Phase field fracture implementation in FEniCS

Emilio Martínez-Pañeda[a,*], Hirshikesh[b]

[a]*Department of Engineering, Cambridge University, CB2 1PZ Cambridge, UK*
[b]*Department of Mechanical Engineering, Indian Institute of Technology Madras, Chennai - 600036, India*

Abstract

Documentation that accompanies the Python script PhaseField.py for implementing the phase field model for fracture in FEniCS. An extension for modelling crack growth in Functionally Graded Materials is also provided. If using this code for research or industrial purposes, please cite:

Hirshikesh, S. Natarajan, R. K. Annabattula, E. Martínez-Pañeda. Phase field modelling of crack propagation in functionally graded materials. Composites Part B: Engineering 169, pp. 239-248 (2019)

Keywords:
FEniCS, Phase Field, Fracture, Crack growth, Finite element analysis

1. Introduction

The phase field model for fracture builds upon the pioneering thermodynamic framework established by Griffith, where crack growth will take place if a critical energy release rate is attained. Frankfort and Marigo [1] were the first to embed Griffith's approach into variational formulations and Bourdin et al. [2, 3] later regularized the discrete crack topology by means of a scalar damage variable and a diffuse crack representation. This variable is termed as the phase field, or phase field order parameter. Important contributions to the model have also been made by Miehe and co-workers [4, 5]. Due to its robustness, the phase field fracture model enjoys great popularity and has

*Corresponding author.
Email address: mail@empaneda.com (Emilio Martínez-Pañeda)

not only been successfully applied to model brittle fracture but also to ductile damage [6, 7], hydraulic fracturing [8, 9], composites damage [10, 11], and hydrogen assisted cracking [12], to name a few. We provide an efficient and robust implementation of the phase field method in the open source finite element package FEniCS [13], enabling to model interactions and branching of cracks of arbitrary topological complexity.

The structure of this document is organized as follows. Section 2 includes an introduction to the phase field fracture method. The weak form and finite element implementation is described in Section 3. Section 4 deals with the usage and verification of the Python script for FEniCS provided. Finally, two appendices are included: one with a line-by-line description of the code for the benefit of those new to FEniCS, and a second one with details of the extension to Functionally Graded Materials (FGMs).

2. The phase field method for fracture

Consider a linear elasto-static body with a discontinuity occupying the domain $\Omega \subset \mathbb{R}^d$, where $d = 2, 3$ as shown in Fig. 1. The boundary Γ with outward normal $\boldsymbol{n}$ is considered to admit decomposition into two disjoint sets Γ_D and Γ_t where Dirichlet and Neumann boundary conditions are respectively specified. The closure of the domain is $\overline{\Omega} \equiv \Omega \cup \Gamma$. In a discrete fracture mechanics context, the strong discontinuity (i.e. the crack) is represented by a discontinuous surface Γ_c, as shown in Fig. 1a. Whereas within the framework of the phase field fracture method, the crack is modeled by a diffuse field variable $\phi \in [0, 1]$ as shown in Fig. 1b. Here, $\phi = 0$ denotes intact material while $\phi = 1$ represents the fully broken material state. The size of the regularized crack surface is governed by the choice of ℓ, the model-inherent length scale.

As shown by Γ-convergence [14], a regularized crack density functional $\Gamma_\ell(\ell, \phi)$ can be defined that converges to the functional of the discrete crack as $\ell \to 0$. Hence, the fracture energy due to the creation of a crack can be approximated as

$$\int_{\Gamma_c} G_c \, \mathrm{d}\Gamma_c \approx \int_\Omega G_c \, \Gamma_\ell(\ell, \phi) \, \mathrm{d}\Omega = \int_\Omega G_c \left(\frac{1}{2\ell}\phi^2 + \frac{\ell}{2} |\nabla \phi|^2 \right) \mathrm{d}\Omega \tag{1}$$

The approximated crack surface energy (1) can be added to the bulk energy to form the total potential energy of the solid Ψ as

$$\Psi = \int_{\Omega} \left((1-\phi)^2 \, \psi(\boldsymbol{\varepsilon}) + G_c \left(\frac{1}{2\ell}\phi^2 + \frac{\ell}{2}|\nabla\phi|^2 \right) \right) \mathrm{d}\Omega \tag{2}$$

where the term $(1-\phi)^2$ describes the degradation of the stored energy with evolving damage.

