FEDERAL UNIVERSITY OF MINAS GERAIS (UFMG)
SCHOOL OF ENGINEERING
DEPARTMENT OF STRUCTURAL ENGINEERING
GRADUATE PROGRAM IN STRUCTURAL ENGINEERING (PROPEES)

Matheus Moreno Fortes

A BOUND-CONSTRAINED SOLVER FOR PHASE-FIELD MODELING OF DIFFUSE FRACTURE

Belo Horizonte, Brazil
29th Aug, 2022

Matheus Moreno Fortes

A BOUND-CONSTRAINED SOLVER FOR PHASE-FIELD MODELING OF DIFFUSE FRACTURE

Final Version

Master's thesis submitted to the Graduate Program in Structural Engineering (PROPEES) of the School of Engineering at the FEDERAL UNIVERSITY OF MINAS GERAIS (UFMG), in partial fulfillment of the requirements for the master degree in Structural Engineering

Supervisor: Prof. Dr. Lapo Gori

Co-Supervisor: Prof. Dr. Roque Luiz da Silva Pitangueira

Belo Horizonte, Brazil
29th Aug, 2022

Fortes, Matheus Moreno.
F736s Um solucionador de restrição para modelagem de campos de fase de fratura difusa [recurso eletrônico] / Matheus Moreno Fortes. - 2022.
1 recurso online (102 f. : il., color.) : pdf.

Orientador: Lapo Gori.

Dissertação (mestrado) - Universidade Federal de Minas Gerais, Escola de Engenharia.

Apêndices: f. 90-102.

Bibliografia: f. 85-88.
Exigências do sistema: Adobe Acrobat Reader.

1. Engenharia de estruturas - Teses. 2. Fratura - Teses. 3. Método dos elementos finitos. I. Gori, Lapo. II. Universidade Federal de Minas Gerais. Escola de Engenharia. III. Título.

CDU: 624(043)

Ficha catalográfica elaborada pela bibliotecária Roseli Alves de Oliveira CRB/6 2121
Biblioteca Prof. Mário Werneck, Escola de Engenharia da UFMG

UNIVERSIDADE FEDERAL DE MINAS GERAIS

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ESTRUTURAS

PROPEEs

UFMG

ATA DA DEFESA DE DISSERTAÇÃO DE MESTRADO EM ENGENHARIA DE ESTRUTURAS Nº: 390 DO ALUNO MATHEUS MORENO FORTES

Às **14:00** horas do dia **29** do mês de **agosto** de **2022**, reuniu-se em ambiente virtual, na Escola de Engenharia da Universidade Federal de Minas Gerais - UFMG, a Comissão Examinadora indicada pelo Colegiado do Programa em **15 de julho de 2022**, para julgar a defesa da Dissertação de Mestrado intitulada **"Um Solucionador de Restrição para Modelagem de Campos de Fase de Fratura Difusa"**, cuja aprovação é um dos requisitos para a obtenção do Grau de MESTRE EM ENGENHARIA DE ESTRUTURAS na área de ESTRUTURAS.

Abrindo a sessão, o Presidente da Comissão, **Prof. Dr. Lapo Gori**, após dar a conhecer aos presentes o teor das Normas Regulamentares passou a palavra ao candidato para apresentação de seu trabalho. Seguiu-se a argüição pelos examinadores, com a respectiva defesa do candidato. Logo após, a Comissão se reuniu, sem a presença do candidato e do público, para julgamento e expedição do resultado final. Foram atribuídas as seguintes indicações:

Prof. Dr. Lapo Gori - DEES - UFMG (Orientador): aprovado

Prof. Dr. Roque Luiz da Silva Pitangueira - DEES - UFMG: aprovado

Prof. Dr. Samuel Silva Penna - DEES - UFMG: aprovado

Profa. Dra. Deane de Mesquita Roehl - PUC-Rio: aprovado

Pelas indicações acima, o candidato foi considerado APROVADO, conforme pareceres em anexo.

O resultado final foi comunicado publicamente ao candidato pelo Presidente da Comissão.

Nada mais havendo a tratar, o Presidente encerrou a reunião e lavrou a presente ATA, que será assinada por todos os membros participantes da Comissão Examinadora.

Belo Horizonte, 29 de agosto de 2022.

Observações:

1. A aprovação do candidato na defesa da Dissertação de Mestrado não significa que o mesmo tenha cumprido todos os requisitos necessários para obtenção do Grau de Mestre em Engenharia de Estruturas;
2. Este documento não terá validade sem a assinatura e carimbo do Coordenador do Programa de Pós-Graduação.

Documento assinado eletronicamente por **Lapo Gori**, **Professor do Magistério Superior**, em 29/08/2022, às 16:59, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Samuel Silva Penna**, **Professor do Magistério Superior**, em 29/08/2022, às 17:02, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Roque Luiz da Silva Pitangueira**, **Professor do Magistério Superior**, em 29/08/2022, às 17:04, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Deane de Mesquita Roehl**, **Usuária Externa**, em 02/09/2022, às 08:34, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Felicio Bruzzi Barros**, **Coordenador(a) de curso de pós-graduação**, em 03/10/2022, às 09:57, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1693015** e o código CRC **D40FE6E2**.

Referência: Processo nº 23072.249946/2022-48 SEI nº 1693015

Agradecimentos

Agradeço primeiramente à minha noiva Vitória Costa e Silva, pelo amor, pela companhia, pelo suporte, pelo carinho, por aguentar minhas crises e me fazer feliz.

À minha família pelo amor, carinho e suporte durante minha vida inteira, em especial à minha mãe Edna e à minha irmã Itala.

Aos professores orientadores Lapo Gori e Roque Luiz da Silva Pitangueira, pela confiança depositada em mim, pelos ensinamentos passados, pela paciência e dedicação comigo.

Aos demais professores do Departamento de Engenharia de Estruturas da UFMG, pela contribuição à minha formação e ao desenvolvimento deste trabalho.

Aos meus amigos de mestrado, que estudaram comigo, em especial à equipe do Phase-Field, Larissa, Hugo, Lívia e Rafael, que me passaram conhecimento e me ajudaram várias vezes. Ao Danilo, um amigo que mesmo longe me ajudou muito a nível pessoal e acadêmico.

Aos meus amigos de vida, o grupo da engenharia e PS4, que me divertem e me fazem rir.

Aos funcionários do Departamento de Engenharia de Estruturas da UFMG, pela disponibilidade.

À UFG, em especial aos meus eternos professores, pela sólida formação proporcionada na graduação, em especial ao professor Zenón José Guzmán Núñez del Prado.

Ao CNPq, pelo auxílio financeiro.

À todos aqueles que de alguma forma me ajudaram para a realização desse trabalho.

Resumo

O estudo da fissuração de um elemento estrutural é importante. Nesse estudos a trinca pode ser modelada com a abordagem discreta ou contínua. O modelo de campos de fases é um modelo de trinca contínua que adiciona uma variável ao problema para representar a degradação do material no ponto. O grupo de pesquisa de campos de fase do Departamento de Engenharia Estrutural (DEES) da Universidade Federal de Minas Gerais (UFMG) já fez alguns trabalhos nesse tema utilizando o software *INSANE* (INteractive Structural ANalysis Environment), e um obstáculo encontrado foi o tipo de solucionador. Um solucionador amplamente usado na literatura de campos de fase é o solucionador histórico, porém ele limita a geometria da trinca a ser utilizada. Para contornar essa limitação precisa-se de outro solucionador. Esse trabalho mostra a implementação de um solucionador com restrição de contorno através da biblioteca externa PETSc (Portable, Extensible Toolkit for Scientific Computation) aplicada no *INSANE*. Com esse novo solucionador mostra-se as novas possibilidades de modelos, compara-se a performance com outros solucionadores que o *INSANE* possui, compara-se a convergência e dependência de malha com o modelo de fissuração distribuída, e por fim, avalia-se a capacidade do modelo de campos de fase em captar os modos de falhas I e II.

Palavras-chave: Modelos de fratura baseado em campo de fase; Solucionador com restrição de contorno; Método dos Elementos Finitos baseado em modelos de campo de fase.

Abstract

The study of the cracking of a structural element is important. In these studies, the crack can be modeled with the discrete or continuous approach. The phase-field model is a continuous crack model that adds a variable to the problem to represent the degradation of the material at a point.The phase-field research group of the Department of Structural Engineering (DEES) of the Federal University of Minas Gerais (UFMG) has already done some work on this topic using the *INSANE* software (INteractive Structural ANalysis Environment). One of the obstacles encountered was the type of solver. A solver widely used in the phase field literature is the historical solver, but it limits the crack geometry to be used. To get around this limitation, you need another solver. This work shows the implementation of a bound-constrained solver through the external library PETSc (Portable, Extensible Toolkit for Scientific Computation) applied in *INSANE*. With this new solver, the new possibilities of models are shown, the performance is compared with other solvers in *INSANE*, the convergence and mesh dependence are compared with the smeared crack model, and finally, the capacity of the phase-field model to capture modes I and II failures is evaluated.

Key-words: Phase field models of Fracture; Bound-Constrained Solver; Finite Element Method based Phase-field models.

List of figures

List of tables

List of abbreviations and acronyms

BLAS	Basic Linear Algebra Subprograms
CMSD	Crack Mouth Sliding Displacement
CST	Constant Strain Triangle
DI	Displacement Iteractions
FEM	Finite Element Method
GI	Global Iteractions
INSANE	INteractive Structural ANalysis Environment
JDK	Java Development Kit
JNI	Java Native Interface
LAPACK	Linear Algebra Package
MKL	Math Kernel Library
MOD	Mouth Open Displacement
OOP	Object-Oriented Programming
PDE	Partial Differential Equation
PETSc	Portable, Extensible Toolkit for Scientific Computation
PFI	Phase-Field Iteractions
PFM	Phase-Field Modelling

List of symbols

E_0	initial elasticity modulus
ν	Poison coefficient
ϕ	phase-field
ϕ'	derivative of ϕ with respect to x
l_0	length scale parameter
G_c	critical energy release rate
t	time
ϵ	perturbation
∇	nabla operator
$\mathcal{G}$	variation of strain energy functional with respect to the crack surface
γ	crack surface density function
$\dot{\gamma}$	crack surface density rate
λ_0	first Lamé constant
μ_0	second Lamé constant
ψ_0^+	active strain energy density
ψ_0^-	inactive strain energy density
H	Heaviside function
Γ	crack surface
$\dot{\Gamma}$	crack surface rate
Ω	body domain
$\partial\Omega$	body boundary
$\partial\Omega_u$	body boundary with prescribed displacements
$\partial\Omega_t$	body boundary with prescribed forces
$\mathcal{B}$	damage part of the body
$\partial\mathcal{B}$	damaged surface of the body
Y	crack driving force
$\bar{Y}$	effective crack driving force
g	energetic degradation function
α	geometrical crack function
C_0	parameter associated with the crack geometric function

p	exponent associated with energy degradation function
Q	continuous function associated with energy degradation function
c_1	parameter associated with energy degradation function
c_2	parameter associated with energy degradation function
c_3	parameter associated with energy degradation function
w	crack opening
w_c	ultimate apparent displacement jump
k_0	initial slope for the softening curve
n_w	parameter associated with cornelissen´s softening law
n_k	parameter associated with cornelissen´s softening law
η_1	parameter associated with cornelissen´s softening law
η_2	parameter associated with cornelissen´s softening law
ψ_0	elastic strain energy density
ψ	strain energy
P_{ext}	external load potential
E_t	energy functional
Ψ_c	crack surface energy
Ψ_s	strain energy
δE_t	first variation of energy functional
$\delta\Psi_c$	first variation of crack surface energy
$\delta\Psi_s$	first variation of strain energy
$\delta\Gamma$	first variation of crack surface
$\dot{E}_t$	energy rate functional
$\dot{\Psi}_c$	crack surface energy rate
$\dot{\Psi}_s$	strain energy rate
$\dot{\Gamma}$	crack surface rate
$\bar{b}$	body forces vector
$\bar{u}$	displacements vector
$\bar{t}$	surface forces vector
$\bar{n}$	surface normal vector
$\delta\bar{u}$	first variation displacements vector
$\dot{\bar{b}}$	body forces rate vector
$\dot{\bar{u}}$	displacements rate vector
$\dot{\bar{t}}$	surface forces rate vector
$I(\phi, \phi')$	functional for phase-field
$\underline{\sigma}$	stress tensor
$\underline{\varepsilon}$	strain tensor
$\underline{\varepsilon}_{ac}^{D}$	deviatoric part of the strain tensor
$\underline{\varepsilon}_{D}$	deviatoric stress tensor

$\underline{\varepsilon}_V$	volumetric stress tensor
$\underline{\varepsilon}^-$	inactive strain tensor
$\underline{\varepsilon}^+$	active strain tensor
$\dot{\underline{\varepsilon}}$	strain rate tensor
$\hat{\mathbf{E}}_0$	linear elastic isotropic constitutive tensor
$\hat{\mathbf{C}}$	non linear constitutive tensor
$\otimes$	outer product
$\underline{\sigma_{eq}}$	equivalent stress
$\underline{J_2}$	second invariant of the deviatory stress tensor calculated from stress tensor
$\underline{\sigma_1}$	higher stress between the principal stresses
β_c	parameter associated with equivalent stress
f_c	uniaxial compressive strength
f_t	uniaxial tensile strength
ε_c	elastic limit deformation in compression
ε_t	elastic limit deformation in tension
I	index relative to nodes
$[\mathbf{N}]_I^u$	shape functions relative to the displacement field displacement
$[\mathbf{N}]_I^\phi$	shape functions relative to the phase-field for each node
$\bar{d}_I$	nodal displacement vector for each node
$\bar{a}_I$	nodal phase-field vector for each node
$[\mathbf{B}]_I^u$	matrix B of finite element methods for displacements for each node
$[\mathbf{B}]_I^\phi$	matrix B of finite element methods for phase-field for each node
$\bar{r}^u$	residual for displacements
$\bar{r}^u$	residual for phase-fied
$[\mathbf{K}]_{IJ}$	full stiffness matrix
$[\mathbf{K}]_{IJ}^{uu}$	portion of the stiffness matrix associated only with displacements
$[\mathbf{K}]_{IJ}^{u\phi}$	portion of the stiffness matrix associated with displacements and phase-field
$[\mathbf{K}]_{IJ}^{\phi u}$	portion of the stiffness matrix associated with phase-field and displacements
$[\mathbf{K}]_{IJ}^{\phi\phi}$	portion of the stiffness matrix associated only with phase-field

Table of contents

Chapter 1

Introduction

One of several important issues in structural engineering is the modelling of cracking, and a scope of research is the appearance of cracks and the path they will take. Therefore, it is important to predict in which situations this will happen, so that projects can be made with greater reliability. Cracking can be described in two ways, the discrete and the continuous forms. In the discrete approach, cracks are modelled as displacement discontinuities in the domain, while in the continuous approach the displacements are continuous, but the stiffness is gradually reduced to model the material degradation process. One of the ways to treat the degradation process, that has been increasingly used, is the phase-field approach.

The phase field model generalizes the Griffith theory, aiming to solve the displacement field and the fracture region by energy minimization and no assumptions for the evolution of the cracks are necessary (Goswami et al., 2020).

The phase-field is a scalar variable that goes from 0 for the undamaged state of the material to 1 for the fully cracked state of the material. In addition to the phase-field variable, another parameter is the length scale parameter, that controls the width of the region where the discrete crack is smoothed on. Small values of this length tends to reproduce the Griffith's theory.

The phase-field has already been widely discussed by several authors. A complete review on the theme of phase-field, which addresses several references on the subject, can be found in Wu et al. (2020). The phase-field research group of the Department of Structural Engineering (DEES) has already implemented in *INSANE* (INteractive Structural ANalysis Environment) the bases of the phase-field theory as shown in Leão (2021). An important question on phase-field models is how the issue of crack irreversibility is handled, since from a physical point of view, a crack cannot heal over time (without external intervention). Depending on the used strategy, there may be limitations on the model. In order to overcome such limitations in *INSANE*, the research group developed two works simultaneously. One of them was one by Bayao et al. (2021), which implemented a bound-constrained solver inside *INSANE*. In Bayão (2021) there is a broader approach to all the

details of this bound-constrained solver in *INSANE*. The other work is this dissertation that was produced using the external library PETSc (Balay et al., 2020) recommended by the literature, so that the research group could have different possible solvers.

The computational implementations were done in the software *INSANE* (INteractive Structural ANalysis Environment) using the library PETSc (Portable, Extensible Toolkit for Scientific Computation). *INSANE* is an open source software based on the Object-Oriented Programing paradigm and developed since 2002 at the Department of Structural Engineering of the Federal University of Minas Gerais (Penna, 2007, Fuina, 2004). PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. In order to use PETSc, a binding will be made (JNI - Java Native Interface) to *INSANE*, because *INSANE* is written in Java and PETSc written in C.

To generate the mesh and as post-processor, the Gmsh[1] (Geuzaine and Remacle, 2009) and Paraview[2] (Ayachit, 2015, Ahrens et al., 2015) software were used.

1.1 Major Objective and Specific Objectives

The main objective of this work was to understand the phase field theory and implement the new bounded solver through PETSc, that addresses the condition of irreversibility without limiting the models. The main consequence of such an implementation is the possibility of studying phase-field models without the limitation of the crack geometry function that will be shown.

In order to achieve the main objective, the following propositions were made:

- Do the binding of PETSc and INSANE.
- Implement the bound-constrained solver in INSANE, using the previously implementation by Leão (2021), with the minimum amount of modifications.
- Model fracture problems using different functions for crack geometry and energy degradation.
- Compare the obtained results with other solvers for phase-field in *INSANE*.
- Compare the results of the phase-field with smeared crack model.
- Validate phase-field models for modes I and II failures.

1.2 Outline

This work is organized in 6 chapters and 2 appendices. After this first chapter, Chapter 2 presents the theoretical foundation of PFM (Phase-Field Model) and the theory for bound-constrained solver. Chapter 3 presents the additions made to *INSANE* code for what was

[1]Gmesh software can be download at `https://gmsh.info/`

[2]Paraview software can be download at `https://www.paraview.org/`

implemented in this work. Chapter 4 presents a numerical study of the bound-constrained solver by PETSc, showing the possibilities associated, an analysis of the results in relation to the other solvers and an analysis of time performance in relation to the other solvers. Chapter 5 presents two case studies. A study showing a comparisson between the phase-field and a smeared cracking model and another study presenting a validation through numerical models for the ability of phase-field models to capture modes I and II failures. Finally, Chapter 6 closes the manuscript, summing up the main contributions of this work and discussing future developments. The Appendix A shows how to make the connection between *INSANE* and PETSc, and Appendix B presents the demonstration for some equations developed in the work.

Chapter 2

Theoretical Foundation

In this chapter, the theory of phase-field models and the idea of the bound-constrained solver will be presented.

This chapter presents a review of the phase-field approach to fracture. The implementation of of the phase-field theory base in *INSANE* was done by Leão (2021), as well as the complete development of equations used in this work. In this work, were adopted $\underline{\cdot}$ for tensor, $\bar{\cdot}$ for vector and $[\cdot]$ for matrix.

To start, let's consider the problem described in Figure 2.1, which is a body whose problem domain is Ω with external boundary $\partial\Omega$ and a crack Γ. The external boundary region relative to displacements is described by $\partial\Omega_u$, while the one relative to loading is described by $\partial\Omega_t$.

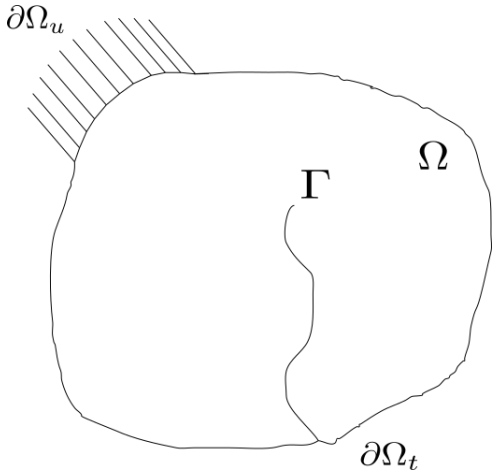


Figure 2.1: A solid body with a crack.

The total energy functional E_t is given by:

$$E_t(\bar{u}, \phi) = \Psi_s + \Psi_c - P_{ext} \tag{2.1}$$

where Ψ_s is the strain energy, Ψ_c is the surface energy due to the crack, and P_{ext} is the external load potential energy functional.

The strain energy Ψ_s of a general solid considering phase-field depends on the strain tensor $\underline{\varepsilon}(\bar{u})$, where $\bar{u}$ is the displacements field, and phase-field ϕ:

$$\Psi_s(\bar{u}, \phi) = \int_\Omega \psi(\underline{\varepsilon}(\bar{u}), \phi) \ \mathrm{d}\mathcal{V} \tag{2.2}$$

where ψ is the strain energy density. The surface energy due to crack growth, is given by:

$$\Psi_c = \int_\Omega G_c \gamma(\phi, \nabla\phi) \ \mathrm{d}\mathcal{A} \tag{2.3}$$

where G_c is a material property that represents the critical energy release rate, γ is the crack surface density function. γ makes the crack smoothing in the phase-field model, it will be addressed in Section 2.3. The external potential energy, P_{ext} is computed using the prescribed boundary force ($\bar{t}$) and the distributed body force ($\bar{b}$).

$$P_{ext} = \int_\Omega \bar{b} \cdot \bar{u} \ \mathrm{d}\mathcal{V} + \int_{\partial\Omega} \bar{t} \cdot \bar{u} \ \mathrm{d}\mathcal{A} \tag{2.4}$$

2.1 Smoothing representation of the discrete crack

As already mentioned, the phase-field (ϕ) is a continuous variable, that can be between 0 and 1, where 0 represents the unbroken state of the material and 1 represents the fully broken state of the material. Since it can take on any value in this range, an exponential decay function, proposed by Miehe et al. (2010b), is introduced to approximate the non-smooth crack topology. To understand the idea let's assume an infinite bar of cross-section Γ, under axial traction, that has a crack at the position x = 0. Figure 2.2 illustrates the function of the phase-field variable versus the axial position.

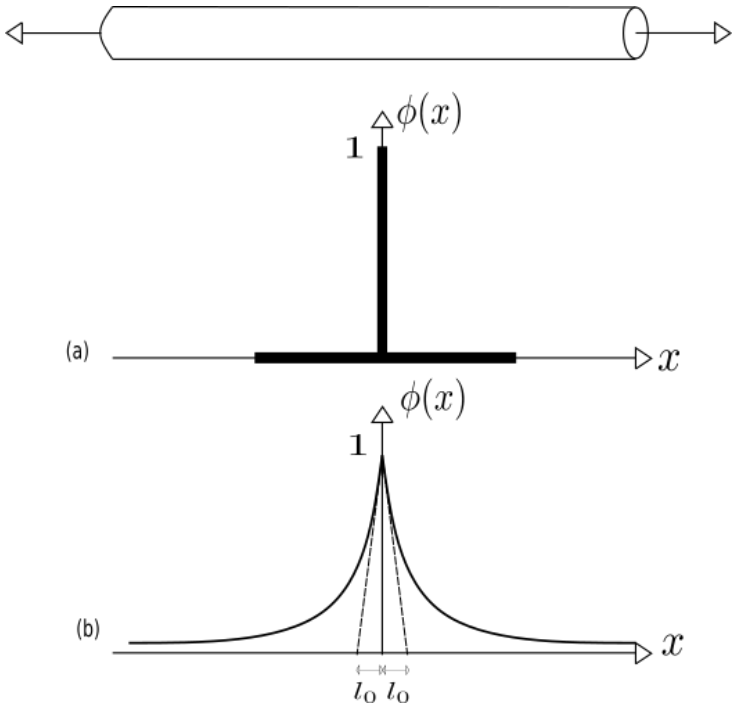


Figure 2.2: Infinity bar with sharp and diffusive crack modeling under it. (a) Sharp crack at x = 0 and (b) diffusive crack at x = 0 modeled with the length scale l_0. Adapted from Miehe et al. (2010b).

l_0 is the length scale parameter that regulates the amount of diffusion of the crack. In the limiting case of sharp crack topology, $l_0 \to 0.0$. When l_0 becomes larger, the broken region of the bar also becomes larger. Then, such parameter introduces a smooth transition between the broken and unbroken state.

From Miehe et al. (2010a), a particular form of the phase field is:

$$\phi(x) = e^{-|x|/l_0} \tag{2.5}$$

where Equation 2.5 gives the phase-field value as a function of the axial position x, in such way that, in the center of the bar the section is fully broken and at $x \to \infty$ the section is fully unbroken. The Equation 2.5 is a solution for minimization of the differential equation whose solution is given by funtional (also give by Miehe et al. (2010a)):

$$I(\phi, \phi') = \frac{1}{2} \int_\Omega \phi^2 + l_0^2 (\phi')^2 \ \mathrm{d}\mathcal{V} \tag{2.6}$$

Solving (2.6) for (2.5), it give us:

$$I(\phi, \phi') = \Gamma l_0 \tag{2.7}$$

So, it can say that, the crack surface area of the sharp crack can be given by:

$$\Gamma_l = \frac{1}{l_0} I(\phi, \phi') = \int_\Omega \gamma(\phi, \phi') \ \mathrm{d}\mathcal{V} \tag{2.8}$$

where the *crack surface density function* is defined as:

$$\gamma(\phi, \phi') = \frac{1}{2l_0}\phi^2 + \frac{l_0}{2}(\phi')^2 \tag{2.9}$$

Without loss of generality, for a 3D case, the Functional 2.8 can be given by:

$$\Gamma_l = \frac{1}{l_0} I(\phi, \nabla\phi) = \int_\Omega \gamma(\phi, \nabla\phi) \ \mathrm{d}\mathcal{V} \tag{2.10}$$

where:

$$\gamma(\phi, \nabla\phi) = \frac{1}{2l_0}\phi^2 + \frac{l_0}{2}|\nabla\phi|^2 \tag{2.11}$$

2.2 Equations of Phase-Field Models in Strong Form

To obtain the strong form of the Phase-Field problem, a domain Ω with a broken part $\mathcal{B} \subset \Omega$ is considered. $\mathcal{B}$ is the region where the fracture is located and smoothed by the phase-field model. The boundary of the solid and its broken surface are, respectively, $\partial\Omega$ and $\partial\mathcal{B}$. The total energy functional E_t becomes:

$$\begin{aligned} E_t = \int_\Omega \psi(\underline{\varepsilon}(\bar{u}), \phi) \ \mathrm{d}\mathcal{V} + \int_\mathcal{B} G_c \gamma(\phi, \nabla\phi) \ \mathrm{d}\mathcal{V} \\ - \int_\Omega \bar{b} \cdot \bar{u} \ \mathrm{d}\mathcal{V} - \int_{\partial\Omega} \bar{t} \cdot \bar{u} \ \mathrm{d}\mathcal{A} \end{aligned} \tag{2.12}$$

To find the displacement and the phase-field $(\bar{u}, \phi)$, we need to minimize the Equation 2.12, whose first variation is:

$$\begin{aligned} \delta E_t = &- \int_\Omega (\nabla \cdot \underline{\sigma} + \bar{b}) \cdot \delta\bar{u} \ \mathrm{d}\mathcal{V} + \int_{\partial\Omega_t} (\underline{\sigma} \cdot \bar{n} - \bar{t}) \cdot \delta\bar{u} \ \mathrm{d}\mathcal{A} \\ &+ \int_\mathcal{B} \left[\frac{\partial\psi}{\partial\phi} + G_c \delta_\phi \gamma \right] \delta\phi \ \mathrm{d}\mathcal{V} + \int_{\partial\mathcal{B}} G_c \left(\frac{\partial\gamma}{\partial\nabla\phi} \cdot \bar{n} \right) \delta\phi \ \mathrm{d}\mathcal{A} \end{aligned} \tag{2.13}$$

where:

$$\delta_\phi \gamma = \left(\frac{\partial\gamma}{\partial\phi} - \nabla \cdot \frac{\partial\gamma}{\partial\nabla\phi} \right) \tag{2.14}$$

From Equation 2.13 the following conditions can be obtained:

$$\nabla \cdot \underline{\sigma} + \bar{b} = \bar{0} \text{ in the domain } \Omega \tag{2.15a}$$

$$\underline{\sigma} \cdot \bar{n} - \bar{t} = \bar{0} \text{ in the domain } \partial\Omega_t \tag{2.15b}$$

Equations 2.15a and 2.15b are the equilibrium equation and the boundary condition

of a boundary value problem in classic elasticity.

Defining $Y = -\dfrac{\partial \psi}{\partial \phi}$ (*crack driving force*), it can be shown that (Leão, 2021):

$$\begin{cases} Y - G_c \delta_\phi \gamma = 0, \text{ for } \dot{\phi} > 0 \\ Y - G_c \delta_\phi \gamma < 0, \text{ for } \dot{\phi} = 0 \end{cases} \tag{2.16}$$

$$\frac{\partial \gamma}{\partial \nabla \phi} \cdot \bar{n} = 0 \text{ in } \partial \mathcal{B} \tag{2.17}$$

The variable γ does not change the value on the boundary $\partial \mathcal{B}$, then Equation 2.17 is a boundary condition.

It is hypothesized that ψ depends on a degradation function $g(\phi)$ as follows:

$$\psi(\bar{\varepsilon}, \phi) = \psi(\bar{\varepsilon}, g(\phi)) \tag{2.18}$$

Therefore, using the chain rule, the *crack driving force* can be rewritten as:

$$Y = -\frac{\partial \psi}{\partial \phi} = -\frac{\partial \psi}{\partial g}\frac{\partial g}{\partial \phi} \tag{2.19}$$

The energy degradation function $g(\phi)$ and its proprieties will be discussed in Section 2.4.

Introducing the *effective crack driving force* $\bar{Y} = \dfrac{\partial \psi}{\partial g}$, Equation 2.19 leads to:

$$Y = -g'(\phi)\bar{Y} \tag{2.20}$$

From all this developed theory it can be observed:

1. **Irreversibility condition**

   The crack only grows, thus the crack opening is irreversible. In terms of the functional Γ_l, discussed on Section 2.1, this results in:

   $$\dot{\Gamma}_l = \int_{\mathcal{B}} \dot{\gamma}(\phi, \nabla\phi) dV \geq 0 \tag{2.21}$$

   with:

   $$\dot{\gamma} = \frac{\partial \gamma}{\partial \phi}\dot{\phi} + \frac{\partial \gamma}{\partial \nabla \phi}\nabla\dot{\phi} \tag{2.22}$$

   After some manipulations, and applying the divergence theorem and the boundary conditions, Equation 2.21 becomes:

   $$\dot{\Gamma}_l = \int_{\mathcal{B}} \dot{\phi}\ \delta_\phi \gamma\ \ \mathrm{d}\mathcal{V} \geq 0 \tag{2.23}$$

   Since $\dot{\phi} \geq 0$, it is necessary that $\delta_\phi \gamma \geq 0$. These are the irreversibility conditions.

2. **Phase-field evolution equation**

Introducing $f(Y,\phi) = Y - G_c\delta_\phi\gamma$ the Karush-Kuhn-Tucker conditions become:

$$\dot{\phi} \geq 0, \quad f(Y,\phi) \leq 0, \quad \dot{\phi}f(Y,\phi) = 0 \tag{2.24}$$

with $G_c\delta_\phi\gamma$ being the limit in which the crack will grow. In analogy to a sharp crack, this therm is the critical energy release rate (G_c).
Considering $\dot{\phi} > 0$, from equation system 2.16, and considering $\bar{Y}$ as defined on Equation 2.20, the following can be obtained:

$$-g'(\phi)\bar{Y} - G_c\delta_\phi\gamma = 0 \longrightarrow \delta_\phi\gamma = \frac{-g'(\phi)\bar{Y}}{G_c} \tag{2.25}$$

It can be observed that the phase-field is zero when $\delta_\phi\gamma$ is zero. Then, a high value of G_c can be adopted as a strategy to prevent crack formation in certain regions.

2.3 Generalisation of the crack surface density function

The crack surface density function γ in Equation 2.11 is one form that can be used to regularise the sharp crack topology. There are other ways to represent the crack surface density function. Wu (2017) proposed a general equation to describe the surface density function of the crack:

$$\gamma(\phi,\nabla\phi) = \frac{1}{C_0}\left[\frac{1}{l_0}\alpha(\phi) + l_0|\nabla\phi|^2\right] \tag{2.26a}$$

$$\delta_\phi\gamma = \frac{1}{C_0}\left[\frac{1}{l_0}\alpha'(\phi) - 2l_0\Delta\phi\right] \tag{2.26b}$$

where it is introduced the *crack geometry function* $\alpha(\phi)$ and the parameter $C_0 = 4\int_0^1 \alpha^{1/2}(\phi)d\phi$. The values of C_0 described by the integral are graphically represented in Figure 2.3. Furthermore, C_0 can be represented by Equation 2.27 given by Wu (2017).

$$C_0 = \begin{cases} \frac{1}{(1-\xi)^{3/2}}\left[\frac{1}{2}\xi^2\ln\left(\frac{\xi}{2\sqrt{1-\xi}+2-\xi}\right) + (2-\xi)\sqrt{1-\xi}\right] & \xi \in (0,1) \\ \frac{1}{(\xi-1)^{3/2}}\left[\frac{1}{2}\xi^2\left(\frac{\pi}{2} - \arcsin\left(\frac{2-\xi}{\xi}\right)\right) - (2-\xi)\sqrt{1-\xi}\right] & \xi \in (1,2] \end{cases} \tag{2.27}$$

A detail to be aware of is that at points $\xi = 0$ and $\xi = 1$ the function is not defined, but it converges to values 2 and 8/3, respectively for each point, as seen in Figure 2.3 and in the literature. Therefore, in the implementation, a conditional is needed to handle this detail.

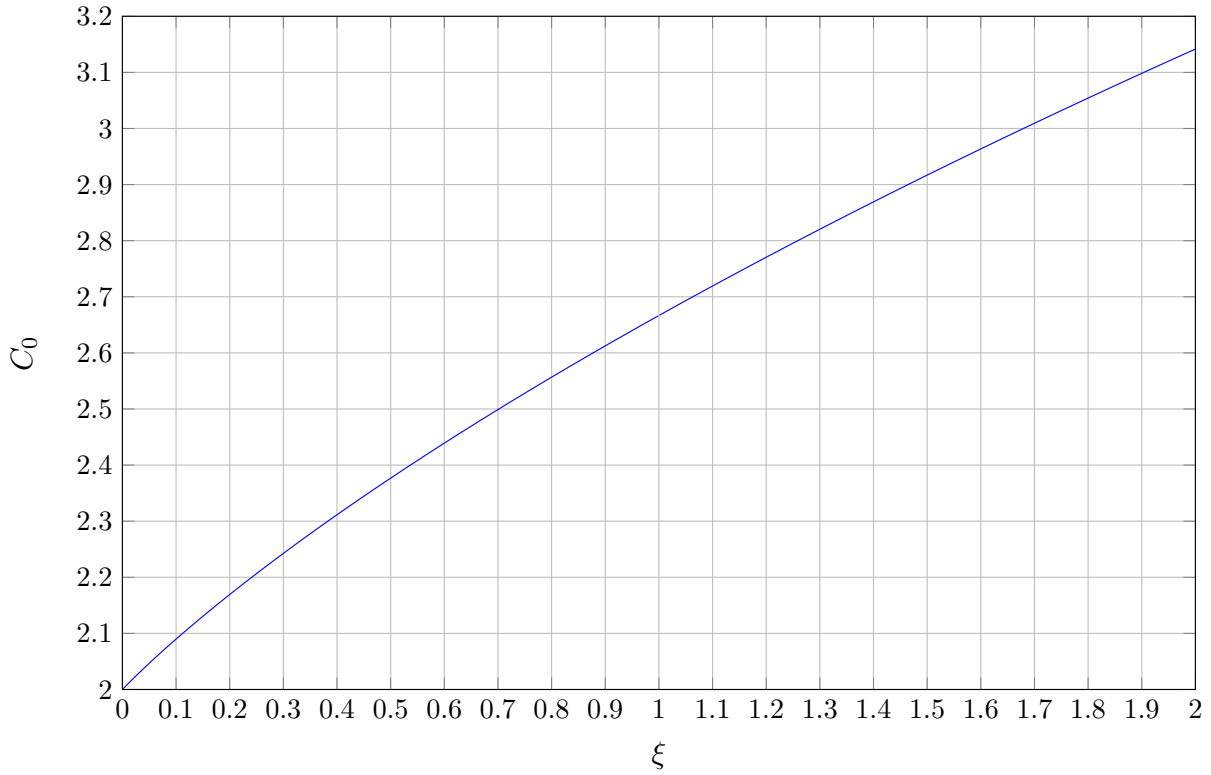


Figure 2.3: Values for C_0 for different ξ.

The function $\alpha(\phi)$ determines how the phase-field will be distributed and it has to satisfy the following properties (according to Wu et al. (2020)):

$$\alpha(0) = 0 \quad \text{and} \quad \alpha(1) = 1 \tag{2.28}$$

There are many functions already proposed in the literature for $\alpha(\phi)$. The functions adopted by Leão (2021) are shown in Table 2.1.

Table 2.1: Crack geometry function $\alpha(\phi)$ implemented by Leão (2021) in *INSANE*

$\alpha(\phi)$	c_0	authors
ϕ	8/3	Pham et al. (2011)
ϕ^2	2	Bourdin et al. (2000)
$1-(1-\phi)^2$	π	(Alessi et al., 2015)
$16\phi^2(1-\phi)^2$	8/3	Karma et al. (2001)

The crack geometry function through higher-oder polynomials can be considered as well. Wu (2017) proposed the following quadratic one:

$$\alpha(\phi) = \xi\phi + (1-\xi)\phi^2 \in [0,1] \quad \forall\phi \in [0,1] \tag{2.29}$$

where $\xi \in [0,2]$, otherwise $\alpha(\phi) \in [0,1]$ cannot be guaranteed, as shown in Figure 2.4. For various values of $\xi \in [0,2]$, the resulting crack phase-fields $\phi(x)$ are summarised in

Table 2.2 and in Figure 2.5. The localisation bandwidth decreases with larger value ξ. The Equation 2.29 has already been implemented in *INSANE* as shown in Bayao et al. (2021).

