DE LA RECHERCHE À L'INDUSTRIE

MFront User Meeting: from TFEL 1.*x* to TFEL 2.0

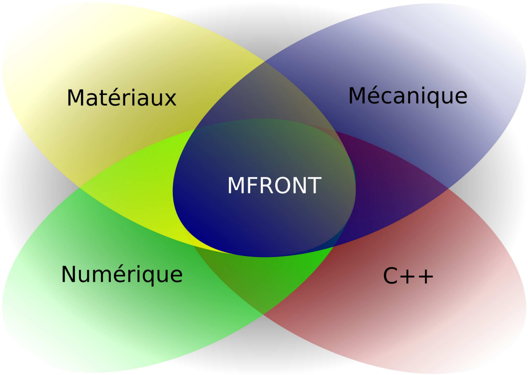
— Thomas Helfer, Jean-Michel Proix

Februar 2015

www.cea.fr

Table of contents

A tour of `MFront`


Matériaux
Mécanique
MFRONT
Numérique
C++

MFront

- `MFront` is a code generator based on C++ for :
  - material properties
  - mechanical behaviours
  - models
- `MFront` provides several *domain specific languages* :
  - ease of use, expressivness, etc..
    - focus on physical content
  - low programming skills requirements
  - numerical efficiency..

A first example : material property

```
@DSL MaterialLaw;                 // treating a material property
@Material UO2;                    // material name
@Law YoungModulus_Martin1989;     // name of the material property
@Output E;                        // output of the material property
@Input T,f;                       // inputs of the material property
@Function                         // implementation body
{
  E = 2.2693e11*(1.-2.5*f)*(1-6.786e-05*T-4.23e-08*T*T);
}
```

$$E\,(T,f) = 2.2693\,10^{11}\,(1-2.5\,f)\,(1-6.786\,10^{-5}\;T-4.23\,10^{-8}\;T^2)$$

A first example : material property

```
@DSL MaterialLaw;                   // treating a material property
@Material UO2;                      // material name
@Law YoungModulus_Martin1989;       // name of the material property
@Author    T. Helfer;               // author name
@Date      04/04/2014;              // implementation date
@Description                        // detailled description
{
  The elastic constants of polycrystalline UO2 and
  (U, Pu) mixed oxides: a review and recommendations
  Martin, DG
  High Temperatures. High Pressures, 1989
}

@Output E;                          // output of the material property
E.setGlossaryName("YoungModulus");
@Input T,f;                         // inputs of the material property
T.setGlossaryName("Temperature");
f.setGlossaryName("Porosity");

@PhysicalBounds T in [0:*[;         // Temperature is positive
@PhysicalBounds f in [0:1.];        // Porosity is positive and lower than one
@Bounds T in [273.15:2610.15];      // Validity range

@Function                           // implementation body
{
  E = 2.2693e11*(1.-2.5*f)*(1-6.786e-05*T-4.23e-08*T*T);
}
```

$$E\,(T,f) = 2.2693\,10^{11}\,(1-2.5\,f)\,(1-6.786\,10^{-5}\,T-4.23\,10^{-8}\,T^2)$$

Mechanical behaviour : a reminder

- mechanical equilibrium, find $\Delta\vec{\mathbb{U}}$ such :

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

Mechanical behaviour : a reminder

- mechanical equilibrium, find $\Delta\vec{\mathbb{U}}$ such :

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

- internal force :

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

Mechanical behaviour : a reminder

- mechanical equilibrium, find $\Delta\vec{\mathbb{U}}$ such :

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

- internal force :

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

- Newton-Raphson algorithm :

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

Mechanical behaviour : a reminder

- mechanical equilibrium, find $\Delta\vec{\mathbb{U}}$ such :

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

- internal force :

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

- Newton-Raphson algorithm :

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

- elementary stiffness :

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

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

Mechanical behaviour function

- the mechanical behaviour must :
  - compute the stress tensor $\underline{\sigma}$ at the end of the time step for a given strain increment $\Delta\,\underline{\epsilon}^{to}$
    - this requires to compute the internal state variables increments
  - give (at least an approximation of) of $\dfrac{\partial\Delta\underline{\sigma}}{\partial\Delta\underline{\epsilon}^{to}}$
    - other operators shall be allowed (unloading)
  - warn if a variable is out of its bounds :
    - physical bounds : computations have gone bad, really really bad (negative temperature)
    - "standar" bounds : correlation is not valid

Mechanical behaviour function

- the mechanical behaviour must :
  - compute the stress tensor $\underline{\sigma}$ at the end of the time step for a given strain increment $\Delta\,\underline{\epsilon}^{to}$
    - this requires to compute the internal state variables increments
  - give (at least an approximation of) of $\dfrac{\partial \Delta \underline{\sigma}}{\partial \Delta \underline{\epsilon}^{to}}$
    - other operators shall be allowed (unloading)
  - warn if a variable is out of its bounds :
    - physical bounds : computations have gone bad, really really bad (negative temperature)
    - "standar" bounds : correlation is not valid
- the mechanical behaviour shall (to be developed) :
  - give an estimate of the next time step
  - warn if the given strain increment is too large
  - warn if the given strain increment leads to unreliable results :
    - an increment of 10 % of the equivalent plastic strain is **not** admissible whatever the integration scheme is used

An example of mechanical behaviour

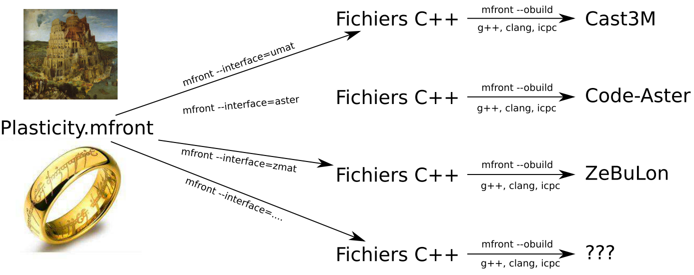
```
@DSL IsotropicPlasticMisesFlow;
@Behaviour plasticflow;
@Author Helfer Thomas;
@Date 23/11/06;

@MaterialProperty stress H;

@FlowRule{
  f       = seq-H*p;
  df_dseq = 1;
  df_dp   = -H;
}
```

