(SubDomain):
        def inside(self,x,on_boundary):
            return near(x[0],-1500.)
    class Right_new(SubDomain):
        def inside(self,x,on_boundary):
            return near(x[0],1500.)
    class Top_new(SubDomain):
        def inside(self,x,on_boundary):
            return near(x[1],500.)
    class Bottom_new(SubDomain):
        def inside(self, x, on_boundary):
            return near(x[1],-500.)
    # Initialize sub-domain instances
    left_new     =   Left_new()
    right_new    =   Right_new()
    top_new      =   Top_new()
    bottom_new   =   Bottom_new()
    # define meshfunction to identify boundaries by numbers
    boundaries_new = MeshFunction("size_t",domain_new,domain_new.topology().
   dim()-1)
    boundaries_new.set_all(0)
    left_new.mark(boundaries_new,1) # mark left as 1
    right_new.mark(boundaries_new,2) # mark right as 2
    top_new.mark(boundaries_new,3) # mark top as 3
    bottom_new.mark(boundaries_new,4) # mark bottom as 4
    # Define the new ds and dx.
    dsN = Measure('ds', domain=domain_new, subdomain_data=boundaries_new)
    dxN = Measure('dx', domain=domain_new)
    #normal vectors
    normal_v_new = FacetNormal ( domain_new)
    # Create new function space for elasticity + Damage
    V_vector_new   =    VectorFunctionSpace(domain_new,"CG",1)
    V_scalar_new   =    FunctionSpace( domain_new,"CG",1)
    V_tensor_new   =    TensorFunctionSpace( domain_new,"DG",0)
    #------------------------------------------------------------------
    # REMARK: To generate a sequence of plots in paraview the name
    # of the variable must be the same. It is achieved by including
    # name="alpha" at the moment of the definition of the structure "alpha".
    #
    #  << alphaN =  Function ( V_scalar_new, name="alpha") >>
    #
    # The same definition needs to be done for displacement "u" and
    # other arrays as the difference of damage without cavity and
    # damage with cavity, for example "alphaDiff".
    #------------------------------------------------------------------
    # Define the function, test and trial fields
    uN           =    Function(V_vector_new,name="u")
    duN          =    TrialFunction(V_vector_new)
    vN           =    TestFunction(V_vector_new)
    alphaN       =    Function(V_scalar_new,name="alpha")
    alphaN_2     =    Function ( V_scalar_new,name="alpha")
    dalphaN      =    TrialFunction ( V_scalar_new)
    betaN        =    TestFunction(V_scalar_new)
    # Project the rerence solution into the new mesh.
    alphaN_0     =    Function(V_scalar_new,name="alpha")
    # Define the initial damage for the new mesh.
    alphaN_0     =    interpolate(alphaAux,V_scalar_new)
    # Boudary conditions.
    zero_v_new       =   Constant((0.,)*ndim)
    u_0_new          =   zero_v_new
    bc_boxbottomN    =   DirichletBC(V_vector_new,u_0_new,boundaries_new,4)
    bc_leftN         =   DirichletBC(V_vector_new.sub(0),0.0,boundaries_new,1)
    bc_rightN        =   DirichletBC(V_vector_new.sub(0),0.0,boundaries_new,2)
    bc_uN            =   [bc_boxbottomN,bc_leftN,bc_rightN]
    # Let us define the total energy of the system as the sum of elastic
    energy, dissipated energy due to the damage and external work due to body
    forces.
    if case == "Marigo":
        elastic_energy1_new =   0.5*inner(sigma(eps(uN),alphaN),eps(uN))*dxN
        elastic_energy2_new =   0.5*inner(sigma(eps(uN),alphaN),eps(uN))*dxN
    if case=="DMSF":
        Id_Tensor              =   as_matrix([[1,0],[0,1]])
        elastic_energy1_new =   (0.5*((lmbda+mu)*(tr(eps(uN))**2)*inner(
    Id_Tensor,Id_Tensor)))*dxN+((A(alphaN)+k_ell)*mu*inner(Dev(eps(uN)),Dev(eps
    (uN))))*dxN
        elastic_energy2_new =   (0.5*((lmbda+mu)*(tr(eps(uN))**2)*inner(
    Id_Tensor,Id_Tensor)))*dxN+((A(alphaN)+k_ell)*mu*inner(Dev(eps(uN)),Dev(eps
    (uN))))*dxN
    if case=="ShearComp":
        elastic_energy1_new =   0.5*inner(sigma(eps(uN),alphaN),eps(uN))*dxN
        elastic_energy2_new =   0.5/E*(inner(Dev(sigma(eps(uN),alphaN)),Dev(
    sigma(eps(uN),alphaN)))\
                                   - kappa*inner(Sph(sigma(eps(uN),alphaN)),Sph(
    sigma(eps(uN),alphaN))))*dxN
    # Dissipated energy.
    dissipated_energy_new = ( w ( alphaN) + ellv**2 *w1 * dot ( grad ( alphaN)
    , grad ( alphaN) ) ) * dxN
    # External work.
    external_work_new      = dot ( body_force, uN) * dxN
    # Definition of the total energy
    total_energy1_new = elastic_energy1_new + dissipated_energy_new-
    external_work_new
    total_energy2_new = elastic_energy2_new + dissipated_energy_new-
    external_work_new
    # Weak form of elasticity problem. This is the formal expression for the
    tangent problem which gives us the equilibrium equations.
    E_uN      =  derivative(total_energy1_new,uN,vN)
    E_alphaN =  derivative(total_energy2_new,alphaN,betaN)
    # Hessian matrix
    E_alpha_alphaN  =   derivative(E_alphaN,alphaN,dalphaN)
    # Writing tangent problems in term of test and trial functions for matrix
    assembly
    E_duN      = replace(E_uN,{uN:duN})
    E_dalphaN = replace(E_alphaN,{alphaN:dalphaN})
    # Once the tangent problems are formulated in terms of trial and text
    functions, we define the variatonal problems.
    # Variational problem for the displacement.
    problem_uN  =   LinearVariationalProblem(lhs(E_duN),rhs(E_duN),uN,bc_uN)
    # Define the classs Optimization Problem for then define the damage.
    # Variational problem for the damage (non-linear to use variational
    inequality solvers of petsc).
    class DamageProblemN(OptimisationProblem) :
        def __init__ (self):
            OptimisationProblem.__init__(self)
        # Objective vector
        def f(self,x):
            alphaN.vector()[:]=x
            return assemble(total_energy2_new)
        # Gradient of the objective function
        def F(self,b,x):
            alphaN.vector()[:]=x
            assemble(E_alphaN,tensor=b)
        # Hessian of the objective function
        def J(self,A,x):
            alphaN.vector()[:]=x
            assemble(E_alpha_alphaN,tensor=A)
    # Define the minimization problem using the class.
    problem_alphaN = DamageProblemN ( )
    # Set up the solvers
    solver_uN = LinearVariationalSolver ( problem_uN)
    solver_uN.parameters.update ( solver_LS_parameters)
    solver_alphaN = PETScTAOSolver ( )
    solver_alphaN.parameters.update ( solver_minimization_parameters)
    #  For the constraint minimization problem we require the lower and upper
    bound, "lbN" and "ubN".  They are initialized though interpolations.
    lbN = alphaN_0
    ubN = interpolate ( Expression ( "9.5", degree = 0), V_scalar_new)
    #---------------------------------------------------------------------------
    # ALTERNATE MINIIZATION.
    #———————————————————————————————————————————————————————————
    iter = 1; err_alphaN = 1; err_alpha_aux=1

    while err_alphaN > toll and iter < maxiter :
        alphaN.vector().gather(a0,np.array(range(V_scalar_new.dim()),"intc"))
        amin=alphaN.vector().min()
        amax=alphaN.vector().max()
        if MPI.COMM_WORLD.Get_rank() == 0:
            print("Job %d: itmesh=%-2d, Iteration:  %2d   , a0:[%.8g,%.8g],
alphaN:[%.8g,%.8g]" \
                    %(MPI.COMM_WORLD.Get_rank(),itmesh,iter,a0.min(),a0.max(),
amin,amax))
        # solve elastic problem
        solver_uN.solve ( )
        # solve damage problem via a constrained minimization algorithm.
        solver_alphaN.solve ( problem_alphaN, alphaN.vector ( ), lbN.vector (
), ubN.vector ( ) )
        alphaN.vector().get_local( )[alphaN.vector().get_local( ) > 0.95] =
0.95
        alphaN.vector().gather(a1, np.array(range(V_scalar_new.dim()), "intc")
)
        # Compute the norm of the the error vector.
        err_alphaN = np.linalg.norm(a1 - a0, ord = np.Inf)
        if C_L != 0.0:
            while err_alphaN>err_alpha_aux:
                alphaN_2 = C_L*alphaN_0+(1.0-C_L)*alphaN
                alphaN.assign(alphaN_2)
                alphaN.vector().gather(a1,np.array(range(V_scalar_new.dim()),
"intc"))
                err_alphaN = np.linalg.norm(a1-a0,ord=np.Inf)
        # Monitor the results for the new mesh.
        if MPI.COMM_WORLD.Get_rank() >= 0:
            print("Job %d: itmesh=%-2d, Iteration:  %2d   , Error: %2.8g,
alpha_max: %.8g" \
                    %(MPI.COMM_WORLD.Get_rank(),itmesh,iter,err_alphaN,alphaN.
vector().max()))
        # update the solution for the current alternate minimization iteration
.
        err_alpha_aux   =   err_alphaN
        alphaN_0.assign(alphaN)
        iter = iter + 1
    if MPI.COMM_WORLD.Get_rank() >= 0:
        print ("————————————————————————————————————————————————————————")
        print (" End of the alternate minimization in Remesh: %d " %(itmesh) )
        print ("————————————————————————————————————————————————————————")
    # Once a new damage has been obtained, we store it into an auxiliary
    variable "alphaAux"
    alphaAux    =   Function(V_scalar_new,name="alpha")
    alphaAux.assign(alphaN)
    # Store u,alpha.
    if MPI.COMM_WORLD.Get_rank() >= 0:
        file_u          << ( uN, 1.0 * itmesh)
        file_alpha      << ( alphaN   , 1.0 * itmesh)
    itmesh = itmesh + 1
    # Free memory for lists depending on the current mesh iteration
    del duN
    del vN
    del alphaN
    del dalphaN
    del betaN
    del alphaN_0
    del ubN
    del lbN
    del bc_uN
    del normal_v_new
    del boundaries_new
    del domain_new
    del V_vector_new
    del V_scalar_new
    del total_energy1_new
    del elastic_energy1_new
    del total_energy2_new
    del elastic_energy2_new
    del external_work_new
    del dissipated_energy_new
    del E_uN
    del E_alphaN
    del E_alpha_alphaN
    del E_duN
    del E_dalphaN
    del solver_uN
    del problem_uN
    del problem_alphaN
    del dsN, dxN
    del DamageProblemN
    del a0
    del a1
    del a1
#------------------------------------------------------------------------
# THE MAIN LOOP FOR REMESHING GAS FINISHED.
#------------------------------------------------------------------------
print("------------------------------------------------------------------------")
print("                  Geometry with cavity is finished.                     ")
print("------------------------------------------------------------------------")
#------------------------------------------------------------------------
# FREE MEMORY
#------------------------------------------------------------------------
del u, du, v
del alpha, dalpha, beta, alpha_0
del alpha_error
del file_alpha
del file_u
del lb, ub
del solver_u
del solver_alpha
del alphaAux
del V_vector
del V_scalar
del normal_v
del mesh, boundaries
#------------------------------------------------------------------------
```

