Outline

Outline

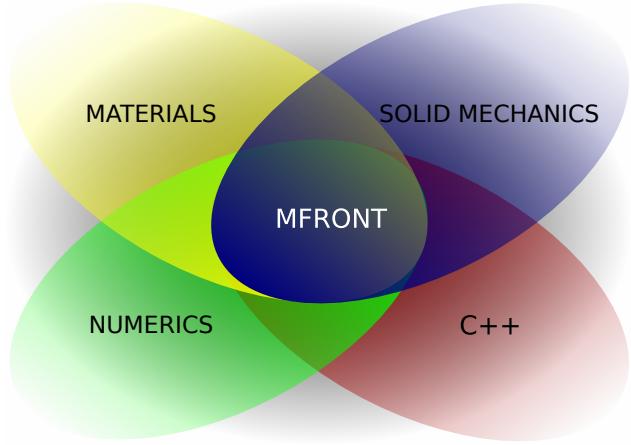
09h30 - 10h00	Accueil	
10h00 - 12h00	Faits marquants 2016 et nouveautés de la version 3.1. Exemples d'application et d'études.	T. Helfer (CEA DEC)
Pause déjeuner		
13h30 - 14h00	Comportements viscoélastiques, application aux bétons	Julien Sanahuja (EDF), Jean-Luc Adia (EDF), Benoît Bary (CEA DPC)
14h00 - 14h30	Méthodologie et outil numérique pour la prédiction de l'évolution du débit de fuite des enceintes internes de bâtiments réacteurs nucléaires	Medhi Assali (OXAND)
14h30 - 15h00	Viscoélasticimétrie par propagation d'ondes	Pierre Lemerle (INRS)
Pause		
15h15 - 15h45	Simulation numérique de la viscoplasticité à l'échelle du polycristal de dioxyde d'uranium	Luc Portelette (CEA DEC)
15h45 - 16h00	Conclusions et retour des utilisateurs	

DE LA RECHERCHE À L'INDUSTRIE

cea

www.cea.fr

TFEL and MFront 3.1



MFront User Day | Thomas Helfer, Olivier Fandeur, Thomas De Soza, Charles Toulemonde, Olivier Jamond, etc..

30 May 2017

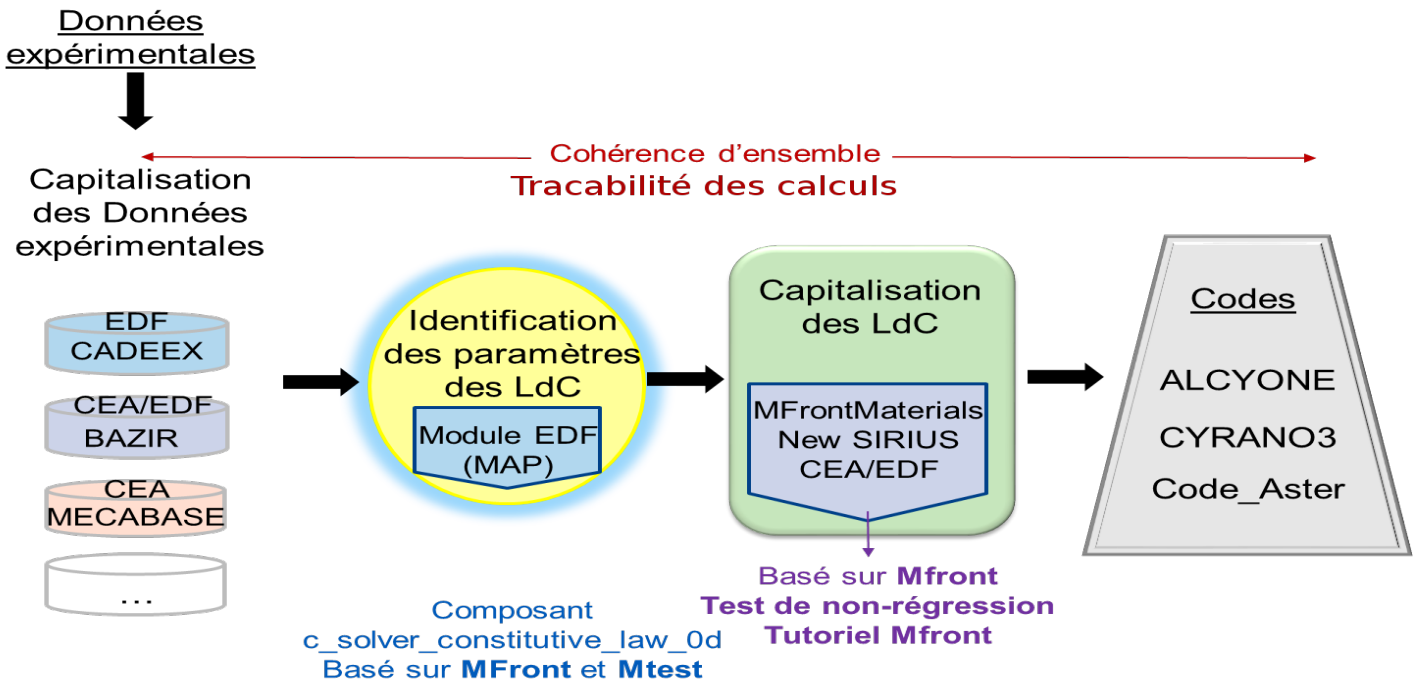
Material knowledge management

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**
- Many mechanical behaviours used in nuclear simulations are identified during Phd's under the supervision of the Centre des Matériaux.

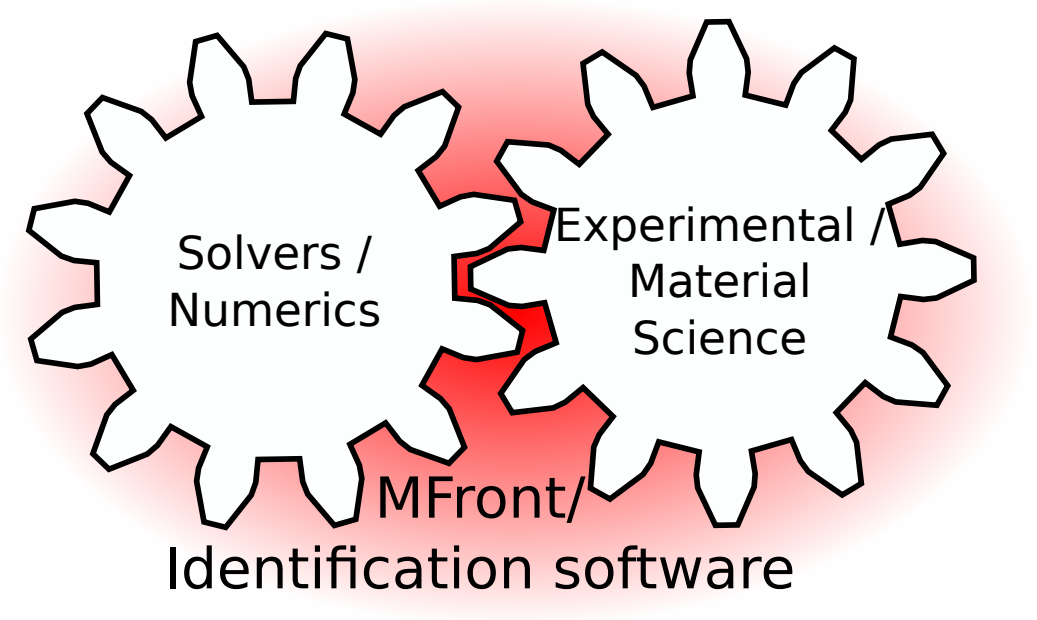
$\Rightarrow$ **Strong interest to interoperate with `ZSet` tools.**

Current situation and perspectives



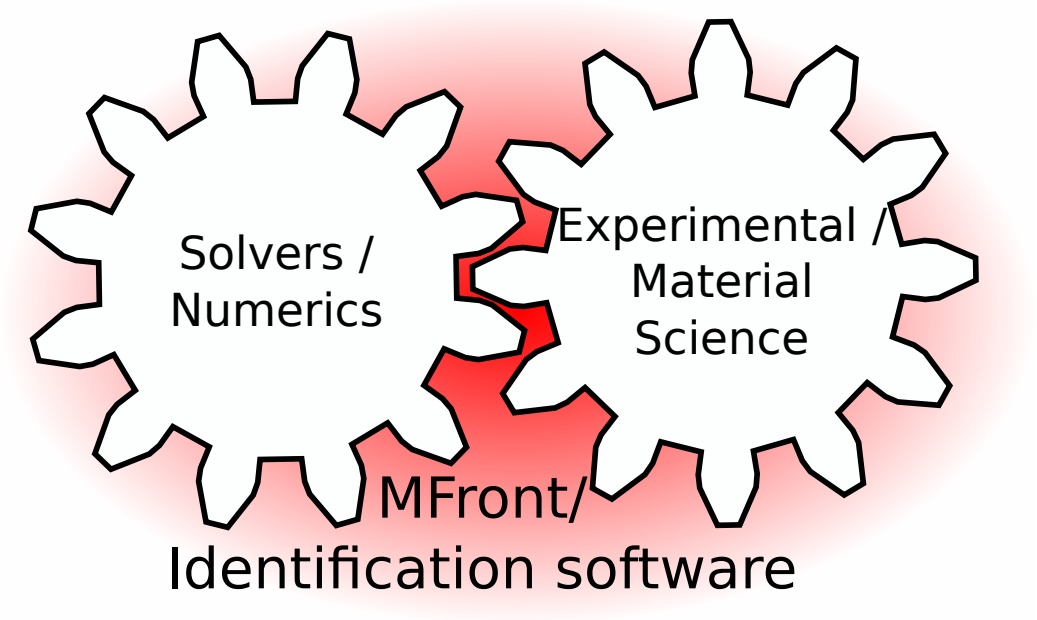
- **Rationalisation and Standardisation** is **required**, in the Nuclear industry (EDF, AREVA, CEA) :
  - Each partner has built its own strategy.
  - **A working group** has recently been set up to tackle this issue.
- However, we are willing to share this effort with the whole (french) community of the non-linear mechanics :
  - MECAMAT working group/workshop/seminar ?

A much needed dialog



- Experimental people must be in charge of identifying mechanicals behaviours

A much needed dialog



- Experimental people must be in charge of identifying mechanicals behaviours
- Solver people must provide the tools (software and theoretical background) to make a proper identification
  - Especially when advanced topics are involved (finite strain, non local models, conventions, etc...)

The TFEL project and the MFront code generator

The TFEL project

- TFEL is a project various libraries/softwares :
  - TFEL/System.
  - TFEL/Utilities.
  - TFEL/Math.
    - TFELMathCubicSpline
    - TFELMathKriging
    - TFELMathParser
  - TFEL/Material.
  - MFront
    - mfront-doc
    - mfront-query
  - MTest
  - tfel-doc

Some optimisation techniques in `TFEL/Math`

- Expression templates :
  - `const auto a=b+c;`
  - `a` is the operation of adding `b` and `c`.
  - the operation is only performed when `a` is assigned to a concrete object (lazy evaluation).
  - `a` **shall be eliminated** by the optimizer.
- Loop unrolling.
- Concepts and partial specialisations.
- Views.
- Compile-time dimensional analysis.

Some optimisation techniques in `TFEL/Math`

- Expression templates.
- Loop unrolling :
  - Consider `const stensor<1u> a=b+c+2*d;`.
  - This operation is equivalent to :

```
a[0]=b[0]+c[0]+2*d[0];
a[1]=b[1]+c[1]+2*d[1];
a[2]=b[2]+c[2]+2*d[2];
```

  - This requires to compile a specialisation of the behaviour several times for every supported modelling hypotheses.
- Concepts and partial specialisations.
- Views.
- Compile-time dimensional analysis.

Some optimisation techniques in TFEL/Math

- Expression templates.
- Loop unrolling.
- Concepts and partial specialisations :

```
template<typename T,typename T2>
typename std::enable_if<
   (( tfel ::meta::Implements<T,StensorConcept>::cond)  && (StensorTraits<T>::dime==3u)&&
    ( tfel ::meta::Implements<T2,StensorConcept>::cond) && (StensorTraits<T2>::dime==3u)&&
    ( tfel :: typetraits ::IsFundamentalNumericType<StensorNumType<T2>>::cond)),
  stensor<3u,StensorNumType<T>>
 >::type
convertCorotationnalCauchyStressToSecondPiolaKirchhoffStress(const T& s, const T2& U){
  using real = tfel :: typetraits ::base_type<StensorNumType<T>>;
  constexpr real cste = Cste<real>::sqrt2;
  const auto J  = det(U);
  const auto iU = invert(U);
  return {J*(s[2]*iU[4]*iU[4]+(cste*s[5]*iU[3]+2*s[4]*iU[0])*iU[4]+s[1]*iU[3]*iU[3]+2*s[3]*iU[0]*iU[3]+2*s[0]*iU[0]*iU[0])/2,
          J*(s[2]*iU[5]*iU[5]+(cste*s[4]*iU[3]+2*s[5]*iU[1])*iU[5]+s[0]*iU[3]*iU[3]+2*s[3]*iU[1]*iU[3]+2*s[1]*iU[1]*iU[1])/2,
          J*(s[1]*iU[5]*iU[5]+(cste*s[3]*iU[4]+2*s[5]*iU[2])*iU[5]+s[0]*iU[4]*iU[4]+2*s[4]*iU[2]*iU[4]+2*s[2]*iU[2]*iU[2])/2,
          J*((cste*s[2]*iU[4]+s[5]*iU[3]+cste*s[4]*iU[0])*iU[5]+(s[4]*iU[3]+cste*s[5]*iU[1])*iU[4]+s[3]*iU[3]*iU[3]+(2*s[1]*iU[1]+2*s[0]*iU[0])*iU[3]+2*s[3]*iU[0]*iU[1])/2,
          J*((s[5]*iU[4]+cste*s[1]*iU[3]+cste*s[3]*iU[0])*iU[5]+s[4]*iU[4]*iU[4]+(s[3]*iU[3]+2*s[2]*iU[2]+2*s[0]*iU[0])*iU[4]+cste*s[5]*iU[2]*iU[3]+2*s[4]*iU[0]*iU[2])/2,
          J*(s[5]*iU[5]*iU[5]+(s[4]*iU[4]+s[3]*iU[3]+2*s[2]*iU[2]+2*s[1]*iU[1])*iU[5]+(cste*s[0]*iU[3]+cste*s[3]*iU[1])*iU[4]+cste*s[4]*iU[2]*iU[3]+2*s[5]*iU[1]*iU[2])/2};
}
```

- Views.
- Compile-time dimensional analysis.