- A simple J_2 (isotropic) plastic behaviour :
  - example of specific behaviour implementation
  - automatic computation of the consistent tangent operator
- various domain specific languages are available to cope with :
  - general small strain behaviours, general finite strain behaviours, cohesive zone models
  - explicit integration schemes
    - various Runge-Kutta algorithms available
  - implicit integration schemes
    - various algorithms available

Implicit schemes : a reminder

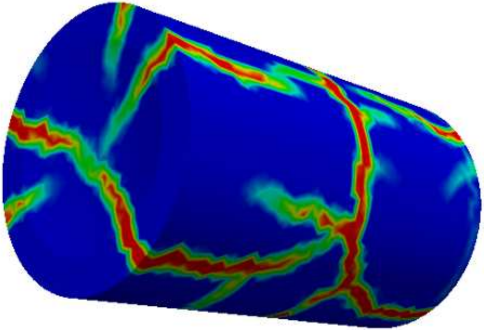
- mechanical behaviour integration is solving the following ode over a time step $\Delta\, t$:
$$\dot{Y} = G\,(Y, t)$$
where Y are the internal state variables packed in a vector, and t stands for the evolution of some external state variables (temperature)
- implicit schemes leads to a non-linear system of equations :
$$F\,(\Delta\, Y) = \Delta\, Y - \Delta\, t\, G\,(Y_t + \theta \Delta Y, t + \theta\, \Delta t) = 0$$
- Most algorithms require $J = \dfrac{\partial F}{\partial \Delta\, Y} = \begin{pmatrix} \dfrac{\partial f_{y_1}}{\partial \Delta\, y_1} & \dots & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \dfrac{\partial f_{y_i}}{\partial \Delta\, y_j} & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & \dots & \dots & \dots & \dfrac{\partial f_{y_N}}{\partial \Delta\, y_N} \end{pmatrix}$
- if an analytical jacobian is provided, it can be compared to a numerical one : `@CompareToNumericalJacobian true;`

Code generation and interfaces

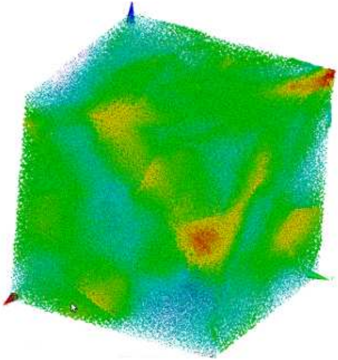


- finite elements solver currently supported :
  - `Cast3M`, Code-Aster, `ZeBuLoN`
- Fast Fourier transform solver :
  - `TMFFT`, `AMITEX_FFT`
- fuel performances code :
  - Cyrano3

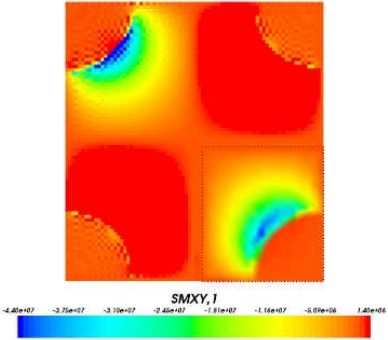
Highlights



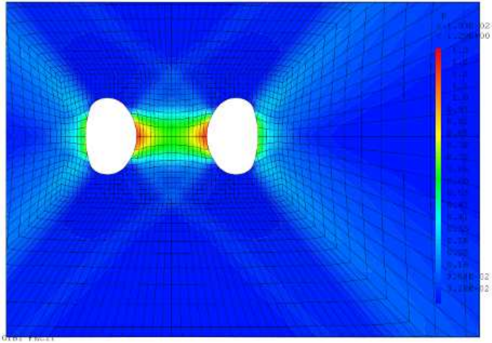
Fuel pellet fragmentation.
Cast3M
B. Michel
2014



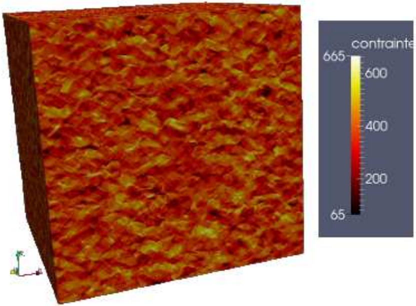
Polycrystals
computations.
Code Aster
J.-M. Proix
2014



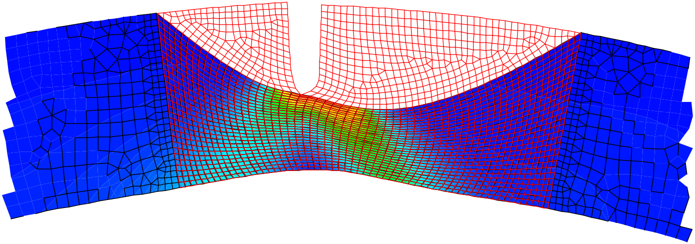
MOX Fuel modelling
Comparison TMFFT/Cast3M
É. Castelier
2013



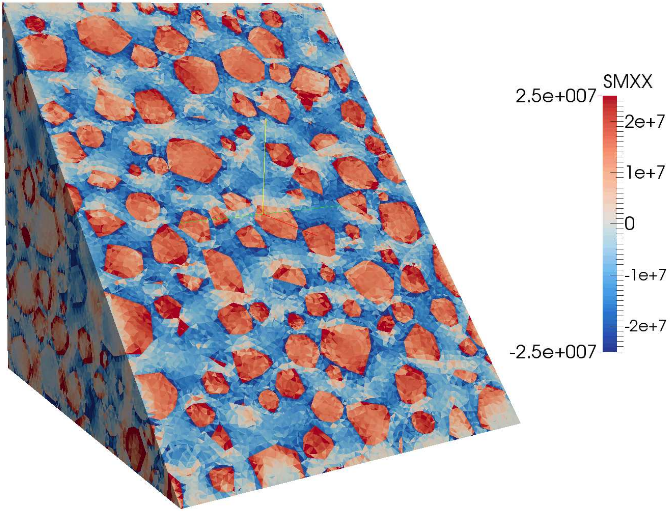
Void interactions
in steel.
Logarithmic strain
framework.
Cast3M
M. Callahan, J. Hure
2014



