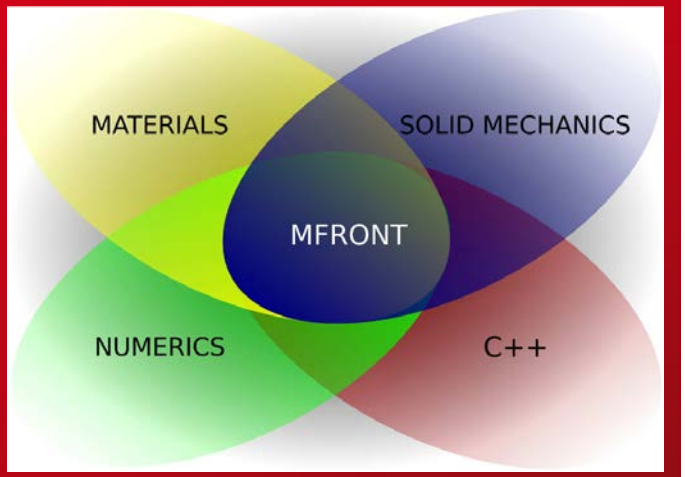
Implémentation de lois de comportement mécanique à l'aide du générateur de code MFront

T. Helfer

DE LA RECHERCHE À L'INDUSTRIE

cea den



Implémentation de lois de comportement mécanique à l'aide du générateur de code MFront

T. Helfer

PLEIADES

Séminaire IMSIA | 27 Janvier 2016

OUTLINE

- Pleiades platform overview
- Role of the mechanical behaviour
- The MFront code generator
- A consistent approach from experiments to fuel simulation codes
- Material knowledge management
- Conclusion / Perspectives

OVERVIEW

Industrial issues for fuel element behavior simulation

- Provide R&D support for exploitation needs (GENII)
- New fuel concept design qualification (GENII, GENIII, JHR, ASTRID)
- Innovative fuel concept design (GEN IV)

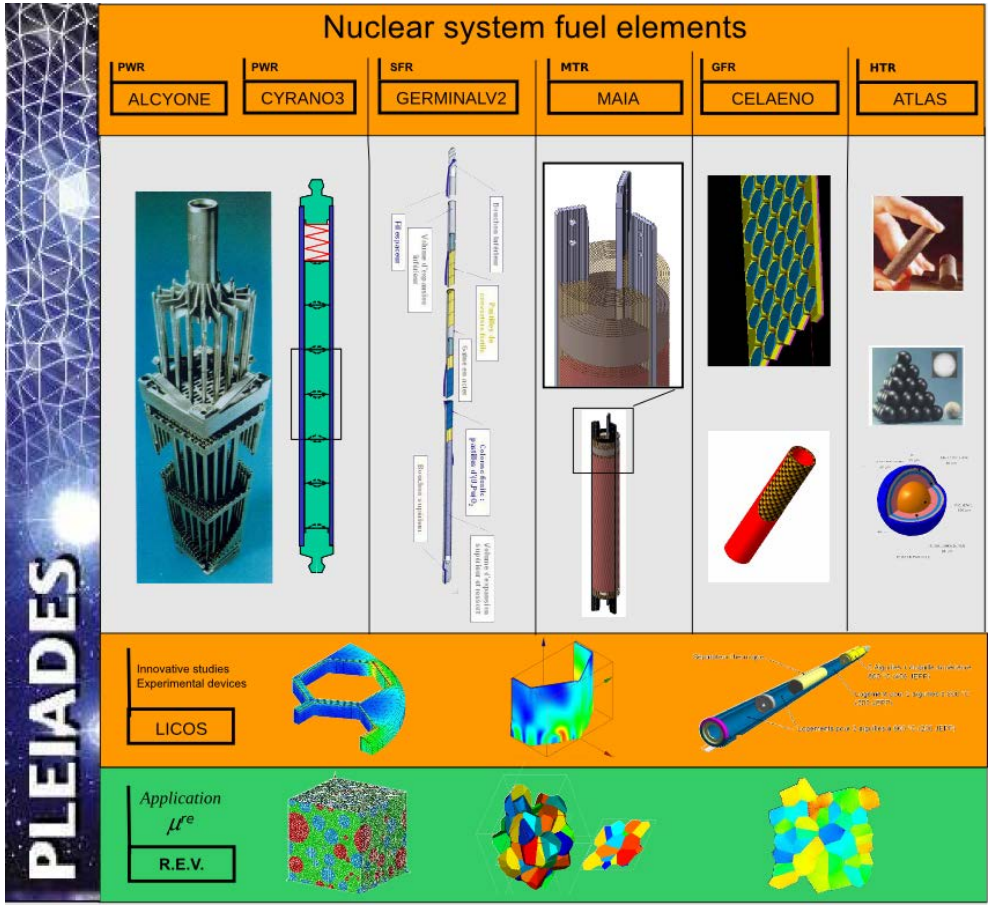
Objectives of the PLEIADES project

- Fuel R&D knowledge capitalization
  - Material data
  - Advanced models
  - Validation

- Provide a unified software environment to share generic methods and tools developed with our industrial partners
- Enable coupling computation with other simulation platforms:

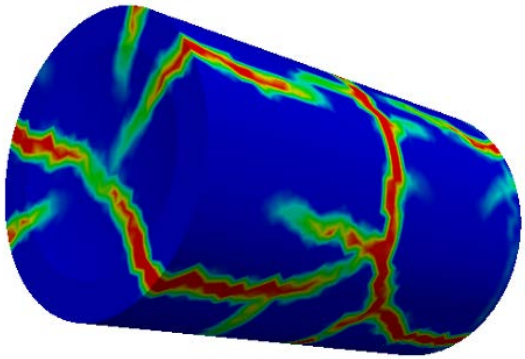
- Increase simulation use in the qualification process of new fuel concepts

PLEIADES' APPLICATIONS



- 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

EXAMPLES OF CRACKS AND FAILURE PHENOMENA IN FUEL ELEMENT SIMULATIONS



- Brittle failure :
  - Oxide fuel at beginning of life
  - Secondary crack network in oxide fuel during power transient (see Michel 2013)
- Ductile failure :
  - Oxide fuel at high temperature :
    - Modeled by porosity growth (Ponte-Casteneda homogeneisation scheme, see [Michel 1992,Monerie 2006, Salvo 2015])
- **Hydrided Zircaloy cladding**:
  - Modeled by an anisotropic GTN behaviour (see [Le Saux 2010, Mac Donald 2014]) which is identified by CEA Nuclear Material Departement (outside PLEIADES)
- **High temperature/pressure failure of cladding with phase transition** :
  - Modeled by the Edgar behaviour (ternary creep and phase transition)
- Grain boundary decohesion :
  - high rate loadings in compression (see [Soulacroix 2014,Salvo 2015])