```
#   End of the main program.
#------------------------------------------------------------------------
```

./codes/Cavity_2D/Cavity2D.py

B.1.2 Parameters.py

```
#------------------------------------------------------------------------
#  IMPORT LIBRARIES TO GET THE CODE WORKING.
#------------------------------------------------------------------------
from dolfin import *
from mshr import *
import sys, os, sympy, shutil, math
import numpy as np
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt
import socket
import datetime
from ufl import replace
from mpi4py import MPI
from inspect import currentframe, getframeinfo,stack
#------------------------------------------------------------------------
# MESH
#------------------------------------------------------------------------
mesh     =   RectangleMesh(Point(-1500.0,-500.0),Point(1500.0,500.0),750,250,"
    crossed")
#------------------------------------------------------------------------
# PARAMETERS
#------------------------------------------------------------------------
itmesh   =   1
NstepW   =   15
model    =   "quad"
w1       =   1.0e5
w11      =   1.0e5
C_L      =   0.
kappa    =   1.0
#Diferent Models to Damage problem, for example: "Marigo";"ShearComp"; "DMSF"
case     =   "Marigo"
#------------------------------------------------------------------------
# MATERIAL CONSTANTS
#------------------------------------------------------------------------
E        =   29e9
nu       =   0.3
mu       =   E/(2.0*(1.0+ nu))
lmbda    =   E*nu/(1.0-nu**2)
# In this case this quantity contains the density, so gravity =  rho * g, with
     g the gravity acceleration.
rho      =   2.7e3
g        =   9.8
gravity  =   rho*g
k_ell    =   Constant(1.e-6) #residual stiffness
# Definition of \ell parameter
h        =   CellDiameter(mesh) # diameters of all elements
hmin     =   mesh.hmin() # minimum of all diameters
hmax     =   mesh.hmax() # maximun of all diameters
```

```
ellv     =    5.0*hmin     #\ell parameter
#Body force
body_force = Constant( (0.0, -gravity) )
#-----------------------------------------------------------------
# NUMERICAL PARAMETERS OF THE ALTERNATE MINIMIZATION.
#-----------------------------------------------------------------
maxiter = 1e3
toll    = 1e-5
#-----------------------------------------------------------------
# THE FILES ARE STORED IN A FOLDER NAMED "modelname".
#-----------------------------------------------------------------
date        =    datetime.datetime.now().strftime("%m-%d-%y_%H.%M.%S")
where       =    socket.gethostname()
modelname   =    "[C_L=0]_[case=%s]_[model=%s]_[w1=%g]_[w11=%g]_[ell=%.2g]_[
    kappa=%.2g]_[Date=%s]_[Where=%s]"%(case,model,w1,w11,ellv,kappa,date,where)
print('modelname='+modelname)
# others
regenerate_mesh =    True
savedir         =    "results/%s"%(modelname)
print('savedir='+savedir)
#-----------------------------------------------------------------
# PARAMETERS IN THE GEOMETRY
#-----------------------------------------------------------------
ndim = mesh.geometry().dim() #get number of space dimensions
# Define boundary sets for boundary conditions
class Left(SubDomain):
    def inside(self,x,on_boundary):
        return near(x[0],-1500.)
class Right(SubDomain):
    def inside(self,x,on_boundary):
        return near(x[0],1500.)
class Top(SubDomain):
    def inside(self,x,on_boundary):
        return near(x[1],500.)
class Bottom(SubDomain):
    def inside(self,x,on_boundary):
        return near(x[1],-500.)
# Initialize sub-domain instances
left     =    Left()
right    =    Right()
top      =    Top()
bottom   =    Bottom()
# define meshfunction to identify boundaries by numbers
boundaries = MeshFunction("size_t",mesh,mesh.topology().dim()-1)
boundaries.set_all(0)
left.mark  (boundaries,1) # mark left as 1
right.mark (boundaries,2) # mark right as 2
top.mark   (boundaries,3) # mark top as 3
bottom.mark(boundaries,4) # mark bottom as 4
# normal vectors
normal_v    =    FacetNormal(mesh)
#-----------------------------------------------------------------
# CREATE FUNCTION SPACE FOR 2D ELASTICITY AND DAMAGE
#-----------------------------------------------------------------
V_vector    =    VectorFunctionSpace(mesh,"CG",1)
V_scalar    =    FunctionSpace(mesh,"CG",1)
```

```
V_tensor      =    TensorFunctionSpace(mesh,"DG",0)
#----------------------------------------------------------------
# BOUNDARY CONDITIONS.
#----------------------------------------------------------------
zero_v             =    Constant((0.,)*ndim)
u_0                =    zero_v
bc_left            =    DirichletBC(V_vector.sub(0),0.0,boundaries,1)
bc_right           =    DirichletBC(V_vector.sub(0),0.0,boundaries,2)
bc_boxbottom       =    DirichletBC(V_vector,u_0,boundaries,4)
bc_u               =    [bc_boxbottom,bc_left,bc_right]
```

./codes/Cavity_2D/Parameters.py

B.2 Code for Cavity in 3D

B.2.1 Cavity3D.py

```
#----------------------------------------------------------------
# IMPORT ALL PARAMETERS FROM Parameters.py AND AuxFunctions.py.
#----------------------------------------------------------------
from parameters import *
from AuxFunctions import *
# read paramaters from command line
parameters.parse()
# set some dolfin specific parameters
parameters["form_compiler"]["optimize"]     = True
parameters["form_compiler"]["cpp_optimize"] = True
parameters["form_compiler"]["representation"] = "uflacs"
parameters["allow_extrapolation"] = True
# The minimization procedure requires parameters to get a suitable performance
    . The following is a suitable set of arrangements.
solver_minimization_parameters =  {"method" : "gpcg",
                                    "linear_solver" : "gmres",
                                #--------------------------------
                                # These are parameters for optimization
                                #--------------------------------
                                "line_search": "armijo",
                                "preconditioner" : "hypre_euclid",
                                "maximum_iterations" :200,
                                "error_on_nonconvergence": False,
                                #--------------------------------
                                # These are parameters for linear solver
                                #--------------------------------
                                "krylov_solver" : {
                                    "maximum_iterations" : 200,
                                    "nonzero_initial_guess" : True,
                                    "report" : True,
                                    "monitor_convergence" : False,
                                    "relative_tolerance" : 1e-8
                                }
                               }
# The linear solver requires parameters to get a suitable performance. The
    following is a suitable set of arrangements.
solver_LS_parameters =  {"linear_solver" : "cg",
                                "symmetric" : True,
                                  "preconditioner" : "jacobi",
                                  "krylov_solver" : {
                                       "report" : True,
                                       "monitor_convergence" : False,
                                       "relative_tolerance" : 1e-8
                                       }
                                  }
#------------------------------------------------------------------------------
# DEFINE THE OPERATORS.
#------------------------------------------------------------------------------
# Constitutive functions of the damage model
def w(alpha):
    if model==1:
        return w11*alpha
    if model==2:
        return w11*alpha**2
    if model==3:
        return w11*(1-(1-alpha)**2)
    if model==4:
        return w11*alpha
def A(alpha):
    if model==1 or model ==2:
        return (1-alpha)**2
    if model==3:
        return (1-alpha)**4
    if model==4:
        return (1-alpha)/(1+alpha)
# Define th Deviator and Spheric Tensors
def Dev(Tensor):
    return Tensor-Sph(Tensor)
def Sph(Tensor):
    Tensor2=as_matrix([[1,0,0],[0,1,0],[0,0,1]])
    return (inner(Tensor,Tensor2)/inner(Tensor2,Tensor2))*Tensor2
# Strain and stress in free damage regime.
def eps(v):
    return sym(grad(v))
def sigma_0(eps):
    return 2.0*mu*(eps)+lmbda*tr(eps)*Identity(ndim)
# modification of the Young modulus.
def sigma(eps,alpha):
    return (A(alpha)+k_ell)*sigma_0(eps)
#  Define the energies.
def energy_w(u,alpha):
    Es=eps(u)-tr(eps(u))/3*Identity(ndim)
    Eploc=tr(eps(u))
    return ((lmbda/2+mu/3)*Eploc**2+mu*inner(Es,Es))
def energy_dev(u,alpha):
    Es=Dev(sigma(eps(u),alpha))
    return (0.5*inner(Es,eps(u)))
def energy_sph(u,alpha):
    Eploc=Sph(sigma(eps(u),alpha))
    return (0.5*inner(Eploc,eps(u)))
def energy_dis (alpha,ell,w1):
    w_a=w(alpha)
    grad_a=ell**2*w1*dot(grad(alpha),grad(alpha))
    return w_a+grad_a
```

```
def energy_p(u,body_force):
    return -inner(body_force,u)
def energy_E(u,alpha,body_force):
    return energy_dev(u,alpha)+\
           energy_sph(u,alpha)+\
           energy_p(u,body_force)
# Function for boundary condition
def k_r(u):
    k=1.0e9
    return k*u
#---------------------------------------------------------------
#  MESH WITHOUT CAVITY.
#---------------------------------------------------------------
# Define the function, test and trial fields.
# "u, du, v" are vectorial expressions.
u    =    Function(V_vector,name="u")
du   =    TrialFunction(V_vector)
v    =    TestFunction(V_vector)
# "alpha, dalpha, beta" are scalar.
alpha    =    Function(V_scalar,name="alpha")
alpha0   =    Function(V_scalar,name="alpha")
dalpha   =    TrialFunction(V_scalar)
beta     =    TestFunction(V_scalar)
# Define energy functions.
W_energy     =    Function(V_scalar,name="energy_w")
dev_energy   =    Function(V_scalar,name="energy_dev")
div_energy   =    Function(V_scalar,name="energy_sph")
E_energy     =    Function(V_scalar,name="energy_E")
p_energy     =    Function(V_scalar,name="energy_p")
dis_energy   =    Function(V_scalar,name="energy_dis")
# Define stress and strain functions.
stressG  =    Function(V_tensor,name="sigma")
strainG  =    Function(V_tensor,name="epsilon")
# Define the alpha auxiliar function.
alphaAux     =    Function(V_scalar,name="alpha")
# Define the function "alpha_error" to measure relative error.
alpha_error  =    Function(V_scalar)
# Interpolate the initial condition for the damage variable "alpha".
alpha_0  =    interpolate(Expression("0.",degree=2),V_scalar)
if FH=="Initial_Damage":
    a_1  =    Constant(900.0)
    a_2  =    Constant(900.0)
    a_3  =    Constant(121.0)
    for i in range(0,5):
        for j in range(0,5):
            alpha_0_aux=interpolate(Expression("(pow(cos(theta)*(x[0]-65*j)+
    sin(theta)*(x[2]-(30+30*i)),2)/a_1+pow((x[1]),2)/a_2+\
                                                        pow(-sin(theta)*(x[0]-65*j
    )+cos(theta)*(x[2]-(30+30*i)),2)/a_3)<=1?0.95:0.",
                                                        degree=2,a_1=a_1,a_2=a_2,
    a_3=a_3,theta=pi/4,j=j,i=i),V_scalar)
            alpha_0.vector()[:]  =    alpha_0.vector()+alpha_0_aux.vector()
alpha.assign(alpha_0)
# Define ds and dx.
ds   =    Measure('ds',domain=mesh,subdomain_data=boundaries)
dx   =    Measure('dx',domain=mesh)
```

