TFEL-5.1 and MGIS 3.1

MFront Users Meeting

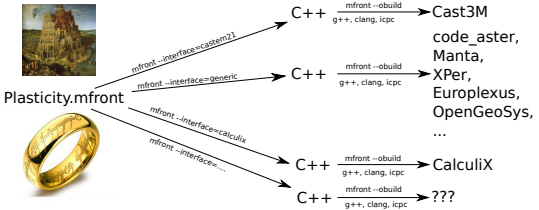
(1) CEA, DES, IRESNE, DEC, SESC, LMPC, Cadarache, France

T. Helfer(1), Antoine Martin(1), Mehran Ghasabeh, Maxime Mollens (and many others!)

Outline

1. Overview

MFront goals



- `MFront` is a code generation tool dedicated to material knowledge (material properties, mechanical behaviours, point-wise models):
  - Support for small and finite strain behaviours, cohesive zone models, **generalised behaviours**.
  - Point-wise models can also be used to create **post-processings**.
- Main goals:
  - Numerical efficiency (see various benchmarks on the website).
  - Portability: `Cast3M`/`Cyrano`/`code_aster`, `Europlexus`/`AMITEX_FFTP`/`Abaqus`/`CalculiX`/`FEniCS`, `CPU`/`GPU`, `C`/`fortran`/`python`/`Julia`, etc.
  - **Long term support and stability**: `MFront`'s implementations reflects valuable knowledge accumulated over decades about materials and are used in safety critical studies.
  - **Ease of use**: *Longum iter est per praecepta, breve et efficax per exempla* (It's a long way by the rules, but short and efficient with examples).

An example of the `StandardElasticityVicoPlasticity` brick

```
@DSL Implicit;
@Behaviour MohrCoulomAbboSloan3;

@Epsilon 1.e-14;
@Theta 1;

@Brick StandardElastoViscoPlasticity{
  stress_potential : "Hooke" {
    young_modulus : 150.e3,
    poisson_ratio : 0.3
  },
  inelastic_flow : "Plastic" {
    criterion : "MohrCoulomb" {
      c : 3.e1,          // cohesion
      phi : 0.5235987755982999,   // friction angle or dilatancy angle
      lodeT : 0.506145483078356,  // transition angle as defined by Abbo and Sloan
      a : 1e1          // tension cuff-off parameter
    },
    flow_criterion : "MohrCoulomb" {
      c : 3.e1,          // cohesion
      phi : 0.174532925199433,   // friction angle or dilatancy angle
      lodeT : 0.506145483078356,  // transition angle as defined by Abbo and Sloan
      a : 3e1          // tension cuff-off parameter
    },
    isotropic_hardening : "Linear" {R0 : "0"}
  }
};
```

- The `StandardElasticityVicoPlasticity` brick allows to implement complex visco-plastic behaviours with a declarative syntax using predefined components.

