An implementation of a unilateral constitutive law for a dynamic damage phase field modeling owing to a FEniCSx/MGIS association

Lamia MERSEL [1, 3]

Thesis director: Julien RETHORE [1]

Co-supervisor: Pascal BOUDA [2], Jérémy GERMAIN [3]

[1] GeM/CNRS, Nantes, [2] Université Paris-Saclay, Cea, [3] Onera/DMAS/CRD, Lille

User Meeting MFront/MGIS, 19th November 2024, Saclay

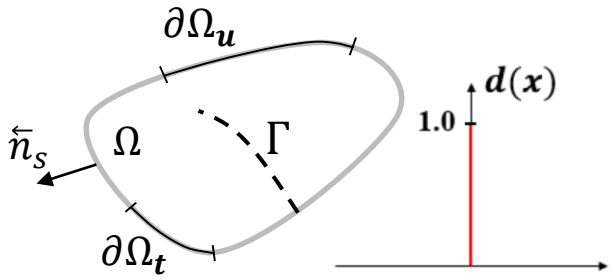
Outline

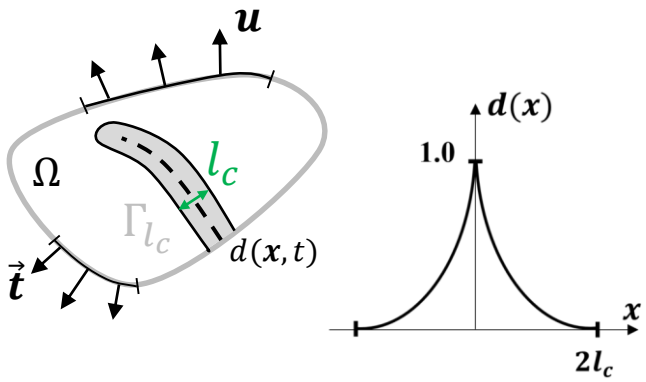
1. **Context and Motivations**
    - Phase field damage modelling for dynamic fracture
    - Unilateral contact condition at the crack lip
2. **Implementation framework**
    - Implementation of the constitutive law
    - FEniCSx – MGIS coupling
3. **Numerical applications**
4. **Discussions, conclusions and perspectives**

Outline

Phase-field mathematic background

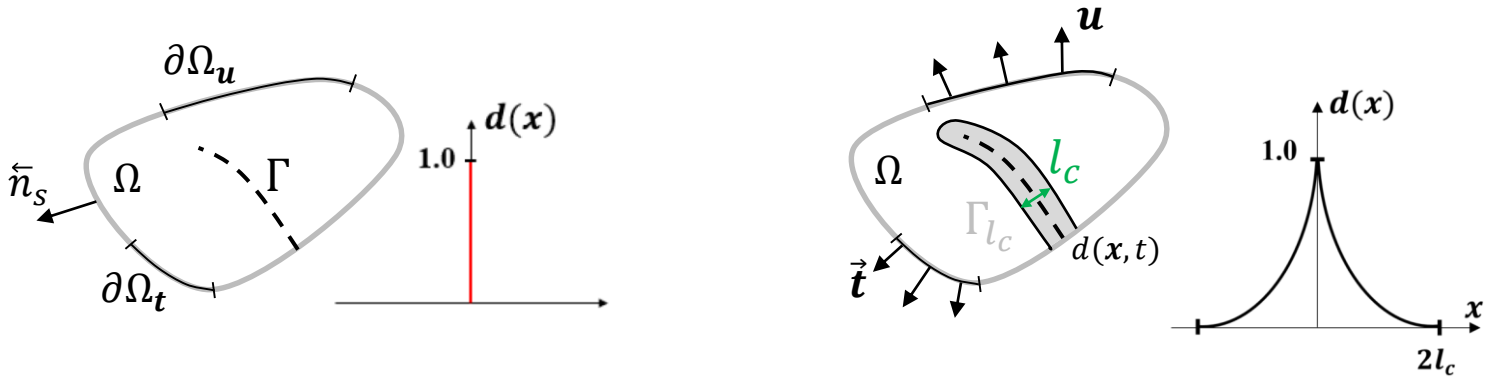
- From a discontinous fracture problem to continous one through an additional field the damage variable « d » *[Francfort, Marigo and Bourdin]*





Phase-field mathematic background

- From a discontinous fracture problem to continous one through an additional field the damage variable « d » *[Francfort, Marigo and Bourdin]*



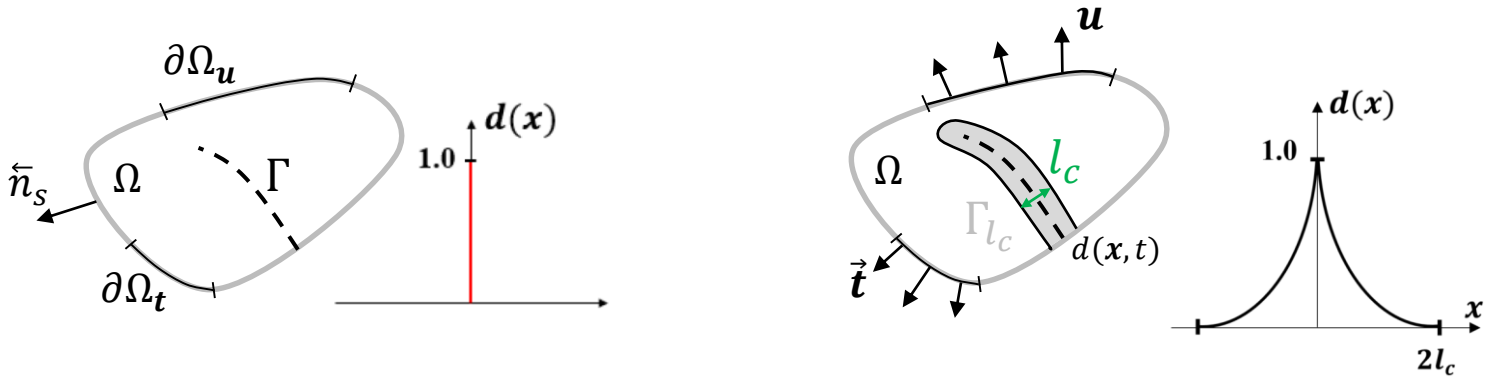
- Minimisation of the total energy of the system, the Lagrangian function, $\left(-L^{dyn}\right)$:

$$\mathrm{E}_{elastic}\,(\boldsymbol{u},d) \quad + \quad \mathrm{E}_{fracture}\,(d) \quad - \; P_{ext}(\boldsymbol{u}) \quad - \; \mathcal{K}_u\,(\dot{\boldsymbol{u}}) - \; D_d\,(\dot{d})$$

$$L^{dyn}\left(t,\boldsymbol{u},\dot{\boldsymbol{u}},d,\dot{d}\right) = \int \Psi([\boldsymbol{\varepsilon}(\boldsymbol{u})],d)\,d\tilde{x} \; + \int G_c\,\gamma(l_c,d,\vec{\boldsymbol{\nabla}}d)\,d\tilde{x} - P_{ext}(\boldsymbol{u}) - \int \frac{\rho_{\boldsymbol{u}}}{2}\dot{\boldsymbol{u}}^2\,d\tilde{x} - \int \frac{\rho_d}{2}\dot{d}^2\,d\tilde{x}$$

Phase-field mathematic background

- From a discontinous fracture problem to continous one through an additional field the damage variable « d » *[Francfort, Marigo and Bourdin]*



- Minimisation of the total energy of the system, the Lagrangian function, $\left(-L^{dyn}\right)$:

$$\mathrm{E}_{elastic}\,(\boldsymbol{u},d) \quad + \quad \mathrm{E}_{fracture}\,(d) \quad - \; P_{ext}(\boldsymbol{u}) \quad - \; \mathcal{K}_u\,(\dot{\boldsymbol{u}}) - D_d\,(\dot{d})$$

$$L^{dyn}\left(t,\boldsymbol{u},\dot{\boldsymbol{u}},d,\dot{d}\right) = \int \Psi([\boldsymbol{\varepsilon}(\boldsymbol{u})],d)\,d\tilde{x} \; + \int G_c\,\gamma(l_c,d,\vec{\boldsymbol{\nabla}}d)\,d\tilde{x} - P_{ext}(\boldsymbol{u}) - \int \frac{\rho_{\boldsymbol{u}}}{2}\dot{\boldsymbol{u}}^2\,d\tilde{x} - \int \frac{\rho_d}{2}\dot{d}^2\,d\tilde{x}$$

