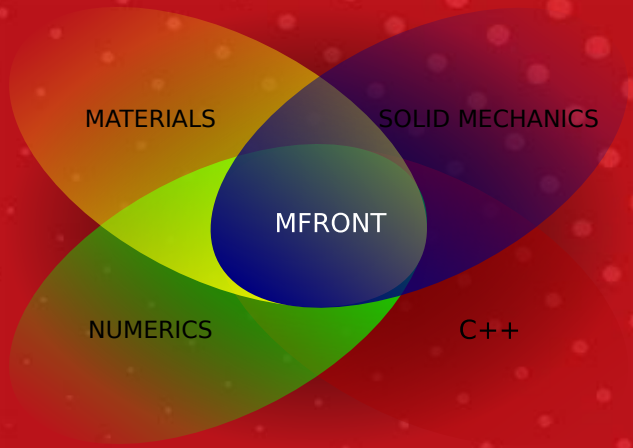
Version 3.3 of the TFEL project



DE LA RECHERCHE À L'INDUSTRIE

MFront User Days

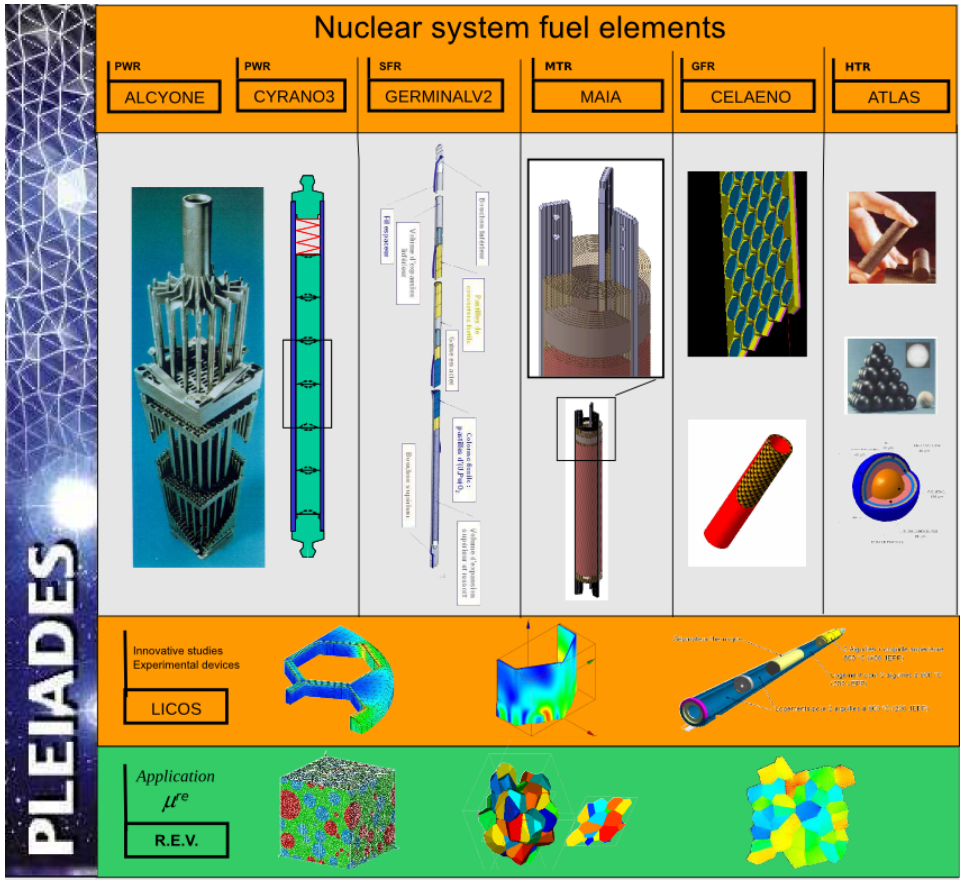
17/10/2019

Thomas Helfer

Material knowledge management in nuclear simulations

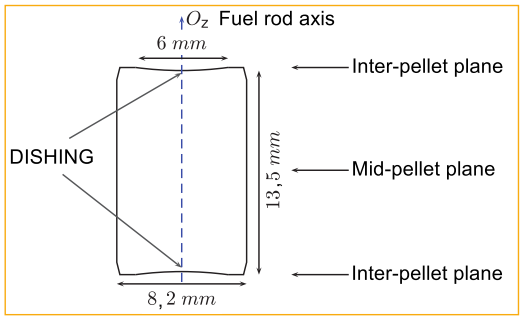
Commissariat à l'énergie atomique et aux énergies alternatives - www.cea.fr

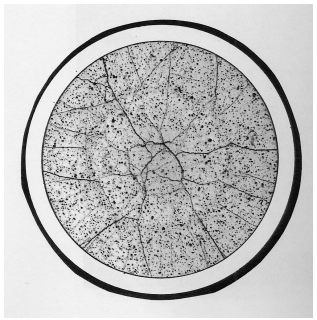
The Pleiades platform

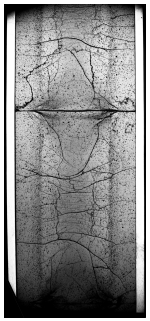


- A wide range of materials (ceramics, metals, composites).
- A wide range of mechanical phenomena and behaviours.
  - Creep, swelling, irradiation effects, phase transitions, etc..
- A wide range of mechanical loadings.

The PWR fuel element







- Brittle fracture.
- Porosity growth.
- Viscoplatic behaviour.

Material knowledge management

- The need to guarantee the quality of engineering studies has never been so high and is constantly growing.
- Every part of a study must be covered by strict AQ procedures :
  - The finite element solver on the one hand (see the `code_aster` documentation and unit tests).
  - The material knowledge (material properties, **mechanical behaviours**) and experimental data on the other hand.
- **One must guarantee a complete consistency from experimental data to engineering studies**

Overview

Role of the mechanical behaviours

- Mechanical equilibrium : find $\Delta\vec{\mathbb{U}}$ such as :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

Role of the mechanical behaviours

- Mechanical equilibrium : find $\Delta\vec{\mathbb{U}}$ such as :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

- element contribution to inner forces :

$$\begin{aligned}\vec{\mathbb{F}}_i^e &= \int_{V^e} \underline{\sigma}_{t+\Delta t}\left(\Delta\,\underline{\epsilon}^{to}, \Delta\, t\right) : \underline{\mathbf{B}}\ \mathrm{d}V \\ &= \sum_{i=1}^{N^G} \left(\underline{\sigma}_{t+\Delta\, t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_i\right), \Delta\, t\right) : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right)\right) w_i\end{aligned}$$

where $\underline{\mathbf{B}}$ gives the relationship between $\Delta\,\underline{\epsilon}^{to}$ and $\Delta\vec{\mathbb{U}}$

Role of the mechanical behaviours

- Mechanical equilibrium : find $\Delta\vec{\mathbb{U}}$ such as :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

- element contribution to inner forces :

$$\vec{\mathbb{F}}_i^e = \sum_{i=1}^{N^G} \left(\underline{\sigma}_{t+\Delta t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_i\right), \Delta t\right) : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right)\right) w_i$$

- Resolution using the Newton-Raphson algorithm :

$$\Delta\vec{\mathbb{U}}^{n+1} = \Delta\vec{\mathbb{U}}^n - \left(\left.\frac{\partial\vec{\mathbb{R}}}{\partial\Delta\vec{\mathbb{U}}}\right|_{\Delta\vec{\mathbb{U}}^n}\right)^{-1} .\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^n\right) = \Delta\vec{\mathbb{U}}^n - \underline{\underline{\mathbb{K}}}^{-1} .\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^n\right)$$

Role of the mechanical behaviours

- Mechanical equilibrium : find$\Delta\vec{\mathbb{U}}$ such as :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

- element contribution to inner forces :

$$\vec{\mathbb{F}}_i^e = \sum_{i=1}^{N^G} \left(\underline{\sigma}_{t+\Delta t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_i\right), \Delta t\right) : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right)\right) w_i$$

- Resolution using the Newton-Raphson algorithm :

$$\Delta\vec{\mathbb{U}}^{n+1} = \Delta\vec{\mathbb{U}}^n - \underline{\underline{\mathbb{K}}}^{-1}.\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^n\right)$$