An simple example with the Implicit DSL

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

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

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

@Integrator{
  constexpr const auto Mᵉ = Stensor4::M();
  const auto μ = computeMu(E, ν);
  const auto σᵉ = sigmaeq(σ);
  const auto iσᵉ = 1 / (max(σᵉ, real(1.e-12) · E));
  const auto vᵖ = A · pow(σᵉ, nn);
  const auto ∂vᵖ/∂σᵉ = nn · vᵖ · iσᵉ;
  const auto n = 3 · deviator(σ) · (iσᵉ / 2);
  // Implicit system
  fεᵉˡ += Δp · n;
  fp -= vᵖ · Δt;
  // jacobian
  ∂fεᵉˡ/∂Δεᵉˡ += 2 · μ · θ · dp · iσᵉ · (Mᵉ - (n ⊗ n));
  ∂fεᵉˡ/∂Δp = n;
  ∂fp/∂Δεᵉˡ = -2 · μ · θ · ∂vᵖ/∂σᵉ · Δt · n;
} // end of @Integrator
```

- Implicit integration.
- Implicit system:

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

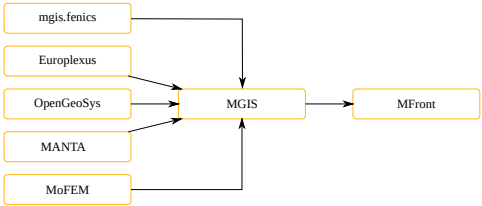
- Jacobian:

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

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

- Tensorial operations are provided by the `TFEL/Math` library
- The `TFEL/Material` library provides useful physical functions

Ecosystem



- The `MFrontGenericInterfaceSupport` (`MGIS`) project eases the support of `MFront`'s generated behviours in solvers.
  - Written in `C++`. Interfaces exist for `C`, `Fortran2003`, `python`, `Julia`.
  - Used in `code_aster`, `MFEM/MGIS`, `Europlexus`, `OpenGeoSys`, `XPer`, `Manta`, `dofinx_materials` and many more...
- `MFrontGallery`: a project to compile, test and deploy `MFront` implementations based on the `CMake` build system.
- `TFELMathEnzyme`: a library leveraging `Enzyme` to provide automatic differentiation (experimental).
- `MFrontJIT`: just in time compilation for `MFront`, to be integrated in `MGIS`

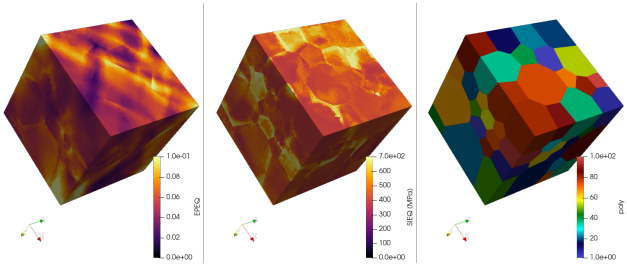
Outline

2. Highlights

Trainings

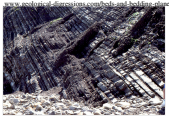
- 2025
  - Centrales Nantes
  - EDF Enery UK, Manchester
  - CEA Saclay (SEMT)
- 2026 (planned)
  - CEA Cadarache (February)

Calculations of aggregates in crystal plasticity with code_aster



- Finite strain single crystal laws
- Phenomenological and physics-based laws successfully implemented in MFront
- Robust and efficient
- "Slowness" inherent in crystalline plasticity formalismwith classical Newton scheme
  - $\approx$ 30 iterations per increment
- Difficulty managing periodic boundary conditions in code_aster
- Plastic incompressibility poorly respected in large transformations?

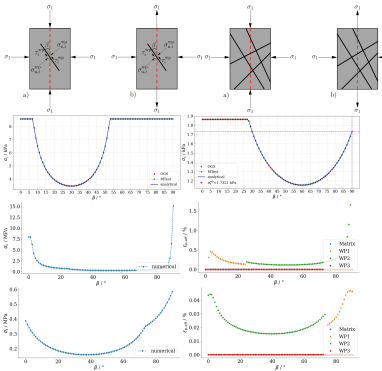
Ubiquitous Joint Model with Three Oblique Joint Sets



$$F^{\mathrm{wp}} = \sigma_{\mathrm{n,i}}^{\mathrm{wp}} \sin\varphi + \left(\tau^m \cos^m\varphi + a^m \sin^m\varphi\right)^{1/m}$$

$$G^{\mathrm{wp}} = \sigma_{\mathrm{n,i}}^{\mathrm{wp}} \sin\psi + \left(\tau^m \cos^m\psi + a_{\mathrm{G}}^m \sin^m\psi\right)^{1/m}$$