Elastic energy density split $\Psi([\boldsymbol{\varepsilon}],d)$
(Traction-compression asymmetry)
$\Psi = \mathrm{g(d)}\Psi^+ + \Psi^-$
$\mathrm{g(d)}$: degradation function

Crack density function $\gamma(l_c,d,\vec{\boldsymbol{\nabla}}d)$
G_c : critical energy release rate
l_c : internal length

Inertia effects :
$\rho_{\boldsymbol{u}}$: mass density
ρ_d : micro-inertia related to microcrack evolution

Motivation

Admissible spaces :

$$\bar{\Omega}_{\boldsymbol{u}} = \left\{ \underline{\boldsymbol{v}} : \mathrm{T} \times \Omega \to H^1\left(\Omega, \boldsymbol{R}^{dim}\right),\ \underline{\boldsymbol{v}} = \underline{\boldsymbol{u}} \text{ on } \boldsymbol{\partial}_{\underline{\boldsymbol{u}}}\Omega \right\}, \qquad \bar{\Omega}_d = \{\beta : T \times \Omega \to [0,1], 0 < d < \beta < 1,\ \beta = d \text{ on } \partial_d \Omega\}$$

Weak formulation : $\left(\underline{\boldsymbol{u}}, \mathrm{d}\right) \in \bar{\Omega}_{\boldsymbol{u}} \times \bar{\Omega}_d,\ \forall\, (\delta\underline{\boldsymbol{u}}, \delta d) \in \bar{\Omega}_{\boldsymbol{u}} \times \bar{\Omega}_d$

$$\int_\Omega \int_t \left(-\rho_{\boldsymbol{u}} \underline{\ddot{\boldsymbol{u}}} + \boldsymbol{\nabla}.([\boldsymbol{\sigma}]) + \boldsymbol{b}\right) \delta\underline{\boldsymbol{u}}\, dt\, d\tilde{x} + \int_{\partial\Omega} -[\boldsymbol{\sigma}].\tilde{n}_s\, \delta\underline{\boldsymbol{u}} ds + \int_{\partial\Omega_n} \underline{\boldsymbol{t}}.\tilde{n}_s\, \delta\mathbf{u}\, ds = 0, \qquad \forall \delta\underline{\boldsymbol{u}} \in \partial\bar{\Omega}_{\boldsymbol{u}}$$

$$\int_\Omega \int_t \left(\rho_d \ddot{d} + \mu_d \dot{d} + \left(\frac{\partial \Psi}{\partial d} + G_c \frac{\partial \gamma}{\partial d} - G_c \boldsymbol{\nabla}.\left(\frac{\partial \gamma}{\partial \underline{\boldsymbol{\nabla d}}} \right) \right) \right) \delta d\, dt\, d\tilde{x} + \int_{\partial\Omega} \frac{\partial \gamma}{\underline{\boldsymbol{\nabla d}}} \delta d.\tilde{n}_s ds = 0, \qquad \forall\, \delta d \in \partial\bar{\Omega}_d$$

Motivation

Admissible spaces :

$$\bar{\Omega}_{\boldsymbol{u}} = \left\{ \underline{\boldsymbol{v}} : \mathrm{T} \times \Omega \to H^1\left(\Omega, \boldsymbol{R}^{dim}\right),\ \underline{\boldsymbol{v}} = \underline{\boldsymbol{u}} \text{ on } \boldsymbol{\partial}_{\underline{\boldsymbol{u}}}\Omega \right\}, \qquad \bar{\Omega}_d = \{\beta : T \times \Omega \to [0,1], 0 < d < \beta < 1,\ \beta = d \text{ on } \partial_d \Omega\}$$

Weak formulation : $\left(\underline{\boldsymbol{u}}, \mathrm{d}\right) \in \bar{\Omega}_{\boldsymbol{u}} \times \bar{\Omega}_d,\ \forall\, (\delta\underline{\boldsymbol{u}}, \delta d) \in \bar{\Omega}_{\boldsymbol{u}} \times \bar{\Omega}_d$

$$\int_\Omega \int_t \left(-\rho_{\boldsymbol{u}} \underline{\ddot{\boldsymbol{u}}} + \boldsymbol{\nabla}.([\boldsymbol{\sigma}]) + \boldsymbol{b}\right) \delta\underline{\boldsymbol{u}}\, dt\, d\tilde{x} + \int_{\partial\Omega} -[\boldsymbol{\sigma}].\tilde{n}_s\, \delta\underline{\boldsymbol{u}} ds + \int_{\partial\Omega_n} \underline{\boldsymbol{t}}.\tilde{n}_s\, \delta \mathbf{u}\, ds = 0, \qquad \forall \delta\underline{\boldsymbol{u}} \in \partial\bar{\Omega}_{\boldsymbol{u}}$$

$$\int_\Omega \int_t \left(\rho_d \ddot{d} + \mu_d \dot{d} + \left(\frac{\partial \Psi}{\partial d} + G_c \frac{\partial \gamma}{\partial d} - G_c \boldsymbol{\nabla}.\left(\frac{\partial \gamma}{\partial \underline{\boldsymbol{\nabla d}}} \right) \right) \right) \delta d\, dt\, d\tilde{x} + \int_{\partial\Omega} \frac{\partial \gamma}{\underline{\boldsymbol{\nabla d}}} \delta d.\tilde{n}_s ds = 0, \qquad \forall\, \delta d \in \partial\bar{\Omega}_d$$

- Energetic crack driving force : $Y = -\frac{\partial \Psi}{\partial d} = -g'(d)\frac{\partial \Psi}{\partial g} = -g'(d)H(\underline{\boldsymbol{u}})$, g a degradation function

Motivation

Admissible spaces :

$$\bar{\Omega}_{\boldsymbol{u}} = \left\{ \underline{\boldsymbol{v}} : \mathrm{T} \times \Omega \to H^1\left(\Omega, \boldsymbol{R}^{dim}\right),\ \underline{\boldsymbol{v}} = \underline{\boldsymbol{u}} \text{ on } \boldsymbol{\partial}_{\underline{\boldsymbol{u}}}\Omega \right\}, \qquad \bar{\Omega}_d = \{\beta : T \times \Omega \to [0,1], 0 < d < \beta < 1,\ \beta = d \text{ on } \partial_d \Omega\}$$

Weak formulation : $\left(\underline{\boldsymbol{u}}, \mathrm{d}\right) \in \bar{\Omega}_{\boldsymbol{u}} \times \bar{\Omega}_d,\ \forall\, (\delta\underline{\boldsymbol{u}}, \delta d) \in \bar{\Omega}_{\boldsymbol{u}} \times \bar{\Omega}_d$

$$\int_\Omega \int_t \left(-\rho_{\boldsymbol{u}} \underline{\ddot{\boldsymbol{u}}} + \boldsymbol{\nabla}.([\boldsymbol{\sigma}]) + \boldsymbol{b}\right) \delta\underline{\boldsymbol{u}}\, dt\, d\tilde{x} + \int_{\partial\Omega} -[\boldsymbol{\sigma}].\tilde{n}_s\, \delta\underline{\boldsymbol{u}} ds + \int_{\partial\Omega_n} \underline{\boldsymbol{t}}.\tilde{n}_s\, \delta \mathbf{u}\, ds = 0, \qquad \forall \delta\underline{\boldsymbol{u}} \in \partial\bar{\Omega}_{\boldsymbol{u}}$$

$$\int_\Omega \int_t \left(\rho_d \ddot{d} + \mu_d \dot{d} + \left(\frac{\partial \Psi}{\partial d} + G_c \frac{\partial \gamma}{\partial d} - G_c \boldsymbol{\nabla}.\left(\frac{\partial \gamma}{\partial \underline{\boldsymbol{\nabla d}}} \right) \right) \right) \delta d\, dt\, d\tilde{x} + \int_{\partial\Omega} \frac{\partial \gamma}{\underline{\boldsymbol{\nabla d}}} \delta d.\tilde{n}_s ds = 0, \qquad \forall\, \delta d \in \partial\bar{\Omega}_d$$