ROLE OF THE MECHANICAL BEHAVIOURS

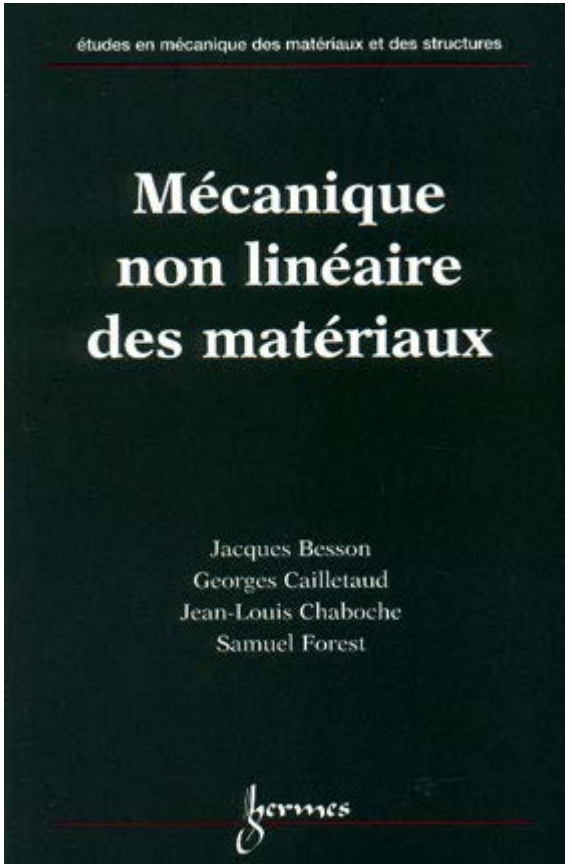
MAIN TASK (IMPLICIT SOLVER)

$$\vec{R}(\Delta\vec{u}) = \vec{F}_i(\Delta\vec{u}) - \vec{F}_e$$

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

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

$$\underline{\underline{\mathbb{K}}}^e = \sum_{i=1}^{N^G} {}^t\underline{\underline{\mathbf{B}}} : \frac{\partial\Delta\underline{\sigma}}{\partial\Delta\underline{\epsilon}^{to}} : \underline{\underline{\mathbf{B}}}\, w_i$$



- Update stresses and states variables
- Compute the consistent tangent operator (if required)
  - Can be an approximation
  - Various variants available (elastic, secant, tangent)

AUXILIARY TASKS

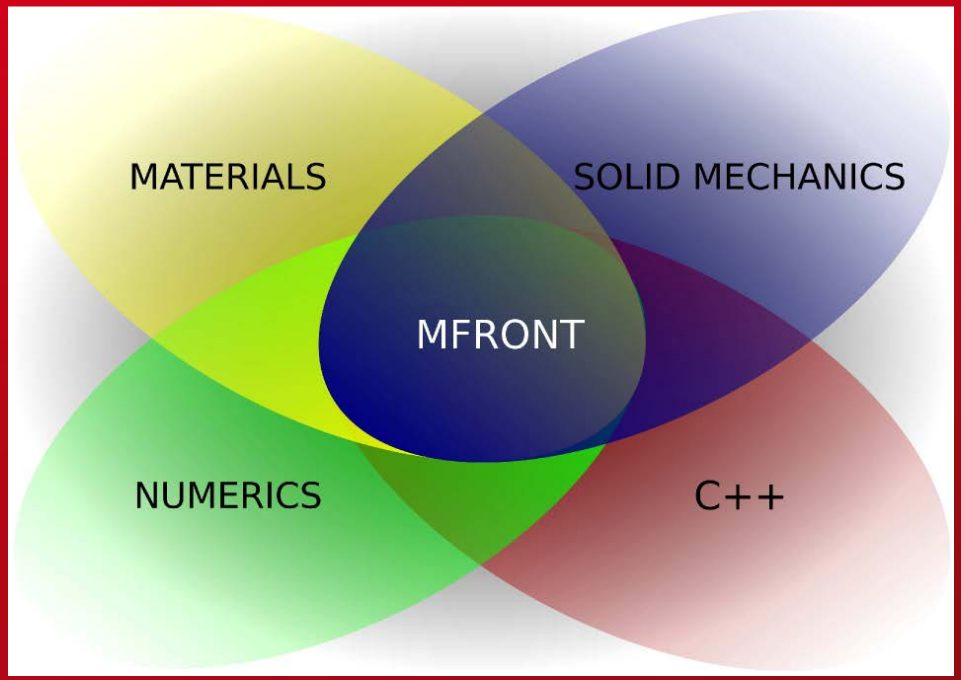
- Provide a estimation of the next time step for time step automatic adaptation

> **WHAT COULD BE IMPROVED**
>
> - Default settings in non-linear solution and automatic time stepping could be improved for better performance. Trying to replicate (and set as default) Abaqus time stepping / convergence analysis would be highly beneficial.

(Pronet – Update2, November 2015)

- 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
- Provide information for dynamic resolution of inputs (MTest/Aster):
  - Numbers
  - Types (scalar, tensors, symmetric tensors)
  - Entry names
  - Glossary names…

MECHANICAL BEHAVIOURS: THE MFRONT CODE GENERATOR



Description

- Code generator to **simplify** constitutive laws and material properties implementation:
  - **User focuses on physics**
  - Implementation are as close as possible to the constitutive equations (tensorial objects, operator overloading, etc..)
  - Usable by standard engineers (no computational skills required)
  - Coding and numerical details are hidden (by default)
- Currently handles small/finite strain behaviours and cohesive zone models
- **Optimisation of numerical performances**:
  - C++ template library optimised for small objects
  - At least on par with native (fortran) implementations
- Used in PLEIADES codes, Cast3M (CEA FEM code) and Code-Aster (EDF FEM code), etc..
- Available on LiNuX, Windows, Mac Os, FreeBSD
- Compatible with standard C++ compilers (g++,clang,intel,VS)

```
@Parser IsotropicPlasticMisesFlow;
@Behaviour Plasticity;
@Author Helfer Thomas;
@Date 23/11/06;

@MaterialProperty stress H;
@MaterialProperty stress s0;

@FlowRule{
  f       = seq-H*p-s0;
  df_dseq = 1;
  df_dp   = -H;
}
```

