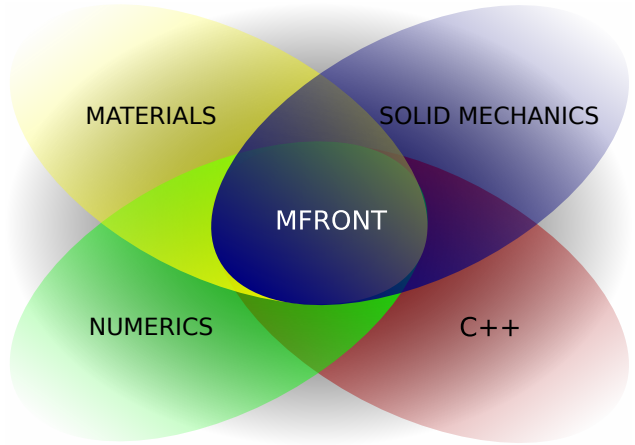
DE LA RECHERCHE À L'INDUSTRIE

Version 3.2 of the TFEL project



MFront User Days | THOMAS HELFER

16/09/2018

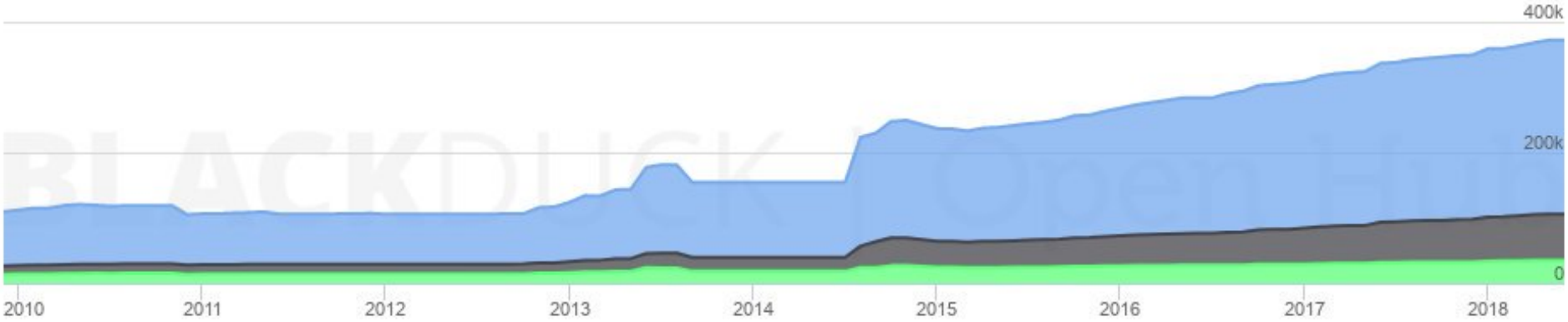
Context

DE LA RECHERCHE À L'INDUSTRIE
cea

Outline

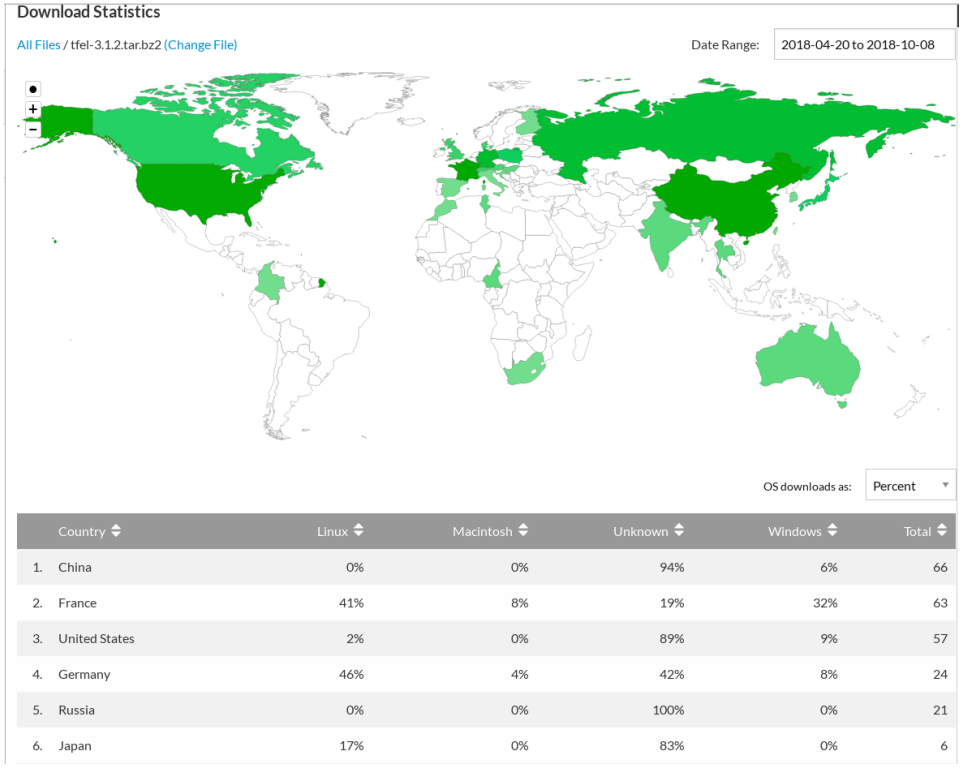
- Context
- Material knowledge management in nuclear simulations
- The `TFEL` project
- `MFront` code generator
- Development of the `Abaqus/Standard` and `Code_Aster` interfaces
- Performances
- Documentation
- TFEL 3.1.x
- TFEL 3.2
- Case of studies
- Conclusions

A few indicators



- In a Nutshell, `TFEL/MFront`
  - has had 1,760 commits made by 3 contributors representing 272,374 lines of code
  - is mostly written in `C++` with an average number of source code comments
  - has a well established, mature codebase maintained by by one developer with decreasing Y-O-Y commits
  - took an estimated 70 years of effort (COCOMO model) starting with its first commit in December, 2009 ending with its most recent commit 6 days ago

A few indicators : number of downloads



Download Statistics

All Files / tfel-3.1.2.tar.bz2 (Change File)

Date Range: 2018-04-20 to 2018-10-08

OS downloads as: Percent

	Country	Linux	Macintosh	Unknown	Windows	Total
1.	China	0%	0%	94%	6%	66
2.	France	41%	8%	19%	32%	63
3.	United States	2%	0%	89%	9%	57
4.	Germany	46%	4%	42%	8%	24
5.	Russia	0%	0%	100%	0%	21
6.	Japan	17%	0%	83%	0%	6

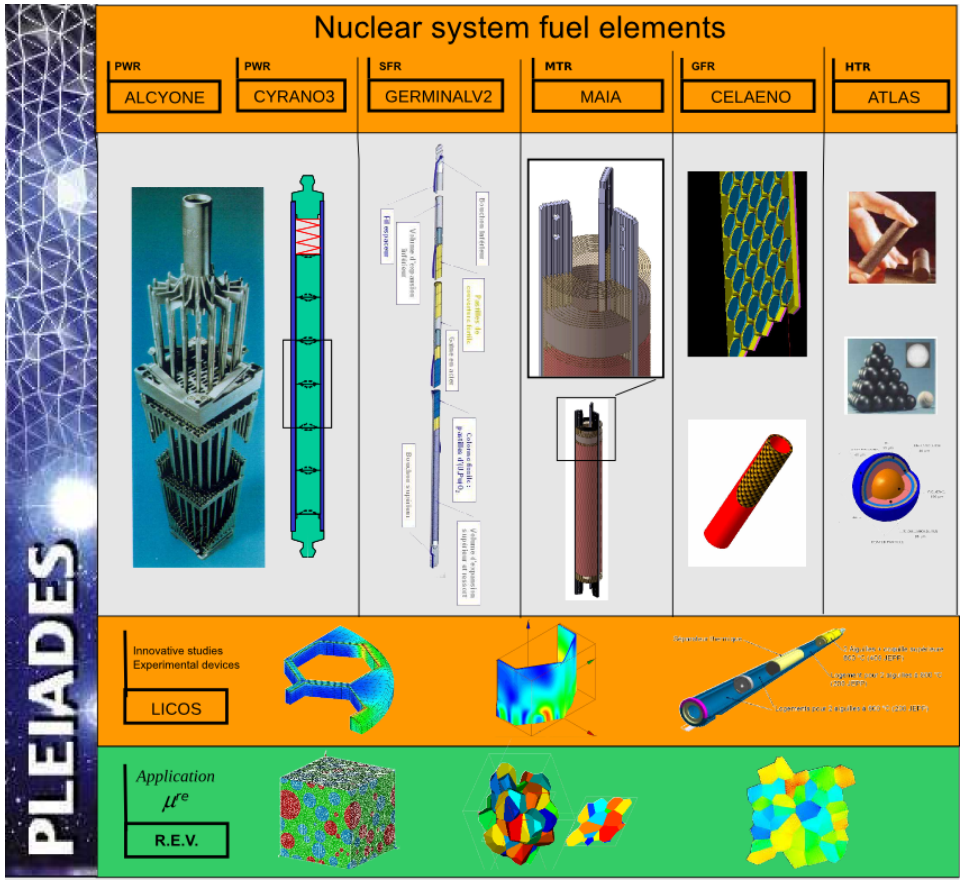
- $\approx$ 280 downloads since may 2018 fort TFEL-3.1.2
- Most users used prepackaged versions

Significant events of the year

- Twitter account :
  - `https://twitter.com/TFEL_MFront`
  - $\approx$ 80 followers
- 7 March 2018 : release of TFEL 3.1.1
- 31 May 2018 : release of TFEL 3.1.2
- Seminar at ENSMM
- Seminar at Centrale Lille
- Research gate project :
  - `https://www.researchgate.net/project/TFEL-MFront`
- July 2018 : 13th World Congress in Computational Mechanics
- 16 October : Fourth user meetings
  - Release of TFEL 3.1.3
  - Release of TFEL 3.2

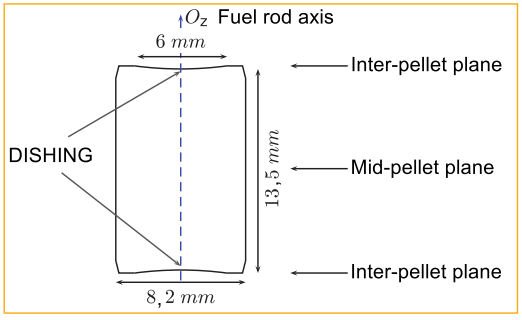
Material knowledge management in nuclear simulations

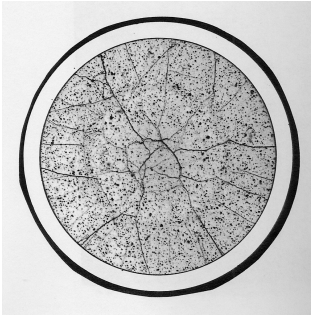
The Pleiades platform

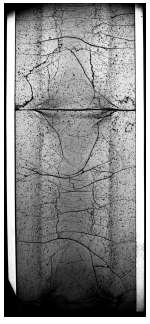


- A wide range of materials (ceramics, metals, composites).
- A wide range of mechanical phenomena and behaviours.
  - Creep, swelling, irradiation effects, phase transitions, etc..
- A wide range of mechanical loadings.

The PWR fuel element







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

Material knowledge management

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

The TFEL project

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 `Code_Aster`/`Salomé-Méca`.
- 2016 : TFEL-3.0 is released :
  - Support of `Europlexus`, `Abaqus/Standard`, `Abaqus/Explicit`
  - `StandardElasticity` brick
  - $\approx$ 250 000 lines of codes
- 2017 : TFEL-3.1 is released :
  - Experimental support of `Ansys`
- October 2018 : TFEL-3.2 is released :
  - Full support of `Ansys`, generic 'MFront' behaviour.
  - The `StandardElastoViscoplasticity` brick

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.