- Energetic crack driving force : $Y = -\frac{\partial \Psi}{\partial d} = -g'(d)\frac{\partial \Psi}{\partial g} = -g'(d)H(\underline{\boldsymbol{u}})$, g a degradation function
- Irreversibility constraint with a history variable H *[Miehe et al., 2010]* : $H = \max_{0 \le \tau \le T}(0, \Psi^+(\varepsilon, \tau))$

Motivation

Admissible spaces :

$$\bar{\Omega}_{\boldsymbol{u}} = \left\{ \underline{\boldsymbol{v}} : \mathrm{T} \times \Omega \to H^1\left(\Omega, \boldsymbol{R}^{dim}\right),\ \underline{\boldsymbol{v}} = \underline{\boldsymbol{u}} \text{ on } \boldsymbol{\partial}_{\underline{\boldsymbol{u}}}\Omega \right\}, \qquad \bar{\Omega}_d = \{\beta : T \times \Omega \to [0,1], 0 < d < \beta < 1,\ \beta = d \text{ on } \partial_d \Omega\}$$

Weak formulation : $\left(\underline{\boldsymbol{u}}, \mathrm{d}\right) \in \bar{\Omega}_{\boldsymbol{u}} \times \bar{\Omega}_d,\ \forall\, (\delta\underline{\boldsymbol{u}}, \delta d) \in \bar{\Omega}_{\boldsymbol{u}} \times \bar{\Omega}_d$

$$\int_\Omega \int_t \left(-\rho_{\boldsymbol{u}} \underline{\ddot{\boldsymbol{u}}} + \boldsymbol{\nabla}.([\boldsymbol{\sigma}]) + \boldsymbol{b}\right) \delta\underline{\boldsymbol{u}}\, dt\, d\tilde{x} + \int_{\partial\Omega} -[\boldsymbol{\sigma}].\tilde{n}_s\, \delta\underline{\boldsymbol{u}} ds + \int_{\partial\Omega_n} \underline{\boldsymbol{t}}.\tilde{n}_s\, \delta \mathbf{u}\, ds = 0, \qquad \forall \delta\underline{\boldsymbol{u}} \in \partial\bar{\Omega}_{\boldsymbol{u}}$$

$$\int_\Omega \int_t \left(\rho_d \ddot{d} + \mu_d \dot{d} + \left(\frac{\partial \Psi}{\partial d} + G_c \frac{\partial \gamma}{\partial d} - G_c \boldsymbol{\nabla}.\left(\frac{\partial \gamma}{\partial \underline{\boldsymbol{\nabla d}}} \right) \right) \right) \delta d\, dt\, d\tilde{x} + \int_{\partial\Omega} \frac{\partial \gamma}{\underline{\boldsymbol{\nabla d}}} \delta d.\tilde{n}_s ds = 0, \qquad \forall\, \delta d \in \partial\bar{\Omega}_d$$

- Energetic crack driving force : $Y = -\frac{\partial \Psi}{\partial d} = -g'(d)\frac{\partial \Psi}{\partial g} = -g'(d)H(\underline{\boldsymbol{u}})$, g a degradation function
- Irreversibility constraint with a history variable *H* *[Miehe et al., 2010]* : $H = \max_{0 \le \tau \le T}(0, \Psi^+(\varepsilon, \tau))$
- Traction/compression assymetry ➔ unphysical crack under compression – unilateral constraint on the crack lips energy split method (non-penetration condition)

Motivation

Admissible spaces :

$$\bar{\Omega}_{\boldsymbol{u}} = \left\{ \underline{\boldsymbol{v}} : \mathrm{T} \times \Omega \to H^1\left(\Omega, \boldsymbol{R}^{dim}\right),\ \underline{\boldsymbol{v}} = \underline{\boldsymbol{u}} \text{ on } \boldsymbol{\partial}_{\underline{\boldsymbol{u}}}\Omega \right\}, \qquad \bar{\Omega}_d = \{\beta : T \times \Omega \to [0,1], 0 < d < \beta < 1,\ \beta = d \text{ on } \partial_d \Omega\}$$

Weak formulation : $\left(\underline{\boldsymbol{u}}, \mathrm{d}\right) \in \bar{\Omega}_{\boldsymbol{u}} \times \bar{\Omega}_d,\ \forall\, (\delta\underline{\boldsymbol{u}}, \delta d) \in \bar{\Omega}_{\boldsymbol{u}} \times \bar{\Omega}_d$

$$\int_\Omega \int_t \left(-\rho_{\boldsymbol{u}} \underline{\ddot{\boldsymbol{u}}} + \boldsymbol{\nabla}.([\boldsymbol{\sigma}]) + \boldsymbol{b}\right) \delta\underline{\boldsymbol{u}}\, dt\, d\tilde{x} + \int_{\partial\Omega} -[\boldsymbol{\sigma}].\tilde{n}_s\, \delta\underline{\boldsymbol{u}} ds + \int_{\partial\Omega_n} \underline{\boldsymbol{t}}.\tilde{n}_s\, \delta\mathbf{u}\, ds = 0, \qquad \forall \delta\underline{\boldsymbol{u}} \in \partial\bar{\Omega}_{\boldsymbol{u}}$$

$$\int_\Omega \int_t \left(\cancel{\rho_d \ddot{d}} + \cancel{\mu_d \dot{d}} + \left(\frac{\partial \Psi}{\partial d} + G_c \frac{\partial \gamma}{\partial d} - G_c \boldsymbol{\nabla}.\left(\frac{\partial \gamma}{\partial \underline{\boldsymbol{\nabla d}}} \right) \right) \right) \delta d\, dt\, d\tilde{x} + \int_{\partial\Omega} \frac{\partial \gamma}{\underline{\boldsymbol{\nabla d}}} \delta d.\tilde{n}_s ds = 0, \qquad \forall\, \delta d \in \partial\bar{\Omega}_d$$

- Energetic crack driving force : $Y = -\frac{\partial \Psi}{\partial d} = -g'(d)\frac{\partial \Psi}{\partial g} = -g'(d)H(\underline{\boldsymbol{u}})$, g a degradation function
- Irreversibility constraint with a history variable H *[Miehe et al., 2010]* : $H = \max_{0 \le \tau \le T}(0, \Psi^+(\varepsilon, \tau))$
- Traction/compression assymetry ➔ unphysical crack under compression – unilateral constraint on the crack lips energy split method (non-penetration condition)

Outline

Energy density split method

Spherical/deviatoric decomposition *[Kotsovos, 1979; Ladeveze, 1983;…; Amor, 2008]*

$$\Psi^+ = \frac{1}{2}\lambda_0\langle tr(\boldsymbol{\varepsilon})\rangle_+^2 + \mu_0\, \boldsymbol{dev}(\boldsymbol{\varepsilon}) : \boldsymbol{dev}(\boldsymbol{\varepsilon}),$$

$$\Psi^- = \frac{1}{2}\lambda_0\langle tr(\boldsymbol{\varepsilon})\rangle_-^2$$

$$\boldsymbol{\sigma}(d,\boldsymbol{\varepsilon}) = g(d)\left(\lambda_0\langle tr(\boldsymbol{\varepsilon})\rangle_+ \boldsymbol{I} + \mu_0 \boldsymbol{dev}(\boldsymbol{\varepsilon})\right) + \left(\lambda_0\langle tr(\boldsymbol{\varepsilon})\rangle_-\right)$$

CONS :

- Developped from isotropic material
- Loss of the thermodynamical framework (loss of the symmetry of the stiffness, continuity of the stress)