- element contribution to the stiffness :

$$\underline{\underline{\mathbb{K}}}^e = \sum_{i=1}^{N^G} {}^t\underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right) : \frac{\partial\Delta\underline{\sigma}}{\partial\Delta\underline{\epsilon}^{to}}\left(\vec{\eta}_i\right) : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right) w_i$$

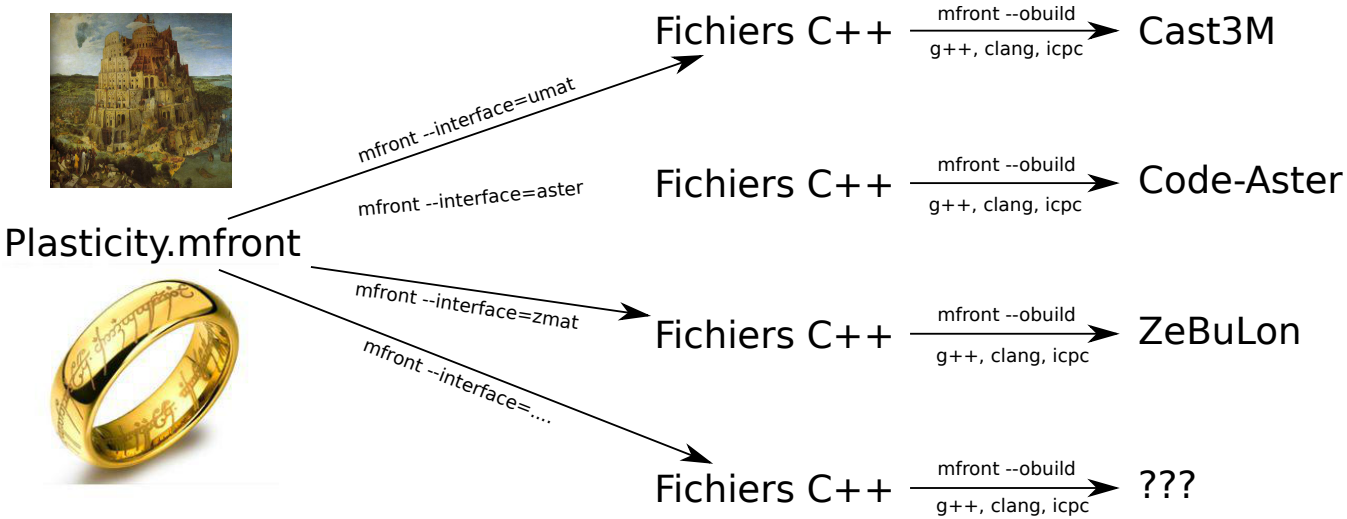
$\frac{\partial\Delta\underline{\sigma}}{\partial\Delta\underline{\epsilon}^{to}}$ is the **consistent tangent operator**

Main functions of the mechanical behaviour

$$\left(\underline{\epsilon}^{to}\big|_t \, , \vec{Y}\Big|_t \, , \Delta \underline{\epsilon}^{to} \, , \Delta t \right) \underbrace{\Longrightarrow}_{behaviour} \left(\underline{\sigma}|_{t+\Delta t} \, , \vec{Y}\Big|_{t+\Delta t} \, , \frac{\partial \Delta \underline{\sigma}}{\partial \Delta \underline{\epsilon}^{to}} \right)$$

- Given a strain increment $\Delta \underline{\epsilon}^{to}$ over a time step Δt, the mechanical behaviour must compute :
  - The value of the stress $\underline{\sigma}|_{t+\Delta t}$ at the end of the time step.
  - The value of internal state variables, noted $\vec{Y}\Big|_{t+\Delta t}$ at the end of the time step.
  - The consistent tangent operator : $\dfrac{\partial \Delta \underline{\sigma}}{\partial \Delta \underline{\epsilon}^{to}}$
- For specific cases, the mechanical behaviour shall also provide :
  - a prediction operator
  - the elastic operator (Abaqus-Explicit, Europlexus)
  - estimation of the stored and dissipated energies (Abaqus-Explicit)

MFront goals



- `TFEL` is a project various libraries/softwares, including `MFront`.
- `MFront` is a code generation tool dedicated to material knowledge (material properties, mechanical behaviours, point-wise models) with focus on :
  - Numerical efficiency (see various benchmarks).
  - Portability (`Cast3M`, `Cyrano`, `code_aster`, `Europlexus`, `TMFTT`, `AMITEX_FFTP`, `Abaqus`, `CalculiX`, `MTest`).
  - **Ease of use** : *Longum iter est per praecepta, breve et efficax per exempla* (It's a long way by the rules, but short and efficient with examples).

An very simple example

```
@DSL       Implicit;
@Behaviour Norton;
@Brick StandardElasticity;

@MaterialProperty stress young;
@MaterialProperty real nu,A,E;
young.setGlossaryName("YoungModulus");
nu.setGlossaryName("PoissonRatio");
A.setEntryName("NortonCoefficient");
E.setEntryName("NortonExponent");

@StateVariable strain p;
p.setGlossaryName("EquivalentViscoplasticStrain");

@Integrator{
  const real  eps = 1.e-12;
  const auto mu  = computeMu(young,nu);
  const auto seq = sigmaeq(sig);
  const auto tmp = A*pow(seq,E-1.);
  const auto df_dseq = E*tmp;
  const auto iseq = 1/max(seq,eps*young);
  const Stensor n = 3*deviator(sig)*iseq/2;
  // implicit system
  // the StandardElasticity already set feel to deel-deto
  feel += dp*n;
  // by default, fp is initialized to dp
  fp    -= tmp*seq*dt;
  // jacobian
  dfeel_ddeel += 2*mu*theta*dp*iseq*(Stensor4::M()-(n^n));
  dfeel_ddp    = n;
  dfp_ddeel    = -2*mu*theta*df_dseq*dt*n;
} // end of @Integrator
```

- Implicit integration.
- Implicit system :

$$\begin{cases} f_{\underline{\epsilon}^{el}} = \Delta\,\underline{\epsilon}^{el} - \Delta\,\underline{\epsilon}^{to} + \Delta\,p\,\underline{\mathbf{n}} \\ f_p = \Delta\,p - A\,\sigma_{eq}^{n} \end{cases}$$

- Jacobian :

$$\begin{cases} \dfrac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta\,\underline{\epsilon}^{el}} = \underline{\underline{\mathbf{I}}} + \dfrac{2\,\mu\,\theta\,\Delta\,p}{\sigma_{eq}}\left(\underline{\underline{\mathbf{M}}} - \underline{\mathbf{n}}\otimes\underline{\mathbf{n}}\right) \\ \dfrac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta\,p} = \underline{\mathbf{n}} \\ \dfrac{\partial f_p}{\partial \Delta\,\underline{\epsilon}^{el}} = -2\,\mu\,\theta\,A\,n\,\sigma_{eq}^{n-1}\,\Delta\,t\,\underline{\mathbf{n}} \end{cases}$$

- **All programming and numerical details are hidden (by default).**

Other functions of the mechanical behaviour

- Provide a estimation of the next time step for time step automatic adaptation
- Check bounds :
  - Physical bounds
  - Standard bounds
- Clear error messages
- Parameters
  - It is all about AQ !
  - Parametric studies, identification, etc. . .
- Generate 'MTest' files on integration failures
- Generated example of usage :
  - Generation of MODELISER/MATERIAU instructions (Cast3M)
  - Input file for Abaqus, Ansys
- Provide information for dynamic resolution of inputs (MTest/Aster/Europlexus) :
  - Numbers Types (scalar, tensors, symmetric tensors)
  - Entry names /Glossary names. . .