Open-source code

- TFEL/MFRONT site created on sourceforge: http://*sourceforge.net/projects/tfel/*
- Distributed with the Code-Aster distribution: http://code-aster.org
- A growing community of users. If anyone is interested: tfel-contact@cea.fr !

code_aster

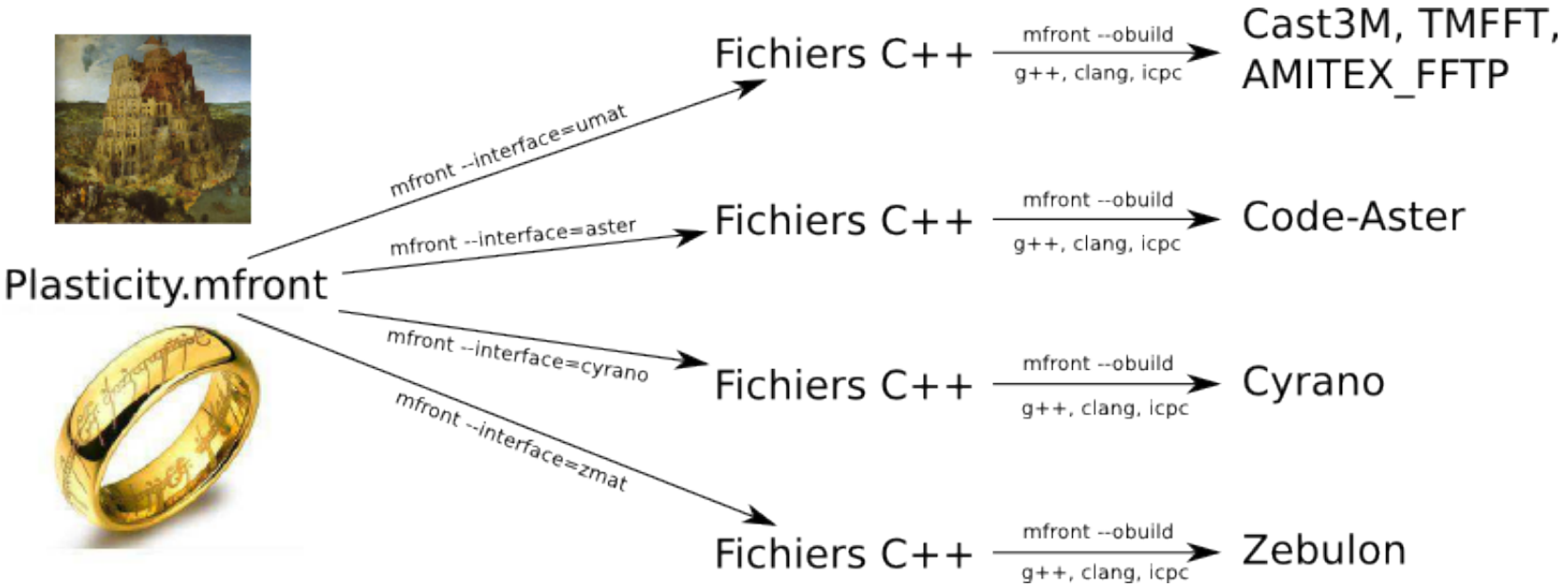
Domain specific languages

- Mfront provides several domain specific languages:
  - One for material properties
  - One for simple physical models (swelling under irradiation)
- For behaviours, several dsl are provided:
  - Specific DSLs for isotropic behaviours
  - Generic DSLs for implementing all kind of behaviours (damage, plasticity, viscoplasticity, single crystal, homogenised schemes, etc…..)
  - Generic DSLs supports a specific integration scheme, either explicit (Runge-Kutta) or implicit (θ-scheme). Various algorithms are available in each cases.

Example of the Implicit DSL

- The implicit scheme turns the behaviour integration into a non-linear system of equations
- See the demo !

MFRONT INTERFACES



- A **single** implementation, **multiple** targets:
  - mfront --interface=xxx generates the appropriate source code
  - mfront --obuild generates the shared library pluggable in solvers
- New interfaces are added when needed:
  - Currently working on an interface to Dassault's Abaqus FEA solver
  - Interface to CEA Europlexus (rapid dynamics) is planned for 2016
  - Interfaces to the CraFT FFT solver or the ANSYS are being considered

IMPLICIT SCHEMES

- le système différentiel devient un système non-linéaire : $[F(\Delta Y) = \Delta Y - \Delta t\, G(Y_t + \theta \Delta Y, t + \theta\, \Delta t) = 0$ avec : $[\Delta Y]^T = \left[\Delta \underline{\epsilon}^{\mathrm{el}}\ ,\ \Delta\alpha\right]$
- pour les lois indépendantes du temps, on annule directement la surface de charge !
- on résout ce système par un NEWTON-RAPHSON
  - il faut la jacobienne $J = \dfrac{\partial F}{\partial \Delta Y}$
- la jacobienne peut être calculée par blocs :

$$J = \frac{\partial F}{\partial Y} = \begin{pmatrix} \frac{\partial f_{y_1}}{\partial y_1} & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \frac{\partial f_{y_i}}{\partial y_j} & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & \cdots & \frac{\partial f_{y_N}}{\partial y_N} \end{pmatrix}$$

- on peut demander une vérification numérique !

```
@CompareToNumericalJacobian true;
```

EXAMPLE: NORTON BEHAVIOUR

```
@Parser Implicit;
@Behaviour Norton;
@Algorithm NewtonRaphson_NumericalJacobian;
@RequireStiffnessTensor;
@MaterialProperty real A;
@MaterialProperty real m;
@StateVariable real    p;
@ComputeStress{  sig = D*eel;}
@Integrator{
  real seq = sigmaeq(sig);
  Stensor n = Stensor(0.);
  if(seq > 1.e-15){
    n   = 1.5*deviator(sig)/seq;
  }
  feel = deel + dp*n-deto;
  fp   = dp - dt*A*pow(seq,m);
}
```