Libraries and executables

- Core libraries :
  - TFELConfig, TFELException, TFELSystem, TFELTests, TFELUtilities
  - TFELGlossary
  - TFELMath, TFELMathCubicSpline, TFELMathKriging, TFELMathParser
  - TFELMaterial, TFELNUMODIS, TFELPhysicalConstants

Libraries and executables

- Core libraries :
  - TFELConfig, TFELException, TFELSystem, TFELTests, TFELUtilities
  - TFELGlossary
  - TFELMath, TFELMathCubicSpline, TFELMathKriging, TFELMathParser
  - TFELMaterial, TFELNUMODIS, TFELPhysicalConstants
- MFront libraries :
  - TFELMFront, MFrontLogStream, MFrontProfiling, MTestFileGenerator
- MTest libraries :
  - TFELMTest

Libraries and executables

- Core libraries :
  - TFELConfig, TFELException, TFELSystem, TFELTests, TFELUtilities
  - TFELGlossary
  - TFELMath, TFELMathCubicSpline, TFELMathKriging, TFELMathParser
  - TFELMaterial, TFELNUMODIS, TFELPhysicalConstants
- MFront libraries :
  - TFELMFront, MFrontLogStream, MFrontProfiling, MTestFileGenerator
- MTest libraries :
  - TFELMTest
- Executables :
  - tfel-check, tfel-config, tfel-doc
  - mfront mfront-doc mfront-query
  - mtest
  - mfm

Python modules

- tfel :
  - tfel.math, tfel.material, tfel.glossary, tfel.system, tfel.utilities, tfel.tests

Python modules

- tfel :
  - tfel.math, tfel.material, tfel.glossary, tfel.system, tfel.utilities, tfel.tests
- mfront

Python modules

- tfel :
  - tfel.math, tfel.material, tfel.glossary, tfel.system, tfel.utilities, tfel.tests
- mfront
- mtest

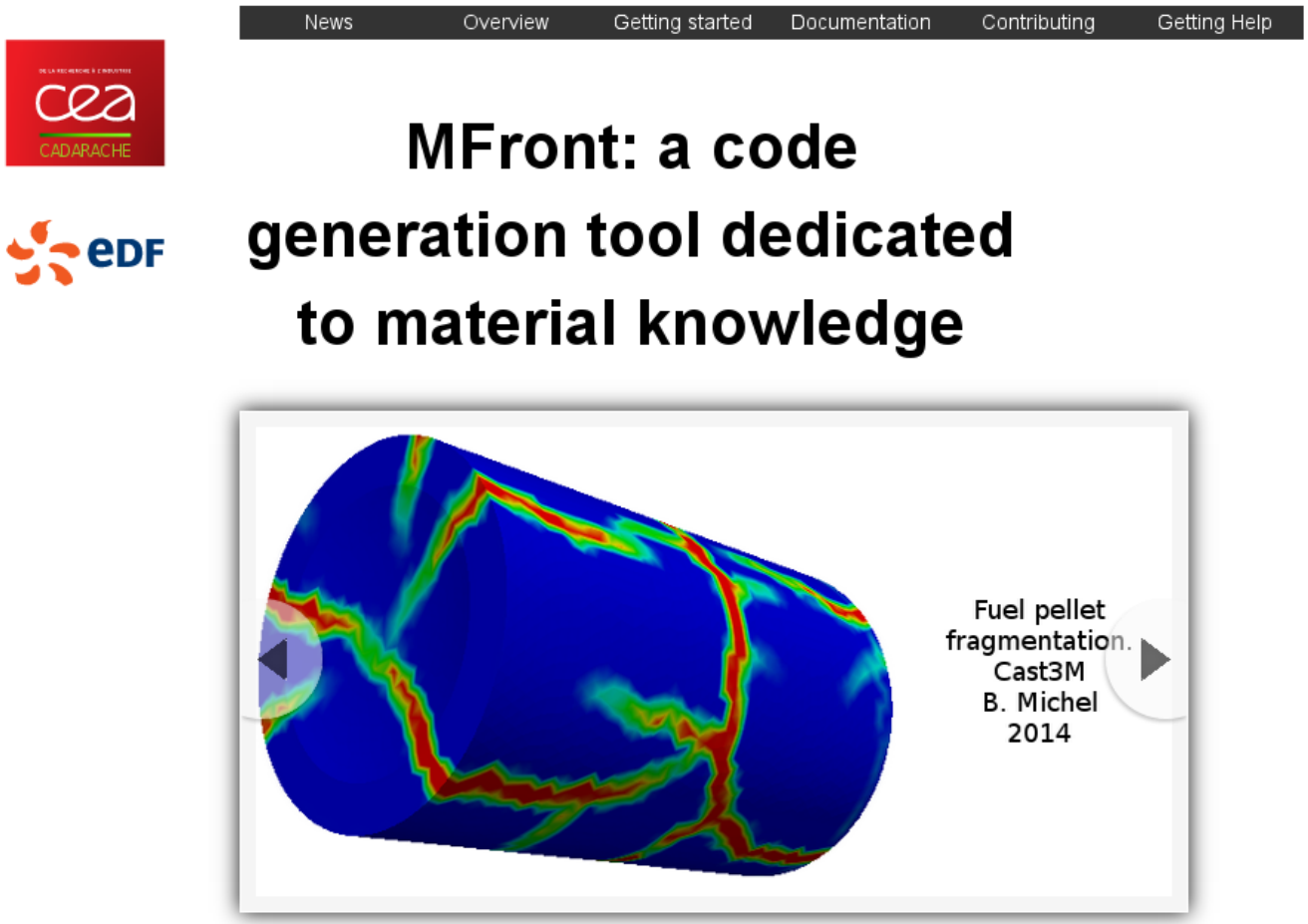
Software Portability

- 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 8.1
  - clang, from version 3.5 to version 5.0
  - icc, version 2016, 2017 and 2018.
  - Visual Studio, version 15 and 17.
- TFEL is mainly developed on LiNuX.
- ports have been made to various POSIX systems, Mac Os, FreeBSD, OpenSolaris, cygwin, Windows Subsystem for LiNuX, Haiku 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 -Wsign-promo -Wsign-conversion -Wold-style-cast -Wnoexcept -Wmissing-include-dirs -Wmissing-declarations -Wlogical-op -Winit-self ...`
- Documentation.
- Continuous integration based on `Jenkins`
- More than 10 000 tests :
  - Most of them are based on `mtest`
- More tests inside `PLEIADES` applications, `MFrontMaterials`, `MFrontGallery`, etc...

The website



http://tfel.sourceforge.net

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

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

MFront code generator

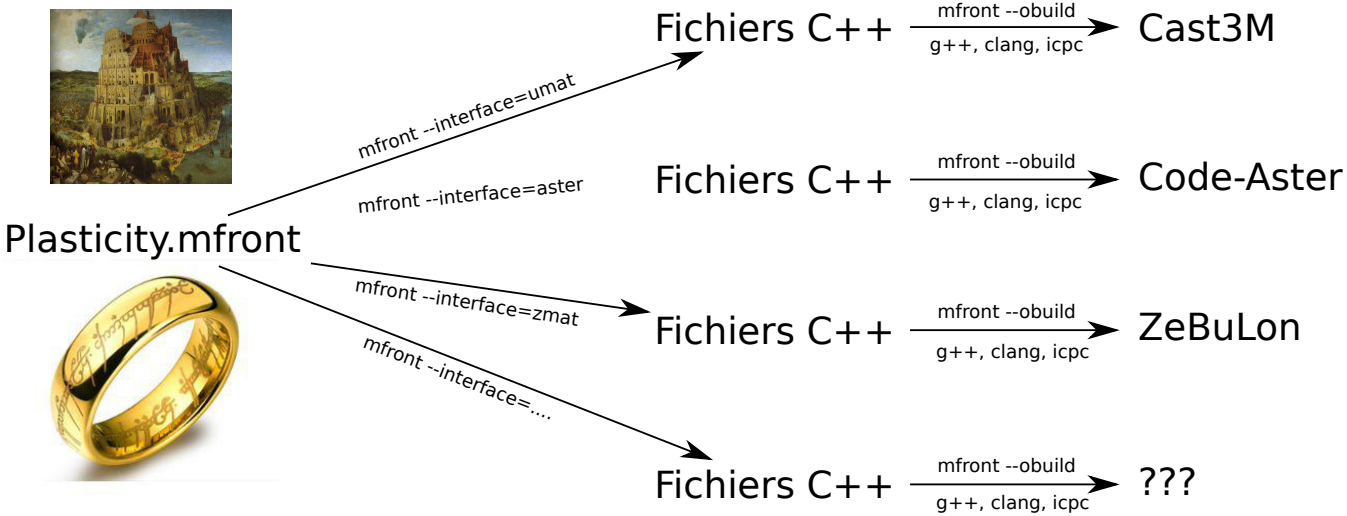
Definitions

- **State variables** : the (thermodynamical) state of a material is assumed to be fulled characterized by a set of `state variables` defined at each point of the material.
- **Model** : description of how the state variables of a set of materials evolves du to one or several physical phenomena :
  - Point-wise models
  - Equilibrium based models (heat transfer, mechanics, etc...)
- **Behaviours** : in equilibrium based models, a specific material is described by a behaviour which relates a gradient to the associated flux and makes the state variables evolves.
- **Material properties** : functions of the current state variables of the material used by some generic models or behaviours to describe a specific material.

Definitions

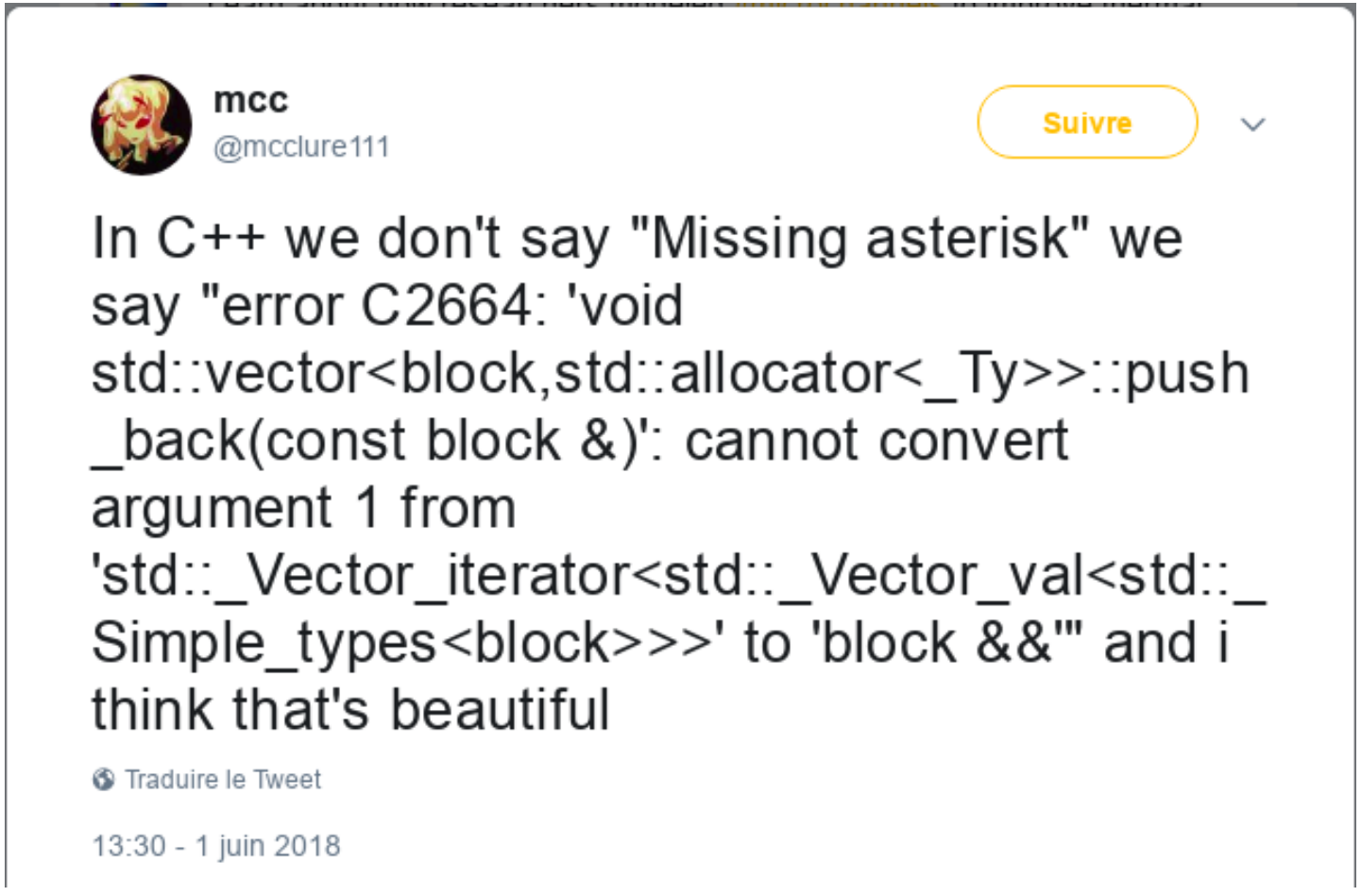
- **State variables** : the (thermodynamical) state of a material is assumed to be fulled characterized by a set of `state variables` defined at each point of the material.
- **Model** : description of how the state variables of a set of materials evolves du to one or several physical phenomena :
  - Point-wise models
  - Equilibrium based models (heat transfer, mechanics, etc...)
- **Behaviours** : in equilibrium based models, a specific material is described by a behaviour which relates a gradient to the associated flux and makes the state variables evolves.
- **Material properties** : functions of the current state variables of the material used by some generic models or behaviours to describe a specific material.

MFront goals



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

Ease of use



mcc
@mcclure111

Suivre

In C++ we don't say "Missing asterisk" we say "error C2664: 'void std::vector<block,std::allocator<_Ty>>::push_back(const block &)': cannot convert argument 1 from 'std::_Vector_iterator<std::_Vector_val<std::_Simple_types<block>>>' to 'block &&'" and i think that's beautiful

Traduire le Tweet

13:30 - 1 juin 2018

- Hopefully, only a very small subset of C/C++ is required to implement very complex behaviours.