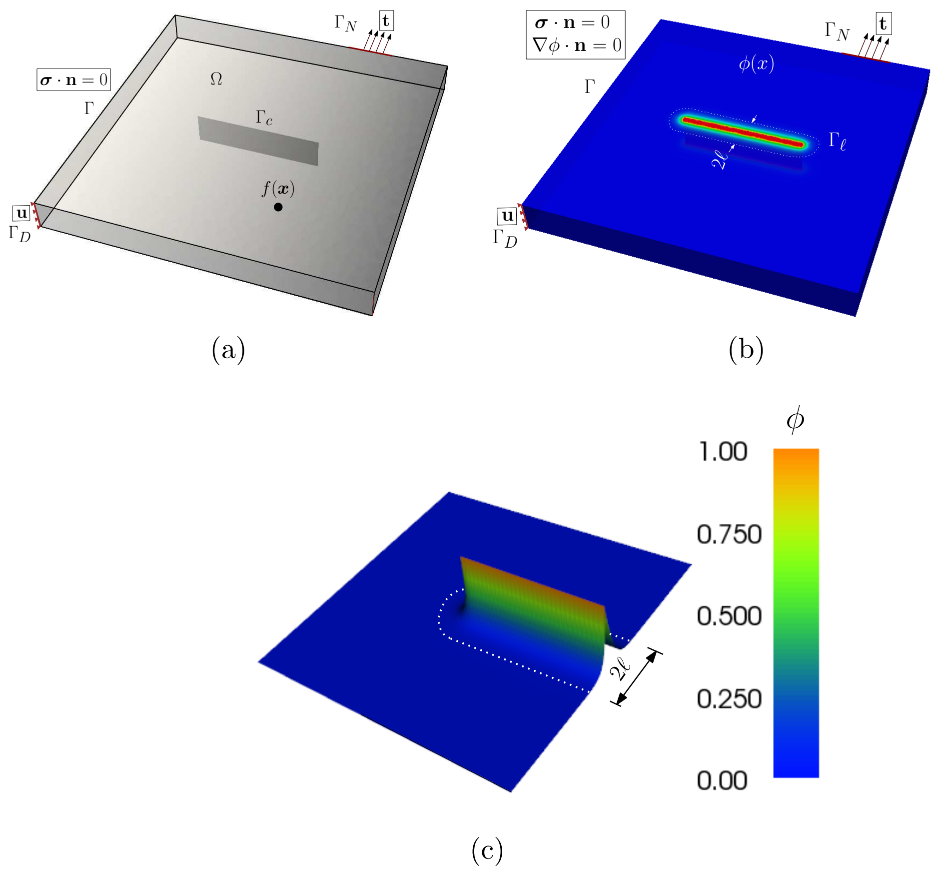

Figure 1: Schematic representation of a domain with a geometric discontinuity: (a) discrete representation, (b) diffuse representation based on the phase field approach, and (c) two-dimensional approximation of the phase field parameter ϕ.

The strain energy density for the undamaged solid is given in terms of the strain field $\boldsymbol{\varepsilon}$ and Lame's parameters λ and μ as

$$\psi(\boldsymbol{\varepsilon}) = \frac{1}{2}\lambda \left(\mathrm{tr}\left(\boldsymbol{\varepsilon}\right)\right)^2 + \mu \, \left(\boldsymbol{\varepsilon} : \boldsymbol{\varepsilon}\right) = \frac{1}{2}\lambda \left(\mathrm{tr}\left(\boldsymbol{\varepsilon}\right)\right)^2 + \mu \mathrm{tr}\left(\boldsymbol{\varepsilon}^2\right) \tag{3}$$

with the strain tensor being related to the displacement field in the usual manner: $\boldsymbol{\varepsilon} = \mathrm{sym}\nabla\boldsymbol{u}$. Upon taking the first variation of (2) and applying Gauss theorem, the following coupled field equations are obtained for any arbitrary value of the kinematic variables $\delta\boldsymbol{u}$ and $\delta\phi$,

$$\begin{aligned} (1-\phi)^2 \, \nabla \cdot \boldsymbol{\sigma} = \mathbf{0} \quad &\text{in} \quad \Omega \\ G_c \left(\frac{1}{\ell}\phi - \ell \Delta \phi \right) - 2(1-\phi) \, \psi\left(\boldsymbol{\varepsilon}\right) = 0 \quad &\text{in} \quad \Omega \end{aligned} \tag{4}$$