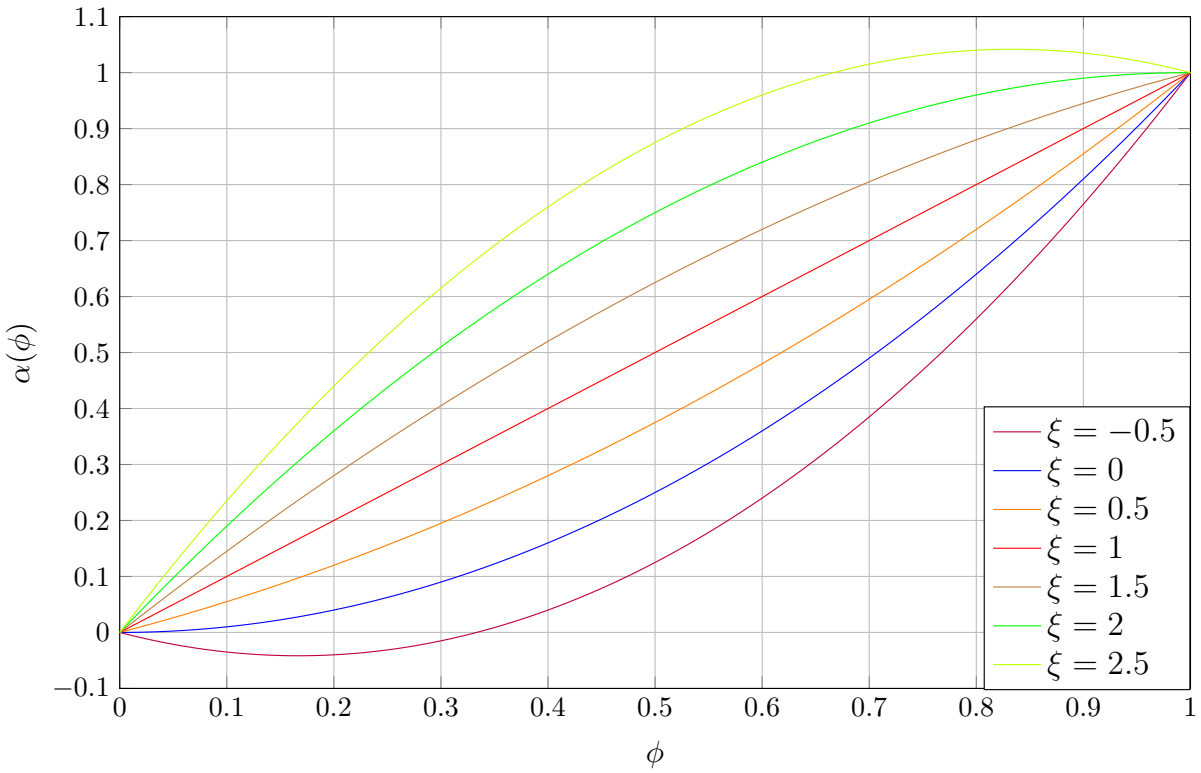


Figure 2.4: Phase-field for different crack geometry function.

Table 2.2: Generic crack geometry function $\alpha(\phi)$ and the resulting crack phase-field $\phi(x)$. Adapted from Wu et al. (2020)

$\alpha(\phi)$	ξ	c_0	ϕ(x)
ϕ^2	0	2	$\exp(-\frac{\lvert x\rvert}{l_0})$
ϕ	1	8/3	$(1-\frac{\lvert x\rvert}{2l_0})^2$
$2\phi-\phi^2$	2	π	$1-\sin\frac{\lvert x\rvert}{l_0}$

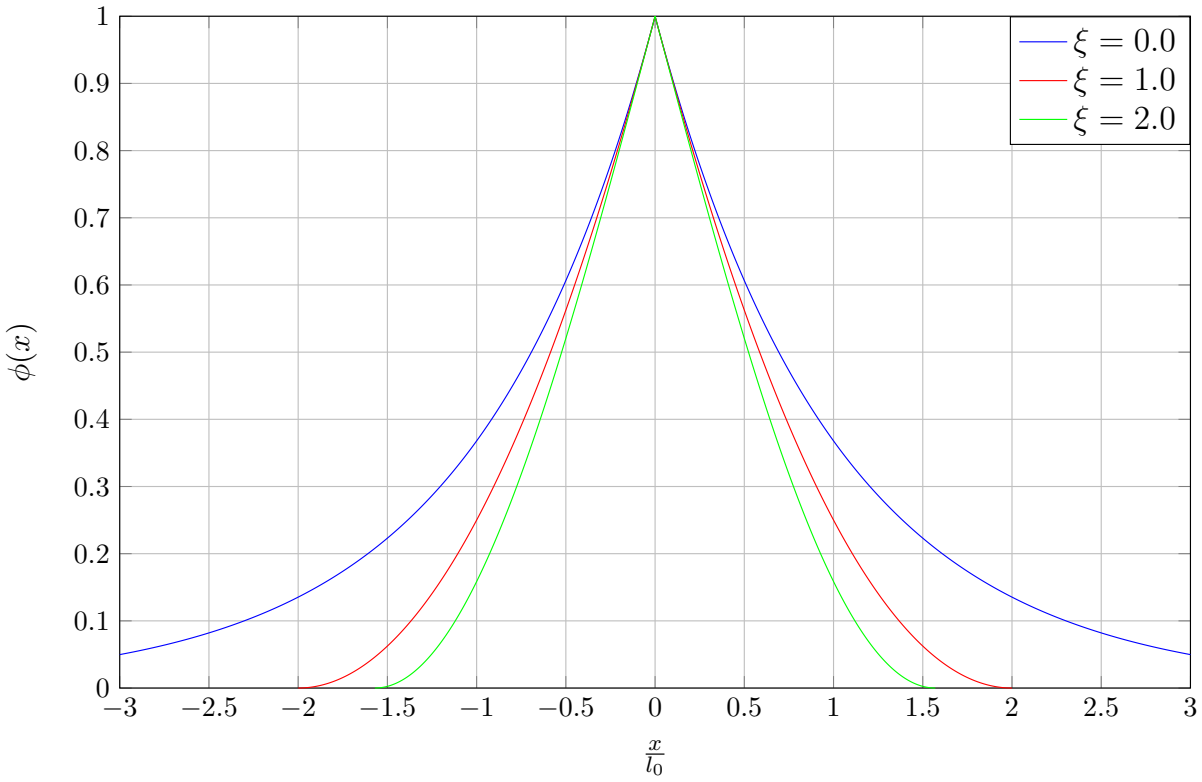


Figure 2.5: Plots of different geometric functions.

2.4 Energy Degradation Function

The energy degradation function, $g(\phi)$, makes the connection between the crack phase-field and the mechanical fields. The energy degradation function $g(\phi)$ has to satisfy the following conditions:

- $g(0) = 1$: there is no degradation in intact material;
- $g(1) = 0$: the energy is completely degraded in fully broken material;
- $g'(\phi) = \dfrac{dg}{d\phi} < 0$: the function $g(\phi)$ has to be monotonically decreasing;
- $g'(1) = 0$: there isn't sudden variation in the interface where the material in fully broken.

There are many functions already proposed in the literature. The functions implemented by Leão (2021) in *INSANE* are shown in Table 2.3 and its graphs are plotted on Figures 2.6 and 2.7.

Table 2.3: Energy degradation functions

$g(\phi)$	authors
$(1-\phi)^2$	Bourdin et al. (2000)
$3(1-\phi)^2 - 2(1-\phi)^3$	Karma et al. (2001)
$4(1-\phi)^3 - 3(1-\phi)^4$	Kuhn et al. (2015)

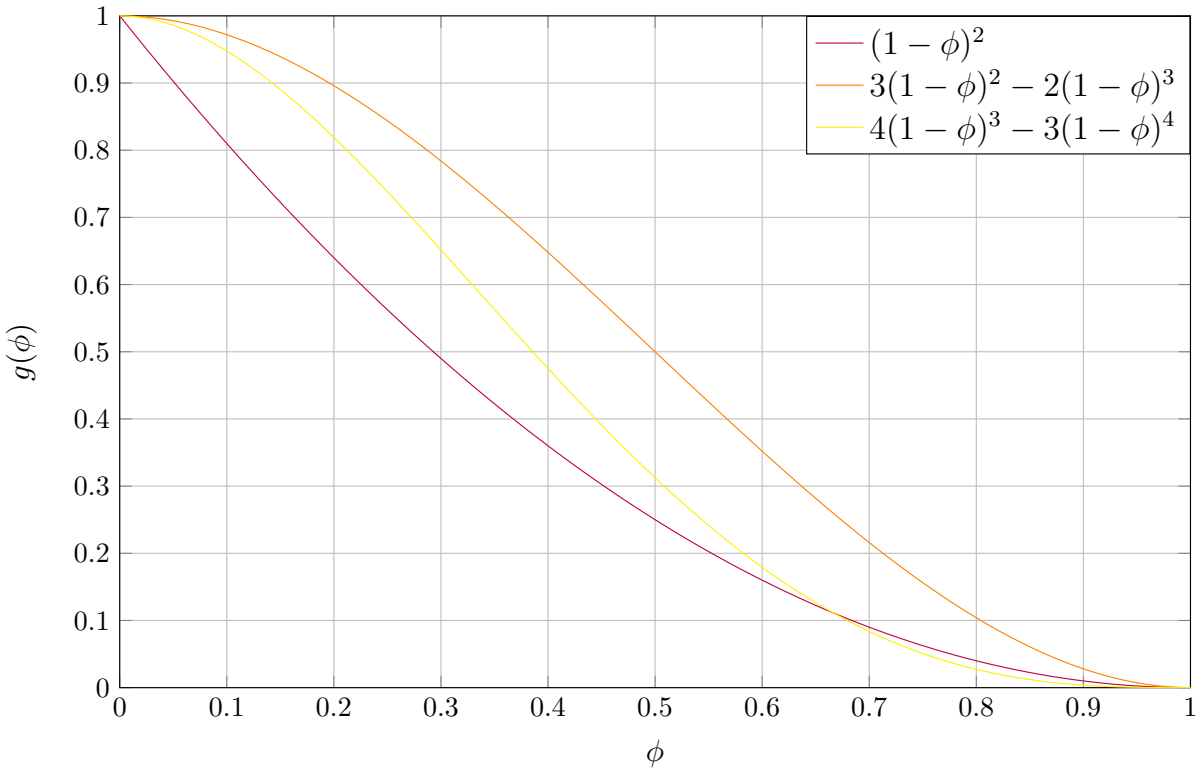


Figure 2.6: Function $g(\phi)$

It can be observed that the cubic and quartic functions have $g'(0) = 0$. These functions represent materials that have an initial linear elastic behaviour. On the other hand the quadratic function have $g'(0) < 0$, meaning that material degradation is activated as soon as the loading starts, as illustrated in Figure 2.7.

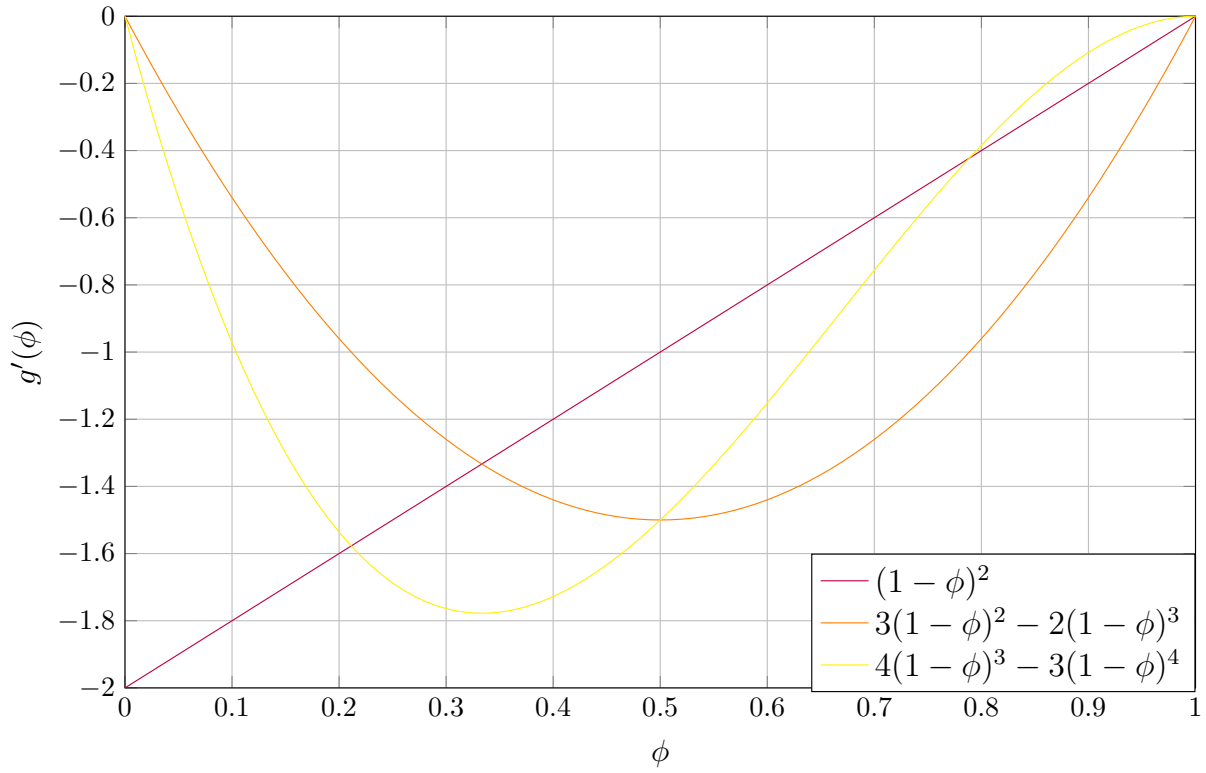


Figure 2.7: Function $g'(\phi)$

There are also other functions in the literature that construct $g(\phi)$ based on parameters that build $\alpha(\phi)$ other than ϕ, showing once again the importance of using an unlimited solver without the $\alpha(\phi)$ limitation. An example of these functions is the function proposed by Wu (2017):

$$g(\phi) = \frac{(1-\phi)^p}{(1-\phi)^p + Q(\phi)} \tag{2.30}$$

where the exponent $p > 0$ and continuous function $Q(\phi) > 0$ is exposed in Equation 2.31.

$$Q(\phi) = c_1\phi + c_1c_2\phi^2 + c_1c_2c_3\phi^3 + \cdots \tag{2.31}$$

where c_1, c_2 and c_3 are given by:

$$c_1 = \frac{2E_0G_c}{f_t^2}\frac{\xi}{C_0l_0} = \frac{2\xi}{C_0}\frac{l_{ch}}{l_0} \tag{2.32a}$$

$$c_2 = \frac{1}{\xi}[(-\frac{4\pi\xi^2}{C_0}\frac{G_c}{f_t^2}k_0) - (p+1)] \tag{2.32b}$$

$$c_3 = \begin{cases} 0 & p > 2 \\ \frac{1}{c_2}[\frac{1}{\xi}(\frac{C_0w_cf_t}{2\pi G_c})^2 - (1+c_2)] \end{cases} \tag{2.32c}$$

where E_0 is Young´s modulus, f_t is the uniaxial tensile strength and l_{ch} (characteristic

length) $= E_0 G_c / f_t^2$ for Griffith´s or Irwin´s Theory. To understand the w_c parameter, assume a bar $x \in [-L, L]$ sufficiently long such that crack evolution is not affected by boundary effects. The bar is loaded at both ends by increasing displacements in opposite directions. The distributed body forces are neglected. For simplicity, it is assumed that the crack is initiated at the symmetric point $x = 0$. Thinking about this scenario, w_c is the ultimate apparent displacement jump (ultimate crack opening) and k_0 is the initial slope for the softening curves, which represent the relationship between stress(σ) and apparent displacement jump(w) across the localization band. It is possible to obtain w_c through Equation 2.33, and k_0 through Equation 2.34. This theory, as well as these equations are shown in Wu (2017),

$$w_c = \frac{2\pi G_c}{C_0 f_t} \sqrt{\xi \frac{Q(1)}{c_1}} \lim_{\phi_*} (1 - \phi_*)^{(1-\frac{p}{2})} \tag{2.33}$$

$$k_0 = -\frac{C_0}{4\pi} \frac{f_t^2}{G_c} \frac{[\xi(c_2 + p + 1) - 1]^{\frac{3}{2}}}{\xi^2} \tag{2.34}$$

where ϕ_* is the maximum phase-field.

2.4.1 Softening Law

An alternative to determine the values of k_0 and w_c, instead of using the equations, is to use the parameterized values for already established softening laws. Softening laws describe the behavior after the material cracks. The expressions for stress in this section are defined by Wu (2017), where they represent the behavior of stress after cracking, that is, in energy dissipation. Wu (2017) defined expressions for w_c and k_0 for some softening laws. It is worth mentioning that the expressions for k_0 are obtained through $\sigma'(0) = k_0$.

2.4.1.1 Linear Softening Law

For the linear softening law we have the Equations 2.35, 2.36 and 2.37.

$$\sigma(w) = f_t \max(-k_0 w, 0) \tag{2.35}$$

$$k_0 = -\frac{f_t^2}{2G_c} \tag{2.36}$$

$$w_c = \frac{2G_c}{f_t} \tag{2.37}$$

To achieve the value of w_c from Equation 2.37, the value of p must be 2 (Wu et al., 2020).

2.4.1.2 Exponential Softening Law

For the exponential softening law we have the Equations 2.35, 2.36 and 2.37.

$$\sigma(w) = f_t \exp(-\frac{ft}{G_c}w) \tag{2.38}$$

$$k_0 = -\frac{f_t^2}{G_c} \tag{2.39}$$

$$w_c = \infty \tag{2.40}$$

Infinite w_c implies a value of c_3 equal to zero. Wu (2017) adopts a p value of 5/2, through data fitting.

2.4.1.3 Hyperbolic Softening Law

For the hyperbolic softening law we have the Equations 2.41, 2.42 and 2.43.

$$\sigma(w) = f_t(1 + \frac{f_t}{G_c}w)^{-2} \tag{2.41}$$

$$k_0 = -\frac{2f_t^2}{G_c} \tag{2.42}$$

$$w_c = \infty \tag{2.43}$$

Infinite w_c implies a value of c_3 equal to zero. Wu (2017) adopts a p value of 4, through data fitting.

2.4.1.4 Cornelissen's Softening Law

Cornelissen's softening is based on Cornelissen et al. (1986). For the Cornelissen´s softening law we have the Equations 2.44, 2.45 and 2.46.

$$\sigma(w) = f_t[(1 + \eta_1^3 r^3)\exp(-\eta_2 r) - r(1 + \eta_1^3)\exp(-\eta_2)] \tag{2.44}$$

$$k_0 = n_k \frac{f_t}{w_c} \tag{2.45}$$

$$w_c = \frac{1}{n_w}\frac{G_c}{f_t} \tag{2.46}$$

,where $r = w/w_c$. The values for n_k and n_w are given by the Equations 2.47 and 2.48, respectively. The typical values $\eta_1 = 3.0$ and $\eta_2 = 6.93$ have been considered for normal concrete (Cornelissen et al., 1986).

$$n_k = -n_2 - (1 + n_1^3)\exp(-n_2) \tag{2.47}$$

$$n_w = -[(1 + n_1^3)(0.5 + \frac{1}{n_2}) + \frac{3n_1^3}{n_2^2}(1 + \frac{2}{n_2} + \frac{2}{n_2^2})]\exp(-n_2) + \frac{1}{n_2} + 6\frac{n_1^3}{n_2^4} \tag{2.48}$$

To develop equation 2.46, it is necessary to integrate equation 2.44 from 0 to w_c, as this equation gives the value of G_c. For more details, see the appendix Appendix B.

For the softening laws presented, the Table 2.4 shown some values of c_2 and c_3 according softening law, ξ and p.

Table 2.4: Examples of c_2 and c_3

Softening Law	p	ξ	c_2	c_3
Linear Softening	2	2	$-\frac{1}{2}$	0
Exponencial Softening	$\frac{5}{2}$	2	0.1748	0
Hyperbolic Softening	4	2	0.5379	0
Cornelissen´s Softening (normal concrete)	2	2	1.3868	0.6567

The energy degradation function proposed by Wu (2017) was implemented by Bayao et al. (2021) at *INSANE* as well as the possibility of choosing the softening laws.

2.5 The strain energy density

The strain energy function describes a smooth transition between the fully broken and unbroken material. To describe it is used the *initial strain energy density function* $\psi_0(\bar{\varepsilon})$ and the degradation function $g(\phi) : [0, 1] \longrightarrow [1, 0]$.

Assuming the material to be linear elastic, the strain energy density is given by:

$$\psi_0(\underline{\varepsilon}) = \frac{1}{2}\underline{\sigma} : \underline{\varepsilon} = \frac{1}{2}\underline{\varepsilon} : \hat{\mathbf{E}}_0 : \underline{\varepsilon} = \frac{1}{2}\lambda_0 tr^2(\underline{\varepsilon}) + \mu_0 \underline{\varepsilon} : \underline{\varepsilon} \tag{2.49}$$

where λ_0 and μ_0 are the Lamé constants.

An anisotropic formulation based on the following additive decomposition of the elastic strain energy is commonly adopted in the literature to prevent crack formation in compression regions:

$$\psi_0(\underline{\varepsilon}) = \psi_0^+(\underline{\varepsilon}) + \psi_0^-(\underline{\varepsilon}) \tag{2.50}$$

where $\psi_0^+(\underline{\varepsilon})$ is the part that comes from tensile strains and $\psi_0^-(\underline{\varepsilon})$ is the part due to compressions. It is called anisotropic model (Miehe et al., 2010a, Amor et al., 2009) because the tensile and compression energies are splited; if the model was not split, it

would be called an isotropic model (Bourdin et al., 2000). These models will be better covered in the Section 2.5.1. The degradation is then assumed to affect just the tensile part:

$$\psi(\underline{\varepsilon}) = g(\phi)\psi_0^+(\underline{\varepsilon}) + \psi_0^-(\underline{\varepsilon}) \tag{2.51}$$

The stress field, the constitutive tensor $\hat{\mathbf{C}}$ and the evolution phase-field law becomes:

$$\sigma = \frac{\partial \psi}{\partial \underline{\varepsilon}} = g(\phi)\frac{\partial \psi_0^+(\underline{\varepsilon})}{\partial \underline{\varepsilon}} + \frac{\partial \psi_0^-(\underline{\varepsilon})}{\partial \underline{\varepsilon}} \tag{2.52a}$$

$$\hat{\mathbf{C}} = \frac{\partial \underline{\sigma}}{\partial \underline{\varepsilon}} = g(\phi)\frac{\partial^2 \psi_0^+(\underline{\varepsilon})}{\partial \underline{\varepsilon}^2} + \frac{\partial^2 \psi_0^-(\underline{\varepsilon})}{\partial \underline{\varepsilon}^2} \tag{2.52b}$$

$$Y = G_c\delta_\phi\gamma = -g'(\phi)\bar{Y},\ \bar{Y} = \frac{\partial \psi}{\partial g} = \psi_0^+(\underline{\varepsilon}) \tag{2.52c}$$

2.5.1 Some split energy models

There are different forms in the literature to divide ψ into ψ_0^+ and ψ_0^-. The complete development of the formulations of the Isotropic Constitutive Model, Lancioni and Royer-Carfagni (2009), Amor et al. (2009) and Miehe et al. (2010b) models are presented in Leão (2021). Here a brief summary of the models will be presented. First, to develop the models it is necessary to define some functions:

- **Ramp function:** Returns the value itself, if it has the same sign as the function, or zero in other cases.
$$\langle x \rangle_\pm = \frac{|x| \pm x}{2} \tag{2.53}$$
- **Sign function:** Returns 1 if the value is positive, -1 if it is negative, or 0 if it is zero.
$$\mathrm{sgn}(x) = \begin{cases} -1 & \text{, if } x < 0 \\ 0 & \text{, if } x = 0 \\ 1 & \text{, if } x > 0 \end{cases} \tag{2.54}$$
- **Heaviside Function:** Returns 1 if the value is positive, 0 if it is negative, or 0.5 if it is zero.
$$\mathrm{H}(x) = \frac{1 + \mathrm{sgn}(x)}{2} \tag{2.55}$$
- $R_n^\pm$ **functions:** Applied to verify the signal of the strain tensor trace and are defined as:
$$R_n^\pm = \mathrm{H}(\pm \mathrm{tr}(\underline{\varepsilon})) \tag{2.56}$$

From Equation 2.56 it is observed that:

$$\langle \mathrm{tr}(\underline{\varepsilon})\rangle_{\pm} = R_n^{\pm}\mathrm{tr}(\underline{\varepsilon}) \tag{2.57}$$

Those definitions will help to demonstrate the constitutive and stress tensors of the Amor et al. (2009) and Miehe et al. (2010b) models.

2.5.1.1 Isotropic Constitutive Model

In this model, no split is considered and the free energy density function $\psi(\bar{\varepsilon}, \phi)$.

$$\begin{aligned} \psi_0^+ &= \frac{1}{2}\lambda_0 \mathrm{tr}(\varepsilon_{ij})(\varepsilon_{11}+\varepsilon_{22}) + \mu(\varepsilon_{11}^2+\varepsilon_{22}^2+2\varepsilon_{12}^2) \\ \psi_0^- &= 0 \end{aligned} \tag{2.58}$$

2.5.1.2 Lancioni and Royer-Carfagni (2009) Constitutive Model

In this model, shear fracture is considered and the strain tensor $\bar{\varepsilon}$ is decomposed in a volumetric part and a deviatoric one.

$$\begin{aligned} \psi_0^+ &= \mu_0\, \underline{\varepsilon}_{pl}^D : \underline{\varepsilon}_{pl}^D \\ \psi_0^- &= \frac{1}{2}\lambda \mathrm{tr}(\underline{\varepsilon})(\varepsilon_{11}+\varepsilon_{22}) + \\ &+\mu_0\left(\frac{2}{3}\mathrm{tr}(\underline{\varepsilon})(\varepsilon_{11}+\varepsilon_{22}) - \frac{2}{9}(\mathrm{tr}(\underline{\varepsilon}))^2\right) \end{aligned} \tag{2.59}$$

where in $\underline{\varepsilon}_{pl}^D$ the $\cdot^D$ means that is deviatoric stress tensor and the $\cdot_{pl}$ means tensor is plane case.

2.5.1.3 Amor et al. (2009) Constitutive Model

They proposed the following positive/negative parts:

$$\begin{aligned} \psi_0^+ &= \mu_0\, \underline{\varepsilon}_{pl}^D : \underline{\varepsilon}_{pl}^D + \frac{1}{2}\lambda_0 R_n^+ \mathrm{tr}(\varepsilon)(\varepsilon_{11}+\varepsilon_{22}) + \\ &+\mu_0 R_n^+\left(\frac{2}{3}\mathrm{tr}(\varepsilon)(\varepsilon_{11}+\varepsilon_{22}) - \frac{2}{9}(\mathrm{tr}(\varepsilon))^2\right) \\ \psi_0^- &= \frac{1}{2}\lambda_0 R_n^- \mathrm{tr}(\varepsilon)(\varepsilon_{11}+\varepsilon_{22}) + \\ &+\mu_0 R_n^-\left(\frac{2}{3}\mathrm{tr}(\varepsilon)(\varepsilon_{11}+\varepsilon_{22}) - \frac{2}{9}(\mathrm{tr}(\varepsilon))^2\right) \end{aligned} \tag{2.60}$$

2.5.1.4 Miehe et al. (2010b) Constitutive Model

The model by Miehe et al. (2010b) part of the spectral decomposition of the strain tensor and the definition of active ε^+ and inactive ε^- components. The spectral decomposition of the strain tensor is given by equation 2.61

$$\underline{\varepsilon} = \sum_{n=1}^{3} \varepsilon_n \bar{p}_n \otimes \bar{p}_n = \underline{\varepsilon}^+ + \underline{\varepsilon}^- \tag{2.61}$$

where $\underline{\varepsilon}^+$ and $\underline{\varepsilon}^-$ are, respectively, the active and inactive strain tensors, defined by:

$$\underline{\varepsilon}^+ = \sum_{n=1}^{3} \langle \varepsilon_n \rangle_+ \bar{p}_n \otimes \bar{p}_n \tag{2.62a}$$

$$\underline{\varepsilon}^- = \sum_{n=1}^{3} \langle \varepsilon_n \rangle_- \bar{p}_n \otimes \bar{p}_n \tag{2.62b}$$

where ε_n and $\bar{p}_n$ represent, respectively, the eigenvalues and eigenvectors of the strain tensor.The strain energy density is separated from equations 2.63a and 2.63b.

$$\psi_0^+ = \frac{1}{2}\lambda_0 R_n^+ (\mathrm{tr}(\underline{\varepsilon}))^2 + \mu_0 \underline{\varepsilon}^+ : \underline{\varepsilon}^+ \tag{2.63a}$$

$$\psi_0^- = \frac{1}{2}\lambda_0 R_n^- (\mathrm{tr}(\underline{\varepsilon}))^2 + \mu_0 \underline{\varepsilon}^- : \underline{\varepsilon}^- \tag{2.63b}$$

For the plane case, the equations are:

$$\begin{aligned} \psi_0^+ &= \frac{1}{2}\lambda_0 R_n^+ \mathrm{tr}(\underline{\varepsilon})(\varepsilon_{11} + \varepsilon_{22}) + \mu_0 \underline{\varepsilon}_{pl}^+ : \underline{\varepsilon}_{pl}^+ \\ \psi_0^- &= \frac{1}{2}\lambda_0 R_n^- \mathrm{tr}(\underline{\varepsilon})(\varepsilon_{11} + \varepsilon_{22}) + \mu_0 \underline{\varepsilon}_{pl}^- : \underline{\varepsilon}_{pl}^- \end{aligned} \tag{2.64}$$

where $\underline{\varepsilon}_{pl}^+$ and $\underline{\varepsilon}_{pl}^-$ corresponds to the plane case of $\underline{\varepsilon}^+$ and $\underline{\varepsilon}^-$.

2.5.1.5 Wu (2017) Constitutive Model

According to Wu et al. (2020), hybrid models were developed as an alternative to maintain the asymmetric behavior with respect to tension and compression, but at the same time avoiding excessive non-linearity caused by the definition of $\bar{Y}$ by the part deformation energy, which makes solving the displacements little efficient in models that make separation of variables. In this hybrid model, ψ_0 is determined by equation 2.65:

$$\psi_0 = \frac{1}{2E_0}{\sigma_{eq}}^2 \tag{2.65}$$

, where σ_{eq} is the equivalent stress calculated from equations 2.66 and 2.67.

$$\underline{\sigma} = \hat{\mathbf{E}}_0 : \underline{\varepsilon} \tag{2.66}$$

$$\sigma_{eq} = \frac{1}{1+\beta_c}\left(\beta_c\langle\sigma_1\rangle + \sqrt{3J_2}\right) \tag{2.67}$$

,where J_2 is the second invariant of the deviatory stress tensor calculated from $\underline{\sigma}$, σ_1 is the largest between the principal stresses and $\beta_c = \frac{f_c}{f_t - 1}$, for f_c the uniaxial compressive strength and f_t uniaxial tensile strength. This model was implemented by Bayao et al. (2021).

2.5.1.6 Wu (2018) Constitutive Model

In this hybrid model, ψ_0 is determined by equation 2.68:

$$\psi_0 = \frac{1}{2E_0}{\sigma_1}^2 \tag{2.68}$$

where σ_1 denotes the major principle value of the effective stress $\underline{\sigma}$. According to Wu (2018), comparing with the formulation in Section 2.5.1.5, the contributions of the deviating effective stress tensors are neglected. This is justified for failures induced by tensile cracks. Details on the implementation of this model at *INSANE* will be addressed in Section 3.2.1.

2.6 Equations of Phase-Field Models in Weak Form

The governing equations of a phase-field model are:

$$\begin{cases} \int_\Omega \underline{\sigma} : \delta\underline{\varepsilon} \ \mathrm{d}\mathcal{V} = \delta P_{ext} \\ \int_\mathcal{B} \left[g'(\phi)\bar{Y}\delta\phi + G_c\delta\gamma\right] \ \mathrm{d}\mathcal{V} \geq 0 \end{cases} \tag{2.69}$$

with:

$$\delta\gamma = \frac{1}{C_0}\left[\frac{1}{l_0}\alpha'(\phi)\delta\phi + 2l_0\nabla\phi\cdot\nabla\delta\phi\right] \tag{2.70}$$

The first equation of 2.69 is the standard weak form of classical elasticity while the second one is obtained from 2.13 (for a detailed proof of this derivation, see e.g. the dissertation by Leão (2021)).

Pre-defined cracks can be considered in modelling by setting $\phi = 1$ in crack points or by mesh discretization. To prevent cracks on regions it is necessary to put $\phi = 0$ in region points or set a very high G_c value. This can be a helpful artifice in support points, where there is a concentrated load.

2.7 Finite Element Discretization

In this section, the equations will be illustrated for the 2D case. In the Finite Element Method, the displacement field and the strain field are written in terms of nodal displacements:

$$\bar{u}(\bar{x}) = [\mathbf{N}]_I^u \, \bar{d}_I \tag{2.71a}$$

$$\bar{\varepsilon}(\bar{x}) = [\mathbf{B}]_I^u \, \bar{d}_I \tag{2.71b}$$

where $\bar{d}$ is the nodal displacements vector and, for each node in the element, it has:

$$[\mathbf{N}]_I^u = \begin{bmatrix} N_I^u & 0 \\ 0 & N_I^u \end{bmatrix} \tag{2.72}$$

$$[\mathbf{B}]_I^u = \begin{bmatrix} N_{I,x}^u & 0 \\ 0 & N_{I,y}^u \\ N_{I,y}^u & N_{I,x}^u \end{bmatrix} \tag{2.73}$$

where I are the nodes. In a similar way, the phase-field is interpolated from its nodal values:

$$\phi(\bar{x}) = [\mathbf{N}]_I^\phi \, \bar{a}_I \tag{2.74a}$$

$$\nabla\phi(\bar{x}) = [\mathbf{B}]_I^\phi \, \bar{a}_I \tag{2.74b}$$

where $\bar{a}$ is the nodal phase-field vector and, for each node in element, it has:

$$[\mathbf{N}]_I^\phi = \begin{bmatrix} N_I^\phi \end{bmatrix} \tag{2.75}$$

$$[\mathbf{B}]_I^\phi = \begin{bmatrix} N_{I,x}^\phi \\ N_{I,y}^\phi \end{bmatrix} \tag{2.76}$$

From those definitions, the Finite Element Discretization of Equations 2.69 can be written as

$$\int_\Omega ([\mathbf{B}]_I^u)^T \bar{\sigma} \, \mathrm{d}\mathcal{V} = \bar{f}_{ext} \tag{2.77}$$

$$\begin{aligned} \int_{\mathcal{B}} g'\bar{Y}([\mathbf{N}]_I^{\phi})^T \, \mathrm{d}\mathcal{V} + \\ + \int_{\mathcal{B}} \frac{G_c}{C_0} \left(\frac{1}{l_0}\alpha'([\mathbf{N}]_I^{\phi})^T + 2l_0([\mathbf{B}]_I^{\phi})^T \nabla\phi \right) \, \mathrm{d}\mathcal{V} \geq \bar{0} \end{aligned} \tag{2.78}$$

From that, the residual form of Equations 2.77 and 2.78 above can be obtained:

$$\bar{r}^u = \int_{\Omega} ([\mathbf{B}]_I^u)^T \bar{\sigma} dV - \bar{f}_{ext} = \bar{0} \tag{2.79}$$

$$\begin{aligned} \bar{r}^{\phi} = - \int_{\mathcal{B}} ([\mathbf{N}]_I^{\phi})^T \left(g'\bar{Y} + \frac{1}{C_0 l_0}\alpha' G_c \right) \, \mathrm{d}\mathcal{V} \\ - \int_{\mathcal{B}} \frac{2l_0}{C_0} G_c ([\mathbf{B}]_I^{\phi})^T \nabla\phi \, \mathrm{d}\mathcal{V} \leq \bar{0} \end{aligned} \tag{2.80}$$

The linearisation of the residuals above leads to the following stiffness matrix:

$$[\mathbf{K}]_{IJ} = \begin{bmatrix} [\mathbf{K}]_{IJ}^{uu} & [\mathbf{K}]_{IJ}^{u\phi} \\ [\mathbf{K}]_{IJ}^{\phi u} & [\mathbf{K}]_{IJ}^{\phi\phi} \end{bmatrix} \tag{2.81}$$

where:

$$[\mathbf{K}]_{IJ}^{uu} = \int_{\Omega} ([\mathbf{B}]_I^u)^T \frac{\partial \bar{\sigma}}{\partial \bar{\varepsilon}} [\mathbf{B}]_J^u \, \mathrm{d}\mathcal{V} \tag{2.82a}$$

$$[\mathbf{K}]_{IJ}^{u\phi} = \int_{\Omega} ([\mathbf{B}]_I^u)^T \frac{\partial \bar{\sigma}}{\partial \phi} [\mathbf{N}]_J^{\phi} \, \mathrm{d}\mathcal{V} \tag{2.82b}$$

$$[\mathbf{K}]_{IJ}^{\phi u} = \int_{\mathcal{B}} ([\mathbf{N}]_I^{\phi})^T g' \frac{\partial \bar{Y}}{\partial \bar{\varepsilon}} [\mathbf{B}]_J^u \, \mathrm{d}\mathcal{V} \tag{2.82c}$$

$$\begin{aligned} [\mathbf{K}]_{IJ}^{\phi\phi} = \int_{\mathcal{B}} ([\mathbf{N}]_I^{\phi})^T \left(g''\bar{Y} + \frac{1}{C_0 l_0}\alpha'' G_c \right) [\mathbf{N}]_J^{\phi} \, \mathrm{d}\mathcal{V} + \\ + \int_{\mathcal{B}} \frac{2l_0}{C_0} G_c ([\mathbf{B}]_I^{\phi})^T [\mathbf{B}]_J^{\phi} \, \mathrm{d}\mathcal{V} \end{aligned} \tag{2.82d}$$

2.8 Solvers

The classification of phase-field solvers has two differentiations. Solvers can be classified as to how they solve equations and how they deal with crack irreversibility.

2.8.1 Solvers regarding the solution to the equation

To solve Equation 2.81 there are two classes of solvers used in the literature which are monolithic and staggered solvers. Both of them were implemented by Leão (2021) in *INSANE* project. The monolithic solves the system of equations 2.81 together, in other

words, solves displacement and phase-field at the same time. Whereas the staggered solver solve the phase-field and displacement separately, solving only the parts $[\mathbf{K}_I^{uu}]$ and $[\mathbf{K}_I^{\phi\phi}]$ from 2.81. The energy functional is non-convex in both unknown variables (u, ϕ), then the monolithic solver performs poorly. The models analyzed by Leão (2021) show that using monolithic solver, the results diverge when the structure starts to be non linear. In this work we only use staggered solver.