Role of the mechanical behaviours

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

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

Role of the mechanical behaviours

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

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

- element contribution to inner forces :

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

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

Role of the mechanical behaviours

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

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

- element contribution to inner forces :

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

- Resolution using the Newton-Raphson algorithm :

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

Role of the mechanical behaviours

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

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

- element contribution to inner forces :

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

- Resolution using the Newton-Raphson algorithm :

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

- element contribution to the stiffness :

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

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

Main functions of the mechanical behaviour

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

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

An very simple example

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

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

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

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

- Implicit integration.
- Implicit system :

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

- Jacobian :

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

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

Other functions of the mechanical behaviour

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

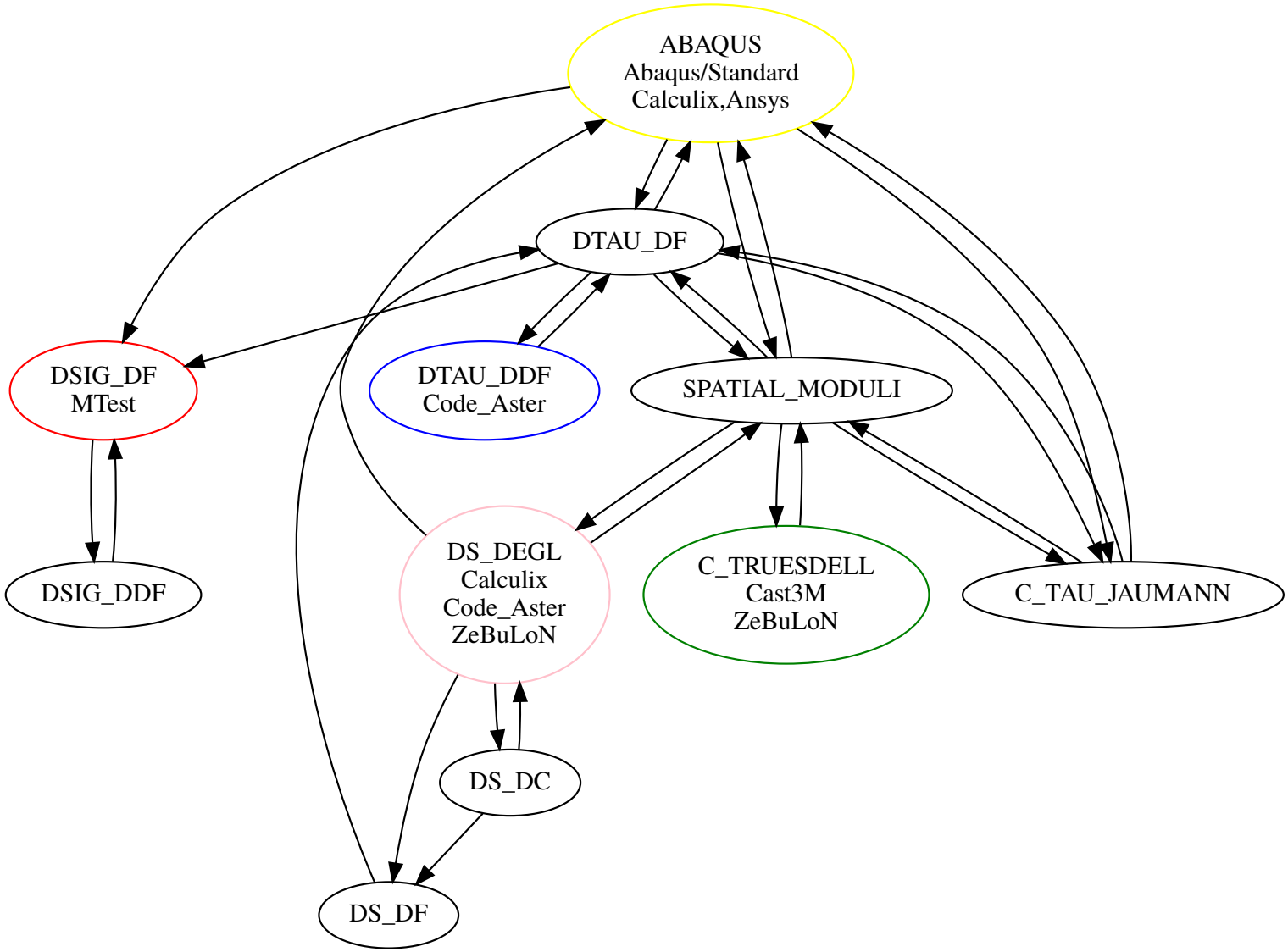
Available behaviour DSLs

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

Bricks

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

Example of portability issue : tangent operator for finite strain behaviours



Development of the Abaqus/Standard and Code_Aster interfaces

The `Abaqus/Standard` interface

- **No work on the solver side**
- The `Abaqus/Standard` interface supports :
  - Isotropic and orthotropic behaviours in small and finite strain analyses.
  - Plane-stress, 3D analyses, etc.. Note that the number of components of the stress tensor is *not* enough (orthotropic behaviours)

The Abaqus/Standard interface

- **No work on the solver side**
- The Abaqus/Standard interface supports :
  - Isotropic and orthotropic behaviours in small and finite strain analyses.
  - Plane-stress, 3D analyses, etc.. Note that the number of components of the stress tensor is *not* enough (orthotropic behaviours)
- Introducing UMAT subroutines in Abaqus/Standard requires to recompile the code :
  - This was not viable option for MFront generated behaviours.
  - We choose to provide a generic UMAT subroutine which handles loading MFront behaviours in shared libraries. This can have a significant cost.

The Abaqus/Standard interface

- **No work on the solver side**
- The Abaqus/Standard interface supports :
  - Isotropic and orthotropic behaviours in small and finite strain analyses.
  - Plane-stress, 3D analyses, etc.. Note that the number of components of the stress tensor is *not* enough (orthotropic behaviours)
- Introducing UMAT subroutines in Abaqus/Standard requires to recompile the code :
  - This was not viable option for MFront generated behaviours.
  - We choose to provide a generic UMAT subroutine which handles loading MFront behaviours in shared libraries. This can have a significant cost.
- The user is responsible to ensure the consistency of the behaviour declaration of the behaviour in its input files !
  - This is error-prone
  - MFront generates an example of input files to reduce the risk of mistakes

The Code_Aster interface

- Code_Aster borrowed the signature of Abaqus/Standard' UMAT functions but already used shared libraries, so we could easily make the first tests.

The Code_Aster interface

- Code_Aster borrowed the signature of Abaqus/Standard' UMAT functions but already used shared libraries, so we could easily make the first tests.
- A specific optimised interface was then set-up :
  - **Significant** work on the solver side
  - Code_Aster is able to retrieve metadata from the generated library :
    - Number and names of the material properties
    - Number and names of internal state variables
    - Number and names of external state variables
    - Finite strain strategy for strain-based behaviour (example : is GDEF_LOG required)
    - etc..

Performances

- C++ is a highly technical language, but :
  - available on all plateform of interest ;
  - almost bare metal performances ;
  - interoperable with almost all languages thanks to C-compatibility ;

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 4.0.

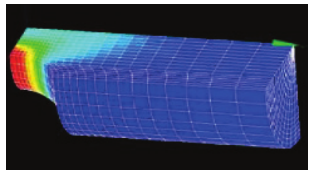
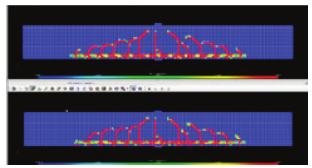
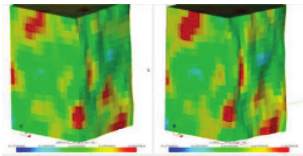
Some optimisation techniques in TFEL/Math

- 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

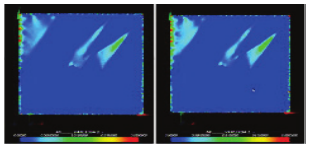
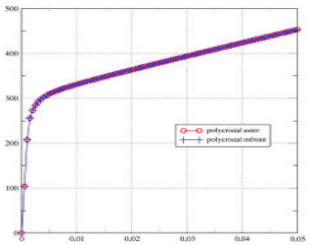
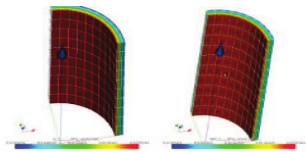
- Expression templates.
- Loop unrolling.
- Concepts and partial specialisations.
- Views.
- Compile-time dimensional analysis.
- **Those techniques far exceed the programming skills of most C++ professional programmers.**
  - We have anticipated the C++ language evolutions.
  - Some parts of TFEL are 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.**

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*	

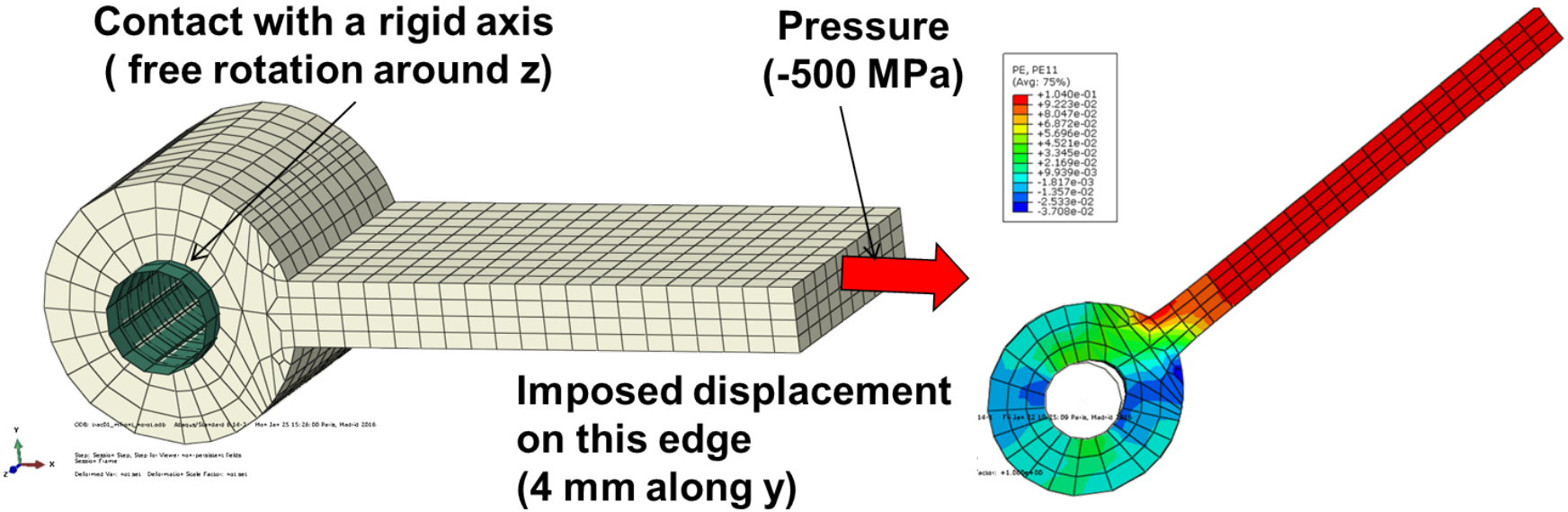
- Benchmarks made in 2013 by the Code_Aster developpers when evaluating MFront.
- Calling external behaviours in Code_Aster has improved...

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!!!

Documentation

User feedback

I was looking for material behaviour modleing through UMAT and I came across Mfront. It is really appricated efforts from your side to develop versatile material behaviour modeling which can be used with various FEA solvers.

I generally have used UMAT for my work. So I used to convert behaviour into numeroc algorithm.

As a beginner, Mfront looks difficlult to me to use.

I searched for documentation but it is in French. I tried to convert it but it was not good enough.

It will be very helpful if documentation in English and with starting from the basics.

Available documentation

- A page referencing the available documentation is available : `http://tfel.sourceforge.net/documentations.html`
  - The core documentation is in French and ageing !
  - It contains lot of material which can be overwhelming for a new user.
  - The tutorial (in French) is much more interesting.
  - *Longum iter est per praecepta, breve et efficax per exempla* (It's a long way by the rules, but short and efficient with examples)

Available documentation

- A page referencing the available documentation is available : `http://tfel.sourceforge.net/documentations.html`
  - The core documentation is in French and ageing !
  - It contains lot of material which can be overwhelming for a new user.
  - The tutorial (in French) is much more interesting.
  - *Longum iter est per praecepta, breve et efficax per exempla* (It's a long way by the rules, but short and efficient with examples)
- A getting started page has been set-up.

Available documentation

- A page referencing the available documentation is available : `http://tfel.sourceforge.net/documentations.html`
  - The core documentation is in French and ageing !
  - It contains lot of material which can be overwhelming for a new user.
  - The tutorial (in French) is much more interesting.
  - *Longum iter est per praecepta, breve et efficax per exemple* (It's a long way by the rules, but short and efficient with examples)
- A getting started page has been set-up.
- The `official MFront' paper` provides a good general overview of `MFront`.