Here, $\boldsymbol{\sigma}$ denotes the Cauchy stress tensor. For a traction $\boldsymbol{T}$, the natural boundary conditions readily follow as

$$\begin{aligned} (1-\phi)^2 \boldsymbol{\sigma} \cdot \boldsymbol{n} =& \boldsymbol{T} \quad \text{on} \quad \Gamma \\ \nabla \phi \cdot \boldsymbol{n} =& 0 \quad \text{on} \quad \Gamma \end{aligned} \tag{5}$$

3. Finite element implementation

The finite element method is used to solved the coupled system of equations (4). We follow the hybrid model by Ambati et al. [15]; to maintain resistance in compression and during crack closure we reformulate (4b) as

$$G_c \left(\frac{1}{\ell}\phi - \ell \Delta \phi \right) - 2(1-\phi) H^+(\boldsymbol{\varepsilon}) = 0 \tag{6}$$

where H^+ is the so-called history variable field, which is defined as

$$H^+ = \max_{t\in[0,\tau]} \psi^+(\boldsymbol{\varepsilon}(t)) \tag{7}$$

with ψ^+ being given by

$$\psi^+(\boldsymbol{\varepsilon}) = \frac{1}{2} K \langle \mathrm{tr}(\boldsymbol{\varepsilon}) \rangle_+^2 + \mu(\boldsymbol{\varepsilon}^{dev} : \boldsymbol{\varepsilon}^{dev}) \tag{8}$$

following Amor et al. [16]. Here, $\langle a \rangle_+ = \frac{1}{2}(a+|a|)$ and K is the bulk modulus.

The resulting weak form can be obtained by considering the dimensional trial $(\mathscr{U}, \mathscr{P})$ and test spaces $(\mathscr{V}, \mathscr{Q})$. Let $\mathscr{W}(\Omega)$ include the linear displacement field and phase field variable:

$$(\mathscr{U}, \mathscr{V}) = \left\{ (\boldsymbol{u}, \boldsymbol{v}) \in [C^0(\Omega)]^d : (\boldsymbol{u}, \boldsymbol{v}) \in [\mathcal{W}(\Omega)]^d \subseteq [H^1(\Omega)]^d \right\} \tag{9a}$$

$$(\mathscr{P}, \mathscr{Q}) = \left\{ (\phi, q) \in [C^0(\Omega)]^d : (\phi, q) \in [\mathcal{W}(\Omega)]^d \subseteq [H^1(\Omega)]^d \right\} \tag{9b}$$

The system of equations can be readily obtained upon applying the standard Bubnov-Galerkin procedure. In the absence of remote tractions and body forces, one can find $\boldsymbol{u} \in \mathscr{U}$ & $\phi \in \mathscr{P}$, for all $\boldsymbol{v} \in \mathscr{V}$ & $q \in \mathscr{Q}$, by solving

$$\begin{aligned} \int_\Omega \left\{ (1-\phi)^2 \boldsymbol{\sigma}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{v}) \right\} \mathrm{d}\Omega &= 0 \\ \int_\Omega \left\{ \nabla q \cdot \nabla \phi \, G_c \, \ell + q \left(\frac{G_c}{\ell} + 2H^+ \right) \phi - 2H^+ q \right\} \mathrm{d}\Omega &= 0 \end{aligned} \tag{10}$$

The system is solved by means of a staggered approach. Note that, taking advantage of the symbolic differentiation capabilities of FEniCS, there is no need to derive and discretize the residuals and the consistent stiffness matrix; the system (10) is the only information provided (see Section 4 and Appendix A).

4. Usage instructions and verification

The main usage steps are outlined, taking as example the paradigmatic benchmark of a plate subjected to uniaxial tension, see Fig. 2a. A line-by-line description of the code used in the present example is given in Appendix A. For details on the installation of FEniCS see https://fenicsproject.org/.

First, the mesh is built. There are several options for this. One is to create the mesh using the open-source package Gmsh [17], save it as .msh and then use the `dolfin-convert` options to create an .xml file that can be read in our python script with the FEniCS command `Mesh`. An alternative route is to create the mesh with any meshing package (e.g., Abaqus) and then convert the mesh to .xdmf using the meshio package.

The material properties are then manually introduced in the script. Here, we will verify our results with those obtained by Miehe et al. [5] for a plate with Young's Modulus $E = 210$ GPa, Poisson's ratio $\nu = 0.3$, critical energy release rate $G_c = 2.7$ MPa mm and two different values of ℓ.