Available behaviour DSLs

- `Default` family :
  - `Default`, `DefaultFiniteStrain`, `DefaultCZM`
  - those DSLs are the most generic ones as they do not make any restriction on the behaviour or the integration method that may be used.
- Specialized DSL's :
  - `IsotropicMisesCreep`, `IsotropicMisesPlasticFlow`, `IsotropicStrainHardeningMisesCreep`
- `Runge-Kutta` family :
  - `RungeKutta`, `RungeKuttaFiniteStrain`
  - `euler`, `rk2`, `rk4`, `rk42`, `rk54`, `rkCastem`
- `Implicit` family :
  - `Implicit`, `ImplicitII`, `ImplicitFiniteStrain`
  - Various algorithms : `NewtonRaphson`, `NewtonRaphson_NumericalJacobian`, `Broyden`, `Broyden2`, `PowellDogLeg_NewtonRaphson`, `PowellDogLeg_NewtonRaphson_NumericalJacobian`, `PowellDogLeg_Broyden`, `LevenbergMarquardt` `LevenbergMarquardt_NumericalJacobian`

Bricks

- Bricks have been introduced in `TFEL 3.0` to simplify the implementation of certain classes of mechanical behaviours.
- `mfront -list-behaviour-bricks`
  - `StandardElasticity`
  - `StandardElastoViscoPlasticity`
    - new in TFEL 3.2
    - described in depth below
  - `DDIF2`
  - `FiniteStrainSingleCrystal`

MGIS

Commissariat à l'énergie atomique et aux énergies alternatives - www.cea.fr

MFront Generic Interface Support (MGIS)

- The generic interface in MFront allows to create shared libraries usable in solvers that do not provide an existing way of adding user defined material behaviours and want to provide a more complete, efficient way to add such behaviours.

MFront Generic Interface Support (MGIS)

- The generic interface in MFront allows to create shared libraries usable in solvers that do not provide an existing way of adding user defined material behaviours and want to provide a more complete, efficient way to add such behaviours.
- The MGIS project provides classes on the solver side to retrieve metadata from an MFront behaviour and call the behaviour integration over a time step.

MFront Generic Interface Support (MGIS)

- The generic interface in MFront allows to create shared libraries usable in solvers that do not provide an existing way of adding user defined material behaviours and want to provide a more complete, efficient way to add such behaviours.
- The MGIS project provides classes on the solver side to retrieve metadata from an MFront behaviour and call the behaviour integration over a time step.
- Two level of integration :
  - around a single integration point
  - around a group of integration points (material)

MFront Generic Interface Support (MGIS)

- The generic interface in MFront allows to create shared libraries usable in solvers that do not provide an existing way of adding user defined material behaviours and want to provide a more complete, efficient way to add such behaviours.
- The MGIS project provides classes on the solver side to retrieve metadata from an MFront behaviour and call the behaviour integration over a time step.
- Two level of integration :
  - around a single integration point
  - around a group of integration points (material)
- Written in 'C++'. Bindings exists for 'C', 'Fortran2003', 'python', 'Julia'

Some new features

UTF-8 support

```
@Integrator{
  const auto σᵉ = sigmaeq(σ);
  const auto iσᵉ = 1 / (max(σᵉ, real(1.e-12) · E));
  const auto vᵖ = A · pow(σᵉ, nn);
  const auto ∂vᵖ/∂σᵉ = nn · vᵖ · iσᵉ;
  const auto n = 3 · deviator(σ) · (iσᵉ / 2);
  // Implicit system
  fεᵉˡ += Δp · n;
  fp -= vᵖ · Δt;
  // jacobian
  ∂fεᵉˡ/∂Δεᵉˡ += 2 · μ · θ · dp · iσᵉ · (Mᵉ - (n ⊗ n));
  ∂fεᵉˡ/∂Δp = n;
  ∂fp/∂Δεᵉˡ = -2 · μ · θ · ∂vᵖ/∂σᵉ · Δt · n;
} // end of @Integrator
```

- A limited subset of UTF-8 can be used to define the names of the variables and some mathematical operations.
  - See `http://tfel.sourceforge.net/unicode.html` for details.
  - Sadly, diacritical marks are not supported properly by text editors, terminals, etc...
- See the Ramberg-Osgood non linear elasticity example.

UTF-8 support

```
E.setGlossaryName("YoungModulus");
@MaterialProperty real ν;
ν.setGlossaryName("PoissonRatio");

// Lame Coefficients
@LocalVariable stress λ, μ;

@InitLocalVariables{
  λ = computeLambda(E, ν);
  μ = computeMu(E, ν);
}

@PredictionOperator{
  Dt = λ ⋅ (I₂⊗I₂) + 2 ⋅ μ ⋅ I₄;
}

@Integrator{
  const auto e = ε^to + Δε^to;
  σ = λ ⋅ trace(e) ⋅ I₂ + 2 ⋅ μ ⋅ e;
}

@TangentOperator {
  Dt = λ ⋅ (I₂⊗I₂) + 2 ⋅ μ ⋅ I₄;
}
```

Generic behaviours - I

- Generic behaviours relates pairs of gradients and fluxes.

Generic behaviours - I

- Generic behaviours relates pairs of gradients and fluxes.
- The consistent tangent operator at least contains the derivatives of the flux with respect to the gradients.

Generic behaviours - I

- Generic behaviours relates pairs of gradients and fluxes.
- The consistent tangent operator at least contains the derivatives of the flux with respect to the gradients.
- However, new terms in the consistent tangent operator may be needed :
  - derivative of state variable with respect to a gradient (non linear heat transfer).
  - derivative of state variable with respect to an external state variable (transient heat transfer).

Generic behaviours - I

- Generic behaviours relates pairs of gradients and fluxes.
- The consistent tangent operator at least contains the derivatives of the flux with respect to the gradients.
- However, new terms in the consistent tangent operator may be needed :
  - derivative of state variable with respect to a gradient (non linear heat transfer).
  - derivative of state variable with respect to an external state variable (transient heat transfer).
- Currently limited by the fact that some mathematical objects are not defined yet :
  - derivative of vectors with respect to symmetric tensors, non symmetric tensors...

Generic behaviours - I

- Generic behaviours relates pairs of gradients and fluxes.
- The consistent tangent operator at least contains the derivatives of the flux with respect to the gradients.
- However, new terms in the consistent tangent operator may be needed :
  - derivative of state variable with respect to a gradient (non linear heat transfer).
  - derivative of state variable with respect to an external state variable (transient heat transfer).
- Currently limited by the fact that some mathematical objects are not defined yet :
  - derivative of vectors with respect to symmetric tensors, non symmetric tensors...
- Currently only supported by the `generic` interface.

Heat transfer examples

```
a_Newton = inner(grad(v), dot(Ct, grad(T_)))∗dxm
res = −inner(grad(T_), j)∗dxm
```

```
@DSL DefaultGenericBehaviour;
@Behaviour StationaryLinearHeatTransfer;

@Gradient TemperatureGradient gT;
gT.setGlossaryName("TemperatureGradient");

@Flux HeatFlux j;
j.setGlossaryName("HeatFlux");

@MaterialProperty thermalconductivity k;
k.setGlossaryName("ThermalConductivity");

@Integrator{
  j = −k ∗ (gT + dgT);
} // end of @Integrator

@TangentOperator {
  Dt = −k ∗ tmatrix<N, N, real>::Id();
}
```

- On the left, the variational form in `FEniCS`
- On the right, the ‘MFront‘ implementation

Heat transfer examples

```
a_Newton = (inner(grad(v), dot(as_tensor(dj_ddgT), grad(T_)))+
            inner(grad(v),  as_vector(dj_ddT))*T_)*dxm
res = -inner(grad(v), j)*dxm
```

```
@DSL DefaultGenericBehaviour;
@Behaviour StationaryNonLinearHeatTransfer;

@Gradient TemperatureGradient gT;
gT.setGlossaryName("TemperatureGradient");

@Flux HeatFlux j;
j.setGlossaryName("HeatFlux");

@AdditionalTangentOperatorBlock dj_ddT;

@LocalVariable thermalconductivity k;

@Integrator{
  constexpr const auto A = 0.0375;
  constexpr const auto B = 2.165e-4;
  const auto T_ = T + dT;
  k = 1 / (A + B * T_);
  j = -k * (gT + dgT);
} // end of @Integrator

@TangentOperator {
  dj_ddgT = -k * tmatrix<N, N, real>::Id();
  dj_ddT =  B * k * k * (gT + dgT);
} // end of @TangentOperator
```

- On the left, the variational form in `FEniCS`
- On the right, the 'MFront' implementation