Available documentation

- A page referencing the available documentation is available : `http://tfel.sourceforge.net/documentations.html`
  - The core documentation is in French and ageing !
  - It contains lot of material which can be overwhelming for a new user.
  - The tutorial (in French) is much more interesting.
  - *Longum iter est per praecepta, breve et efficax per exemple* (It's a long way by the rules, but short and efficient with examples)
- A getting started page has been set-up.
- The `official MFront' paper` provides a good general overview of `MFront`.
- The `gallery` gathers a set of fully described examples of well written behaviours :
  - In most cases, all the steps are described in depth.

Available documentation

- A page referencing the available documentation is available : `http://tfel.sourceforge.net/documentations.html`
  - The core documentation is in French and ageing !
  - It contains lot of material which can be overwhelming for a new user.
  - The tutorial (in French) is much more interesting.
  - *Longum iter est per praecepta, breve et efficax per exemple* (It's a long way by the rules, but short and efficient with examples)
- A getting started page has been set-up.
- The `official MFront' paper` provides a good general overview of `MFront`.
- The `gallery` gathers a set of fully described examples of well written behaviours :
  - In most cases, all the steps are described in depth.
- `https://github.com/thelfer/tfel-doc`

Available documentation

- A page referencing the available documentation is available : `http://tfel.sourceforge.net/documentations.html`
  - The core documentation is in French and ageing !
  - It contains lot of material which can be overwhelming for a new user.
  - The tutorial (in French) is much more interesting.
  - *Longum iter est per praecepta, breve et efficax per exemple* (It's a long way by the rules, but short and efficient with examples)
- A getting started page has been set-up.
- The `official MFront' paper` provides a good general overview of `MFront`.
- The `gallery` gathers a set of fully described examples of well written behaviours :
  - In most cases, all the steps are described in depth.
- `https://github.com/thelfer/tfel-doc`
- For developers only, one can refer to the http ://tfel.sourceforge.net/doxygen/index.html.

TFEL 3.1.x

Releases

- TFEL 3.1.1
- TFEL 3.1.2
- TFEL 3.1.3

Releases

- TFEL 3.1.1
- TFEL 3.1.2
- TFEL 3.1.3
- Major tickets solved :
  - Ticket 16 : Change error management in `Aster` interface
  - Ticket 99 : Functions defined in `Barlat.ixx` and `Hosford.ixx` does not work for floats
  - Ticket 127 : Substepping in the `Cast3M` and `Cyrano` interface may lead to a invalid convergence du to integer overflow
  - Ticket 126 : Jacobian error in `DDIF2Base.ixx`
  - Ticket 131 : Plane stress support in
  - Abaqus/Explicit is broken
  - Ticket 135 : Declaration error of `'material_properties_nb'` in `CyranoBehaviourHandler` with the `cyrano` interface

TFEL 3.2

Inline material properties

- Various keywords (such as `@ElasticMaterialProperties`, `@ComputeThermalExpansion`, `@HillTensor`, etc.) expects one or more material properties.

Inline material properties

- Various keywords (such as `@ElasticMaterialProperties`, `@ComputeThermalExpansion`, `@HillTensor`, etc.) expects one or more material properties.
- In previous versions, those material properties were constants or defined by an external ‘MFront‘ file.

Inline material properties

- Various keywords (such as `@ElasticMaterialProperties`, `@ComputeThermalExpansion`, `@HillTensor`, etc.) expects one or more material properties.
- In previous versions, those material properties were constants or defined by an external ‘MFront‘ file.
- This new version allows those material properties to be defined by formulae, as follows :

```
@Parameter E0 =2.1421e11,E1 = −3.8654e7,E2 = −3.1636e4;
@ElasticMaterialProperties {"E0+(T−273.15)∗(E1+E2∗(T−273.15))",0.3}
```

Inline material properties

- Various keywords (such as `@ElasticMaterialProperties`, `@ComputeThermalExpansion`, `@HillTensor`, etc.) expects one or more material properties.
- In previous versions, those material properties were constants or defined by an external ‘MFront‘ file.
- This new version allows those material properties to be defined by formulae, as follows :

```
@Parameter E0 =2.1421e11,E1 = −3.8654e7,E2 = −3.1636e4;
@ElasticMaterialProperties {"E0+(T−273.15)∗(E1+E2∗(T−273.15))",0.3}
```

- As for material properties defined in external ‘MFront‘ files, the material properties evaluated by formulae will be computed for updated values of their parameters.

Inline material properties

- Various keywords (such as `@ElasticMaterialProperties`, `@ComputeThermalExpansion`, `@HillTensor`, etc.) expects one or more material properties.
- In previous versions, those material properties were constants or defined by an external ‘MFront‘ file.
- This new version allows those material properties to be defined by formulae, as follows :

```
@Parameter E0 =2.1421e11,E1 = −3.8654e7,E2 = −3.1636e4;
@ElasticMaterialProperties {"E0+(T−273.15)∗(E1+E2∗(T−273.15))",0.3}
```

- As for material properties defined in external ‘MFront‘ files, the material properties evaluated by formulae will be computed for updated values of their parameters.
- For example, if the previous lines were used in the `Implicit DSL`, two variables `young` and `young_tdt` will be automatically made available.

The `StandardElastoViscoplasticity` brick : overview

- The `StandardElastoViscoplasticity` brick describes a strain based behaviour using an additive decomposition of the total strain in an elastic part and one or several inelastic strains describing plastic (time-independent) flows and/or viscoplastic (time-dependent) flows, as follows :

$$\underline{\epsilon}^{to} = \underline{\epsilon}^{el} + \sum_{i_p=0}^{n_p} \underline{\epsilon}^{p}_{i_p} + \sum_{i_{vp}=0}^{n_{vp}} \underline{\epsilon}^{vis}_{i_{vp}}$$

The `StandardElastoViscoplasticity` brick : overview

- The `StandardElastoViscoplasticity` brick describes a strain based behaviour using an additive decomposition of the total strain in an elastic part and one or several inelastic strains describing plastic (time-independent) flows and/or viscoplastic (time-dependent) flows, as follows :

$$\underline{\epsilon}^{to} = \underline{\epsilon}^{el} + \sum_{i_{\mathrm{p}}=0}^{n_{\mathrm{p}}} \underline{\epsilon}^{p}_{i_{\mathrm{p}}} + \sum_{i_{\mathrm{vp}}=0}^{n_{\mathrm{vp}}} \underline{\epsilon}^{vis}_{i_{\mathrm{vp}}}$$

- This brick decomposes the behaviour into two components :
  - The **stress potential** which defines the relation between the elastic strain $\underline{\epsilon}^{el}$ and possibly some damage variables and the stress measure $\underline{\sigma}$. As the definition of the elastic properties can be part of the definition of the stress potential, the thermal expansion coefficients can also be defined in the block corresponding to the stress potential.
  - A list of **inelastic flows**.
- This work greatly benefited from the work of J. Tirari

The `StandardElastoViscoplasticity` brick : inelastic flows

- The evolution of the inelastic strain is given by :

$$\begin{cases} \underline{\epsilon}^{p}_{i_{\text{p}}} = \dot{\lambda}_{i_{\text{p}}} \dfrac{\partial g_{i_{\text{p}}}}{\partial \underline{\sigma}} \text{ with } \begin{cases} f_{i_{\text{p}}}\left(\underline{\sigma}\right) = \left(\underline{\sigma} - \displaystyle\sum_{j=1}^{N_k} \underline{\mathbf{X}}_{i_{\text{p}}j}\right)_{\text{eq } i} - \displaystyle\sum_{j=1}^{N_r} \underline{\mathbf{R}}_{i_{\text{p}}j}\left(p_{i_{\text{p}}}\right) \leq 0 \\ \dot{\lambda}_{i_{\text{p}}} \geq 0 \\ \dot{\lambda}_{i_{\text{p}}} f_{i_{\text{p}}}\left(\underline{\sigma}\right) = 0 \end{cases} \\ \underline{\epsilon}^{vis}_{i_{\text{vp}}} = f_{i_{\text{vp}}}\left(\left\langle \left(\underline{\sigma} - \displaystyle\sum_{j=1}^{N_k} \underline{\mathbf{X}}_{i_{\text{vp}}j}\right)_{\text{eq } i} - \displaystyle\sum_{j=1}^{N_r} \underline{\mathbf{R}}_{i_{\text{vp}}j}\left(p_{i_{\text{vp}}}\right) \right\rangle\right) \dfrac{\partial g_{i_{\text{vp}}}}{\partial \underline{\sigma}} \end{cases}$$

- $\underline{\mathbf{g}}_{i_{\text{vp}}}$: plastic potential for non associated flows
- $\underline{\mathbf{R}}_{i_{\text{vp}}j}$: isotropic hardening rules
- $\underline{\mathbf{X}}_{i_{\text{vp}}j}$: kinematic hardening rules

The `StandardElastoViscoplasticity` brick : inelastic flows

- The evolution of the inelastic strain is given by :

$$\begin{cases} \underline{\epsilon}^{p}_{i_p} = \dot{\lambda}_{i_p} \dfrac{\partial g_{i_p}}{\partial \underline{\sigma}} \text{ with } \begin{cases} f_{i_p}\left(\underline{\sigma}\right) = \left(\underline{\sigma} - \displaystyle\sum_{j=1}^{N_k} \underline{\mathbf{X}}_{i_p j}\right)_{\text{eq } i} - \displaystyle\sum_{j=1}^{N_r} \underline{\mathbf{R}}_{i_p j}\left(p_{i_p}\right) \leq 0 \\ \dot{\lambda}_{i_p} \geq 0 \\ \dot{\lambda}_{i_p} f_{i_p}\left(\underline{\sigma}\right) = 0 \end{cases} \\ \underline{\epsilon}^{vis}_{i_{vp}} = f_{i_{vp}}\left(\left\langle \left(\underline{\sigma} - \displaystyle\sum_{j=1}^{N_k} \underline{\mathbf{X}}_{i_{vp} j}\right)_{\text{eq } i} - \displaystyle\sum_{j=1}^{N_r} \underline{\mathbf{R}}_{i_{vp} j}\left(p_{i_{vp}}\right) \right\rangle\right) \dfrac{\partial g_{i_{vp}}}{\partial \underline{\sigma}} \end{cases}$$

- $\underline{\mathbf{g}}_{i_{vp}}$: plastic potential for non associated flows
- $\underline{\mathbf{R}}_{i_{vp} j}$: isotropic hardening rules
- $\underline{\mathbf{X}}_{i_{vp} j}$: kinematic hardening rules
- `mfront -list-inelastic-flows`
  - `HyperbolicSine`
  - `Norton`
  - `Plastic`

The `StandardElastoViscoplasticity` brick : a simple plastic behaviour based on the Hosford criterion

```
@DSL Implicit;
@Behaviour Hosford;

@ModellingHypotheses {".+"};
@Epsilon 1.e-16;
@IterMax 100;
@Theta 1;

@Brick "StandardElastoViscoPlasticity" {
  stress_potential : "Hooke" {
    young_modulus : 160e9,
    poisson_ratio : 0.3
  },
  inelastic_flow : "Plastic" {
    criterion : "Hosford" {a : 6},
    isotropic_hardening : "Linear" {R0 : 120e6}
  }
};
```

- The Hosford equivalent stress is defined by :

$$\sigma_{eq}^{H} = \sqrt[a]{\frac{1}{2}\left(|\sigma_1 - \sigma_2|^a + |\sigma_1 - \sigma_3|^a + |\sigma_2 - \sigma_3|^a\right)}$$

- The `StandardElastoViscoplasticity` brick provides a way to implement complex elasto-viscoplastic behaviours in a

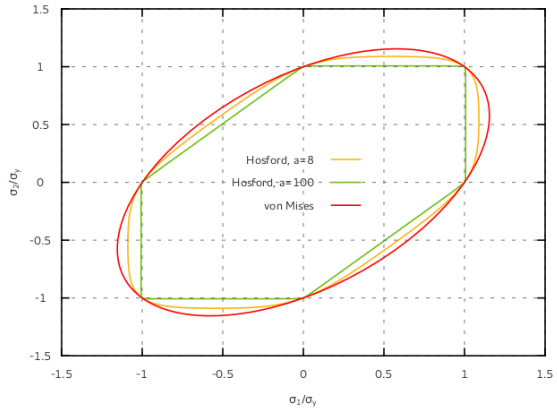
The `StandardElastoViscoplasticity` brick : a complex example

```
@Brick "StandardElastoViscoPlasticity" {
  // Here the stress potential is given by the Hooke law. We define:
  // - the elastic properties (Young modulus and Poisson ratio).
  //   Here the Young modulus is a function of the temperature.
  //   The Poisson ratio is constant.
  // - the thermal expansion coefficient
  // - the reference temperature for the thermal expansion
  stress_potential : "Hooke" {
    young_modulus : "2.e5 - (1.e5*((T - 100.)/960.)**2)",
    poisson_ratio : 0.3,
    thermal_expansion : "1.e-5 + (1.e-5 * ((T - 100.)/960.) ** 4)",
    thermal_expansion_reference_temperature : 0
  },
  // Here we define only one viscplastic flow defined by the Norton law,
  // which is based:
  // - the von Mises stress criterion
  // - one isotorpic hardening rule based on Voce formalism
  // - one kinematic hardening rule following the Armstrong-Frederick law
  inelastic_flow : "Norton" {
    criterion : "Mises",
    isotropic_hardening : "Voce" {R0 : 200, Rinf : 100, b : 20},
    kinematic_hardening : "Armstrong-Frederick" {
      C : "1.e6 - 98500 * (T - 100) / 96",
      D : "5000 - 5* (T - 100)"
    },
    K : "(4200. * (T + 20.) - 3. * (T + 20.0)**2)/4900.",
    n : "7. - (T - 100.) / 160.",
    Ksf : 3
  }
};
```

The `StandardElastoViscoplasticity` brick : stress potentials

- `mfront -list-stress-potentials`
  - `DDIF2`
  - `Hooke`
  - `IsotropicDamage`
- Note that the `StandardElasticity` and the `DDIF2` bricks are now mostly wrappers around the associated stress potentials.