We then define the boundary conditions. That involves identifying the boundaries and prescribing the appropriate Dirichlet boundary conditions on $\boldsymbol{u}$ and ϕ. On the displacemente side, as in [5], we clamp the bottom of the plate and prescribe a remote displacement u_r in the vertical direction. The magnitude of the remote displacement is assigned to the variable `u_r` and equals 0.007 mm. Regarding ϕ, we prescribe $\phi = 1$ along the initial crack path.

Finally, we request the necessary output. For this particular example we are interested in the force versus displacement curve, see Fig. 2b. A very good agreement with the results obtained by Miehe et al. [5] is observed, validating the present numerical implementation. In addition, the crack contour is obtained by printing information in an .pvd/.vtu file, to be read with Paraview [18].

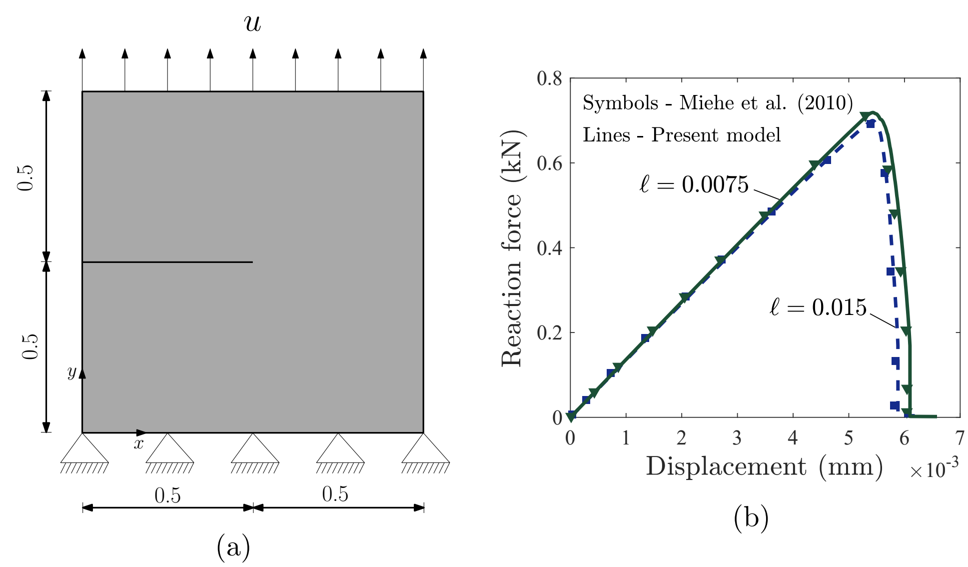

Figure 2: Verification example: (a) geometry and boundary conditions, (b) load versus displacement curve, comparison with Miehe et al. [5]. All dimensions are given in mm.

The code prints, at the end of each load step, the number of iterations required to achieve convergence and the percentage of the computation performed, based on the value of the remote displacement. Other FEniCS-related information is also provided, such as calls to the linear solver or warnings regarding the use of the error norm to compare two numerical solutions (as opposed to a numerical and an exact solution). These can be deactivated by adding the line `set_log_active(False)`. Note that, for illustrative purposes, a simple, Newton-Raphson like algorithm has been implemented. One can improve the scheme in many ways, but the enhancements that we have tested (computing residuals, requiring a minimum of 2 iterations per load increment, etc.) did not bring any changes to the results. Most load steps converge in a few iterations, but a couple of load steps (in the area where the load drops drastically) require numerous iterations to converge. The file provided, `PhaseField.py`, will typically complete in less than 2 hours (on a single core) and reproduce the result by Miehe et al. [5]. Nevertheless, to conduct rapid tests another file is provided with a coarser time stepping (`PhaseFieldCoarse.py`). It should take minutes to run and provide a result that is not so far away from the precise one.

4.1. 3D case studies

One of the main advantages of implementing the code in FEniCS is the little development time that extensions such as 3D require. A file named `PhaseField3D.py` is also attached, where the edge crack problem outlined in the previous section is solved in a 3D setting. As it can be readily observed by comparing `PhaseField3D.py` with `PhaseField.py`, the codes differ on a single line: the application of the boundary conditions. Specifically, we not only clamp u_x and u_y at the bottom but also u_z.