Heat transfer examples

```
a_Newton = -(v*dH1_ddT*T_-
                dt*theta*(inner(grad(v), dot(as_tensor(dj1_ddgT),
          grad(T_)))+
                          inner(grad(v),  as_vector(dj1_ddT))*T_))
          *dxm
res = (v*(H1-H0)-dt*(theta*inner(grad(v), j1)+(1-theta)*
          inner(grad(v), j0)))*dxm
```

```
...
@StateVariable real H;
H.setEntryName("Enthalpy");
@AdditionalTangentOperatorBlock dH_ddT;
...

@Integrator {
  const auto T_ = T + dT;
  k = (T_<Tm) ? ks :  kl;
  j = -k*(gT+dgT);
  // enthalpy
  if (T_<Tm){
    Ce = Cs;
    H = Cs*T_;
  } else {
    Ce = Cl;
    H = Cl*(T_-Tm)+dHsl+Cs*Tm;
  }
}  // end of @Integrator

@TangentOperator {
  dj_ddgT = -k * tmatrix<N, N, real>::Id();
  dj_ddT = tvector<N, real>(real(0));
  dH_ddT = Ce;
}  // end of @TangentOperator
```

- On the left, the variational form in `FEniCS`
- On the right, the 'MFront' implementation

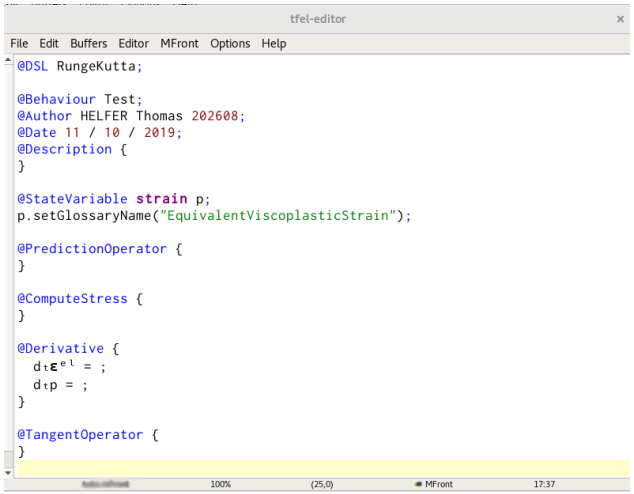
Heat transfer examples

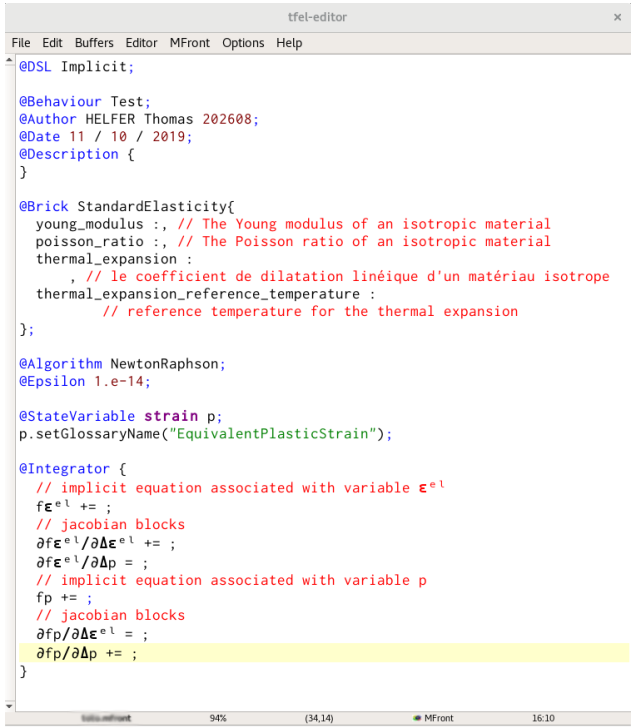
```
a_Newton = -(v*dH1_ddT*T_-
                dt*theta*(inner(grad(v), dot(as_tensor(dj1_ddgT),
          grad(T_)))+
                          inner(grad(v),  as_vector(dj1_ddT))*T_))
          *dxm
res = (v*(H1-H0)-dt*(theta*inner(grad(v), j1)+(1-theta)*
          inner(grad(v), j0)))*dxm
```

```
...
@StateVariable real H;
H.setEntryName("Enthalpy");
@AdditionalTangentOperatorBlock dH_ddT;
...

@Integrator {
  const auto T_ = T + dT;
  k = (T_<Tm) ? ks :  kl;
  j = -k*(gT+dgT);
  // enthalpy
  if (T_<Tm){
    Ce = Cs;
    H = Cs*T_;
  } else {
    Ce = Cl;
    H = Cl*(T_-Tm)+dHsl+Cs*Tm;
  }
}  // end of @Integrator

@TangentOperator {
  dj_ddgT = -k * tmatrix<N, N, real>::Id();
  dj_ddT = tvector<N, real>(real(0));
  dH_ddT = Ce;
}  // end of @TangentOperator
```

- On the left, the variational form in `FEniCS`
- On the right, the 'MFront' implementation

The behaviour wizard of ‘tfel-editor‘

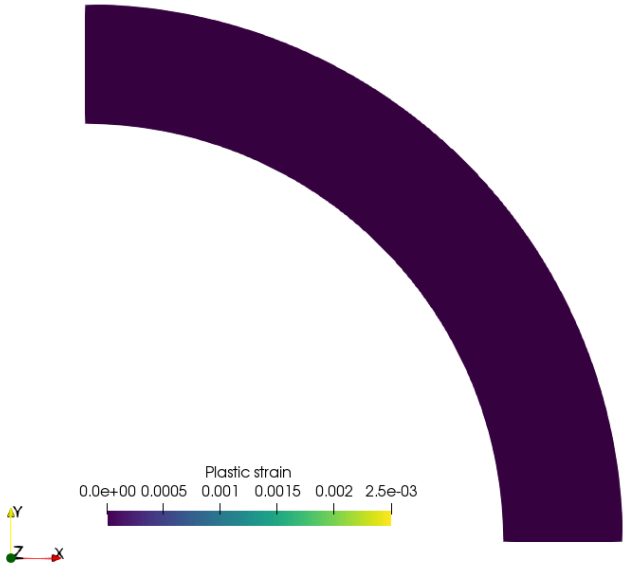


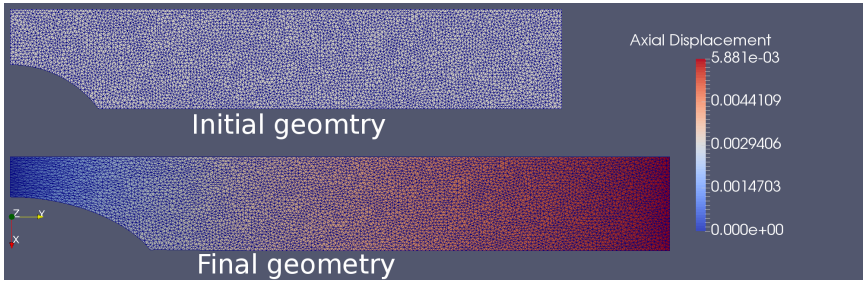


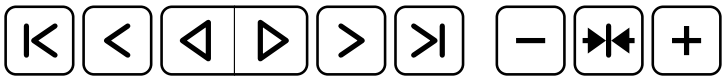
- The behaviour wizard of ‘tfel-editor‘ allows creating templates of ‘MFront‘ files.