2.8.2 Irreversibility condition

When using a solver, it is necessary to guarantee the irreversibility of the crack and the domain of phase-field variation ($\phi \in [0, 1]$), that is, the second equation of system 2.69 have to be guaranteed. There are a few ways to do this, discussed in the literature. For example, Bourdin et al. (2000) enforced the irreversibility condition when the crack phase-field is close to one: $\phi(x, t > t_0) = 1$ if $\phi(x, t_0) \approx 1$. Miehe et al. (2010b) considered the *effective crack driving force* ($\bar{Y}$) as a historical variable ($\mathcal{H}$) that represents the maximum tensile energy the material had experimented. This strategy has been widely used in the literature, and is the one adopted by Leão (2021), but cannot guarantee crack irreversibility for non-quadratic crack geometry functions $\alpha(\phi) \neq \phi^2$. It is worth mentioning here that when $\alpha = \phi^2$, the value of $\xi = 0$ in Equation 2.29, therefore, the energy degradation function 2.30 cannot be used, as it would imply in indeterminate c_2. This makes the degradation energy function to be used with the historical solver ($\alpha = \phi^2$) to be $g = (1-\phi)^2$ (Bourdin et al., 2000), therefore not having, for example, the parameter f_t in the material description, being something important for damage models. So the historical solver limits the crack geometric function but also the degradation energy function, which greatly influences the results as it will be shown in Section 4.2.1.

Amor et al. (2009) considered the phase-field evolution equation as a bound-constrained optimization problem. Heister et al. (2015) proposed a technique named primal-dual active set method to solve optimization problem and deal with convexify the energy functional. Many articles are based on Heister et al. (2015) modifying it, like done by Hu et al. (2020). Basically, the primal-dual active set method uses lagrange multipliers to define two sets, one set called inactive and one active. The active set represents the subdomain with the restrictions applied and where no PDE (Partial Differential Equation) is resolved, while in the inactive set the PDE is solved, and the restrictions are satisfied. The inactive set is used to find the phase-field variation over time (in each step). Another work that uses lagrange multipliers to restrict irreversibility is the one of Geelen et al. (2019).

Other authors proposed to minimize the energy subject to a constraint condition. Some authors have used some kind of bound-constrained solver to handle this constraint condition. Li et al. (2016) wrote the phase-field problem with quadratic functional, solves

through a scheme Gradient Projection Conjugate Gradient in PETSc (Portable, Extensible Toolkit for Scientific Computation) (Balay et al., 2020). Farrell and Maurini (2017) and Wu (2018) used PETSc for solving a reduced-space active-set method, based on Benson and Munson (2006). More about this method will be discussed in next section, since in this work this strategy was adopted to ensure crack irreversibility without limiting the crack geometry function and energy degradation function. This method is rather robust, but its convergence rate is slow. Then, to overcome this issue, Farrell and Maurini (2017) improved computational efficiency through a nonlinear Gauss-Seidel iterative scheme and used an over-relaxed parameter to accelerate the iteration.

2.8.3 Bound-Constrained Solver

To deal with the boundedness and irreversibility conditions, it's convenient to regard the governing equation from residual phase-field as an optimization problem bounded by the following conditions:

$$0 \leq a_{I,n} \leq a_{I,n+1} \leq 1 \tag{2.83}$$

According to Farrell and Maurini (2017), under the above condition the residual phase-field constitutes a mixed complementarity problem, that can be written as:

$$\begin{cases} a_{I,n} < a_{I,n+1} < 1 \quad r_I^{\phi} = 0 \\ a_{I,n} = a_{I,n+1} \quad r_I^{\phi} \leq 0 \\ a_{I,n+1} = 1 \quad r_I^{\phi} \geq 0 \end{cases} \tag{2.84}$$

where r_I^{ϕ} is the residual of the phase-field. Equation 2.84 says that the solution of residual phase-field needs to be for each node, precisely, one of the conditions shown. This work intends to use a reduced-space active set Newton method, included in the the open-source toolkit PETSc (Balay et al., 2020), to solve this problem, just as Wu et al. (2020) and Farrell and Maurini (2017) did. This solver is based on Benson and Munson (2006), and its main idea is illustrated in the Figure 2.8.

The k index in Figure 2.8 refers local phase-field iteration. In the algorithm the active set is the subdomain with the restrictions applied and no equation is solved, while in the inactive set the equations are solved, in such way that the restrictions are satisfied. So the idea of the algorithm is to reduce the number of equations to be solved at each step, as it updates the inactive set.

Data: a^0, $k = 0$
Result: A solution of (r^ϕ)
Given a^0, the initial guess
while $||\bar{r}^\phi(a^k)|| > TOL$ **do**
  Compute active($\mathscr{A}$) and inactive($\mathscr{I}$) sets:
  $\mathscr{A}(a) = a_{I,n+1} = a_{I,n}$ `and` $r^\phi < 0$ *or* $a_{I,n+1} = 1$ `and` $r^\phi > 0$
  $\mathscr{I}(a)$ = the remaining nodes: $a_{I,n+1} = a_{I,n}$ `and` $r^\phi > 0$ *or* $a_{I,n+1} = 1$ `and` $r^\phi < 0$ `and` $1 > a_{I,n+1} > a_{I,n}$
  Set $\bar{dir}_{\mathscr{A}} = 0$, where $\bar{dir}$ is the direction of the line search
  Solve the reduce Newton step for $\bar{dir}_{\mathscr{I}}$:
  $[\nabla \bar{r}^\phi(a^k)]_{\mathscr{I}^k,\mathscr{I}^k} \bar{dir}_{\mathscr{I}^k} = -\bar{r}^\phi_{\mathscr{I}^k}(a^k)$
  Choose the step length μ such that $||r^\phi||^2$ is minimized, via line search on $\bar{a}^{k+1} = \bar{a}^k + \mu\, dir$; if this search direction fails, use the steepest descent direction instead.
  k = k +1
end

Figure 2.8: Reduced-space active-set method adapted from Benson and Munson (2006)

The objective of this work is not to implement this code itself, but to communicate *INSANE* with PETSc, which already has this implementation, so that it solves and returns the new value for nodal phase-field. In order to use PETSc, a binding was made to *INSANE*, because *INSANE* is written in Java and PETSc written in C. To do this, a Java Native Interface (JNI) was based on the work of Azevedo (2019). It is worth remembering that Bayao et al. (2021), produced of the same time as this work, implemented in *INSANE* a bound-constrained solver in the program itself, without the use of external libraries, which will be used to compare results and performance.

Chapter 3

Implementation

In this chapter the *INSANE* system will be presented together with all the changes that was necessary to implement the Wu (2018) constitutive model and the bound-constrained solver by PETSc.

3.1 A brief introduction to *INSANE*

The *INSANE* System (INteractive Structural ANalysis Environment) is an open-source software developed at the Structural Engineering Department of UFMG. Almost all the code is developed in Java and the full potential of object-oriented programming (OOP) is used.

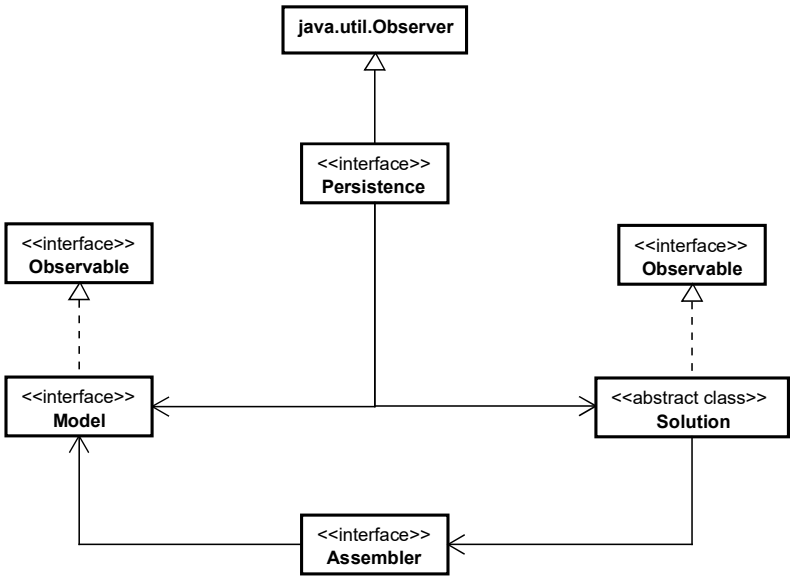


Figure 3.1: *INSANE* core organization (Penna, 2011)

As pointed out in Figure 3.1, *INSANE* core is composed by the interfaces `Model`, `Solution` and `Assembler` that are responsible to abstract and to solve the model. The

`Persistence` collects and writes the input and output data in files and `Model` stores the lists of nodes, elements, loadings, etc. `Assembler` is responsible to mount the system of equations and `Solution` has the necessary methods to solve the problem.

The classes and interfaces necessary for the implementation of this work are the `ConstitutiveModel` for implementation of Wu (2018) constitutive model and `Step` for bound-constrained solver by PETSc. The `ConstitutiveModel` is responsible to calculate the constitutive material relations and the stress. The `Step` implements the necessary methods to solve each step of a non linear analysis, for example, the Standard Newton Raphson. More information of *INSANE* working and its organization is very well described in Penna (2011).

3.2 Modifications in *INSANE* structure

In the UML diagrams presented throughout this section, the modified classes will be depicted in yellow, the new classes in green and the non modified classes in white (Fig. 3.2).

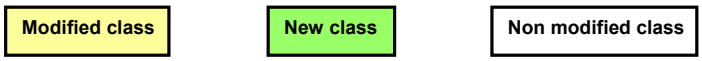


Figure 3.2: Classes representation in UML diagram.

3.2.1 Implementation of Wu (2018) constituive model

The constitutive models of Phase-Field in the *INSANE* system are defined differently for the monolithic and sttagered solution methods. The constitutive model Wu (2018), implemented in this work, has as class superior to PhaseFieldStaggeredConstitutiveModel, which defines default methods for all classes developed for staggered solver.

As described in Section 2.5.1.5, the model in question is defined as a hybrid, as it does not have in its formulation the realization of the separation of components, but, even so, define a asymmetric behavior for tensile and compressive stresses. In this way, the formulations closely resemble the models without component separation.

The strategy used for the implementation was the same used for the `StgPfWu2013ConstModel` class (which represents the Wu (2017) constitutive model) in Bayão (2021), that is, the definition of this new model as a class inferior, `StgPfWu2018ConstModel`, to the isotropic model class of Bourdin et al. (2000), `StgPfIsotropicConstModel`. Thus, it was only necessary to override the "mount-DualInternalVariableVector()" that returns the stress of each node and updates the value of $\bar{Y}$.

The Figure Fig. 3.3 shows the implementation of the Wu (2018) constitutive model in *INSANE* and the other phase-field models already implemented. It is worth mentioning

that there are several other constitutive models in *INSANE*, but only those that are related to phase-field were represented.

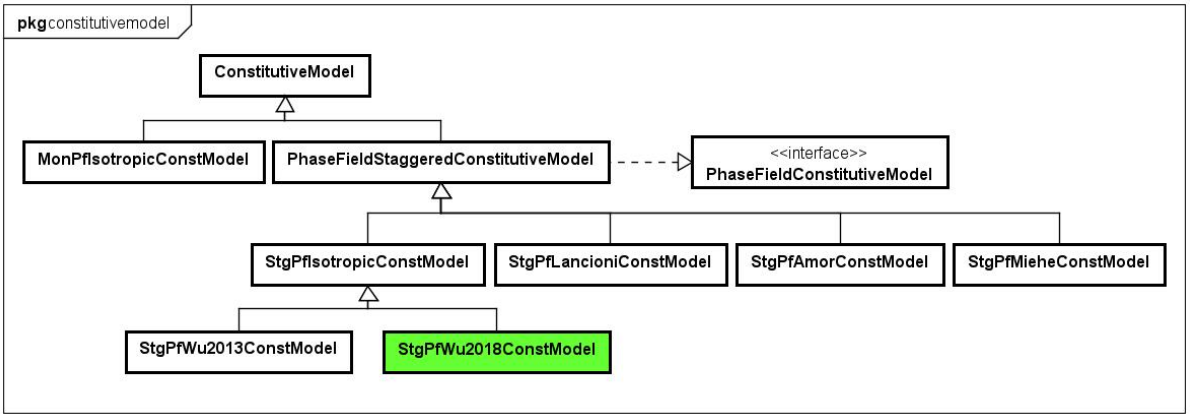


Figure 3.3: Diagram with phase-field constitutive models and the Wu (2018) constitutive model.

3.2.2 Implementation of Bound-Constrained Solver

Leão (2021) created a class called `PhaseFieldStandardNewtonRaphsonStaggeredSolver` for the staggered solver, that is, a class that solves phase-field and displacements in alternating iterations. To use PETSc only for the phase-field, a class was created that extends `PhaseFieldStandardNewtonRaphsonStaggeredSolver` (see Figure 3.4) and overrides the "convergePhaseField()" method, which will now call PETSc.

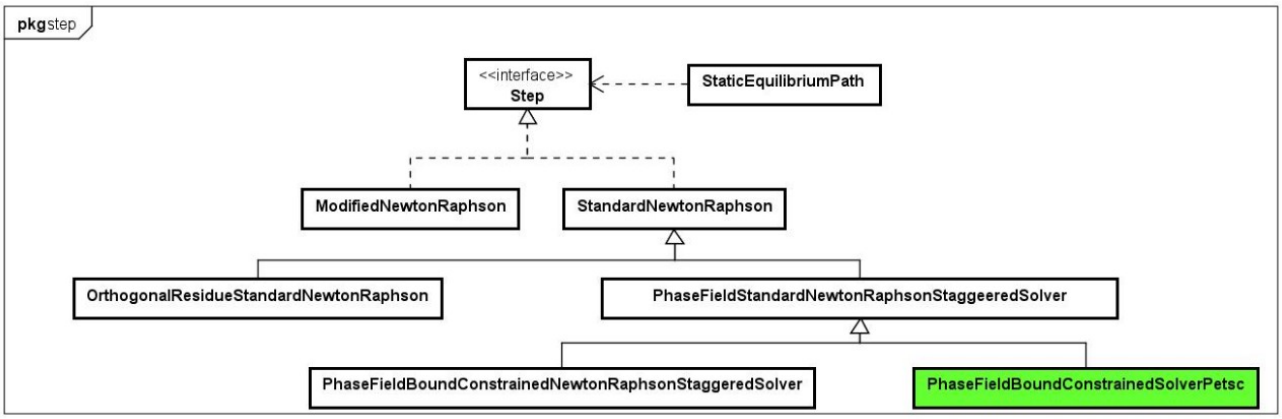


Figure 3.4: Diagram with PhaseFieldBoundConsrainedSolverPetsc.

The solver idea was demonstrated in Section 2.8.3, and it basically has two parts. One is splitting the nodes into active and inactive sets and resolving through the inactive sets to find the new phase-field value, while the other part is line search. When implementing the solver by PETSc in *INSANE*, this first part was implemented, without using a line search. The convergence for the solver by PETSc is the same used by Leão (2021), which will be discussed later, so the objective of PETSc is just to find the new phase-field value

obeying the crack irreversibility but without the crack geometric and energy degradation functions limitation.

For the PETSc call, we need to pass four arguments, the phase-field stiffness matrix, the phase-field residual, the phase-field lower limit and the phase-field value of the current local iteration. As explained in Section 2.8.3, the bound-constrained solver has two limits. The upper bound is 1, it comes from the phase-field theory itself and is defined in the library's C code. The lower limit will be the phase-field of the current step, so that the crack does not decrease, guaranteeing the irreversibility of the crack. The phase-field value of the current local iteration will serve as a guess for the phase-field value to be found in the new iteration. Section 3.3 will talk more about PETSc and the implementation related to it.

In the `PhaseFieldStandardNewtonRaphsonStaggeredSolver` class there are two convergences, one local and one global. The local convergence is verified when the displacement or the phase-field is being calculated, and globally, when the step convergence is checked. The local convergence is reached when the error calculated by

$$\text{Error} = \frac{\left\|\delta\bar{X}\right\|}{\left\|\bar{X}\right\|} \tag{3.1}$$

is smaller then a defined tolerance, where $\delta\bar{X}$ and $\bar{X}$ can be the residual load and the forces vector, or the incremental displacements and the displacements vector, depending on the convergence type. After the phase-field is calculated, the residual forces and the incremental displacements vector are updated, and then Equation 3.1 is used to check the global convergence by testing if the obtained error is less then an specified tolerance. In short, local tolerance refers to the separate convergence of displacement and phase-field variables, while the global tolerance checks if the displacement convergence, after the phase-field convergence, were not unbalanced, it would be a local displacement tolerance after phase-field convergence.

Figure 3.5 shows the active global diagram developed by Leão (2021), Figure 3.6 shows the active diagram for the phase-field and Figure 3.7 shows a simplification of the JNI implementation idea. The indices i, j and k in this section refer, respectively, to the step, global iteration and local iteration. The global diagram shows all activities in the staggered solver, and within that diagram there would be other diagrams for the convergence of displacements and phase-field. Leão (2021) shows the displacement convergence diagram. Here, only the activity diagram for the phase-field convergence is shown, since only this one has been modified. And finally, a clearer diagram is shown about where the JNI is and the arguments that are passed between the binding. It is worth noting that in the global diagram the global tolerance is used, and in the phase-field and displacement diagrams the local tolerance is used.

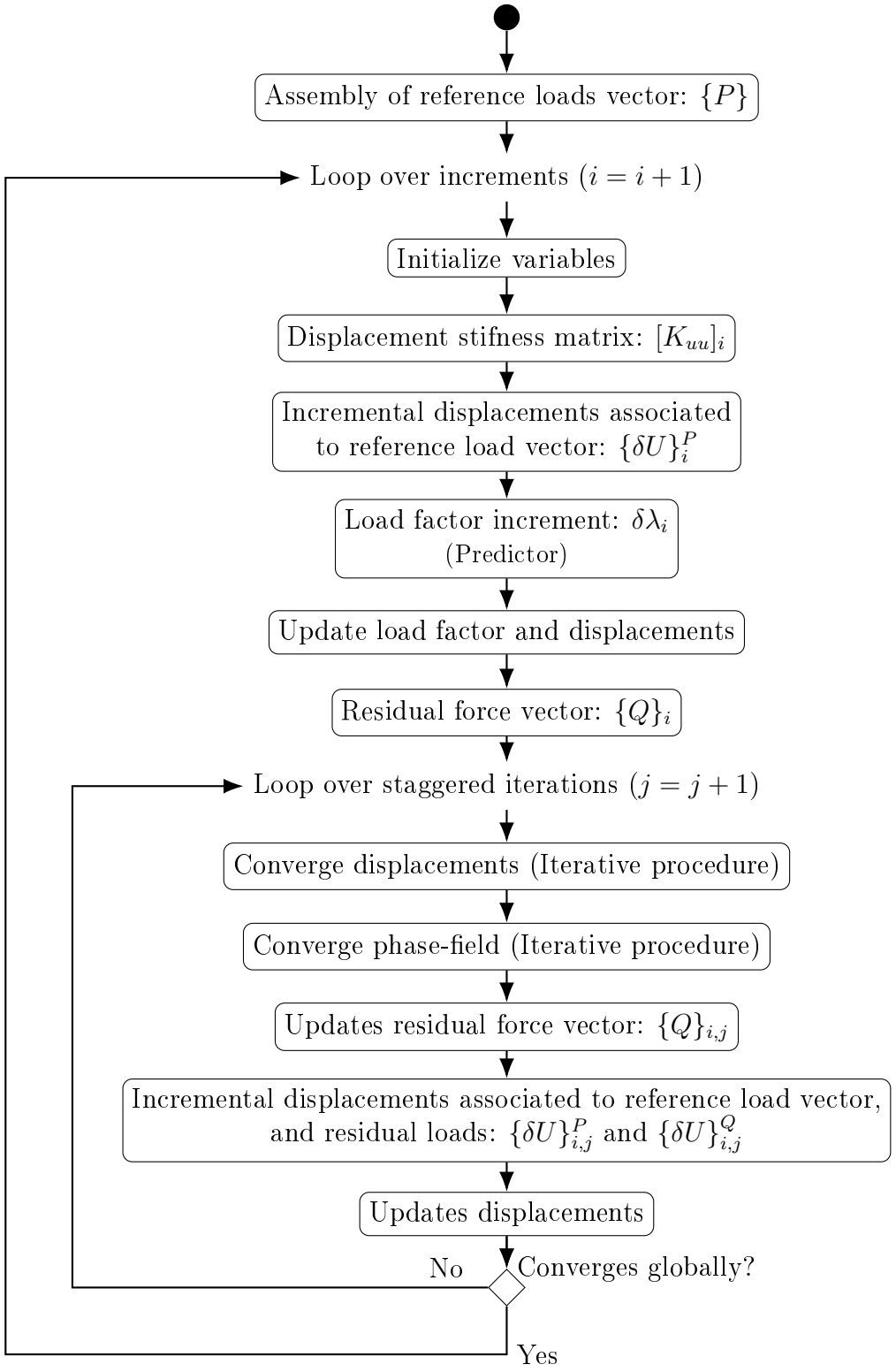


Figure 3.5: Global activity diagram. From Leão (2021)

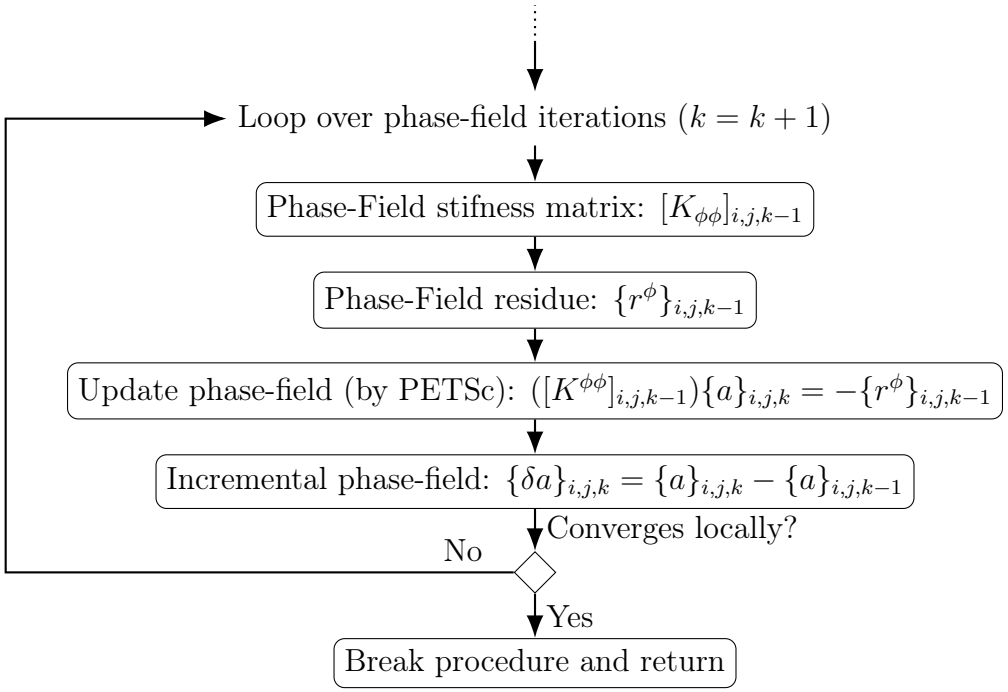


Figure 3.6: Activity diagram to converge phase-field with PETSc.

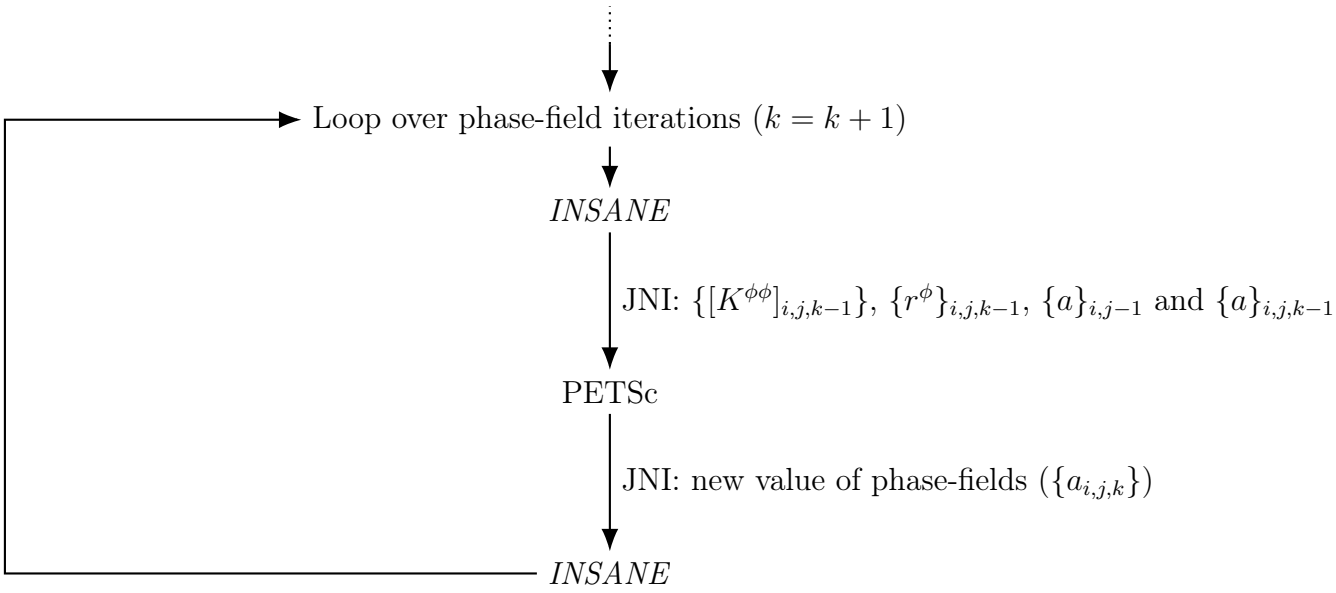


Figure 3.7: Activity diagram to JNI.

3.3 PETSc

According to the developers, the library has a set of routines and data structures that provide foundational tools for parallel applications on a large scale by the MPI commu-

nication standard. In addition to essential features with parallel matrices and vectors are also included various types of linear, non-linear and different equation solvers ordinary cials and differential algebraic equations. Also, it is possible to integrate it into languages such as FORTRAN, C, C++ and Python, as well as CUDA and OpenCL, to GPUs. Furthermore, there is an extensive documentation with examples, facilitating your understanding and use (Azevedo, 2019, Balay et al., 2020).

To use PETSc we need to make a JNI. The description of a basic JNI process as well as some details for the PETSc are described in Appendix A. The methods used in the library are highlighted here, but for more specific details you need to look at the library's manual. The AIJ sparse matrix from PETSc was used, and for that need to pass the *INSANE* sparse matrix variables, which are the values of the non-null elements, the pointers of these non-null elements per row and the number of non-null elements in a column to assemble the AIJ sparse matrix. In addition, it also passed the phase-field residual vector and the phase-field lower limit as already mentioned. The bound-constrained solver is used by calling the SNESVINEWTONRSLS method and setting the bounds by the SNESVISetVariableBounds method.

By using PETSc with these methods, we are separating nodes into sets into active and inactive, then finding a new phase-field value and using the convergence criterion presented in Section 3.2.2, but the line search part as shown in Figure 2.8 is not used. The reason for not using it and the implications of this discussed in Section 4.2.2 and Section 4.2.3. Furthermore, in section 3.4, the idea of the line search implemented by Bayão (2021) will be briefly presented, as the line search will be discussed in the work.

For a better performance of PETSc, it was build with with Intel® oneAPI Math Kernel Library (oneMKL) BLAS and LAPACK (see more details in Appendix A.2). The BLAS (Basic Linear Algebra Subprograms) standard is a proposal for the creation of high-performance libraries with basic kernel functions for operations on vectors and matrices, facilitating the incorporation of these exceptionally efficient routines scientific computing programs (Azevedo, 2019). LAPACK (Linear Algebra Package) is a library that provides implementations of routines for solving systems of equations and eigenvalues (Azevedo, 2019).

3.4 Line Search

In this work, the line search implemented in *INSANE* by Bayão (2021) will be used in some parts, so the idea will be presented here. The linear search consists of finding an optimal size for the incremental phase-field (δa - which would be the search direction) of an iteration found in the solver, trying to minimize the phase-field residue. A restriction operator presented in Equation 3.2 is used for this minimization:

$$F_\Theta(\bar{a}_{n+1})_I = \begin{cases} r_I^\phi, \text{ for } a_{I,n} < a_{I,n+1} < 1 \\ \min(r_I^\phi, 0), \text{ for } a_{I,n+1} = 1 \\ \max(r_I^\phi, 0), \text{ for } a_{I,n} = a_{I,n} \end{cases} \tag{3.2}$$

With this operator, a linear search is performed through Equation 3.3:

$$||F_\Theta(\pi[a_n^k + \lambda_l \delta a])|| \leq (1 - \tau\lambda_l)||F_\Theta(a_n^k)|| \quad \text{with} \quad \lambda_l = \lambda_{l-1}\lambda_0 \tag{3.3}$$

where τ is a reduction coefficient and λ_l, with $l = [1, 2, 3, ...]$, the multiplier defined for values above a previously established threshold Λ. In this work, the values $\lambda_1 = 1$, $\lambda_0 = 0.5$, $\tau = 10^{-4}$ and $\Lambda = 10^{-12}$ were used, taken from Benson and Munson (2006). The linear search finds an optimal value when the above condition is met, if no value of λ_l satisfies the relationship, the λ_l that promoted the greatest minimization during the routine is used. When using this line search, a new convergence criterion is used that makes use of the minimization of the modulus of the vector generated by the restriction operator. The summary of this whole idea is in Figure 3.8.

Data: a^0, $k = 0$
Result: A solution of (δa)
while $||\bar{F_\Theta}(\bar{a}_n^k + 1)|| > TOL$ **do**
  Compute active($\mathscr{A}$) and inactive($\mathscr{I}$) sets
  Solve the reduce Newton step: δa
  $l = 1$
  Boolean = false
  while $\lambda_l > \Lambda$ **do**
    if $||F_\Theta(\pi[a_n^{k,l} + \lambda_l \delta a])|| \leq (1 - \tau\lambda_l)||F_\Theta(a_n^k)||$ **then**
      $\bar{a}^{k+1} = \bar{a}^k + \lambda \delta \bar{a}$
      Boolean = true
      Break
    end
    if $||F_\Theta(\pi[a_n^{k,l} + \lambda_l \delta a])|| \leq ||F_\Theta(a_n^k)||$ **then**
      $\lambda_{min} = \lambda_l$
    end
    $l = l + 1$
    $\lambda_l = \lambda_{l-1}\lambda_0$
  end
  if *Boolean == false* **then**
    $\bar{a}^{k+1} = \bar{a}^k + \lambda_{min} \delta \bar{a}$
  end
  k = k +1
end

Figure 3.8: Algorithm for line search from Bayão (2021)

Chapter 4

Numerical study of the bound-constrained solver by PETSc

The objective of this chapter is to validate and analyze the bound-constrained solver by PETSc. In the first section, the importance of the bound-constrained solver will be shown. Later, the results of bound-constrained solver by PETSc will be analyzed with a historical and bound-constrained solver in *INSANE*. Finally, an analysis of time performance will be made between the existing solvers. In all examples displacement control was used, the local tolerance used was of $1 \cdot 10^{-4}$ and the global tolerance was of $1 \cdot 10^{-4}$. All cases are in a plane stress state.

4.1 Utility of the solver

As mentioned before, the objective of using the bound-constrained solver is to be able to use other crack geometry functions well and consequently other energy degradation functions. Therefore, in this section we will show the results that can now be obtained with the bound-constrained solver that were not possible with the historical solver.

In this section, the L-Shaped Panel shown in Figure 4.1 with measurements in millimeters will be considered throughout the discussion. A vertical point load F is applied upward at a distance of 30 mm to the right edge of the horizontal leg, with the corresponding vertical displacement at the same place recorded. This example is taken from Wu (2017). The problem was modeled with a mesh of CST (Constant strain triangle) elements. The mesh used is illustrated in Figure 4.2, where the refined region has a size of 4.25 mm, while the unrefined region has a size of 25 mm. The values used for the numerical model were: Young´s Modulus $E_0 = 25850$ N/mm^2, Poisson´s ratio $\nu = 0.18$, strength failure $f_t = 2.7$ MPa, critical energy release rate $G_c = 0.065$N/mm and length scale $l_0 = 8.5$ mm. Increments of $2 \cdot 10^{-3}$ were considered in all cases.

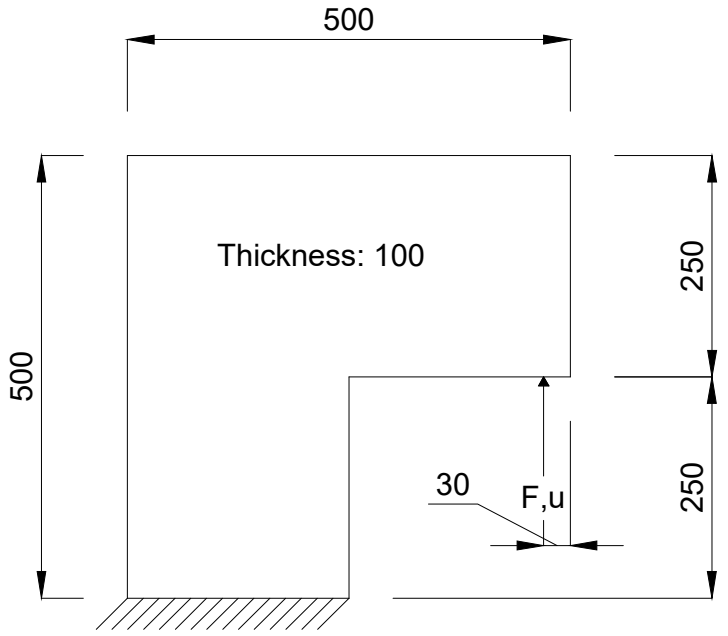
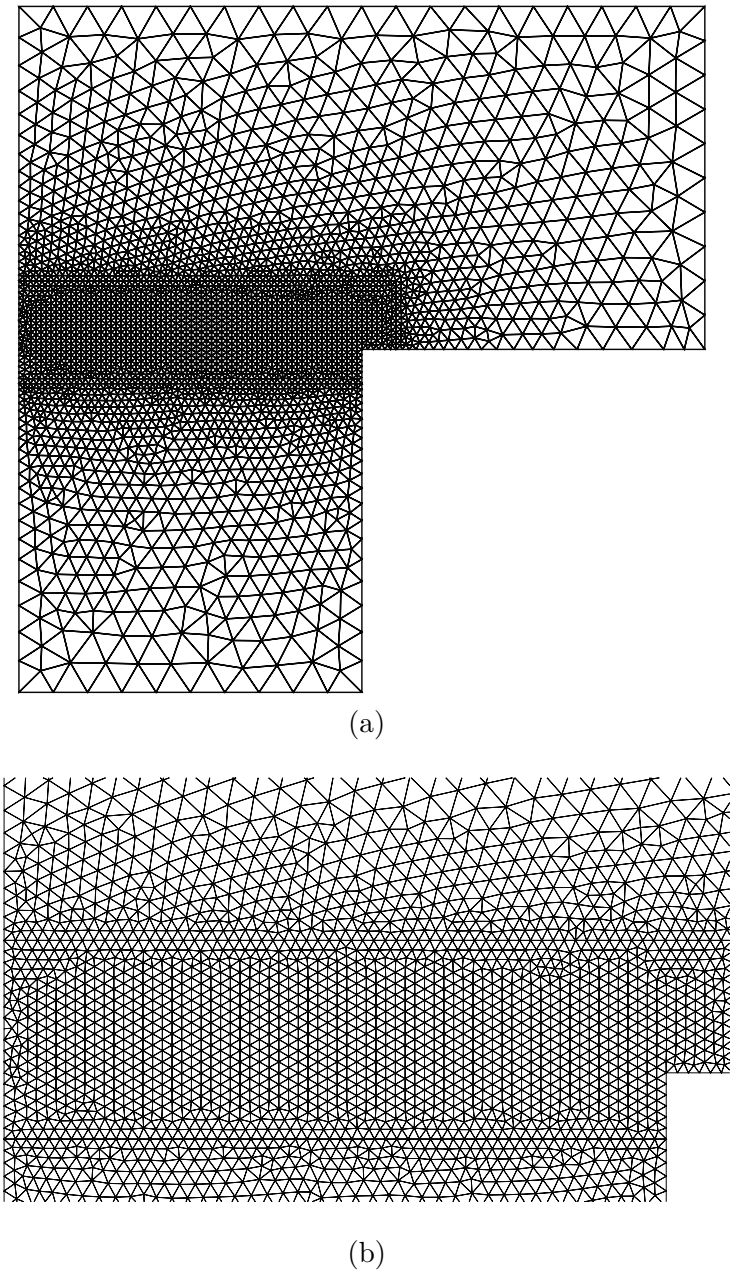


Figure 4.1: Problem setting for L-shaped panel with measurements in millimeters.



(a)

(b)

Figure 4.2: T3 mesh for L-shaped panel. (a) Whole Structure (b) Zoom in Refined Region.

4.1.1 Possibility of other energy degradation functions

In all the models processed in this section, the Wu (2018) constitutive model and the crack geometry function given by Equation 2.29 with $\xi = 2$ were used, only changing the softening laws. This section demonstrates the use of the energy degradation function proposed by Wu (2017) in Equation 2.30 with different softening laws (Section 2.4.1) with the bound-constrained solver by PETSc. The results for the load versus vertical displacement of the point where the load is applied for each softening law are shown in Figure 4.3. The final crack path for each softening law are shown in Figure 4.4.