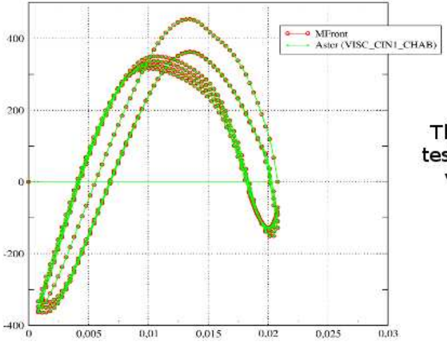
Polycrystals
computations
10^9 voxels
AMITEX_FFTP
L Gélébart
2014



Cladding failure in RIA. T. Helfer. Alcyone/Licos. 2015



Cement. B. Bary. Cast3M. 2015



Thermo-mechanical unit testing of a Chaboche-like viscoplastic behaviour
MTest vs Code-Aster
J.M. Proix, 2014

The ”product”

Licences : Open-source (again) !

- To meet CEA and EDF needs, TFEL 2.0 is released under a multi-licensing scheme :
  - open-source licences :
    - GNU Public License : This licence is used by the Code-Aster finite element solver.
    - CECILL-A : License developped by CEA, EDF and INRIA, compatible with the GNU Public License and designed for conformity with the French law.
  - CEA and EDF are free to distribute `TFEL` under custom licences : Mandatory for the `PLEIADES` plateform.
- number of downloads since October 2014 :
  - `tfel-2.0` : 66
  - `tfel-2.0.1` : 35
  - we did only little promotion :
    - the `Cast3M` site
    - Materiaux 2014

Versions and development branches

- two official versions :
  - version 2.0 : first open-source release
  - version 2.0.1 : minor corrections to 2.0 and improved packaging.
- suberversion repository :

  `https://svn-pleiades.cea.fr/SVN/TFEL/trunk`
- current branches :
  - `rliv-2.0`
  - `trunk`
  - `rdev-3.0` port to `C++-11`

Portability

- TFEL 2.0.x is based on the C++98 standard
- TFEL can be compiled with various C++ compilers :
  - gcc, from version 3.4 to version 4.9.2
  - clang, from version 3.3 to version 3.5
  - icc, from version 10 to version 13
- TFEL is mainly developed on LiNuX
- ports have been made to various POSIX systems FreeBSD, OpenSolaris, etc...
- a windows port is also available (requires MSYS)

Documentation

- reference manuals :
  - general introduction. Material properties and models
  - writing mechanical behaviours
  - how to handle plane stress in implicit schemes
  - finite strain behaviours
  - tutorial
- various talks and tutorials
- documentation of solver interfaces

Software quality

- very stringent compilers warnings :
  - `g++ -Wall -W -Wextra -pedantic -Wdisabled-optimization -Wlong-long -Winline -Wswitch -Wsequence-point -Wignored-qualifiers -Wzero-as-null-pointer-constant -Wvector-operation-performance -Wtrampolines -Wstrict-null-sentinel -Wsign-promo -Wsign-conversion -Wold-style-cast -Wnoexcept -Wmissing-include-dirs -Wmissing-declarations -Wlogical-op -Winit-self ...`
  - `clang -Weverything -Wno-c++98-compat -Wno-global-constructors -Wno-exit-time-destructors -Wno-documentation -Wno-padded`
- continuous integration based on `Jenkins`
- more than 600 tests :
  - most of them are based on `mtest`
  - not enough unit testing (low coverage ratio)
- more tests inside `PLEIADES` applications.

Communications

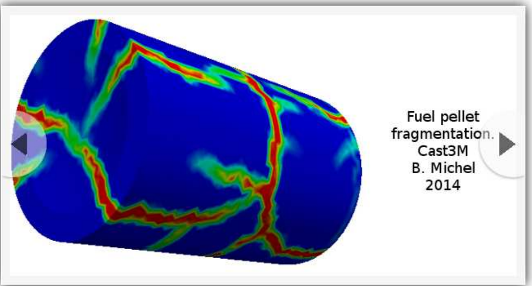
The website

News | Overview | Getting started | Documentation | Contributing | Getting Help

CADARACHE

eDF

MFront: a code generation tool dedicated to material knowledge



`http://tfel.sourceforge.net`

Papers and Talks

- a list of papers and talks describing works done with MFront is available in the appendix
- *Introducing the open-source `MFront` code generator : application to mechanical behaviours and material knowledge management within the PLEIADES fuel element modelling plateform*. Helfer, Thomas and Proix, Jean-Michel and Michel, Bruno and Salvo, Maxime and Sercombe, Jérôme and Casella, Michel. Under review. Computers & Mathematics with Application.
- *Implantation de lois de comportement mécanique à l'aide de MFront : simplicité, efficacité, robustesse et portabilité.* T. Helfer, J.M. Proix, O. Fandeur. CSMA. 12ème colloque national en calcul des structures. May 18-22 2015. Giens, France.