Highlights

‘FEniCS‘

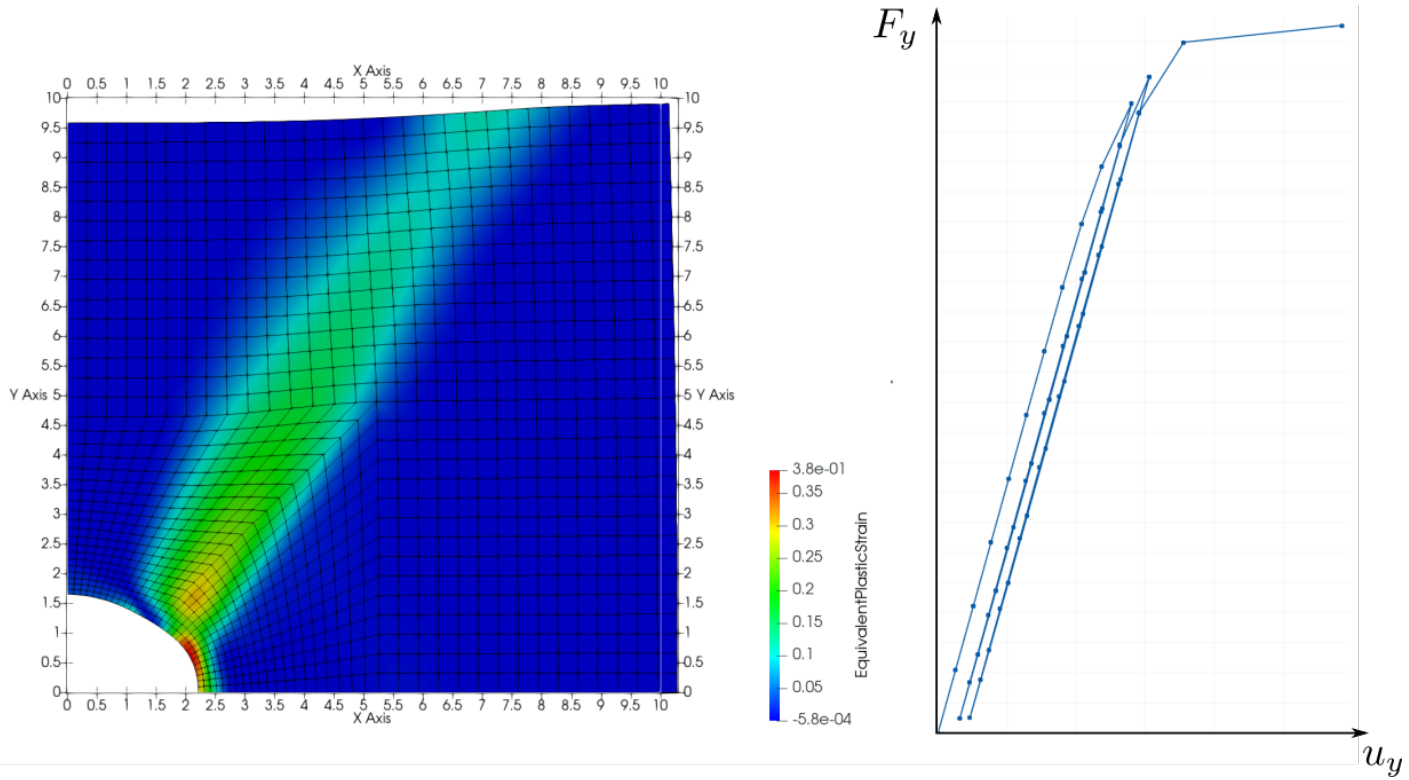




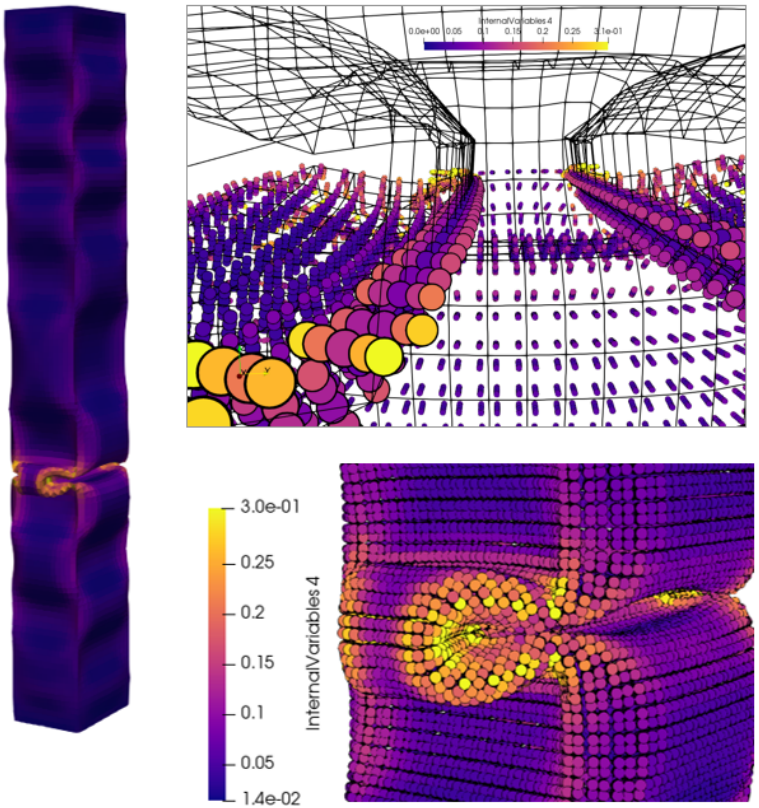


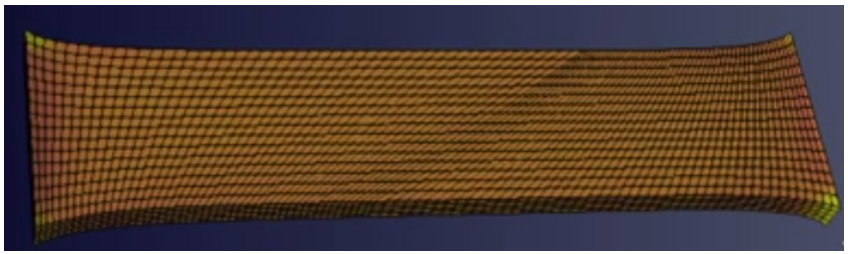
- *Elasto-plastic analysis implemented using the MFront code generator* : `https://comet-fenics.readthedocs.io/en/latest/demo/plasticity_mfront/plasticity_mfront.py.html`
- See J. Bleyer’ talk this afternoon.

‘OpenGeoSys‘

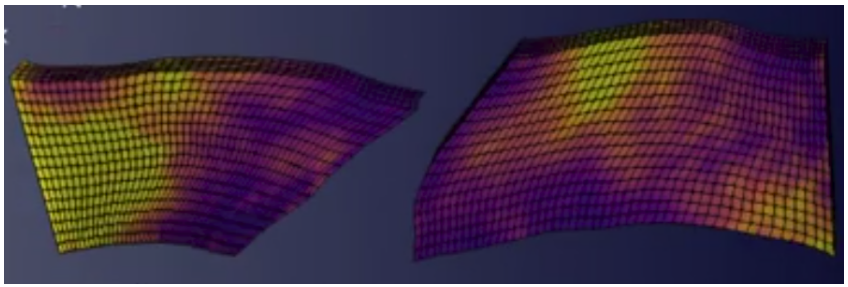


- Based on `MGIS’ C` bindings.
- See T. Nagel this afternoon
- Very nice contribution with an implementation of the Mohr-Coulomb plastic criterion.





Hyperelastic strip : Loading (implicit)



Hyperelastic strip : Post-rupture behaviour (explicit)

- A CEA' prototypical solver targeting exascale computations.
- Based on `MGIS`.
- See O. Jamond' talk this afternoon.

Julia bindings and 'MFrontInterface.jl'

MFrontInterface testing

This code tests `MFrontInterface` for a single material point. The strain history is predefined and supplied to the MFront interface to get stress response.

Shared library `libBehaviour.so` was obtained using MFront Generic interface with the following command

```
mfront --interface=generic --obuild=level0 StandardElastoViscoPlasticityPlasticityTest11.mfron
t
```

`--obuild` argument tells MFront to generate the shared library `libBehaviour.so`.

`level0` is used to switch off machine specific optimization, to obtain portable `.so` file.