Spectral decomposition on the strain (or stress)

[Mazars et al. 1989, Lemaitre et al. 2000,…Miehe, 2010 ; Cevera, 2021]

$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^+ + \boldsymbol{\varepsilon}^-$, with $\boldsymbol{\varepsilon}^\pm = \sum_{n=1}^{3}\langle \varepsilon_n\rangle_\pm \underline{p_n}\otimes\underline{p_n}$,

$(\varepsilon_n, \underline{p_n})$: nth eigenvalue, eigenvector

$\mathbb{D}^\pm(\boldsymbol{\varepsilon}) = \partial_{\boldsymbol{\varepsilon}}[\boldsymbol{\varepsilon}^\pm]$ (projection tensor)

$$\boldsymbol{\sigma}(\boldsymbol{\varepsilon}, d) = \mathbb{C}(d) : \boldsymbol{\varepsilon},$$

$$\text{with } \mathbb{C}(d) = g(d)\,\mathbb{D}^+ : (\mathbb{C} : \mathbb{D}^+) + \mathbb{D}^- : (\mathbb{C} : \mathbb{D}^-)$$

Isotropic elasticity assumption : $\mathbb{C} = 2\boldsymbol{\mu}\mathbb{I} + \boldsymbol{\lambda}\mathbf{I}\otimes\mathbf{I}$,

$$\Psi^\pm = \frac{1}{2}\lambda_0\langle tr(\boldsymbol{\varepsilon})\rangle_\pm^2 + \mu_0\boldsymbol{\varepsilon}^\pm : \boldsymbol{\varepsilon}^\pm$$

$$\boldsymbol{\sigma}(d,\boldsymbol{\varepsilon}) = g(d)\,(\lambda_0\langle tr(\boldsymbol{\varepsilon})\rangle_+ \boldsymbol{I} + \mu_0\boldsymbol{\varepsilon}^+) + (\lambda_0\langle tr(\boldsymbol{\varepsilon})\rangle_- + \mu_0\boldsymbol{\varepsilon}^-)$$

Energy density split method

Orthogonal decomposition on the elastic tensor $\mathbb{C}$

[Marc François, 1998, Desmorat, 2000; He and Shao, 2019; …]

$$\Psi^{\pm} = \frac{1}{2}\boldsymbol{\varepsilon}^{\pm}\colon\mathbb{C}\colon\boldsymbol{\varepsilon}^{\pm} \text{ with } \boldsymbol{\varepsilon}^{+}\colon(\mathbb{C}\colon\boldsymbol{\varepsilon}^{-}) = \mathbf{0}$$

Kelvin decomposition of the stiffness tensor :

$\mathbb{C} = \sum_{i=1}^{3}\Lambda_i\boldsymbol{\omega_i}\otimes\boldsymbol{\omega_i}$, 4th order elastic tensor $\mathbb{C}$

$\Lambda_i, \boldsymbol{\omega_i}$: eigenvalue, 2nd order eigentensor of $\mathbb{C}$

- $\mathbb{C}^{\pm 1/2} = \sum_{i=1}^{3}\Lambda_i{}^{\pm 1/2}\boldsymbol{\omega_i}\otimes\boldsymbol{\omega_i}$
- $\tilde{\boldsymbol{\varepsilon}} = \mathbb{C}^{1/2}\boldsymbol{\varepsilon}$ (transformed strain),
- $\tilde{\boldsymbol{\varepsilon}}^{\pm}$: positive and negative part from the spectral decomposition of $\tilde{\boldsymbol{\varepsilon}}$
- $\widetilde{\mathbb{P}}^{\pm}(\tilde{\boldsymbol{\varepsilon}}) = \partial_{\tilde{\boldsymbol{\varepsilon}}}[\tilde{\varepsilon}^{\pm}]$ (projection tensor) for implicit solver
- $\boldsymbol{\varepsilon}^{\pm} = \mathbb{C}^{-1/2}\tilde{\boldsymbol{\varepsilon}}^{\pm}$

$$\boldsymbol{\sigma}(\boldsymbol{\varepsilon}, d) = \mathbb{C}(d)\colon\boldsymbol{\varepsilon},$$

with $\mathbb{C}(d) = g(d)\mathbb{P}^{+}\colon(\mathbb{C}:\mathbb{P}^{+}) + \mathbb{P}^{-}\colon(\mathbb{C}:\mathbb{P}^{-})$ and

$\mathbb{P}^{\pm} = \mathbb{C}^{-1/2}\colon(\widetilde{\mathbb{P}}^{\pm}\colon\mathbb{C}^{+1/2})$

PROS : Anisotropic material, explicit resolution no need of the projectors $\mathbb{P}^{\pm}$

Outline

MFront Implementation : Orthogonal split

➢ Variable definition

```
Yg.setGlossaryName("YoungModulus");
@MaterialProperty real nu;
nu.setGlossaryName("PoissonRatio");
@ComputeStiffnessTensor<UnAltered> { "Yg" , "nu" };

@Parameter real kres = 1e-6;  // residual stiffness

@StateVariable real H;
H.setEntryName("HistoryFunction");
@StateVariable real psi_p;
psi_p.setEntryName("PositiveEnergyDensity");

@ExternalStateVariable real psi_0;
psi_0.setEntryName("ThresholdEnergyDensity");
@ExternalStateVariable real d;
d.setGlossaryName("Damage");
```

Isotrop linear elastic

$$\mathbb{C} = \frac{\mathrm{E}}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ & 1-\nu & \nu & 0 & 0 & 0 \\ & & 1-\nu & 0 & 0 & 0 \\ & & & (1-2\nu)/2 & 0 & 0 \\ & & & & (1-2\nu)/2 & 0 \\ & & & & & (1-2\nu)/2 \end{bmatrix}$$

$$H = \max_{0\le\tau\le T}(0, \Psi^{+}(\varepsilon,\tau) - \Psi_0)$$

https://thelfer.github.io/mgis/web/mgis_fenics_phase_field.html

MFront Implementation : Integrator block


```
@Integrator {
  constexpr const strain emin = 1.e-12;
  constexpr const auto id4 = Stensor4::Id();
  constexpr const auto esolver = StrainStensor::
   GTESYMMETRICQREIGENSOLVER;

  // call diagonalize() function
  auto [vp, ns, Dp1_2, Dn1_2] = diagonalize(D);

  // positive part
  const auto f = [](const real x) { return x > 0 ? x : 0; };
  // derivative of the positive part
  const auto df = [&emin](const real x) {
    return std::abs(x) < emin ? 0.5 : ((x < 0) ? 0 : 1); };

  // total strain
  const auto e = eto + deto;
  // transformed strain tensor
  const auto et = eval(Dp1_2 * e);
  // Positive part of 'et' tensor and its projector
  const auto [etp,Ptp] = et.template
   computeIsotropicFunctionAndDerivative<esolver>(
      f, df, emin * 0.1);
  // Negative part of 'et' tensor
  const auto etn = et - etp;
  // Positive and Negative of the strain
  const auto ep = Dn1_2 * etp;
  const auto en = Dn1_2 * etn;

  // Degradation function
  const auto g = ((1-d)*(1-d)) +kres ;
  // Compute stress
  sig = g * D * ep + D * en;

  // positive elastic energy density and history H
  psi_p = 0.5*(ep|(D*ep));
  const auto h0 = max(psi_p-psi_0, real(0));
  H = max(H, h0);
  const strain tr  = trace(e);

  // Tangent operator
  static_cast<void>(smt);
  //static_cast<void>(computeTangentOperator_);
}
```