What is new in `TFEL/Math`

Non symmetric tensors

- non symmetric tensors `tensor`
- various functions :
  - `polar_decomposition`, `transpose`, `det`, `convertSecondPiolaKirchhoffStressToCauchyStress`, `convertCauchyStressToSecondPiolaKirchhoffStress` `computeDeterminantDerivative`, `push_forward`, `computeGreenLagrangeTensor`, `computeRightCauchyGreenTensor`, `syme`, `trace`

Various fourth order tensors

- `t2tot2`, `t2tost2`, `st2tot2`
- various functions :
  - `computeKirchoffStressDerivativeFromCauchyStressDerivative`,
  - `computePushForwardDerivative`
- static methods :
  - `t2tot2<N,T>::tpld` (tensor product left derivative)
  - `t2tot2<N,T>::tprd` (tensor product right derivative)
  - $\mathrm{d}\left(\underset{\sim}{\mathbf{A}} \star \underset{\sim}{\mathbf{B}}\right) = \underbrace{\mathrm{d}_{\mathrm{l}}^{\star}\left(\underset{\sim}{\mathbf{A}}\right)}_{\texttt{tpld(A)}} : \mathrm{d}\,\underset{\sim}{\mathbf{B}} + \underbrace{\mathrm{d}_{\mathrm{l}}^{\star}\left(\underset{\sim}{\mathbf{B}}\right)}_{\texttt{tprd(B)}} : \mathrm{d}\,\underset{\sim}{\mathbf{A}}$
  - "optimised" version :

$$\frac{\partial}{\partial \underset{\sim}{\mathbf{X}}}\left(\underset{\sim}{\mathbf{A}} \star \underset{\sim}{\mathbf{B}}\right) = \underbrace{\mathrm{d}_{\mathrm{l}}^{\star}\left(\underset{\sim}{\mathbf{A}}\right) : \frac{\partial \underset{\sim}{\mathbf{B}}}{\partial \underset{\sim}{\mathbf{X}}}}_{\texttt{tpld(A,dBdX)}} + \underbrace{\mathrm{d}_{\mathrm{l}}^{\star}\left(\underset{\sim}{\mathbf{B}}\right) : \frac{\partial \underset{\sim}{\mathbf{A}}}{\partial \underset{\sim}{\mathbf{X}}}}_{\texttt{tprd(B,dAdX)}}$$

Eigenvalues and eigentensors derivatives

- if $\vec{e}_i$ are the eigenvectors of $\underline{\mathbf{s}}$, then one may define the following symmetric tensors :

$$\underline{\mathbf{n}}_{ij} = \begin{cases} \vec{e}_i \otimes \vec{e}_i \text{ if } & i = j \\ \dfrac{1}{\sqrt{2}} \left(\vec{e}_i \otimes \vec{e}_j + \vec{e}_j \otimes \vec{e}_i\right) \text{ if } & i \neq j \end{cases}$$

- $\underline{\mathbf{n}}_{ii}$ are the *eigentensors* of $\underline{\mathbf{s}}$:

$$\underline{\mathbf{s}} = \sum_{i=0}^{3} \lambda_i \, \underline{\mathbf{n}}_{ii}$$

- the derivatives of λ_1 and $\underline{\mathbf{n}}_{11}$ are[1] :

$$\begin{cases} \dfrac{\partial \lambda_1}{\partial \underline{\mathbf{s}}} = \underline{\mathbf{n}}_{11} \\ \dfrac{\partial \underline{\mathbf{n}}_{11}}{\partial \underline{\mathbf{s}}} = \dfrac{1}{\lambda_1 - \lambda_2} \underline{\mathbf{n}}_{12} \otimes \underline{\mathbf{n}}_{12} + \dfrac{1}{\lambda_1 - \lambda_3} \underline{\mathbf{n}}_{13} \otimes \underline{\mathbf{n}}_{13} \end{cases}$$

1. There is an equivalent (but less explicit) expression in [Miehe 02]

Isotropic function of tensors derivatives

- if f is a scalar function, one may define :

$$f\left(\underline{\mathbf{s}}\right) = \sum_{i=0}^{3} f\left(\lambda_i\right)\,\underline{\mathbf{n}}_{ii}$$

- if $f\left(x\right)$ is $\mathcal{C}^1$, then $f\left(\underline{\mathbf{s}}\right)$ is also differentiable.
- applications :
  - convert the dual stress of the logarithmic strain (or any of the Seth-Hill family of strains) to the second Piola-Kirchhoff stress (see below)
  - derivatives of the positive/negative parts of a tensor that appears in :
    - the jacobian matrix of behaviours mixing Mazars damage and creep
    - unilateral effects of some damage behaviour (modified Marigo/Lorentz model)

Various new features in `MFront`

Consistent tangent operator

- Almost mandatory for Code-Aster, Cyrano3 and `ZeBuLoN`
- `@TangentOperator` keyword
- Various prediction operators can be requested :
  - elastic
  - secant
  - tangent
  - consistent tangent operator
- Automatically provided for specific parsers :
  - Except for `MultipleIsotropicMisesFlows` (although documented...)
- some interfaces (`Cast3M`, Code-Aster) allow a comparison of the computed tangent operator to a numerical one.

getPartialJacobianInvert

- for implicit schemes, **in most cases**, the consistent tangent operator can be derived « for free » using the upper left part of the implicit system jacobian invert (!) [Besson 04] :

$$\frac{\partial \Delta \underline{\sigma}}{\partial \Delta \underline{\epsilon}^{to}} = \underline{\underline{\mathbf{D}}} : \frac{\partial \Delta \underline{\epsilon}^{el}}{\partial \Delta \underline{\epsilon}^{to}} = \underline{\underline{\mathbf{D}}} : J^{-1}_{\underline{\epsilon}^{el}}$$

- requires :
  - small strain behaviour (possible extension to finite strain, but this is more complex)
  - $\underline{\epsilon}^{el}$ must be the first variable of the implicit scheme
  - partition of strain :

$$f_{\underline{\epsilon}^{el}} = \Delta \underline{\epsilon}^{el} - \Delta \underline{\epsilon}^{to} + \ldots$$