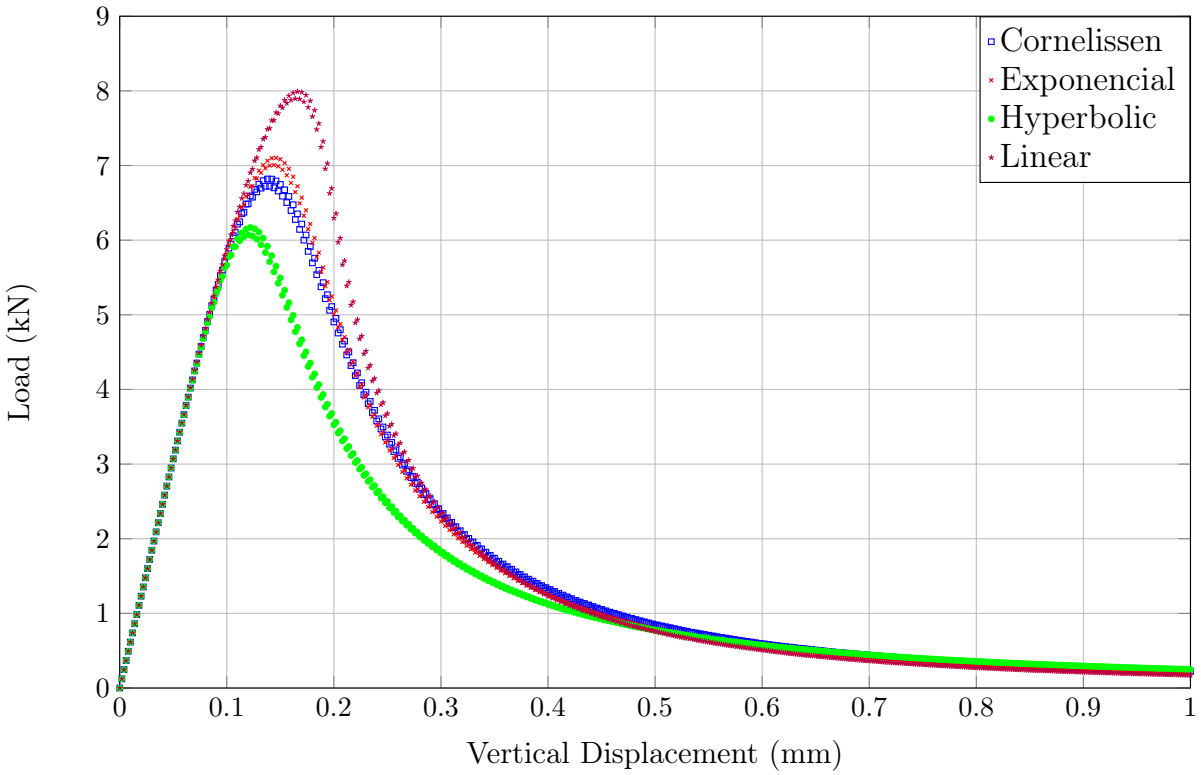


Figure 4.3: Curves of load versus displacement of the loading application node for different softening laws.

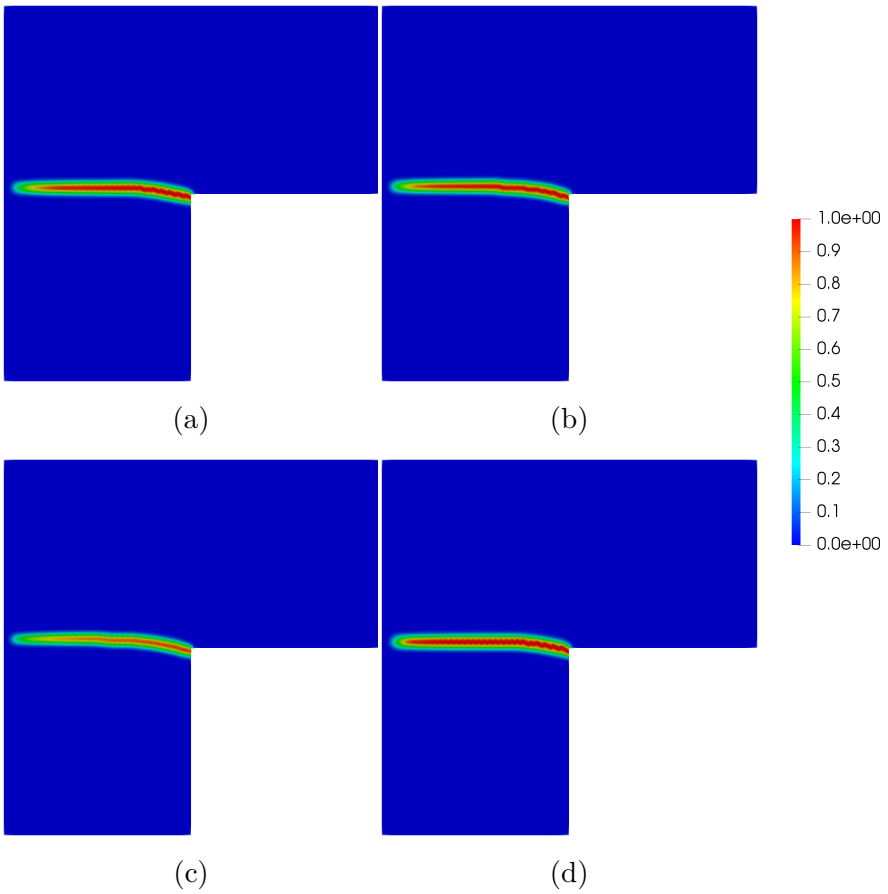


Figure 4.4: Phase-field contour plots for: (a) Cornelissen's Softening Law (b) Exponential Softening Law (c) Hyperbolic Softening Law (d) Linear Softening Law.

The first detail to highlight is that it would not be possible to obtain these results with the historical solver. It is noticed that Cornelissen´s law and exponential law have very similar behavior. Hyperbolic law presents a smaller crack with a lower peak load, showing a more ductile behavior, while the linear law presents a higher peak load with a greater crack, showing a more brittle behavior.

4.1.2 Possibility of other crack geometry functions

This section demonstrates the use of the crack geometric function proposed by Wu (2017) in Equation 2.29 with different values for ξ with the bound-constrained solver by PETSc. In all models processed in this section, the Wu (2018) constitutive model and the energy degradation function given by Equation 2.30 with Cornelissens's law for normal concrete were used. The results for the load versus vertical displacement of the point where the load is applied for each ξ are shown in Figure 4.5. The final crack path for each ξ are shown in Figure 4.6.

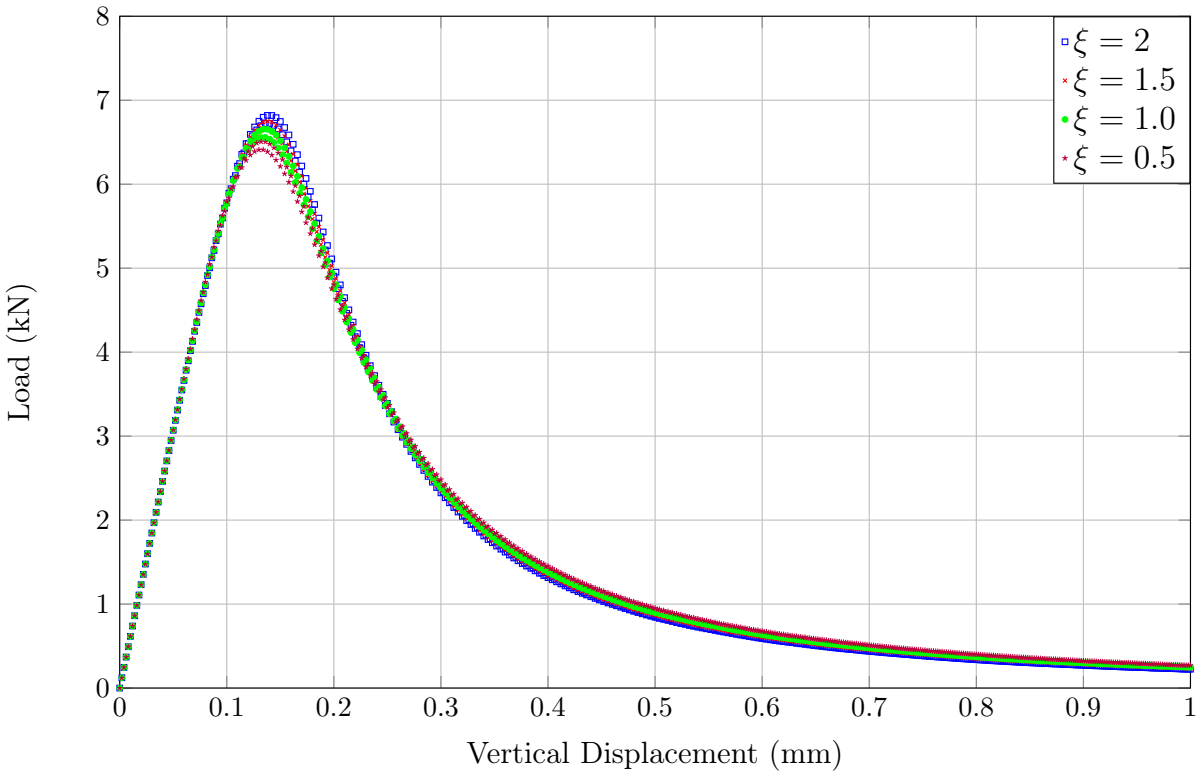


Figure 4.5: Curves of load versus displacement of the loading application node for different values of ξ.

Again, it should be noted that it would not be possible to obtain these results with the historical solver. There is not much difference in the results of load versus displacements, however it is noted that as ξ increases the peak curve decreases and the material becomes more ductile. Furthermore, although it is difficult to notice in Figure 4.6, as the ξ increases, the crack width increases, as seen in the Section 2.3.

The value of $\xi = 0$ was not used because for Equation 2.30 it can be seen that if you place ξ close to zero the energy degradation function will have a value equal to 1 and there will be no cracking in the model. In this case, $\xi = 0$ cannot be used because the value of c_2 (see Equation 2.32b) would be undetermined.

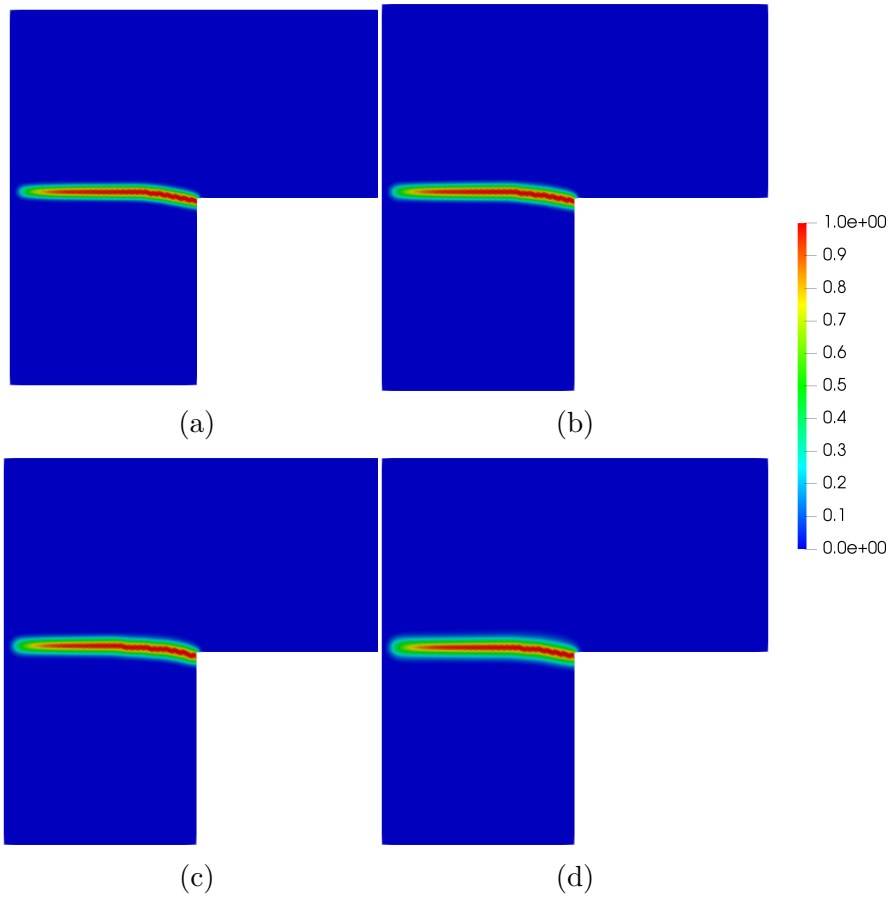


Figure 4.6: Phase-field contour plots for: (a) $\xi = 2.0$ (b) $\xi = 1.5$ (c) $\xi = 1.0$ (d) $\xi = 0.5$.

4.2 Analysis of results among solvers

In this section, some analyzes will be made in relation to the results of load versus displacements between the existing solvers with the bound-constrained solver by PETSc.

4.2.1 Comparison with Historical Solver

The objective of this section is to compare the results of the bound-constrained solver by PETSc with the historical solver and with other numerical results available in the literature. The same L-Shaped Panel problem shown in Section 4.1 was used with the same mesh, measurements and same material parameters changing only the l_0, because now it is necessary to make a material calibration. The calibration is necessary to have a better numerical result in relation to the experimental results. Clarifying that previously in section 4.1 the objective was to show the new possibilities with the bound-constrained solver, and now the objective is to compare its result with the experimental result and analyze its performance with respect to other numerical models. The calibration of the

material of the phase-field model was made with the Carreira-Ingraffea material and with a generalized fixed-smeared crack model implemented in *INSANE* by Penna (2011), who obtained a good result in relation to the experimental one.

To calibrate the material for analysis, uniaxial tensile tests were performed with unit load on the right face on a quadrilateral element with a side of 1 mm, as illustrated in Figure 4.7. This allowed visualizing the behavior of the material under tension and choosing for the phase-field models a value of l_0 that would guarantee them a behavior similar to that of elastic degradation. The values for the mechanical properties used in fixed-smeared crack model were: uniaxial compressive strength $f_c = 31$ N/mm^2, characteristic length of the material L_c equals to 28 mm and shear retention factor β_r equals to zero. It is worth mentioning that the parameter for elastic limit deformation in tension (ε_t) for the fixed-smeared crack model is obtained by making f_t/E_0. The parameter for elastic limit deformation in compression (ε_c) makes no difference in the value in the calibration between the models, this was verified by numerical tests and it also makes sense since for the phase-field model only the parameters related to tension are relevant, having linear compression. This applies to the other calibrations made throughout the text. The model was calibrated with the Carreira-Ingraffea material, in order to present the same response material. The phase-field model with bound-constrained solver by PETSc with Wu (2018) constitutive model, $\xi = 2$ and energy degradation function given by Equation 2.30 with Cornelissens's law for normal concrete was calibrated with the fixed-smeared crack in *INSANE*. Figure 4.8 shows the calibration for $l_0 = 8.5$ mm, that will be used.

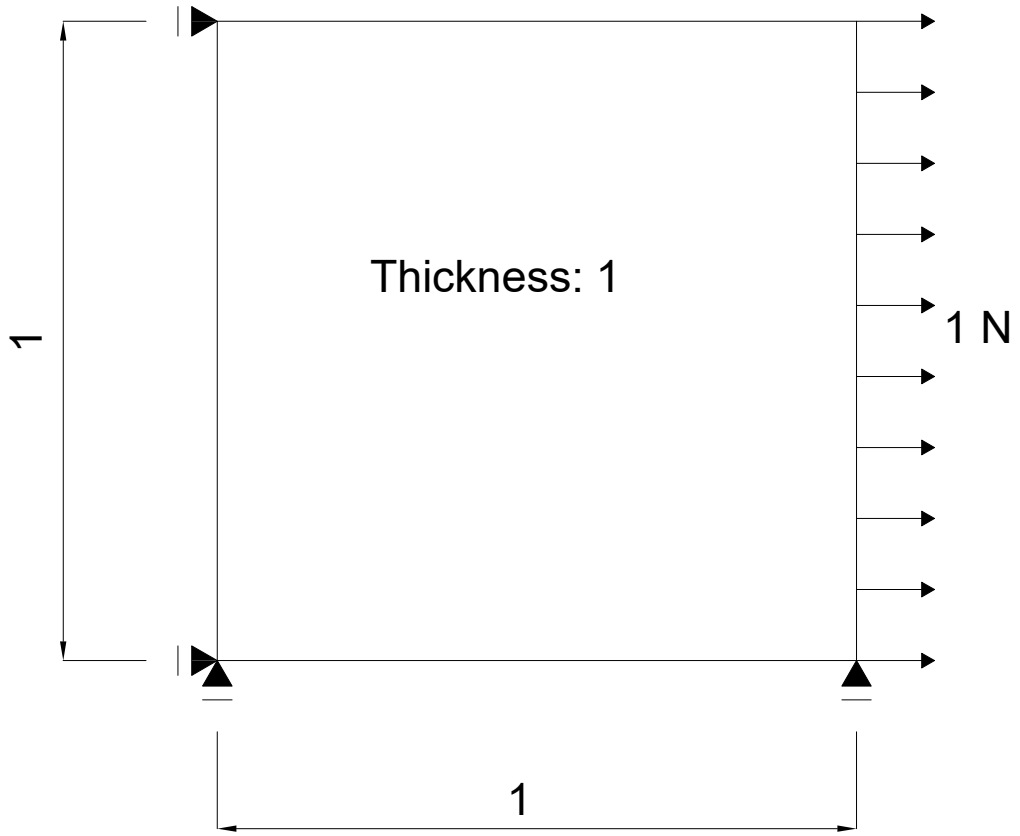


Figure 4.7: Problem setting for calibration with measurements in millimeters.

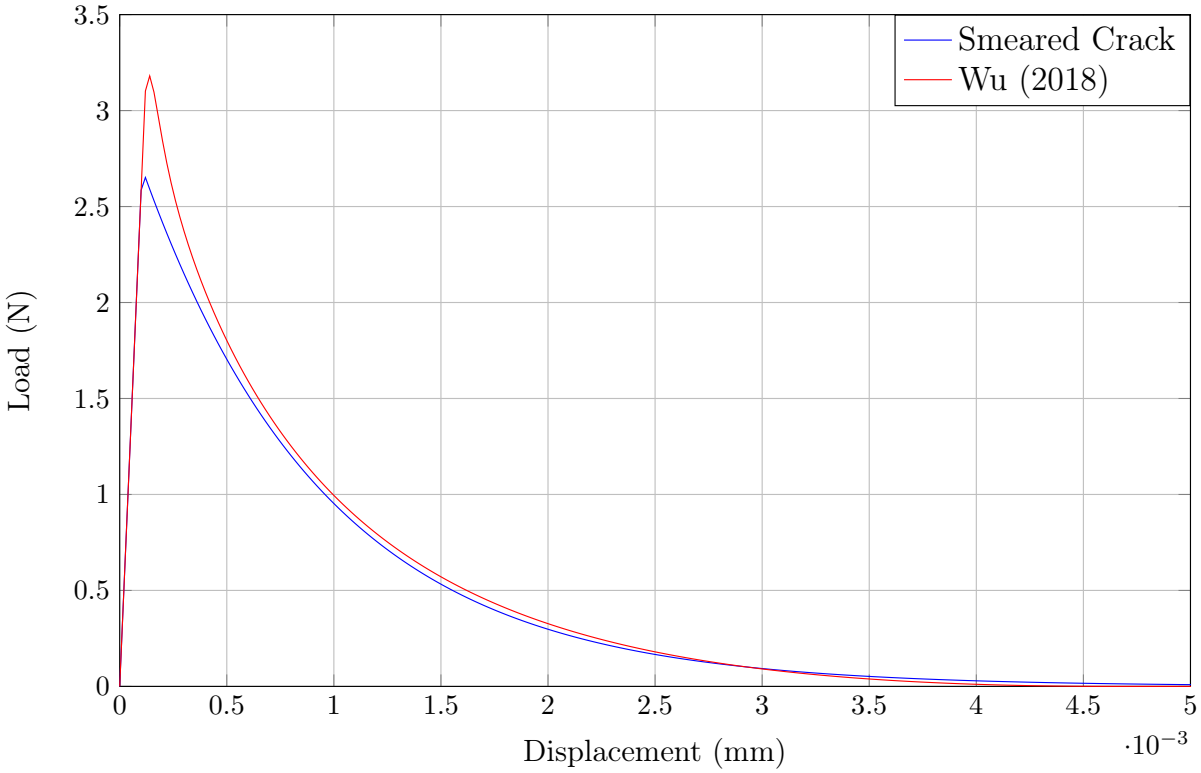


Figure 4.8: L-shaped panel calibration.

The reason why the calibration does not have the same peak curve (f_t) in both models in the uniaxial tensile model presented will be discussed in Section 4.2.3.

As we want to show the limitation of the historical solver, we present the result with this solver using the crack geometry function given by $\alpha = \phi^2$ (Bourdin et al., 2000), $\xi = 0$) and energy degradation function given by $g = (1 - \phi)^2$ (Bourdin et al., 2000). Remembering that for the historical solver it is necessary to use this crack geometry function, therefore not allowing to use the energy degradation function given by Equation 2.30. Finally, the bound-constrained solver by PETSc was used to show the improvement in the results. For this solver, we used the with crack geometry function given by Equation 2.29 with $\xi = 2$ and energy degradation function given by Equation 2.30 with Cornelissens's law for normal concrete.

In both solvers, the Wu (2018) constitutive model was used, to show that it is not the constitutive model that limited but crack geometry function and energy degradation function, which can be observed between them by looking at Figures 4.9 and 4.10. The results for both solvers are shown in Figure 4.11, the experimental results being given by Winkler (2001).

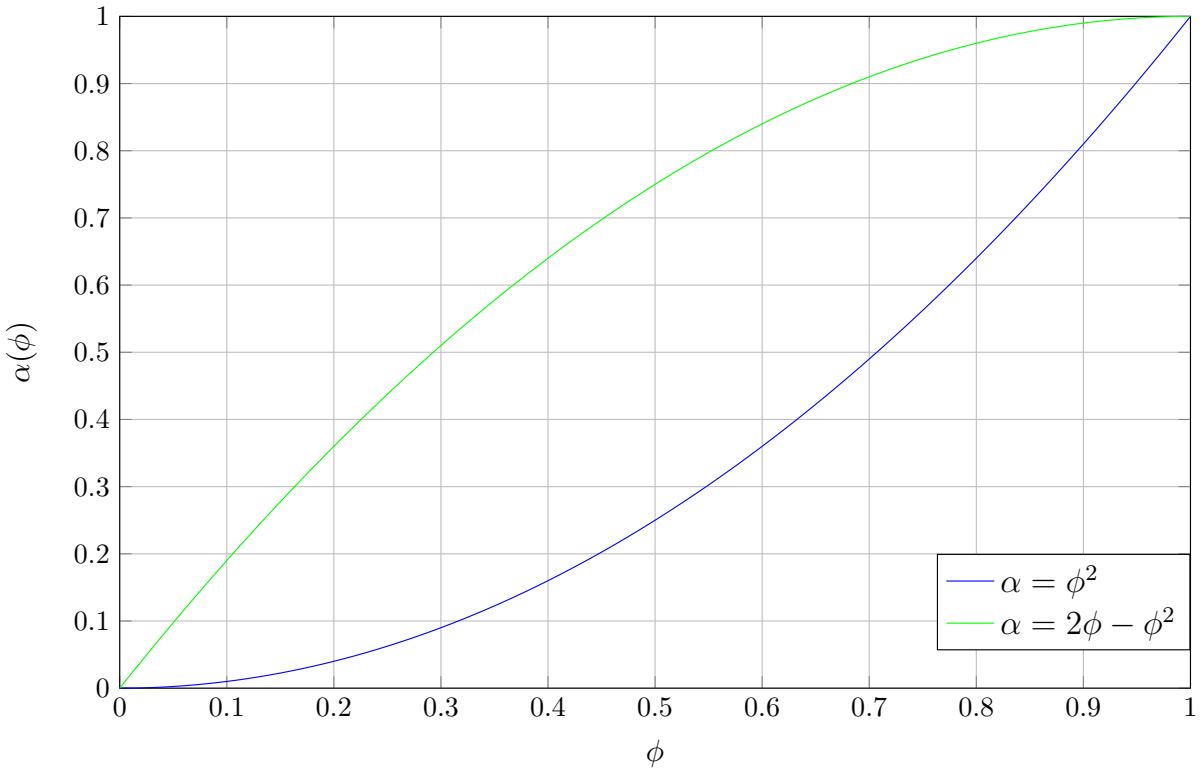


Figure 4.9: Comparation between $\alpha(\phi)$ functions.

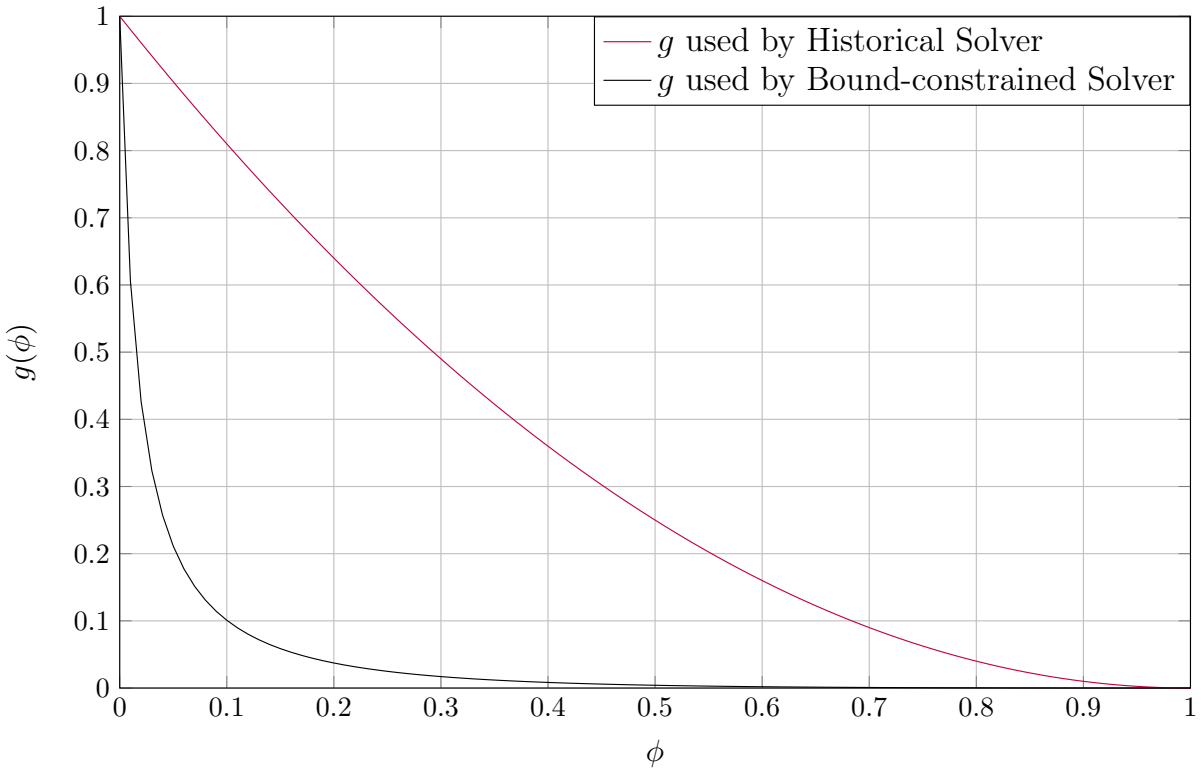


Figure 4.10: Comparation between $g(\phi)$ functions.

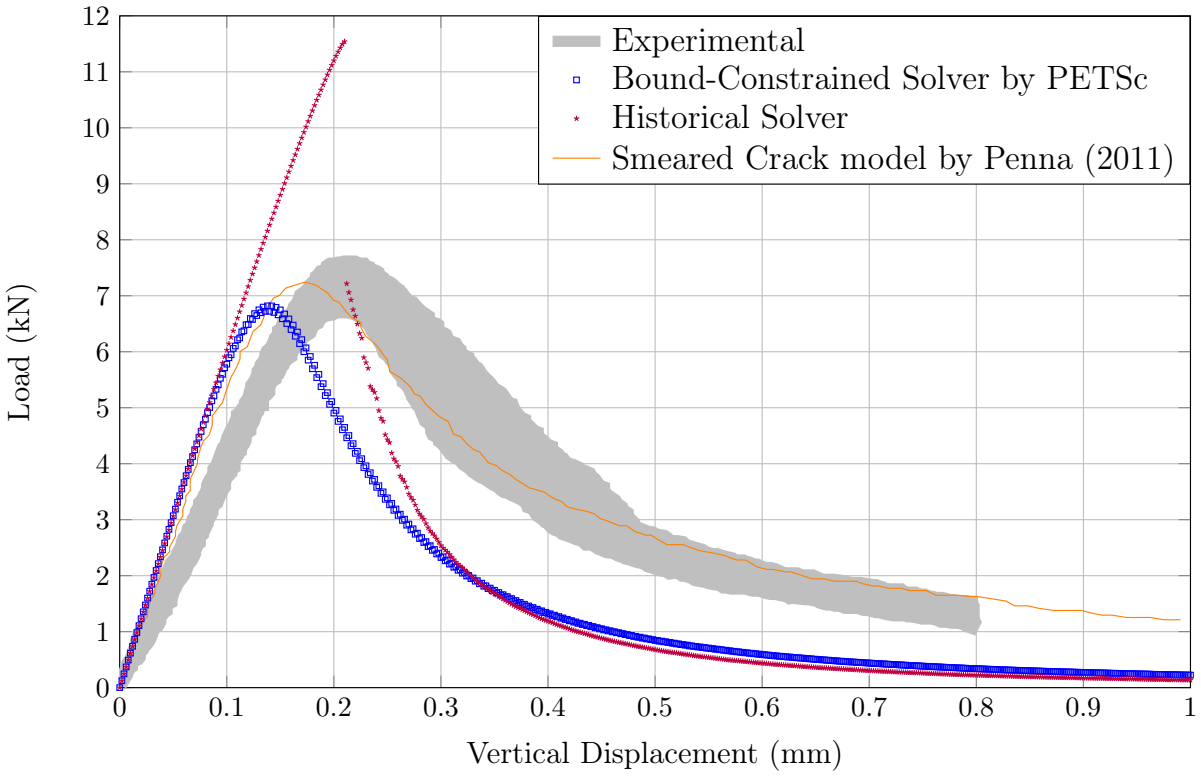


Figure 4.11: Curves of load versus displacement of the loading application node.

It can be seen then that with the bound-constrained solver being able to use other crack geometry function and energy degradation function that are associated with the

failure strength parameter (f_t), the phase-field result improves a lot when compared to the experimental one and the result obtained for smeared crack model for Carreira-Ingraffea material by Penna (2011). It is noticed that there is an instability in the points by the bound-constrained solver. This instability will be discussed in the Section 4.2.2.

4.2.2 Validation with the Bound-Constrained Solver developed in *INSANE* by Bayão (2021)

To validate the solver by PETSc, the graphs responses will be compared with the solver in *INSANE* developed by Bayao et al. (2021). One detail is that, as mentioned before, the solver by PETSc presents a numerical instability, while this does not happen in the bound-constrained solver in *INSANE*, because it uses a line search algorithm and convergence criterion by minimizing the modulus of the vector generated by the constraint operator. If we remove this line search algorithm in the source code of the program and put the same convergence table already presented here in Section 3.2.2, we can see that the answer is the same as the solver by PETSc (see Figure 4.12), soon showing that the solver by PETSc was implemented correctly but with the exception of numerical instability. To make it clear, the modifications made to the code presented in bound-constrained solver in *INSANE* do not compromise the answer, since the separations are still made in an active and inactive set, as well as finding the new phase-field value, restricted by the phase-field limits through the definition of the sets.

The same L-Shaped Panel of Section 4.2.1 was used, with the same measurements, mesh and material characteristics, only changing the solver. In all models processed in this section, the Wu (2018) constitutive model, the crack geometry function given by Equation 2.29 with $\xi = 2$ and energy degradation function given by Equation 2.30 with Cornelissens's law for normal concrete were used. The results for the load versus vertical displacement of the point where the load is applied for each solver are shown in Figure 4.12. The final crack path obtained by solver by PETSc and solver by Bayão (2021) are shown in Figure 4.13.

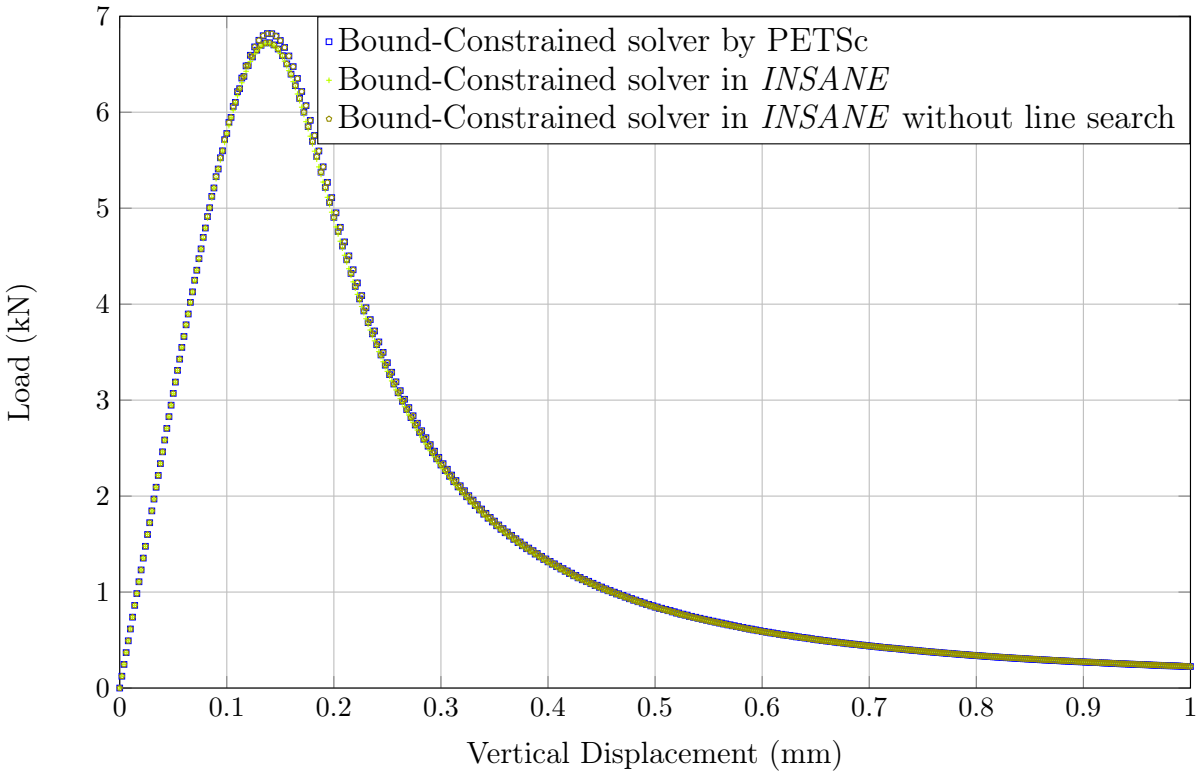
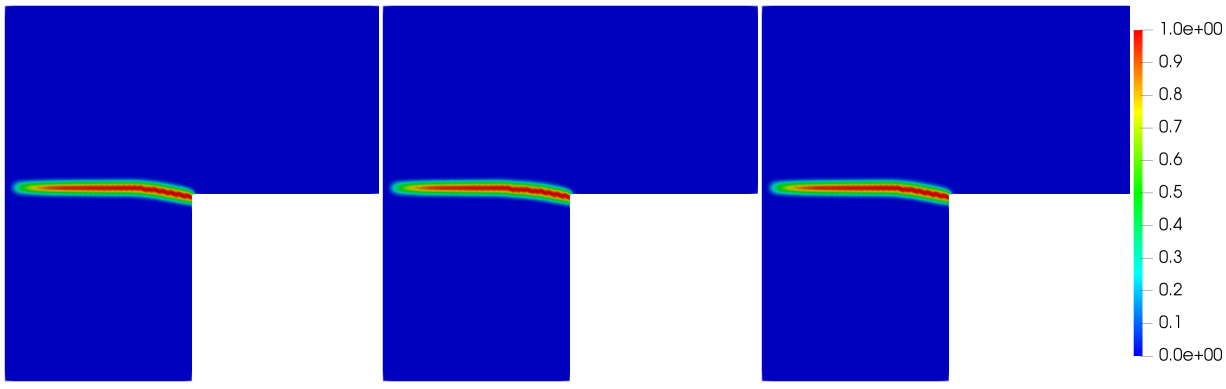


Figure 4.12: Curves of load versus displacement of the loading application node for diferrent bound-constrained solvers.



(a) (b) (c)

Figure 4.13: Phase-field contour plots for: (a) Solver by PETSC (b) Solver by Bayão (2021) (c) Solver by Bayão (2021) without line search.

It can be seen then that solver by PETSc can serve the purpose of finding the peak of the load vs displacement curve, as well as the path of this curve, even with numerical instability, so there is no problem in using the solver by PETSc.

4.2.3 Importance of a line search

The purpose of this section is to show the importance of a line search algorithm in code. For this, the same uniaxial tensile model shown in Section 4.2.1 for a quadrilateral element with a side of 1 mm will be used. Figure 4.14 shows the results, where the responses for the bound-constrained solver in *INSANE* as the bound-constrained solver by PETSc with the line search implemented in Bayao et al. (2021) are the same. That is, the green line represents the response of both solvers with the linear search. Furthermore, the answers for both solvers without linear search are also the same, that is, the red line represents the answer for both solvers without linear search. So this again this validates the solver by PETSc.