MFront Implementation : Integrator block


```
@Integrator {
  constexpr const strain emin = 1.e-12;
  constexpr const auto id4 = Stensor4::Id();
  constexpr const auto esolver = StrainStensor::
   GTESYMMETRICQREIGENSOLVER;

  // call diagonalize() function
  auto [vp, ns, Dp1_2, Dn1_2] = diagonalize(D);

  // positive part
  const auto f = [](const real x) { return x > 0 ? x : 0; };
  // derivative of the positive part
  const auto df = [&emin](const real x) {
    return std::abs(x) < emin ? 0.5 : ((x < 0) ? 0 : 1); };

  // total strain
  const auto e = eto + deto;
  // transformed strain tensor
  const auto et = eval(Dp1_2 * e);
  // Positive part of 'et' tensor and its projector
  const auto [etp,Ptp] = et.template
   computeIsotropicFunctionAndDerivative<esolver>(
      f, df, emin * 0.1);
  // Negative part of 'et' tensor
  const auto etn = et - etp;
```

$$\tilde{\boldsymbol{\varepsilon}} = \mathbb{D}^{1/2}\boldsymbol{\varepsilon}$$

$$\tilde{\boldsymbol{\varepsilon}}^{+}, \tilde{\mathbb{P}}^{+}(\tilde{\boldsymbol{\varepsilon}})$$

$$\tilde{\boldsymbol{\varepsilon}}^{-} = \tilde{\boldsymbol{\varepsilon}} - \tilde{\boldsymbol{\varepsilon}}^{+}$$

```
  // Positive and Negative of the strain
  const auto ep = Dn1_2 * etp;
  const auto en = Dn1_2 * etn;

  // Degradation function
  const auto g = ((1-d)*(1-d)) +kres ;
  // Compute stress
  sig = g * D * ep + D * en;

  // positive elastic energy density and history H
  psi_p = 0.5*(ep|(D*ep));
  const auto h0 = max(psi_p-psi_0, real(0));
  H = max(H, h0);
  const strain tr  = trace(e);

  // Tangent operator
  static_cast<void>(smt);
  //static_cast<void>(computeTangentOperator_);
}
```

MFront Implementation : Integrator block


```
@Integrator {
  constexpr const strain emin = 1.e-12;
  constexpr const auto id4 = Stensor4::Id();
  constexpr const auto esolver = StrainStensor::
   GTESYMMETRICQREIGENSOLVER;

  // call diagonalize() function
  auto [vp, ns, Dp1_2, Dn1_2] = diagonalize(D);

  // positive part
  const auto f = [](const real x) { return x > 0 ? x : 0; };
  // derivative of the positive part
  const auto df = [&emin](const real x) {
    return std::abs(x) < emin ? 0.5 : ((x < 0) ? 0 : 1); };

  // total strain
  const auto e = eto + deto;
  // transformed strain tensor
  const auto et = eval(Dp1_2 * e);
  // Positive part of 'et' tensor and its projector
  const auto [etp,Ptp] = et.template
   computeIsotropicFunctionAndDerivative<esolver>(
      f, df, emin * 0.1);
  // Negative part of 'et' tensor
  const auto etn = et - etp;
```

$$\tilde{\boldsymbol{\varepsilon}} = \mathbb{D}^{1/2}\boldsymbol{\varepsilon}$$

$$\tilde{\boldsymbol{\varepsilon}}^{+}, \tilde{\mathbb{P}}^{+}(\tilde{\boldsymbol{\varepsilon}})$$

$$\tilde{\boldsymbol{\varepsilon}}^{-} = \tilde{\boldsymbol{\varepsilon}} - \tilde{\boldsymbol{\varepsilon}}^{+}$$

```
  // Positive and Negative of the strain
  const auto ep = Dn1_2 * etp;
  const auto en = Dn1_2 * etn;

  // Degradation function
  const auto g = ((1-d)*(1-d)) +kres ;
  // Compute stress
  sig = g * D * ep + D * en;

  // positive elastic energy density and history H
  psi_p = 0.5*(ep|(D*ep));
  const auto h0 = max(psi_p-psi_0, real(0));
  H = max(H, h0);
  const strain tr  = trace(e);

  // Tangent operator
  static_cast<void>(smt);
  //static_cast<void>(computeTangentOperator_);
}
```

$$\boldsymbol{\varepsilon}^{\pm} = \mathbb{D}^{-1/2}\tilde{\boldsymbol{\varepsilon}}^{\pm}$$

$$\boldsymbol{\sigma}(\boldsymbol{\varepsilon}, d) = g(d)\mathbb{C}\boldsymbol{\varepsilon}^{+} + \mathbb{C}\boldsymbol{\varepsilon}^{-}$$

$$H = \max_{0 \le \tau \le T}(0, \Psi^{+}(\varepsilon, \tau) - \Psi_0)$$

MFront Implementation : Kelvin decomposition (1/2)


```
@OrthotropicBehaviour<Pipe>;

@Includes {

    #ifndef TFEL_MATH_ST2TOST2_DIAGONALIZE
    #define TFEL_MATH_ST2TOST2_DIAGONALIZE

    namespace tfel::math {
      /*!
      * return the eigenvalues and eigen tensors of the of a
   given orthotropic
      * stiffness tensor, as well as its positive square root
   and inverse of the positive square roots
      */
      template <unsigned short N, typename ValueType>
      std::tuple<
          tvector<StensorDimeToSize<N>::value, ValueType>,
          tvector<StensorDimeToSize<N>::value,
                stensor<N, base_type<ValueType>>>,
          const st2tost2<N, ValueType>,
          const st2tost2<N, ValueType>>
```

```
    diagonalize(const st2tost2<N, ValueType> &C) {
      using real = base_type<ValueType>;
      auto e = [](const unsigned short i) constexpr {
              auto s = stensor<N, real>(real(0));
              s[i] = 1;
              return s;
              };
      constexpr const auto cste = Cste<ValueType>::sqrt2;
      constexpr auto e1 = e(0);
      constexpr auto e2 = e(1);
      constexpr auto e3 = e(2);
      auto tmp =
      stensor<3u, ValueType>{C(0, 0), C(1, 1),
                             C(2, 2), cste * C(0, 1),
                              cste * C(0, 2), cste * C(1, 2)
};
      const auto [vp, m] = tmp.template computeEigenVectors<
          stensor_common::FSESJACOBIEIGENSOLVER>();
      auto ns = tvector<StensorDimeToSize<N>::value, stensor<
N, real>>{};
      ns[0] = m(0, 0) * e1 + m(1, 0) * e2 + m(2, 0) * e3;
      ns[1] = m(0, 1) * e1 + m(1, 1) * e2 + m(2, 1) * e3;
      ns[2] = m(0, 2) * e1 + m(1, 2) * e2 + m(2, 2) * e3;
```

Eigenvectors, ns[0], ns[1], ns[2]

MFront Implementation : Kelvin decomposition (2/2)


```
      if constexpr (N == 1) {
        // positive and negative square root
        // of stiffness tensor C
        const auto Cp1_2 =
        power<1,2>(vp(0)) * (ns[0] ^ ns[0]) +
        power<1,2>(vp(1)) * (ns[1] ^ ns[1]) +
        power<1,2>(vp(2)) * (ns[2] ^ ns[2]);
        const auto Cn1_2 =
        power<-1, 2>(vp(0)) * (ns[0] ^ ns[0]) +
        power<-1, 2>(vp(1)) * (ns[1] ^ ns[1]) +
        power<-1, 2>(vp(2)) * (ns[2] ^ ns[2]);
        return {vp, ns, Cp1_2, Cn1_2};
      } else if constexpr (N == 2) {
        ns[3] = e(3);
        const auto Cp1_2 =
        power<1, 2>(vp(0)) * (ns[0] ^ ns[0]) +
        power<1, 2>(vp(1)) * (ns[1] ^ ns[1]) +
        power<1, 2>(vp(2)) * (ns[2] ^ ns[2]) +
        power<1, 2>(C(3, 3)) * (ns[3] ^ ns[3]);
        const auto Cn1_2 =
        power<-1, 2>(vp(0)) * (ns[0] ^ ns[0]) +
        power<-1, 2>(vp(1)) * (ns[1] ^ ns[1]) +
        power<-1, 2>(vp(2)) * (ns[2] ^ ns[2]) +
        power<-1, 2>(C(3, 3)) * (ns[3] ^ ns[3]);
        return {tvector<4u, ValueType>{
          vp(0), vp(1), vp(2), C(3, 3)}, ns, Cp1_2, Cn1_2};
```