The `StandardElastoViscoplasticity` brick : stress criteria



- `mfront -list-stress-criteria`
  - Isotropic criteria :
    - `Mises`
    - `Cazacu 2001`
    - `Drucker 1949, (Drucker1949)`
    - `Hosford (Hosford 1972, Hosford1972)`
    - `Isotropic Cazacu 2004 (IsotropicCazacu2004)`
  - Orthotropic criteria :
    - `Barlat (Barlat 2004, Barlat2004)`
    - `Hill 1948, (Hill1948)`
    - `Orthotropic Cazacu 2004 (OrthotropicCazacu2004)`

The `StandardElastoViscoplasticity` brick : hardening rules

- `mfront -list-isotropic-hardening-rules`
  - `Linear` : $R(p) = R_0 + H\,p$
  - `Swift` : $R(p) = R_0 \left(\frac{p + p_0}{p_0}\right)^n$
  - `Voce` : $R(p) = R_\infty + (R_0 - R_\infty)\ exp(-b\,p)$

The `StandardElastoViscoplasticity` brick : hardening rules

- `mfront -list-isotropic-hardening-rules`
  - `Linear` : $R(p) = R_0 + H\,p$
  - `Swift` : $R(p) = R_0 \left(\frac{p + p_0}{p_0}\right)^n$
  - `Voce` : $R(p) = R_\infty + (R_0 - R_\infty)\, exp(-b\,p)$
- `mfront -list-kinematic-hardening-rules`
  - `Prager`

$$\underline{\mathbf{X}} = \frac{2}{3} C\,\underline{\mathbf{a}} \quad \text{and} \quad \underline{\mathbf{a}} = \dot{p}\,\underline{\mathbf{n}}$$

  - `Armstrong-Frederick` (`ArmstrongFrederick`) :

$$\underline{\mathbf{X}} = \frac{2}{3} C\,\underline{\mathbf{a}} \quad \text{and} \quad \underline{\mathbf{a}} = \dot{p}\,\underline{\mathbf{n}} - D\,\dot{p}\,\underline{\mathbf{a}}$$

  - `Burlet-Cailletaud` (`Burlet-Cailletaud 1987`, `BurletCailletaud1987`)
  - `Chaboche 2012` (`Chaboche2012`)

The `generic` interface

- The `generic` interface has been created after several requests for using `MFront` behaviours in small in-house code ;
- This interface is made very general :
  - small and finite strain behaviours.
  - cohesive zone models.
  - general behaviours (currently under work).
  - isotropic and orthotropic behaviours.
- Strongly coupled with the `MGIS` project described later.

Logarithimc strain support in the `Cyrano` interface

- 3.2 As Cyrano provides a mono-dimensional description of the fuel rod, its extension to finite strain follows the same treatment used for :
  - the extension of MTest to pipes.
  - the extension of Alcyone1D to finite strain analysis.

Case of studies

The Gatt-Monerie's behaviours

- Complex viscoplastic behaviour :
  - Two coupled dissipation potentials.
  - Describe the porosity evolution.
  - Takes into account the irradiation, grain size and temperature effects.
- Same constitutive equations, various identifications.
- Identified coupled with the `StandardElasticity` brick, used coupled with the `DDIF2` brick in fuel performance code.
- Great example of the usage of the `@Import` keyword

How to enhance the numerical robustness of the implicit scheme : Scherzinger' test

- Scherzinger, W. M. 2017. *"A Return Mapping Algorithm for Isotropic and Anisotropic Plasticity Models Using a Line Search Method."* Computer Methods in Applied Mechanics and Engineering 317 (April) : 526–53. `https://doi.org/10.1016/j.cma.2016.11.026`.
- From the null stress state, impose loading increment up to 30 times the yield stress in each direction of the π-plane
- Applied to a perfect plastic behaviour following the `Hosford` criterion :

$$\sigma_{eq}^{H} = \sqrt[a]{\frac{1}{2}\left(|\sigma_1 - \sigma_2|^a + |\sigma_1 - \sigma_3|^a + |\sigma_2 - \sigma_3|^a\right)}$$

- The behaviour becomes harder to integrate as a grows.
- Pratical exponents are 6 and 8.
- For an exponent of 100, the yield surface is very close to TRESCA'

How to enhance the numerical robustness of the implicit scheme : Scherzinger' test implementation

```
from math import pi,cos,sin
import tfel.math as tmath
import tfel.material as tmaterial
import mtest

nmax = 1000
nmax2 = 1000
E = 150e9
nu = 0.3
s0 = 150e6

for a in [pi*(-1.+(2.*i)/(nmax-1)) for i in range(0,nmax)]:
    for x in [1+(29.*i)/(nmax2-1) for i in range(0,nmax2)]:
        nbiter=0
        s = tmath.makeStensor3D(tmaterial.buildFromPiPlane(cos(a),sin(a)))
        seq = tmaterial.computeHosfordStress(s,100,1.e-12);
        s *= x*(s0/seq)
        e0 = (s[0]-nu*(s[1]+s[2]))/E
        e1 = (s[1]-nu*(s[0]+s[2]))/E
        e2 = (s[2]-nu*(s[0]+s[1]))/E
        m = mtest.MTest()
        mtest.setVerboseMode(mtest.VerboseLevel.VERBOSE_QUIET)
        m.setModellingHypothesis('Tridimensional')
        m.setMaximumNumberOfSubSteps(1)
        m.setBehaviour('abaqus','src/libABAQUSBEHAVIOUR.so',
                       'HOSFORDPERFECTPLASTICITY100_3D');
        m.setExternalStateVariable("Temperature",293.15)
        m.setImposedStrain('EXX',{0:0,1:e0})
        m.setImposedStrain('EYY',{0:0,1:e1})
        m.setImposedStrain('EZZ',{0:0,1:e2})
        m.setImposedStrain('EXY',0)
        m.setImposedStrain('EXZ',0)
        m.setImposedStrain('EYZ',0)
        s = mtest.MTestCurrentState()
        wk = mtest.MTestWorkSpace()
        m.completeInitialisation()
        m.initializeCurrentState(s)
        m.initializeWorkSpace(wk)
        try:
            m.execute(s,wk,0,1)
            nbiter=s.getInternalStateVariableValue("NumberOfIterations")
        except:
            nbiter=100
            break
        print(str((x/seq)*cos(a))+" "+str((x/seq)*sin(a))+" "+str(nbiter))
    print()
```

How to enhance the numerical robustness of the implicit scheme : tested algorithms

- **Algorithm 1** : The standard Newton-Raphson algorithm.

How to enhance the numerical robustness of the implicit scheme : tested algorithms

- **Algorithm 1** : The standard Newton-Raphson algorithm.
- **Algorithm 2** : The Newton-Raphson algorithm with a test on the value of the equivalent Hosford stress $\sigma_{eq}^{H}\big|_{t+\theta\,\Delta\,t}$ for the current estimation of the elastic strain increment. If this $\sigma_{eq}^{H}\big|_{t+\theta\,\Delta\,t}$ is greater than $\beta\,\sigma_Y$, the Newton step is rejected : the direction is kept unchanged but the amplitude of the correction to the internal state variables increment is divided by two. In the following, β is choosen equal to $\frac{3}{2}$. The test has to be made in the `@Integrator` code block : if the test fails, one just have to return the `false` value.

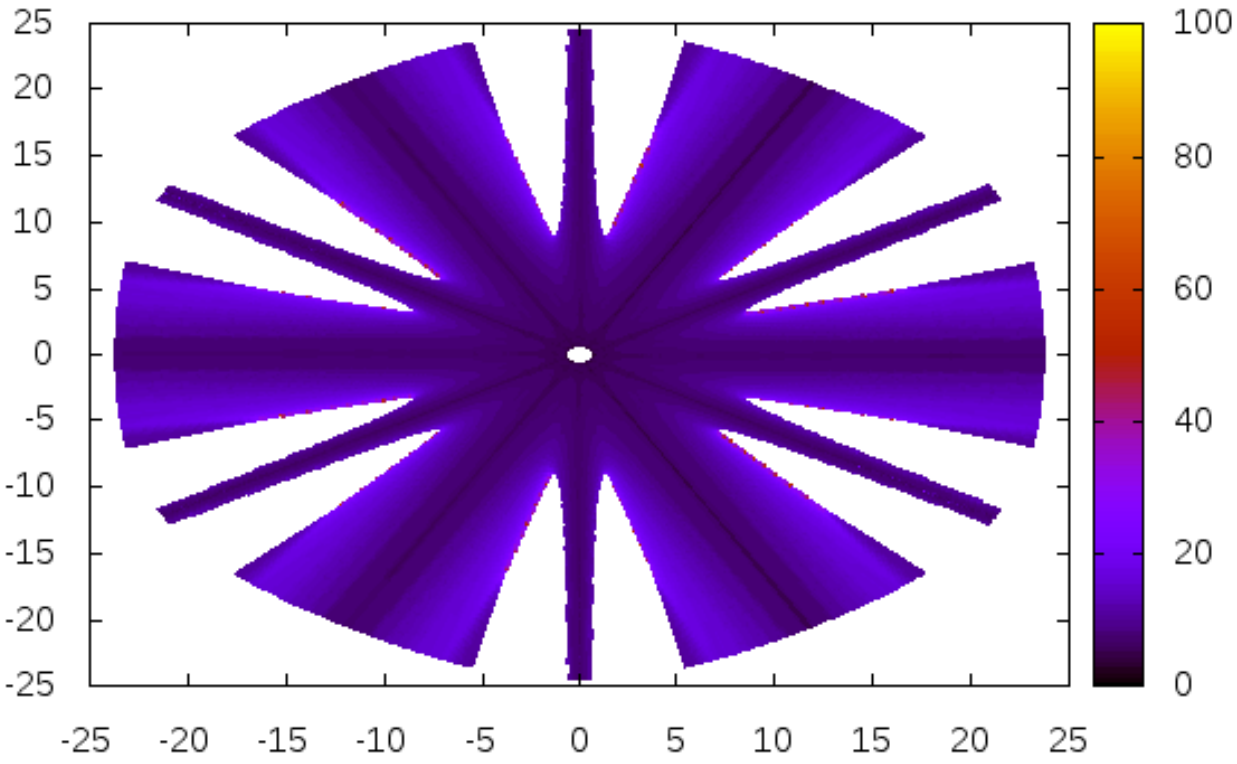
How to enhance the numerical robustness of the implicit scheme : tested algorithms

- **Algorithm 1** : The standard Newton-Raphson algorithm.
- **Algorithm 2** : The Newton-Raphson algorithm with a test on the value of the equivalent Hosford stress $\sigma_{eq}^{H}\big|_{t+\theta\,\Delta\,t}$ for the current estimation of the elastic strain increment. If this $\sigma_{eq}^{H}\big|_{t+\theta\,\Delta\,t}$ is greater than $\beta\,\sigma_Y$, the Newton step is rejected : the direction is kept unchanged but the amplitude of the correction to the internal state variables increment is divided by two. In the following, β is choosen equal to $\frac{3}{2}$. The test has to be made in the `@Integrator` code block : if the test fails, one just have to return the `false` value.
- **Algorithm 3** : The Newton-Raphson algorithm with a limitation of the increment of the elastic strain increment at each iteration. This can be activated by using the `setMaximumIncrementValuePerIteration` method on the `eel` variable.