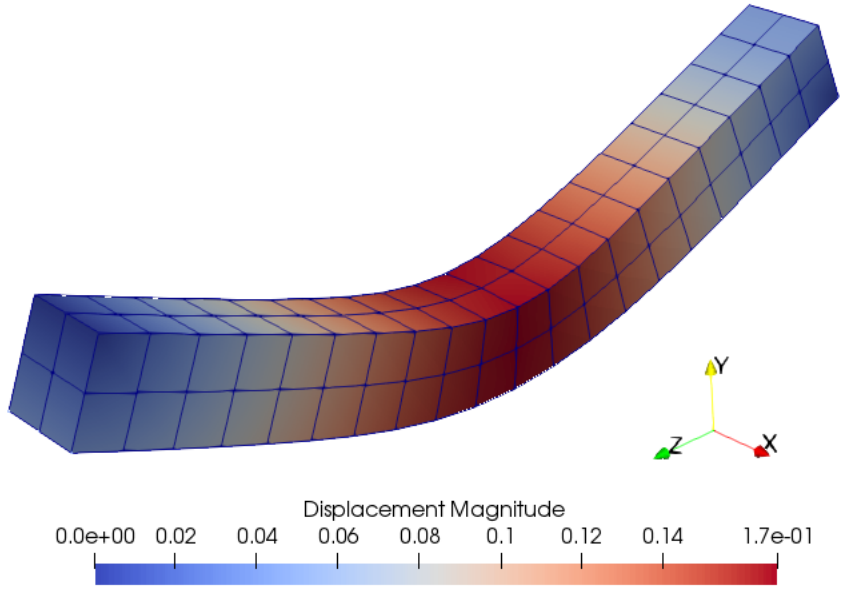
The MFront routine was taken here https://github.com/thelfer/MFrontGenericInterfaceSupport/blob/master/tests/StandardElastoViscoPlasticityPlasticityTest11.mfront

In [1]:
```
using MFrontInterface, PyPlot
```

In [2]:
```
mgis_bv = MFrontInterface.behaviour
```

Out[2]: MFrontInterface.behaviour

In [3]:
```
# Define strain history
e11 = vcat(Array(range(0, stop=1.5e-2, length=30)), Array(range(1.5e-2, stop=-1.5e-2, length=30)), Ar
ray(range(-1.5e-2, stop=3e-2, length=40)))
e22 = -e11/2
e33 = e22
strains = [[e11[i], e22[i], e33[i], 0.0, 0.0, 0.0] for i in 1:100]
plot(strains)
xlabel("Time step")
ylabel("Strain")
grid()
show()
```



- Based on `MGIS' Julia` bindings.
- Installation is, as easy as any 'Julia' package :

```
pkg> add MFrontInterface
```

Fichant La Borderie damage model coupled with Load Induced Thermal Strain (LITS)

$$D = 1 - \frac{\varepsilon_{d0}}{\varepsilon_{eq}} exp\{B_t(\varepsilon_{d0} - \varepsilon_{eq})\}$$

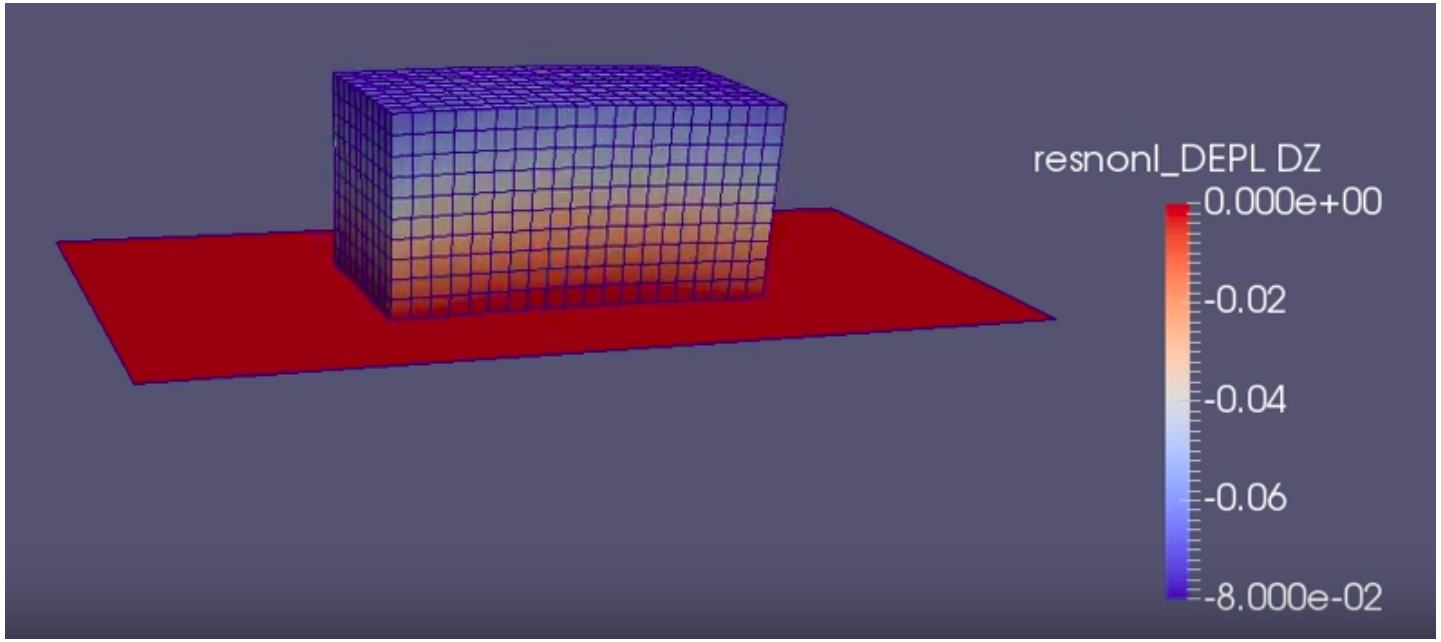
$$\varepsilon_{eq} = \sqrt{\sum_{i=1}^{3} \langle \varepsilon_i^e \rangle^2}$$

$$B_t = \frac{f_t h}{G_t - 0.5 \varepsilon_{d0} h f_t}$$

$$\sigma_{ij} = (1 - D)\langle \tilde{\sigma}_{ij} \rangle_+ + (1 - D^\alpha)\langle \tilde{\sigma}_{ij} \rangle_-$$

- Extensive testing by A. Gangnant (Comparison with build-in behaviour of `Cast3M`, then port to `code_aster`. Comparison with Mazars model).
- *Development of a Novel Damage Model for Concrete Subjected to High Temperature and Constraint*. J. Draup et al. Transactions, SMiRT-25 Charlotte, NC, USA, August 4-9, 2019

Skorohold-Olevsky Viscous Sintering model

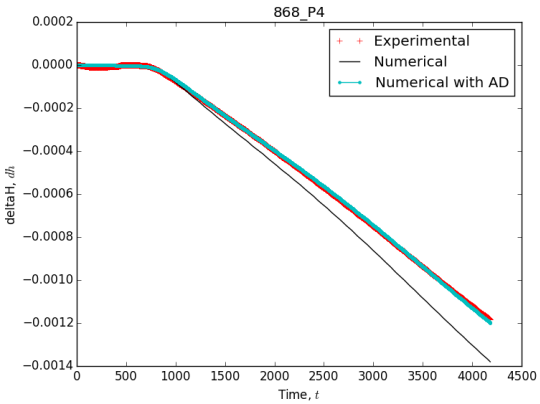


- SOVS model predicts sintering shrinkage and densification of ceramics and composites.
- Contribution from J. Balaguer (Instituto de Tecnología Cerámica).
- `http://tfel.sourceforge.net/sovs.html`

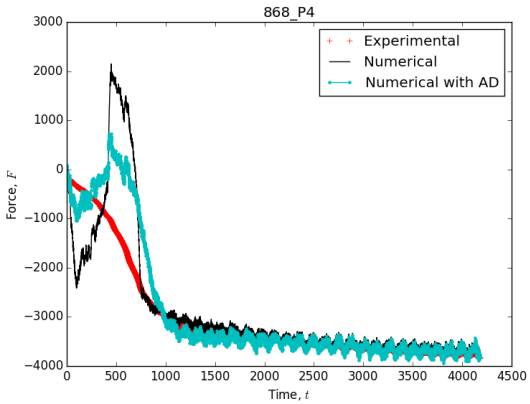
Co-development of MAP' LCOD component

- The LCOD component is an identification tool developed by EDF in the MAP project.