Newton, calcul de J

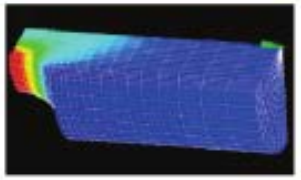
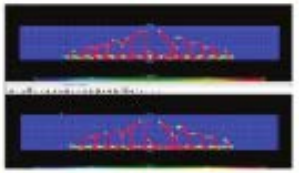
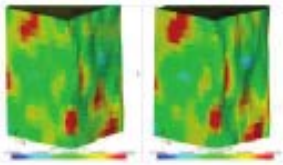
$$\sigma = \underline{\underline{\mathbf{D}}} : \underline{\epsilon}^{\mathrm{el}}$$

$$\underline{n} = \frac{3}{2}\frac{\underline{s}}{\sigma_{\mathrm{eq}}}$$

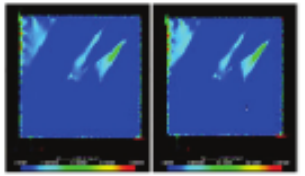
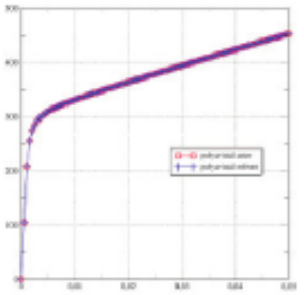
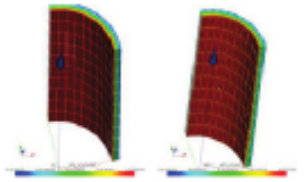
$$\Delta\,\underline{\epsilon}^{\mathrm{el}} + \Delta\,p\,\underline{n} - \Delta\,\underline{\epsilon}^{\mathrm{to}} = 0$$
$$\Delta\,p - \Delta\,t\,A\,\sigma_{\mathrm{eq}}^{m} = 0$$

- Numerical Jacobian
- No consistent tangent operator

BENCHMARKS

Description	Algorithme	Temps CPU total (Aster vs MFront)	Illustration
Visco-plastic and damaging for steel (Mustata and Hayhurst 2005; EDF 2012)	Implicit	17*mn*43*s* vs 7*mn*58*s*	
Damaging for concrete (Mazars and Hamon; EDF 2013a)	Default	45*mn* vs 63*mn*	
Generic Single crystal viscoplasticity (Méric and Cailletaud 1991; EDF 2013b)	Implicit	28*mn* vs 24*mn*	

BENCHMARKS-II

Description	Algorithme	Temps CPU	Illustration
FCC single crystal viscoplasticity (Monnet, Naamane, and Devincre 2011 EDF (2013b))	Implicit	33*m*54*s* vs 29*m*30*s*	
FCC homogeneized polycrystals 30 grains (Berveiller and Zaoui 1978; EDF 2013b)	Runge-Kutta 4/5	9*s*67 vs 8*s*22	
Anisotropic creep with phase transformation (EDF 2013c)	Implicit	180*s* vs 171*s*	

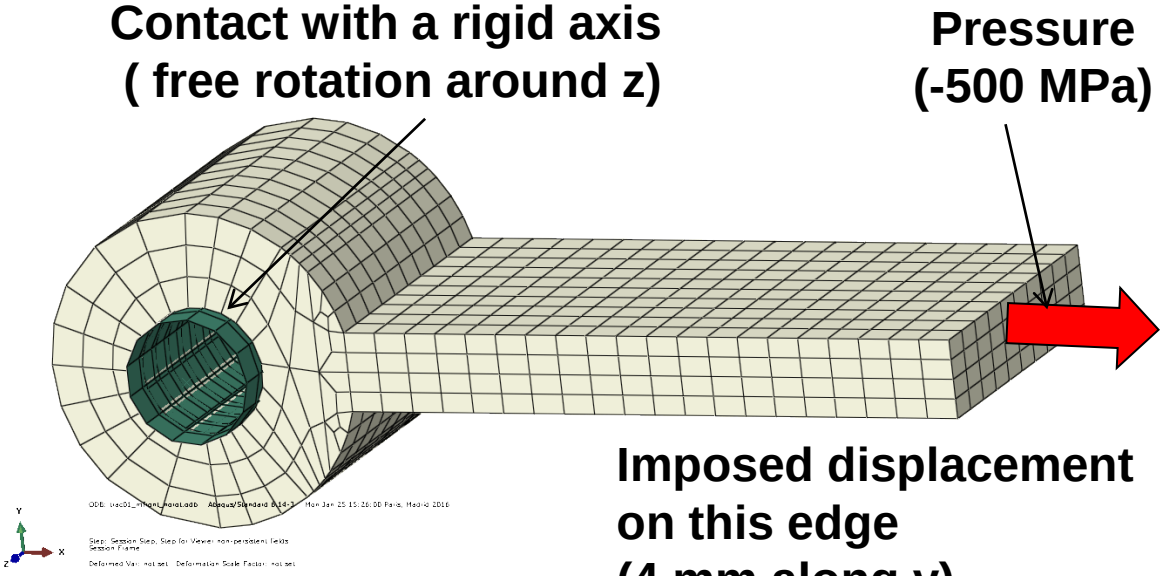
GENERAL CONCLUSIONS

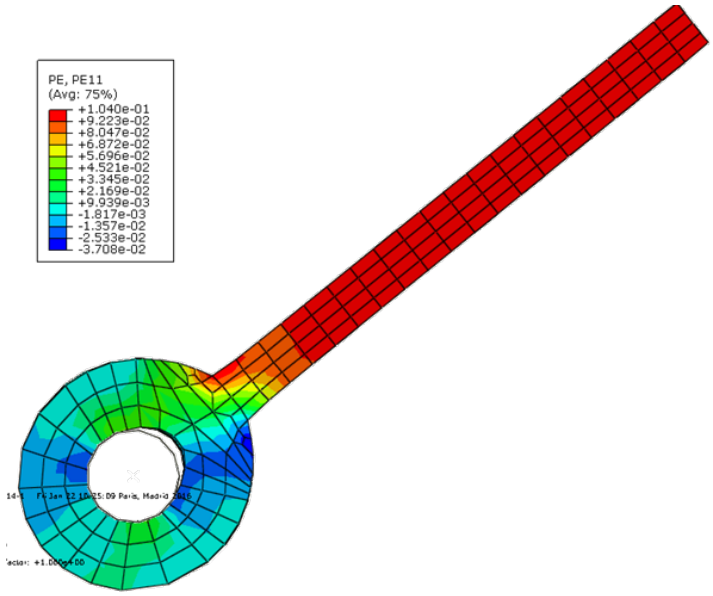
- Developers of the Code-Aster general purpose finite element solver, made independent extensive tests:
  - comparing their own native implementations to the ones generated with MFront
- native implementations offers superior performances:
  - in the case of simple explicit behaviours (Mazars)
  - in the case of isotropic behaviours that can be reduce to one scalar equations
- for more complex/less specific behaviours, MFront implementation are on par or outperforms the native implementations.
- For a given behaviour, **the development time was found significantly lower with Mfront**
- Internal and external benchmarks with Cast3m native implementations also highlight the good performances of the Mfront generated implementations
  - Comparison with other finite solver will be interesting