How to enhance the numerical robustness of the implicit scheme : tested algorithms

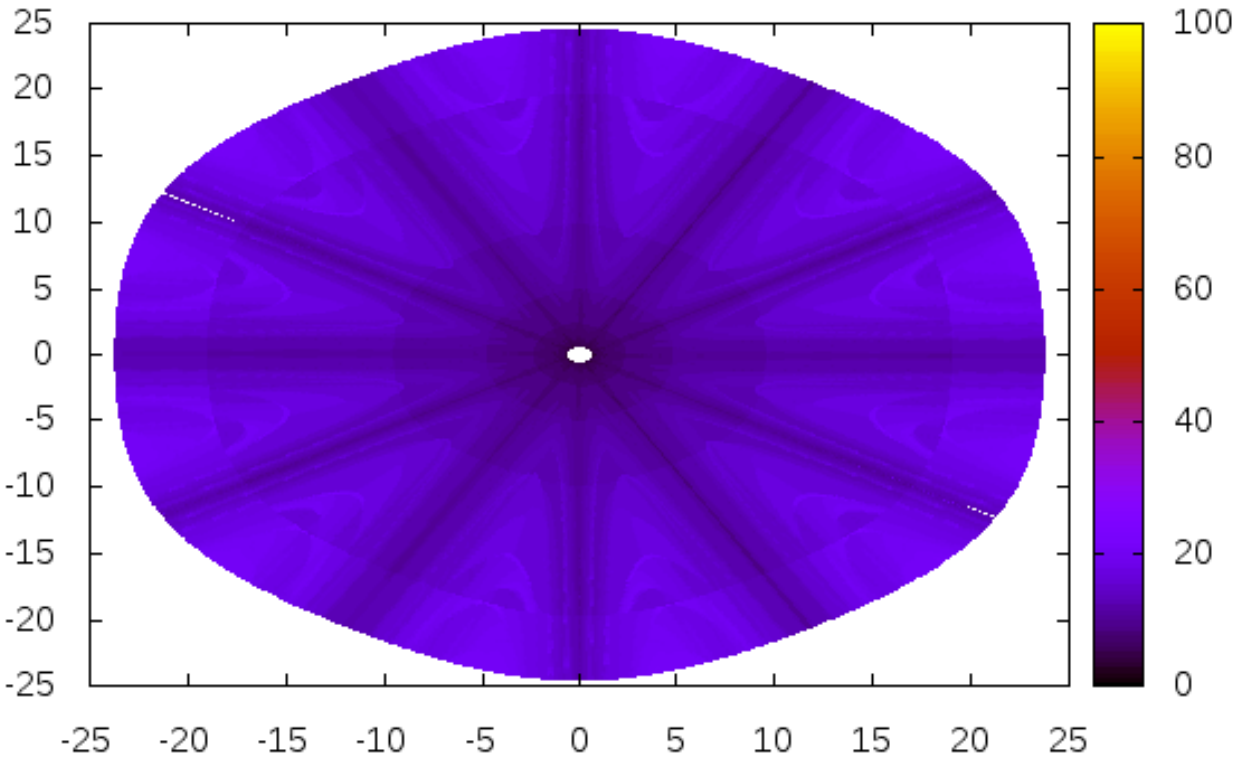
- **Algorithm 1** : The standard Newton-Raphson algorithm.
- **Algorithm 2** : The Newton-Raphson algorithm with a test on the value of the equivalent Hosford stress $\sigma_{eq}^{H}|_{t+\theta\,\Delta\,t}$ for the current estimation of the elastic strain increment. If this $\sigma_{eq}^{H}|_{t+\theta\,\Delta\,t}$ is greater than $\beta\,\sigma_Y$, the Newton step is rejected : the direction is kept unchanged but the amplitude of the correction to the internal state variables increment is divided by two. In the following, β is choosen equal to $\frac{3}{2}$. The test has to be made in the `@Integrator` code block : if the test fails, one just have to return the `false` value.
- **Algorithm 3** : The Newton-Raphson algorithm with a limitation of the increment of the elastic strain increment at each iteration. This can be activated by using the `setMaximumIncrementValuePerIteration` method on the `eel` variable.
- **Algorithm 4** : The standard Levenberg-Marquardt algorithm. This algorithm is selected using the `@Algorithm` keyword.

How to enhance the numerical robustness of the implicit scheme : results ($a = 6$)



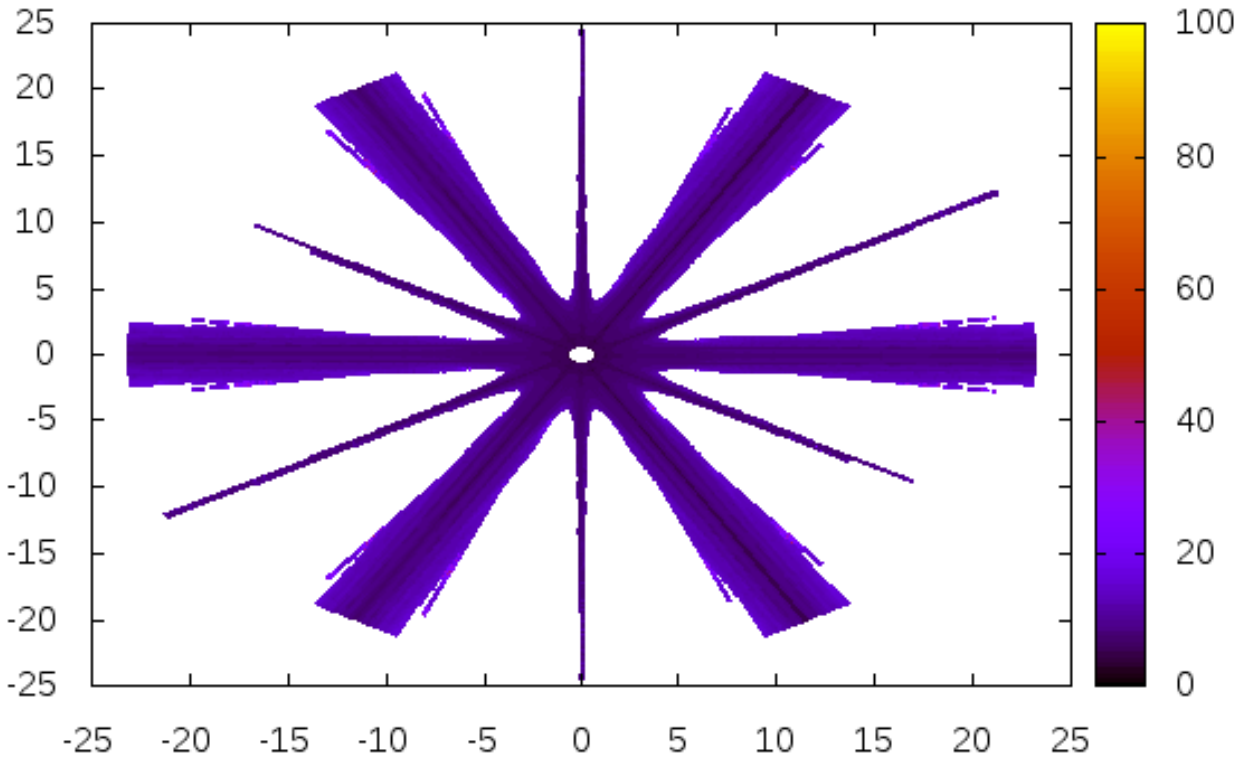
- Algorithme 1 : Results are consistent with Scherzinger' ones

How to enhance the numerical robustness of the implicit scheme : results ($a = 6$)



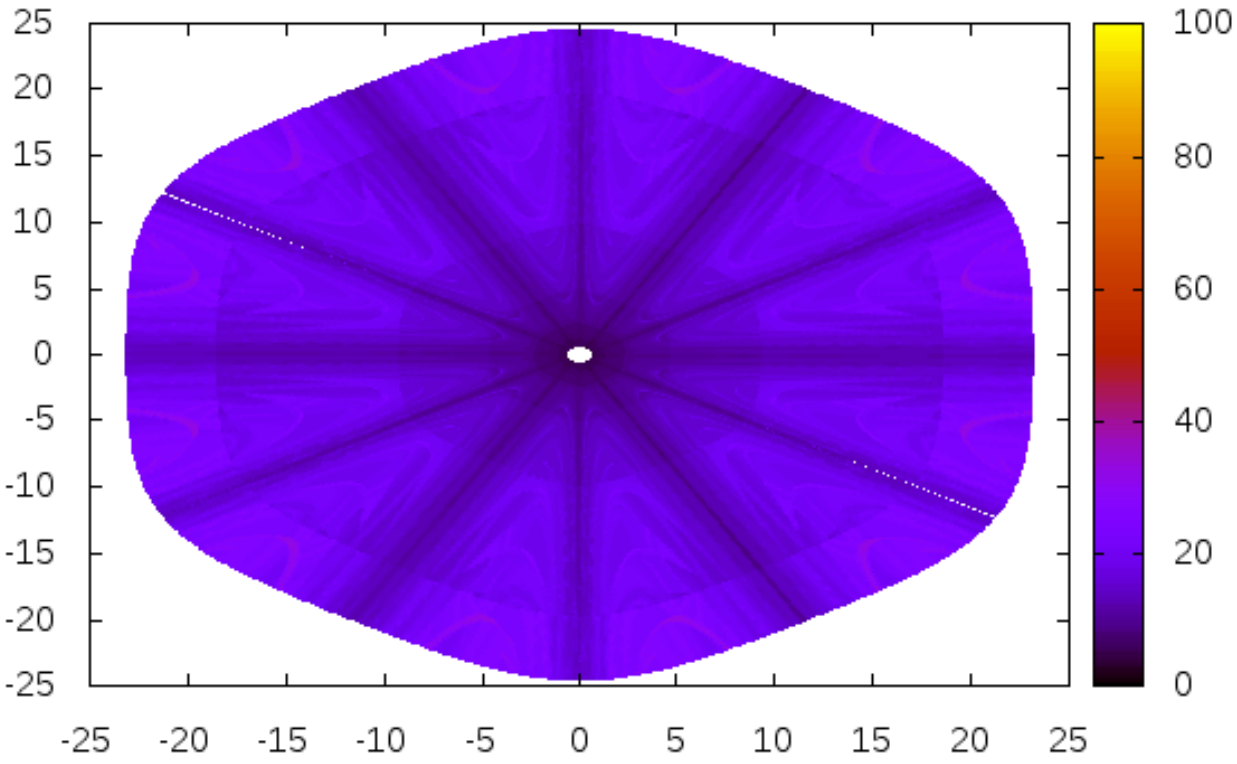
- Algorithme 2 : great improvement of the algorithm robustness for a very slight modification of the Newton algorithm

How to enhance the numerical robustness of the implicit scheme : results ($a = 8$)



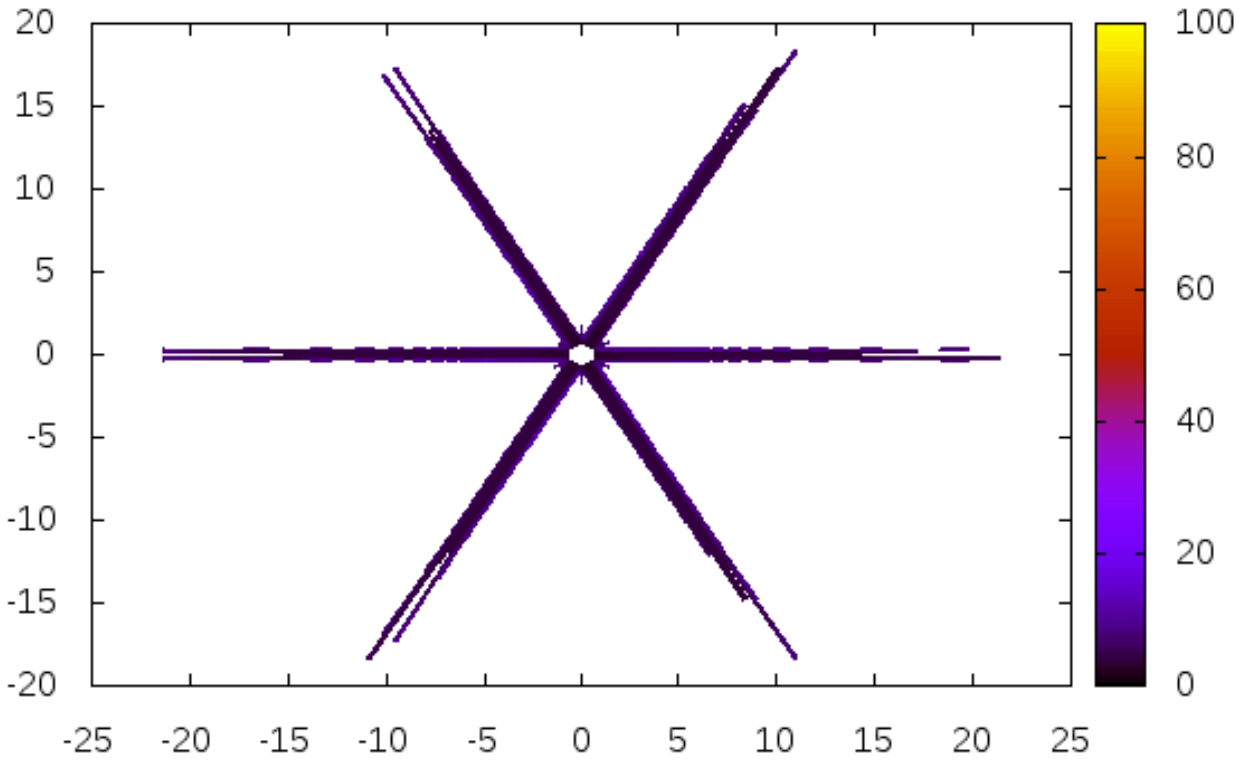
- Algorithme 1

How to enhance the numerical robustness of the implicit scheme : results ($a = 8$)