$$\mathbb{C} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & 0 \\ & C_{22} & C_{23} & 0 \\ & & C_{33} & 0 \\ & & & C_{44} \end{bmatrix}$$

```
      } else {
        ns[3] = e(3);
        ns[4] = e(4);
        ns[5] = e(5);
        const auto Cp1_2 =
        power<1, 2>(vp(0)) * (ns[0] ^ ns[0]) +
        power<1, 2>(vp(1)) * (ns[1] ^ ns[1]) +
        power<1, 2>(vp(2)) * (ns[2] ^ ns[2]) +
        power<1, 2>(C(3, 3)) * (ns[3] ^ ns[3]) +
        power<1, 2>(C(4, 4)) * (ns[4] ^ ns[4]) +
        power<1, 2>(C(5, 5)) * (ns[5] ^ ns[5]);
        const auto Cn1_2 =
        power<-1, 2>(vp(0)) * (ns[0] ^ ns[0]) +
        power<-1, 2>(vp(1)) * (ns[1] ^ ns[1]) +
        power<-1, 2>(vp(2)) * (ns[2] ^ ns[2]) +
        power<-1, 2>(C(3, 3)) * (ns[3] ^ ns[3]) +
        power<-1, 2>(C(4, 4)) * (ns[4] ^ ns[4]) +
        power<-1, 2>(C(5, 5)) * (ns[5] ^ ns[5]);
        return {tvector<6u, ValueType>{
          vp(0), vp(1), vp(2), C(3, 3), C(4, 4), C(5, 5)
          }, ns, Cp1_2, Cn1_2};
      }
    }  // end of diagonalize

  }  // end of namespace tfel::math

  #endif
}
```

$$\mathbb{C} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ & C_{22} & C_{23} & 0 & 0 & 0 \\ & & C_{33} & 0 & 0 & 0 \\ & & & C_{44} & 0 & 0 \\ & & & & C_{55} & 0 \\ & & & & & C_{66} \end{bmatrix}$$

Outline

Implementation of a flexible environment within a toolbox

- **Toolbox** implemented to test multiple resolution strategy through a channel

(**) https://github.com/FEniCS

Displacement
Damage

Field
- Space function
- Dirichlet, Load
- Mesh features
- parameters

Elliptic
Parabolic
Hyperbolic

Formulation

Implicit method
(Newton Raphson)
Explicit integrator

Numerical resolution strategy

Metric error estimation
Balance energy
Method of crack tracking a(t)

Post-processing

Implementation of a flexible environment within a toolbox

➢ **Toolbox** implemented to test multiple resolution strategy through a channel

(**) https://github.com/FEniCS

Field
- Space function
- Dirichlet, Load
- Mesh features
- parameters

Displacement
Damage

Elliptic
Parabolic
Hyperbolic

Formulation

Implicit method
(Newton Raphson)
Explicit integrator

Numerical resolution strategy

Metric error estimation
Balance energy
Method of crack tracking a(t)

Post-processing

Constitutive law (*) :
- **traction/compression splitting law**

(Deviatoric/volumetric, spectral and orthogonal decomposition)

(*) https://github.com/thelfer/

$[\boldsymbol{\nabla u}]^{k-1}, [\boldsymbol{\varepsilon}]^{k-1}$

Solver

MFront/MGIS

$\Psi([\varepsilon], d)^k, \left(\frac{\partial \Psi}{\partial [\boldsymbol{\varepsilon}]}\right)^k, \left(\frac{\partial^2 \Psi}{\partial [\boldsymbol{\varepsilon}]^2}\right)^k$

(∗) MFront/MGIS : A code generator for material constitutive models knowledge

Benchmark with reference solution and **sensitivity analysis**
- Mean square error estimation on physical quantities

Implementation Coupling with FEniCSx


```python
def set_mfront_environnement(self):
    hypothesis = mgis_bv.Hypothesis.PlaneStrain
    split_model_path = self.problem.parameters.mfront.ngy
    self.mfront_behaviour = mgis_bv.load(
                        split_model_path.lib,
                        split_model_path.behavior_label,
                        hypothesis)
    local_ncells = self.problem.msh.topology.index_map(
                self.problem.msh.topology.dim).size_local
    num_ghost_cell = self.problem.msh.topology.index_map(
                self.problem.msh.topology.dim).num_ghosts
    ngauss  = 4*(local_ncells + num_ghost_cell)
    self.mfront_mdm = mgis_bv.MaterialDataManager(
                self.mfront_behaviour, ngauss)


    for s in [self.mfront_mdm.s0, self.mfront_mdm.s1]:
        mgis_bv.setMaterialProperty(s, "YoungModulus",
                                    self.problem.mp.E)
        mgis_bv.setMaterialProperty(s, "PoissonRatio",
                                    self.problem.mp.nu)
        mgis_bv.setExternalStateVariable(s,"Temperature",
                                         293.15e0)
        mgis_bv.setExternalStateVariable(s, "Damage",0e0)
        psi_0 = self.thresold_psi0()
        mgis_bv.setExternalStateVariable(s,
                        "ThresholdEnergyDensity", psi_0)
```

https://github.com/thelfer/MFrontGenericInterfaceSupport/tree/master/bindings/python/mgis

```python
def sigma_from_mfront(self):
    mgis_bv.update(self.mfront_mdm)
    self.quadrature_local_projection(self.problem.msh,
            epsilon_mfront(self.fp.u_cg),
            self.fp.strain_quad.function_space,
            self.fp.strain_quad)

    self.mfront_mdm.s1.gradients[:,:] =
        self.fp.strain_quad.x.array.reshape(
        (self.mfront_mdm.n,
        self.fp.stress_quad.ufl_shape[0]))

    res = mgis_bv.integrate(self.mfront_mdm,
                    optTgOp,
                    self.problem.time_stepper.dt,
                    0,
                    self.mfront_mdm.n)

    self.set_stress(
        self.mfront_mdm.s1.thermodynamic_forces.flatten()
)
```

```python
def set_external_data_into_mfront(self):
    self.quadrature_local_projection(self.problem.msh,
                    self.fp.alpha_cg,
                    self.fp.alpha_quad.function_space,
                    self.fp.alpha_quad)
    mgis_bv.setExternalStateVariable(self.mfront_mdm.s1,
                'Damage', self.fp.alpha_quad.x.array,
    mgis_bv.MaterialStateManagerStorageMode.LocalStorage)
```

Implementation Coupling with FEniCSx


```
    def set_mfront_environnement(self):
        hypothesis = mgis_bv.Hypothesis.PlaneStrain
        split_model_path = self.problem.parameters.mfront.ngy
        self.mfront_behaviour = mgis_bv.load(
                            split_model_path.lib,
                            split_model_path.behavior_label,
                            hypothesis)
        local_ncells = self.problem.msh.topology.index_map(
                    self.problem.msh.topology.dim).size_local
        num_ghost_cell = self.problem.msh.topology.index_map(
                    self.problem.msh.topology.dim).num_ghosts
        ngauss  = 4*(local_ncells + num_ghost_cell)
        self.mfront_mdm = mgis_bv.MaterialDataManager(
                    self.mfront_behaviour, ngauss)


        for s in [self.mfront_mdm.s0, self.mfront_mdm.s1]:
            mgis_bv.setMaterialProperty(s, "YoungModulus",
                                        self.problem.mp.E)
            mgis_bv.setMaterialProperty(s, "PoissonRatio",
                                        self.problem.mp.nu)
            mgis_bv.setExternalStateVariable(s,"Temperature",
                                             293.15e0)
            mgis_bv.setExternalStateVariable(s, "Damage",0e0)
            psi_0 = self.thresold_psi0()
            mgis_bv.setExternalStateVariable(s,
                            "ThresholdEnergyDensity", psi_0)
```