ABAQUS BENCHMARKS (DELOISON DOMINIQUE, AIRBUS)

Contact with a rigid axis (free rotation around z)

Pressure (-500 MPa)



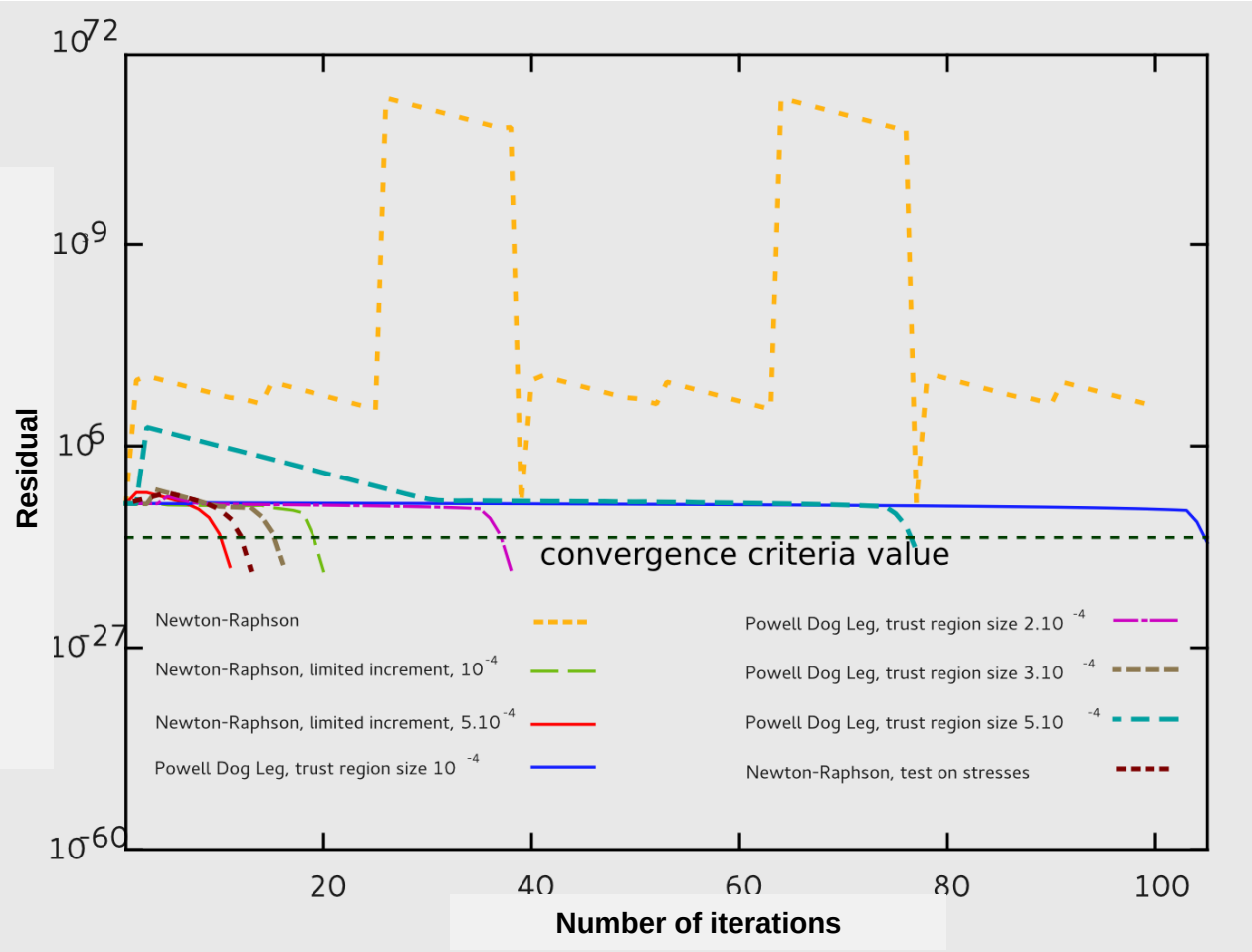


Imposed displacement on this edge (4 mm along y)

	CPU Time (s)	Difference	Number of increments	Difference
ABAQUS	192	-	52	-
ABAQUS/UMAT(*)	1200	525%	295	467%
ABAQUS/MFRONT	201	5%	52	0%
(*) Numerical Jacobian				

MFRONT seems to be much more efficient than a badly implemented UMAT!!!