  - $\Delta \underline{\epsilon}^{to}$ only appears in $f_{\underline{\epsilon}^{el}}$
  - stress-strain relationship is linear elastic (easy extension to damage/non linear elasticity) :

$$\underline{\sigma} = \underline{\underline{\mathbf{D}}} : \underline{\epsilon}^{el}$$

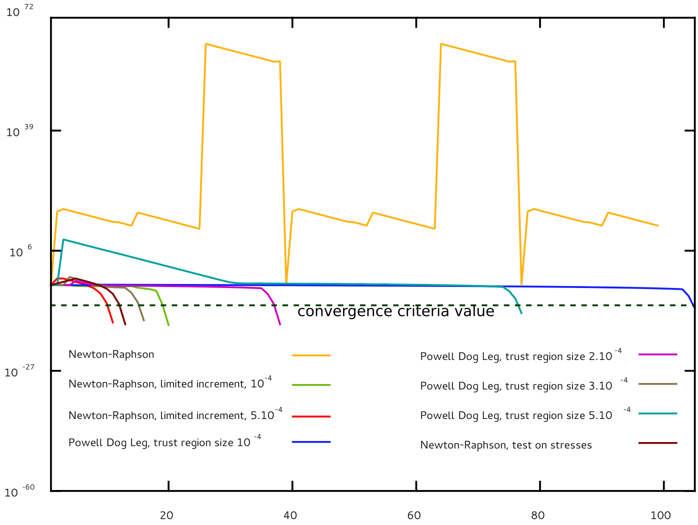
Prediction operator

- `@PredictionOperator` keyword
- Various prediction operators can be requested :
  - Elastic
  - Secant
  - Tangent
- Automatically provided for specific parsers :
  - Except for `MultipleIsotropicMisesFlows`
  - Plastic parser does not have the tangent operator (requires an additional state variable)

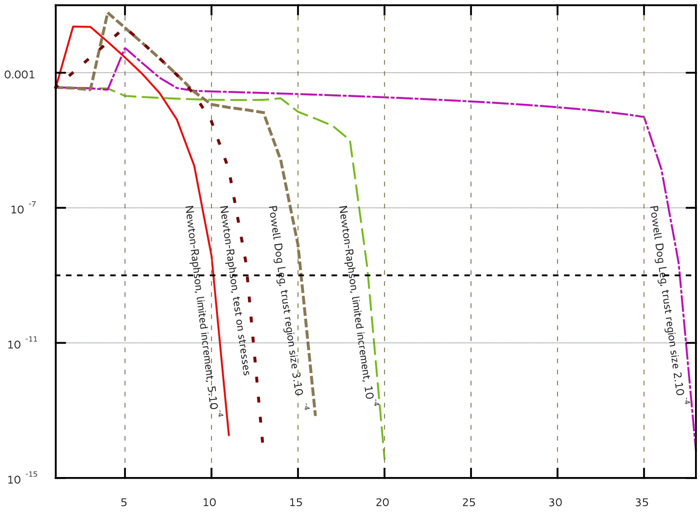
New algorithms for the Implicit parser

- increasing the robustness of the standard Newton-Raphson schemes :
  - limiting the size of the increments :
    - `@MaximumIncrementValuePerIteration 1.e-4;`
    - `return false;`
- globalisation techniques :
  - Levenberg-Marquart
  - coupling Newton-Raphson and the Gauss method through a **simplified** Powell's dogleg algorithm (constant trust-region size)
  - introduction of the class `MFrontNonLinearSolver` :
    - new algorithms can easily be added

Improved robustness



Improved robustness



Finite strain

Two approaches

- reusing small-strain behaviours implementations :
  - using *finite strain strategies* (see below)
  - this leads to **new** behaviours
- writing behaviours in a general finite strain framework :
  - $\left.\underset{\sim}{\mathbf{F}}\right|_{t}$ and $\left.\underset{\sim}{\mathbf{F}}\right|_{t+\Delta\,t}$ as inputs
  - Cauchy stress σ as output.
  - what is the consistent tangent operator ?

```
@UMATFiniteStrainStrategies[umat] {None,FiniteRotationSmallStrain ,
                                   MieheApelLambrechtLogarithmicStrain};
```

- small strain formalism can be reused to build *consistent* finite strain behaviours :
  - this creates a **new** behaviour !
- two lagrangian finite strain strategies are available :
  - finite rotation, small deformation. Available in Code-Aster [EDF 13a].
  - logarithmic strains based on Miehe et al. [Miehe 02]. Available in Code-Aster [EDF 13b] and ZeBuLoN.
- **restriction :**
  - swelling and thermal expansion must be taken into account within the behaviour.

Logarithmic strains - Principle

Preprocessing

$$\underline{\varepsilon}^{\mathrm{to}}_{\log} = \log\left(\underset{\sim}{U}\right) = \frac{1}{2}\log\left({}^{\mathrm{tr}}\underset{\sim}{C}\,.\,\underset{\sim}{C}\right)$$

$\rightarrow$ **Small strain behaviour**

$$\left(\underline{\epsilon}^{to}_{\log}, \Delta\,\underline{\epsilon}^{to}_{\log}\right) \Rightarrow \underline{\mathbf{T}}$$

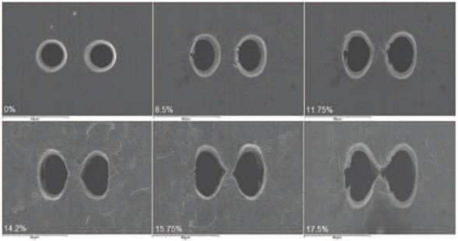
$\rightarrow$ **Postprocessing**

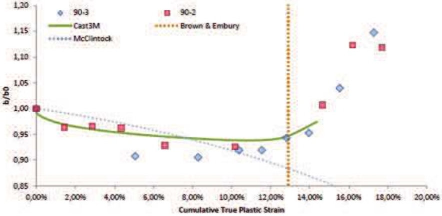
$$\underline{\mathbf{T}} \Rightarrow \underline{\sigma}$$