Given the larger mesh size, the 3D case will benefit of running in parallel. From the terminal, the following command should do the job:

```
mpirun np 4 python3 PhaseField.py
```

5. Conclusions

We have provided a robust implementation of the phase field fracture method for the open source finite element package FEniCS. As discrete methods (see, e.g., [19]), the phase field fracture method requires a refined mesh along the potential crack propagation path to resolve the fracture process zone. However, by decoupling the damage and displacement variables the method enables overcoming snap-back phenomena without the need of control algorithms [20, 21]. Also, as shown in our journal publication [22], the phase field method is well-suited to deal with arbitrary crack propagation paths. In summary, the method holds promise and we hope that the present implementation will facilitate research in this field.

6. Acknowledgements

E. Martínez-Pañeda acknowledges financial support from the Royal Commission for the 1851 Exhibition through their Research Fellowship programme (RF496/2018).

Appendix A. Line-by-line description

A line-by-line description of the script PhaseField.py follows. Efforts have been placed to make the code as simple and concise as possible; improvement suggestions in these two fronts are welcomed. The script currently has 76 lines of code.

```
from dolfin import *
```

This imports the dolfin python package, on which FEniCS heavily relies. It will be always needed.

```
mesh = Mesh('mesh.xml')
```

The variable mesh stores the .xml finite element mesh created with Gmsh (mesh.msh) and converted with dolfin-convert.

```
V = FunctionSpace(mesh, 'CG', 1)
```

We define a discrete function space V over the mesh for a scalar field ϕ. The discretization employs first-order Lagrangian finite elements (CG stands for Continuous Galerkin); see the Dolfin/FEniCS documentation for the FunctionSpace class.

```
W = VectorFunctionSpace(mesh, 'CG', 1)
```

We define a discrete function space W over the mesh for the vector field $\boldsymbol{u}$, with the same finite element discretization as for ϕ.

```
WW = FunctionSpace(mesh, 'DG', 0)
```

We need to define a discrete function space WW over the mesh to project the scalar field H^+. Since we use linear triangular elements, we employ discontinuous Lagrange elements with degree 0 to obtain the element value of H^+.

```
p, q = TrialFunction(V), TestFunction(V)
```

We define a trial function, ϕ, and a test function, $\delta\phi$ or q, in the scalar function space V.

```
u, v = TrialFunction(W), TestFunction(W)
```

We define a trial function, $\boldsymbol{u}$, and a test function, $\delta\boldsymbol{u}$ or $\boldsymbol{v}$, in the scalar function space V.

```
Gc =  2.7
```

We assign a value of 2.7 MPa mm to the critical energy release rate G_c.

```
l = 0.015
```

We assign a value of 0.015 mm to the phase field length scale ℓ.

```
lmbda = 121.1538e3
```

We assign a value of 121153.8 MPa to Lame's first parameter λ, as computed from the Young's modulus and Poisson's ratio $\lambda = E\nu/((1+\nu)(1-2\nu))$. Note that one cannot use the variable `lambda`, as it is used for other purposes in Dolfin.

```
mu = 80.7692e3
```

We assign a value of 80769.2 MPa to the shear modulus μ, as computed from the Young's modulus and Poisson's ratio $\mu = E/(2(1+\nu))$.

```
def epsilon(u):
    return sym(grad(u))
```

We define the strains as a function of the displacement field, $\boldsymbol{\varepsilon} = \mathrm{sym}\nabla\boldsymbol{u}$.

```
def sigma(u):
    return 2.0*mu*epsilon(u)+lmbda*tr(epsilon(u))*Identity(len(u))
```

We define the stresses as a function of the strains, as given by Hooke's law $\boldsymbol{\sigma} = 2\mu\,\boldsymbol{\varepsilon}(\boldsymbol{u}) + \lambda\,\mathrm{tr}(\boldsymbol{\varepsilon}(\boldsymbol{u}))\,I$. The in-built variable `Identity(2)` of Dolfin gives the 2D identity matrix (I).

```
def psi(u):
    return 0.5*(lmbda+mu)*(0.5*(tr(epsilon(u))+abs(tr(epsilon(u)))))**2+\
mu*inner(dev(epsilon(u),dev(epsilon(u))
```

We define the strain energy density ψ^+, as given by Eq. (8).

```
def H(uold,unew,Hold):
    return conditional(lt(psi(uold),psi(unew)),psi(unew),Hold)
```

We enforce irreversibility (Kuhn-Tucker conditions) by using the conditional operator. If $(\psi^+)_{t-1} < (\psi^+)_t$ then $H^+ = (\psi^+)_t$, otherwise, $H^+ = (\psi^+)_{t-1}$.

```
top = CompiledSubDomain("near(x[1], 0.5) && on_boundary")
```