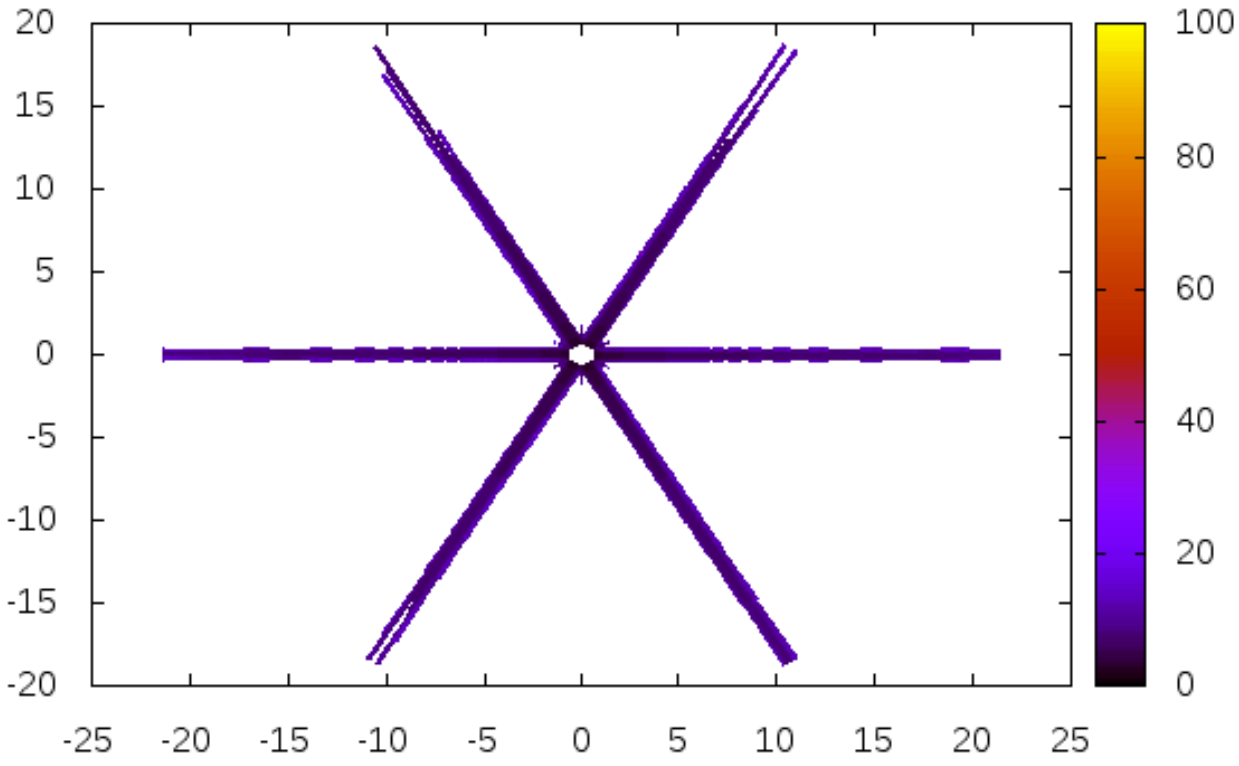
- Algorithme 2

How to enhance the numerical robustness of the implicit scheme : results ($a = 100$)



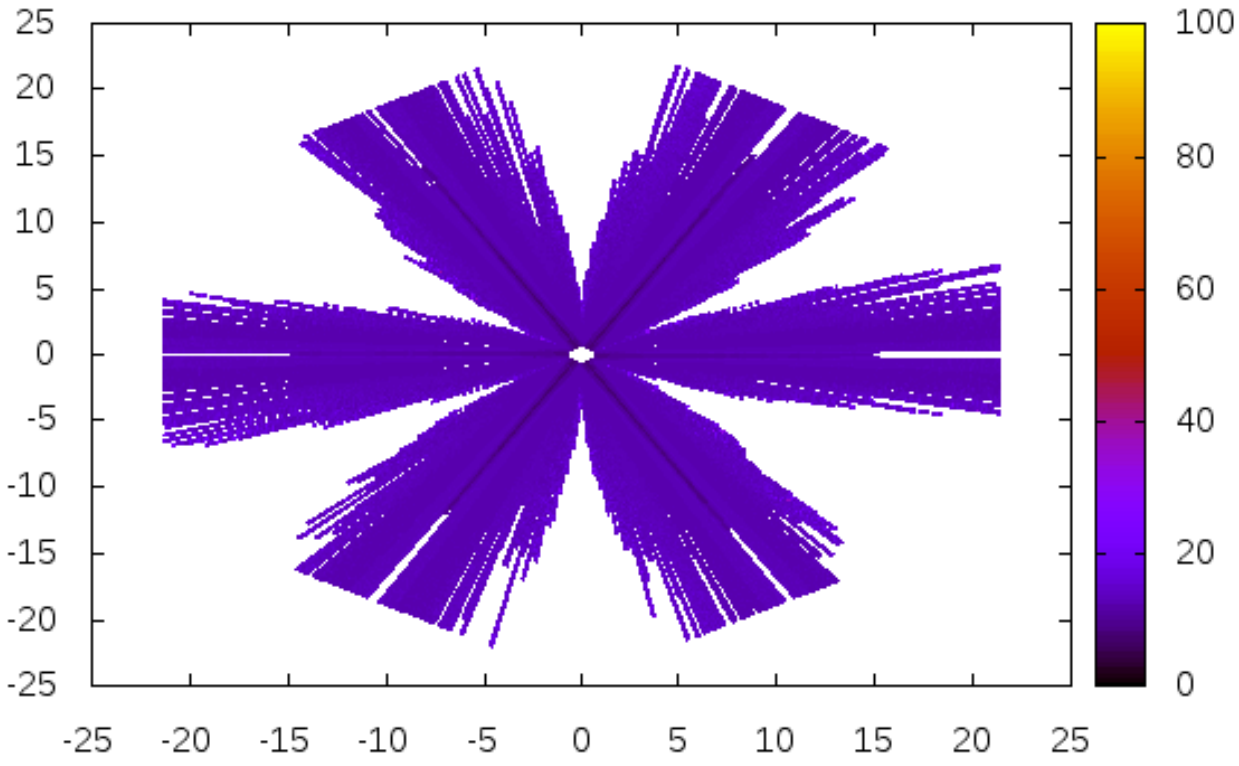
- Algorithme 1

How to enhance the numerical robustness of the implicit scheme : results ($a = 100$)



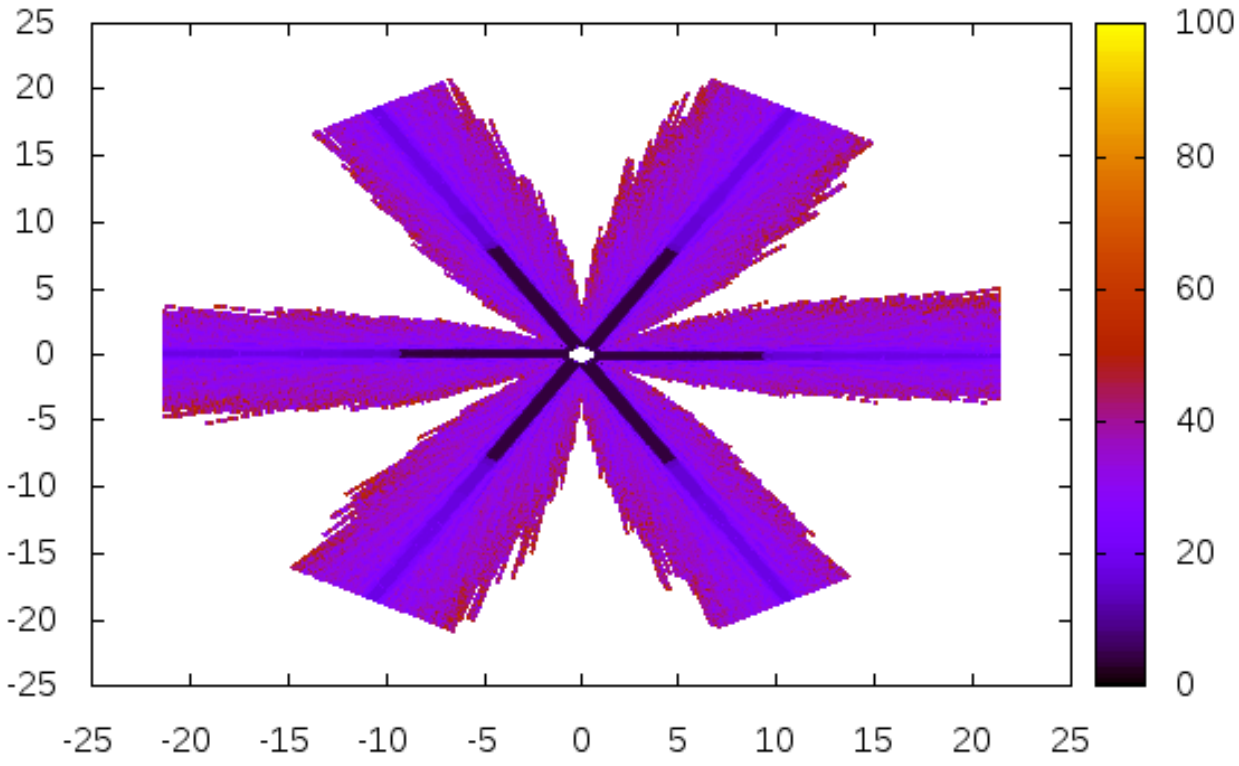
- Algorithme 2

How to enhance the numerical robustness of the implicit scheme : results ($a = 100$)



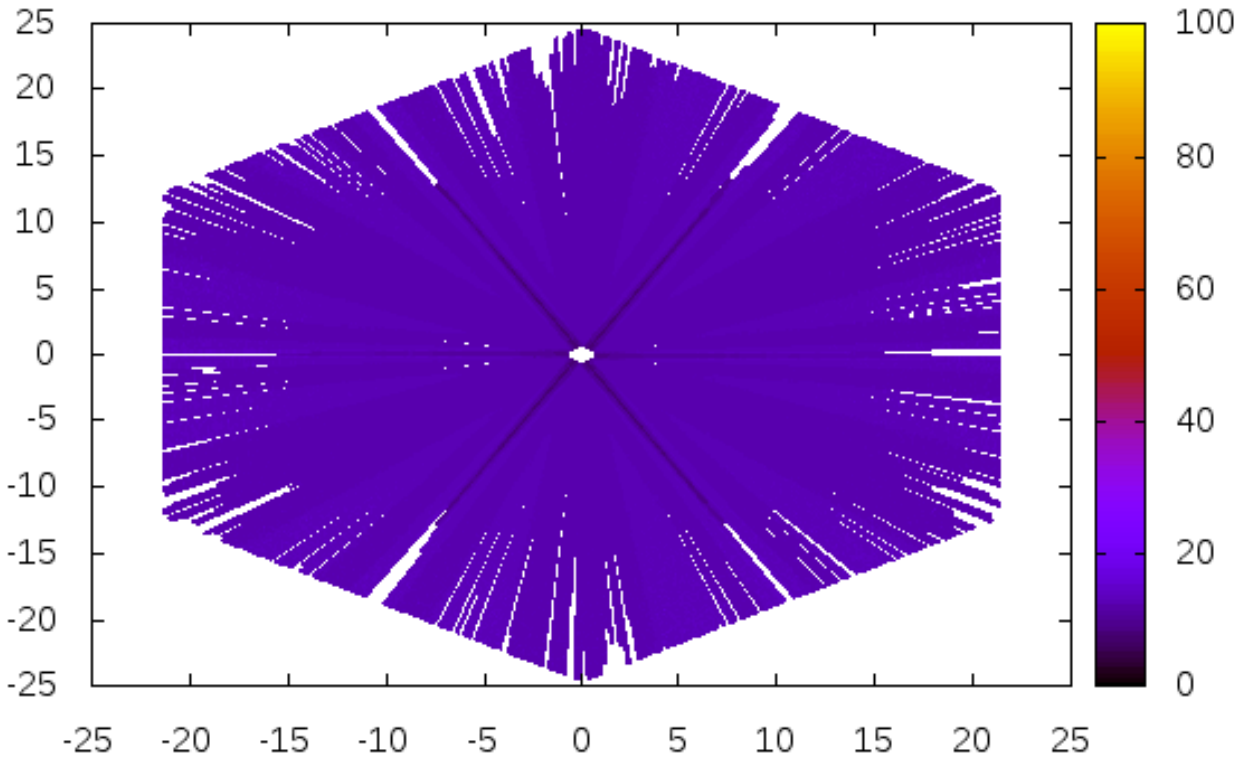
- Algorithme 3

How to enhance the numerical robustness of the implicit scheme : results ($a = 100$)