```
# Let us define the total energy of the system as the sum of elastic energy,
    dissipated energy due to the damage and external work due to body forces.
# Elastic energy.
elastic_energy1 =   0.5*inner(sigma(eps(u),alpha),eps(u))*dx
elastic_energy2 =   0.5/E*(inner(Dev(sigma(eps(u),alpha)),Dev(sigma(eps(u),
    alpha)))\
                         -2.0/3.0*kappa2*inner(Sph(sigma(eps(u),alpha)),Sph(
    sigma(eps(u),alpha))))*dx
# External work.
external_work   =   dot(body_force,u)*dx
# Neumand BC.
external_bc =   0.5*((dot(k_r(u),normal_v)*dot(u,normal_v))*ds(BOXMIDX1)
                +(dot(k_r(u),normal_v)*dot(u,normal_v))*ds(BOXMIDX2)
                +(dot(k_r(u),normal_v)*dot(u,normal_v))*ds(BOXMIDY1)
                +(dot(k_r(u),normal_v)*dot(u,normal_v))*ds(BOXMIDY2)
                -dot(g_bc_zz*normal_v,u)*ds(BOXMIDX1)
                -dot(g_bc_zz*normal_v,u)*ds(BOXMIDX2)
                -dot(g_bc_zz*normal_v,u)*ds(BOXMIDY1)
                -dot(g_bc_zz*normal_v,u)*ds(BOXMIDY2))
# Dissipated energy.
dissipated_energy   =   (w(alpha)+ell**2*w1*dot(grad(alpha),grad(alpha)))*dx
# Definition of the total energy
total_energy1   =   elastic_energy1+dissipated_energy-external_work+
    external_bc
total_energy2   =   elastic_energy2+dissipated_energy-external_work+
    external_bc
# Weak form of damage problem. This is the formal expression for the tangent
    problem which gives us the equilibrium equations.
E_u     =   derivative(total_energy1,u,v)
E_alpha =   derivative(total_energy2,alpha,beta)
# Hessian matrix
E_alpha_alpha   =   derivative(E_alpha,alpha,dalpha)
# Writing tangent problems in term of test and trial functions for matrix
    assembly.
E_du        =   replace(E_u,{u:du})
E_dalpha    =   replace(E_alpha,{alpha:dalpha})
# Once the tangent problems are formulated in terms of trial and text
    functions, we define the variatonal problems.
# Variational problem for the displacement.
problem_u   =   LinearVariationalProblem(lhs(E_du),rhs(E_du),u,bc_u)
# Define the classs Optimization Problem for then define the damage.
# Variational problem for the damage (non-linear to use variational inequality
     solvers of petsc).
class DamageProblem(OptimisationProblem):
    def __init__(self):
        OptimisationProblem.__init__(self)
    # Objective vector
    def f(self,x):
        alpha.vector()[:]=x
        return assemble(total_energy2)
    # Gradient of the objective function
    def F(self,b,x):
        alpha.vector()[:]=x
        assemble(E_alpha,tensor=b)
    # Hessian of the objective function
    def J(self,A,x):
        alpha.vector()[:]= x
        assemble(E_alpha_alpha,tensor=A)
# define the minimization problem using the class.
problem_alpha   =   DamageProblem()
# Set up the solvers. Define the object for solving the displacement problem,
    "solver_u".
solver_u    =   LinearVariationalSolver(problem_u)
# Get the set of paramters for the class "solver_u". This only requires the
    solution of linear system solver.
solver_u.parameters.update(solver_LS_parameters)
# Define the corresponding object, "solver_alpha".
# The object associated to minimization is created.
solver_alpha    =   PETScTAOSolver()
# Get the set of paramters for the class "solver_alpha". This requires the
    solution of a minimization problem.
solver_alpha.parameters.update(solver_minimization_parameters)
# As the optimization is a constrained type we need to provide the
    corresponding lower and upper  bounds.
lb  =   interpolate(Expression("0.0",degree=0),V_scalar)
ub  =   interpolate(Expression("0.95",degree=0),V_scalar)
lb.vector()[:]  =   alpha.vector()
# Crete the files to store the solutions
file_alpha      =   File(savedir + "/alpha.pvd")
file_energW     =   File(savedir + "/energy_w.pvd")
file_energDev   =   File(savedir + "/energy_dev.pvd")
file_energDiv   =   File(savedir + "/energy_sph.pvd")
file_energDis =   File(savedir + "/energy_dis.pvd")
file_energP     =   File(savedir + "/energy_p.pvd")
file_energE     =   File(savedir + "/energy_E.pvd")
file_u          =   File(savedir + "/u.pvd")
file_sigma      =   File(savedir+"/sigma.pvd")
file_epsilon    =   File(savedir+"/epsilon.pvd")
#------------------------------------------------------------------------
# ELASTICITY PROBLEM.
#------------------------------------------------------------------------
if caso=="Elasticity":
        solver_u.solve()
        print ("
    ---------------------------------------------------------------------")
        print ("    End of the Elasticity Problem in Remesh: %d     " %( 0 ))
        print ("
    ---------------------------------------------------------------------")
#------------------------------------------------------------------------
# ALTERNATE MINIIZATION.
#------------------------------------------------------------------------
# Initialization
if caso =="Damage":
    iter = 1; err_alpha = 1
    a0 = Vector(MPI.COMM_SELF)
    a1 = Vector(MPI.COMM_SELF)
    # Iterations of the alternate minimization stop if an error limit is
    reached or a maximim number of iterations have been done.
    while True and err_alpha > toll and iter < maxiter :
        alpha.vector().gather(a0,np.array(range(V_scalar.dim()),"intc"))
        if MPI.COMM_WORLD.Get_rank()==0:
            print("Job %d: Iteration:  %2d, a0:[%.8g,%.8g]"%(MPI.COMM_WORLD.Get_rank(),iter,a0.min(),a0.max()))
        # solve elastic problem
        solver_u.solve()
        # solve damage problem via a constrained minimization algorithm.
        solver_alpha.solve(problem_alpha,alpha.vector(),lb.vector(),ub.vector())
        alpha.vector().get_local()[alpha.vector().get_local()>0.95]=0.95
        alpha.vector().gather(a1,np.array(range(V_scalar.dim()),"intc"))
        # Compute the norm of the the error vector.
        err_alpha = np.linalg.norm(a1 - a0, ord = np.Inf) #np.linalg.norm (alpha_error.vector ( ).get_local ( ),ord = np.Inf)
        # Numerical Improve.
        if C_L != 0.0:
            while err_alpha>err_alpha_aux:
                alpha_2=C_L*alpha_0+(1.0-C_L)*alpha
                alpha.assign(alpha_2)
                alpha.vector().gather(a1,np.array(range(V_scalar.dim()),"intc"))
                err_alpha=np.linalg.norm(a1-a0,ord=np.Inf)
        # monitor the results
        if MPI.COMM_WORLD.Get_rank()>=0:
            print ("Iteration:  %2d, Error: %2.8g, alpha_max: %.8g" %(iter,err_alpha,alpha.vector().max()))
        # update the solution for the current alternate minimization iteration.
        err_alpha_aux   =   err_alpha
        alpha_0.assign(alpha)
        iter = iter + 1
    # updating the lower bound with the solution of the solution corresponding to the current global iteration, it is for accounting for the irreversibility.
    lb.vector()[:]=alpha.vector()
    print ("-----------------------------------------------------------------")
    print (" End of the alternate minimization without cavity. ")
    print ("-----------------------------------------------------------------")
#-----------------------------------------------------------------------------------
#  END ALTERNATE MINIMIZATION.
#-----------------------------------------------------------------------------------
# Store u,alpha.
if MPI.COMM_WORLD.Get_rank()>=0:
    file_u  <<  (u,0.)
    if caso=="Damage":
        file_alpha  <<  (alpha,0.)
# Store strain and stress.
strain  =   eps(u)
stress  =   project(sigma(strain,alpha),V_tensor,solver_type="cg",preconditioner_type="petsc_amg")
stressG.assign(stress)
strainG.assign(project(strain,V_tensor,solver_type="cg", preconditioner_type="petsc_amg"))
if MPI.COMM_WORLD.Get_rank()>= 0:
    file_sigma      <<  (stressG,0.)
    file_epsilon    <<  (strainG,0.)
# Store energies.
```