Some optimisation techniques in `TFEL/Math`

- Expression templates.
- Loop unrolling.
- Concepts and partial specialisations.
- Views :
  - $J = \begin{pmatrix} \frac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta \underline{\epsilon}^{el}} & \frac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta p} \\ \frac{\partial f_p}{\partial \Delta \underline{\epsilon}^{el}} & \frac{\partial f_p}{\partial \Delta p} \end{pmatrix}$
  - `dfeel_ddeel` is view of J :
    - Implementing the `ST2toST2Concept` concept.
    - Direct modification of J.
    - Compile-time computation of the offset.
- Compile-time dimensional analysis.

Some optimisation techniques in TFEL/Math

- Expression templates.
- Loop unrolling.
- Concepts and partial specialisations.
- Views.
- Compile-time dimensional analysis :
  - Adding a strain and a stress tensor leads to a **compile-time** error.
  - Available in **TFEL** for years.
  - Not yet available in MFront.
  - Feature planned for TFEL 3.1.

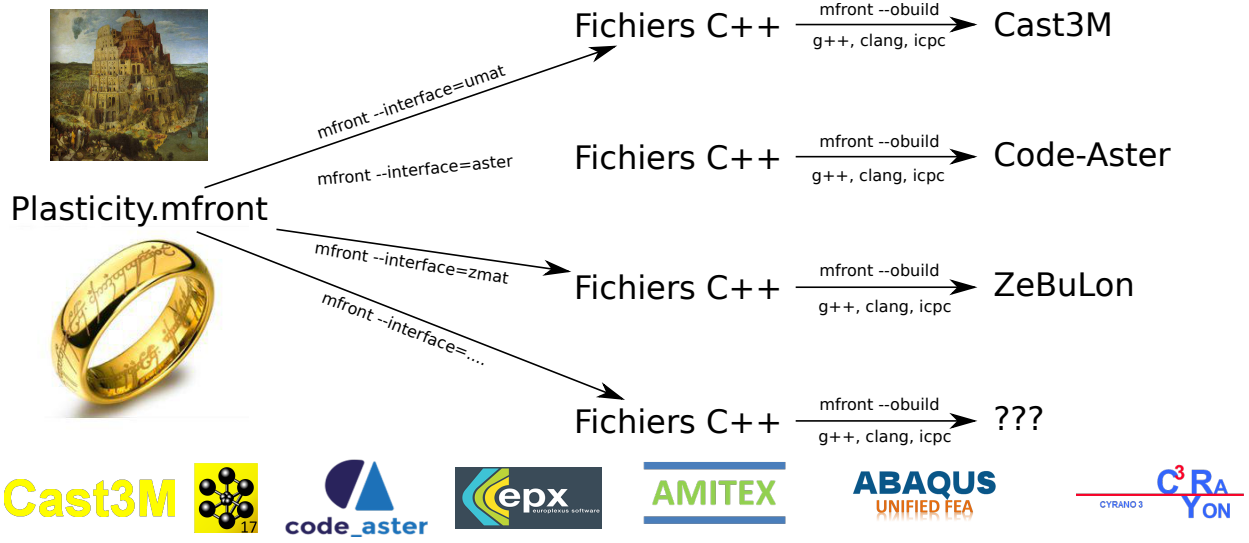
Some optimisation techniques in TFEL/Math

- Expression templates.
- Loop unrolling.
- Concepts and partial specialisations.
- Views.
- Compile-time dimensional analysis.
- **Those techniques far exceed the programming skills of most C++ professional programmers.**
  - We have anticipated the C++ language evolutions.
  - Some parts of TFEL are deprecated and removed when an equivalent features are introduced in the standard.
  - Concepts will have build-in language support in C++ -20.
  - It explains the great gap between TFEL-2.0 and TFEL-3.0, but « old » MFront files are unaffected ⇒ transparent for the end users.

Some optimisation techniques in TFEL/Math

- Expression templates.
- Loop unrolling.
- Concepts and partial specialisations.
- Views.
- Compile-time dimensional analysis.
- **Those techniques far exceed the programming skills of most C++ professional programmers.**
  - We have anticipated the C++ language evolutions.
  - Some parts of TFEL are deprecated and removed when an equivalent features are introduced in the standard.
  - Concepts will have build-in language support in C++ -20.
  - It explains the great gap between TFEL-2.0 and TFEL-3.0, but « old » MFront files are unaffected ⇒ transparent for the end users.
- **A code generator is required for the library to be used by "standard" engineers ⇒ MFront.**

MFront goals



- `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).

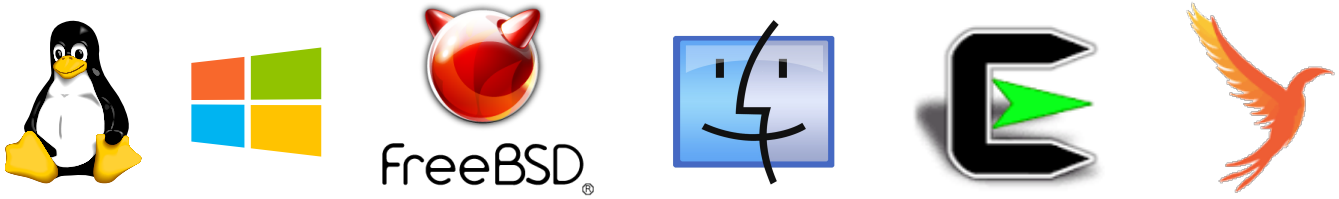
Timeline : more than 10 years of development

- 2006 : first line of codes :
  - Mainly focused on the `Cast3M` finite solver.
- 2009 : officially part of the PLEIADES project :
  - Development of the `Cyrano` interface.
- 2013 : first contact with the `Code_Aster` team.
- 2014 : TFEL-2.0 is released as an open-source project :
  - Officially co-developped by CEA and ÉDF.
  - Implementing an interface to `ZMAT`.
- 2015 : Officially part of the `Code_Aster/Salomé-Méca` distribution.
- 2016 : TFEL-3.0 is released :
  - Support of `Europlexus`, `Abaqus/Standard`, `Abaqus/Explicit`
  - $\approx$ 250 000 lines of codes

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.

Software Portability



- TFEL 3.*x* is based on the C++ 11 standard.
- TFEL 3.*x* can be compiled with various C++ compilers :
  - gcc, from version 4.7 to version 6.3 

.
  - clang, from version 3.5 to version 3.9  .
  - icc, version 2016 

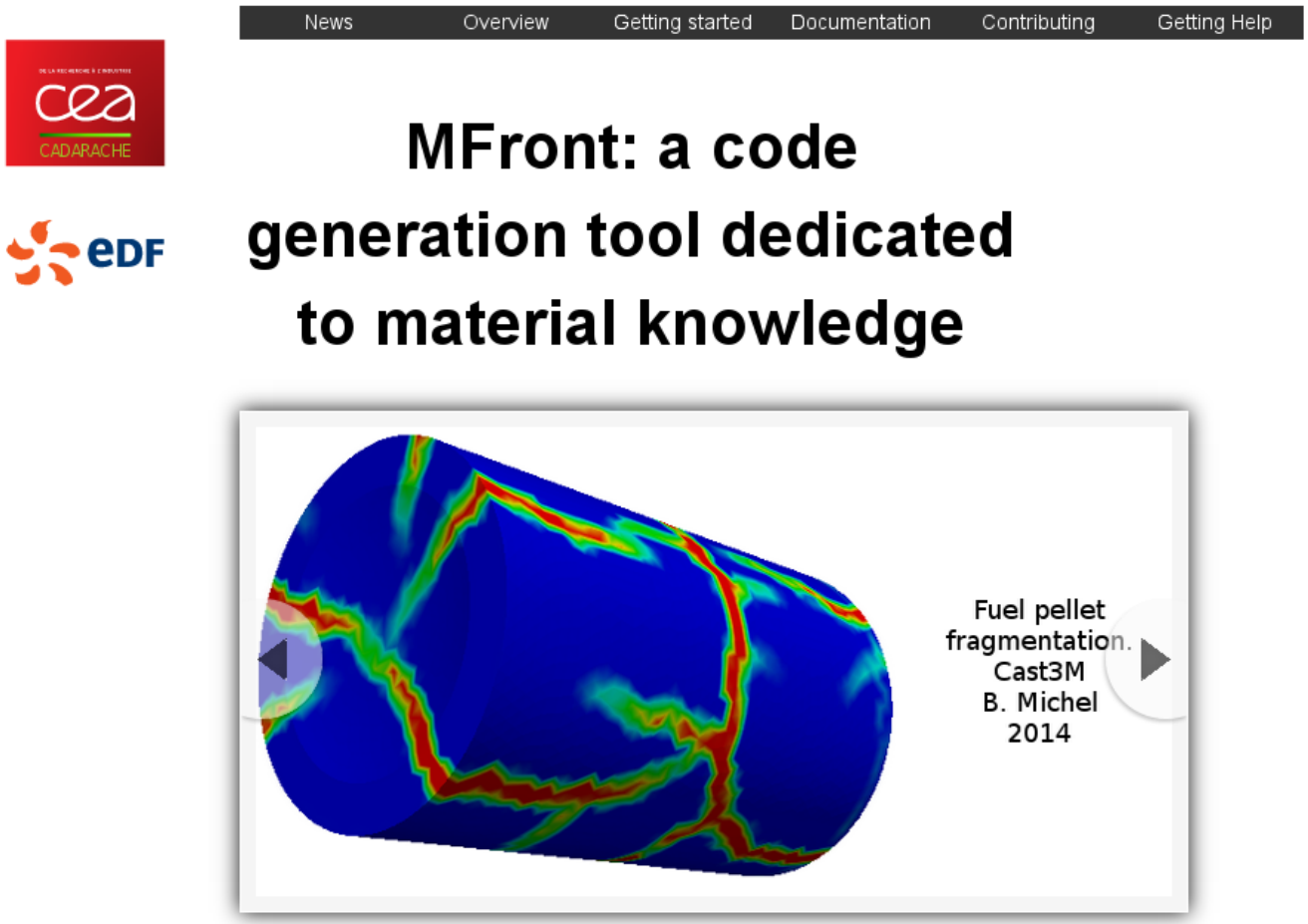
.
  - Visual Studio, version 2016 

.
- TFEL is mainly developed on LiNuX.
- ports have been made to various POSIX systems, Mac Os, FreeBSD, OpenSolaris, etc... and Windows !

Software quality : an industrial strength software

- 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 ...`
- Documentation :
  - `http://tfel.sourceforge.net/documentations.html`
  - `http://tfel.sourceforge.net/gallery.html`
  - `https://github.com/thelfer/tfel-doc`
- static analysis : `coverity`, `PVS-Studio`, `clang-tidy`, `cppcheck`, `CppDepend`, etc...
  - one metric 0.34 defect density (coverity)
- Continuous integration based on `Jenkins`
- More than 6 600 tests (mostly based on `MTest`)
- More tests inside `PLEIADES` applications and the `MFrontMaterials` project.

The website



http://tfel.sourceforge.net

http://tfel.sourceforge.net/about.html

http://tfel.sourceforge.net/gallery.html

Mechanical behaviours

Resolution

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

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

Resolution

- 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 :

$$\vec{\mathbb{F}}_i^e = \int_{V^e} \underline{\sigma}_{t+\Delta t}\left(\Delta\,\underline{\epsilon}^{to}, \Delta\, t\right) : \underline{\mathbf{B}}\, \mathrm{d}V$$

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

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

Resolution

- 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 :

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

Resolution

- 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

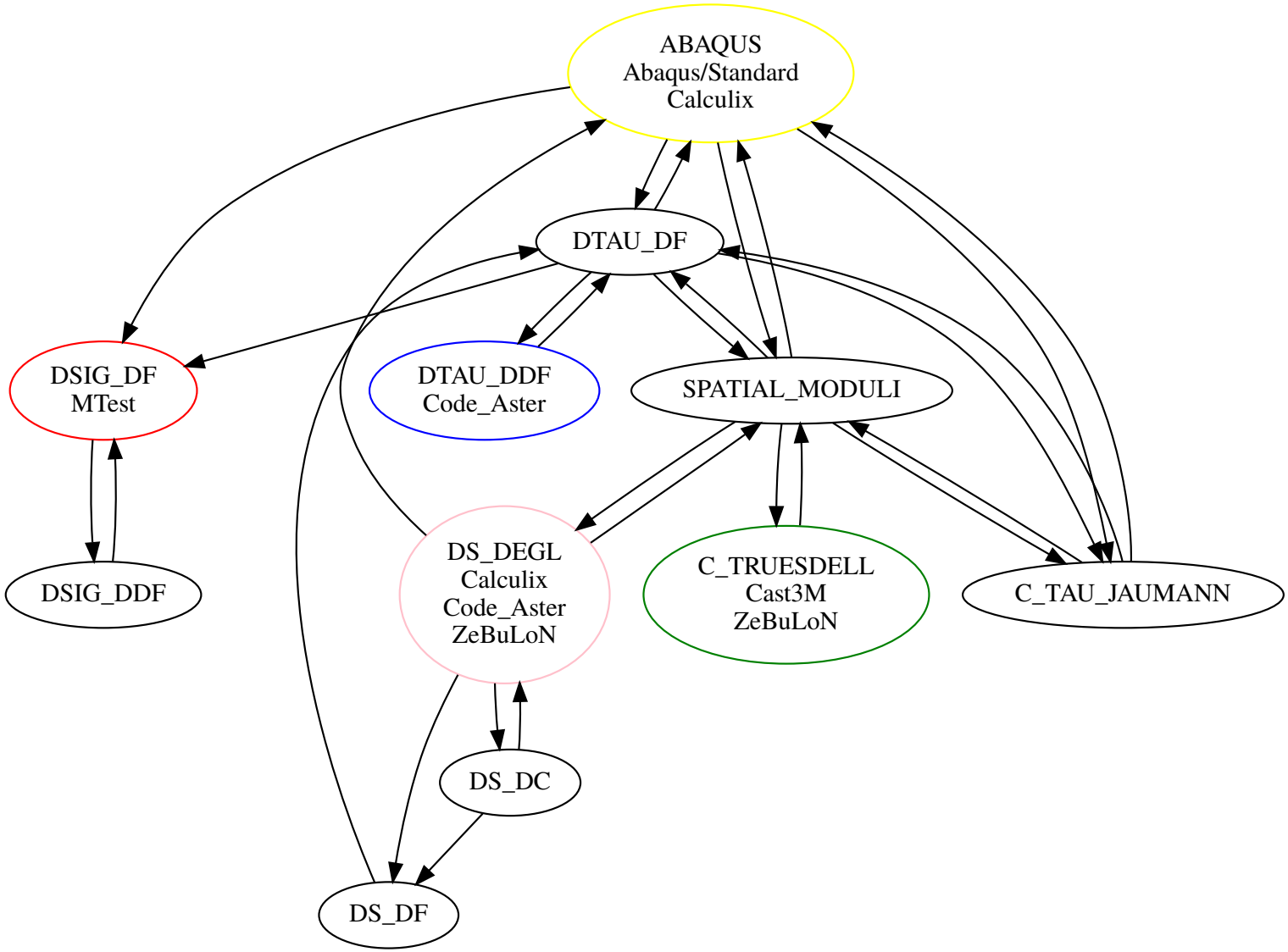
$$\left(\underline{\epsilon}^{to}\big|_t \, , \, \vec{Y}\Big|_t \, , \Delta \underline{\epsilon}^{to} \, , \Delta t \right) \underbrace{\Longrightarrow}_{behaviour} \left(\underline{\sigma}|_{t+\Delta t} \, , \, \vec{Y}\Big|_{t+\Delta t} \, , \frac{\partial \Delta \underline{\epsilon}^{to}}{\partial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

- 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)

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/Europlexus/ZMAT`
- Provide information for dynamic resolution of inputs (`MTest/Code_Aster/Europlexus`) :
  - Numbers Types (scalar, tensors, symmetric tensors)
  - Entry names /Glossary names. . .