We define a new domain for the top boundary, which is located at $y = 0.5$. The function `near(value1,value2)` checks that `value1` is within machine precision of `value2`.

```
bot = CompiledSubDomain("near(x[1], -0.5) && on_boundary")
```

We define a new domain for the bottom boundary, which is located at $y = -0.5$.

```
def Crack(x):
    return abs(x[1]) < 1e-03 and x[0] <= 0.0
```

We define the crack to latter assign $\phi = 1$ over the initial crack length.

```
load = Expression("t", t = 0.0, degree=1)
```

We define a user expression to prescribe the load that increases with time.

```
bcbot= DirichletBC(W, Constant((0.0,0.0)), bot)
```

We use the `DirichletBC` class to prescribe both components of $\boldsymbol{u}$ in the bottom of the plate.

```
bctop = DirichletBC(W.sub(1), load, top)
```

We prescribe the displacement at the top of the plate. By defining the function space `W.sub(1)` we indicate that the boundary condition applies only to the 2nd degree of freedom (u_y).

```
bc_u = [bcbot, bctop]
```

We group the displacement-related boundary conditions in the variable `bc_u`.

```
bc_phi = [DirichletBC(V, Constant(1.0), Crack)]
```

We define the ϕ-related boundary conditions by using the class `DirichletBC`.

```
boundaries = MeshFunction("size_t", mesh, mesh.topology().dim() - 1)
```

We use the function `MeshFunction` to identify the boundaries of the domain. This is needed to then extract the reaction force.

```
boundaries.set_all(0)
```

We initialize all boundaries before marking them.

```
top.mark(boundaries,1)
```

We mark the top boundary, where we will measure the applied force.

```
ds = Measure("ds")(subdomain_data=domainBoundaries)
```

FEniCS/Dolfin predefines the measure `ds` to integrate over exterior (boundary) facets.

```
n = FacetNormal(mesh)
```

We define the normal to the surface using the `FacetNormal` command.

```
unew, uold = Function(W), Function(W)
```

We define the solution space variables for the displacement ($\boldsymbol{u}_t$ and $\boldsymbol{u}_{t-1}$).

```
pnew, pold, Hold = Function(V), Function(V), Function(V)
```

We define the solution space variables for the phase field ($\boldsymbol{\phi}_t$ and $\boldsymbol{\phi}_{t-1}$), as well as the previous history field ($\boldsymbol{H}_{t-1}$).


```
pnew, pold, Hold = Function(V), Function(V), Function(V)
```

We define the solution space variables for the phase field ($\boldsymbol{\phi}_t$ and $\boldsymbol{\phi}_{t-1}$), as well as the previous history field ($\boldsymbol{H}_{t-1}$).


```
E_du = ((1.0-pold)**2)*inner(grad(v),sigma(u))*dx
```

We write down the weak form of the displacement problem, as given by Eq. (10a). Here, `dx` represents integration over cells.

```
E_phi = (Gc*l*inner(grad(p),grad(q))+((Gc/l)+2.0*H(uold,unew,Hold))\
            *inner(p,q)-2.0*H(uold,unew,Hold)*q)*dx
```

We write down the weak form of the phase field problem, as given by (10b).

```
p_disp = LinearVariationalProblem(lhs(E_du), rhs(E_du), unew, bc_u)
```

We use the class `LinearVariationalProblem` to define the modified elasticity problem.

```
p_phi = LinearVariationalProblem(lhs(E_phi), rhs(E_phi), pnew, bc_phi)
```

We define the linear variational phase field problem.

```
solver_disp = LinearVariationalSolver(p_disp)
```

We store in `solver_disp` the call to the linear solver to obtain $\boldsymbol{u}$.

```
solver_phi = LinearVariationalSolver(p_phi)
```

We store in `solver_phi` the call to the linear solver to obtain ϕ.

```
t = 0
```

We initialize the total computation time, which will ramp up to 1.

```
u_r = 0.007
```

We assign a magnitude of 0.007 mm to the remote displacement u_r.

```
deltaT  = 0.1
```

We initialize the time/load increment $\Delta t = 0.1$.

```
tol = 1e-3
```

We define a tolerance value for the convergence check.

```
conc_f = File ("./ResultsDir/phi.pvd")
```

We create a new folder and store there the file to be read with Paraview.

```
fname = open('ForcevsDisp.txt', 'w')
```