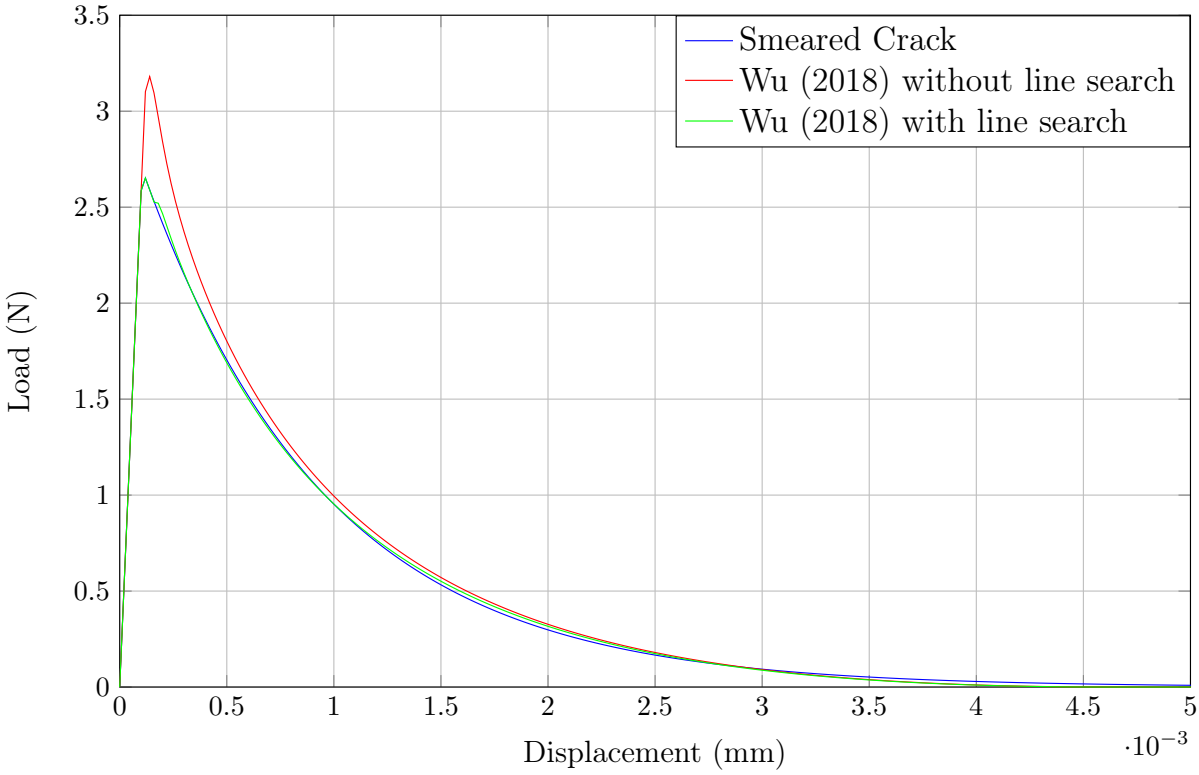


Figure 4.14: Solver result comparison with or without line search for L-shaped panel calibration.

From Figure 4.14, it can be seen that without a line search, the peak of the curve that would be described by the value of f_t is different, thus showing the need for a line search. Attempts to combine the PETSc solver with the line search algorithm implemented in Bayao et al. (2021) were made, but in some cases the number of iterations burst, never converging. Therefore, it is necessary to delve deeper into the studies involving this code to verify if it was only a problem located at the time of the search. It is worth mentioning that with some previous investigations it is noticed that it manages to circumvent this problem by reducing the step size for displacement control. Despite this, this work at the time it was developed had not completed this analysis, so in the rest of the work line

search was not used.

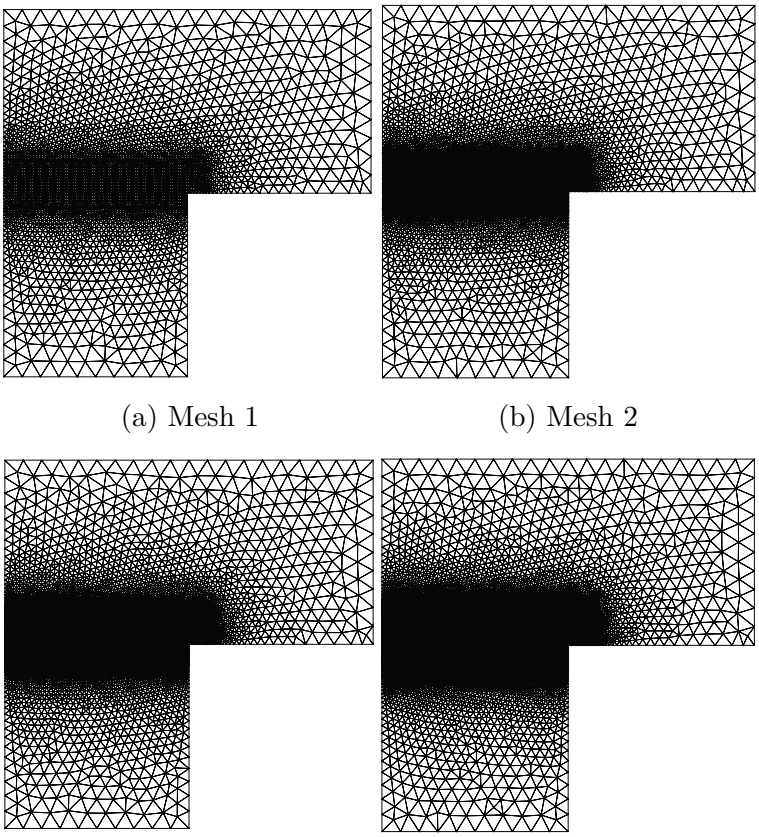
4.3 Performance Comparison

The purpose of this section is to compare time performance between solvers. Three numerical models were made for a better validation of the performance. The first model will be an L-Shaped Panel, the second model will be an Three-Point Bending Beam and the third model will be an Wedge Split Test. The models were processed on a machine with the following characteristics: Intel® Core™ i5-5200U CPU with 2 cores and 4 threads, with a frequency of 2.20 GHz, and 8GB of RAM DDR3L with 1600 MHz.

The comparison with the bound-constrained solver in *INSANE* was done by removing the line search strategy and using the same convergence of the bound-constrained solver by PETSc without line search, as mentioned in Section 4.2.2. This was done to have a fairer comparison, since when using the line search the solver time in *INSANE* increases considerably.

4.3.1 L-Shaped Panel

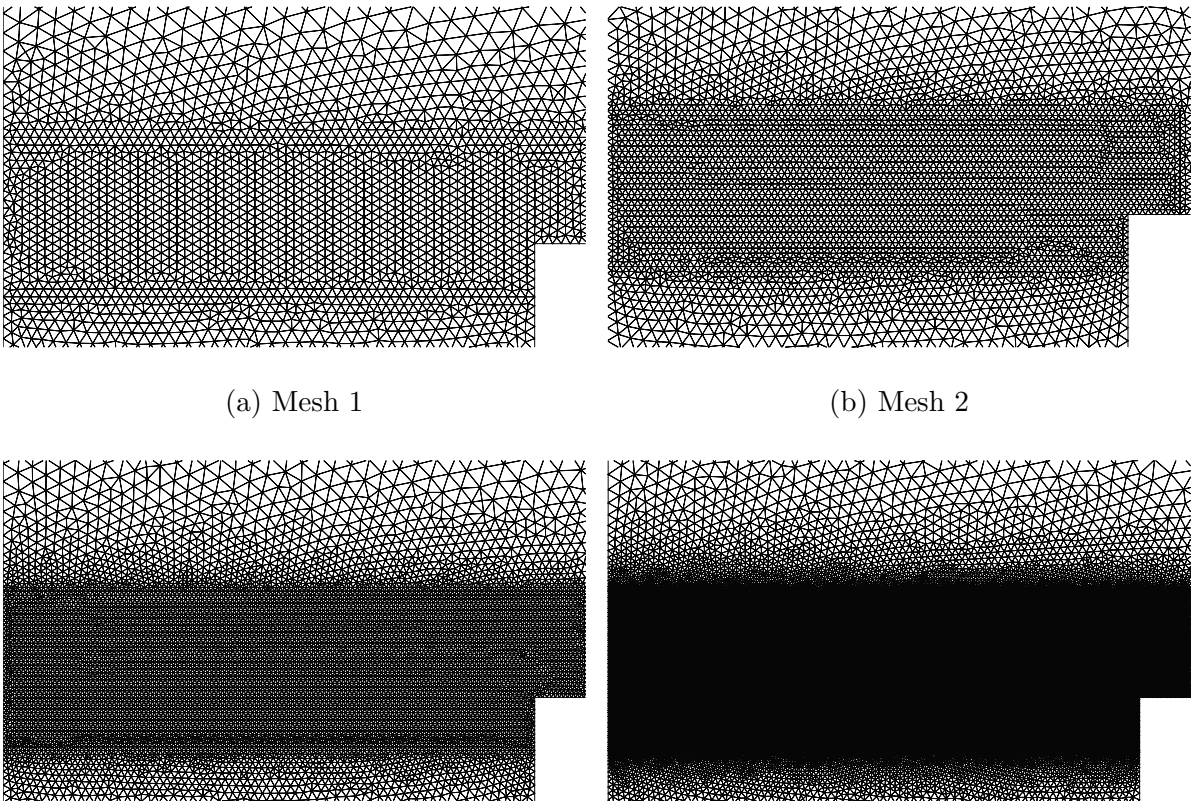
The problem used in this section will be the same as shown in Figure 4.1. Four different meshes CST elements were built for the analysis, which are shown in Figure 4.15, and a zoom in the refined region is shown in Figure 4.16. In the unrefined region, all meshes have a size of 25 mm, while in the refined region each mesh has a value as shown in Table 4.1. The size of 4.25 mm in the refined region of the mesh was adopted for the initial mesh since the l_0 adopted is 8.5 mm, as it will be shown, due to Miehe et al. (2010a) says that for elements inside the crack, the mesh size should be at most half the length scale ($h \leq l_0/2$).



(a) Mesh 1 (b) Mesh 2

(c) Mesh 3 (d) Mesh 4

Figure 4.15: Different meshes for L-Shaped Panel.



(a) Mesh 1 (b) Mesh 2

(c) Mesh 3 (d) Mesh 4

Figure 4.16: Zoom into the refined region of each mesh.

Table 4.1: Table with description of mesh refinement for the L-Shaped Panel

Mesh	Mesh size in the fine region (mm)	Number of elements
Mesh 1	4.25	5955
Mesh 2	3	9936
Mesh 3	2	18376
Mesh 4	1	59281

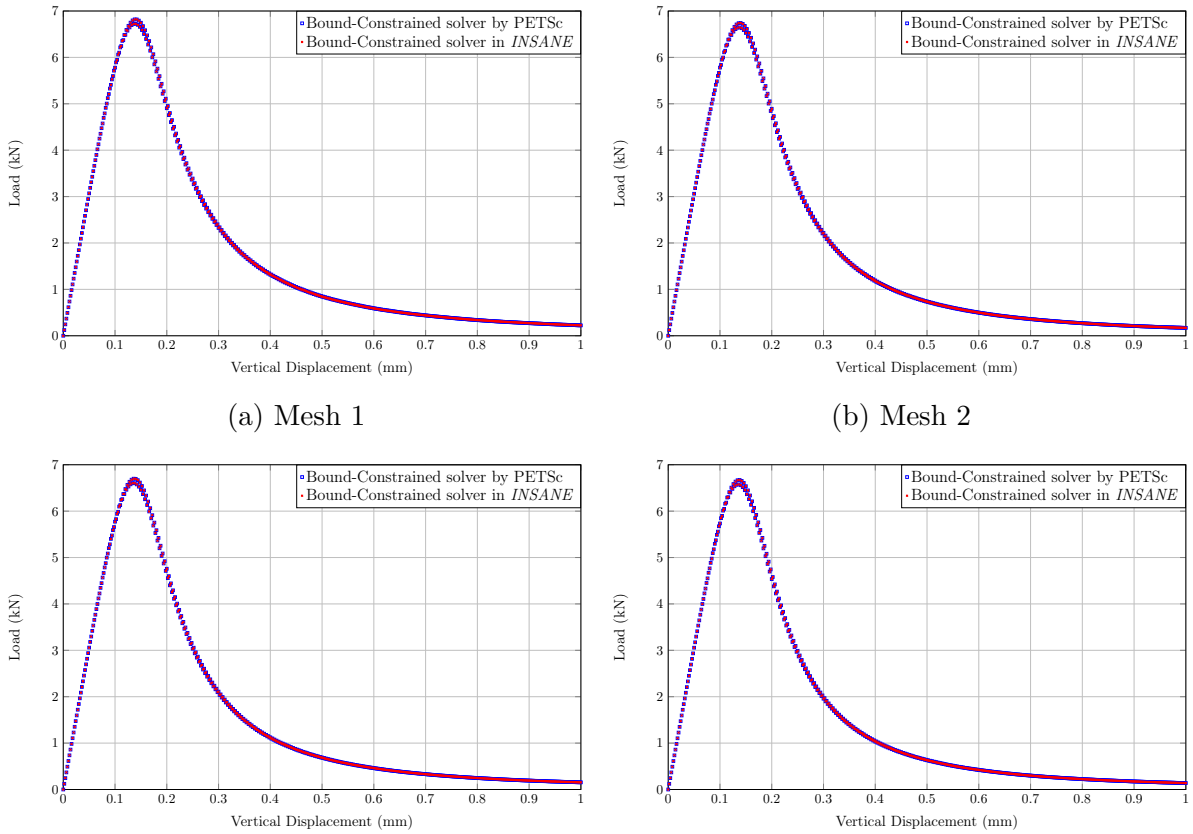
The following data for the L-Shaped Panel are valid for the following subsections: $E_0 = 25850$ N/mm^2, $\nu = 0.18$, $G_c = 0.065$ N/mm, $l_0 = 8.5$ mm. Increments of $2 \cdot 10^{-3}$ were considered in all cases for this section.

4.3.1.1 Comparison with Bound-Constrained Solver developed in *INSANE* by Bayao et al. (2021)

For the comparisson between bound-constrained solver by PETSc and bound-constrained solver developed in *INSANE*, the following data were used: $f_t = 2.7$ MPa, crack geometry function given by Equation 2.29 with $\xi = 2$, energy degradation function given by

Equation 2.30 with Cornelissens's law for normal concrete ($\eta_1 = 3$ and $\eta_2 = 6.93$) and Wu (2018) constitutive model.

The results for the load versus vertical displacement of the point where the load is applied for each mesh are shown in Figure 4.17. The final crack path to mesh 4 for each solver is shown in Figure 4.18.



(a) Mesh 1

(b) Mesh 2

(c) Mesh 3

(d) Mesh 4

Figure 4.17: Curves of load versus displacement of the loading application node for L-Shaped Panel with different meshes.

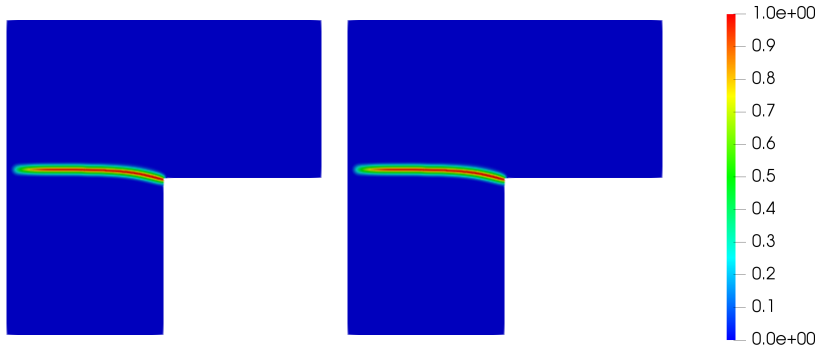


Figure 4.18: Phase-field contour plots for L-Shaped Panel with Mesh 4: (a) Bound-Constrained solver by PETSc (b) Bound-Constrained solver in *INSANE*.

Table 4.2 shows the times for each solver and for each mesh. It can be seen that when refining the mesh, the bound-constrained solver by PETSc gains more time advantage compared to the solver in *INSANE*. A detail to be highlighted is that the stiffness matrix needs to be reassembled in the PETSc library, as already mentioned in Section 3.3, so even with this reassembly time, PETSc is still faster.

Figure 4.19 compare all types of iterations for each solver for mesh 4. This Figure presents some acronyms to represent each type of iteration: GI stands for Global Iteration, thus presenting the Global Iterations in a step; PFI stands for Phase-Field Iterations representing the number of phase-field iterations in step, that is, it represents the sum of the number of local iterations of phase-field of the global iterations in a step; DI stands for Dislacement Iterations which represents the number of iterations of displacements in a step, that is, it represents the sum of the number of local iterations of displacements of the global iterations in a step. To better understand the idea, analyze the global activity diagram illustrated in Figure 3.5, showing that there are iterative phase-field and displacement processes in a global iteration. So each step can have several global iterations and each global iteration can have several phase-field and displacement iterations.

Analyzing Figure 4.19, it can be seen that in terms of iterations there is no advantage of one solver over the other. This indicates that the different processing time is not due to an increase in iterations in the solution process, but only to differences between the two algorithms.

Table 4.2: Table with processing times for each solver

Mesh	Time for solver by PETSc	Time for solver in *INSANE*	Fastest Percentage (%)
Mesh 1	45 min 47 s	46 min 9 s	0.79
Mesh 2	1 h 20 min 21 s	1 h 26 min 2 s	6.61
Mesh 3	2 h 43 min 45 s	3 h 3 min 16 s	10.65
Mesh 4	11 h 8 min 5 s	16 h 36 min 57 s	49.03

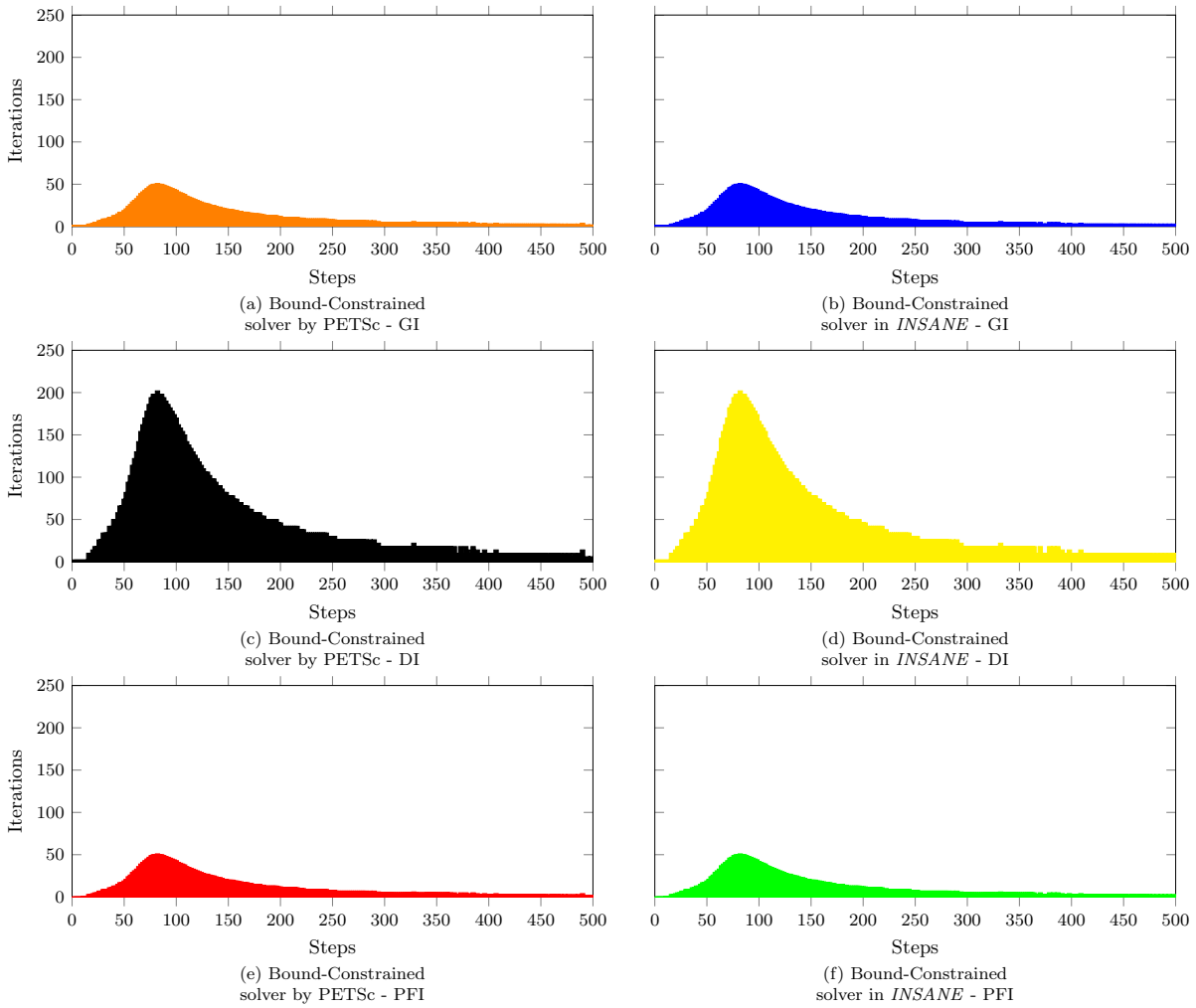


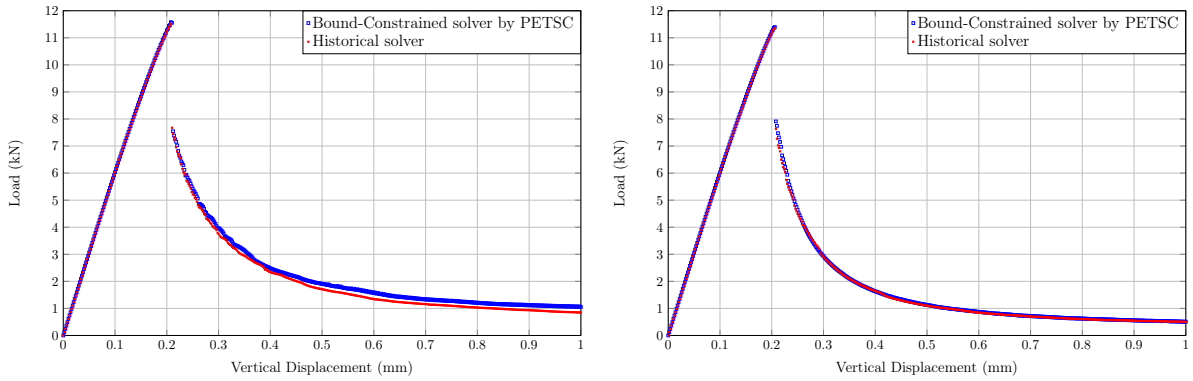
Figure 4.19: Number of Iterations for each solver and for iteration type for the L-Shaped Panel with Mesh 4.

4.3.1.2 Comparison with Historical Solver

For the comparisson between bound-constrained solver by PETSc and historical solver, the following data were used: $\alpha = \phi^2$, $g = (1-\phi)^2$ and Miehe et al. (2010b) constitutive model.

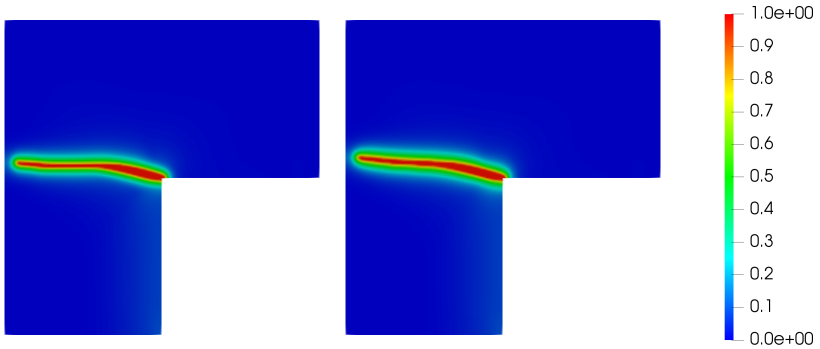
The results for the load versus vertical displacement of the point where the load is applied for each mesh are shown in Figure 4.20. The final crack path to mesh 2 for each

solver is shown in Figure 4.21.



(a) Mesh 1 (b) Mesh 2

Figure 4.20: Curves of load versus displacement of the loading application node for L-Shaped Panel with different meshes.



(a) (b)

Figure 4.21: Phase-field contour plots for L-Shaped Panel with Mesh 2: (a) Bound-Constrained solver by PETSc (b) Historical solver.

Mesh 3 and 4 were not used for this case, because for the bound-constrained solver did not converge, overflowing the number of iterations. This is not due to a limitation in the solver but to the control method used. It can be seen in Figure 4.20 that the displacement retreats more to the left after the load drop for the more refined mesh, decreasing the displacement there, so probably for a refined mesh this control method was not able to identify the point.

Table 4.2 shows the times for each solver for each mesh. The solver by PETSc also shows a time performance improvement over the historical solver as the mesh is refined, but a very small improvement. It is worth noting that the Miehe et al. (2010b) constitutive model takes much longer than the Wu (2018) constitutive model, due to how each energy separation is performed. Figure 4.22 compare all types of iterations for each solver. In

this case the comparison is not so significant since they are different type of solvers. At one point, a solver presents more or less one type of iteration than the other, as will be seen in the other numerical models in the next sections.

Table 4.3: Table with processing times for each solver

Mesh	Time for solver by PETSc	Historical solver	Fastest Percentage (%)
Mesh 1	2 h 16 min 16 s	2 h 20 min 17 s	2.86
Mesh 2	3 h 17 min 39 s	3 horas 27 min 8 s	4.58

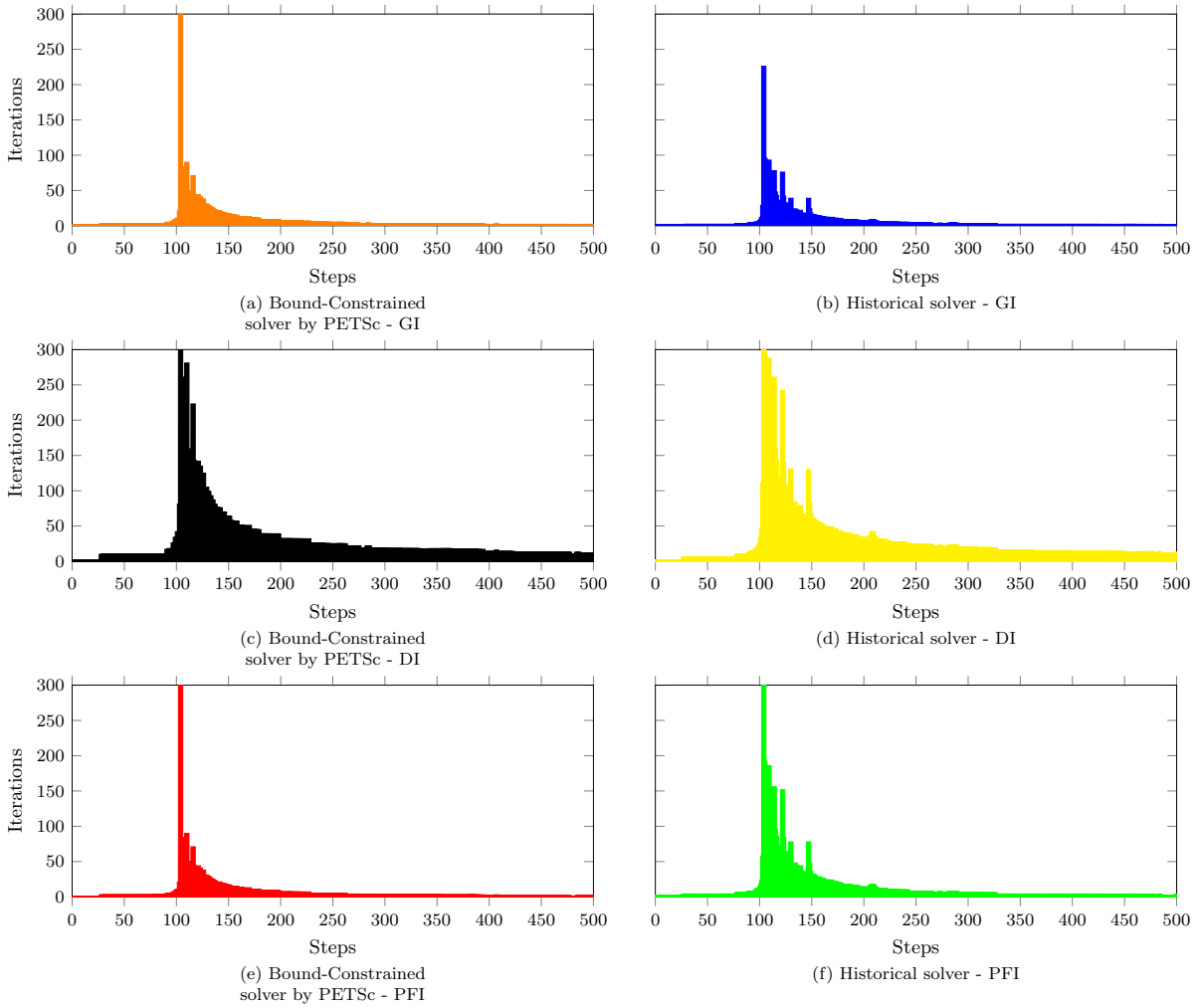


Figure 4.22: Number of Iterations for each solver and for iteration type for the L-Shaped Panel with Mesh 2.

4.3.2 Three Point Bending Beam

The problem used in this section is presented in Figure 4.23 with measurements in millimeters. The example was taken from Penna (2011). The displacement control point will be the same as the load application indicated in Figure 4.23. Three different meshes of

CST elements were built for the analysis, which are shown in Figure 4.24, and a zoom in the refined region is shown in Figure 4.25. In the unrefined region, all meshes have a size of 25 mm, while in the refined region each mesh has a value as shown in Table 4.4. The size of 6.25 mm in the refined region of the mesh was adopted for the initial mesh since the l_0 adopted is 12.5 mm, since Miehe et al. (2010a) says that for elements inside the crack, the mesh size should be at most half the length scale ($h \leq l_0/2$).

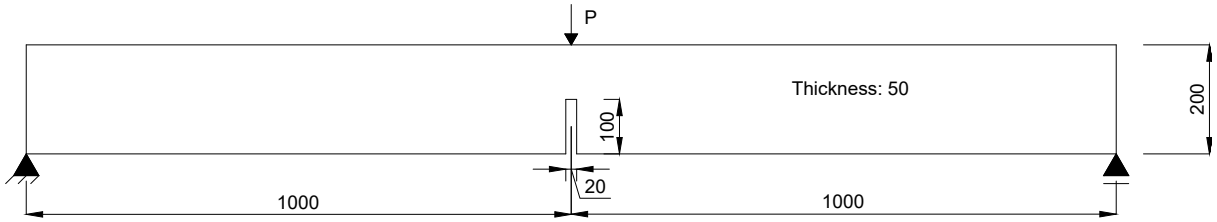
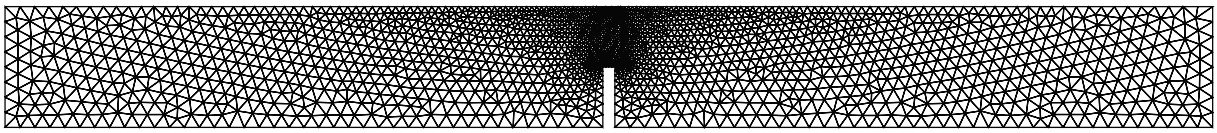
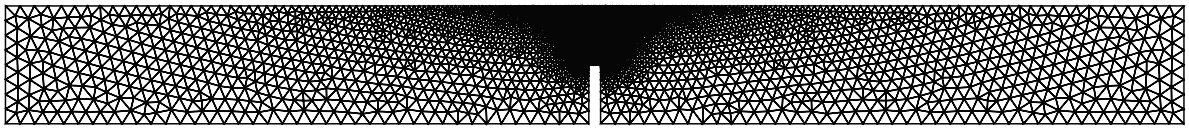


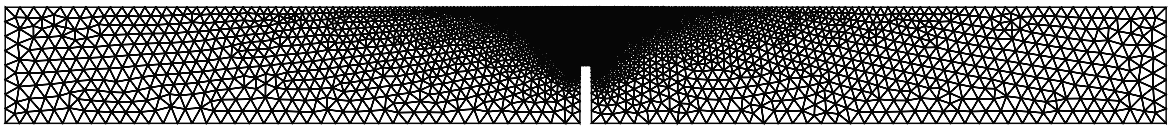
Figure 4.23: Problem setting for Three-Point Bending Beam with measurements in millimeters.



(a) Mesh 1

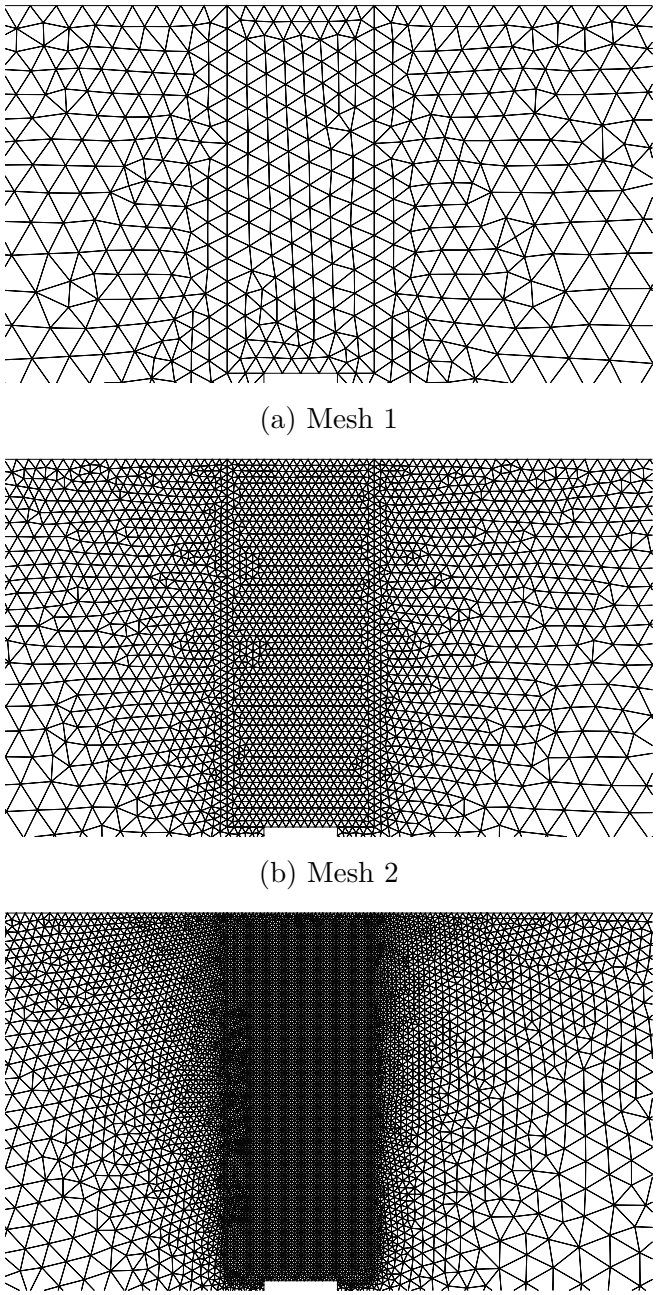


(b) Mesh 2



(c) Mesh 3

Figure 4.24: Different meshes for Three Point Bending Beam.



(a) Mesh 1

(b) Mesh 2

(c) Mesh 3

Figure 4.25: Zoom into the refined region of each mesh for the Three Pointing Bending Beam.

Table 4.4: Table with description of mesh refinement for the Three Pointing Bending Beam

Mesh	Mesh size in the fine region (mm)	Number of elements
Mesh 1	6.25	3686
Mesh 2	2	8872
Mesh 3	1	19830

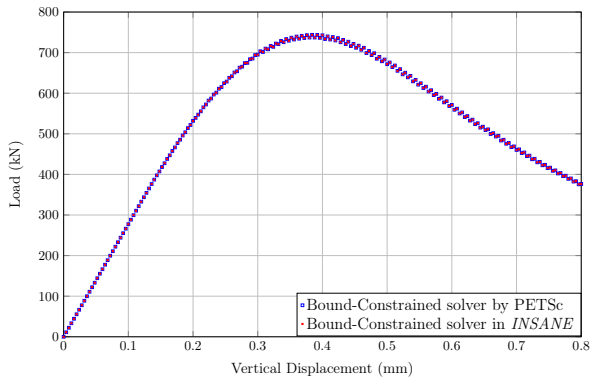
The following data for the Theree Point Bending are valid for the following subsections:

$E_0 = 30000$ N/mm^2, $\nu = 0.2$, $G_c = 0.124$ N/mm, $l_0 = 12.5$ mm. Increments of $4 \cdot 10^{-3}$ were considered in all cases for this section.

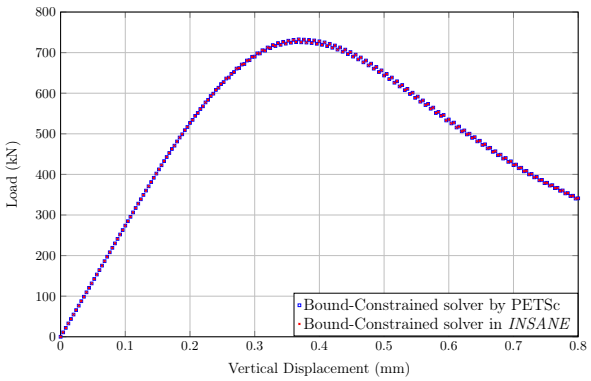
4.3.2.1 Comparison with Bound-Constrained Solver developed in *INSANE* by Bayao et al. (2021)

For the comparisson between bound-constrained solver by PETSc and bound-constrained solver developed in *INSANE*, the following data were used: $f_t = 3.3$ MPa, crack geometry function given by Equation 2.29 with $\xi = 2$, energy degradation function given by Equation 2.30 with Cornelissens's law for normal concrete ($\eta_1 = 3$ and $\eta_2 = 6.93$) and Wu (2018) constitutive model.

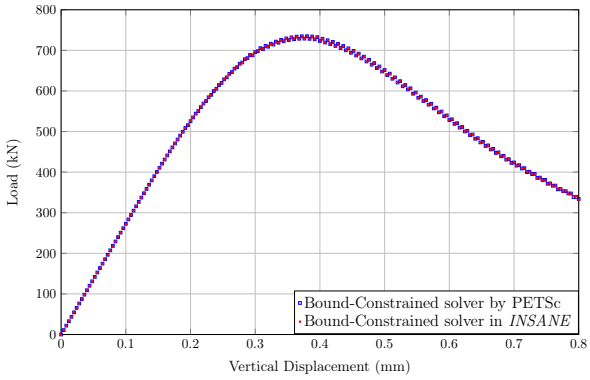
The results for the load versus vertical displacement of the point where the load is applied for each mesh are shown in Figure 4.26. The final crack path to mesh 4 for each solver is shown in Figure 4.27.



(a) Mesh 1

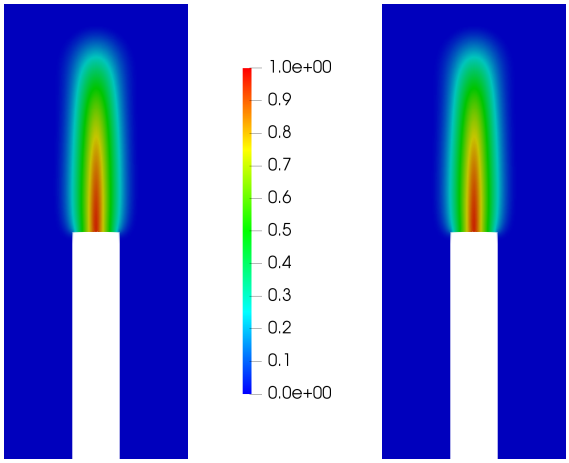


(b) Mesh 2



(c) Mesh 3

Figure 4.26: Curves of load versus displacement of the loading application node for Three Point Bending Beam with different meshes.