Example of portability issue : tangent operator for finite strain behaviours



Some definitions for behaviours

- `MaterialProperty` : value describing the material that is given in the input file (used to build "generic" behaviours).

Some definitions for behaviours

- `MaterialProperty` : value describing the material that is given in the input file (used to build "generic" behaviours).
- `Parameter` : value describing the material whose default value is defined in the behaviour implementation.

Some definitions for behaviours

- `MaterialProperty` : value describing the material that is given in the input file (used to build "generic" behaviours).
- `Parameter` : value describing the material whose default value is defined in the behaviour implementation.
- `PersistentVariable` : variable saved from one step to the other (state variable from the calling code point of view).

Some definitions for behaviours

- `MaterialProperty` : value describing the material that is given in the input file (used to build "generic" behaviours).
- `Parameter` : value describing the material whose default value is defined in the behaviour implementation.
- `PersistentVariable` : variable saved from one step to the other (state variable from the calling code point of view).
- `IntegrationVariable` : variable used in the mechanical behaviour integration.

Some definitions for behaviours

- `MaterialProperty` : value describing the material that is given in the input file (used to build "generic" behaviours).
- `Parameter` : value describing the material whose default value is defined in the behaviour implementation.
- `PersistentVariable` : variable saved from one step to the other (state variable from the calling code point of view).
- `IntegrationVariable` : variable used in the mechanical behaviour integration.
- `StateVariable` : both a `PersistentVariable` and an `IntegrationVariable`.

Some definitions for behaviours

- `MaterialProperty` : value describing the material that is given in the input file (used to build "generic" behaviours).
- `Parameter` : value describing the material whose default value is defined in the behaviour implementation.
- `PersistentVariable` : variable saved from one step to the other (state variable from the calling code point of view).
- `IntegrationVariable` : variable used in the mechanical behaviour integration.
- `StateVariable` : both a `PersistentVariable` and an `IntegrationVariable`.
- `AuxiliaryStateVariable` : a `PersistentVariable` but not an `IntegrationVariable`.

Some definitions for behaviours

- `MaterialProperty` : value describing the material that is given in the input file (used to build "generic" behaviours).
- `Parameter` : value describing the material whose default value is defined in the behaviour implementation.
- `PersistentVariable` : variable saved from one step to the other (state variable from the calling code point of view).
- `IntegrationVariable` : variable used in the mechanical behaviour integration.
- `StateVariable` : both a `PersistentVariable` and an `IntegrationVariable`.
- `AuxiliaryStateVariable` : a `PersistentVariable` but not an `IntegrationVariable`.
- `ExternalStateVariable` : variable describing the material state whose evolution is not described by the mechanical behaviour.

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
  feel  += dp*n;
  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 :

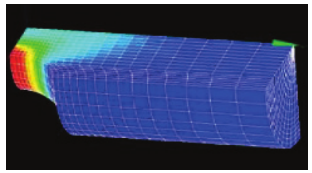
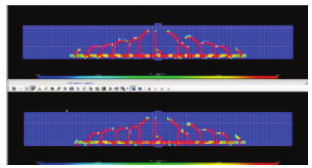
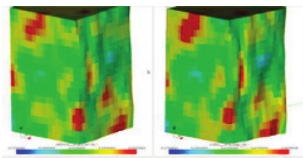
$$\begin{cases} \dfrac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta\,\underline{\epsilon}^{el}} = \underline{\underline{\mathbf{I}}} + \dfrac{2\,\mu\,\theta\,\Delta\,p}{\sigma_{eq}}\left(\underline{\underline{\mathbf{M}}} - \underline{\mathbf{n}} \otimes \underline{\mathbf{n}}\right) \\ \dfrac{\partial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

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

A more sophisticated example

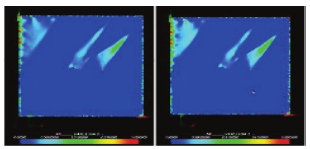
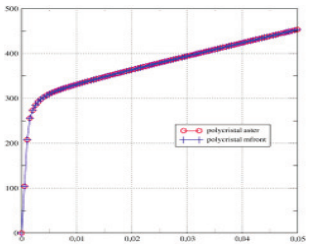
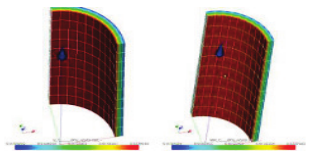
- Implementation of the Edgar behaviour :
  - taking into account evolution of elastic properties with temperature.
  - taking into account the phase transition $\alpha \rightarrow \beta$.
- Illustration of the `StandardElasticity` brick :
  - `http://tfel.sourceforge.net/BehaviourBricks.html`
- Illustration of the `@Model` keyword :
  - `mfront -help-keyword=Implicit:@Model`

Benchmarks with Code_Aster

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*	

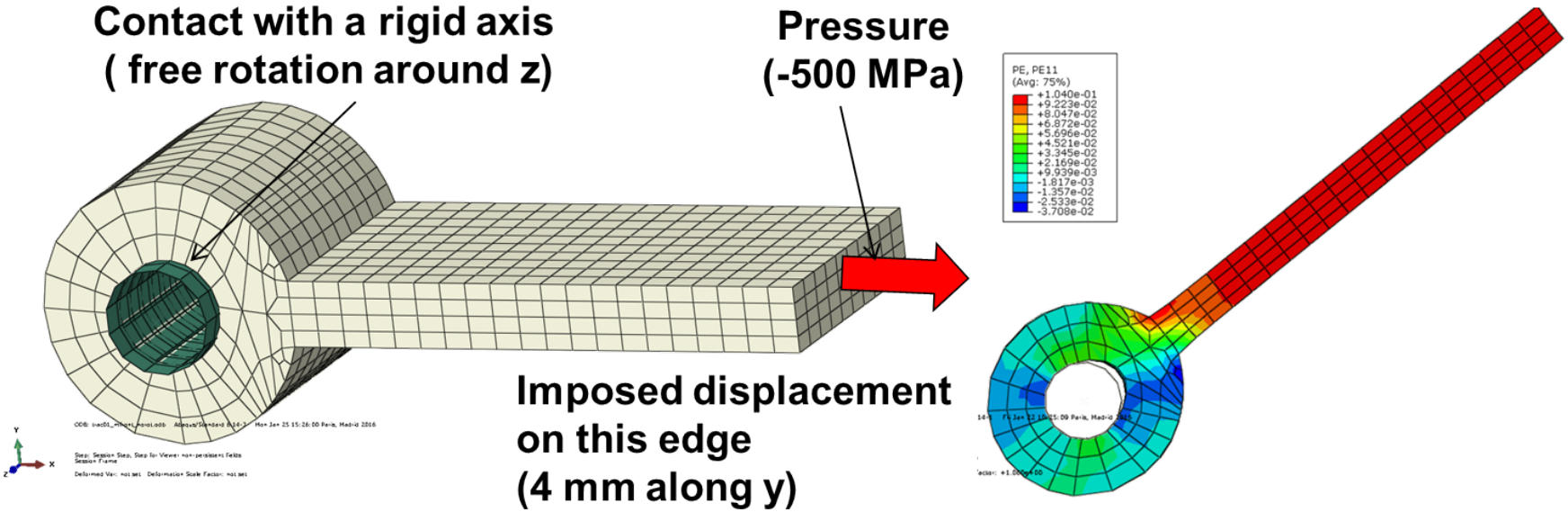
- Benchmarks made in 2013 by the Code_Aster developpers when evaluating MFront.
- Calling external behaviours in Code_Aster has improved...

Benchmarks with Code_Aster

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*	

- Benchmarks made in 2013 by the Code_Aster developpers when evaluating MFront.
- Calling external behaviours in Code_Aster has improved...

Benchmarks with Abaqus/Standard



	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!!!

Improvements to `TFEL/Math` in **TFEL** 3.1

References

- http://tfel.sourceforge.net/release-notes-2.0.5.html
- http://tfel.sourceforge.net/release-notes-3.0.1.html
- http://tfel.sourceforge.net/release-notes-3.0.2.html
- http://tfel.sourceforge.net/release-notes-3.1.html

The Releases notes are written during the development : a version can be tagged at any moment

Eigenvalues and Eigenvectors computations

Algorithm	Failure ratio	Δ_∞	Times (ns)	Time ratio
TFELEIGENSOLVER	0.000632	$7.75e-14$	252663338	1
GTESYMMETRICQREIGENSOLVER	0	$2.06e-15$	525845499	2.08
FSESJACOBIEIGENSOLVER	0	$1.05e-15$	489507133	1.94
FSESQLEIGENSOLVER	0.000422	$3.30e-15$	367599140	1.45
FSESCUPPENEIGENSOLVER	0.020174	$5.79e-15$	374190684	1.48
FSESHYBRIDEIGENSOLVER	0.090065	$3.53e-10$	154911762	0.61
FSESANALYTICALEIGENSOLVER	0.110399	$1.09e-09$	157613994	0.62

- Failure : $\Delta_\infty > 10\,\varepsilon$ with $\Delta_\infty = \max_{i\in[1,2,3]} \left\| \underline{\mathbf{s}} \cdot \vec{v}_i - \lambda_i \vec{v}_i \right\|$
- Mosty Adapted from Joachim Kopp works. Available as a library called FSES (Fast Symmetric Eigen Solver) :
  - Joachim Kopp. "Efficient numerical diagonalization of hermitian 3x3 matrices". Int.J.Mod.Phys.C19 :523-548,2008
- Usage : `const auto [vp,m] = s.computeEigenVectors<Stensor::GTESYMMETRICQREIGENSOLVER>();`
- TFELEIGENSOLVER is still the default.
- FSESJACOBIEIGENSOLVER is used for the "Miehe Apel Lambrecht logarithmic strain finite strain strategy" in all interfaces.
  - Better portability of MTest tests !

Interface improvements

```
const auto [vp,m] = s.computeEigenVectors();
const auto evp    = map([](const auto x){return exp(x)},vp);
const auto [f,df] = Stensor::computeIsotropicFunctionAndDerivative(evp,evp,vp,m,1.e-12);
```

- Many overloaded functions have been introduced.
- Very convenient when C++17 structured bindings are available.

Improvements to the overall numerical stability

- Portable implementation of the `fpclassify`, `isnan`, `isfinite` functions :
  - Replaces `glibc` broken versions :
    - `std::isnan(a)` always `false`.
    - `a!=a` always `false` (for `Fortran` users).
  - Extracted from the `MUSL` library (`https://www.musl-libc.org`).
  - One can now enable the `-fast-math` flag more securely.

Improvements to `TFEL/System` in `TFEL` 3.1

Improvements to the `ExternalLibraryManager` class

```
auto ab = std::vector<std::string>{};
const auto l = "AsterBehaviour";
auto& elm = ExternalLibraryManager::getExternalLibraryManager();
for(const auto& e : elm.getEntryPoints(l)){
  if ((elm.getMaterialKnowledgeType(l,e)==1u)&&(elm.getInterface(l,e)=="Aster")){
    ab.push_back(e);
  }
}
```

- ...

Improvements to MFront in TFEL 3.1

The `FiniteStrainSingleCrystal` brick

```
// definition of elastic material properites
@ElasticMaterialProperties {...};
// definition of sliding systems
@SlidingSystems {...};
// A model building an excellent
// approximation to the interaction matrix
@InteractionMatrix<Dupuy2017>;
```

- Based on code extracted from `Numodis`  .
- With the `FiniteStrainSingleCrystal`, complex behaviours can be build very rapidly with focus on the definition of the sliding systems and internal state variables evolution !
- Work in progress ! 

Improvements to isotropic DSLs

```
@DSL IsotropicStrainHardeningMisesCreep;
@Behaviour StrainHardeningCreep2;

@ElasticMaterialProperties {"Inconel600_YoungModulus.mfront",0.3};

@MaterialProperty real A;
@MaterialProperty real Ns;
@MaterialProperty real Np;

@FlowRule{
  const real  p0  = 1.e-6;
  const real  tmp = A*pow(seq,Ns-1.)*pow(p+p0,-Np-1);
  f          = tmp*seq*(p+p0);
  df_dseq =  Ns*tmp*(p+p0);
  df_dp    = -Np*tmp*seq;
}
```

- The `ElasticMaterialProperties` is now available for domain specific languages (DSL) describing isotropics behaviours.
- Work in progress.

Various minor improvements

- Ticket #58 : Fix initialisation of an array of parameters
- Ticket #57 : More robust detection of python versions : avoid mismatch between the python libraries, the python interpreter and the boost python library
- Ticket #54 : Bug in `StensorFromTinyVectorView` : standard assignement fails
- Ticket #53 : Add comments handling in parameters txt files
- Ticket #52 : More robust handling of parameters txt file
- Ticket #39 : Handling of Lagrange Multiplier in the `ImposedDrivingVariable` class
- Ticket #38 : Bad normalisation of the tensor when computing eigenvalues
- Ticket #35 : regression when using `@CastemFiniteStrainStrategy` or `@CastemFiniteStrainStrategies` in the `castem` interface

Improvements to `mfront-query`

About `mfront-query`

- `mfront-query` is a small tool used to retrieve various information about a `MFront` file :
  - metadata (description, author, date) ;
  - symmetry ;
  - list of material properties ;
  - list of parameters ;
  - ...