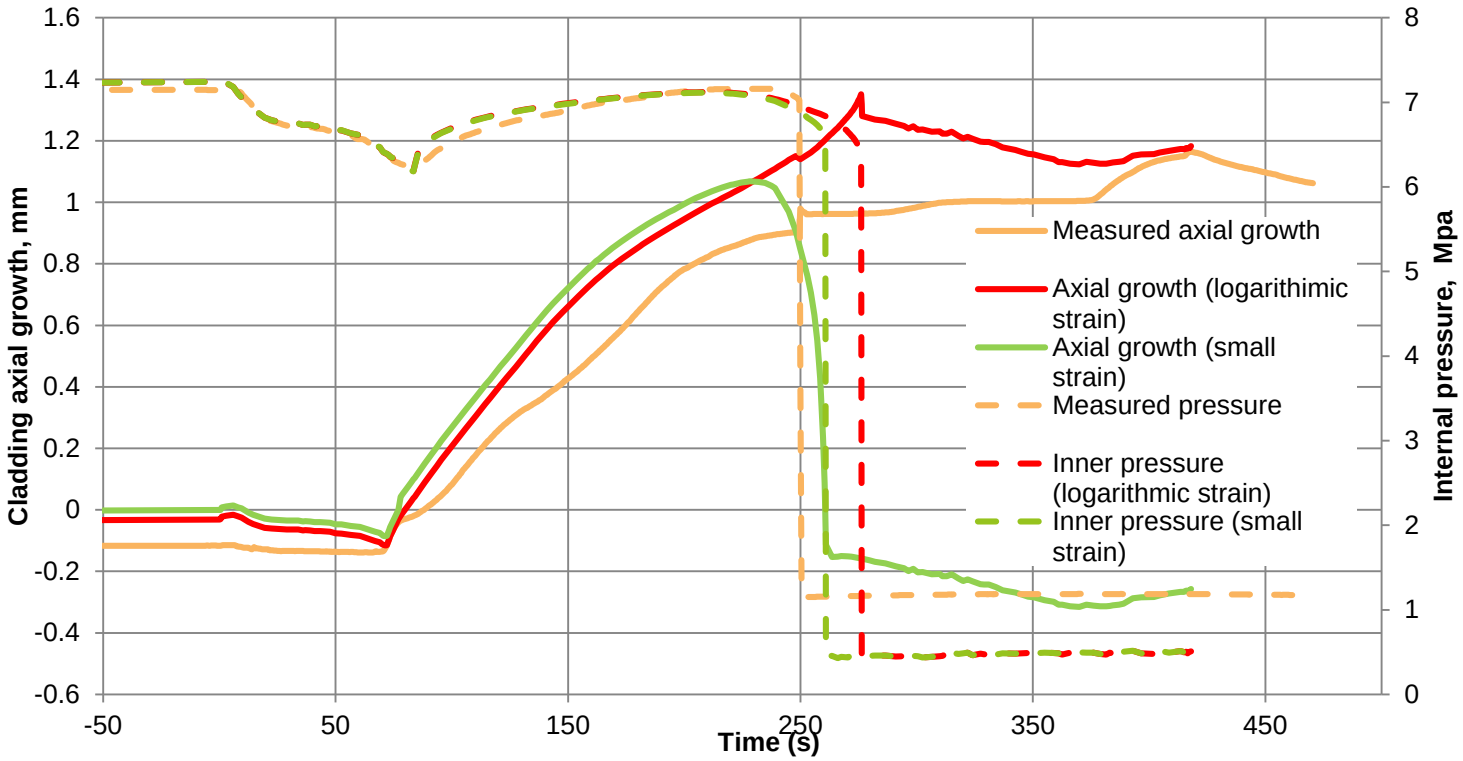
ADVANCED ALGORITHMS



- Up to a 5 times gain in real world computations

A CONSISTENT APPROACH FROM EXPERIMENTS TO FUEL PERFORMANCE CODE

APPLICATION TO THE IFA-650.10 EXPERIMENT

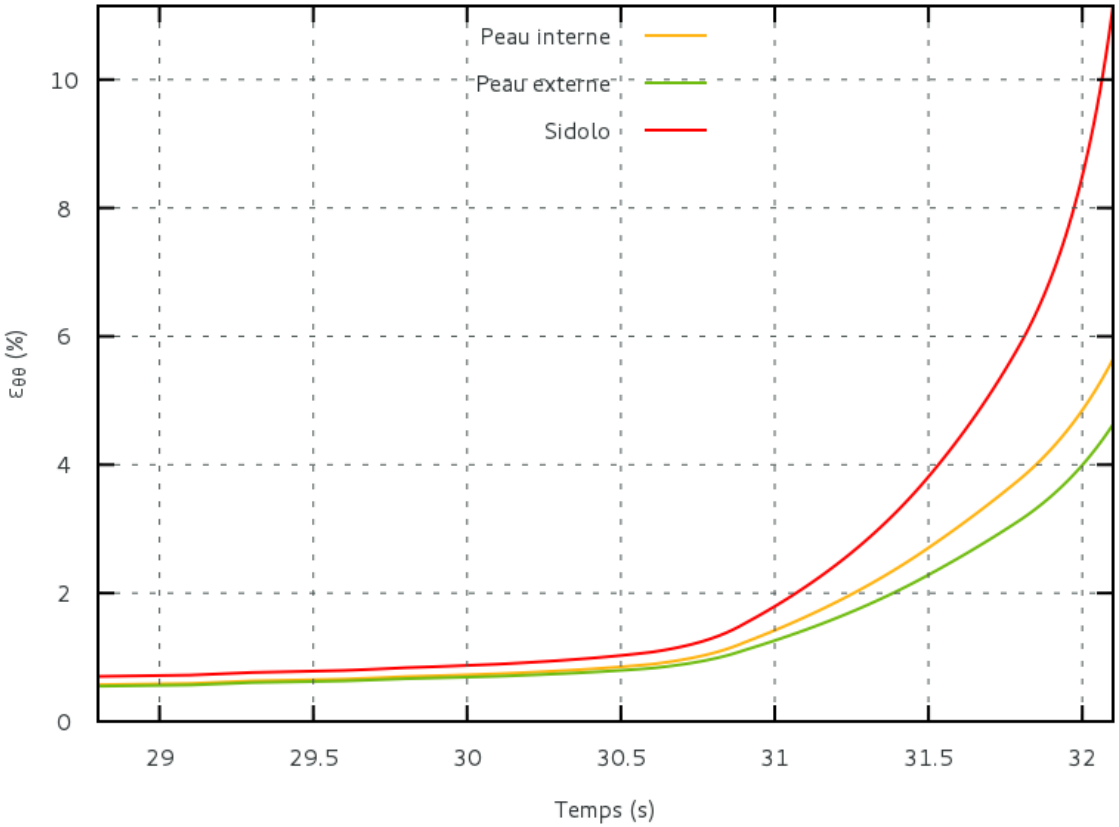


- LOCA test on a 5 cycles UO2-Zy4 fuel rod
- Using small strain formulation leads to:
  - Numerical instabilities
  - Physically wrong prediction (loss of 50% of the cladding volume)
  - Prediction at odds with measured data (here the cladding axial growth)
- Those problems are now solved using the logarithmic strain framework