```
W_energy.assign(project(energy_w(u,alpha),V_scalar,solver_type='cg',
    preconditioner_type="petsc_amg"))
dev_energy.assign(project(energy_dev(u,alpha),V_scalar,solver_type='cg',
    preconditioner_type="petsc_amg"))
div_energy.assign(project(energy_sph(u,alpha),V_scalar,solver_type='cg',
    preconditioner_type="petsc_amg"))
E_energy.assign(project(energy_E(u,alpha,body_force),V_scalar,solver_type='cg'
    ,preconditioner_type="petsc_amg"))
p_energy.assign(project(energy_p(u,body_force),V_scalar,solver_type='cg',
    preconditioner_type="petsc_amg"))
if caso=="Damage":
     dis_energy.assign(project(energy_dis(alpha,ell,w1),V_scalar,solver_type='
    cg',preconditioner_type="petsc_amg"))
# Control if energies are too small
W_energy.vector().get_local()[W_energy.vector().get_local()<1e-12]=0.0
dev_energy.vector().get_local()[dev_energy.vector().get_local()<1e-12]=0.0
div_energy.vector().get_local()[div_energy.vector().get_local()<1e-12]=0.0
E_energy.vector().get_local()[E_energy.vector().get_local()<1e-12]=0.0
p_energy.vector().get_local()[p_energy.vector().get_local()<1e-12]=0.0
if caso=="Damage":
     dis_energy.vector().get_local()[dis_energy.vector().get_local()<1e-12]=0.0
if MPI.COMM_WORLD.Get_rank()>=0:
     file_energW     <<  (W_energy,0.)
     file_energDev   <<  (dev_energy,0.)
     file_energDiv   <<  (div_energy,0.)
     file_energE     <<  (E_energy,0.)
     file_energP     <<  (p_energy,0.)
     if caso == "Damage":
          file_energDis   <<  (dis_energy,0.)
# Store the damage for this geometry
alphaAux.assign(alpha)
alpha0.assign(alpha)
print ("------------------------------------------------------------------")
print ("               Geometry without cavity is finished.              ")
print ("------------------------------------------------------------------")
# Remove previous integrating factors "dx, ds"
del ds, dx
#----------------------------------------------------------------
#  MESH WITH CAVITY.
#----------------------------------------------------------------
# Start loop over new geometries. These are obtained from a sequence of
    geometries which are obtained from an external folder. The number of
    external files is "NstepW" and the call is driven by the counter "itmesh".
#  Starting the loop of the mesh sequence. It is driven by the index "itmesh".
while itmesh <= NstepW :
     a0 = Vector(MPI.COMM_SELF)
     a1 = Vector(MPI.COMM_SELF)
     # Read mesh from a sequence of meshes generated externally.
     if malla=="v3":
          mesh_new=Mesh("Meshes/V3/"+meshname+str(itmesh)+".xml.gz")
     if malla=="esmeralda":
          mesh_new=Mesh("Meshes/Esmeralda/"+meshname+str(itmesh)+".xml.gz")
     if malla=="sub6":
          mesh_new=Mesh("Meshes/Sub6/"+meshname+str(itmesh)+".xml.gz")
     if malla=="nuevonivel":
          mesh_new=Mesh("Meshes/Nuevo_Nivel/"+meshname+str(itmesh)+".xml.gz")
    # Read files for interpolation
    if malla=="v3":
        with open("Meshes/V3/"+meshname+str(itmesh)+"_"+str(0)+".txt","r") as
file:
            GVertexI_2_GVertex0=eval(file.readline())
    if malla=="esmeralda":
        with open("Meshes/Esmeralda_2/"+meshname+str(itmesh)+"_"+str(0)+".txt"
,"r") as file:
            GVertexI_2_GVertex0=eval(file.readline())
    if malla=="sub6":
        with open("Meshes/Sub6/"+meshname+str(itmesh)+"_"+str(0)+".txt","r")
as file:
            GVertexI_2_GVertex0=eval(file.readline())
    if malla=="nuevonivel":
        with open("Meshes/Nuevo_Nivel/"+meshname+str(itmesh)+"_"+str(0)+".txt"
,"r") as file:
            GVertexI_2_GVertex0=eval(file.readline())
    if malla=="v3":
        with open("Meshes/V3/"+meshname+str(0)+"_"+str(itmesh)+".txt","r") as
file:
            GVertex0_2_GVertexI=eval(file.readline())
    if malla=="esmeralda":
        with open("Meshes/Esmeralda_2/"+meshname+str(0)+"_"+str(itmesh)+".txt"
,"r") as file:
            GVertex0_2_GVertexI=eval(file.readline())
    if malla=="sub6":
        with open("Meshes/Sub6/"+meshname+str(0)+"_"+str(itmesh)+".txt","r")
as file:
            GVertex0_2_GVertexI=eval(file.readline())
    if malla=="nuevonivel":
        with open("Meshes/Nuevo_Nivel/"+meshname+str(0)+"_"+str(itmesh)+".txt"
,"r") as file:
            GVertex0_2_GVertexI=eval(file.readline())
    mesh_new.init()
    # Read boundaries for new mesh
    if malla=="v3":
        boundaries_new=MeshFunction('size_t',mesh_new,"Meshes/V3/"+meshname+
str(itmesh)+"_faces.xml.gz")
    if malla=="esmeralda":
        boundaries_new=MeshFunction('size_t',mesh_new,"Meshes/Esmeralda_2/"+
meshname+str(itmesh)+"_faces.xml.gz")
    if malla=="sub6":
        boundaries_new=MeshFunction('size_t', mesh_new,"Meshes/Sub6/"+meshname
+str(itmesh)+"_faces.xml.gz")
    if malla=="nuevonivel":
        boundaries_new=MeshFunction('size_t',mesh_new,"Meshes/Nuevo_Nivel/"+
meshname+str(itmesh)+"_faces.xml.gz")
    MeshFunction("size_t", mesh, mesh.topology().dim()-1)
    # Define the new ds and dx.
    dsN =    Measure('ds',domain=mesh_new,subdomain_data=boundaries_new)
    dxN =    Measure('dx',domain=mesh_new)
    #normal vectors
    normal_v_new = FacetNormal ( mesh_new)
    # Create new function spaces
    V_vector_new     =    VectorFunctionSpace(mesh_new,"CG",1)
    V_scalar_new     =    FunctionSpace(mesh_new,"CG",1)
```

```
 V_tensor_new     =    TensorFunctionSpace(mesh_new,"DG",0)
 strainGN         =    Function(V_tensor_new,name="epsilon")
 stressGN         =    Function(V_tensor_new,name="sigma")
#-------------------------------------------------------------------------
# REMARK: To generate a sequence of plots in paraview the name
# of the variable must be the same. It is achieved by including
# name="alpha" at the moment of the definition of the structure "alpha".
#
#  << alphaN =  Function ( V_scalar_new, name="alpha") >>
#
# The same definition needs to be done for displacement "u" and
# other arrays as the difference of damage without cavity and
# damage with cavity, for example "alphaDiff".
#-------------------------------------------------------------------------
# Define the function, test and trial fields
uN             =    Function(V_vector_new,name="u")
duN            =    TrialFunction(V_vector_new)
vN             =    TestFunction(V_vector_new)
 alphaN        =    Function(V_scalar_new,name="alpha")
 alphaN_2      =    Function(V_scalar_new,name="alpha")
 dalphaN       =    TrialFunction(V_scalar_new)
 betaN         =    TestFunction(V_scalar_new)
# Define energy functions.
 W_energyN     =   Function(V_scalar_new,name="energy_w")
 dev_energyN   =   Function(V_scalar_new,name="energy_dev")
 div_energyN   =   Function(V_scalar_new,name="energy_sph")
 E_energyN     =   Function(V_scalar_new,name="energy_E")
 p_energyN     =   Function(V_scalar_new,name="energy_p")
 dis_energyN   =   Function(V_scalar_new,name="energy_dis")
# Define the initial damage for the new mesh.
 alphaN_0      =    interpolate(Expression("0.0",degree=1),V_scalar_new)
# Interpolate the previous damage.
 Interpola_In_Out(alphaAux,alphaN_0,GVertexI_2_GVertex0)
 alphaN_0.vector().get_local()[alphaN_0.vector().get_local()<1e-12]=0.0
 alphaN_0.vector().get_local( )[alphaN_0.vector().get_local()>0.95]=0.95
 alphaN.assign(interpolate(alphaN_0,V_scalar_new))
# Boudary conditions.
 faceCornerBottom.mark(boundaries_new, FACECORNERBOTTOM)
 bc_facecornerbottomN      =    DirichletBC(V_vector_new.sub(2),Constant(0.0),
boundaries_new,FACECORNERBOTTOM)
 bc_boxmidx1N              =    DirichletBC(V_vector_new.sub(0),Constant(0.0),
boundaries_new,BOXMIDX1)
 bc_boxmidx2N              =    DirichletBC(V_vector_new.sub(0),Constant(0.0),
boundaries_new,BOXMIDX2)
 bc_boxmidy1N              =    DirichletBC(V_vector_new.sub(1),Constant(0.0),
boundaries_new,BOXMIDY1)
 bc_boxmidy2N              =    DirichletBC(V_vector_new.sub(1),Constant(0.0),
boundaries_new,BOXMIDY2)
 bc_boxbottomN             =    DirichletBC(V_vector_new.sub(2),Constant(0.0),
boundaries_new,BOXBOTTOM)
 bc_uN                     =    [bc_boxbottomN,bc_facecornerbottomN]
# Let us define the total energy of the system as the sum of elastic
energy, dissipated energy due to the damage and external work due to body
forces.
 elastic_energy1_new =    0.5*inner(sigma(eps(uN),alphaN),eps(uN))*dxN
```

```
    elastic_energy2_new =   0.5/E*(inner(Dev(sigma(eps(uN),alphaN)),Dev(sigma(eps(uN),alphaN)))\
                                   -2.0/3.0*kappa2*inner(Sph(sigma(eps(uN),alphaN)),Sph(sigma(eps(uN),alphaN))))*dxN
    # External work.
    external_work_new   =   dot(body_force,uN)*dxN
    # Neuman BC.
    external_bc_new =   (0.5*((dot(k_r(uN),normal_v_new)*dot(uN,normal_v_new))*dsN(BOXMIDX1)
                          +(dot(k_r(uN),normal_v_new)*dot(uN,normal_v_new))*dsN(BOXMIDX2)
                          +(dot(k_r(uN),normal_v_new)*dot(uN,normal_v_new))*dsN(BOXMIDY1)
                          +(dot(k_r(uN),normal_v_new)*dot(uN,normal_v_new))*dsN(BOXMIDY2))
                          -dot(g_bc_zz*normal_v_new,uN)*dsN(BOXMIDX1)
                          -dot(g_bc_zz*normal_v_new,uN)*dsN(BOXMIDX2)
                          -dot(g_bc_zz*normal_v_new,uN)*dsN(BOXMIDY1)
                          -dot(g_bc_zz*normal_v_new,uN)*dsN(BOXMIDY2))
    # Dissipated energy.
    dissipated_energy_new   =   (w(alphaN)+ell**2*w1*dot(grad(alphaN),grad(alphaN)))*dxN
    # Definition of the total energy
    total_energy1_new   =   elastic_energy1_new+dissipated_energy_new-external_work_new+external_bc_new
    total_energy2_new   =   elastic_energy2_new+dissipated_energy_new-external_work_new+external_bc_new
    # Weak form of elasticity problem. This is the formal expression for the tangent problem which gives us the equilibrium equations.
    E_uN        =   derivative(total_energy1_new,uN,vN)
    E_alphaN    =   derivative(total_energy2_new,alphaN,betaN)
    # Hessian matrix
    E_alpha_alphaN  =   derivative(E_alphaN,alphaN,dalphaN)
    # Writing tangent problems in term of test and trial functions for matrix assembly
    E_duN       =   replace(E_uN,{uN:duN})
    E_dalphaN   =   replace(E_alphaN,{alphaN:dalphaN})
    # Once the tangent problems are formulated in terms of trial and text functions, we define the variatonal problems.
    # Variational problem for the displacement.
    problem_uN  =   LinearVariationalProblem(lhs(E_duN),rhs(E_duN),uN,bc_uN)
    # Define the classs Optimization Problem for then define the damage.
    # Variational problem for the damage (non-linear to use variational inequality solvers of petsc).
    class DamageProblemN(OptimisationProblem):
        def __init__(self):
            OptimisationProblem.__init__(self)
        # Objective vector
        def f(self,x):
            alphaN.vector()[:]=x
            return assemble(total_energy2_new)
        # Gradient of the objective function
        def F(self,b,x):
            alphaN.vector()[:]=x
            assemble(E_alphaN, tensor = b)
        # Hessian of the objective function
    def J(self,A,x):
        alphaN.vector()[:]=x
        assemble(E_alpha_alphaN,tensor=A)
# Define the minimization problem using the class.
problem_alphaN  =  DamageProblemN()
# Set up the solvers
solver_uN = LinearVariationalSolver(problem_uN)
solver_uN.parameters.update(solver_LS_parameters)
solver_alphaN = PETScTAOSolver()
solver_alphaN.parameters.update(solver_minimization_parameters)
#  For the constraint minimization problem we require the lower and upper bound, "lbN" and "ubN".  They are initialized though interpolations.
lbN = alphaN_0
ubN = interpolate ( Expression ( "0.95", degree = 0), V_scalar_new)
#----------------------------------------------------------------------------
# ALTERNATE MINIIZATION.
#----------------------------------------------------------------------------
if caso == "Elasticity":
    solver_uN.solve()
    print ("----------------------------------------------------------------------")
    print ("    End of the Elasticity Problem in Remesh: %d    " %( itmesh ))
    print ("----------------------------------------------------------------------")
if caso == "Damage":
    iter = 1; err_alphaN = 1; err_alpha_aux=1
    count_arreglo=0
    while err_alphaN>toll and iter<maxiter:
        alphaN.vector().gather(a0,np.array(range(V_scalar_new.dim()),"intc"))
        amin=alphaN.vector().min()
        amax=alphaN.vector().max()
        if MPI.COMM_WORLD.Get_rank() == 0:
            print("Job %d: itmesh=%-2d, Iteration:  %2d   , a0:[%.8g,%.8g], alphaN:[%.8g,%.8g]" \
                  %(MPI.COMM_WORLD.Get_rank(),itmesh,iter,a0.min(),a0.max(),amin,amax))
        # solve elastic problem
        solver_uN.solve()
        # solve damage problem via a constrained minimization algorithm.
        solver_alphaN.solve(problem_alphaN,alphaN.vector(),lbN.vector(),ubN.vector())
        alphaN.vector().get_local()[alphaN.vector().get_local()>0.95]=0.95
        alphaN.vector().gather(a2,np.array(range(V_scalar_new.dim()),"intc"))
        # Compute the norm of the the error vector.
        err_alphaN = np.linalg.norm(a2 - a0, ord = np.Inf)
        if C_L != 0.0:
            while err_alphaN > err_alpha_aux:
                alphaN_2 = C_L*alphaN_0+(1.0-C_L)*alphaN
                alphaN.assign(alphaN_2)
                alphaN.vector().gather(a2, np.array(range(V_scalar_new.dim()), "intc"))
                err_alphaN = np.linalg.norm(a2 - a0, ord = np.Inf)
        # Monitor the results for the new mesh
```