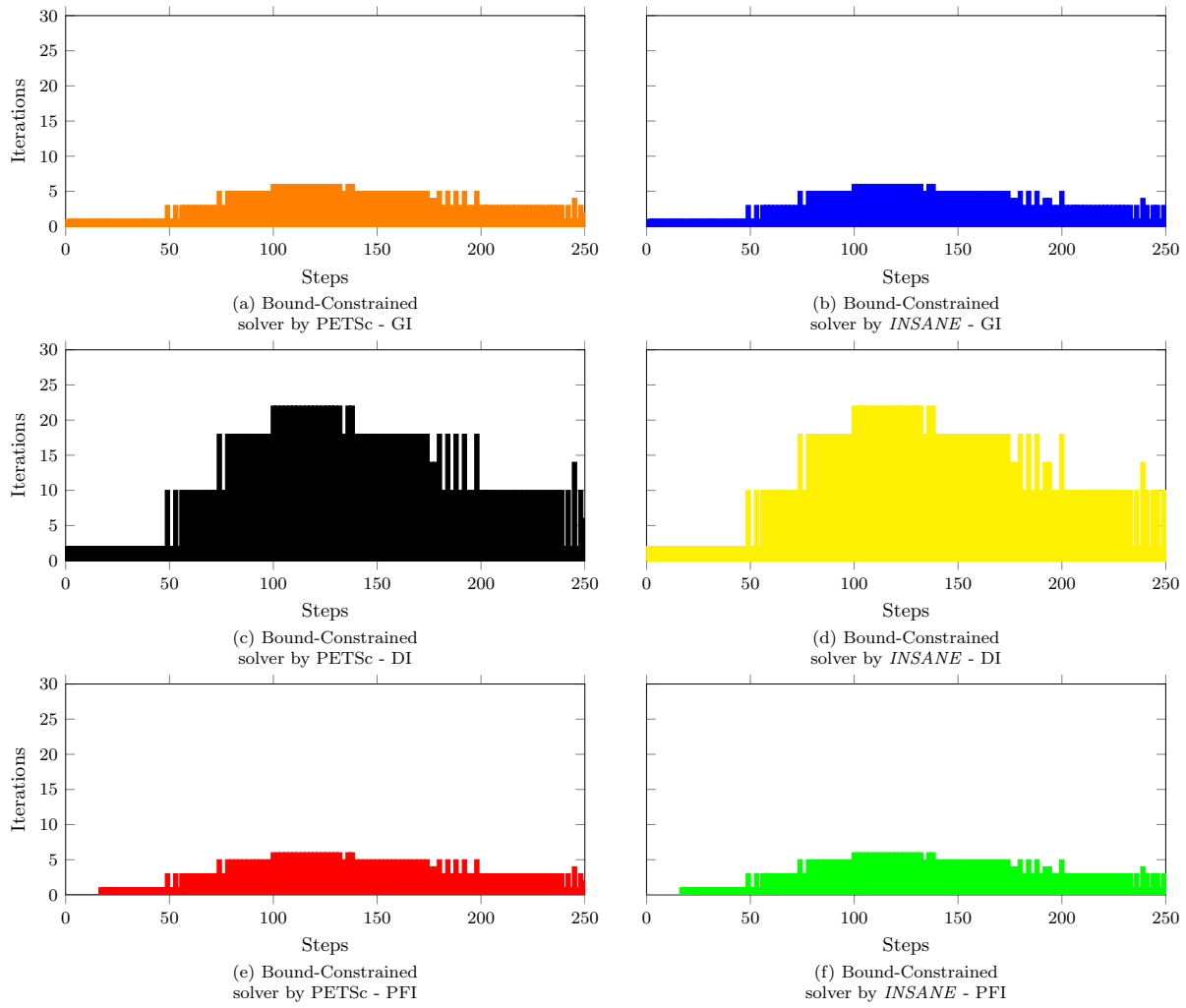
(a) (b)

Figure 4.27: Phase-field contour plots for Three Point Bending Beam with Mesh 3: (a) Bound-Constrained solver by PETSc (b) Bound-Constrained solver in *INSANE*.

Table 4.5 shows the times for each solver for each mesh. It is noticed again that when refining the mesh, the bound-constrained solver by PETSc is faster in relation to solver in *INSANE*. Figure 4.28 compares all types of iterations for each solver for mesh 3. Once again, it can be seen that both solvers have similar if not equal number of iterations, thus proving once again the equivalence of result between them.

Table 4.5: Table with processing times for each solver

Mesh	Time for solver by PETSc	Time for solver in *INSANE*	Fastest Percentage (%)
Mesh 1	7 min 11 s	7 min 26 s	3.36
Mesh 2	15 min 27 s	17 min 8 s	9.82
Mesh 3	31 min 53 s	42 min 9 s	24.36



(a) Bound-Constrained solver by PETSc - GI

(b) Bound-Constrained solver by *INSANE* - GI

(c) Bound-Constrained solver by PETSc - DI

(d) Bound-Constrained solver by *INSANE* - DI

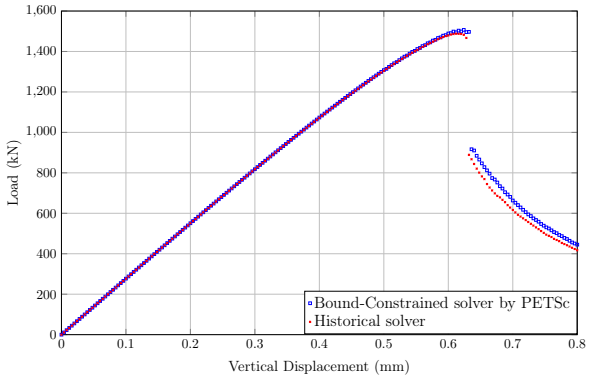
(e) Bound-Constrained solver by PETSc - PFI

(f) Bound-Constrained solver by *INSANE* - PFI

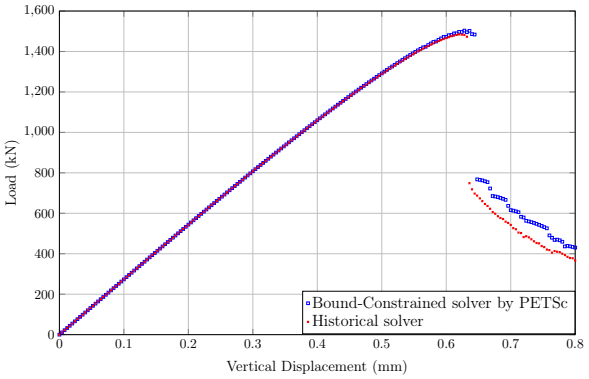
Figure 4.28: Number of Iterations for each solver and for iteration type for the Three Point Bendind Beam with Mesh 3.

4.3.2.2 Comparison with Historical Solver

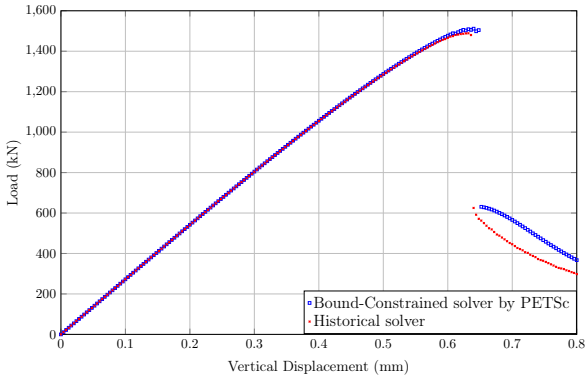
For the comparisson between bound-constrained solver by PETSc and historical solver, the following data were used: $\alpha = \phi^2$, $g = (1-\phi)^2$ and Miehe et al. (2010b) constitutive model.



(a) Mesh 1

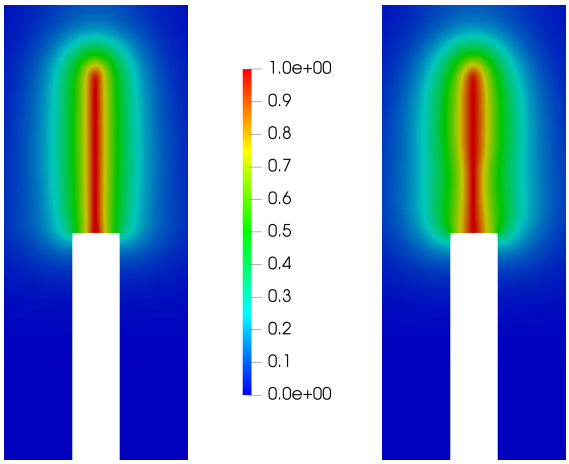


(b) Mesh 2



(c) Mesh 3

Figure 4.29: Curves of load versus displacement of the loading application node for Three Point Bending Beam with different meshes.



(a) (b)

Figure 4.30: Phase-field contour plots for Three Point Bending with Mesh 3: (a) Bound-Constrained solver by PETSc (b) Historical solver

Table 4.6 shows the times for each solver for each mesh. It is worth noting here that initially the negative percentage value represents that the historical solver was faster for

a coarser mesh. For mesh 3, the PETSc solver presented a lower percentage in relation to the historical solver, so more comparison analyzes between the solvers for increasingly refined meshes need to be performed, despite the fact that PETSc appears to be faster. Again, the Miehe et al. (2010b) constitutive model takes much longer than the Wu (2018) constitutive model. Figure 4.31 compare all types of iterations for each solver. Again, in this case the comparison is not so significant since they are different types of solvers, because at a time one solver has more or less one type of iteration than the other.

Table 4.6: Table with processing times for each solver

Mesh	Time for solver by PETSc	Historical solver	Fastest Percentage (%)
Mesh 1	10 min 46 s	10 min 41 s	-0.31
Mesh 2	21 min 11 s	24 min 27 s	13.36
Mesh 3	41 min 31 s	45 min 48 s	9.35

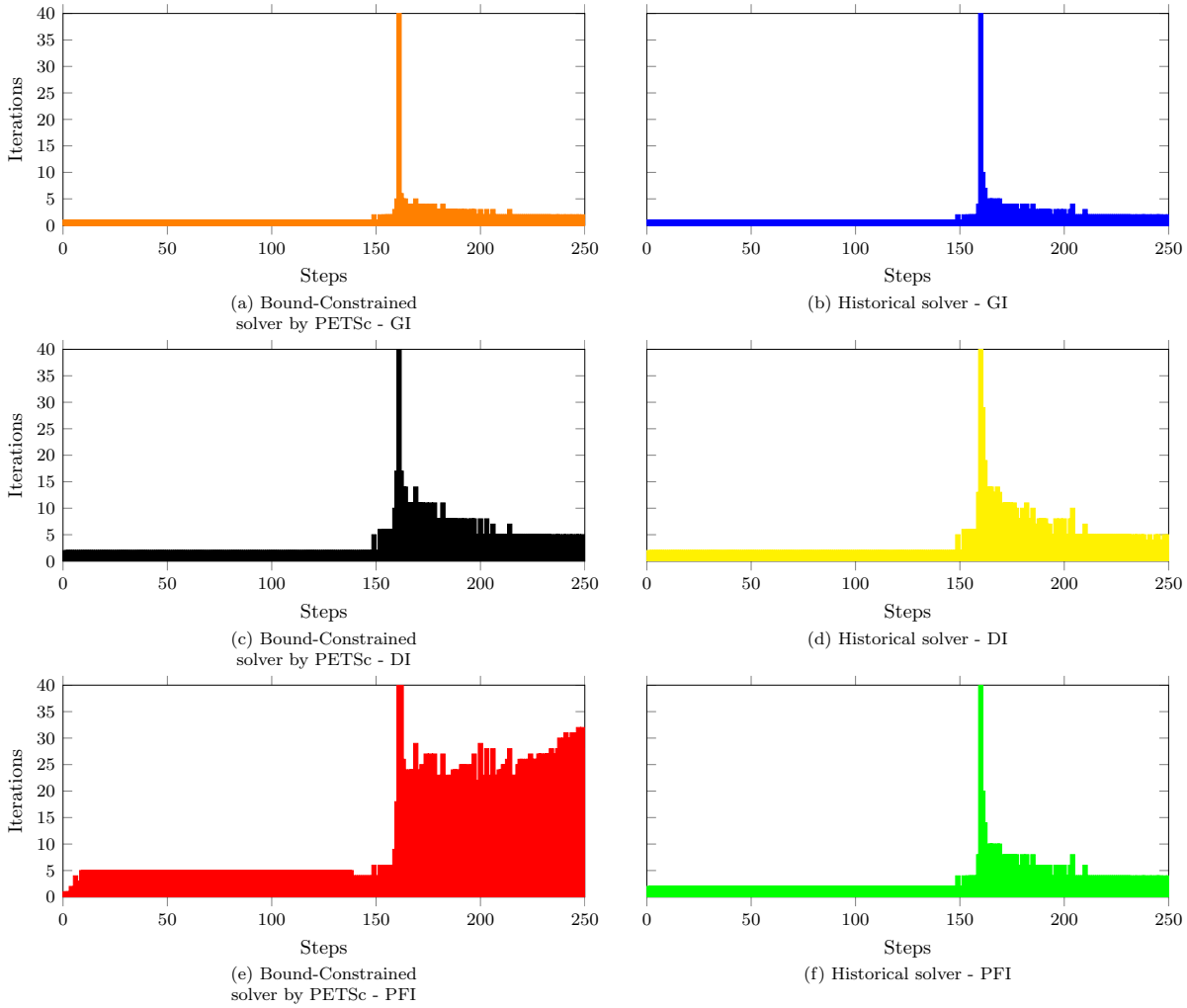


Figure 4.31: Number of Iterations for each solver and for iteration type for Three Point Bending Beam Test with Mesh 1.

4.3.3 Wedge Split Test

The problem in this section is presented in Figure 4.32 with measurements in millimeters. The example was taken from Wu (2018). For this problem, the section symmetry will be used as shown in Figure 4.33. The displacement control point will be the same as the load application indicated in Figure 4.33. In the load versus displacements curves, the opening value at that point will be shown in relation to its symmetrical one, being here called MOD (Mouth Open Displacement). Two different meshes of CST elements were built for the analysis, which are shown in Figure 4.34, and a zoom in the refined region is shown in Figure 4.35. In the unrefined region, all meshes have a size of 80 mm, while in the refined region each mesh has a value as shown in Table 4.4. The size of 4.25 mm in the refined region of the mesh was adopted for the initial mesh since the l_0 adopted is 8.5 mm, since Miehe et al. (2010a) says that for elements inside the crack, the mesh size should be at most half the length scale ($h \leq l_0/2$).

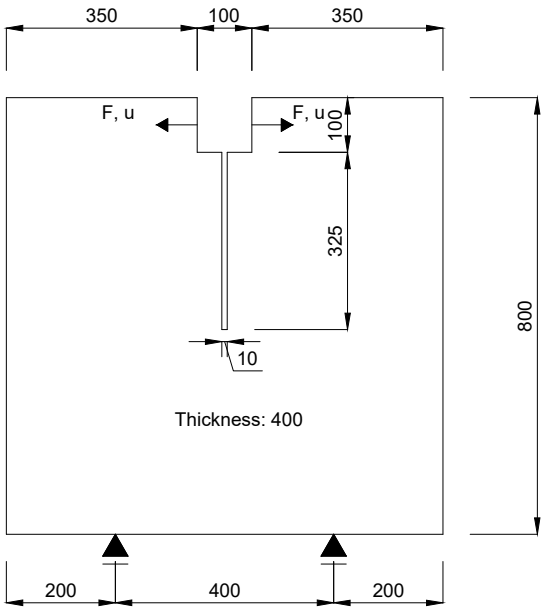


Figure 4.32: Problem setting for Wedge Split Test with measurements in millimeters.

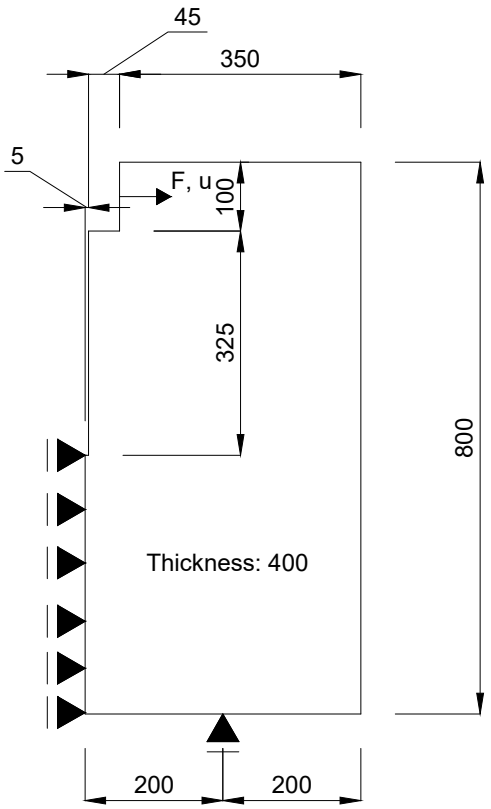
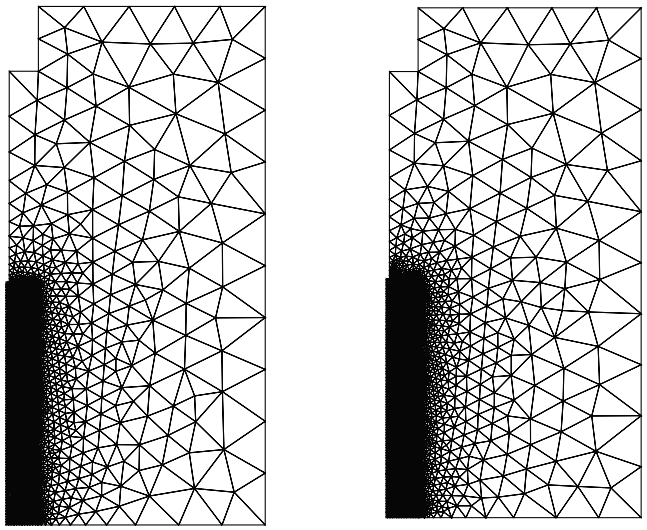
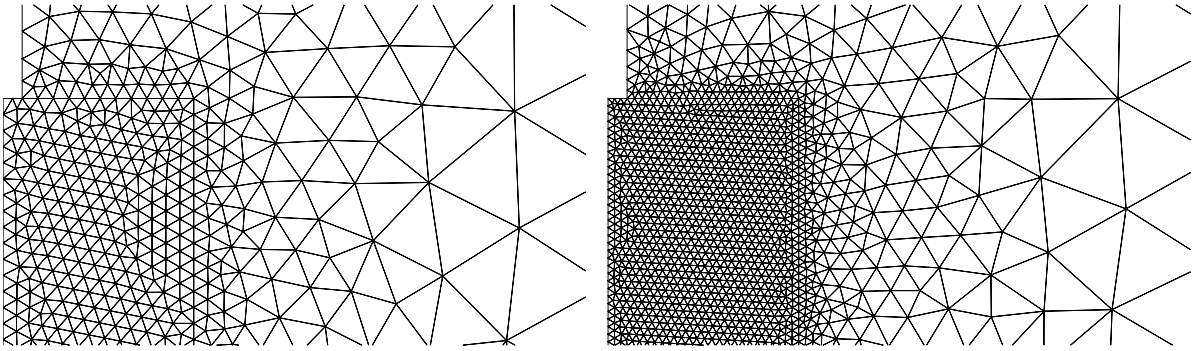


Figure 4.33: Symetric problem setting for Wedge Split Test with measurements in millimeters.



(a) Mesh 1 (b) Mesh 2

Figure 4.34: Different meshes for Wedge Split Test.



(a) Mesh 1 (b) Mesh 2

Figure 4.35: Zoom into the refined region of each mesh for the Wedge Split Test.

Table 4.7: Table with description of mesh refinement for the Wedge Split Test

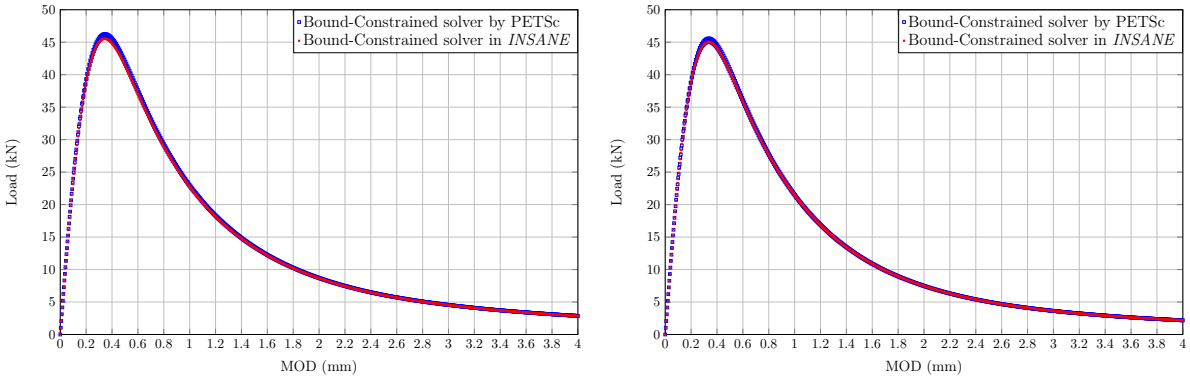
Mesh	Mesh size in the fine region (mm)	Number of elements
Mesh 1	4.25	3737
Mesh 2	2	13467

The following data for the Wedge Split Test are valid for the following subsections: $E_0 = 28300$ N/mm^2, $\nu = 0.18$, $G_c = 0.373$ N/mm, $l_0 = 8.5$ mm. Increments of $2 \cdot 10^{-3}$ were considered in all cases for this section.

4.3.3.1 Comparison with Bound-Constrained Solver developed in *INSANE* by Bayao et al. (2021)

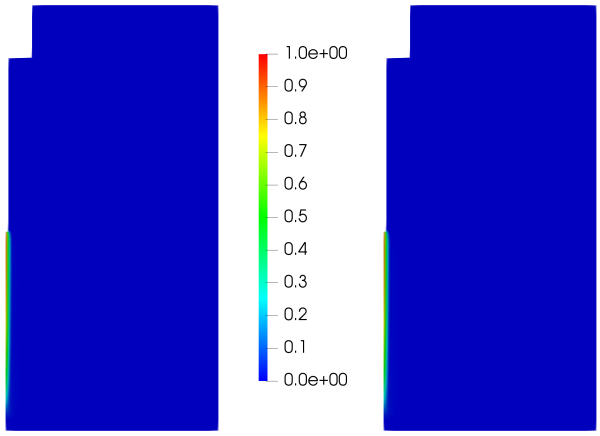
For the comparisson between bound-constrained solver by PETSc and bound-constrained solver developed in *INSANE*, the following data were used: $f_t = 2.12$ MPa, crack geometry function given by Equation 2.29 with $\xi = 2$, energy degradation function given by Equation 2.30 with Cornelissens's law for normal concrete ($\eta_1 = 3$ and $\eta_2 = 6.93$) and Wu (2018) constitutive model.

The results for the load versus MOD for each mesh are shown in Figure 4.36. The final crack path to mesh 2 for each solver is shown in Figure 4.37.



(a) Mesh 1 (b) Mesh 2

Figure 4.36: Curves of load versus MOD for different meshes.



(a) (b)

Figure 4.37: Phase-field contour plots for Wedge Split Test for Mesh 2: (a) Bound-Constrained solver by PETSc (b) Bound-Constrained solver in *INSANE*.

Table 4.8 shows the times for each solver and mesh. Again, the bound-constrained solver by PETSc is faster in relation to solver in *INSANE*. Figure 4.38 compares all types of iterations for each solver for mesh 2. Once again, it can be seen that both solvers have similar if not equal number of iterations, thus proving once again the equivalence of result between them.

Table 4.8: Table with processing times for each solver

Mesh	Time for solver by PETSc	Time for solver in *INSANE*	Fastest Percentage (%)
Mesh 1	34 min 58 s	36 min 26 s	4.02
Mesh 2	2 min 9 min 52 s	2 h 34 min 57 s	16.19

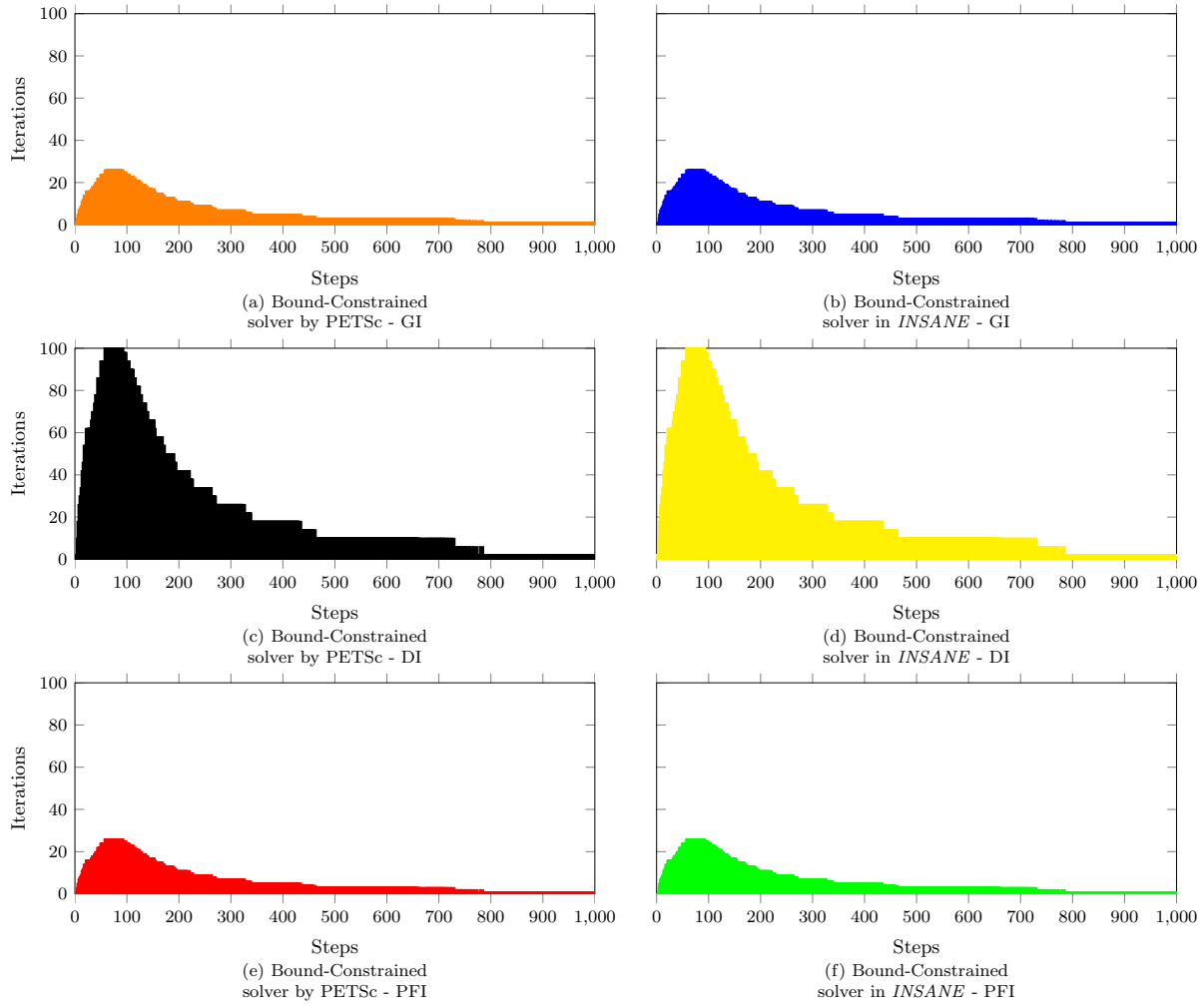
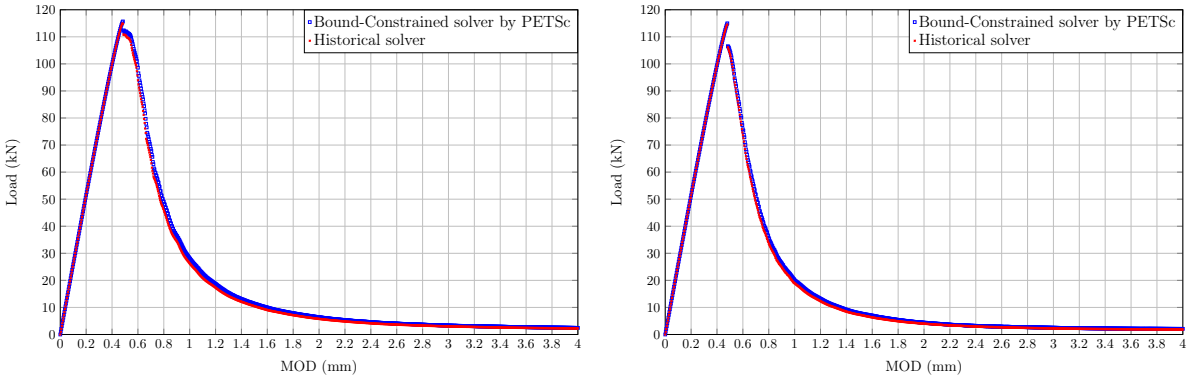


Figure 4.38: Number of Iterations for each solver and for iteration type for the Wedge Split Test with Mesh 2.

4.3.3.2 Comparison with Historical Solver

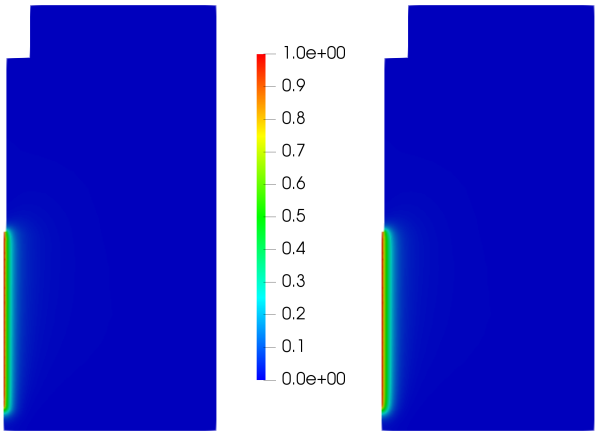
For the comparisson between bound-constrained solver by PETSc and historical solver, the following data were used: $\alpha = \phi^2$, $g = (1-\phi)^2$ and Miehe et al. (2010b) constitutive model.

The results for the load versus MOD for each mesh are shown in Figure 4.39. The final crack path to mesh 2 for each solver is shown in Figure 4.40.



(a) Mesh 1 (b) Mesh 2

Figure 4.39: Curves of load versus MOD for different meshes.



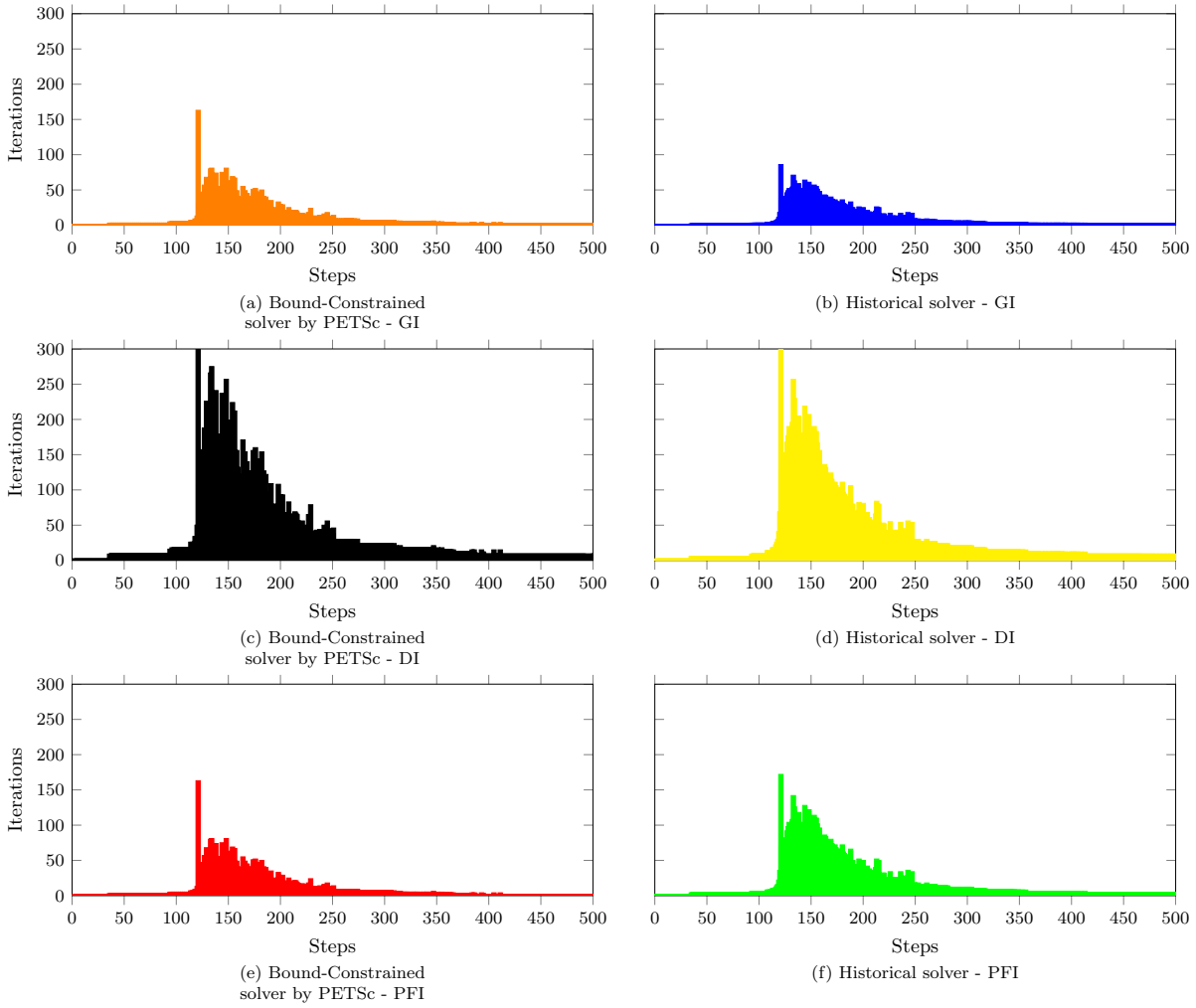
(a) (b)

Figure 4.40: Phase-field contour plots for Wedge Split Test for Mesh 2: (a) Bound-Constrained solver by PETSC (b) Historical solver.

Table 4.9 shows the times for each solver for each mesh. The solver by PETSc also shows a time performance improvement over the historical solver as the mesh is refined, but a very small improvement. It is worth noting here that initially the negative percentage value represents that the historical solver was faster for a coarser mesh. Again, the Miehe et al. (2010b) constitutive model takes much longer than the Wu (2018) constitutive model. Figure 4.41 compares all types of iterations for each solver. Again, in this case the comparison is not so significant since they are different types of solvers.

Table 4.9: Table with processing times for each solver

Mesh	Time for solver by PETSc	Historical solver	Fastest Percentage (%)
Mesh 1	1 h 36 min 14 s	1 h 31 min 56 s	-4.68
Mesh 2	6 h 6 min 38 s	6 h 24 min 28 s	4.68



(a) Bound-Constrained solver by PETSc - GI

(b) Historical solver - GI

(c) Bound-Constrained solver by PETSc - DI

(d) Historical solver - DI

(e) Bound-Constrained solver by PETSc - PFI

(f) Historical solver - PFI

Figure 4.41: Number of Iterations for each solver and for iteration type for the Wedge Split Test with Mesh 2.

Chapter 5

Case Studies

The purpose of this chapter is to present two case studies made. The first study is a comparison made between the phase-field model and the fixed-smeared crack present in *INSANE*. The second study is show the efficiency of the phase-field models in relation to modes I and II failures. In all examples displacement control was used, the local tolerance used was of $1 \cdot 10^{-4}$ and the global tolerance was of $1 \cdot 10^{-4}$. All cases are in a plane stress state.

5.1 Comparation between the phase-field model and smeared cracking model

In this section, the phase-field model will be compared with the generalized fixed-smeared crack model implemented in *INSANE* by Penna (2011). The comparison that will be made is about the load by displacement responses as the mesh is refined.

In the comparison, the same example of Figure 4.32 will be used, but now with the triangular crack, because if the model was made equal to Figure 4.33, the initial mesh at the beginning of the analysis for the fixed-smeared crack model would present localization problems, with problems in the numerical response, due to the presence of a small triangular element. Again, symmetry was used as shown in Figure 5.1. Again, the displacement control point will be the same as the load application indicated in Figure 5.1 and in the load versus displacements curves, the opening value at that point will be shown in relation to its symmetrical one, being here called MOD (Mouth Open Displacement).