We open a text file to store the force versus displacement curve.

```
while t<=1.0:
```

We start the iterative scheme.

```
t += deltaT
if t >=0.7:
   deltaT = 0.0001
```

We increase progressively the time, based on the Δt defined before, but we reduce Δt when approaching the crack growth regime.

```
load.t=t*u_r
```

We increase the remote load with the time.

```
iter = 0
err = 1
```

We initialize the iteration counter and the error.

```
while err > tol:
iter += 1
```

We iterate within the same load increment until achieving convergence.

```
solver_disp.solve()
solver_phi.solve()
```

We solve the linearized problem for the displacements and the phase field.

```
err_u = errornorm(unew,uold,norm_type = 'l2',mesh = None)
err_phi = errornorm(pnew,pold,norm_type = 'l2',mesh = None)
```

We compute the error in both the $\boldsymbol{u}$ and the ϕ solutions by comparing with the previous solution by means of the L2 norm.

```
err = max(err_u,err_phi)
```

We take the maximum error from the $\boldsymbol{u}$ and the ϕ solutions, to compare with the tolerance value.

```
uold.assign(unew)
pold.assign(pnew)
Hold.assign(project(psi(unew), WW))
```

We store the $\boldsymbol{u}$ and ϕ solutions, and compute and project Ψ^+.

```
if err < tol:
print ('Iterations:', iter, ', Total time', t)
```

If convergence has been achieved, we print the number of iterations required, as well as the current value of the total computation time (out of 1).

```
if round(t*1e4) % 10 == 0:
```

For efficiency, we choose to print output information for, at most, 1000 load steps.

```
conc_f << pnew
```

We write to Paraview the solution for ϕ.

```
Traction = dot(sigma(unew),n)
fy = Traction[1]*ds(1)
fname.write(str(t*u_r) + "\t")
fname.write(str(assemble(fy)) + "\n")
```

We compute the traction as $\boldsymbol{T} = \boldsymbol{\sigma} \cdot \boldsymbol{n}$, compute then the vertical reaction force, and write it into the the text file, together with the current value of u_r.

```
fname.close()
print ('Simulation completed')
```

Finally we close the .txt file and print an end message.

Appendix B. Extension to Functionally Graded Materials

We extend the phase field fracture formulation to Functionally Graded Materials (FGMs), see `PhaseFieldFGM.py`. The stiffness and fracture resistance dependence on $\boldsymbol{x}$ is inferred from the spatial variation of the volume fractions of constituent materials via homogenization. Consider a FGM specimen that gradually changes from 100% volume fraction of compound 1 to 100% volume fraction of compound 2. Assuming an FGM beam with thickness h and material gradation along a y-axis centered at the mid-plane, the volume fraction of material 1, V_1, reads,

$$V_1 = \left(\frac{1}{2} + \frac{y}{h} \right)^k \tag{B.1}$$

where k is the material gradient index or volume fraction exponent. This is captured in the code by defining

```
self.Vf = pow((0.5 + self.x[1]), self.K)
```

We employ a Mori-Tanaka homogenization scheme to obtain the local effective elastic properties as a function of the volume fraction. Hence, the effective bulk modulus K_e and shear modulus μ_e can be obtained as,

$$\frac{K_e - K_1}{K_2 - K_1} = \frac{V_2}{1 + 3V_1 \left(K_2 - K_1 \right) / \left(3K_1 + 4\mu_1 \right)} \tag{B.2}$$

$$\frac{\mu_e - \mu_1}{\mu_2 - \mu_1} = \frac{V_2}{1 + V_1 \left(\mu_2 - \mu_1\right) / \left(\mu_1 + \mu_1 \left(9K_1 + 8\mu_1\right) / \left(6 \left(K_1 + 2\mu_1\right)\right)\right)} \tag{B.3}$$

The associated lines of code are:

```
self.Ke = self.K1 + (self.K2-self.K1)*(1.-self.Vf)\
/(1.+3.*self.Vf*((self.K2-self.K1)/(3.*self.K1 + 4.*self.G1)))
```

and,

```
term1 = self.Vf*(self.G2 - self.G1)
term2 = (9.0*self.K1+8.0*self.G1)/(6.0*(self.K1+2.0*self.G1))
self.Ge = self.G1 + (self.G2-self.G1)*(1.-self.Vf)\
/(1.0+term1/(self.G1+self.G1*term2))
```

From K_e and μ_e one can readily compute the effective Young's modulus E_e, Poisson's ratio ν_e and Lame's first parameter λ_e using the standard relations.