- $\underline{\mathbf{T}}$ is the dual of the logarithmic strain $\underline{\epsilon}^{to}{}_{\log}$:

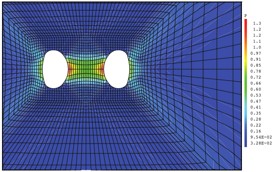
$$P = \underline{\mathbf{T}} : \dot{\underline{\epsilon}}^{to}{}_{\log} = \underline{\mathbf{S}} : \dot{\underline{\epsilon}}^{to}{}_{GL}$$

- if the small strain behaviour is *thermodynamically consistent*, so does the corresponding finite strain behaviour.
- the behaviour is *objective* due to its lagrangian nature.
- *no restriction* on the small strain behaviour (initial and induced *orthotropy* can be handled appropriately)
- **drawbacks :** the pre- and post-processing stage are non trivial and may have a significant computation costs.
- $1D$ is a very interesting case (fuel performance codes) :
  - see appendix





- steel used for PWR reactors internals
- elasto-plastic behaviour
- ability to cope for very large strains
- Results by J. Hure et al. (see Cast3M User Meeting 2014)



Saint-Venant Kirchhoff

```
@DSL DefaultFiniteStrainParser;
@Behaviour SaintVenantKirchhoffElasticity;

@MaterialProperty stress young;
young.setGlossaryName("YoungModulus");
@MaterialProperty real nu;
nu.setGlossaryName("PoissonRatio");

@LocalVariable stress lambda;
@LocalVariable stress mu;

@LocalVariable StressStensor s;

@Includes{
#include"TFEL/Material/Lame.hxx"
}

@Integrator{
  using namespace tfel::material::lame;
  lambda = computeLambda(young,nu);
  mu     = computeMu(young,nu);
  const StrainStensor e = computeGreenLagrangeTensor(F1);
  s   = lambda*trace(e)*StrainStensor::Id()+2*mu*e;
  sig = convertSecondPiolaKirchhoffStressToCauchyStress(s,F1);
}
```

$$\underline{\mathbf{S}} = \lambda\,\mathrm{tr}\,\underline{\epsilon}^{to}_{GL}\,\underline{\mathbf{I}} + 2\,\mu\,\underline{\epsilon}^{to}_{GL}$$

Consistent tangent operator - I

- each finite element solver has its own finite strain formulation which defines the expected consistent tangent operator (except `Cast3M`) :

$$\underline{\underline{\mathbf{D}}}^{\mathtt{tang}} = \frac{\partial\, \underline{\tau}\big|_{t+\Delta\, t}}{\partial \Delta\, \underset{\sim}{\mathbf{F}}} \quad \text{avec} \quad \Delta\, \underset{\sim}{\mathbf{F}} = \underset{\sim}{\mathbf{F}}\Big|_{t+\Delta\, t} \,.\, \underset{\sim}{\mathbf{F}}\Big|_{t}^{-1}$$

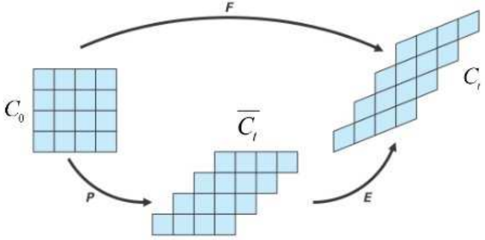
  where τ is the Kirchhoff stress and $\Delta\, \underset{\sim}{\mathbf{F}}$ is the "spatial" increment of the deformation gradient ($\Delta\, \underset{\sim}{\mathbf{F}} = \underset{\sim}{\mathbf{F}}\Big|_{t+\Delta\, t} \,.\, \underset{\sim}{\mathbf{F}}\Big|_{t}^{-1}$)
- each finite strain behaviour has a "natural" consistent tangent operator :
  - Saint-Venant Kirchoff : $\dfrac{\partial \Delta\, \underline{\mathbf{S}}}{\partial \Delta\, \underline{\epsilon}^{to}_{GL}}$

Consistent tangent operator - II

```
@TangentOperator<DS_DEGL>{
  Dt = lambda*Stensor4::IxI()+2*mu*Stensor4::Id();
}
```

- The user is allowed to define one or more expressions of the tangent operator. A special algorithm tries to find the shortest conversion path to provide the consistent tangent operator expected by the code :

$$\frac{\partial \left.\underline{\mathbf{S}}\right|_{t+\Delta t}}{\partial \Delta \underline{\epsilon}^{to}_{GL}} \Rightarrow \frac{\partial \left.\underline{\mathbf{S}}\right|_{t+\Delta t}}{\partial \left.\underline{\mathbf{C}}\right|_{t+\Delta t}} \Rightarrow \frac{\partial \left.\underline{\tau}\right|_{t+\Delta t}}{\partial \left.\underset{\sim}{\mathbf{F}}\right|_{t+\Delta t}} \Rightarrow \frac{\partial \left.\underline{\tau}\right|_{t+\Delta t}}{\partial \Delta \left.\underset{\sim}{\mathbf{F}}\right|_{t+\Delta t}}$$



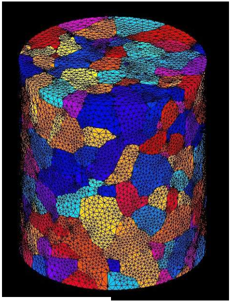
- Multiplicative split of the deformation gradient $\underset{\sim}{\mathbf{F}} = \underset{\sim}{\mathbf{F}}_e \,.\, \underset{\sim}{\mathbf{F}}_p$
- Elasticity : $\underline{\mathbf{S}} = \underline{\underline{\mathbf{D}}} : \underline{\epsilon}^{to}_{GL}$
- Viscoplatic flow :

$$\dot{\underset{\sim}{\mathbf{F}}}_{\mathbf{p}} \,.\, \underset{\sim}{\mathbf{F}}_{\mathbf{p}}^{-1} = \sum_s \dot{\gamma}\left(\tau^s\right) \underset{\sim}{\mathbf{N}}^s$$