```
            if MPI.COMM_WORLD.Get_rank() >= 0:
                print ("Job %d: itmesh=%-2d, Iteration:  %2d   , Error: %2.8g, alpha_max: %.8g" \
                        % ( MPI.COMM_WORLD.Get_rank(),itmesh, iter, err_alphaN, alphaN.vector ( ).max ( ) ))
            # update the solution for the current alternate minimization iteration.
            err_alpha_aux   =   err_alphaN
            alphaN_0.assign(alphaN)
            iter = iter + 1
        if MPI.COMM_WORLD.Get_rank() >= 0:
            print ("----------------------------------------------------------------------------")
            print ("    End of the alternate minimization in Remesh: %d    " %( itmesh ))
            print ("----------------------------------------------------------------------------")
    # Interpolation.
    Interpola_In_Out(alphaN, alphaAux, GVertex0_2_GVertexI)
    # Store strain and stress.
    strainN =   eps(uN)
    stressN =   project(sigma(strainN,alpha),V_tensor_new,solver_type='cg',preconditioner_type="petsc_amg")
    stressGN.assign(stressN)
    strainGN.assign(project(strainN,V_tensor_new,solver_type='cg',preconditioner_type="petsc_amg"))
    if MPI.COMM_WORLD.Get_rank()>=0:
        file_u          <<  (uN,1.0*itmesh)
        file_sigma      <<  (stressGN,1.0*itmesh)
        file_epsilon    <<  (strainGN,1.0*itmesh)
        if caso=="Damage":
            file_alpha  <<  (alphaN,1.0*itmesh)
    # Eval the energy
    W_energyN.assign(project(energy_w(uN,alphaN),V_scalar_new,solver_type='cg',preconditioner_type="petsc_amg"))
    dev_energyN.assign(project(energy_dev(uN,alphaN),V_scalar_new,solver_type='cg',preconditioner_type="petsc_amg"))
    div_energyN.assign(project(energy_sph(uN,alphaN),V_scalar_new,solver_type='cg',preconditioner_type="petsc_amg"))
    E_energyN.assign(project(energy_E(uN,alphaN,body_force),V_scalar_new,solver_type='cg',preconditioner_type="petsc_amg"))
    p_energyN.assign(project(energy_p(uN,body_force),V_scalar_new,solver_type='cg',preconditioner_type="petsc_amg"))
    if caso=="Damage":
        dis_energyN.assign(project(energy_dis(alphaN,ell,w1),V_scalar_new,solver_type='cg',preconditioner_type="petsc_amg"))
    # Control if energies are too small
    W_energyN.vector().get_local()[W_energyN.vector().get_local()<1e-12]=0.0
    dev_energyN.vector().get_local()[dev_energyN.vector().get_local()<1e-12]=0.0
    div_energyN.vector().get_local()[div_energyN.vector().get_local()<1e-12]=0.0
    E_energyN.vector().get_local()[E_energyN.vector().get_local()<1e-12]=0.0
    p_energyN.vector().get_local()[p_energyN.vector().get_local()<1e-12]=0.0
    if caso=="Damage":
            dis_energyN.vector().get_local()[dis_energyN.vector().get_local()<1e
        -12]=0.0
        # Store the energy
        if MPI.COMM_WORLD.Get_rank()>=0:
            file_energW      <<  (W_energyN,1.0*itmesh)
            file_energDev    <<  (dev_energyN,1.0*itmesh)
            file_energDiv    <<  (div_energyN,1.0*itmesh)
            file_energE      <<  (W_energyN,1.0*itmesh)
            file_energP      <<  (dev_energyN,1.0*itmesh)
            if caso == "Damage":
                file_energDis    <<  (dis_energyN,1.0*itmesh)
        # Integral value of alpha
        int_alpha = assemble (alphaN*dxN)
        vol       = assemble (1.0*dxN)
        intalpha[itmesh] = ([1.0*itmesh, int_alpha, vol])
        np.savetxt(savedir + '/int_alpha.txt', intalpha)
        itmesh = itmesh + 1
        # Free memory for lists depending on the current mesh iteration
        del duN
        del vN
        del alphaN
        del dalphaN
        del betaN
        del alphaN_0
        del W_energyN
        del div_energyN
        del dev_energyN
        del ubN
        del lbN
        del bc_uN
        del bc_boxmidx1N
        del bc_boxmidx2N
        del bc_boxmidy1N
        del bc_boxmidy2N
        del bc_boxbottomN
        del bc_alpha_upN
        del bc_alphaN
        del normal_v_new
        del boundaries_new
        del mesh_new
        del V_vector_new
        del V_scalar_new
        del total_energy1_new
        del total_energy2_new
        del elastic_energy1_new
        del external_work_new
        del dissipated_energy_new
        del E_uN
        del E_alphaN
        del E_alpha_alphaN
        del E_duN
        del E_dalphaN
        del solver_uN
        del problem_uN
        del problem_alphaN
        del dsN, dxN
```

```
    del DamageProblemN
    del a0
    del a1
#------------------------------------------------------------------------
# THE MAIN LOOP FOR REMESHING GAS FINISHED.
#------------------------------------------------------------------------
print("-------------------------------------------------------------------")
print("                Geometry with cavity is finished.                  ")
print("-------------------------------------------------------------------")
# Plot integral of damage
def plot_int_alpha():
    p1, = plt.plot(intalpha[:,0], intalpha[:,1],'r-o',linewidth=2)
    plt.xlabel('Mesh Iterations')
    plt.ylabel('Integral of damage')
    plt.savefig(savedir + '/int_alpha.png')
    plt.grid(True)
plot_int_alpha()
#------------------------------------------------------------------------
# FREE MEMORY
#------------------------------------------------------------------------
del u, du, v
del alpha, dalpha, beta, alpha_0
del alpha_error
del W_energy, div_energy, dev_energy
del file_alpha
del file_u
del lb, ub
del solver_u
del solver_alpha
del alphaAux
del V_vector
del V_scalar
del bc_u
del bc_boxmidx1, bc_boxmidx2, bc_boxmidy1, bc_boxmidy2, bc_boxbottom
del normal_v
del mesh, boundaries
#------------------------------------------------------------------------
#   End of the main program.
#------------------------------------------------------------------------
```

./codes/Cavity_3D/Cavity3D.py

B.2.2 Parameters.py

```
#------------------------------------------------------------------------
# IMPORT LIBRARIES TO GET THE CODE WORKING.
#------------------------------------------------------------------------
from dolfin import *
from mshr import *
import sys, os, sympy, shutil, math
import numpy as np
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt
import socket
import datetime
```

```
from ufl import replace
from mpi4py import MPI
from inspect import currentframe, getframeinfo,stack
#------------------------------------------------------------------------------
# PARAMETERS
#------------------------------------------------------------------------------
itmesh   =   1
NstepW   =   40
Kw       =   Constant(100)
kappa  =   Constant(1.0)
# Diferents models for w(alpha)
model    =   1
w1       =   Constant(1.0e6)
w11      =   Constant(1000)
C_L      =   0.0
# Diferent meshes to Damage problem, for example: "v3";"esmeralda"; "sub6"; "
    nuevonivel"
malla    =   "v3"
# Select cases: Damage or Elasticity
caso     =   "Damage"
# Consider Initial Damage for FH: Initial_Damage or No_Initial_Damage
FH       =   "NO_Initial_Damage"
rk       =   MPI.COMM_WORLD.Get_rank()
#------------------------------------------------------------------------------
# MATERIAL CONSTANTS
#------------------------------------------------------------------------------
E            =   2.9e10
nu           =   0.3
mu           =   E/(2.0*(1.0+nu))
lmbda        =   E*nu/(1.0-nu**2)
ffD          =   lmbda/(lmbda+2*mu)
# In this case this quantity contains the density, so gravity =  rho * g, with
     g the gravity acceleration.
rho          =   2.7e3
g            =   9.8
gravity      =   rho*g
ell          =   1.0
k_ell        =   Constant(1.e-6)
energies     =   np.zeros((NstepW,6))
#Body force
body_force   =   Constant((0.0,0.0,-gravity))
ndim         =   3
#------------------------------------------------------------------------------
# NUMERICAL PARAMETERS OF THE ALTERNATE MINIMIZATION.
#------------------------------------------------------------------------------
maxiter =   1000
toll     =   1e-5
#------------------------------------------------------------------------------
# THE FILES ARE STORED IN A FOLDER NAMED "modelname".
#------------------------------------------------------------------------------
now =    datetime.datetime.now().strftime("%m-%d_%H.%M.%S")+'_'+socket.
    gethostname()
if FH=="Initial_Damage":
    if caso=="Elasticity":
        modelname="[FH][caso=%s]_[malla=%s]_[np=%d]_%s"%(caso,malla,MPI.
    COMM_WORLD.Get_size(),now)
    if caso=="Damage":
        modelname="[FH][caso=%s]_[malla=%s]_[model=%d]_[w11=%.0f]_[np=%d]_[k2
    =%.2f]_[C_L=%.2f]_%s"%(caso,malla,model,w11,MPI.COMM_WORLD.Get_size(),kappa
    ,C_L,now)
else:
    if caso=="Elasticity":
        modelname="[caso=%s]_[malla=%s]_[np=%d]_%s"%(caso,malla,MPI.COMM_WORLD
    .Get_size(),now)
    if caso=="Damage":
        modelname="[caso=%s]_[malla=%s]_[model=%d]_[w11=%.0f]_[np=%d]_[k2=%.2f
    ]_[C_L=%.2f]_%s"%(caso,malla,model,w11,MPI.COMM_WORLD.Get_size(),kappa,C_L,
    now)
if MPI.COMM_WORLD.Get_rank() == 0:
    print('modelname ='+modelname)
# others
regenerate_mesh =   True
savedir         =   "results/%s"%(modelname)
if MPI.COMM_WORLD.Get_rank() == 0:
    print('savedir='+savedir)
#--------------------------------------------------------------------------
# READ MESH AND BOUNDARIES FROM EXTERNAL FILES.
#--------------------------------------------------------------------------
# In this block we define boundary sets for boundary conditions.
# This depends on particular tests. In this case the mesh is readed
# from external files.
# boundaries labels. These are availables in the file "*_faces.xml.gz"
# Labels for cavities
CAVEUP      =   101
CAVEBOTTOM  =   102
CAVEMID     =   103
# Labels for external boundaries
BOXUP       =   201
BOXMIDX1    =   202
BOXMIDX2    =   203
BOXMIDY1    =   204
BOXMIDY2    =   205
BOXBOTTOM   =   206
# Meshname.
if malla=="v3":
    meshname="Socavacion_incrArea5000_maxArea200000_v3_"
if malla=="esmeralda":
    meshname="esmeralda"
if malla=="sub6":
    meshname="sub6_"
if malla=="nuevonivel":
    meshname="nuevonivel_"
# Read mesh
if malla=="v3":
    mesh=Mesh("Meshes/V3/"+meshname+str(0)+".xml.gz")
if malla=="esmeralda":
    mesh=Mesh("Meshes/Esmeralda/"+meshname+str(0)+".xml.gz")
if malla=="sub6":
    mesh=Mesh("Meshes/Sub6/"+meshname+str(0)+".xml.gz")
if malla=="nuevonivel":
    mesh=Mesh("Meshes/Nuevo_Nivel/"+meshname+str(0)+".xml.gz")
mesh.init()
```