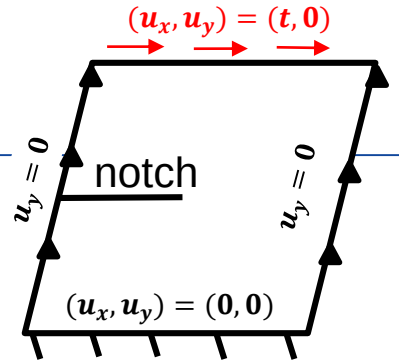
https://github.com/thelfer/MFrontGenericInterfaceSupport/tree/master/bindings/python/mgis

```
    def sigma_from_mfront(self):
        mgis_bv.update(self.mfront_mdm)
        self.quadrature_local_projection(self.problem.msh,
                epsilon_mfront(self.fp.u_cg),
                self.fp.strain_quad.function_space,
                self.fp.strain_quad)

        self.mfront_mdm.s1.gradients[:,:] =
            self.fp.strain_quad.x.array.reshape(
            (self.mfront_mdm.n,
            self.fp.stress_quad.ufl_shape[0]))

        res = mgis_bv.integrate(self.mfront_mdm,
                        optTgOp,
                        self.problem.time_stepper.dt,
                        0,
                        self.mfront_mdm.n)

        self.set_stress(
            self.mfront_mdm.s1.thermodynamic_forces.flatten()
    )
```

```
    def tangent_matrix(self):
        self.fp.K_tangent_quad.x.array[:]= self.mfront_mdm.K.
    flatten()

    def history_miehe(self):
        self.fp.H_quad.x.array[:] = self.mfront_mdm.s1.
    internal_state_variables[:,0]
```

Outline

First numerical results (Quasi-static regime)



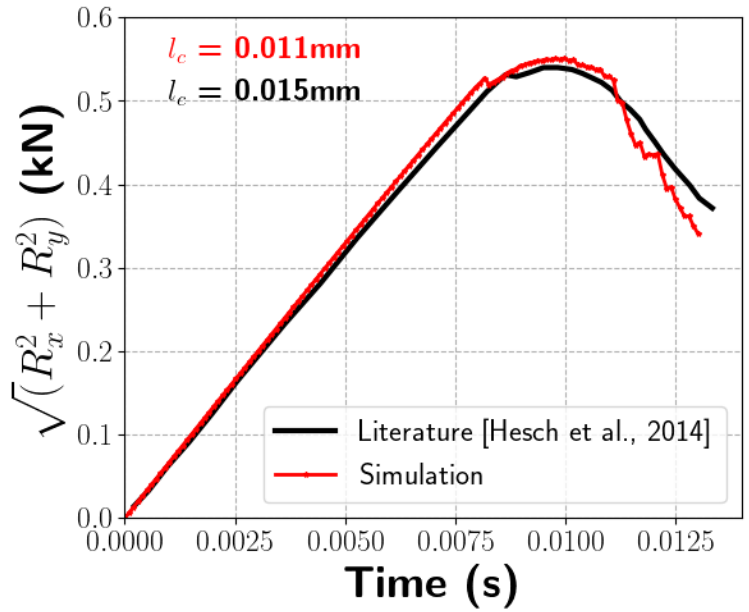
➢ Single edge notch shear test (SENS)

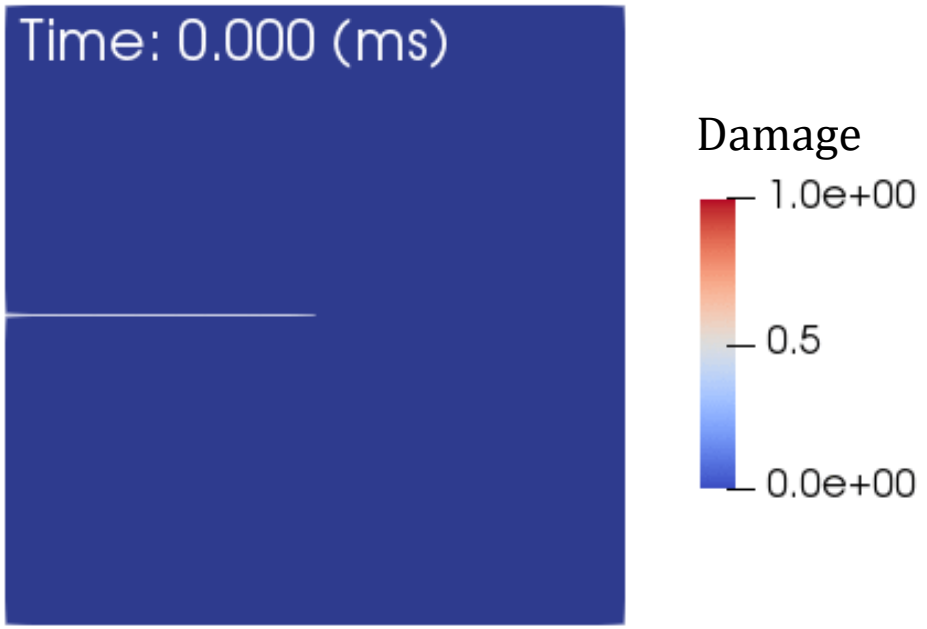
Strategy : Elliptic PDE (u-problem) and Elliptic PDE (d-problem) ➔ Implicit / Implicit

Mesh features : 256*256 elements,

Loading step : $\Delta u_x = 10^{-4} mm$, $\Delta u_x = 10^{-6} mm$

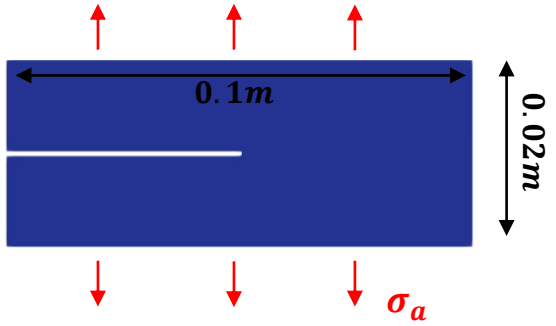
Spectral decomposition split *[Miehe et al., 2010]*





Dynamic simulation cases

Crack branching



$Mesh$: ~ 1.02 million of elements, $h_{mesh} \approx 10^{-5}m$

Constant given strength : $\sigma_a = 300$ MPa

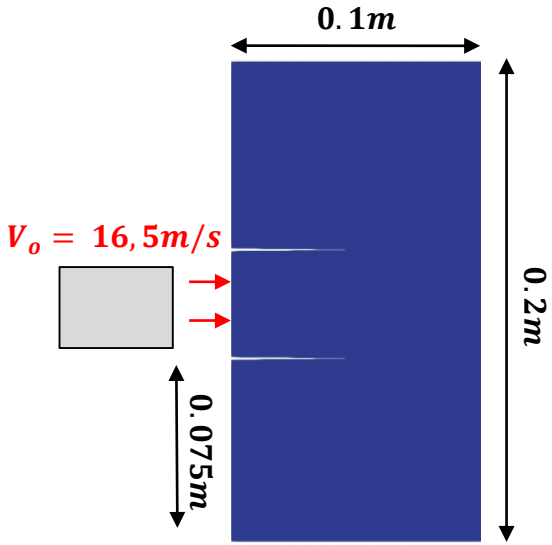
Steel material parameters :