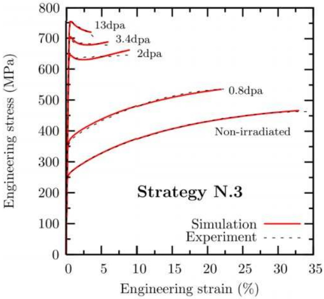
- Resolved shear stress : $\tau^s = \underline{\mathbf{M}} : \dfrac{1}{2}\left(\underset{\sim}{\mathbf{N}}^s + \underset{\sim}{\mathbf{N}}^{T\,s}\right)$

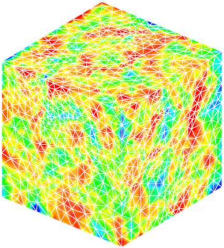
This forms covers most single crystal behaviours

Some results



- Computations on aggregates (Voronoi or realistic)
  - macroscopic behaviour
  - ductile failure (porous model)





Plane stress and generalised plane stress in Implicit Parser

Implicit parser

```
@ModellingHypotheses {".+"};
...
@StateVariable<PlaneStress> real etozz;
PlaneStress::etozz.setGlossaryName("AxialStrain");
...
@Integrator<PlaneStress,Append,AtEnd>{
  // the plane stress equation is satisfied at the end of the time step
  const stress szz = (lambda+2*mu)*(eel(2)+deel(2))
  +lambda*(eel(0)+deel(0)+eel(1)+deel(1));
  fetozz   = szz/young;
  // modification of the partition of strain
  feel(2) -= detozz;
  // jacobian
  dfeel_ddetozz(2)=-1;
  dfetozz_ddetozz  = real(0);
  dfetozz_ddeel(2) = (lambda+2*mu)/young;
  dfetozz_ddeel(0) = lambda/young;
  dfetozz_ddeel(1) = lambda/young;
}
```

- introduced for the Cyrano fuel performance code.
- explicitly declare the behaviour usable in plane stress.
- define the axial strain as a new state variable.
- its evolution is implicitly given by the plane strain condition : $\sigma_z = 0$

mtest

Overview

```
@UseCastemAccelerationAlgorithm true;
@ModellingHypothesis 'Axisymmetrical';
@Behaviour<umat> 'src/libUmatBehaviour.so' 'umatnorton';
@MaterialProperty<constant> 'YoungModulus'     150.e9;
@MaterialProperty<constant> 'PoissonRatio'       0.3;
@MaterialProperty<constant> 'A' 8.e-67;
@MaterialProperty<constant> 'E' 8.2;

@ImposedStrain 'EZZ' {0:0,3600.:1.e3};

@ExternalStateVariable 'Temperature' 293.15;

@Times {0.,3600 in 20};
```

- `mtest` is a tool used to simulate the mechanical behaviour a single material point :
  - unit tests (primary job)
    - able to compare results to reference values or analytical solution
    - generates XML output with `Jenkins`
  - simulations of simple tests :
    - uniaxial tensile and compression tests
    - Satoh tests
- available as an executable or a `python` library

Acceleration algorithms

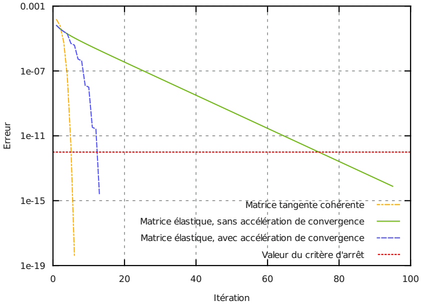
- equilibrium algorithm :

$$R\left(\Delta\,\underline{\epsilon}^{to}\right)=0 \Leftrightarrow \Delta\,\underline{\epsilon}^{to}=\Delta\,\underline{\epsilon}^{to}-K^{-1}\,R\left(\Delta\,\underline{\epsilon}^{to}\right)=0$$

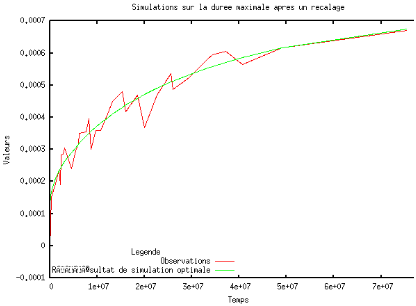
  K is elastic stiffness
- fixed point method (linear and (very) slow convergence) :

$$\Delta\,\underline{\epsilon}^{to}_{n+1}=\Delta\,\underline{\epsilon}^{to}_{n}-K^{-1}\,R\left(\Delta\,\underline{\epsilon}^{to}_{n}\right)=G\left(\Delta\,\underline{\epsilon}^{to}_{n}\right)$$

- acceleration method :
  - Anderson (used in `Cast3M` for years, much improved after 2007 to take contact into account)
  - Secant, Irons-Tuck
- slower than Newton-Raphson, but does not requires the consistent tangent operator !



Behaviour parameter adjustment



- `mtest` can be coupled with Mathlab and `adao` :
  - `adao` is a tool developed within `salome`
  - use case of `mtest` `python` bindings.

Assurance quality

Embedded documentation (alpha stage)

- doxygen-like comments cant be used to comment `MFront` files
- comment are used to build a markdown file using pandoc syntax :
  - file is then convertible to Word, PDF, etc...

Pedantic mode

- `mfront --pedantic`
- name comes from gcc comment line option
- pedantic mode provides advances warnings about a file :
  - unused variables :
  - variables with no glossary/entry names ;
  - variables with no bounds ;
  - variables that can be converted to integration variables ;
  - etc...