FINITE STRAIN MECHANICAL ANALYSIS IN 1D

- Most 1D fuel performance codes are written in the infinitesimal strain framework
  - Equilibrium in the initial configuration
  - Mechanical behaviours relates the strain ε^{to} to the stresses
  - Inappropriate for proper LOCA simulations (see example below)
- **1D fuel performance codes can be easily adapted to finite strain analysis**
- Equilibrium in the initial configuration using the first Piola-Kirchhoff tensor π
  - Div $\pi = 0$
  - Pressure boundary conditions must be corrected to take into accounts geometric evolutions (change of diameter and axial growth):
    - $\pi.dS=-P.ds$ (P: applied pressure, dS: surface element in the initial configuration, ds: surface element in the current one)
    - correction up to 30%
    - easy since the outer normals to the fuel and the cladding remains unchanged
    - done in Alcyone at the beginning of each mechanical resolution (weak non-linearity associated with small time steps during LOCA simulations)
    - *only modification done to the equilibrium resolution*
- Mechanical behaviours:
  - **Computation of the deformation gradient** $F=1+\varepsilon^{to}$
  - Mechanical behaviours shall compute the first Piola-Kirchhoff stress
  - **We adapted the logarithmic strain framework of Miehe et al**.

PROBLEM: HOW WAS THE MECHANICAL BEHAVIOUR IDENTIFIED ?



- Most cladding behaviours are identified using pointwise models that approximate the behaviour of the pipe
- However, when a behaviour identified this way is put back in a FE solver, **discrepancies appears** !

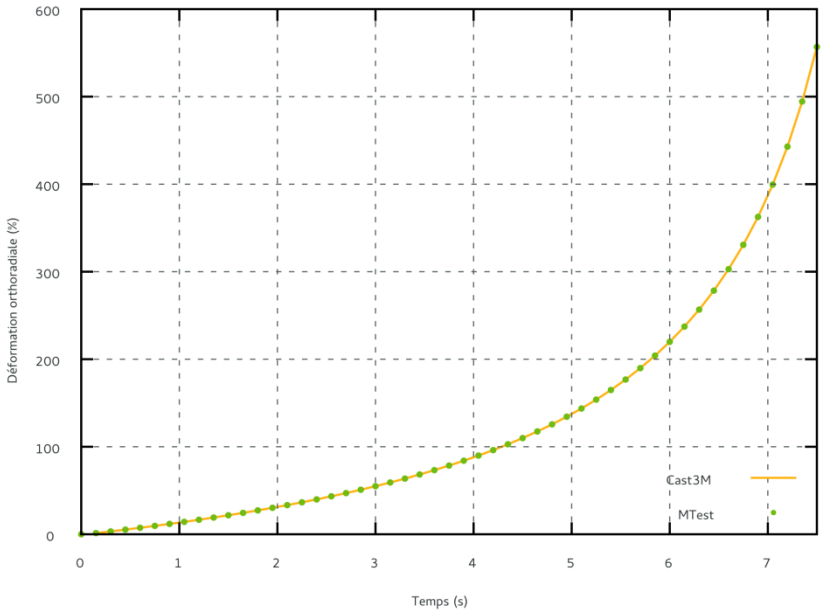
A consistent approach is mandatory

EXTENSION OF MTEST TO PIPES

Programme	Méthode de résolution	Temps d'exécution
mtest (C++)	tangente cohérente	0,024s
mtest (python)	tangente cohérente	0,071s
MEPL	tangente cohérente	0,328s
INCREPL	tangente cohérente	1,161s
MEPL	accélération Cast3M	0,599s (divergence)
INCREPL	accélération Cast3M	0,570s (divergence)
PASAPAS	accélération Cast3M	3,932s (divergence)
PASAPAS (2015/COMP)	accélération Cast3M	5,805s (divergence)

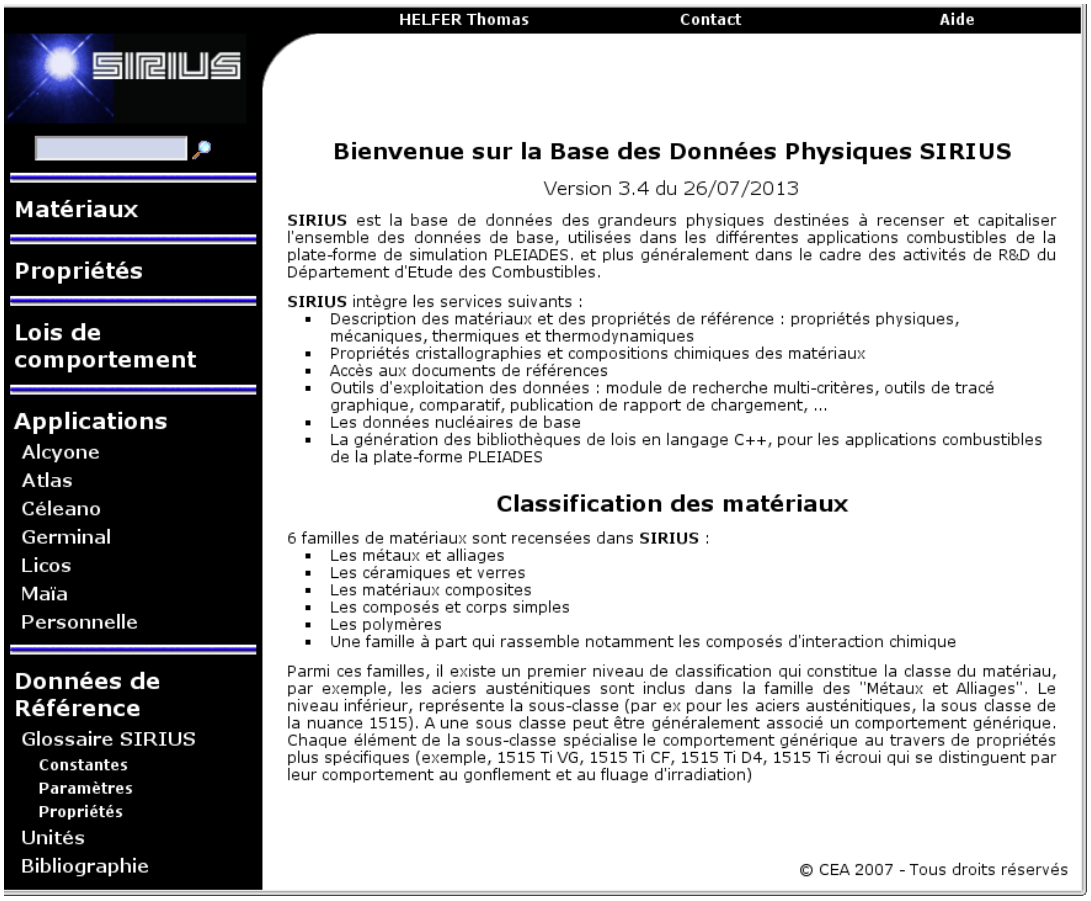
Norton in Miehe logarithmic strain framework, no effect of geometry on pressure