About `mfront-query`

- `mfront-query` is a small tool used to retrieve various information about a `MFront` file :
  - metadata (description, author, date) ;
  - symmetry ;
  - list of material properties ;
  - list of parameters ;
  - ...
- Mostly an illustration of how to use the `TFELMFront C++` library or the `MFront Python` module :
  - In particular, the usage of the `BehaviourDescription` and `BehaviourData`.

About `mfront-query`

- `mfront-query` is a small tool used to retrieve various information about a `MFront` file :
  - metadata (description, author, date) ;
  - symmetry ;
  - list of material properties ;
  - list of parameters ;
  - ...
- Mostly an illustration of how to use the `TFELMFront C++` library or the `MFront Python` module :
  - In particular, the usage of the `BehaviourDescription` and `BehaviourData`.
- Does not take into account interface specific features :
  - For example, additional material properties or state variables :
  - See the `mtest::Behaviour` class for that purpose.

Improvements to `MTest`

Non linear constraints

```
// impose the first piola kirchoff stress
@Real 'Pi0' -40e6
@NonLinearConstraint<Stress> 'SXX*FYY*FZZ-Pi0';
```

- Arbitrary constraints on driving variables/thermodynamic forces.
- Automatic (exact) differentiation of the constraints :
  - Misses the second derivatives for constraints involving thermodynamic forces. Quadratic convergence can't be reached, so usage of convergence acceleration algorithms is advised.
- Examples of use :
  - imposed stress triaxiality ;
  - imposed PK1 ;

Tests

- All `MTest` tests are now run 5 times :
  - one for each standard IEEE754 rounding modes (`UpWard`, `DownWard`, `ToZero`, `ToNearest`) and a `Random` mode which switches to one of the previous modes at various stages of the computation.
  - see the `-rounding-direction-mode` (`-rdm`) command line parameter.

Improvements to the `python` bindings in `TFEL` 3.1

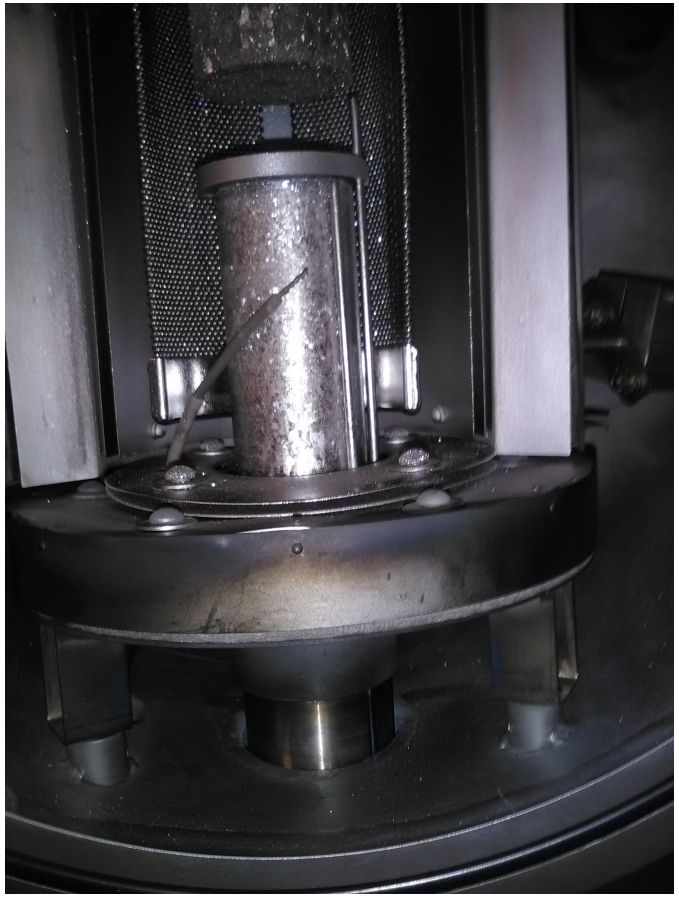
Improvements to the `python` bindings

```
import mtest
b= mtest.Behaviour('AsterBehaviour','asternorton', 'Tridimensional');
for p in b.getParametersNames():
    print('- '+p+': '+str(b.getRealParameterDefaultValue(p)))
for p in b.getIntegerParametersNames():
    print('- '+p+': '+str(b.getIntegerParameterDefaultValue(p)))
for p in b.getUnsignedShortParametersNames():
        print('- '+p+': '+str(b.getUnsignedShortParameterDefaultValue(p)))
```

- The `mtest.Behaviour` class :
  - Retrieve all the information for properly calling a behaviour in a shared library.
  - Takes into account all the additional information (material properties, internal state variables) required by the interface...

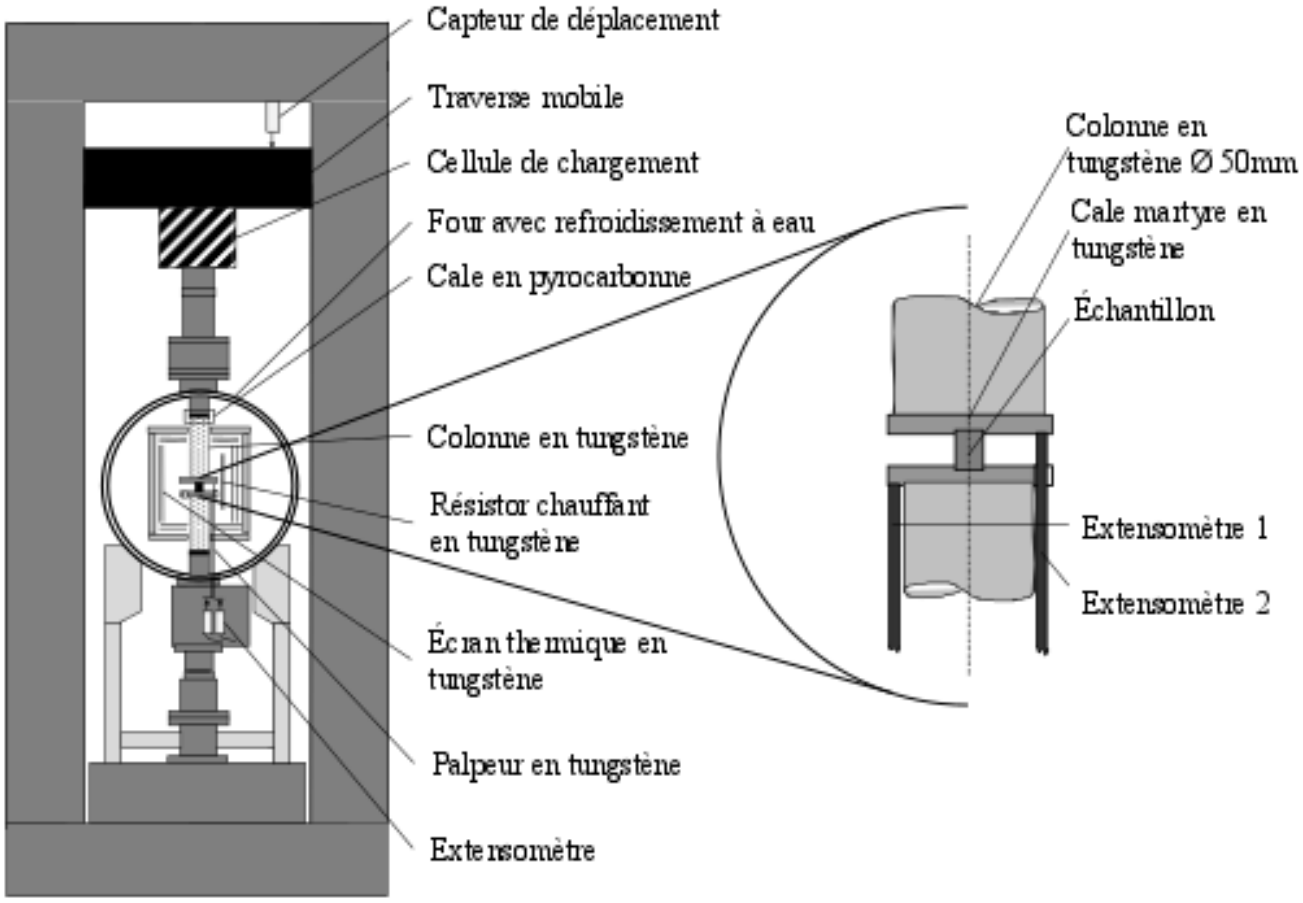
Highlights

Identification of UO_2 viscoplastic behaviour : the experimental device



- Work in collaboration with the DEC/SFER/LCU :
  - 2 internships in parallel, followed by a PhD.
- Instron 1185 (max load 50 kN).

Identification of UO_2 viscoplastic behaviour : the experimental device

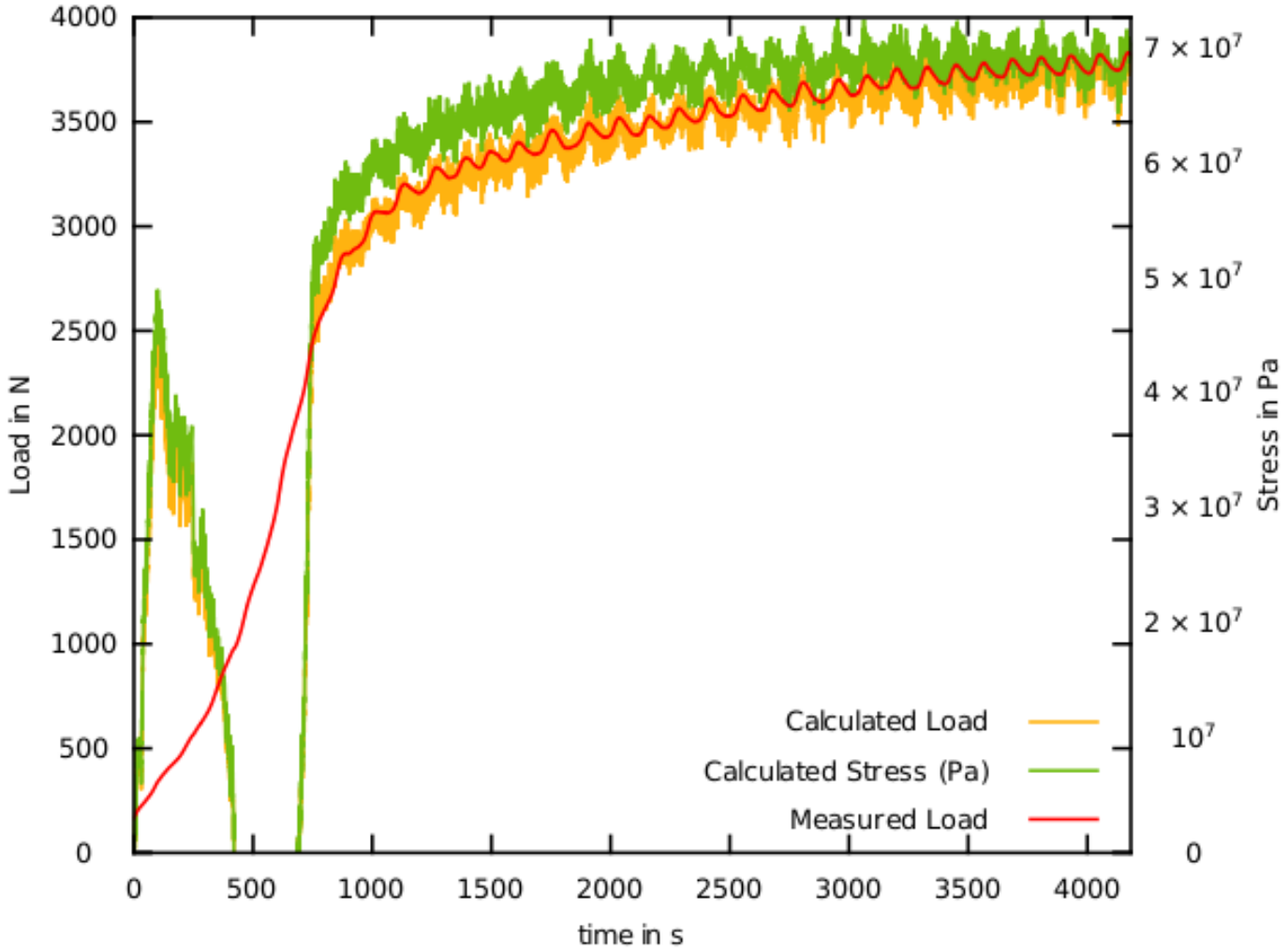


- Work in collaboration with the DEC/SFER/LCU :
  - 2 internships in parallel, followed by a PhD.
- Instron 1185 (max load 50 kN).