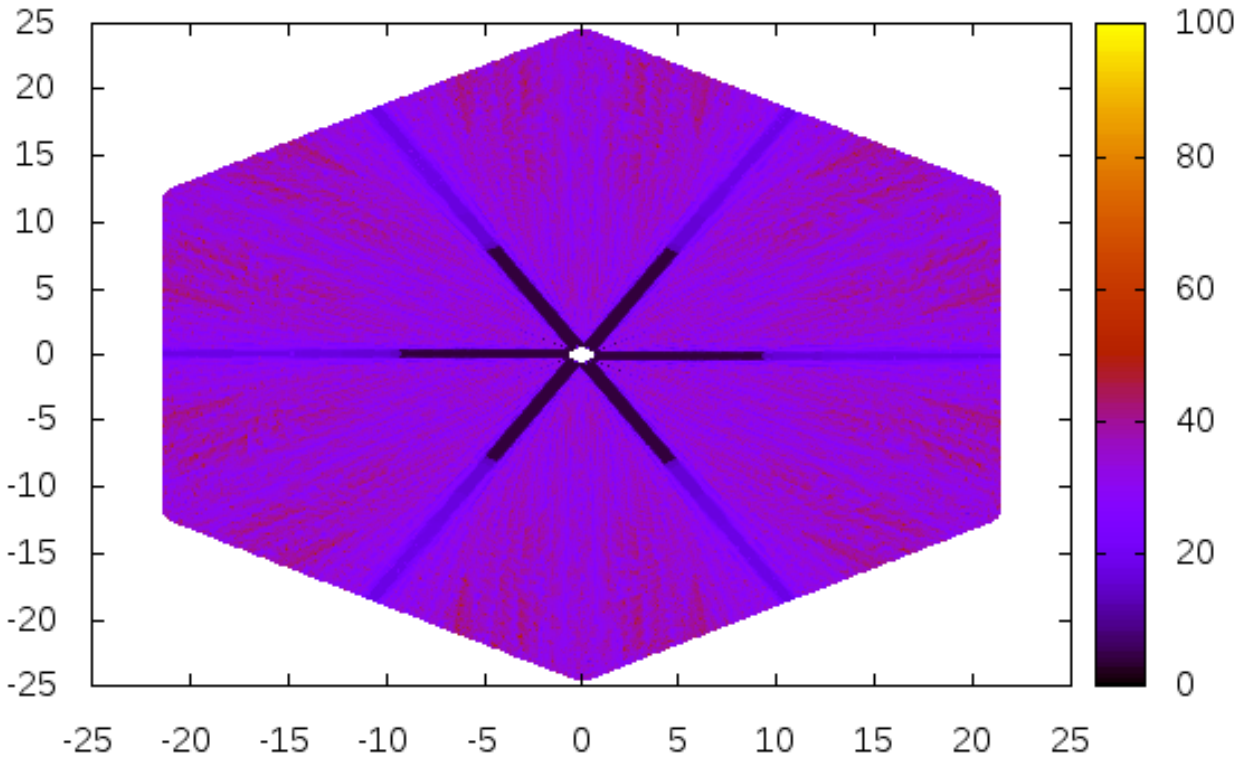
- Algorithme 4

How to enhance the numerical robustness of the implicit scheme : results ($a = 100$)



- Algorithme 3 and Jacobi eigen solver

How to enhance the numerical robustness of the implicit scheme : results ($a = 100$)



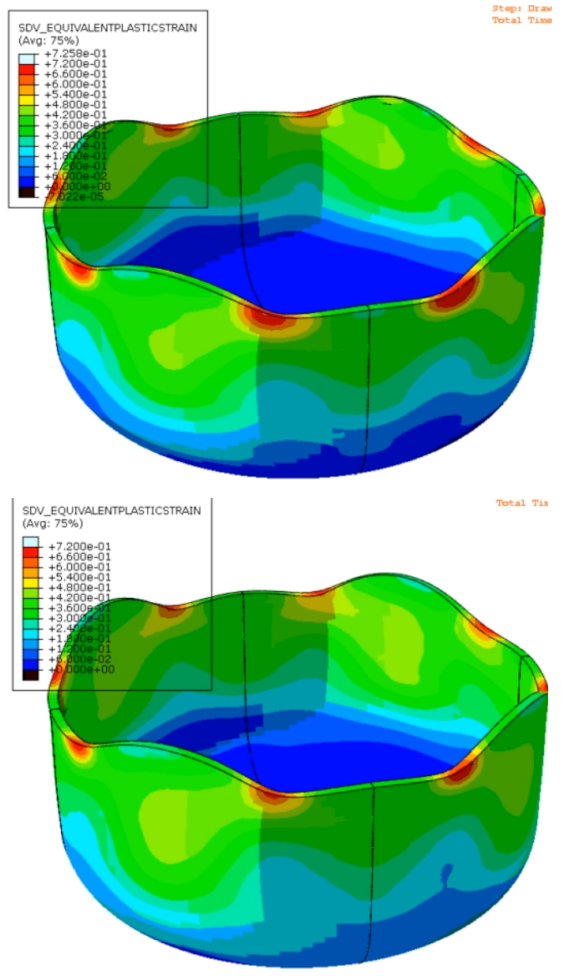
- Algorithme 4 and Jacobi eigen solver

Deep drawing simulation (Computations by D. Deloison from ArianeGroup)

```
@DSL       Implicit ;
@Behaviour Barlat2090T3;

@AbaqusFiniteStrainStrategy[abaqus,abaqusexplicit] Native;
@ModellingHypotheses {".+"};
@OrthotropicBehaviour<Plate>;
@Epsilon 1.e−16;
@Theta   1;

@Brick StandardElastoViscoPlasticity{
   stress_potential  :  "Hooke" {
     young_modulus : 150e9,
     poisson_ratio  :  0.3},
   inelastic_flow   :  "Plastic"  {
      criterion   :  "Barlat"  {
        a  :  8,
        l1   :  {−0.069888, 0.079143, 0.936408, 0.524741,
                 1.00306, 1.36318, 0.954322, 1.06906, 1.02377},
        l2   :  {0.981171, 0.575316, 0.476741, 1.14501,
                 0.866827, −0.079294, 1.40462, 1.1471, 1.05166}
      },
     isotropic_hardening :  "Linear"  {R0 : 150e6}
   }
};
```



- Prediction of six or eight ears in a drawn cup... Yoon et Al.
- **Same results in** `Abaqus/Standard` **and** `Abaqus/Explicit`.

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.
  - Identification procedures.
  - Behaviour implementations.
  - Unit testing.
  - Documentation.

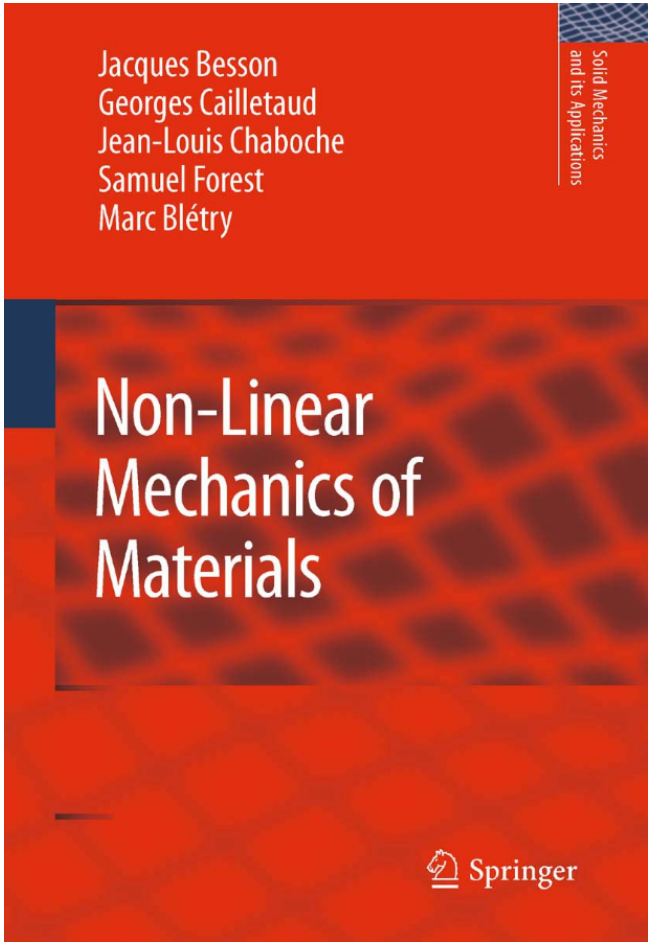
Future works

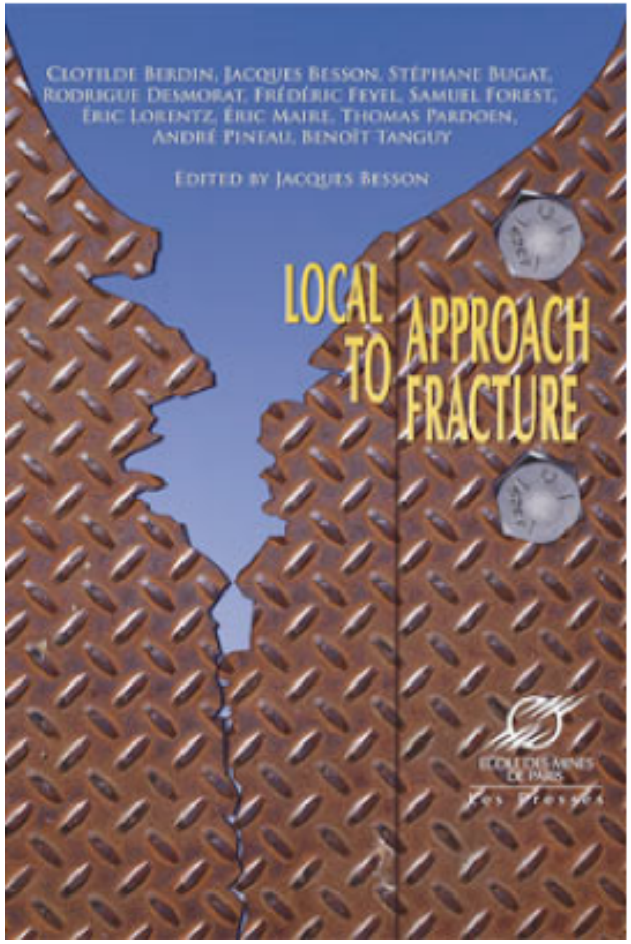
- Documentation :
  - Better code documentation.
  - New entries in the gallery.
- Numerical stability tests based on the `CADNA` library and `VERROU` :
  - `www.lip6.fr/cadna`, `https://github.com/thelfer/cadna`
  - `https://github.com/edf-hpc/verrou`
- Support of units :
  - Compile-time checks without any runtime overhead.
- More work on implicit algorithms :
  - Trust-region algorithms.
  - Homotopy methods.
- New interfaces (ANSYS, DIANA FEA, ...).
- An official Debian/Ubuntu package.

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

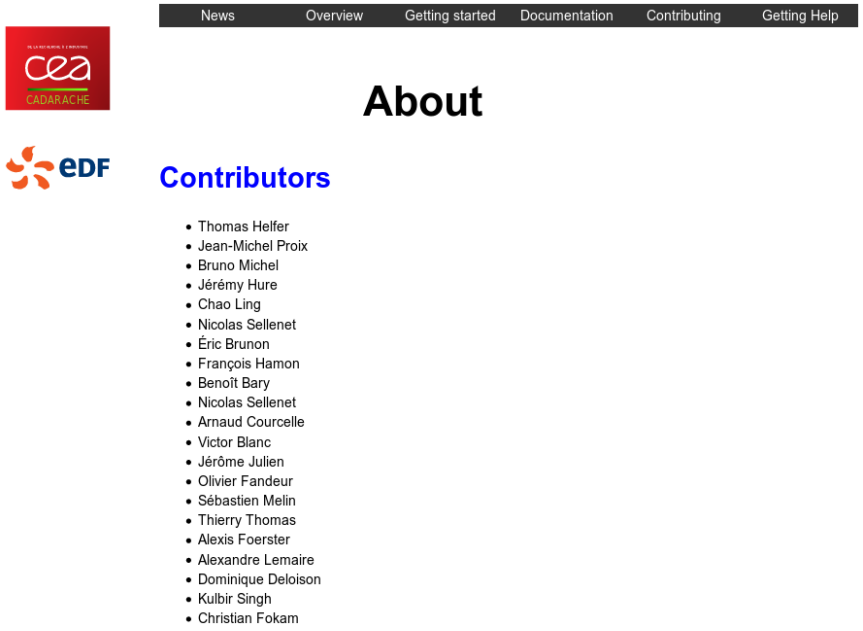
- Port to C++−17

A huge technical depth





How to contribute



- Citations and illustrations
- Feed-backs, feed-backs, and feed-backs !
  - Please use the forum.
  - Enhancement suggestions (code, documentation, algorithm, etc...)
- Submit new behaviours implementation and tests.
- Submit pages to the gallery.
- Code (for the braves)

Thank you for your attention.

Time for discussion !

https://tfel.sourceforge.net
https://twitter.com/TFEL_MFront
https://www.openhub.net/p/tfel
https://github.com/thelfer/

tfel-contact@cea.fr

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