- MTest has been extented to pipes:
  - Proper finite strain handling
  - Performances compatible with identification
  - Python interfaces



MATERIAL KNOWLEDGE MANAGEMENT: THE SIRIUS DATABASE

MATERIAL KNOWLEDGE MANAGEMENT IN PLEIADES



- Strong emphasis on Assurance Quality
- We build a common database (Sirius) for more than100 materials

MY CURRENT VIEW

Finite element sovler / Fuel applications

Material knowlegde management project (MFront based)

- Developpement of the solver and the material knowledge management project must be **independant** and have their own assurance quality process
  - Experimental people must take car of the material knowledge manangement project
  - A Study is covered by assurance

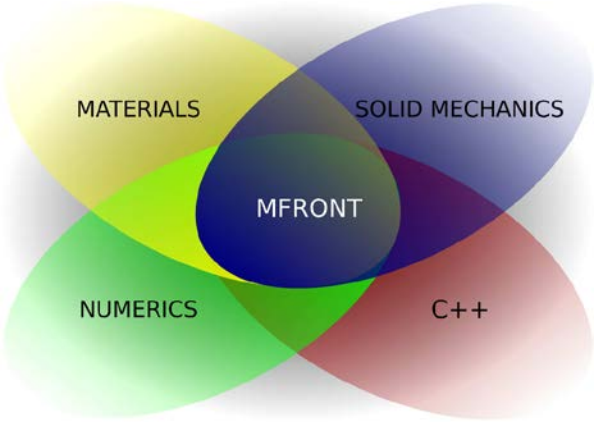
CONCLUSIONS

MFRONT IN PLEIADES CURRENT DEVELOPMENTS

- MFront plays an important role in the current thermomechanical developments of the PLEIADES platform
- **Advanced material modelling**:
  - Grain boundary decohesion coupled to damage and viscoplasticity
  - Single crystal description of UO2
- **Simplify implementation of finite strain behaviours**:
  - Single crystal implementation taking into account crystal rotation
  - Finite strain strategies:
    - « finite rotation, small strain »,
    - logarithmic strain (Miehe et al., see below)
- **Non local approach to rupture** (see below)
- **Numerical optimization of the platform**:
  - consistent tangent operator
  - local criteria to reduce or increase the time step
- More importantly:

MFront allows much better cooperation with experimental team in charge of identifying mechanical behaviours

SECOND MFRONT USER MEETING



PLEIADES

EDF

code_aster

CEA CADARACHE

Cast3M

Second MFront Users Meeting

EDF Saclay on May, 20th 2016

The second MFront Users meeting will take place on May, 20th 2016 at EDF Lab Paris Saclay and will be organized by EDF R&D.

Researchers and engineers willing to present their works are welcomed. They may contact the organizers for information at:

tfel-contact@cea.fr

Registration is *required* to ensure proper organization and will be opened on February using a dedicated form on the Code_Aster website:

http://www.code-aster.org

CALCUL QUASI-AUTOMATIQUE DE LA TANGENTE COHERENTE

Il faut calculer : $\dfrac{\partial \Delta \underline{\sigma}}{\partial \Delta \underline{\epsilon}^{\mathrm{to}}} = \left.\dfrac{\partial \underline{\sigma}}{\partial \underline{\epsilon}^{\mathrm{el}}}\right|_{\underline{\epsilon}^{\mathrm{el}}+\Delta \underline{\epsilon}^{\mathrm{el}}} : \dfrac{\partial \Delta \underline{\epsilon}^{\mathrm{el}}}{\partial \Delta \underline{\epsilon}^{\mathrm{to}}}$

On a résolu : $F\left(\Delta Y, \Delta \underline{\epsilon}^{\mathrm{to}}\right) = 0$ avec : $[\Delta Y]^T = \left[\Delta \underline{\epsilon}^{\mathrm{el}}\ ,\ \Delta\alpha\right]$

Par différentiation : $\dfrac{\partial F}{\partial \Delta Y}\,\mathrm{d}\,\Delta Y + \dfrac{\partial F}{\partial \Delta \underline{\epsilon}^{\mathrm{to}}}\,\mathrm{d}\,\Delta \underline{\epsilon}^{\mathrm{to}} = 0$

$\dfrac{\partial F}{\partial \Delta Y}$ est la jacobienne J, connue après la résolution.

Hyp. l'incrément de déformation $\Delta \underline{\epsilon}^{\mathrm{to}}$ n'apparaît que dans :

$$F_\epsilon = \Delta \underline{\epsilon}^{\mathrm{el}} + \Delta \underline{\epsilon}_i^{\mathrm{p}} - \Delta \underline{\epsilon}^{\mathrm{to}} = 0$$

donc : $J\,\mathrm{d}\,\Delta Y = -\dfrac{\partial F}{\partial \Delta \underline{\epsilon}^{\mathrm{to}}}\,\mathrm{d}\,\Delta \underline{\epsilon}^{\mathrm{to}} = \begin{pmatrix} \mathrm{d}\,\Delta \underline{\epsilon}^{\mathrm{to}} \\ 0 \end{pmatrix}$

Du 1er bloc on déduit : $\mathrm{d}\,\Delta \underline{\epsilon}^{\mathrm{el}} = J^{-1}_{\underline{\epsilon}^{\mathrm{el}}} :\ \mathrm{d}\,\Delta \underline{\epsilon}^{\mathrm{to}}$ où $J^{-1}_{\underline{\epsilon}^{\mathrm{el}}}$ est la partie supérieure gauche de J^{-1}.

Finalement, nous obtenons : $\dfrac{\partial \Delta \underline{\sigma}}{\partial \Delta \underline{\epsilon}^{\mathrm{to}}} = D :\ J^{-1}_{\underline{\epsilon}^{\mathrm{el}}}$

$J^{-1}_{\underline{\epsilon}^{\mathrm{el}}}$ est calculée par getPartialJacobianInvert