```
# Read boundaries
if malla=="v3":
    boundaries=MeshFunction('size_t',mesh,"Meshes/V3/"+meshname+str(0)+"_faces
    .xml.gz")
if malla=="esmeralda":
    boundaries=MeshFunction('size_t',mesh,"Meshes/Esmeralda/"+meshname+str(0)+
    "_faces.xml.gz")
if malla=="sub6":
    boundaries=MeshFunction('size_t',mesh,"Meshes/Sub6/"+meshname+str(0)+"
    _faces.xml.gz")
if malla=="nuevonivel":
    boundaries=MeshFunction('size_t',mesh,"Meshes/Nuevo_Nivel/"+meshname+str
    (0)+"_faces.xml.gz")
# Maximum values of mesh (local)
meshl_xmax,meshl_ymax,meshl_zmax = mesh.coordinates().max(axis=0)
# Maximum values of mesh (global)
mesh_xmax    =    MPI.COMM_WORLD.allreduce(meshl_xmax, op=MPI.MAX)
mesh_ymax    =    MPI.COMM_WORLD.allreduce(meshl_ymax, op=MPI.MAX)
mesh_zmax    =    MPI.COMM_WORLD.allreduce(meshl_zmax, op=MPI.MAX)
# Minimum values of mesh (local)
meshl_xmin,meshl_ymin,meshl_zmin = mesh.coordinates().min(axis=0)
# Minimum values of mesh (global)
mesh_xmin    =    MPI.COMM_WORLD.allreduce(meshl_xmin, op=MPI.MIN)
mesh_ymin    =    MPI.COMM_WORLD.allreduce(meshl_ymin, op=MPI.MIN)
mesh_zmin    =    MPI.COMM_WORLD.allreduce(meshl_zmin, op=MPI.MIN)
print ("ZMAX(G,L):",mesh_zmax,meshl_ymax)
print ("ZMIN(G,L):",mesh_zmin,meshl_ymin)
# normal vectors
normal_v     =    FacetNormal(mesh)
# Face Corner Bottom for the uniqueness
class FaceCornerBottom(SubDomain):
    def inside(self,x,on_boundary):
        px,py,pz=x
        return (on_boundary and near(pz,mesh_zmin)
                and (px<mesh_xmin+mesh.hmax())
                and (py<mesh_ymin+mesh.hmax())
                )
FACECORNERBOTTOM    =    400
faceCornerBottom    =    FaceCornerBottom()
faceCornerBottom.mark(boundaries, FACECORNERBOTTOM)
#------------------------------------------------------------------------
# CREATE FUNCTION SPACE FOR 3D ELASTICITY AND DAMAGE
#------------------------------------------------------------------------
V_vector     =    VectorFunctionSpace(mesh,"CG",1)
V_scalar     =    FunctionSpace(mesh,"CG",1)
V_tensor     =    TensorFunctionSpace(mesh,"DG",0)
#------------------------------------------------------------------------
# BOUNDARY CONDITIONS.
#------------------------------------------------------------------------
bc_facecornerbottom =    DirichletBC(V_vector.sub(2),Constant(0.0),boundaries,
    FACECORNERBOTTOM)
bc_boxmidx1         =    DirichletBC(V_vector.sub(1),Constant(0.0),boundaries,
    BOXMIDX1)
bc_boxmidx2         =    DirichletBC(V_vector.sub(1),Constant(0.0),boundaries,
    BOXMIDX2)
```

```
bc_boxmidy1         =   DirichletBC(V_vector.sub(0),Constant(0.0),boundaries,
    BOXMIDY1)
bc_boxmidy2         =   DirichletBC(V_vector.sub(0),Constant(0.0),boundaries,
    BOXMIDY2)
bc_boxbottom        =   DirichletBC(V_vector.sub(2),Constant(0.0),boundaries,
    BOXBOTTOM)
bc_u                =   [bc_boxbottom,bc_facecornerbottom]
# Newmann boundary condition
kx      =   gravity
g_bc_zz =   Expression('ffD*k*(x[2]-mesh_zmax)',degree=2,k=kx,mesh_zmax=
    mesh_zmax,ffD=ffD)
```

./codes/Cavity_3D/Parameters.py

B.2.3 AuxFunctions.py

```
#-----------------------------------------------------------------------
# AUXILIARY FUNCTION FOR INTERPOLATION.
#-----------------------------------------------------------------------
from dolfin import dof_to_vertex_map,vertex_to_dof_map,Function,Vector
from numpy import array
from mpi4py import MPI
#-----------------------------------------------------------------------
# INTERPOLATION.
#-----------------------------------------------------------------------
def Interpola_In_Out(Var_In,Var_Out,GVertex_Out_2_GVertex_In) :
    V_In          =   Var_In.function_space()
    mesh_In       =   Var_In.function_space().mesh()
    l2gvi_In      =   mesh_In.topology().global_indices(0)
    g2lvi_In      =   {}
    for j in range(mesh_In.num_vertices()):
        g2lvi_In[l2gvi_In[j]]=j
    dof_2_vl_In                 =   dof_to_vertex_map(V_In)
    vl_2_dof_In                 =   vertex_to_dof_map(V_In)
    l2g_dof_vector_In           =   Function(V_In)
    LDof_2_GVertex_arr_In       =   l2g_dof_vector_In.vector().get_local()
    for i, v in enumerate(LDof_2_GVertex_arr_In):
        LDof_2_GVertex_arr_In[i]=l2gvi_In[dof_2_vl_In[i]]
    l2g_dof_vector_In.vector().set_local(LDof_2_GVertex_arr_In)
    Var_G_In                    =   Vector(MPI.COMM_SELF)
    GDof_2_GVertex_Vself_In =   Vector(MPI.COMM_SELF)
    Var_In.vector().gather(Var_G_In,array(range(V_In.dim()),"intc"))
    l2g_dof_vector_In.vector().gather(GDof_2_GVertex_Vself_In,array(range(V_In
    .dim()),"intc"))
    Var_G_sorted_In =   [Var_G_In.get_local()[i[0]] for i in sorted(enumerate(
    GDof_2_GVertex_Vself_In.get_local()),key=lambda x:x[1])]
    V_Out             =   Var_Out.function_space()
    mesh_Out          =   Var_Out.function_space().mesh()
    l2gvi_Out         =   mesh_Out.topology().global_indices(0)
    g2lvi_Out         =   {}
    for j in range(mesh_Out.num_vertices()):
        g2lvi_Out[l2gvi_Out[j]]=j
    dof_2_vl_Out                =   dof_to_vertex_map(V_Out)
    vl_2_dof_Out                =   vertex_to_dof_map(V_Out)
    LDof_2_GVertex_Fn_Out       =   Function ( V_Out)
    LDof_2_GVertex_arr_Out      =   LDof_2_GVertex_Fn_Out.vector().get_local()
    for i, v in enumerate(LDof_2_GVertex_arr_Out):
        LDof_2_GVertex_arr_Out[i]=l2gvi_Out[dof_2_vl_Out[i]]
    LDof_2_GVertex_Fn_Out.vector().set_local(LDof_2_GVertex_arr_Out)
    Var_G_Out                   =   Vector(MPI.COMM_SELF)
    GDof_2_GVertex_Vself_Out    =   Vector(MPI.COMM_SELF)
    Var_Out.vector().gather(Var_G_Out, array(range(V_Out.dim()),"intc"))
    LDof_2_GVertex_Fn_Out.vector().gather(GDof_2_GVertex_Vself_Out,array(range
(V_Out.dim()),"intc"))
    Var_G_sorted_Out=[Var_G_Out.get_local()[i[0]] for i in sorted(enumerate(
GDof_2_GVertex_Vself_Out.get_local()), key=lambda x:x[1])]
    for i in range(len(Var_G_sorted_In),len(Var_G_sorted_Out)) :
        Var_G_sorted_In.append(Var_G_sorted_Out[i])
    Var_array_Out   =   Var_Out.vector().get_local()
    rk              =   MPI.COMM_WORLD.Get_rank()
    for i,GV_Out in enumerate(LDof_2_GVertex_arr_Out) :
        try:
            iGV_Out=int(GV_Out)
            Var_array_Out[i]=Var_G_sorted_In[GVertex_Out_2_GVertex_In[iGV_Out
]]
        except IOError:
            print ("iGV_Out=",iGV_Out)
    Var_Out.vector().set_local(Var_array_Out)
    Var_Out.vector().apply("")
```
./codes/Cavity_3D/AuxFunctions.py

Appendix C

Codes in MATLAB

This appendix presents the computer codes used in the Part II, which have been developed in Python and with the use of MATLAB software. MATLAB is a proprietary multi-paradigm programming language and numerical computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.

C.1 main.m