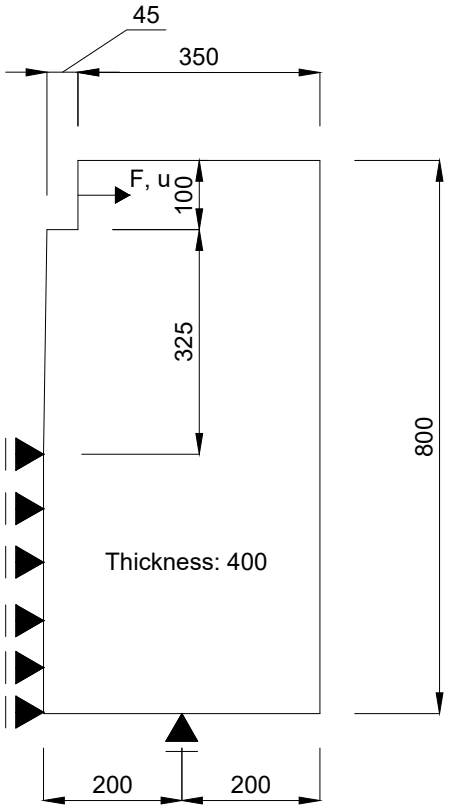


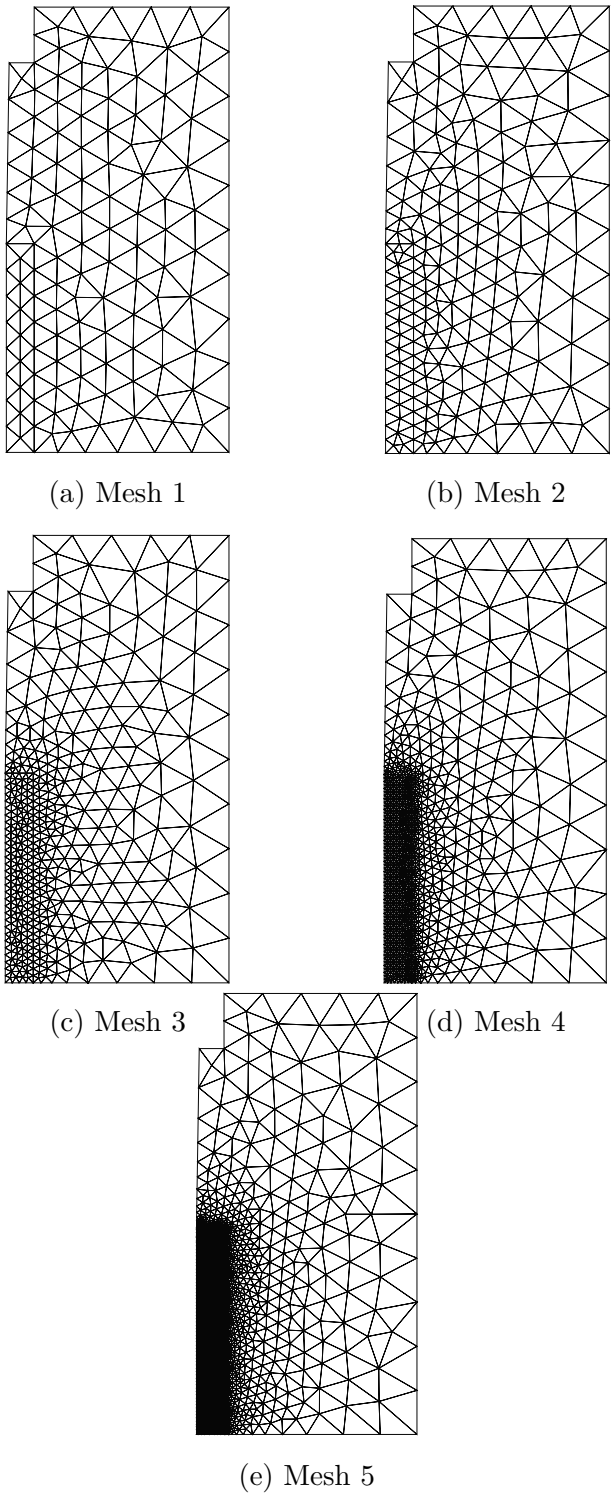
Figure 5.1: Problem setting for Wedge Split Test with measurements in millimeters.

This analysis will check both the convergence capacity of the models and the mesh dependence. For this, it is observed that in Figure 5.2, different refinements will be made in the region below the crack. In the unrefined region, all meshes have a size of 80 mm, while in the refined region each mesh has a value as shown in Table 5.1.

Table 5.1: Table with description of mesh refinement for the Wedge Split Test

Mesh	Mesh size in the fine region (mm)
Mesh 1	50
Mesh 2	25
Mesh 3	12.5
Mesh 4	6.25
Mesh 5	3

For the phase-field model, the following data were used: $E_0 = 28300.0$ N/mm^2, Poisson´s ratio $\nu = 0.18$, $G_c = 0.373$ N/mm, $l_0 = 50$ mm, $f_t = 2.12$, $\xi = 2$, Cornelissen's softening law for normal concrete ($\eta_1 = 3$ and $\eta_2 = 6.93$), Wu (2018) constitutive model and without linear search. For the fixed smeared crack model, the same following data were used with $f_c = 31$ Nmm^2, L_c (characteristic length of the material) equals to 150 mm and shear retention factor (β_r) equals to zero. The model was calibrated with the Carreira-Ingraffea material, in order to present the same response material. The phase-



(a) Mesh 1 (b) Mesh 2

(c) Mesh 3 (d) Mesh 4

(e) Mesh 5

Figure 5.2: Different meshes for the Wedge Split Test.

field model with bound-constrained solver by PETSc with Wu (2018) constitutive model was calibrated with the fixed-smeared crack in *INSANE* (Figure 5.3), obtaining the aforementioned values of $l_0 = 50$ mm for a $L_c = 150$ mm. Increments of $2 \cdot 10^{-3}$ were considered in all cases in this section.

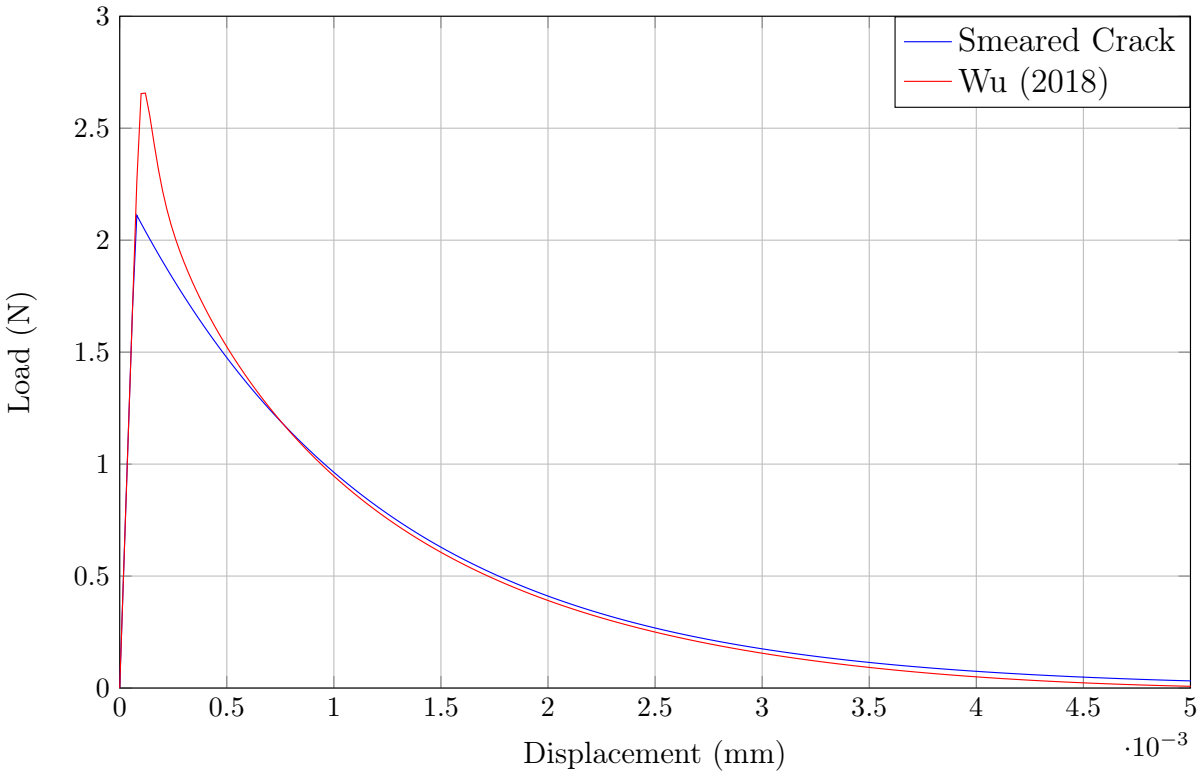


Figure 5.3: Wedge Split Test calibration.

The results for load versus MOD for phase-field model are shown in Figure 5.4, while for the fixed-smeared crack model are shown in Figure 5.5. Analyzing Figure 5.4, it can be seen that for phase-field models, the numerical model always processes for all steps, and as it refines the mesh, the result converges to a response. Now, analyzing Figure 5.5, it can be seen that for the fixed-smeared crack models, the numerical model does not process for all steps, and as it refines the mesh, it processes fewer and fewer steps. Furthermore, the result does not converge to the fixed-smeared crack model. From these analyses, it can be concluded that the phase-field model has an advantage over the fixed-crack model, since it does not depend on how the mesh is arranged to process results, and converges as the mesh refines. That is, the fixed-smeared crack has a large mesh dependency while the phase-field model does not.

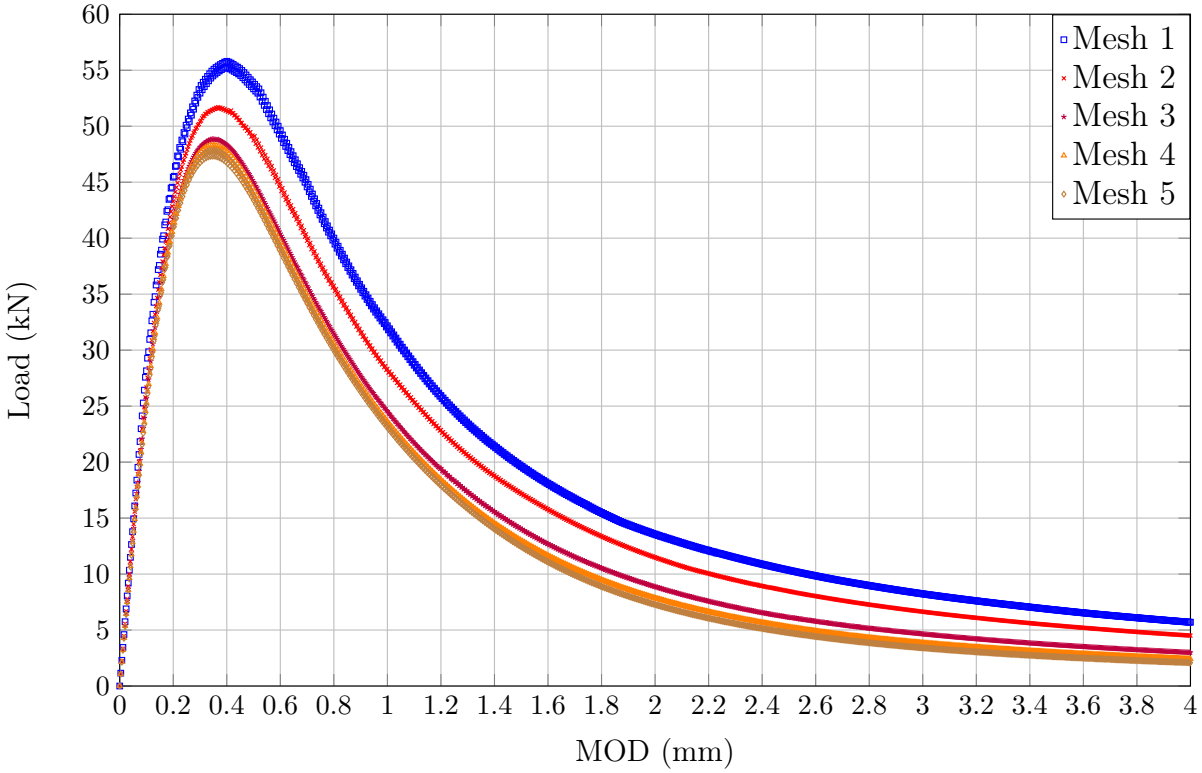


Figure 5.4: Curves of load versus MOD for phase-field model.

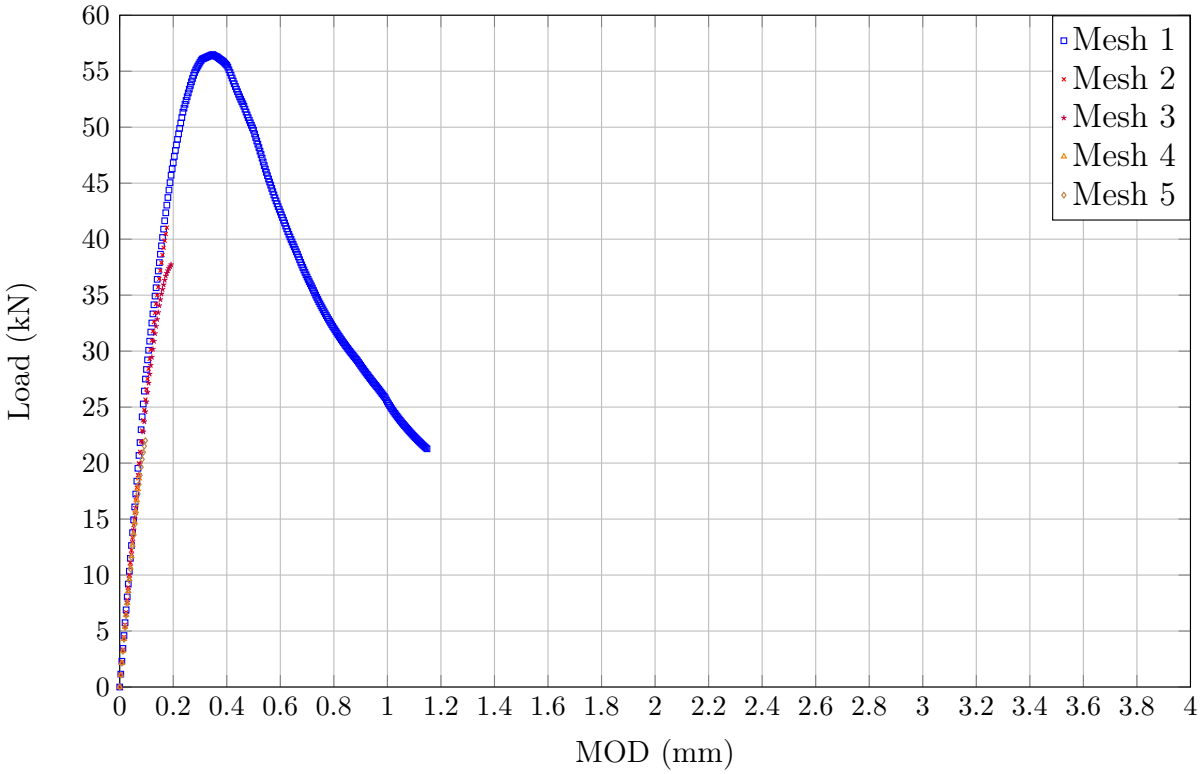


Figure 5.5: Curves of load versus MOD for smeared crack model.

5.2 Analysis of phase-field models with failure modes

The development of Equations 2.34 and 2.33, as well as the softening laws was done by Wu (2017) for mode I failure, but it is used as models that have mode II in Wu (2018). Wu et al. (2021) states that even having been developed for mode I failure, the theory can be used for mixed modes I+II/III failure as well as for 3D models. The objective of this section is to compare the results of the phase-field model in relation to the experimental results for mode I, using examples already covered in this dissertation, and also compares with an example that uses mode II.

5.2.1 Models with mode I failure

5.2.1.1 L-Shaped Panel

To begin, the result of Figure 4.11 will be shown again in Figure 5.6, but now only with the result of the bound-constrained solver by PETSc to better compare with the experimental result given by Winkler (2001) and with smeared crack model for Carreira-Ingraffea material obtained by Penna (2011). It is worth remembering here that the data for this problem are described in Section 4.2.1, the panel measurements in Figure 4.1 and the mesh in Figure 4.2.

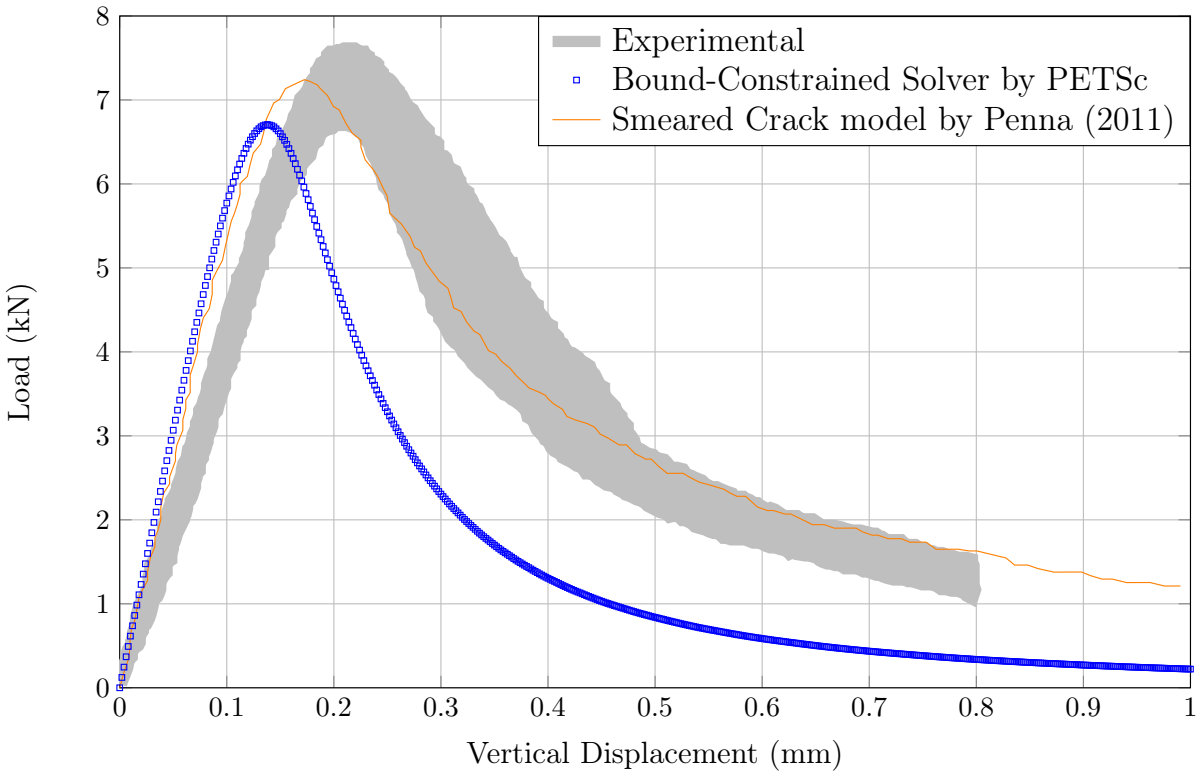


Figure 5.6: Curves of load versus displacement of the loading application node.

It is noticed that the result for the L-shaped panel is not completely the same as it

does not fit the experimental range, even when calibrated with the fixed-smeared crack. However, as they describe the format well, and much better than the historical solver as explained in Section 4.2.1, it can be said that the phase-field model manages to capture mode I failure reasonably well for this case.

5.2.1.2 Three Point Bending Beam

The second example will be the three point bending beam with the measurements shown in Figure 4.23 with mesh 3 shown in Figure 4.24. In addition, l_0 was calibrated with the data used by Penna (2011), which are: $f_c = 33$ N/mm^2, L_c equals to 40 mm and shear retention factor (β_r) equals to zero. The model was calibrated with the Carreira-Ingraffea material, in order to present the same response material. Thus, phase-field model with bound-constrained solver by PETSc without line search and with Wu (2018) constitutive model was calibrated 5.11 shows the calibration for $l_0 = 12.5$ mm, that will be used. So for this problem you have the same data presented in Section 4.3.2.1.

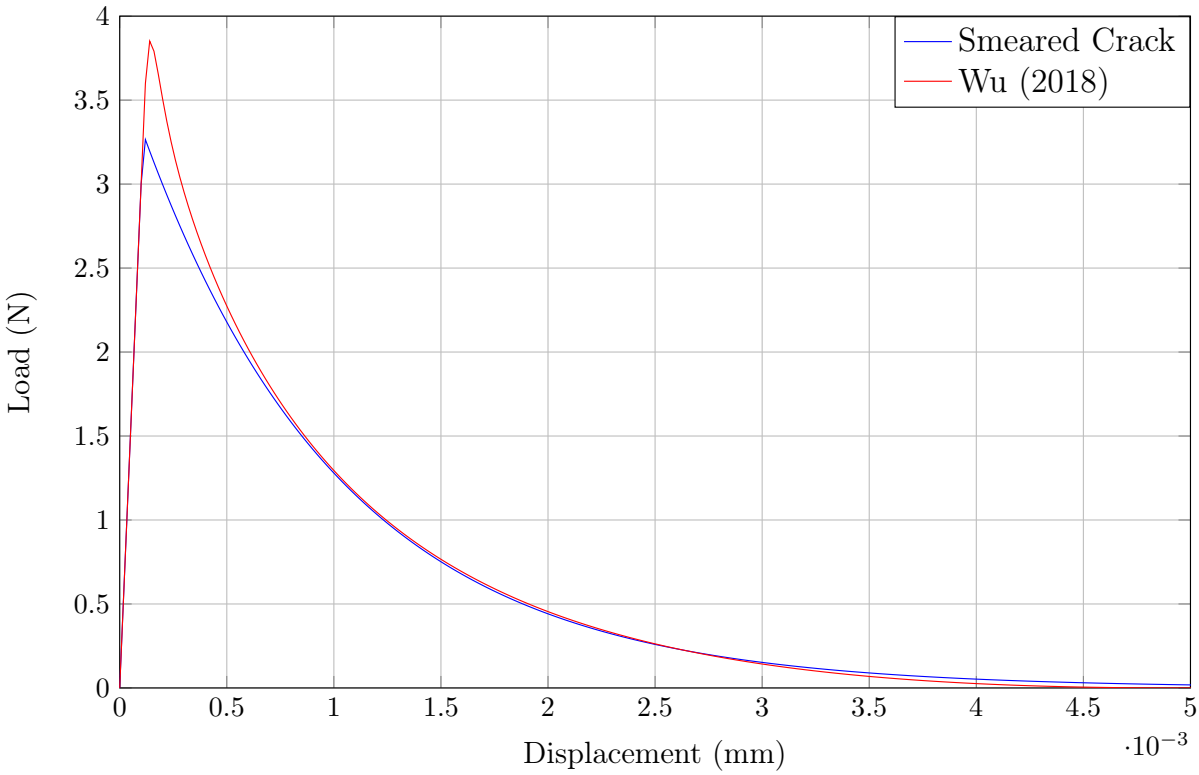


Figure 5.7: Single-Edge Notched Beam calibration.

The result of Figure 4.26 for mesh 3 will be shown again in Figure 5.6, but now only with the result of the bound-constrained solver by PETSc to better compare with the experimental result given by Petersson (1981) and with smeared crack model for Carreira-Ingraffea material obtained by Penna (2011). It is noticed that this model fits very well with the experimental results, validating the phase-field model with mode I failure.

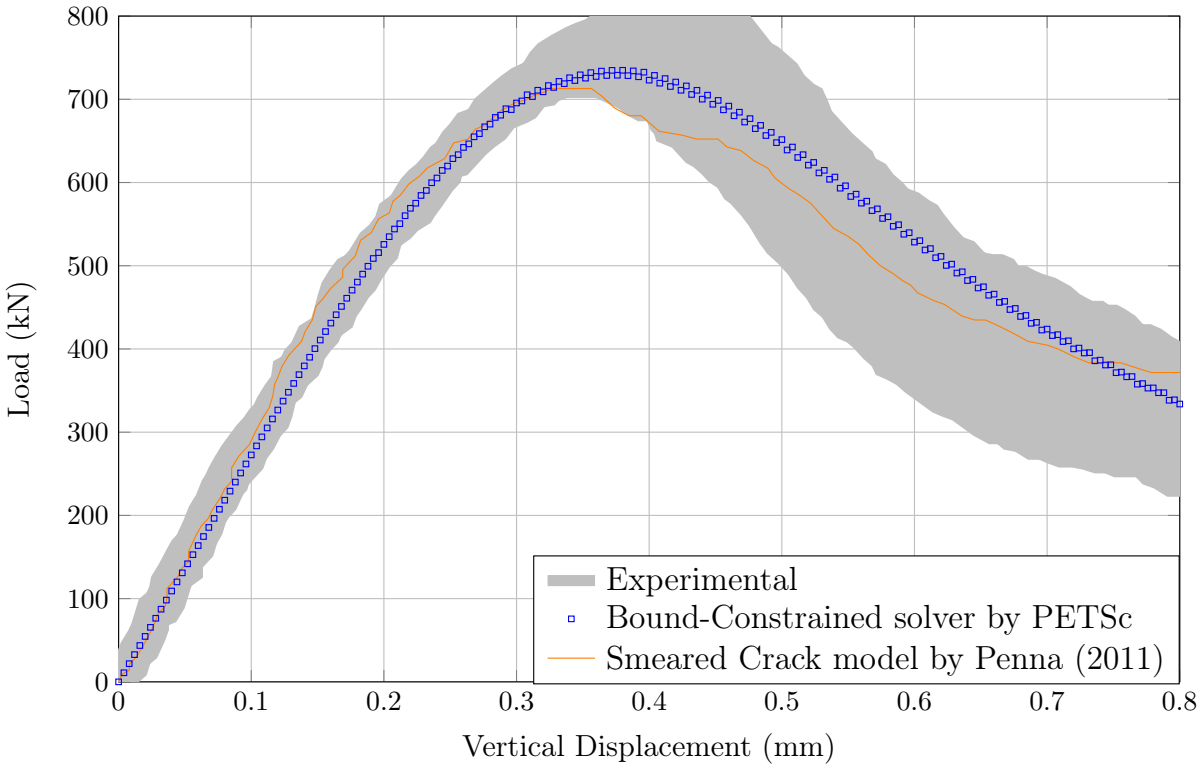


Figure 5.8: Curves of load versus displacement of the loading application node.

5.2.2 Model with mode II failure

Finally, a new problem will be shown that has mode II failure as predominant. This is the single-edge notched beam under proportional loading reported by Arrea and Ingrafeaa (1982). The problem data are presented in Figure 5.9 with measurements in millimeters and the mesh used in Figure 5.10, where the thin region has an element with a size of 5 mm and the thick region has an element with a size of 25 mm.

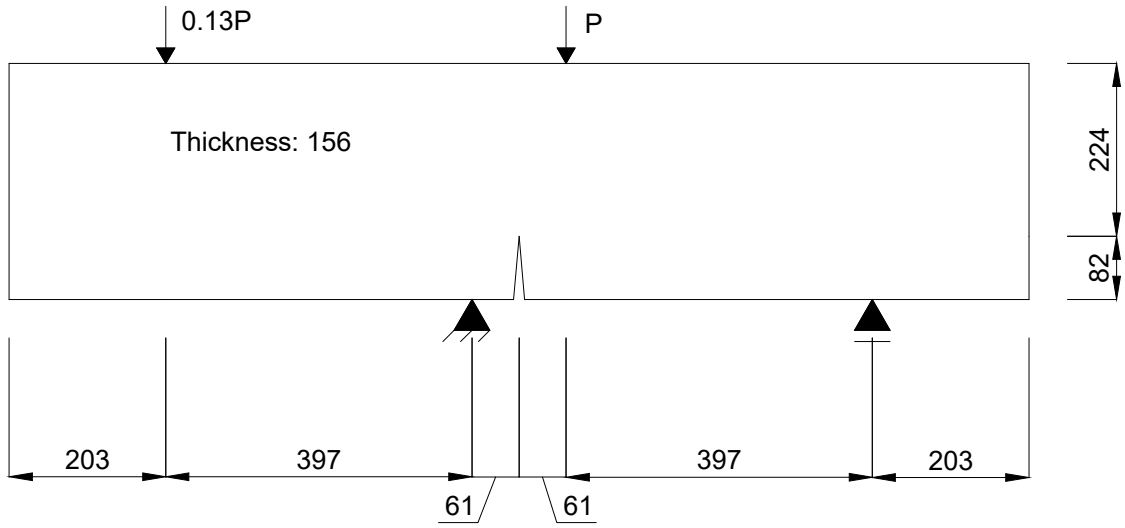


Figure 5.9: Problem setting for Single-Edge Notched Beam with measurements in millimeters.

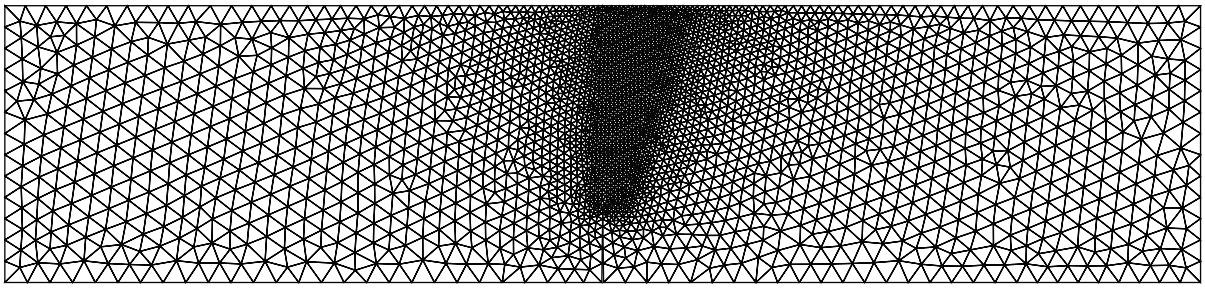


Figure 5.10: T3 mesh for Single-Edge Notched Beam.

The data used for the problem are the same used by Arrea and Ingrafeaa (1982), which are: $E_0 = 24800$ N/mm^2 and $\nu = 0.18$. Arrea and Ingrafeaa (1982) varied the values of G_c and f_t. Here, the intermediate values of this interval will be used, as they are the same values used for the smeared crack problem by Penna (2011), which corresponds to $G_c = 3.4$ N/mm and $f_t = 0.12$ MPa. In addition, l_0 was calibrated with the data used by Penna (2011), which are: $f_c = 34$ N/mm^2, L_c equals to 28 mm and shear retention factor (β_r) equals to 0.02. The model was calibrated with the Carreira-Ingraffea material, in order to present the same response material. The phase-field model with bound-constrained solver by PETSc without line search and with Wu (2018) constitutive model was calibrated with the fixed-smeared crack in *INSANE*. Figure 5.11 shows the calibration for $l_0 = 12.5$ mm, that will be used.

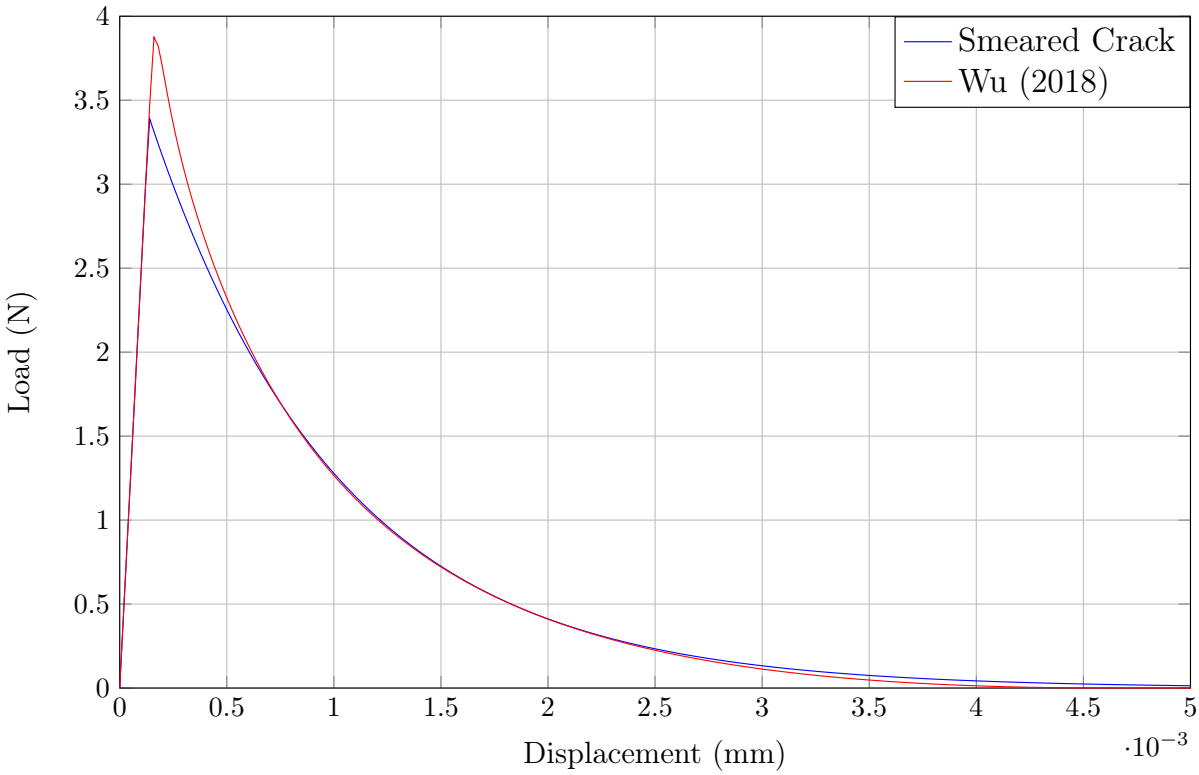


Figure 5.11: Single-Edge Notched Beam calibration.

The results presented are related to the relative vertical displacement of the crack ends. This measure is known as CMSD (Crack Mouth Sliding Displacement). The result for the load versus CMSD is shown in Figure 5.12 with the experimental results being given by Arrea and Ingrafeaa (1982) and the result for smeared crack model for Carreira-Ingraffea material obtained by Penna (2011). For the phase-field model, were used crack geometry function given by Equation 2.29 with $\xi = 2$, energy degradation function given by Equation 2.30 with Cornelissens's law for normal concrete ($\eta_1 = 3$ and $\eta_2 = 6.93$) and Wu (2018) constitutive model. The displacement control point is the right point of the crack. Increments of $2 \cdot 10^{-3}$ were considered. The final crack path for phase-fiel model is shown in Figure 5.13.

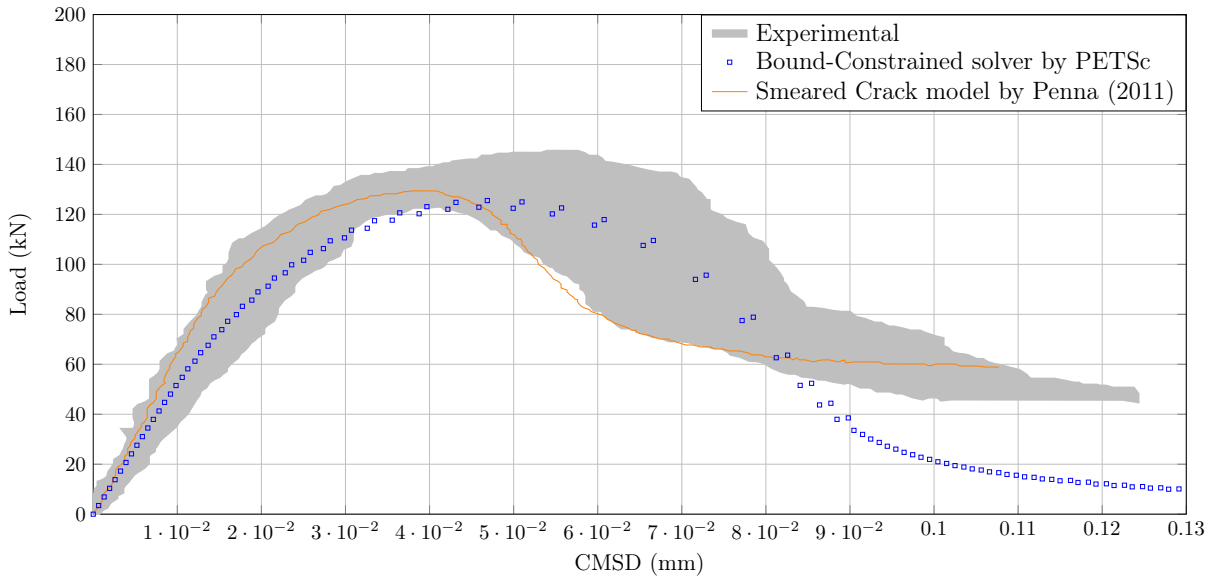


Figure 5.12: Curves of load versus CMSD.

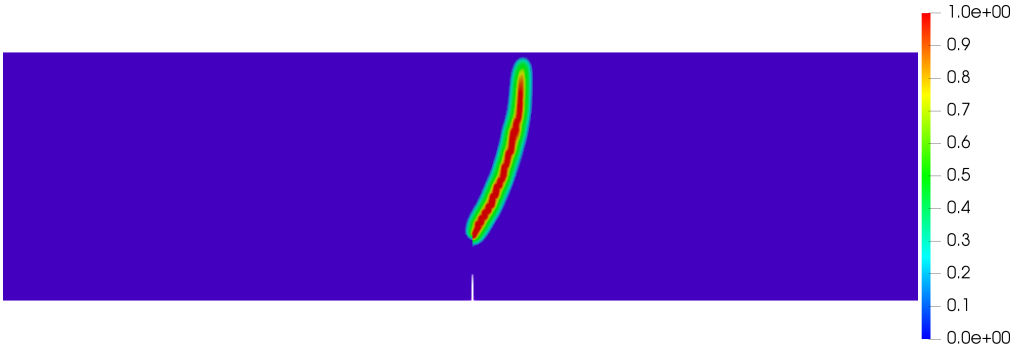


Figure 5.13: Phase-field contour plot for Single-Edge Notched Beam.

It is worth remembering here that the issue of numerical instability presented for the phase-field model in Figure 5.12 was discussed in Section 4.2.2. It can be seen in Figure 5.12 that the result does not fit completely with the experimental results, but the phase-field model captures mode II failure well.

Chapter 6

Conclusions and future research topics

The main objective of the work was achieved, which was the understanding of the phase-field model and the implementation of the bound-constrained solver by PETSc.

The phase-field and bound-constrained solver theory were studied and presented. The communication between *INSANE* and the bound-constrained solver by PETSc was successfully implemented, enabling to explore differents crack geometry functions and different energy degradation functions, without limitations. The need for a bound-constrained solver was shown. In addition, the efficiency of the bound-constrained solver by PETSc with respect to the bound-constrained solver in *INSANE* as the mesh is refined was shown, which was already expected due to being an optimized external library. A clear advantage of the bound-constreained solver in relation to the historical solver from the performance point of view was not evidenced. However it must be remembered that the historical solver has limitations in the energy degradation functions and crack geometry function that can be used with this solver. It was shown that the phase-field models show convergence with refinement and less mesh dependence in relation to the smeared cracking models. Finally, it was shown that the theory developed by Wu (2017) for mode I failure applies well to mode I and II failures, through numerical models.

This work shows the importance of developing a software like *INSANE*, where numerical problems can be processed with different models and solvers in order to be able to compare and validate. Not only that, this work shows the ease of expansion of a research and computational implementation by *INSANE*, as it used the base made by the work of Leão (2021) and Bayao et al. (2021), as well as other parts of the software that were developed over several years.