$E = 190GPa$, $\rho = 8000kg/m^3$,
$\nu = 0.3$, $G_c = 22130J/m$

Traction/compression asymmetry : Orthogonal decomposition

Internal length $l_c = 3\, h_{mesh}$

Kalthoff and Winkler test *[Kalthoff et al., 2000]*

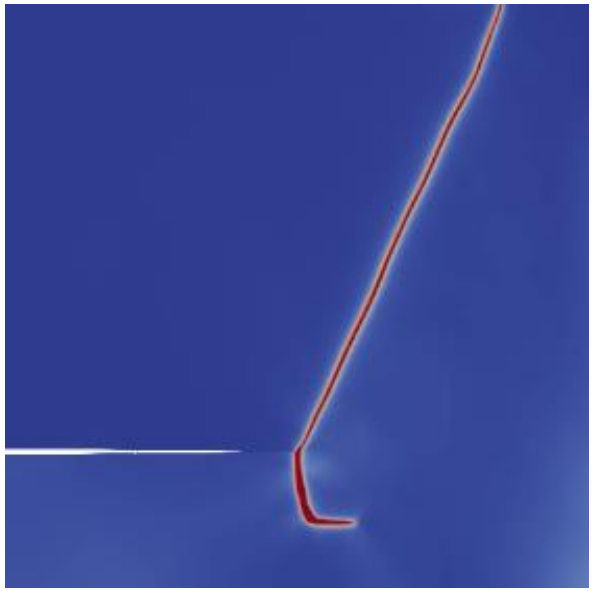


$Mesh$: ~ 1 million of elements, $h_{mesh} \sim 10^{-4}m$,

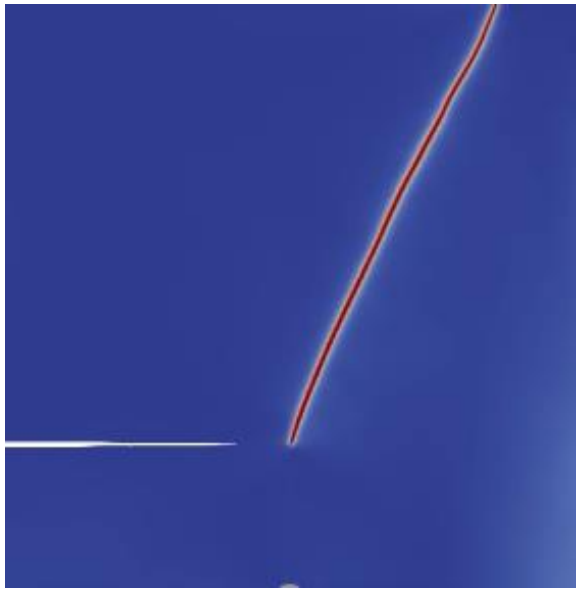
Given displacement :

$$\overrightarrow{u_a}.\vec{x} = \begin{cases} \frac{1}{2}\frac{t^2}{T_o}V_o\,, t < T_o \\ V_o(t - T_o) + \frac{1}{2}T_oV_o, \quad t \geq T_o \end{cases}$$

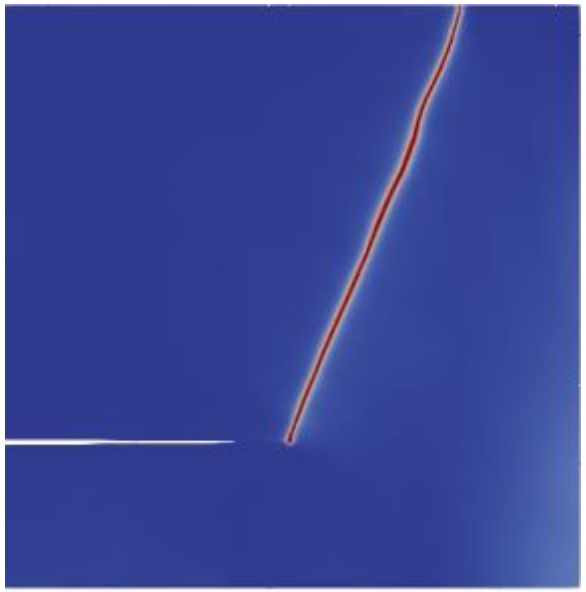
Comparison of the energy density splits



Spherical/Deviatoric split

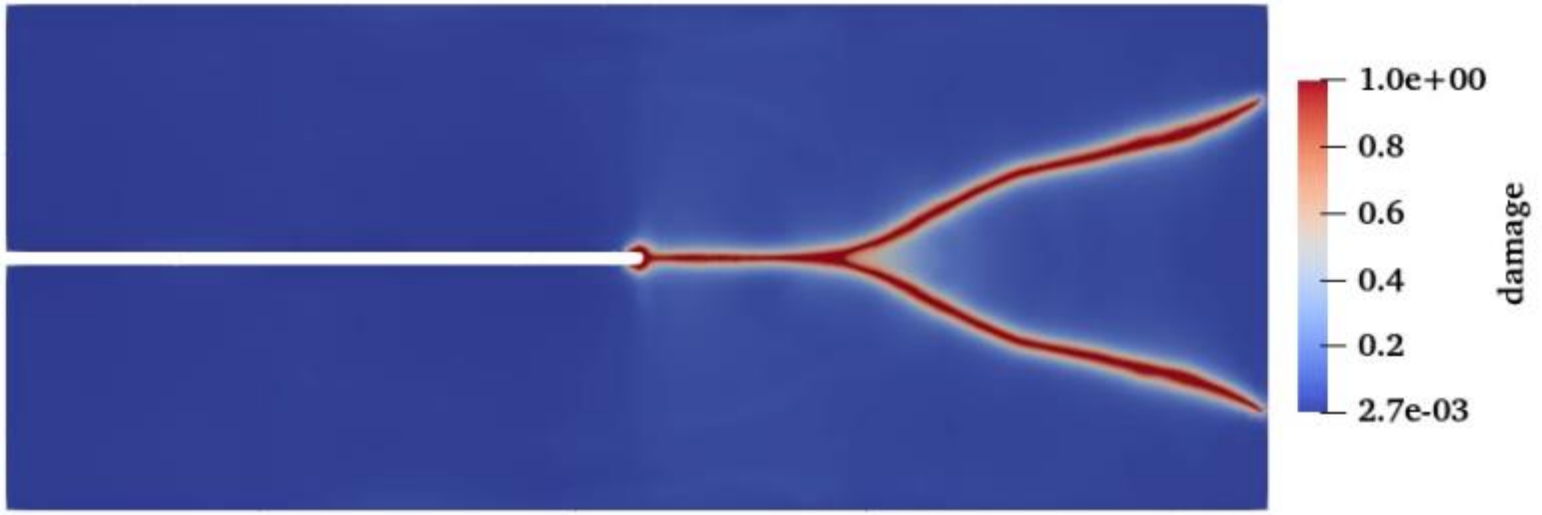


Spectral decomposition on the strain



Orthogonal decomposition (Spectral decomposition on the elastic tensor)

Crack branching – orthogonal split



Discussion and perspectives

- Implementation on FEniCSx (0.6 version) coupling with Mfront/MGIS codes (4.2.0 version)

Extension or improvement :

- Extension of the orthogonal split to higher order of material anisotropy (triclinic, monoclinic …)
- Only a unique damage scalar used ➔ Damage anisotropy (tensorial form) not involve
- Implicit resolution not perform (Tangent operator not computed in my case)

Challenge

- Big challenge mixing transient regime and anisotropic material

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

cea

GeM
Institut de Recherche en Génie Civil et Mécanique

ONERA
THE FRENCH AEROSPACE LAB

Thank you for your attention

References

- Francfort, G. A., & Marigo, J. J. (1998). Revisiting brittle fracture as an energy minimization problem. Journal of the Mechanics and Physics of Solids, 46(8), 1319-1342.
- Bourdin, B., Francfort, G. A., & Marigo, J. J. (2000). Numerical experiments in revisited brittle fracture. Journal of the Mechanics and Physics of Solids, 48(4), 797-826.
- Doitrand, A., Molnár, G., Estevez, R., & Gravouil, A. (2023). Strength-based regularization length in phase field fracture. Theoretical and Applied Fracture Mechanics, 103728.
- Amor, H., Marigo, J. J., & Maurini, C. (2009). Regularized formulation of the variational brittle fracture with unilateral contact: Numerical experiments. Journal of the Mechanics and Physics of Solids, 57(8), 1209-1229.
- Miehe, C., Hofacker, M., & Welschinger, F. (2010). 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), 2765-2778.
- Frémond, M., & Nedjar, B. (1993). Damage and principle of virtual power. *Comptes Rendus de l'Academie des Sciences, Serie II*, *317*, 857-864.
- Kamensky, D., Moutsanidis, G., & Bazilevs, Y. (2018). Hyperbolic phase field modeling of brittle fracture: Part I—theory and simulations. Journal of the Mechanics and Physics of Solids, 121, 81-98.