Identification of UO_2 viscoplastic behaviour : 1D Finite strain modelling (Imposed F_{zz})

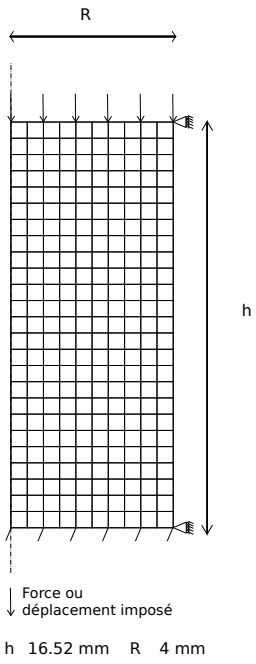
```
@Author th202608@pleiades098.intra.cea.fr;
@Date   28 fevr. 2017;
@Description{
};

@AccelerationAlgorithm 'UAnderson';
@PredictionPolicy 'LinearPrediction' ;
@MaximumNumberOfSubSteps 10;

@ModellingHypothesis 'AxisymmetricalGeneralisedPlaneStrain';
@Behaviour<castem> 'src/libUmatBehaviour.so' 'umatisotropicviscoplasticityamstrongfrederickinematichardening';
@MaterialProperty<constant> 'YoungModulus' 150e9;
@MaterialProperty<constant> 'PoissonRatio'  0.3;

// Calculation of test 868_P4
@Real 'alpha' '20.882*1.e-6/60.'; //alpha is the constant displacement rate
@Real 'h0' '16.520*1e-3'; //h0 is the height of the pellet at the beginning of test
@Real 'timemax' 8000.; //timemax is the duration of test

@ExternalStateVariable 'Temperature' 1773.15;
@ExternalStateVariable 'Porosity' 0.02;

@ImposedDeformationGradient<data> 'FRR' 'charg_868_P4.prn' using '$1*60.':'1.-$3*1.e-6/h0';

@Times<data> 'charg_868_P4.prn' using '$1*60';
```

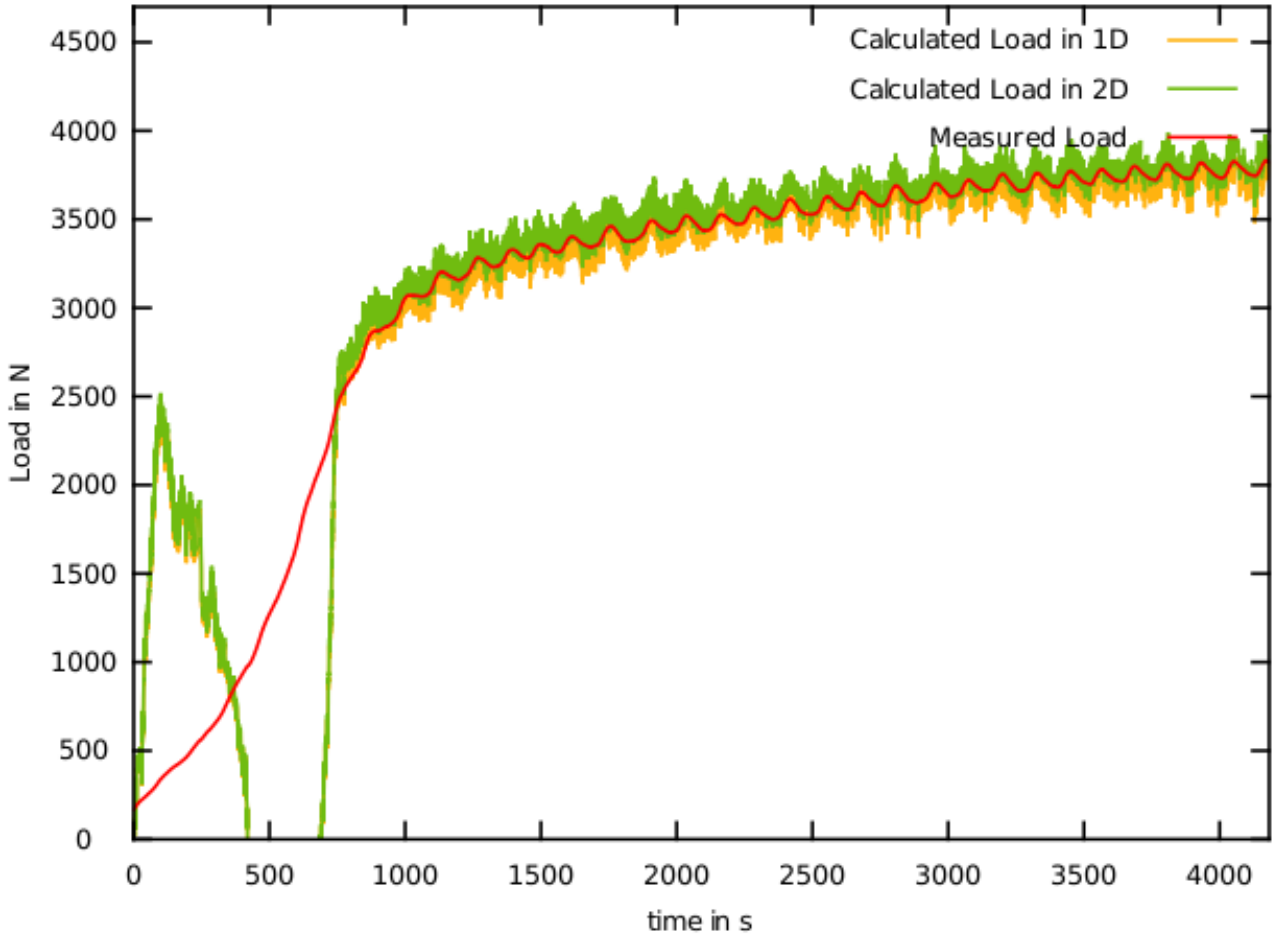
Identification of UO_2 viscoplastic behaviour : First result

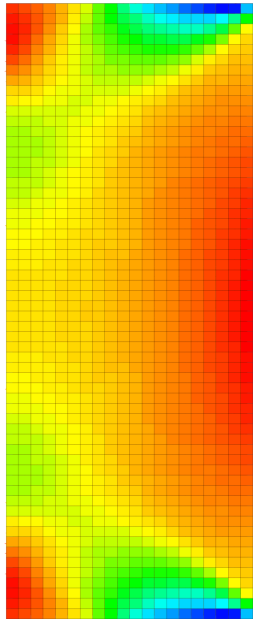


- Norton behaviour with non-linear kinematic hardening.

Identification of UO_2 viscoplastic behaviour : Comparison 1D and 2D modelling





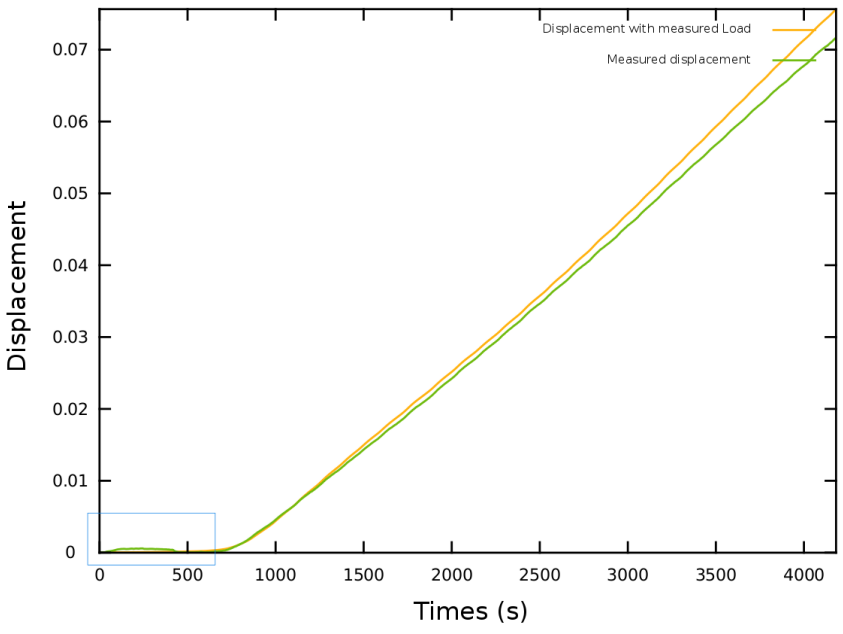


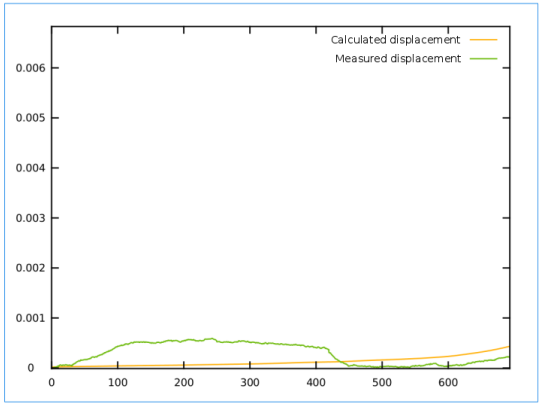
- Apparently very close results : is 2D really worth ?

Identification of UO_2 viscoplastic behaviour : 1D Finite strain modelling (Imposed Π_{zz})

```
@Author th202608@pleiades098.intra.cea.fr;
@Date   28 fevr. 2017;
@Description{

};

@AccelerationAlgorithm 'UAnderson';
@PredictionPolicy 'LinearPrediction' ;
@ModellingHypothesis 'AxisymmetricalGeneralisedPlaneStrain';
@Behaviour<castem> 'src/libUmatBehaviour.so' 'umatgarcia';
@MaterialProperty<constant> 'YoungModulus' 150e9;
@MaterialProperty<constant> 'PoissonRatio'  0.3;

@ExternalStateVariable 'Temperature' 1773.15;
@ExternalStateVariable 'Porosity'  0.02;

@Real 'S0' '3.1415*4e-3**2';
@Evolution<data> 'F' 'charg_868_P4.prn' using '$1*60.':'-$2';
@NonLinearConstraint<Stress> 'SRR*FTT*FZZ-F/S0';
@NonLinearConstraint<Stress> 'SZZ';
@NonLinearConstraint<Stress> 'STT';

@Times<data> 'charg_868_P4.prn' using '$1*60';
```

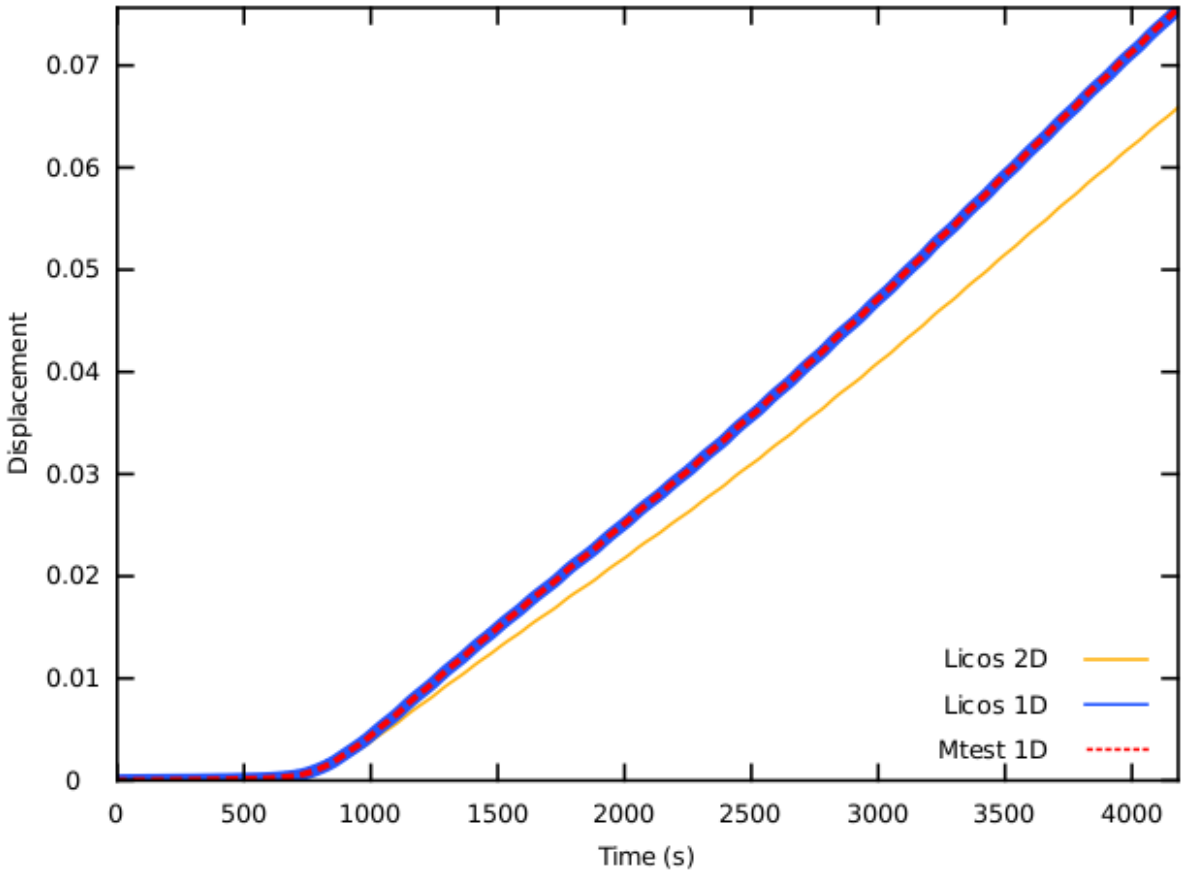
Identification of UO_2 viscoplastic behaviour : Second result





- Cumulative error => shift in time

Identification of UO_2 viscoplastic behaviour : Comparison with 2D results, imposed Π_{zz}

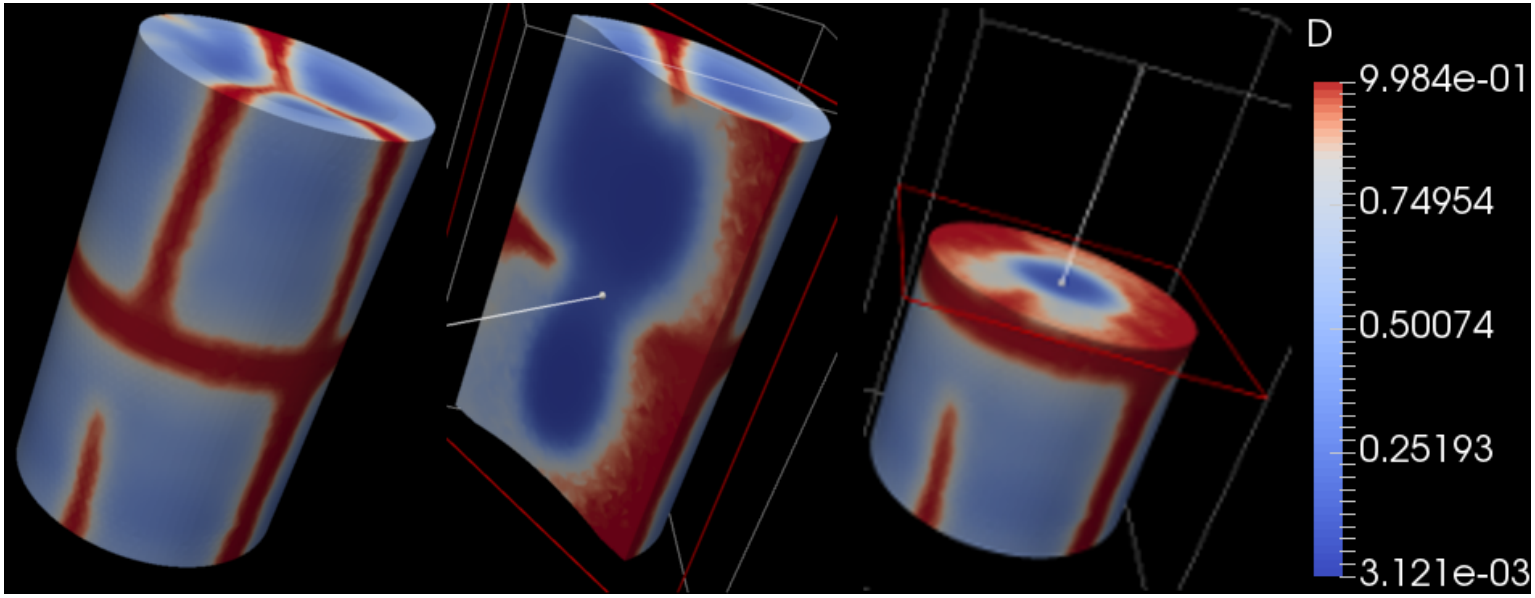


- In 2D, anchoring was assumed to maximise the differences.