Co-development of MAP' LCOD component



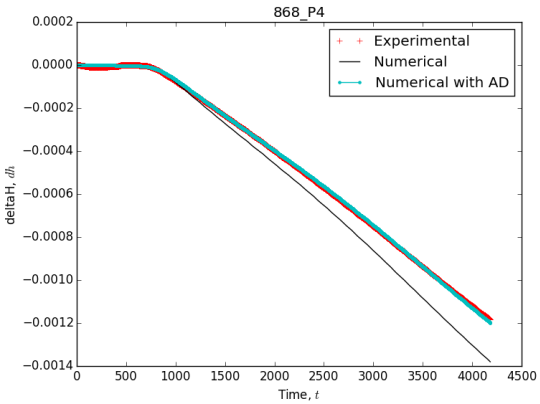
Imposed force $n \approx 6, 9$



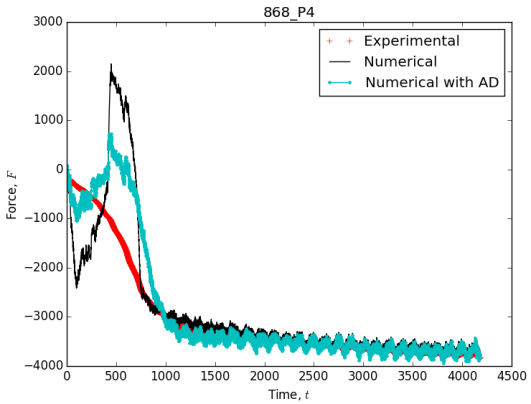
Imposed displacement $n \approx 2$

- The LCOD component is an identification tool developed by EDF in the MAP project.
- First CEA contributions :
  - Allow user defined case of study.

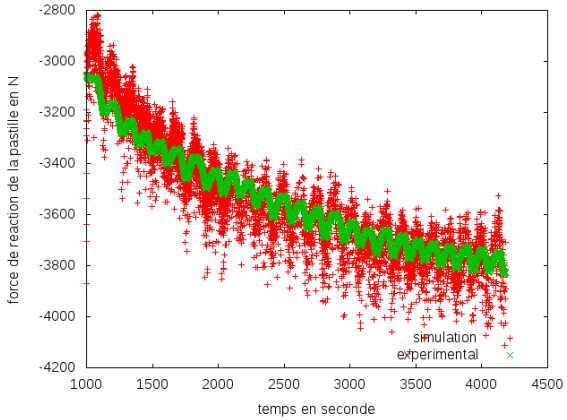
Co-development of MAP' LCOD component



Imposed force $n \approx 6,9$



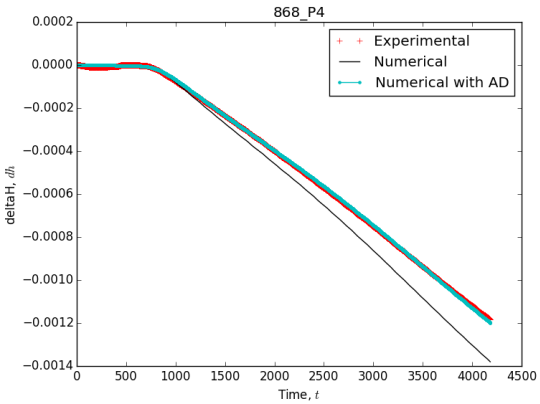
Imposed displacement $n \approx 2$



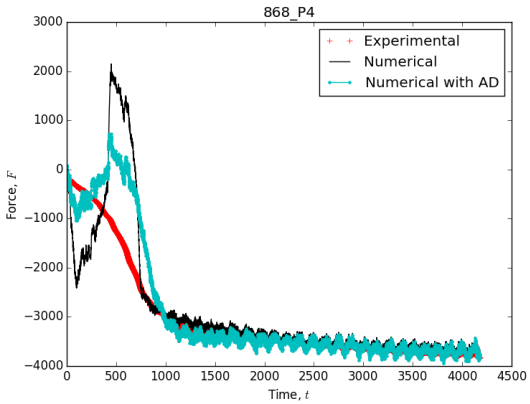
Imposed displacement and adjustement windows $n \approx 7$

- The LCOD component is an identification tool developed by EDF in the MAP project.
- First CEA contributions :
  - Allow user defined case of study.
  - Allow adding an adjustement window.

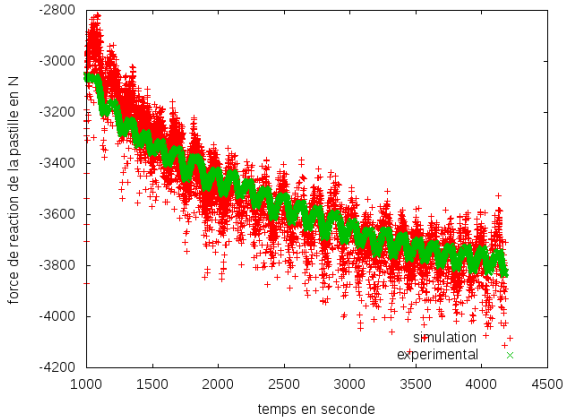
Co-development of MAP' LCOD component



Imposed force $n \approx 6, 9$



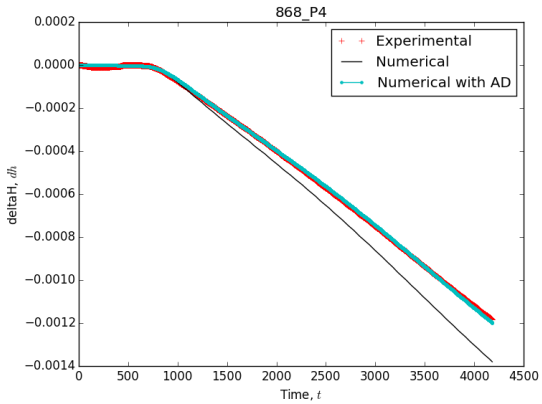
Imposed displacement $n \approx 2$



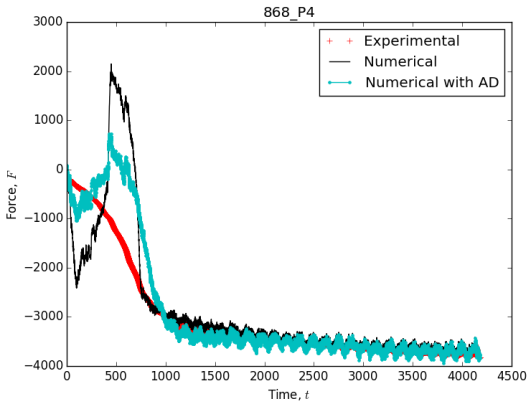
Imposed displacement and adjustement windows $n \approx 7$

- The LCOD component is an identification tool developed by EDF in the MAP project.
- First CEA contributions :
  - Allow user defined case of study.
  - Allow adding an adjustement window.
  - Allow other solvers than MTest.