Although there are works and materials on JNI, this work facilitates even more this implementation with which it is exposed in the appendix Appendix A.

Finally, some recommendations for future research for the research group are:

- Study more about possible linear searches for the bound-constrained solver.

- Adaptive refinement so don't need to do a pre-refinement.
- Implement the phase-field modelling for the 3D case.
- To further study the need to develop a theory for mode II failure.
- Implement and study new hybrid constitutive energy separation models.
- Deepen the study of the relationship between the parameters for the PFMs, such as the length scale, with the properties of the materials.
- Expand possibilities for studying the phase-field with other problems such as anisotropy of materials, fatigue, dynamics, failure in composite materials, among other topics.

Bibliography

Ahrens, J., Geveci, B. and Law, C. (2015), 'Paraview: An end-user tool for large data visualization', *Elsevier* .

Alessi, R., Marigo, J.-J. and Vidoli, S. (2015), 'Gradient damage models coupled with plasticity: Variational formulation and main properties', *Mechanics of Materials* vol. 80, 351 – 367.

Amor, H., Marigo, J.-J. and Maurini, C. (2009), 'Regularized formulation of the variational brittle fracture with unilateral contact: Numerical experiments', *Journal of the Mechanics and Physics of Solids* vol. 57(8), 1209–1229.

Arrea, M. and Ingrafeaa, A. (1982), Mixed-mode crack propagation in mortar and concrete., Technical Report No. 81-13, Department of Structural Engineering, Cornell University, Ithaca, NY.

Ayachit, U. (2015), The paraview guide: A parallel visualization application, Technical Report ISBN-13: 978-0123875822, Kitware.

Azevedo, G. M. (2019), Implementação paralela para analises estáticas lineares pelo métodos dos elementos finitos, Master dissetation, UFMG - Unversidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil.

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Karpeyev, D., Kaushik, D., Knepley, M. G., May, D. A., McInnes, L. C., Mills, R. T., Munson, T., Rupp, K., Sanan, P., Smith, B. F., Zampini, S., Zhang, H. and Zhang, H. (2020), PETSc users manual, Technical Report ANL-95/11 - Revision 3.14, Argonne National Laboratory.
URL: *https://www.mcs.anl.gov/petsc*

Bayão, R. (2021), 'Estudo de modelos de campo de fase para análise de fratura'. Monografia (Engenharia Mecânica), UFMG (Universidade Federal de Minas Gerais), Belo Horizonte, MG, Brazil.

Bayao, R., Leao, H. M., Fortes, M. M., Pitangueira, R. L. S. and Gori, L. G. (2021), Implementation of bound-constrained solver in phase-field modeling of fractrure, *in*

'Proceedings of 42nd Ibero-Latin-American Congress on Computational Methods in Engineering (XLII CILAMCE) and 3rd Pan American Congress on Computational Mechanics (III PANACM)'.
URL: *https://cilamce-panacm2021.com.br/*

Benson, S. J. and Munson, T. S. (2006), 'Flexible complementarity solvers for large-scale applications', *Optimization Methods and Software* vol. 21(1), 155–168.

bogotobogo (2020), 'Eclipse cdt & jni (java native interface) with 64 bit mingw - 2020', Available in: `https://www.bogotobogo.com/cplusplus/eclipseCDTJNIMinGW64bit.php`. Acess in: 28/02/2021.

Bourdin, B., Francfort, G. and Marigo, J. J. (2000), 'Numerical experiments in revisited brittle fracture', *Journal of the Mechanics and Physics of Solids* vol. 48, 797–826.

Cornelissen, H. A. W., Hordijk, D. A. and Reinhardt, H. W. (1986), 'Experimental determination of crack softening characteristics of normalweight and lightweight concrete', *Heron* vol. 31(2), 45–56.

Farrell, P. and Maurini, C. (2017), 'Linear and nonlinear solvers for variational phase-field models of brittle fracture', *International Journal for Numerical Methods in Engineering* vol. 109(5), 648–667.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.5300*

Fuina, J. S. (2004), Métodos de controle de deformações para análise nao-linear de estruturas, Master dissetation, Universidade Federal de Minas Gerais (UFMG). (in portuguese).

Geelen, R. J., Liu, Y., Hu, T., Tupek, M. R. and Dolbow, J. E. (2019), 'A phase-field formulation for dynamic cohesive fracture', *Computer Methods in Applied Mechanics and Engineering* vol. 348, 680–711.
URL: *https://www.sciencedirect.com/science/article/pii/S0045782519300519*

Geuzaine, C. and Remacle, J.-F. (2009), 'Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities', *International Journal for Numerical Methods in Engineering* vol. 79(11), 1309–1331.

Goswami, S., Anitescu, C. and Rabczuk, T. (2020), 'Adaptive fourth-order phase field analysis for brittle fracture', *Computer Methods in Applied Mechanics and Engineering* vol. 361, 112808.
URL: *https://www.sciencedirect.com/science/article/pii/S0045782519307005*

Heister, T., Wheeler, M. F. and Wick, T. (2015), 'A primal-dual active set method and predictor-corrector mesh adaptivity for computing fracture propagation using a phase-field approach', *Computer Methods in Applied Mechanics and Engineering* vol. 290, 466–495.

Hock-Chuan, C. (2018), 'Java Programming Tutorial java native interface (jni)', Available in: `https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html`. Acess in: 28/02/2021.

Hu, T., Guilleminot, J. and Dolbow, J. E. (2020), 'A phase-field model of fracture with frictionless contact and random fracture properties: Application to thin-film fracture and soil desiccation', *Computer Methods in Applied Mechanics and Engineering* vol. 368, 113106.
URL: *https://www.sciencedirect.com/science/article/pii/S0045782520302905*

Hu, Y. (2018), 'Using intel onemkl blas and lapack with petsc', Available in: `https://www.intel.com/content/www/us/en/developer/articles/technical/mkl-blas-lapack-with-petsc.html`. Acess in: 28/01/2022.

Karma, A., Kessler, D. A. and Levine, H. (2001), 'Numerical experiments in revisited brittle fracture', *Physics Review Letter* vol. 87:045501.

Kuhn, C., Schuter, A. and Müller, R. (2015), 'On degradation functions in phase field fracture models', *Computational Materials Science* vol. 108, 374–384.

Lancioni, G. and Royer-Carfagni, G. (2009), 'The variational approach to fracture mechanics', *Journal of Elasticity* vol. 95, 1–30.

Leão, H. M. (2021), A critical study on phase-field modeling of fracture, Master dissetation, UFMG - Unversidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil.

Li, T., Marigo, J.-J., Guilbaud, D. and Potapov, S. (2016), 'Gradient damage modeling of brittle fracture in an explicit dynamics context', *International Journal for Numerical Methods in Engineering* vol. 108.

Liang, S. (1999), *Java Native Interface: Programmer's Guide and Specification*, ADDISON-WESLEY.

Miehe, C., Hofacker, M. and Welschinger, F. (2010b), 'A phase field model for rate-independent crack propagation: Robust algorithmic implementation based on operator splits', *Computer Methods in Applied Mechanics and Engineering* vol. 199, 2765–2778.

Miehe, C., Welschinger, F. and Hofacker, M. (2010a), 'Thermodynamically consistent phase-field models of fracture: Variational principles and multi-field FE implementations', *International Journal for Numerical Methods in Engineering* vol. 83, 1273–1311.

Penna, S. S. (2007), Pós-processador para modelos bidimensionais não-lineares do método dos elementos finitos, Master dissetation, Universidade Federal de Minas Gerais (UFMG). (in portuguese).

Penna, S. S. (2011), Formulação multipotencial para modelos de degradação elástica: unificação teórica, proposta de novo modelo, implementação computacional e modelagem de estruturas de concreto, Phd thesis, Universidade Federal de Minas Gerais (UFMG). (in portuguese).

Petersson, P. E. (1981), 'Crack growth and development of fracture zones in plain concrete and similar materials', *Report TVBM* vol. 1006.

Pham, K., Amor, H., Marigo, J.-J. and Maurini, C. (2011), 'Gradient damage models and their use to approximate brittle fracture', *International Journal of Damage Mechanics* vol. 20, 618–652.

Upgrdman (2020), 'Introduction to jni with eclipse, gcc and msys2', Available in: `https://www.youtube.com/watch?v=uuN5qkNgrkY`. Acess in: 28/02/2021.

Winkler, B. (2001), Traglastuntersuchungen von unbewehrten und bewehrten Betonstrukturen auf der Grundlage eines objektiven Werkstoffgesetzes für Beton, PhD thesis, Universitat Innsbruck, Austria. (in german).

Wu, J.-Y. (2017), 'A unified phase-field theory for the mechanics of damage and quasi-brittle failure', *Journal of the Mechanics and Physics of Solids* vol. 103, 72–99.

Wu, J.-Y. (2018), 'Robust numerical implementation of non-standard phase-field damage models for failure in solids', *Computer Methods in Applied Mechanics and Engineering* vol. 340, 767–797.
URL: *https://www.sciencedirect.com/science/article/pii/S0045782518303050*

Wu, J.-Y., Huang, Y., Zhou, H. and Nguyen, V. P. (2021), 'Three-dimensional phase-field modeling of mode i + ii/iii failure in solids', *Computer Methods in Applied Mechanics and Engineering* vol. 373, 113537.
URL: *https://www.sciencedirect.com/science/article/pii/S0045782520307222*

Wu, J.-Y., Nguyen, V. P., Thanh Nguyen, C., Sutula, D., Bordas, S. and Sinaie, S. (2020), 'Chapter One - Phase-field modeling of fracture', *Advances in Applied Mechanics* vol. 53, 1–183.
URL: *https://www.sciencedirect.com/science/article/pii/S0065215619300134*

Appendices

Appendix A

JNI

In this appendix it will be demonstrated how to make a JNI in the user's own virtual machine. For *INSANE*, it was made using the bases shown in Azevedo (2019) and with what has already been done previously shown on the website `https://git.insane.dees.ufmg.br/insane/insane_dev_jni`. The material developed here was based on references Hock-Chuan (2018), bogotobogo (2020), Upgrdman (2020), Liang (1999), Azevedo (2019).

A.1 Basic

A.1.1 JNI - HelloWorld

The purpose of this section is to show how JNI was made with the C language, with basic example shown here. All steps in this subsection were done using the command terminal. To begin with, it will be made clear that what was developed here was using Linux distributed by Ubuntu. There is a way to do it for Microsoft Windows, check the references to see how.

First, the focus will be on JNI through the command line. A folder is created where the code will be made. It creates a Java file with the code you want. Here the example used is to print a "Hello World". To create a java file, create a ".txt" file and save it as ".java". You can do this using the command terminal on ubuntu via the command "gedit HelloJNI.java". The file name used here will be "HelloJNI", but it could be another one, you only need to pay attention to where it should be replaced in the next steps. The code written in the HelloJNI file is shown in Figure A.1.

```java
public class HelloJNI {  // Save as HelloJNI.java
   static {
      System.loadLibrary("hello"); // Load native library hello.dll (Windows) or libhello.so (Unixes)
                                   //  at runtime
                                   // This library contains a native method called sayHello()
   }

   // Declare an instance native method sayHello() which receives no parameter and returns void
   private native void sayHello();

   // Test Driver
   public static void main(String[] args) {
      new HelloJNI().sayHello();  // Create an instance and invoke the native method
   }
}
```

Figure A.1: HelloJNI java code. Adapted from Hock-Chuan (2018).

The static initializer invokes System.loadLibrary() to load the native library "hello" (which contains a native method called sayHello()) during the class loading. It will be mapped to "hello.dll" in Windows or "libhello.so" in Unixes/Mac OS X. Next, we declare the method sayHello() as a native instance method, via keyword native which denotes that this method is implemented in another language. A native method does not contain a body. The sayHello() shall be found in the native library loaded. The main() method allocates an instance of HelloJNI and invoke the native method sayHello().

Now the header file called HelloJNI.h is created. Starting from JDK 8, you should use "javac -h" to compile the Java program and generate C/C++ header file called HelloJNI.h as follows "javac -h . HelloJNI.java". The "-h dir" option generates C/C++ header and places it in the directory specified (in the above example, "." for the current directory). The generated header file will look like in the Figure A.2.

```c
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class HelloJNI */

#ifndef _Included_HelloJNI
#define _Included_HelloJNI
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     HelloJNI
 * Method:    sayHello
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_HelloJNI_sayHello
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

.

Figure A.2: HelloJNI header. Adapted from Hock-Chuan (2018).

Before JDK 8, you need to compile the Java program using 'javac" and generate the C/C ++ header using a dedicated "javah", for example, "javac HelloJNI.java" and

"javah HelloJNI". The javah utility is no longer available in JDK 10. The header declares a C function Java_HelloJNI_sayHello, for example, "JNIEXPORT void JNICALL Java_HelloJNI_sayHello(JNIEnv *, jobject);". The naming convention for the C function is Java_{package_and_classname}_{function_name}(JNI_arguments). The dot in package name is replaced by underscore. This naming issue will be further explored in Section A.1.2.1.The arguments are:

- JNIEnv*: reference to JNI environment, which lets you access all the JNI functions.
- jobject: reference to "this" Java object.

Now a file called "Hello.JNI.c" is created, which like java can be done by the command terminal. The code is described in Figure A.3.

```c
// Save as "HelloJNI.c"
#include <jni.h>        // JNI header provided by JDK
#include <stdio.h>      // C Standard IO Header
#include "HelloJNI.h"   // Generated

// Implementation of the native method sayHello()
JNIEXPORT void JNICALL Java_HelloJNI_sayHello(JNIEnv *env, jobject thisObj) {
   printf("Hello World!\n");
   return;
}
```

Figure A.3: HelloJNI C code. Adapted from Hock-Chuan (2018).´

Before proceeding, check that the JAVA_HOME environment is set. Compile the C program HelloJNI.c into share module "libhello.so" using gcc, which is included in all Unixess through "gcc -fPIC -I"$JAVA_HOME/include" -I"$JAVA_HOME/include/linux" -shared -o libhello.so HelloJNI.c". And finally type in the command terminal "java -Djava.library.path=. HelloJNI".

A.1.2 JNI - Eclipse

In this section we will talk about the JNI made by the Eclipse software. To start, the java code is created in eclipse. Then go to the folder where the ".java" code is located and do the same procedure mentioned earlier to generate the ".so" file, that is, generate the header file, also create the C code and finally generate the ".so", all this again through the terminal command.

Now right-click on the project (inside the eclipse) where the class that created the Java code is, click on Run As → Run Configurations. Go to Java Application, double-click to create a new one. In this new one, in the Main tab give the name with the name of the project and in Main Class, put the class that will have the JNI, like in Figure A.4. If you are going to use more than one class create a Java Application for each. Go to Arguments and place it in VM arguments and type "-Djava.library.path=src"(like in Figure A.5), where what is after the equals symbol represents the folder inside the HelloJNI project in

which ".so" is, in the case shown so far it should be in the "src" folder, since a package was not built, but the direct class.

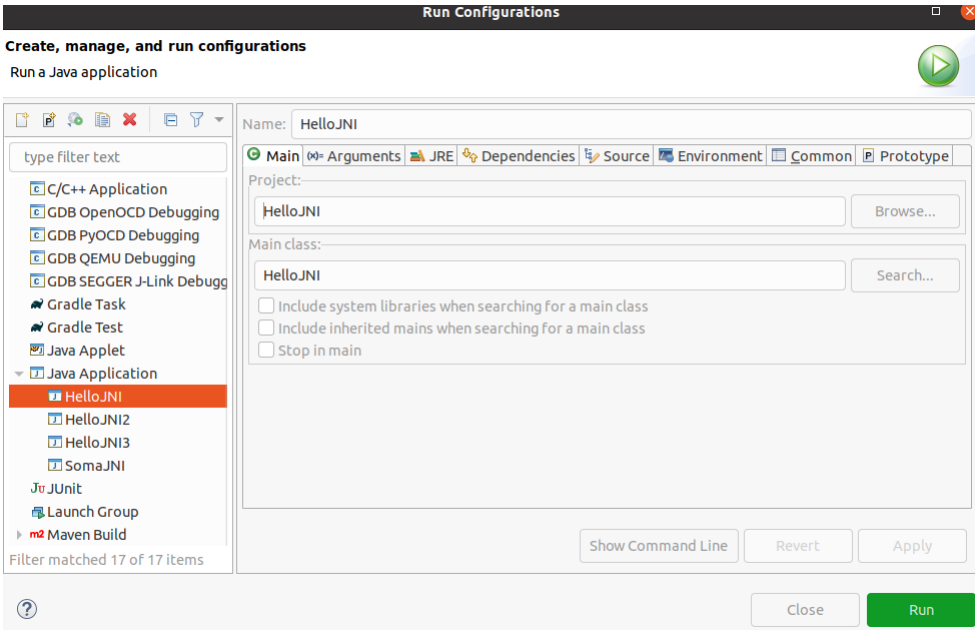


Figure A.4: HelloJNI in Java Application

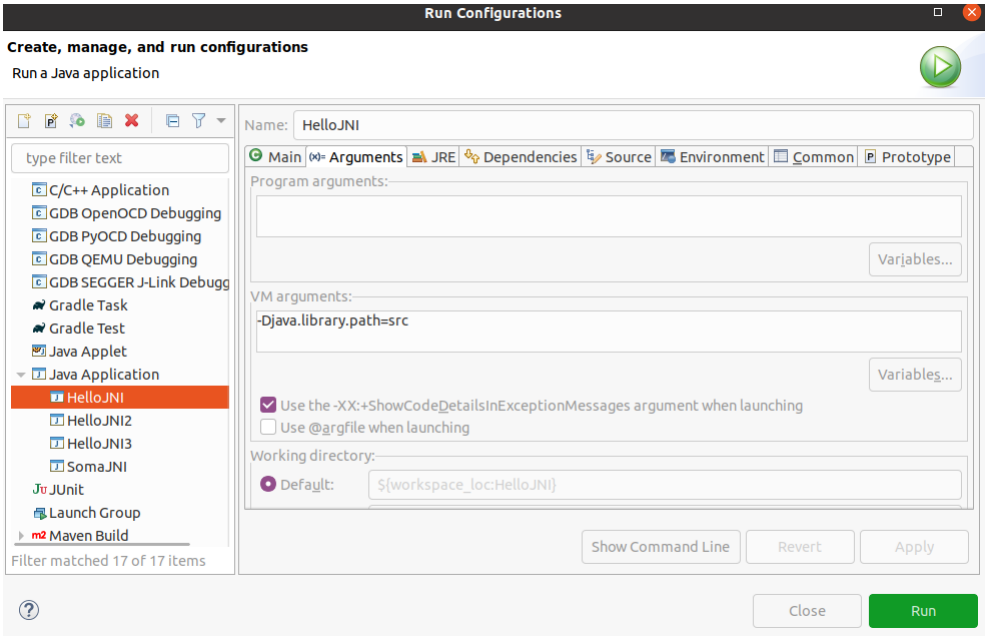


Figure A.5: Including a library in the Java library path via VM argument.

A.1.2.1 A class within a package

A package-free class was previously built. If done with a package, some care must be taken. Basically the same thing remains, but when giving the command to build the header file, that is, "javac -h",a file with the following name will have been built: "package_and_classname.h". This changes when importing the name of the header file into C

code, in addition to both the C file and the header after JNICALL you should pay attention to the naming convention for the C function is Java_{package_ and_classname} _{function_name}(JNI_arguments). For example, if the "HelloJNI" class is created inside a package called "HelloJNIpack" then the header file will be the same as described in the Figure A.6 and the C code as in the Figure A.7, pay attention to line 15 in header file and to line 5 in code C.

```
1 /* DO NOT EDIT THIS FILE - it is machine generated */
2 #include <jni.h>
3 /* Header for class HelloJNIpack_HelloJNI */
4
5 #ifndef _Included_HelloJNIpack_HelloJNI
6 #define _Included_HelloJNIpack_HelloJNI
7 #ifdef __cplusplus
8 extern "C" {
9 #endif
10 /*
11  * Class:     HelloJNIpack_HelloJNI
12  * Method:    sayHello
13  * Signature: ()V
14  */
15 JNIEXPORT void JNICALL Java_HelloJNIpack_HelloJNI_sayHello
16   (JNIEnv *, jobject);
17
18 #ifdef __cplusplus
19 }
20 #endif
21 #endif
```

Figure A.6: HelloJNI header with a java package.

```
1 #include <jni.h>
2 #include <stdio.h>
3 #include "HelloJNIpack_HelloJNI.h"
4
5 JNIEXPORT void JNICALL Java_HelloJNIpack_HelloJNI_sayHello(JNIEnv *env, jobject thisObj)
6 {
7    printf("Hello World!\n");
8    return;
9 }
```

Figure A.7: HelloJNI C code with a java package.

Another important detail is that in Java Application (in Run Configurations in Eclipse), in the Main tab, in Main class it is now "packagename.classaname", in our example it is "HelloJNIpack.HelloJNI".

A.1.2.2 Building everything through eclipse

It is possible to build the header file, the C code and the .so all by eclipse. To do this, download the CDT extension for eclipse, which is Eclipse Embedded C/C ++. To download go to Eclipse Marketplace in Help. After downloading, you can create a C code normally, for that look for manuals online. The focus here is on how to create the header and ".so" file. For that you need to create a makefile file, so first convert the project to

C/C ++. Right-click on the “HelloJNI” Java project → New → Other... → Convert to a C/C++ Project (Adds C/C++ Nature) → Next. The “Convert” to a C/C++ Project” dialog appears. In “Project type”, select ”Makefile Project” and in “Toolchains” select what you want. Now, it’s possible run this project as a Java as well as C/C++ project.

For to the makefile rigth-click on project → new → File → In “File name”, enter “makefile”. Through the makefile it is possible to create the header file and the “.so” file. For the header file, rigth-click on the project → Build Targets → Create → In “Target Name” enter with “header”. For the header enter the code in Figure A.8.

makefile

```
1 header:
2     javac   -h   .   src/HelloJNI.java
```

Figure A.8: Makefile for header.

After that build the target header double-clicking it or rigth-click on the project → Build Targets → Build and choose the target what you want. After that it is necessary to create a tagert to create the “.so”. Figure A.9 shows the target “all” for this case, with the same command to create the “.so” as described previously . For target “all” to work, the header file as well as the “.c” file need to be in the same folder for this makefile code to work, as shown in Figure A.10.

```
all:
    gcc -fPIC -I"$(JAVA_HOME)/include" -I"$(JAVA_HOME)/include/linux" -shared -o libhello.so HelloJNI.c
```

Figure A.9: Makefile for “.so”.

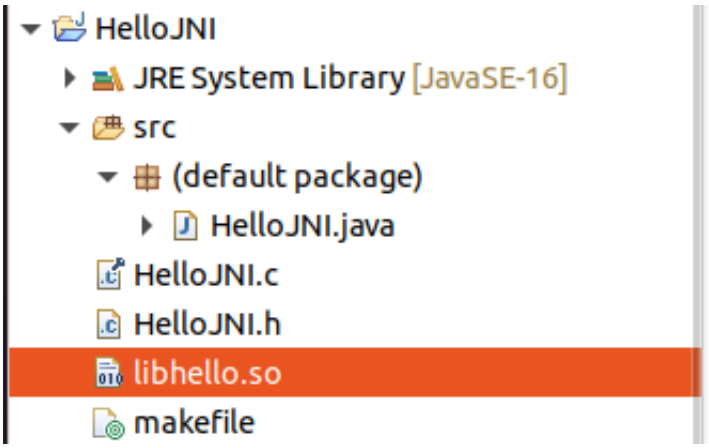


Figure A.10: Examples organization scheme in eclipse

Obviously the user can reorganize and structure the files in the best way after generating the “.so”, because the objective here is to demonstrate how to do the whole process. In the case of figure A.10 the path in the Java application (in Run Configuration) would

be just "-Djava.library.path=.". Another point to be highlighted is that you can optimize everything through the makefile with commands copy and paste files from one folder to another or delete files. An example would be the adads command "cp -f HelloJNI.h destiny" that copies a "HelloJNI.h" file from the folder where the makefile is to a folder called "destiny" inside the folder where the makefile is. This goes for the Linux operating system.

A.1.3 JNI - Correction of primitive variables

When transferring primitive variables from Java to C you need to convert them into C code. Table A.1 shows some conversions.

Table A.1: Conversion of primitive variables

Java Primitive	C Primitive
byte	jbyte
char	jchart
short	jshort
int	jint
long	jlong
float	jfloat
double	jdouble

In the case of Arrays or Strings, the transmission from Java to C, or C++, needs to be done through special functions. These are found through the JNIenv pointer and declared in the jni.h header, located in the include folder of the JDK (Java Development Kit), and are GetStringUTFChars, GetIntArrayElements, GetDoubleArrayElements, GetObjectArrayElements, etc (Azevedo, 2019).

After using the Arrays and Strings, or if necessary changing, at the end of the function, when these elements are no longer needed, they must be released from memory, for use in Java, with the methods ReleaseStringUTFChars, ReleaseIntArrayElements, ReleaseDoubleArrayElements, ReleaseObjectArrayElements, etc. If this procedure is not performed, a memory leak occurs (Azevedo, 2019).

Figure A.11 shows a Java code sending an array of doubles and a double variable to the C code. Figure A.12 shows the header of this class in Java. Figure A.13 shows how to receive the variables in C as well as the treatment that should be done for arrays. It is worth noting that when generate the header file by the Java compiler these variables in C are already created in the method signature.

```java
public class ExampleClass {
        static {
          System.loadLibrary("biblio");
        }

        private native double[] ExampleMethod(double[] ai, double bi);

        public static void main(String[] args) {

           double[] ci, answer;
           double di;

           ci = new double[3];
           answer = new double[3];
           ci[0] = 7;
           ci[1] = 8;
           ci[2] = 9;
           di = 5;

           answer = new ExampleClass().ExampleMethod(ci,di);

        }

}
```

Figure A.11: Java code for correction of primitive variables

```c
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class ExamplePack_ExampleClass */

#ifndef _Included_ExamplePack_ExampleClass
#define _Included_ExamplePack_ExampleClass
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     ExamplePack_ExampleClass
 * Method:    ExampleMethod
 * Signature: ([DD)[D
 */
JNIEXPORT jdoubleArray JNICALL Java_ExamplePack_ExampleClass_ExampleMethod
  (JNIEnv *, jobject, jdoubleArray, jdouble);

#ifdef __cplusplus
}
#endif
#endif
```

Figure A.12: Header for correction of primitive variables

```
JNIEXPORT jdoubleArray JNICALL Java_ExamplePack_ExampleClass_ExampleMethod(
        JNIEnv* env, jobject thisObj, jdoubleArray fi, jdouble gi)
{

    // Preparing Java Arguments
    jdouble *Li = (*env)->GetDoubleArrayElements(env, fi, 0);
    int length = (*env)->GetArrayLength(env, fi);

    double Ki[3];

    Ki[0] = gi*Li[0];
    Ki[1] = gi*Li[1];
    Ki[2] = gi*Li[2];

    // Releasing Java Variables
    (*env)->ReleaseDoubleArrayElements(env, fi, Li, 0);

    // Setting the array Ki, to be passed to Java
    jdoubleArray Ji = (*env)->NewDoubleArray(env, length);
    (*env)->SetDoubleArrayRegion(env, Ji, 0, length, Ki);

    return Ji;
}
```

Figure A.13: C code for correction of primitive variables

It is worth mentioning that when passing the matrix from *INSANE* to PETSc, it must be passed sparsely, because when using "jobjectArray" it ends up giving an error, so it is best to use a set of "jdoubleArray" when passing matrix data.

A.2 PETSc

For JNI with PETSc you need some special details. As previously mentioned thePETSc was build with with Intel® oneAPI Math Kernel Library (oneMKL) BLAS and LAPACK, so check the PETSc installation manual as well as reference Hu (2018). Due to MKL and an error loading a library from Java itself, which is the library "libjsig.so", you need to do a preload before the program will process. For that you need to set Run Configurations in eclipse, in Java Application, in the application where you have the external library, after that in Environment. In Enviroment set the "LD_PRELOAD" in variable and in value put the path to the external libraries of mkl and "libjsig.so" that are not loaded correctly. Figure A.14 shows how this preloading is done and Figure A.15 shows the text in Value. It is worth noting that the path shown in Figure A.15 varies from computer to computer.

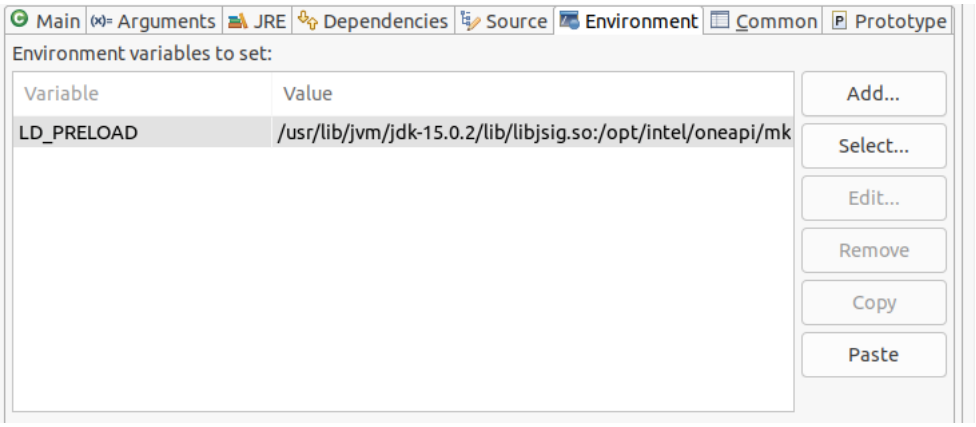


Figure A.14: Example how to set LD_PRELOAD

```
/usr/lib/jvm/jdk-15.0.2/lib/libjsig.so:/opt/intel/oneapi/mkl/-
2021.3.0/lib/intel64/libmkl_def.so.1:/opt/intel/oneapi/mkl/2021.3.0/-
lib/intel64/libmkl_avx2.so.1:/opt/intel/oneapi/mkl/2021.3.0/lib/-
intel64/libmkl_core.so:/opt/intel/oneapi/mkl/2021.3.0/lib/intel64/-
libmkl_intel_lp64.so:/opt/intel/oneapi/mkl/2021.3.0/lib/intel64/-
libmkl_intel_thread.so:/opt/intel/oneapi/compiler/2021.3.0/linux/-
compiler/lib/intel64/libiomp5.so
```

Figure A.15: Value for LD_PRELOAD

To create the makefile for PETSc is more complicated. After installing PETSc, create a code in C that uses PETSc or process one of the examples that already comes within it. When using the make command in the command terminal to generate the code executable, the code from this processing appear in the command terminal. This is code to use for the makefile. For example, take example 9 from PETSc in the tutorials inside snes folder (folders and examples that come inside the library). When using the "make" command, the code shown in Figure A.16 appears. After that, it adapts this code to generate the makefile as shown in Figure A.17. It is worth noting again here that there are computer-dependent paths, so the user needs to adapt.

```
matheus@matheus-Inspiron-3443:~/petsc/src/snes/tutorials$ make ex9
mpicc -fPIC -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fst
ack-protector -fvisibility=hidden -g3  -fPIC -Wall -Wwrite-strings -Wno-strict-a
liasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -g3    -I/hom
e/matheus/petsc/include -I/home/matheus/petsc/arch-linux-c-debug/include -I/opt/
intel/oneapi/mkl/2021.3.0/lib/intel64/../../include     ex9.c  -Wl,-rpath,/home/
matheus/petsc/arch-linux-c-debug/lib -L/home/matheus/petsc/arch-linux-c-debug/li
b -Wl,-rpath,/opt/intel/oneapi/mkl/2021.3.0/lib/intel64 -L/opt/intel/oneapi/mkl/
2021.3.0/lib/intel64 -Wl,-rpath,/usr/lib/x86_64-linux-gnu/openmpi/lib -L/usr/lib
/x86_64-linux-gnu/openmpi/lib -Wl,-rpath,/usr/lib/gcc/x86_64-linux-gnu/9 -L/usr/
lib/gcc/x86_64-linux-gnu/9 -Wl,-rpath,/usr/lib/x86_64-linux-gnu -L/usr/lib/x86_6
4-linux-gnu -Wl,-rpath,/lib/x86_64-linux-gnu -L/lib/x86_64-linux-gnu -lpetsc -lm
kl_intel_lp64 -lmkl_core -lmkl_sequential -lpthread -lm -lstdc++ -ldl -lmpi_usem
pif08 -lmpi_usempi_ignore_tkr -lmpi_mpifh -lmpi -lgfortran -lm -lgfortran -lm -l
gcc_s -lquadmath -lpthread -lquadmath -lstdc++ -ldl -o ex9
```

Figure A.16: Code generated by "make" command in command terminal

```
mpicc   -fPIC   -I/usr/lib/jvm/java-16-openjdk-amd64/include    -I/usr/lib/jvm/java-16-openjdk-amd64/include/linux  \
-Wall   -Wwrite-strings -Wno-strict-aliasing    -Wno-unknown-pragmas    -fstack-protector   \
-fvisibility=hidden -g3 -fPIC   -Wall   -Wwrite-strings -Wno-strict-aliasing    \
-Wno-unknown-pragmas    -fstack-protector   -fvisibility=hidden -g3 \
-I/home/matheus/petsc/include   -I/home/matheus/petsc/arch-linux-c-debug/include    \
-shared -o  libinsanepetsc.so   insanepetsc.c   \
-Wl,-rpath,/home/matheus/petsc/arch-linux-c-debug/lib   \
-L/home/matheus/petsc/arch-linux-c-debug/lib    \
-Wl,-rpath,/opt/intel/oneapi/mkl/2021.3.0/lib/intel64   \
-L/opt/intel/oneapi/mkl/2021.3.0/lib/intel64    \
-Wl,-rpath,/usr/lib/x86_64-linux-gnu/openmpi/lib -L/usr/lib/x86_64-linux-gnu/openmpi/lib    \
-Wl,-rpath,/usr/lib/gcc/x86_64-linux-gnu/9  \
-L/usr/lib/gcc/x86_64-linux-gnu/9   -Wl,-rpath,/usr/lib/x86_64-linux-gnu    \
-L/usr/lib/x86_64-linux-gnu -Wl,-rpath,/lib/x86_64-linux-gnu    \
-L/lib/x86_64-linux-gnu -lpetsc -lmkl_intel_lp64 -lmkl_core -lmkl_sequential    \
-lpthread -lm -lstdc++  \
-ldl    -lmpi_usempif08 -lmpi_usempi_ignore_tkr -lmpi_mpifh \
-lmpi -lgfortran -lm -lgfortran -lm -lgcc_s -lquadmath  -lpthread -lquadmath    -lstdc++ -ldl
```

Figure A.17: Makefile example for PETSc

The primary difference from Figure A.16 to Figure A.17 for building the makefile is the substitution of “ex9.c” for “-shared -o libinsanepetsc.so insanepetsc.c”. When creating your example’s makefile, ensure that it gets the functions it needs in PETSc, that is, when using the “make” command strategy, get a file that has the function you need. Other examples for JNI are found on the website `https://git.insane.dees.ufmg.br/insane/insane_dev_jni`.

Appendix B

Demonstration of Equation 2.46

As stated in Section 2.4.1.4, it started from the following relationship:

$$G_c = \int_0^{w_c} \sigma(w)dw = \int_0^{w_c} f_t[(1+\eta_1^3 r^3)\exp(-\eta_2 r) - r(1+\eta_1^3)\exp(-\eta_2)]dw \tag{B.1}$$

Using mathematical software such as Maple, the following equation can be arrived at:

$$\begin{aligned} G_c = \frac{1}{2} w_c f_t \exp(-\eta_2)\frac{-\eta_1^3\eta_2^4 - 2\eta_1^3\eta_2^3 - 6\eta_1^3\eta_2^2 + 12\exp(\eta_2)\eta_1^3}{\eta_2^4} + \\ \frac{1}{2} w_c f_t \exp(-\eta_2)\frac{2\exp(\eta_2)\eta_2^3 - 12\eta_1^3\eta_2 - \eta_2^4 - 12\eta_1^3 - \eta_2^3}{\eta_2^4} \end{aligned} \tag{B.2}$$

Rearranging we have the following development:

$$\begin{aligned} G_c = \frac{1}{2} w_c f_t \exp(-\eta_2)\frac{-\eta_1^3\eta_2^4 - 2\eta_1^3\eta_2^3 - \eta_2^4 - \eta_2^3}{\eta_2^4} + \\ \frac{1}{2} w_c f_t \exp(-\eta_2)\frac{-6\eta_1^3\eta_2^2 - 12\eta_1^3\eta_2 - 12\eta_1^3}{\eta_2^4} + \\ \frac{1}{2} w_c f_t \exp(-\eta_2)\frac{(12\exp(\eta_2)\eta_1^3 + 2\exp(\eta_2)\eta_2^3)}{\eta_2^4} \end{aligned} \tag{B.3}$$

$$\begin{aligned} \frac{G_c}{w_c f_t} = -\exp(-\eta_2)(\frac{\eta_1^3}{2} + \frac{\eta_1^3}{\eta_2} + \frac{1}{2} + \frac{1}{\eta_2}) + \\ -\exp(-\eta_2)(\frac{3\eta_1^3}{\eta_2^2} + \frac{6\eta_1^3}{\eta_2^3} + \frac{6\eta_1^3}{\eta_2^4}) + \\ 6\frac{\eta_1^3}{\eta_2^4} + \frac{1}{\eta_2} \end{aligned} \tag{B.4}$$

$$\frac{G_c}{w_c f_t} = -\exp(-\eta_2)(1+\eta_1^3)(\frac{1}{2}+\frac{1}{\eta_2}) - \exp(-\eta_2)\frac{3\eta_1^3}{\eta_2^2}(1+\frac{2}{\eta_2}+\frac{2}{\eta_2^2}) + 6\frac{\eta_1^3}{\eta_2^4}+\frac{1}{\eta_2} \tag{B.5}$$

$$\frac{G_c}{w_c f_t} = \eta_w \tag{B.6}$$

$$w_c = \frac{G_c}{f_t \eta_w} \tag{B.7}$$

Equation B.7 is Equation 2.46 that we wanted to demonstrate. If we substitute the values for normal concrete ($\eta_1 = 3$, $\eta_2 = 6.93$) obtained by Cornelissen et al. (1986), we have Equation B.8 which is the same used by Wu (2017).

$$w_c = 5.1361\frac{G_c}{f_t} \tag{B.8}$$