Identification of UO_2 viscoplastic behaviour : perspectives

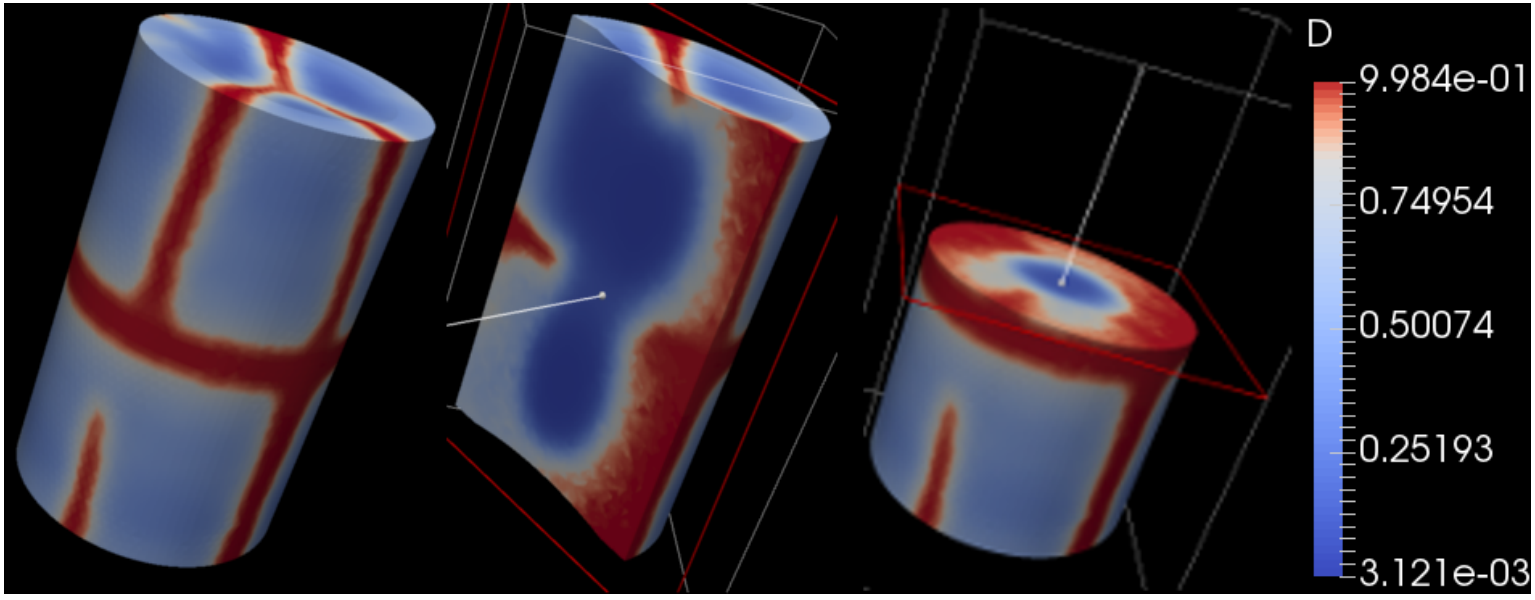
- Create a new case of study in MAP's identification component.
- Have a first identification pass in 1D.
- Try to introduce a friction coefficient as a new parameter to identify in 2D modelling :
  - Use the final shape as experimental data for the fit.
  - Quantify the gain on the reduction of the uncertainty on the behaviour's coefficients.

Phase field approach of brittle failure



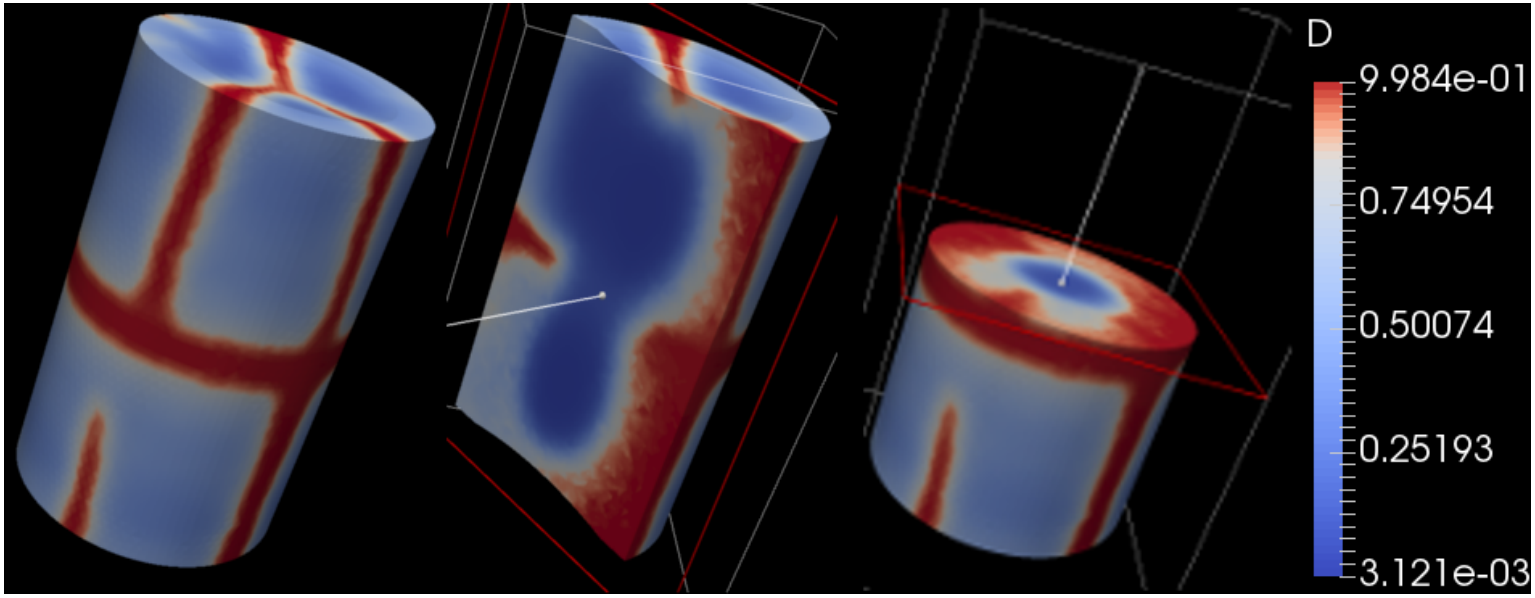
- Work in collaboration with CEA/DEN/DPC and CEA/DEN/DM2S with application to concrete and nuclear fuel.
- Initial formulation by C. Miehe, based on Marigo/Bourdin approach.

Phase field approach of brittle failure



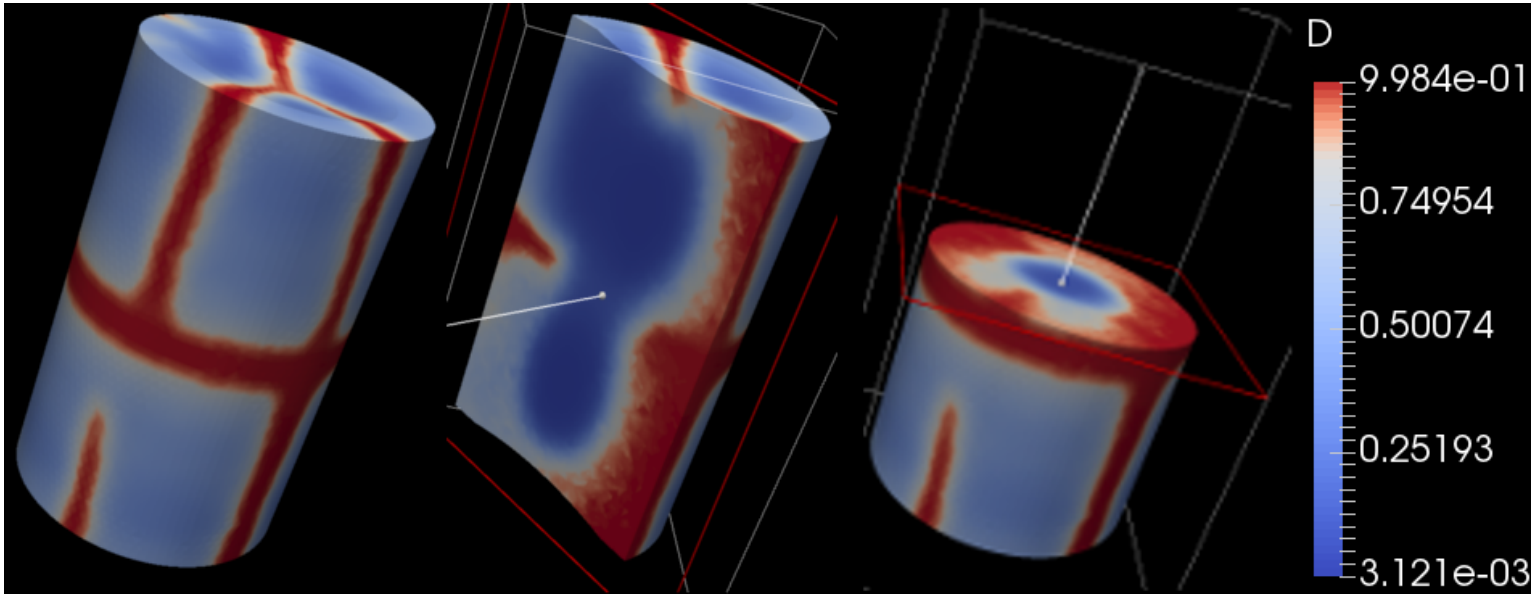
- Work in collaboration with CEA/DEN/DPC and CEA/DEN/DM2S with application to concrete and nuclear fuel.
- Initial formulation by C. Miehe, based on Marigo/Bourdin approach.
- Implementation mostly based on `MFront` and `COPL`

Phase field approach of brittle failure



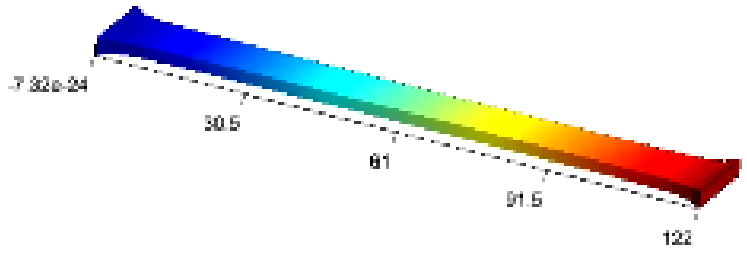
- Work in collaboration with CEA/DEN/DPC and CEA/DEN/DM2S with application to concrete and nuclear fuel.
- Initial formulation by C. Miehe, based on Marigo/Bourdin approach.
- Implementation mostly based on `MFront` and `COPL`
- Algorithm recasted in a purley implicit scheme with appropriate handling of unstable crack propagation.

Phase field approach of brittle failure



- Work in collaboration with CEA/DEN/DPC and CEA/DEN/DM2S with application to concrete and nuclear fuel.
- Initial formulation by C. Miehe, based on Marigo/Bourdin approach.
- Implementation mostly based on `MFront` and `COPL`
- Algorithm recasted in a purley implicit scheme with appropriate handling of unstable crack propagation.
- Extension to finite strain.

Hyper(visco)elasticity



Test case by T. Baranger

- A general derivation of the stress and the consistent tangent operator for this kind of hyperelastic behaviours.
  - `http://tfel.sourceforge.net/hyperelasticity.html`
- Various examples of hyperelastic behaviours described in the gallery :
  - Application to the Signorini behaviour :
    - `http://tfel.sourceforge.net/signorini.html`
  - Application to the Ogden behaviour :
    - `http://tfel.sourceforge.net/ogden.html`
- A general framework to develop hyperviscoelastic behaviours :
  - `http://tfel.sourceforge.net/hyperviscoelasticity.html`

Nonuniform Transformation Field Analysis (NTFA) : context

- *Extension of the Nonuniform Transformation Field Analysis to linear viscoelastic composites in the presence of aging and swelling*. R. Largenton, J.-C. Michel, P. Suquet. Mechanics of Materials.
- Initial form :
  - $\underline{\mathbf{S}} = \underline{\underline{\mathbf{L}}} : \underline{\mathbf{E}}^{\text{tot}} + \sum_{m=1}^{M} \underline{\overline{\rho}}_m \varepsilon_m^{\text{vp}} + \sum_{p=1}^{N} \underline{\overline{\xi}}_p \varepsilon_p^{\text{s}}$
  - Appropriate (linear) evolutions laws.