```
%% CGO solutions (vectorial case)
clear all;
%% parameters
% choose the type of example
% example1: A=I*sigma.
% example2: A=[2 0;0 3]*sigma.
% example3: A=[2 1;1 3]*sigma
% example4: A=[sigma1 sigma2;sigma3 sigma4]
example = 'example0';

s    =    2.1;  %square [-s,s)
m    =    8;
M    =    2^m;
h    =    (2*s)/M;
k    =    2;
% Create Grid
[x1,x2] =    crea_grid(m,h);
z       =    complex(x1,x2);
%% functions eta, g and b (Cauchy and Beurling Transforms)
eta=crea_eta(x1,x2,M,2);
g=eta./(pi*z);
g(abs(z)==0)=0;
b=eta./(pi*z.^2);
b(abs(z)==0)=0;
% Fourier transform
Fg=fft2(fftshift(g));
Fb=fft2(fftshift(b));
```

```
%% Sigma y mu
if example == 'example0'
    disp('Example 0.')
    sigma_1   =   0.5*sigma5(z);
    sigma_2   =   0.2*sigma6(z);
    sigma_3   =   0.3*sigma7(z);
    sigma_4   =   0.5*sigma8(z);
end
if example == 'example1'
    disp('Example 1.')
    sigma_1   =   sigma1(z);
    sigma_2   =   0*sigma1(z);
    sigma_3   =   0*sigma1(z);
    sigma_4   =   sigma1(z);
end
if example == 'example2'
    disp('Example 2.')
    sigma_1   =   2*sigma1(z);
    sigma_2   =   0*sigma1(z);
    sigma_3   =   0*sigma1(z);
    sigma_4   =   3*sigma1(z);
end
if example == 'example3'
    disp('Example 3.')
    sigma_1   =   2*sigma1(z);
    sigma_2   =   1*sigma1(z);
    sigma_3   =   1*sigma1(z);
    sigma_4   =   3*sigma1(z);
end
if example == 'example4'
    disp('Example 4.')
    sigma_1   =   sigma1(z);
    sigma_2   =   sigma2(z);
    sigma_3   =   sigma3(z);
    sigma_4   =   sigma4(z);
end
sigma_2(abs(z)>1)    =   0;
sigma_3(abs(z)>1)    =   0;
% SIGMA MATRIX
SIGMA   =   [sigma_1 sigma_2; sigma_3 sigma_4];
% Create MU matrix  (MU=(I-SIGMA)(I+SIGMA)^{-1})
% Determinant of Sigma
detS   =   ((1+sigma_1).*(1+sigma_4))-(sigma_2.*sigma_3);
mu1     =   ((1-sigma_1).*(1+sigma_4)-(sigma_2.*sigma_3))./detS;
mu2     =   (2*(sigma_1.*sigma_2))./detS;
mu3     =   (2*(sigma_3.*sigma_4))./detS;
mu4     =   ((1+sigma_1).*(1-sigma_4)-(sigma_2.*sigma_3))./detS;
MU      =   [mu1 mu2;mu3 mu4];
% Plot Sigma and MU
sigma_1tmp    =   sigma_1;
sigma_1tmp(abs(z)>1)  =   nan;
f1=figure(1);
surf(x1,x2,sigma_1tmp);
saveas(f1,'example5_sigma_1','png')
sigma_2tmp    =   sigma_2;
sigma_2tmp(abs(z)>1)  =   nan;
```

```
f2=figure(2);
surf(x1,x2,sigma_2tmp);
saveas(f2,'example5_sigma_2','png')
sigma_3tmp    =   sigma_3;
sigma_3tmp(abs(z)>1)  =   nan;
f3=figure(3);
surf(x1,x2,sigma_3tmp);
saveas(f3,'example5_sigma_3','png')
sigma_4tmp    =   sigma_4;
sigma_4tmp(abs(z)>1)  =   nan;
f4=figure(4);
surf(x1,x2,sigma_4tmp);
saveas(f4,'example5_sigma_4','png')
%% alpha and nu
E=exp(-1i*((k*z) + (conj(k)*conj(z))));
a1  =   -1i*conj(k)*E.*mu1;
a2  =   -1i*conj(k)*E.*mu2;
a3  =   -1i*conj(k)*E.*mu3;
a4  =   -1i*conj(k)*E.*mu4;
nu1 =   E.*mu1;
nu2 =   E.*mu2;
nu3 =   E.*mu3;
nu4 =   E.*mu4;
alpha   =   [a1 a2;a3 a4];
nu      =   [nu1 nu2;nu3 nu4];
A=[a1(:)+a2(:);a3(:)+a4(:)];
%% Solve for V from   (I-A*conj(A))V=-conj(alpha) using GMRES
v = gmres('operator', -conj(A), 50, 1e-6, 500, [], [], -conj(A), Fg, Fb,h,nu1,
    nu2,nu3,nu4,a1,a2,a3,a4,M);
%% Calculate U=(I-A*rho)V
vbar     =   conj(v);
v1       =   v(1:M^2);
v2       =   v((M^2)+1:2*M^2);
v1tmp    =   reshape(v1,M,M);
v2tmp    =   reshape(v2,M,M);
vtmp     =   [v1;v2];
Sv1    =   (h^2*(ifft2(Fb.*fft2(conj(v1tmp)))));
Sv2    =   (h^2*(ifft2(Fb.*fft2(conj(v2tmp)))));
Pv1    =   (h^2*(ifft2(Fg.*fft2(conj(v1tmp)))));
Pv2    =   (h^2*(ifft2(Fg.*fft2(conj(v2tmp)))));
Av1     =  (-conj(nu1).*Sv1-conj(nu2).*Sv2)-(conj(a1).*Pv1+conj(a2).*Pv2);
Av2     =  (-conj(nu3).*Sv1-conj(nu4).*Sv2)-(conj(a3).*Pv1+conj(a4).*Pv2);
Av       =   [Av1(:);Av2(:)];
u        =   vtmp-Av;
%%  Compute N = - P*conj(U)
ubar     =   conj(u);
u1       =   u(1:M^2);
u2       =   u((M^2)+1:2*M^2);
u1tmp    =   reshape(u1,M,M);
u2tmp    =   reshape(u2,M,M);
Pu1    =   (h^2*(ifft2(Fg.*fft2(conj(u1tmp)))));
Pu2    =   (h^2*(ifft2(Fg.*fft2(conj(u2tmp)))));
N1  =   -Pu1;
N2  =   -Pu2;
%% Plot real(N)
N1tmp     =    real(N1);
```

```
N1tmp(abs(z)>1)  =   nan;
f5=figure(5);
surf(x1,x2,N1tmp);
saveas(f5,'example5_N_1_m8','png')
N2tmp    =   real(N2);
N2tmp(abs(z)>1)  =   nan;
f6=figure(6);
surf(x1,x2,N2tmp);
saveas(f6,'example5_N_2_m8','png')
%% functions
function [x1,x2] = crea_grid(m,h)
j1=-2^(m-1):2^(m-1)-1;
j2=-2^(m-1):2^(m-1)-1;
hj1=h*j1;
hj2=h*j2;
[x1,x2]=meshgrid(hj1,hj2);
end
function eta = crea_eta(x1,x2,M,s)
etatemp=zeros(M,M);
for i=1:M
for j=1:M
        if sqrt(x1(1,i)^2+x2(j,1)^2)<2
        etatemp(i,j)=1;
        end
        if 2<=sqrt(x1(1,i)^2+x2(j,1)^2) && sqrt(x1(1,i)^2+x2(j,1)^2)<2+((s-2)
    /2)
        etatemp(i,j)=(-2/(s-2))*(sqrt(x1(1,i)^2+x2(j,1)^2)-2)+1;
        end
        if 2+((s-2)/2)<=sqrt(x1(1,i)^2+x2(j,1)^2)
        etatemp(i,j)=0;
        end
end
end
 eta = etatemp;
end
%% Sigma
function result = sigma1(z)
% Conductivities of heart and lung (background is 1)
heart = 2;
lung  = 0.7;
% Initialize
[zfila,zcol] = size(z);
z           = z(:);
result          = ones(size(z));
x1          = real(z);
x2          = imag(z);
% Build coarse representation of heart. Planar point (hc1,hc2) is the center
    of the ellipse
% describing the heart; numbers he1 and he2 give the eccentricities with respect
     to radius hR.
hc1 = -.1;
hc2 = .4;
he1 = .8;
he2 = 1;
hR  = .2;
% Compute elliptical "distance" of the evaluation points from heart
```

```
hd  = sqrt(he1*(x1-hc1).^2 + he2*(x2-hc2).^2);
% Set value of conductivity inside the heart
result(hd <= hR) = heart;
% Build coarse representation of two lungs
l1c1  = .5;
l1c2  = 0;
l1e1  = 3;
l1e2  = 1;
l1R   = .5;
fii   = -pi/7;
rot11 = cos(fii);
rot12 = sin(fii);
rot21 = -sin(fii);
rot22 = cos(fii);
l1d   = sqrt(l1e1*((rot11*x1+rot12*x2)-l1c1).^2 + l1e2*((rot21*x1+rot22*x2)-
    l1c2).^2);
result(l1d <= l1R) = lung;
l2c1 = -.6;
l2c2 = 0;
l2e1 = 3;
l2e2 = 1;
l2R  = .4;
fii   = pi/7;
rot11 = cos(fii);
rot12 = sin(fii);
rot21 = -sin(fii);
rot22 = cos(fii);
l2d  = sqrt(l2e1*((rot11*x1+rot12*x2)-l2c1).^2 + l2e2*((rot21*x1+rot22*x2)-
    l2c2).^2);
result(l2d <= l2R) = lung;
 result = reshape(result,[zfila,zcol]);
end
```

./codes/Vectorial_CGO/main.m

C.2 operator.m

```
% Implements the operator
% v |-> v-Aconj(A)v,
% where A=(-conj(nu)S-conj(alpha)P).
function result = operator(v,Fg,Fb,h,nu1,nu2,nu3,nu4,a1,a2,a3,a4,M)
v        =    v(:);
v1       =    v(1:M^2);
v2       =    v((M^2)+1:2*M^2);
vtmp     =    [v1;v2];
v1tmp    =    reshape(v1,M,M);
v2tmp    =    reshape(v2,M,M);
Sconjv1    =    (h^2*(ifft2(Fb.*fft2(conj(v1tmp)))));
Sconjv2    =    (h^2*(ifft2(Fb.*fft2(conj(v2tmp)))));
Pconjv1    =    (h^2*(ifft2(Fg.*fft2(conj(v1tmp)))));
Pconjv2    =    (h^2*(ifft2(Fg.*fft2(conj(v2tmp)))));
A1     =  (-conj(nu1).*Sconjv1-conj(nu2).*Sconjv2)-(conj(a1).*Pconjv1+conj(a2)
    .*Pconjv2);
A2     =  (-conj(nu3).*Sconjv1-conj(nu4).*Sconjv2)-(conj(a3).*Pconjv1+conj(a4)
    .*Pconjv2);
```