The fracture behaviour is typically given by the fracture toughness of the components K_{Ic}. Accordingly, G_c is computed from

$$G_c = \frac{\left(1 - \nu^2\right) K_{Ic}^2}{E} \tag{B.4}$$

References

[1] G. Francfort, J.-J. Marigo, Revisiting brittle fracture as an energy minimization problem, Journal of the Mechanics and Physics of Solids 46 (8) (1998) 1319–1342.

[2] B. Bourdin, G. A. Francfort, J. J. Marigo, Numerical experiments in revisited brittle fracture, Journal of the Mechanics and Physics of Solids 48 (4) (2000) 797–826.

[3] B. Bourdin, G. A. Francfort, J. J. Marigo, The variational approach to fracture, Springer Netherlands, 2008.

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

[5] C. Miehe, M. Hofacker, F. Welschinger, A phase field model for rate-independent crack propagation: Robust algorithmic implementation based on operator splits, Computer Methods in Applied Mechanics and Engineering 199 (45-48) (2010) 2765–2778.

[6] M. J. Borden, T. J. R. Hughes, C. M. Landis, A. Anvari, I. J. Lee, A phase-field formulation for fracture in ductile materials: Finite deformation balance law derivation, plastic degradation, and stress triaxiality effects, Computer Methods in Applied Mechanics and Engineering 312 (2016) 130–166.

[7] C. Miehe, F. Aldakheel, A. Raina, Phase field modeling of ductile fracture at finite strains: A variational gradient-extended plasticity-damage theory, International Journal of Plasticity 84 (2016) 1–32.

[8] A. Mikelic, M. Wheeler, T. Wick, A phase-field method for propagating fluid-filled fractures coupled to a surrounding porous medium, Multiscale Modeling and Simullation 13 (2015) 367–398.

[9] Z. A. Wilson, C. M. Landis, Phase-field modeling of hydraulic fracture, Journal of the Mechanics and Physics of Solids 96 (2016) 264–290.

[10] J. Reinoso, M. Paggi, C. Linder, Phase field modeling of brittle fracture for enhanced assumed strain shells at large deformations: formulation and finite element implementation, Computational Mechanics 59 (6) (2017) 981–1001.

[11] V. Carollo, J. Reinoso, M. Paggi, A 3D finite strain model for intralayer and interlayer crack simulation coupling the phase field approach and cohesive zone model, Composite Structures 182 (2017) 636–651.

[12] E. Martínez-Pañeda, A. Golahmar, C. F. Niordson, A phase field formulation for hydrogen assisted cracking, Computer Methods in Applied Mechanics and Engineering 342 (2018) 742–761.

[13] M. S. Alnaes, J. H. J. Blechta, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, G. N. Wells, The FEniCS Project Version 1.5, Archive of Numerical Software 3 (100) (2007) 9–23.

[14] G. Bellettini, A. Coscia, Discrete approximation of a free discontinuity problem, Numerical Functional Analysis and Optimization 15 (3-4) (1994) 201–224.

[15] M. Ambati, T. Gerasimov, L. De Lorenzis, A review on phase-field models of brittle fracture and a new fast hybrid formulation, Computational Mechanics 55 (2015) 383–405.

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

[17] C. Geuzaine, J. F. Remacle, Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities, International Journal for Numerical Methods in Engineering 79 (2009) 1309–1331.

[18] J. Ahrens, B. Geveci, C. Law, ParaView : An End-User Tool for Large Data Visualization, Visualization Handbook, Elsevier, 2005.

[19] S. del Busto, C. Betegón, E. Martínez-Pañeda, A cohesive zone framework for environmentally assisted fatigue, Engineering Fracture Mechanics 185 (2017) 210–226.

[20] J. Segurado, J. LLorca, A new three-dimensional interface finite element to simulate fracture in composites, International Journal of Solids and Structures 41 (11-12) (2004) 2977–2993.

[21] E. Martínez-Pañeda, S. del Busto, C. Betegón, Non-local plasticity effects on notch fracture mechanics, Theoretical and Applied Fracture Mechanics 92 (2017) 276–287.

[22] Hirshikesh, S. Natarajan, R. K. Annabattula, E. Martínez-Pañeda, Phase field modelling of crack propagation in functionally graded materials, Composites Part B: Engineering 169 (2019) 239–248.