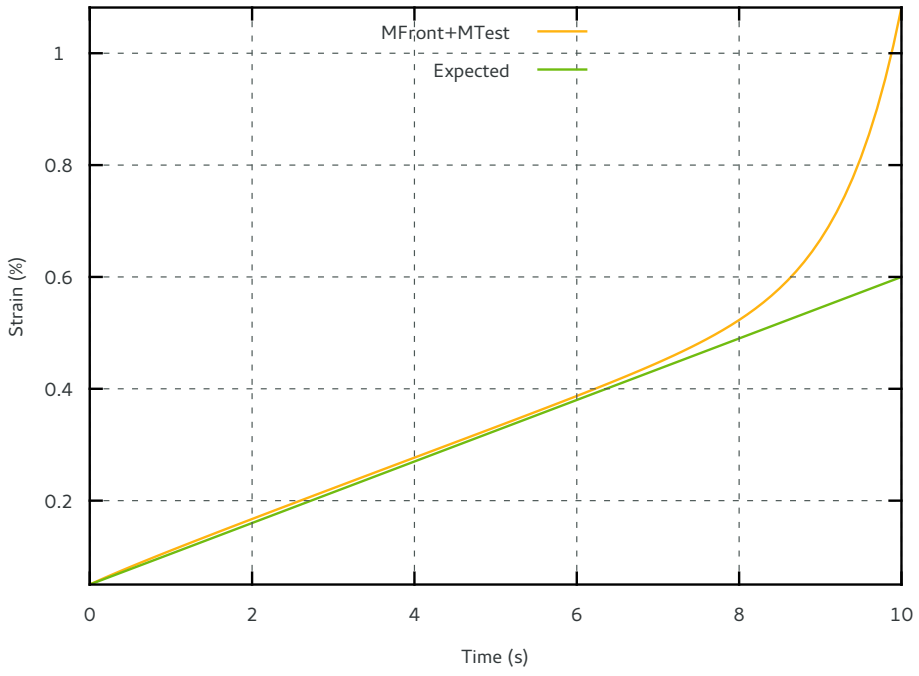
Nonuniform Transformation Field Analysis (NTFA) : context

- *Extension of the Nonuniform Transformation Field Analysis to linear viscoelastic composites in the presence of aging and swelling*. R. Largenton, J.-C. Michel, P. Suquet. Mechanics of Materials.
- Initial form :
  - $\underline{\mathbf{S}} = \underline{\underline{\mathbf{L}}} : \underline{\mathbf{E}}^{\text{tot}} + \sum_{m=1}^{M} \underline{\overline{\rho}}_m \varepsilon_m^{\text{vp}} + \sum_{p=1}^{N} \underline{\overline{\xi}}_p \varepsilon_p^{\text{s}}$
  - Appropriate (linear) evolutions laws.
- Recast for coupling with damage :

$$\begin{cases} \underline{\mathbf{S}} = \underline{\underline{\mathbf{L}}} : \underline{\mathbf{E}}^{\text{el}} \\ \underline{\mathbf{E}}^{\text{el}} = \underline{\mathbf{E}}^{\text{to}} + \sum_{m=1}^{M} \underline{\underline{\mathbf{L}}}^{-1} : \underline{\overline{\rho}}_m \varepsilon_m^{\text{vp}} + \sum_{p=1}^{N} \underline{\underline{\mathbf{L}}}^{-1} : \underline{\overline{\xi}}_p \varepsilon_p^{\text{s}} \end{cases}$$

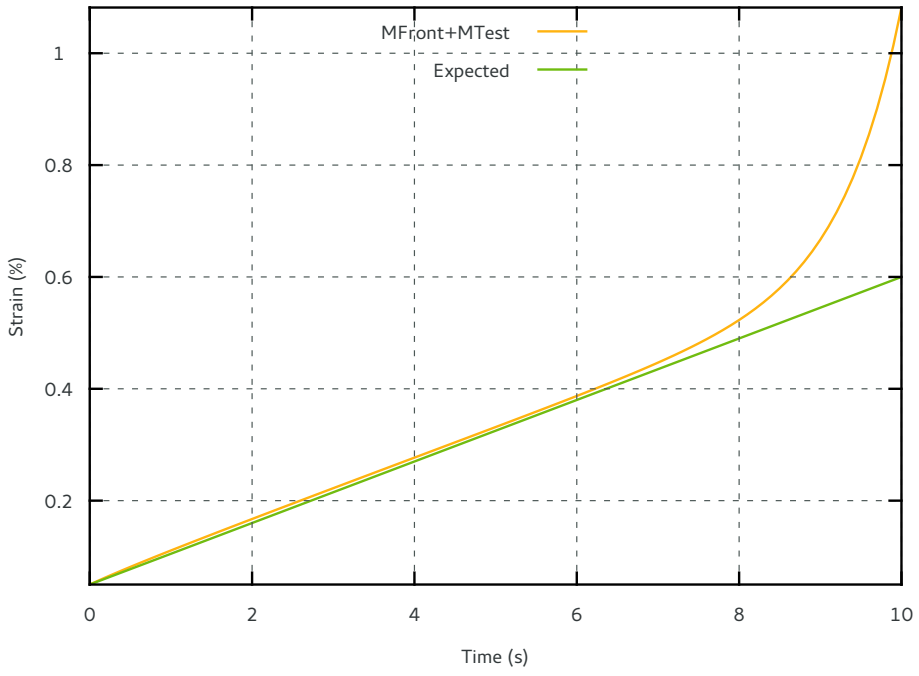
- Just a matter of linear transformations and some variable changes...

Nonuniform Transformation Field Analysis (NTFA) : context



- A simple creep test : good agreement at the beginning of the curve, then exponential discrepancy.

Nonuniform Transformation Field Analysis (NTFA) : context



- A simple creep test : good agreement at the beginning of the curve, then exponential discrepancy.
- Possible explanations :
  - Have we made mistakes while rewritting the coefficients ?
  - Have we introduce a (very) small positive eigenvalue in the evolution matrix ?

Analysis with the `verrou` tool : a floating-point rounding errors checker

- Randomly change the rounding mode of most numerical operations :
  - Based on `valgrind`.
  - Very easy to set up and very convenient to use.
  - `http://edf-hpc.github.io/verrou/vr-manual.html`
- Used for the analysis of Ticket #51 : "Variation of results between machines"

Analysis with the `verrou` tool : a floating-point rounding errors checker

- Randomly change the rounding mode of most numerical operations :
  - Based on `valgrind`.
  - Very easy to set up and very convenient to use.
  - `http://edf-hpc.github.io/verrou/vr-manual.html`
- Used for the analysis of Ticket #51 : "Variation of results between machines"
- Analysis of one implementation of the Iwan behaviour :
  - Multi-surface plasticity.
  - Numerical jacobian.
  - Loose convergence criterion (1.e-10).

Analysis with the `verrou` tool : a floating-point rounding errors checker

- Randomly change the rounding mode of most numerical operations :
  - Based on `valgrind`.
  - Very easy to set up and very convenient to use.
  - `http://edf-hpc.github.io/verrou/vr-manual.html`
- Used for the analysis of Ticket #51 : "Variation of results between machines"
- Analysis of one implementation of the Iwan behaviour :
  - Multi-surface plasticity.
  - Numerical jacobian.
  - Loose convergence criterion (1.e-10).
- `Verrou` usage exhibited the crucial role of the numerical parameter used to compute the jacobian numerically :
  - Shall be lowered to get better results (but not too much !)

Analysis with the `verrou` tool : a floating-point rounding errors checker

- Randomly change the rounding mode of most numerical operations :
  - Based on `valgrind`.
  - Very easy to set up and very convenient to use.
  - `http://edf-hpc.github.io/verrou/vr-manual.html`
- Used for the analysis of Ticket #51 : "Variation of results between machines"
- Analysis of one implementation of the Iwan behaviour :
  - Multi-surface plasticity.
  - Numerical jacobian.
  - Loose convergence criterion (1.e-10).
- `Verrou` usage exhibited the crucial role of the numerical parameter used to compute the jacobian numerically :
  - Shall be lowered to get better results (but not too much !)
- The convergence criterion can't be tighten :
  - Need of an analytical jacobian.
  - **Emphasise the need of special developments for better support of multi-surface plasticity in MFront**.

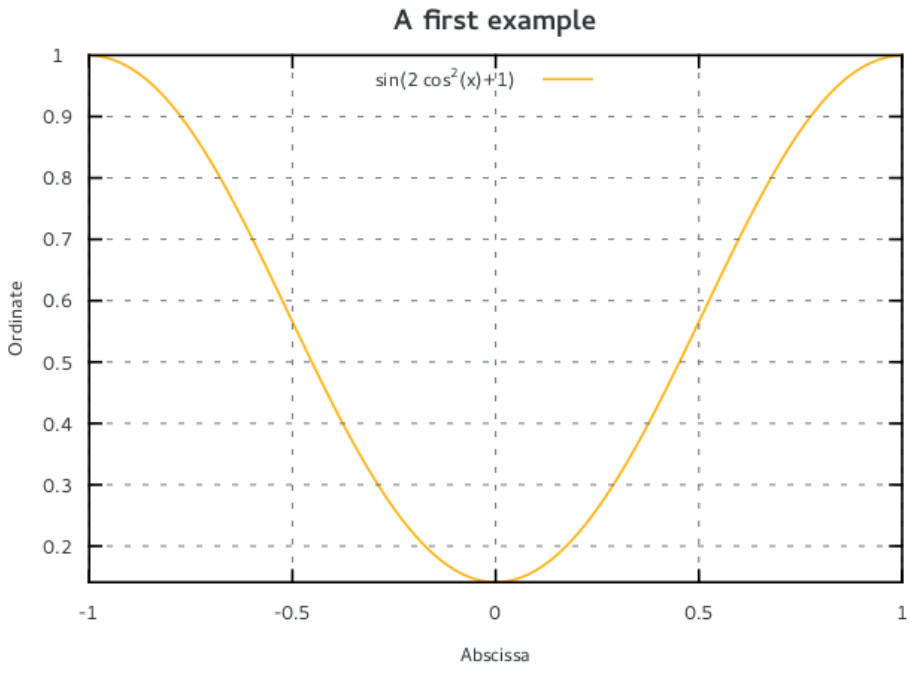
The `MFrontMaterials` projet

Goals

- A common repository between EDF, CEA and AREVA.
- Could be the principle of the a dedicated project which would gather the behaviours described in the gallery.
- Demo

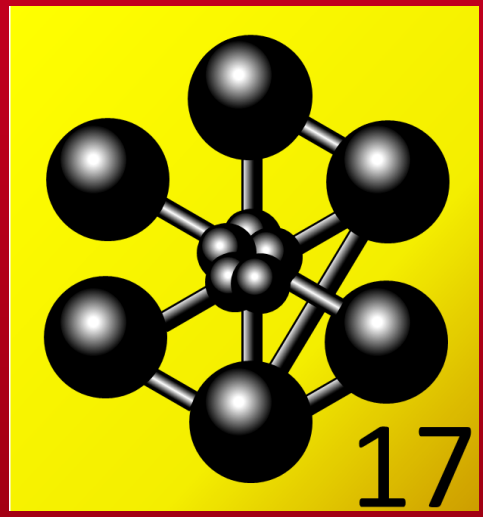
The `tfel-plot` projet

Goals



- A drop-in replacement for `gnuplot` based on `Qt`.
- `https://github.com/thelfer/tfel-plot`
- Demo

Improvements to the Cast3M interface



Current status

- Support for small and finite strain behaviours.
  - Support of the `FiniteRotationSmallStrain` and `MieheApelLambrechtLogarithmicStrain` finite strain strategies.
- Support for cohesive zone models.
- Support consistent tangent operator.
- Support for time step management from the behaviour.
- Support for time step management from the behaviour.
- Automatic generation of the input file.

The INCREPL operator

- Support for consistent tangent operator at finite strain :
  - Associated with the Truesdell rate of the Cauchy stress.
- No more support for elasticity.
- No more support various kinematics.
- No more support for stress-free imposed strain (swelling, thermal expansion, axial growth) :
  - Shall be handled by the behaviour.

Known issues

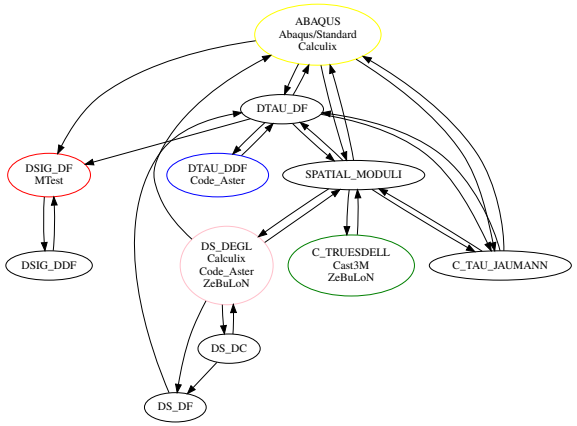
- No consistent tangent operator for the `MieheApelLambrechtLogarithmicStrain` finite strain strategies.
- Very loose coupling and `MFront` support could be improved with tighter integration :
  - Would greatly reduce the risk of user misuse.
  - See `MTest`,`Europlexus`,`Code_Aster`.

Improvements to the Code_Aster interface

Current status

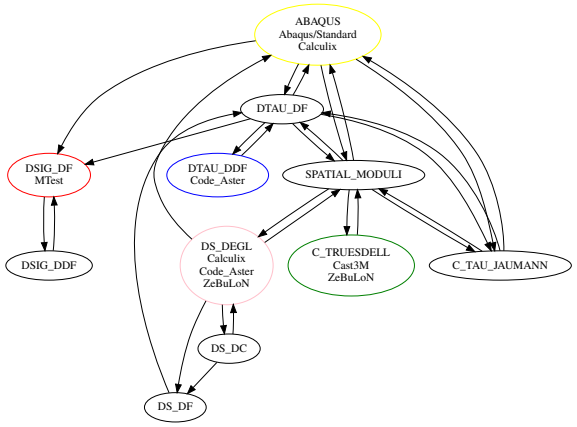
- Small and finite strain behaviours :
  - With the `SIMO_MIEHE` finite strain formulation.
- Cohesize zone models.

Support for the GROT_GDEP finite strain formulation



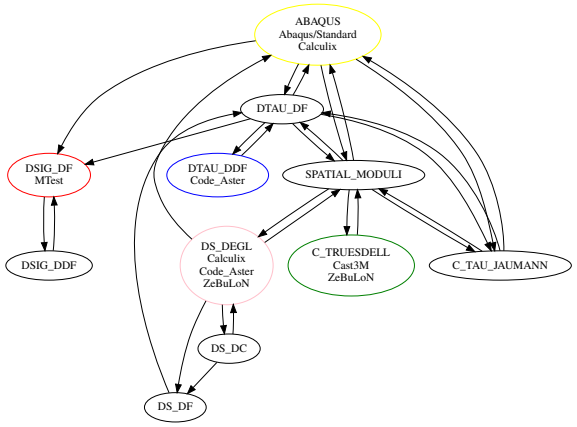
- GROT_GDEP is the name in Code_Aster of the 'Total Lagrangian' finite strain formulation.
- New keyword : @AsterFiniteStrainFormulation which must be followed by : SIMO_MIEHE, GROT_GDEP or TotalLagrangian.

Support for the GROT_GDEP finite strain formulation



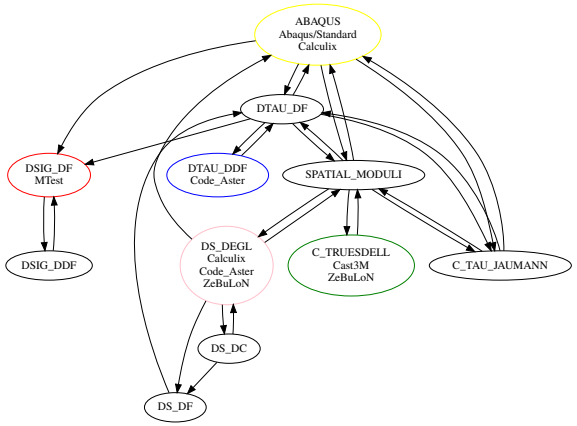
- GROT_GDEP is the name in Code_Aster of the 'Total Lagrangian' finite strain formulation.
- New keyword : @AsterFiniteStrainFormulation which must be followed by : SIMO_MIEHE, GROT_GDEP or TotalLagrangian.
- See ExternalLibraryManager::getAsterFiniteStrainFormulation

Support for the GROT_GDEP finite strain formulation



- GROT_GDEP is the name in Code_Aster of the 'Total Lagrangian' finite strain formulation.
- New keyword : @AsterFiniteStrainFormulation which must be followed by : SIMO_MIEHE, GROT_GDEP or TotalLagrangian.
- See ExternalLibraryManager::getAsterFiniteStrainFormulation
- **This shall circumvent the poor performances of the SIMO_MIEHE finite strain formulation** :
  - *"Performances and robustness :* MFront/SIMO_MIEHE *vs* Native/GROT_GDEP*"* on the Code_Aster forum.