```
A1bar    =    conj(A1);
A2bar    =    conj(A2);
SA1    =    (h^2*(ifft2(Fb.*fft2(A1bar))));
SA2    =    (h^2*(ifft2(Fb.*fft2(A2bar))));
PA1    =    (h^2*(ifft2(Fg.*fft2(A1bar))));
PA2    =    (h^2*(ifft2(Fg.*fft2(A2bar))));
AA1     =  (-conj(nu1).*SA1-conj(nu2).*SA2)-(conj(a1).*PA1+conj(a2).*PA2);
AA2     =  (-conj(nu3).*SA1-conj(nu4).*SA2)-(conj(a3).*PA1+conj(a4).*PA2);
AA1     =     AA1(:);
AA2     =     AA2(:);
A       =     [AA1;AA2];
result   = vtmp-A;
```

./codes/Vectorial_CGO/operator.m

Bibliography

[1] Luigi Ambrosio and Andrea Braides. Energies in sbv and variational models in fracture mechanics. *Homogenization and applications to material sciences*, 9:1–22, 1997.

[2] Luigi Ambrosio and Vincenzo Maria Tortorelli. Approximation of functional depending on jumps by elliptic functional via t-convergence. *Communications on Pure and Applied Mathematics*, 43(8):999–1036, 1990.

[3] Kari Astala, Tadeusz Iwaniec, Eero Saksman, et al. Beltrami operators in the plane. *Duke Mathematical Journal*, 107(1):27–56, 2001.

[4] Kari Astala, Jennifer L Mueller, Lassi Päivärinta, and Samuli Siltanen. Numerical computation of complex geometrical optics solutions to the conductivity equation. *Applied and Computational Harmonic Analysis*, 29(1):2–17, 2010.

[5] Kari Astala and Lassi Päivärinta. Calderón's inverse conductivity problem in the plane. *Annals of Mathematics*, pages 265–299, 2006.

[6] Kari Astala, Lassi Päivärinta, and Matti Lassas. Calderóns' inverse problem for anisotropic conductivity in the plane. *Communications in Partial Difference Equations*, 30(1-2):207–224, 2005.

[7] Satish Balay, Kris Buschelman, Victor Eijkhout, William D Gropp, Dinesh Kaushik, Matthew G Knepley, Lois Curfman McInnes, Barry F Smith, and Hong Zhang. Petsc users manual. Technical report, Technical Report ANL-95/11-Revision 2.1. 5, Argonne National Laboratory, 2004.

[8] R Beals and R Coifman. Transformation spectrales et equation d'evolution non-lineares. *Seminaire Goulaouic-Meyer-Schwarz, exp*, 21:1981–1982, 1981.

[9] B. Bourdin, G. A. Francfort, and J.-J. Marigo. Numerical experiments in revisited brittle fracture. *J. Mech. Phys. Solids*, 48(4):797–826, 2000.

[10] Blaise Bourdin. Numerical implementation of the variational formulation for quasi-static brittle fracture. *Interfaces Free Bound.*, 9(3):411–430, 2007.

[11] Andrea Braides. *Approximation of free-discontinuity problems*, volume 1694 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1998.

[12] Edwin Thomas Brown. Block caving geomechanics. 2002.

[13] R Brown and G Uhlmann. Uniqueness in the inverse conductivity problem with less regular conductivities, comm. *PDE*, 22:1009–1027, 1997.

[14] Alberto P Calderón. On an inverse boundary value problem. *Computational & Applied Mathematics*, 25(2-3):133–138, 2006.

[15] Gianni Dal Maso and Rodica Toader. A model for the quasi-static growth of brittle fractures: existence and approximation results. *Arch. Ration. Mech. Anal.*, 162(2):101–135, 2002.

[16] Prabir Daripa and Daoud Mashat. Singular integral transforms and fast numerical algorithms. *Numerical algorithms*, 18(2):133–157, 1998.

[17] Kui Du. A simple numerical method for complex geometrical optics solutions to the conductivity equation. *SIAM Journal on Scientific Computing*, 33(1):328–341, 2011.

[18] Ludvig Dmitrievich Faddeev. Increasing solutions of the schrödinger equation. *SPhD*, 10:1033, 1966.

[19] G. A. Francfort and J.-J. Marigo. Revisiting brittle fracture as an energy minimization problem. *J. Mech. Phys. Solids*, 46(8):1319–1342, 1998.

[20] Gilles A. Francfort and Christopher J. Larsen. Existence and convergence for quasi-static evolution in brittle fracture. *Comm. Pure Appl. Math.*, 56(10):1465–1500, 2003.

[21] Alessandro Giacomini. Ambrosio-Tortorelli approximation of quasi-static evolution of brittle fractures. *Calc. Var. Partial Differential Equations*, 22(2):129–172, 2005.

[22] Alan Arnold Griffith. The phenomena of rupture and flow in solids. *Philosophical transactions of the royal society of london. Series A, containing papers of a mathematical or physical character*, 221(582-593):163–198, 1921.

[23] Bernard Halphen and Nguyen Quoc Son. Sur les matériaux standards généralisés. *J. Mécanique*, 14:39–63, 1975.

[24] Howard L Hartman and Jan M Mutmansky. *Introductory mining engineering.* John Wiley & Sons, 2002.

[25] Horst Heck, Xiaosheng Li, and Jenn-Nan Wang. Identification of viscosity in an incompressible fluid. *Indiana University mathematics journal*, pages 2489–2510, 2007.

[26] Michael Hinze and Tran Nhan Tam Quyen. Matrix coefficient identification in an elliptic equation with the convex energy functional method. *Inverse problems*, 32(8):085007, 2016.

[27] K-H Hoffmann and Jürgen Sprekels. On the identification of coefficients of elliptic problems by asymptotic regularization. *Numerical functional analysis and optimization*,

7(2-3):157–177, 1985.

[28] Marko Huhtanen and Allan Perämäki. Numerical solution of the $\mathbb{R}$-linear beltrami equation. *Mathematics of Computation*, 81(277):387–397, 2012.

[29] D Isaacson, JC Newell, JC Goble, and M Cheney. Thoracic impedance images during ventilation. In *[1990] Proceedings of the Twelfth Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 106–107. IEEE, 1990.

[30] David Isaacson, Jennifer L Mueller, Jonathan C Newell, and Samuli Siltanen. Reconstructions of chest phantoms by the d-bar method for electrical impedance tomography. *IEEE Transactions on Medical Imaging*, 23(7):821–828, 2004.

[31] Josep Jordana, Manel Gasulla, and Ramon Pallàs-Areny. Electrical resistance tomography to detect leaks from buried pipes. *Measurement Science and Technology*, 12(8):1061, 2001.

[32] Jacques Jossinet. The impedivity of freshly excised human breast tissue. *Physiological measurement*, 19(1):61, 1998.

[33] Robert Kohn and Michael Vogelius. Determining conductivity by boundary measurements. *Communications on pure and applied mathematics*, 37(3):289–298, 1984.

[34] Robert V Kohn and Michael Vogelius. Determining conductivity by boundary measurements ii. interior results. *Communications on Pure and Applied Mathematics*, 38(5):643–667, 1985.

[35] Giovanni Lancioni and Gianni Royer-Carfagni. The variational approach to fracture mechanics. a practical application to the french panthéon in paris. *Journal of elasticity*, 95(1-2):1–30, 2009.

[36] Hans Petter Langtangen, Anders Logg, and Aslak Tveito. *Solving PDEs in Python: The FEniCS Tutorial I.* Springer International Publishing, 2016.

[37] Tianyi Li, Jean-Jacques Marigo, Daniel Guilbaud, and Serguei Potapov. Gradient damage modeling of brittle fracture in an explicit dynamics context. *Internat. J. Numer. Methods Engrg.*, 108(11):1381–1405, 2016.

[38] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media, 2012.

[39] Jean-Jacques Marigo and Kyrylo Kazymyrenko. A micromechanical inspired model for the coupled to damage elasto-plastic behavior of geomaterials under compression. *Mechanics & Industry*, 20(1):105, 2019.

[40] Jean-Jacques Marigo, Corrado Maurini, and Kim Pham. An overview of the modelling of fracture by gradient damage models. *Meccanica*, 51(12):3107–3128, 2016.

[41] JJ Marigo. Constitutive relations in plasticity, damage and fracture mechanics based on a work property. *Nuclear Engineering and Design*, 114(3):249–272, 1989.

[42] Jennifer L Mueller and Samuli Siltanen. *Linear and nonlinear inverse problems with practical applications*. SIAM, 2012.

[43] Jennifer L Mueller, Samuli Siltanen, and David Isaacson. A direct reconstruction algorithm for electrical impedance tomography. *IEEE Transactions on medical imaging*, 21(6):555–559, 2002.

[44] David Mumford and Jayant Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Comm. Pure Appl. Math.*, 42(5):577–685, 1989.

[45] Adrian I Nachman. Global uniqueness for a two-dimensional inverse boundary value problem. *Annals of Mathematics*, pages 71–96, 1996.

[46] Kim Pham, Hanen Amor, Jean-Jacques Marigo, and Corrado Maurini. Gradient damage models and their use to approximate brittle fracture. *International Journal of Damage Mechanics*, 20(4):618–652, 2011.

[47] Kim Pham and Jean-Jacques Marigo. Approche variationnelle de l'endommagement : I. les concepts fondamentaux. *Comptes Rendus Mécanique*, 338(4):191–198, 2010.

[48] Kim Pham and Jean-Jacques Marigo. Approche variationnelle de l'endommagement: Ii. les modèles à gradient. *Comptes Rendus Mécanique*, 338(4):199–206, 2010.

[49] Rolf Rannacher and Boris Vexler. A priori error estimates for the finite element discretization of elliptic parameter identification problems with pointwise measurements. *SIAM Journal on Control and Optimization*, 44(5):1844–1863, 2005.

[50] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[51] Matteo Santacesaria. Note on calderón's inverse problem for measurable conductivities, 2019.

[52] Jacques Simon. Differentiation with respect to the domain in boundary value problems. *Numerical Functional Analysis and Optimization*, 2(7-8):649–687, 1980.

[53] Oliver K. Smith. Eigenvalues of a symmetric 3×3 matrix. *Comm. ACM*, 4:168, 1961.

[54] Ziqi Sun, Gunther Uhlmann, et al. Generic uniqueness for an inverse boundary value problem. *Duke Mathematical Journal*, 62(1):131–155, 1991.

[55] John Sylvester and Gunther Uhlmann. A uniqueness theorem for an inverse boundary value problem in electrical prospection. *Communications on Pure and Applied Mathematics*, 39(1):91–112, 1986.

[56] John Sylvester and Gunther Uhlmann. A global uniqueness theorem for an inverse boundary value problem. *Annals of mathematics*, pages 153–169, 1987.

[57] Gunther Uhlmann. Electrical impedance tomography and calderón's problem. *Inverse problems*, 25(12):123011, 2009.

[58] Gunther Uhlmann and Jenn-Nan Wang. Complex spherical waves for the elasticity system and probing of inclusions. *SIAM journal on mathematical analysis*, 38(6):1967–1980, 2007.

[59] I. Nestorovich Vekua. *Generalized analytic functions*. Elsevier, 2014.

[60] Y Zou and Z Guo. A review of electrical impedance techniques for breast cancer detection. *Medical engineering & physics*, 25(2):79–90, 2003.