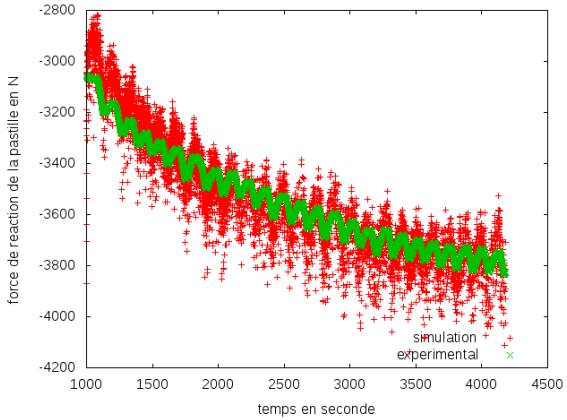
Co-development of MAP' LCOD component



Imposed force $n \approx 6, 9$



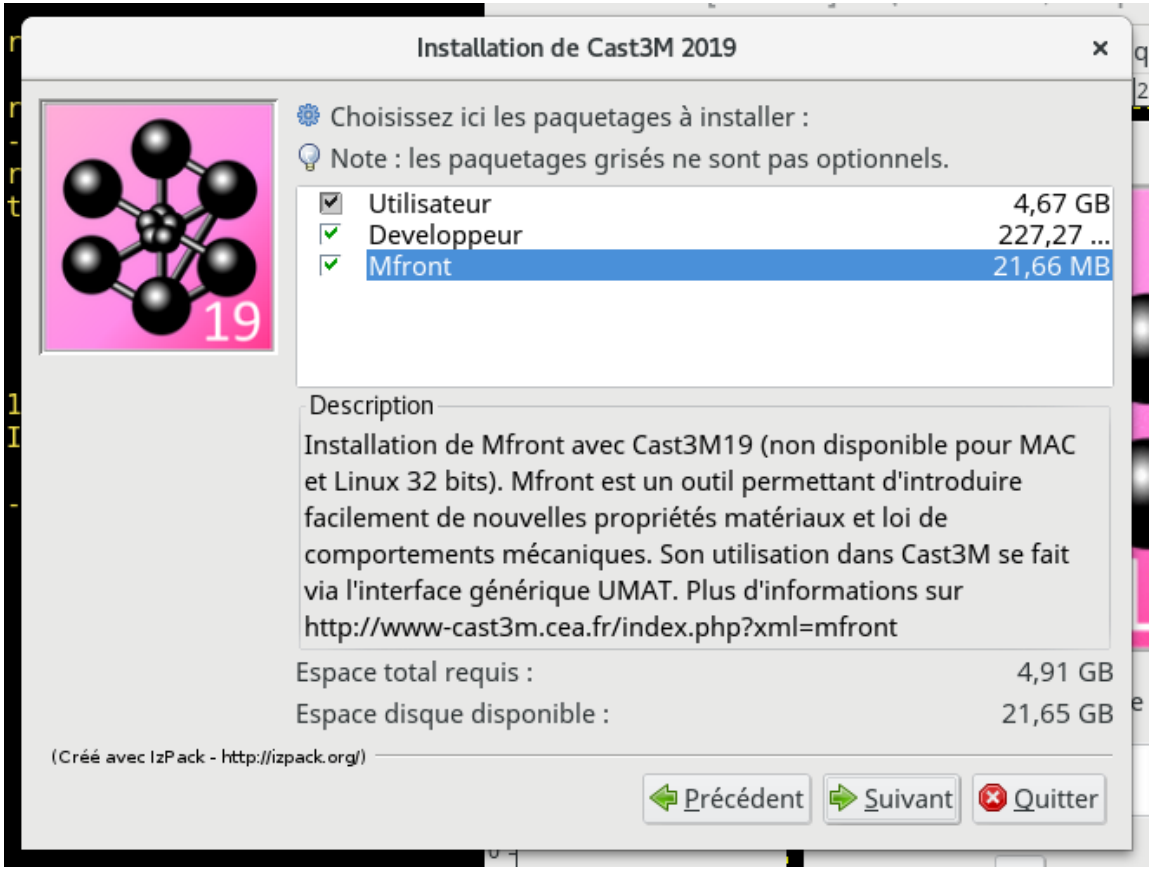
Imposed displacement $n \approx 2$



Imposed displacement and adjustement windows $n \approx 7$

- The LCOD component is an identification tool developed by EDF in the MAP project.
- First CEA contributions :
  - Allow user defined case of study.
  - Allow adding an adjustement window.
  - Allow other solvers than MTest.
  - Automatic generation of non regression tests and integration tests (see mfm-test-generator)
- Not yet open-source...

MFront now ships with Cast3M



- Thanks to C. Berthinier' work

Some publications using MFront

Bernachy-Barbe, Fabien, and Benoît Bary. 2019. "Effect of Aggregate Shapes on Local Fields in 3D Mesoscale Simulations of the Concrete Creep Behavior." *Finite Elements in Analysis and Design* 156 (April): 13–23. https://doi.org/10.1016/j.finel.2019.01.001.

Bleyer, Jérémy, and Thomas Helfer. 2019. "Elasto-Plastic Analysis Implemented Using the MFront Code Generator. Numerical Tours of Continuum Mechanics Using FEniCS." 2019. https://comet-fenics.readthedocs.io/en/latest/demo/plasticity_mfront/plasticity_mfront.py.html.

Dörfler, A., G. Feiertag, M. Schmidt, A. Ruediger, and U. Wagner. 2019. "Numerical Optimization of Thermally Induced Hysteresis Effects in the Packaging of MEMS Pressure Sensors." *IEEE Sensors Journal* 19 (10): 3633–9. https://doi.org/10.1109/JSEN.2019.2896916.

Forre, Agathe, Thomas Helfer, and Khouloud Derouiche. 2019. "Comparison of Different Implicit Integration Procedures for an Elasto-Viscoplastic Model." Oral communication. Stockholm: Oral communication. https://indico.lunarc.lu.se/event/1/contributions/125/.

Helfer, Thomas, Olivier Fandeur, Dominique Geoffroy, Charles Toulemonde, Jérémy Hure, Laurent Dupuy, Agathe Forré, et al. 2019. "New Functionalities of Versions 3.1 and 3.2 of TFEL/MFront." In, 12. Giens, Var, France: CSMA. https://csma2019.sciencesconf.org/240102/document.

Manso, J., J. Marcelino, and L. Caldeira. 2019. "Settlement Analysis of the Montesinho Concrete-Face Rockfill Dam." In *Geotechnical Engineering Foundation of the Future*. Reykjavik Iceland: ISSMGE. https://doi.org/10.32075/17ECSMGE-2019-0528.

Marano, Aldo, Lionel Gélébart, and Samuel Forest. 2019. "Intragranular Localization Induced by Softening Crystal Plasticity: Analysis of Slip and Kink Bands Localization Modes from High Resolution FFT-Simulations Results." *Acta Materialia* 175 (August): 262–75. https://doi.org/10.1016/j.actamat.2019.06.010.

Perales, F, F Dubois, Y Monerie, R Mozul, and F Babik. 2019. "Xper : une plateforme pour la simulation numérique distribuée d'in- teractions multiphysiques entre corps." In, 8. Giens, Var, France: CSMA. https://csma2019.sciencesconf.org/240138/document.

Singh, Kulbir, Christian Robertson, and Arun Kumar Bhaduri. 2019. "Brittle Fracture Model Parameter Estimation for Irradiated BCC Material Through Dislocation Based Crystal Plasticity Model." *Frattura Ed Integrità Strutturale* 13 (50): 319–30. https://doi.org/10.3221/IGF-ESIS.50.27.

Szmytka, Fabien, Pierre Osmond, Luc Rémy, Pierre-Damien Masson, Agathe Forré, and François-Xavier Hoche. 2019. "Some Recent Advances on Thermal–Mechanical Fatigue Design and Upcoming Challenges for the Automotive Industry." *Metals* 9 (7): 794. https://doi.org/10.3390/met9070794.

Tuset, Lluís, Gerard Fortuny, Joan Herrero, Dolors Puigjaner, and Josep M. López. 2019. "Implementation of a New Constitutive Model for Abdominal Muscles." *Computer Methods and Programs in Biomedicine* 179 (October): 104988. https://doi.org/10.1016/j.cmpb.2019.104988.

TFEL 3.4 and beyond

Commissariat à l'énergie atomique et aux énergies alternatives - www.cea.fr

- More work on generic behaviours
- Treatment of plasticity based on the Fischer-Burmeister complementary function :
  - $\phi(a, b) = a + b - \sqrt{a^2 + b^2}$
  - $\phi(a, b) = 0 \quad \Leftrightarrow \quad a \geq 0, \quad b \geq 0, \quad a\,b = 0$
- Homotopy methods.
- Extension of the 'StandardElastoViscoPlasticity' brick to porous behaviours.

```
template <typename StensorType>
std::enable_if_t<matchesStensorConcept<StensorType>, StensorNumType<StensorType>> sigmaeq(
      const StensorType& s) {
   using NumType = StensorNumType<T>;
   using real  =  tfel :: typetraits :: base_type<NumType>;
   constexpr const auto one = real(1);
   constexpr const auto one_third = one / 3;
   constexpr const auto cste = one / 2;
   auto square = [](const auto& v) { return v * v };
   const auto tr = one_third * trace(s);
    if
      constexpr(StensorTraits<T>::dime == 1u) {
        return std::sqrt(cste * (square(s(0) — tr) + square(s(1) — tr) + square(s(2) — tr)));
      }
   else  if  (StensorTraits<T>::dime == 2u) {
      return std::sqrt(cste *
                       (square(s(0) — tr) + square(s(1) — tr) + square(s(2) — tr) + square(s(3))));
   } else {
      return std::sqrt(cste * (square(s(0) — tr) + square(s(1) — tr) + square(s(2) — tr) +
                               square(s(3)) + square(s(4)) + square(s(5))));
   }
}
```

- Port to C++-17

Thank you for your attention. Time for discussion!

https://tfel.sourceforge.net

https://twitter.com/TFEL_MFront

https://github.com/thelfer/

tfel-contact@cea.fr