Profiling

- must use `--enable-cxx11` when calling `configure` or `-Denable-cxx11` when calling `cmake`
- enables the `@Profiling keyword` :

```
@Profiling true;
```

Conclusions

Acknowledgements to contributors

- J. M. Proix, J. Hure, F. Hamon, I. Ramière, , É. Castelier O. Fandeur, V. Blanc, J. Julien, B. Michel, B. Bary, F. Milliard, A. Courcelle, and all the others for various contributions.
- all the persons who contributed to the open-source release of `TFEL` : J. P. Defain, T. De Soza, V. Marelle, É. Lorentz, C. Toulemonde, F. Curtit, R. Masson, and all the others.
- the authors are grateful to J. Besson and S. Quilici for their valuable help in building the `ZeBuLoN` interface.

Acknowledgements

- This research was conducted in the framework of the PLEIADES project, which was supported financially by the CEA (Commissariat à l'Énergie Atomique et aux Énergies Alternatives), EDF (Électricité de France) and AREVA and in the framework of the Simu-Meca2015 project hold within EDF R&D.

Appendix

Publications - I

- *Experimental characterization and modelling of UO_2 behavior at high temperatures and high strain rates* Salvo, Maxime and Sercombe, Jérôme and Ménard, Jean-Claude and Julien, Jérôme and Helfer, Thomas and Désoyer, Thierry. Jan. 2015. Journal of Nuclear Materials.
- *Introducing the open-source `MFront` code generator : application to mechanical behaviours and material knowledge management within the PLEIADES fuel element modelling plateform*. Helfer, Thomas and Proix, Jean-Michel and Michel, Bruno and Salvo, Maxime and Sercombe, Jérôme and Casella, Michel. Under review. Computers & Mathematics with Application.
- *Licos, a fuel performance code for innovative fuel elements or experimental devices design*. Helfer Thomas and Bejaoui Syriac. Under review. Nuclear Engineering And Design.

Publications - II

- *Extension of monodimensional fuel performance codes to finite strain analysis using a lagrangian logarithmic strain framework*. Helfer, Thomas. Under review. Nuclear Engineering And Design.
- *Iterative residual based vector methods to accelerate fixed point iterations*. Ramière, Isabelle and Helfer, Thomas. Under writing.

Talks - 2014

- *Modélisation de la fissuration inter-granulaire par corrosion sous contraintes pour les aciers inoxydables*. Couvant, T. and Meunier, S. and Haboussa, D. and Proix, J.-M. Journée des utilisateurs Code-Aster et `Salome-Meca`. Paris, 18 march 2014.
- *A thermo-metallurgical-mechanical model of EM10 at high temperature*. Courcelle, Arnaud and Millard, Franck and Gavoille, Pierre and Le Saux Matthieu and Bosonnet, Sophie and Flament Jean-Luc and Guilbert Thomas. Numat 2014. Florida, october 2014.

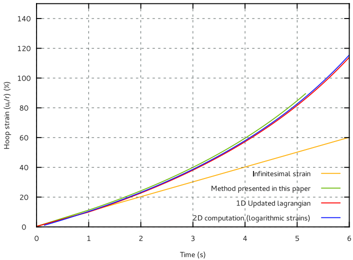
Talks - 2014 - II

- *Présentation de `MFront` : applications aux lois de comportement mecanique et mesure de performance à l'aide de Code-Aster*. Helfer, Thomas and Proix, Jean-Michel and Michel, Bruno. Matériaux 2014. Montpellier, november 2014.
- *Microfissuration induite par la viscoplasticité dan les céramiques nucléaires*. Michel, Bruno and Soulacroix Julian, and Helfer, Thomas. Matériaux 2014. Montpellier, november 2014.
- *Quelques exemples d'utilisation de lois de comportement en grandes déformations générées avec l'outil MFront.* Hure, Jérémy and Callahan, Mike and Ling, Chao and Tanguy, Benoît and Helfer, Thomas. `Cast3M` User Meeting. Paris, 28 november 2014.

Talks - 2015

- *Implantation de lois de comportement mécanique à l'aide de MFront : simplicité, efficacité, robustesse et portabilité.* T. Helfer, J.M. Proix, O. Fandeur. CSMA. 12ème colloque national en calcul des structures. May 18-22 2015. Giens, France.
- *Numerical analysis of concrete creep on mesoscopic 3D specimens.* B. Bary, C. Bourcier, T. Helfer. CONCREEP-10. September 21-23 2015, Autrichia

Logarithmic strains - 1D simulations



- fuel performance code are written using small strain formalism but can be adapted to use the logarithmic strains :

$$\underline{\epsilon}^{to}_{HPP} = \frac{1}{2}\left[\vec{\nabla}\,\vec{u} + {}^{tr}\vec{\nabla}\,\vec{u}\right] = \vec{\nabla}\,\vec{u} \rightarrow \underset{\sim}{\mathbf{F}} = \underline{\mathbf{I}} + \underline{\epsilon}^{to}_{HPP}$$

$$\left.\epsilon^{\mathrm{to}}_{\log}\right|_{rr} = \log\left(1 + \left.\epsilon^{\mathrm{to}}_{\mathrm{HPP}}\right|_{rr}\right) \rightarrow \left.\sigma_{HPP}\right|_{rr} = \frac{1}{1 + \left.\epsilon^{\mathrm{to}}_{\mathrm{HPP}}\right|_{rr}}\, T_{rr}$$

- applications to RIA and LOCA...

References

Besson J. et Desmorat D.
Numerical implementation of constitutive models.
Local approach to fracture. Besson J. École des Mines de Paris - les presses, 2004.

EDF .
Loi de comportement en grandes rotations et petites déformations.
Référence du Code Aster R5.03.22 révision : 11536, EDF-R&D/AMA, Septembre 2013.

EDF .
Modèles de grandes déformations GDEF_LOG et GDEF_HYPO_ELAS.
Référence du Code Aster R5.03.24 révision : 10464, EDF-R&D/AMA, 2013.

Miehe C., Apel N. et Lambrecht M.
Anisotropic additive plasticity in the logarithmic strain space : modular kinematic formulation and implementation based on incremental minimization principles for standard materials.