Support for the GROT_GDEP finite strain formulation



- GROT_GDEP is the name in Code_Aster of the 'Total Lagrangian' finite strain formulation.
- New keyword : @AsterFiniteStrainFormulation which must be followed by : SIMO_MIEHE, GROT_GDEP or TotalLagrangian.
- See ExternalLibraryManager::getAsterFiniteStrainFormulation
- **This shall circumvent the poor performances of the SIMO_MIEHE finite strain formulation** :
  - *"Performances and robustness :* MFront/SIMO_MIEHE *vs* Native/GROT_GDEP*"* on the Code_Aster forum.
- Requires support on the Code_Aster side.

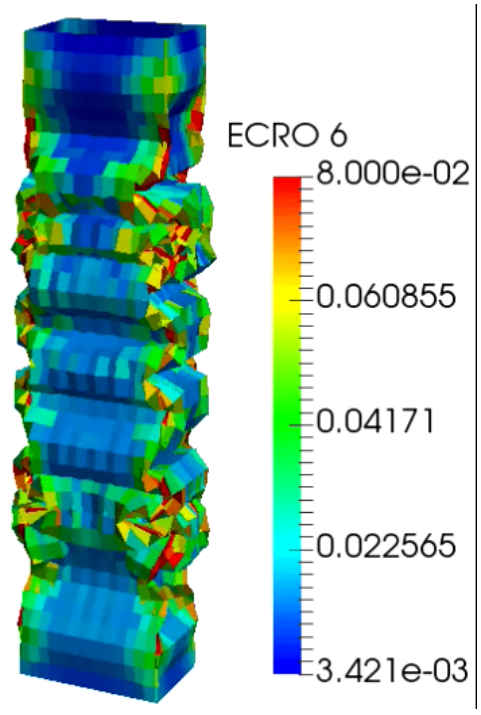
Known issues

https://bitbucket.org/code_aster/codeaster-src/issues

- Ticket #103 : Allow modification of parameters
- Ticket #100 : Restrict the finite strain strategies usable
- Ticket #99 : Minimal length in `LIST_COEF` is not meaningfull
- Ticket #98 : Check if the behaviour is small strain or finite strain
- Ticket #97 : Finite strain behaviours and `GROT_GDEP`
- Ticket #96 : Small strain behaviours and `GROT_GDEP`
- Ticket #93 : Better support of external state variables for MFront behaviours

Improvements to the `Europlexus` interface

Current status



- Support for finite strain behaviours for solid elements :
  - Support of the `FiniteRotationSmallStrain` and `MieheApelLambrechtLogarithmicStrain` finite strain strategies.
- Automatic generation of input file.
- Tight integration base on `UMAT++`.
- Orthotropic behaviour still needs to be tested.

Improvements to the ZMAT interface

Current status of the ZMAT interface

- The ZMAT interface is functional :
  - Thanks to S. Quilici and J. Besson.
  - The easiest interface to set up, up to now.
  - Support of small strain and finite strain behaviours.

Current status of the ZMAT interface

- The ZMAT interface is functional :
  - Thanks to S. Quilici and J. Besson.
  - The easiest interface to set up, up to now.
  - Support of small strain and finite strain behaviours.
- **but :**
  - No extensive and systematic testing yet.
  - No benchmarks against native behaviours.
  - No review by ZSet core developers.

Generating shared libraries

- `mfront` usage :

```
$ mfront --obuild --interface=zmat norton.mfront
Treating target : all
The following library has been built :
- libZMATBehaviour.so : Norton
```

- The previous command generates :
  - A set of C++ source files.
  - A shared library `libZMATBehaviour.so` containing one `ZMAT` class named `Norton` located in the `src` directory.

Using the generated library

- Use like any other `ZMAT` behaviour :

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

```
***behavior Norton
  **model_coef
        YoungModulus 150.e9
        PoissonRatio 0.3
        NortonCoefficient 8.e-67
        NortonExponent 8.2
***return
```

- Standard postprocessing :

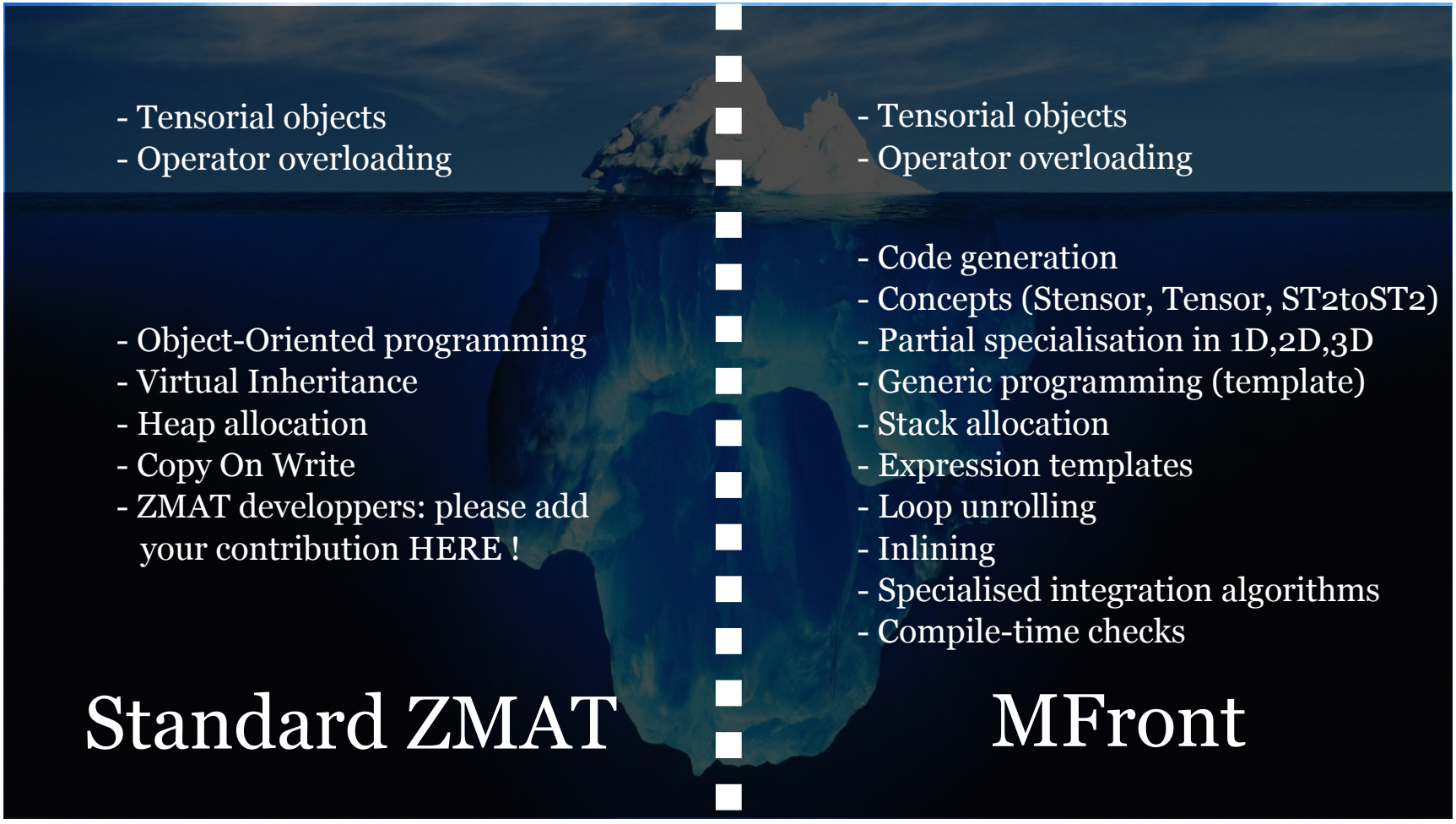
```
@StateVariable strain p;
p.setGlossaryName("EquivalentViscoplasticStrain");
```

```
***output
   **value_at_integration
   **curve
      *gauss_var
         1,1  sig22 eto22 EquivalentViscoplasticStrain
```

- **The temperature must be defined** :

```
***parameter temperature
   0.  uniform 293.15
   2.  uniform 293.15
```

Comparing standard ZMAT behaviours and MFront behaviours



- The ZMAT interface has been set up to integrate naturally Phd's work in CEA and EDF solvers.

Improvements

```
# Behaviour declaration for 3D
## List of state variables:
# - ElasticStrain (StrainStensor): elastic strain
# - EquivalentViscoplasticStrain (real): no description available
 ....
# - CriticalViscoplasticStrainEnergyDensity (real): no description available
## List of external state variables:
# temperature (temperature): the temperature
# - pres (real): no description available

***behavior M5ViscoplasticBehaviour_EdgarV3_SRMA2017
# list of parameters, which are *optional*
  PoissonRatio 0.372
  theta 0.5
  epsilon 1e-16
  iterMax 100
 ....
  brP_b 32
  arTdot_b 33
  brTdot_b 34
  minimal_time_step_scaling_factor 0.1
  maximal_time_step_scaling_factor 1.79769e+308
  numerical_jacobian_epsilon 1e-17
# Available output of bounds policies: None Strict Warning
**out_of_bounds_policy None
```

- An example of material declaration is now automatically generated.

Known issues

- Finite strain behaviours are restricted to updated lagrangian.
- Orthotropic axes management in $2D$ is not possible, which leads to inconsistent definition of the stiffness tensor, Hill tensor...
- No support for cohesive zone model.
- Some features are missing (e.g. removal of broken elements).

Improvements to the Abaqus interface

Current status

```
** Example for the 'Axisymmetrical' modelling hypothesis
*Material, name=ABAQUSBEHAVIOUR_NORTON_3D
*Depvar,
5,
1, ElasticStrain_11
2, ElasticStrain_22
3, ElasticStrain_33
4, ElasticStrain_12
5, EquivalentViscoplasticStrain
*User Material, constants=4
<YoungModulus>, <PoissonRatio>, <NortonCoefficient>, <NortonExponent>
```

- Support for Abaqus/Standard (`UMAT`) and Abaqus/Explicit (`VUMAT`)
- Support for small and finite strain behaviours (isotropic and orthotropic).

Known issues

- The generic ‘umat.cpp‘ file is not compatible with Visual Studio 2008 (Ticket #66).
- A behaviour can only be assigned to one material only (Ticket #67) :
  - A suffix to the material name is required (ABAQUS interface).
  - The generic ‘umat.cpp‘ has to be updated.
- The input file example generated by `MFront` does not take into account material property arrays (Ticket #68).
- Consistent tangent operator for the Miehe, Apel, Lambrecht logarithmic strain strategy is not available yet in Abaqus/Standard.
- Thanks to Cheng Liu.

Improvements to the `CalculiX` support

Support for external behaviours in CalculiX

```
*Solid Section, elset=v1, material=@ABAQUSNL_AbaqusBehaviour_Chaboche_3D@1
```

- Developments made on version 2.11.
- Based on the ABAQUS an ABAQUSNL interface :
  - The in-house interface is not supported yet.
- Integrated in CalculiX 2.12, accepted by Guido Dhondt.
- See the CalculiX documentation and the dedicated page on the TFEL website :

  https://tfel.sourceforce.net/calculix.html

Known issues

- A behaviour can only be assigned to one material only (Ticket #67).
- `CalculiX` can't handle more than one external behaviour properly (not `MFront` related)
- A missing information in the `KSTEP` array given by `CalculiX` the use of `MFront` generated finite strain behaviours :
  - One has to remove a check by hand in the generated sources.
- Those developments are broken under `Cygwin` :
  - `https://sourceforge.net/p/tfel/discussion/installation/thread/933a5021/`
  - But a patch exists !
- Thanks to Rafal Brzegowy.

Conclusions

On going efforts

- `MFront` is a building block for the material knowledge management strategy of `EDF` and the `PLEIADES` plateform :
  - Focused of software quality and numerical performances.
- There are ongoing efforts on defining open tools to ensure a fully-consistent strategy encompassing all the steps up to engineering studies :
  - Experimental data :
    - `https://github.com/thelfer/madnex`
  - Identification procedures.
  - Behaviour implementations.
  - Unit testing.
  - Documentation.

Blocking features

- Complete the `FiniteStrainSingleCrystal` brick :
  - Integrate code from `NUMODIS`.
- Consistent tangent operator for the `MieheApelLambrechtLogarithmicStrain` finite strain strategy.
- Complete the `ANSYS` interface.

Future works

- Documentation :
  - Better code documentation ("getting started" page).
  - New entries in the gallery.
- Numerical stability tests based on the `CADNA` library :
  - `www.lip6.fr/cadna`
  - `https://github.com/thelfer/cadna`
- Support of units :
  - Compile-time checks without any runtime overhead.
- More work on implicit algorithms :
  - Trust-region algorithms.
- New bricks :
  - Finite strain single crystal (visco-)plasticity.
- Better support of multi-surfaces plasticity.
- Generalised behaviours.
- New interfaces (ANSYS,...).
- An official Debian/Ubuntu package.

How to contribute

- Feedbacks,Feedbacks,Feedbacks !

How to contribute

- Feedbacks,Feedbacks,Feedbacks !
- Report any errors in the documentation/website.

How to contribute

- Feedbacks,Feedbacks,Feedbacks !
- Tell you colleagues about `MFront`

How to contribute

- Feedbacks,Feedbacks,Feedbacks !

- Cite the `TFEL` website and the reference paper in your publication : "Introducing the open-source mfront code generator : Application to mechanical behaviours and material knowledge management within the PLEIADES fuel element modelling platform", Computers & Mathematics with Applications. 2005.

How to contribute

- Feedbacks,Feedbacks,Feedbacks !

- Send your behaviour and your test cases.

How to contribute

- Feedbacks,Feedbacks,Feedbacks !

- Write a entry in the gallery.

Thank you for your attention.
Time for discussion !

Commissariat à l'énergie atomique et aux énergies alternatives
Centre de Cadarache | DEN/DEC/SESC b151 - 13108 Saint-Paul-Lez-Durance
T. +33 (0)4.42.25.23.66 | F. +33 (0)4.42.25.47.47
Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019

Direction de l'Énergie Nucléaire
Département d'Études des Combustibles
Service d'Études et de Simulation du comportement des Combustibles