$$\sigma_{\mathrm{n,i}}^{\mathrm{wp}} = \boldsymbol{\sigma} : \boldsymbol{n}_{\mathrm{i}}^{\mathrm{wp}} \otimes \boldsymbol{n}_{\mathrm{i}}^{\mathrm{wp}}, \quad \text{and} \quad \tau_{\mathrm{i}} = \sqrt{\boldsymbol{t}_{\mathrm{i}} \cdot \boldsymbol{t}_{\mathrm{i}} - (\boldsymbol{\sigma} : \boldsymbol{n}_{\mathrm{i}}^{\mathrm{wp}} \otimes \boldsymbol{n}_{\mathrm{i}}^{\mathrm{wp}})^2}$$



- Introduces ubiquitous joint Model incorporating three distinct oblique joint sets.
- Captures anisotropic mechanical behaviour arising from multiple non-orthogonal discontinuities.
- Suitable for simulating geomaterials with complex joint networks under various loading conditions.
- Implements tension cutoff criterion to limit unrealistic tensile stresses.
- Coupling with OGS processes.

Ghasabeh and Nagel (2025)
TU Bergakademie Freiberg, Germany

- Implementation available in OGS:
  - https://gitlab.opengeosys.org/MehranGhasabeh/ogs-mg/-/blob/ogs-Oblique/MaterialLib/SolidModels/MFront/UBI3ObliqueTC.mfront?ref_type=heads

Transversely Isotropic Elastoplastic Damage Model

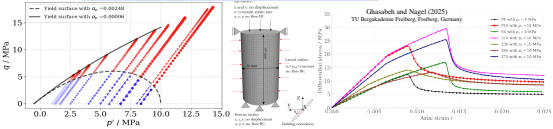
The plastic strain rate is calculated through a flow rule equation

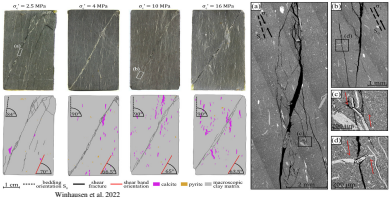
$$\dot{\varepsilon}^{\mathrm{pl}} = \dot{\lambda}\frac{\partial G}{\partial \sigma}$$

The yield and the plastic potential functions are

$$F := \hat{F}(I_1, J_2, \theta, \alpha_p) = J_2 - \frac{(-\alpha_p I_1^n + \gamma I_1^2)}{\sqrt{\exp(\beta_1 I_1) - \beta\cos(3\theta)}}$$

$$G := \hat{G}(I_1, J_2, \theta, \alpha_q) = J_2 - \frac{(-\alpha_q I_1^n + \gamma I_1^2)}{\sqrt{\exp(\beta_1 I_1) - \beta\cos(3\theta)}}$$





Winhausen et al. 2022

- Develops a single plastic surface with capping response.
- Ansisotropic stiffness and strength due to bedding orientation.
- Pre-peak strain hardening.
- Post peak damage evolution for softening.
- Suitable for model developed for clay rock.
- Coupling with OGS processes.

Ghasabeh and Nagel (2025)
TU Bergakademie Freiberg, Germany

- Implementation available in OGS:
  - https://gitlab.opengeosys.org/MehranGhasabeh/ogs-mg/-/blob/ogs-Oblique/MaterialLib/SolidModels/MFront/DesaiTransverselyIsotropicHardeningDamage.mfront?ref_type=heads

MFrontGallery

```
cmake ../MFrontGallery/ -DCMAKE_BUILD_TYPE=Release -Denable-c=ON -Denable-c++=ON -Denable-excel=ON -Denable-fortran=ON  \
-Denable-python=ON -Denable-java=ON -Denable-octave=ON -Denable-generic=ON -Denable-castem=ON -Denable-aster=ON -Denable-cyrano=ON  \
-Denable-ansys=ON -Denable-europlexus=ON -Denable-calculix=ON -Denable-abaqus=ON -Denable-abaqus-explicit=ON -Denable-diana-fea=ON  \
-Denable-zmat=OFF -Denable-fortran-behaviours-wrappers=ON -Denable-mfront-documentation-generation=ON  \
-DCMAKE_INSTALL_PREFIX=/home/th202608/codes/MFrontGallery/master/install
```

- The `MFrontGallery` project provides a `cmake` infrastructure to build, deploy and tests MFront implementations.
  - https://thelfer.github.io/MFrontGallery/web/index.html
  - https://github.com/thelfer/MFrontGallery
- An overview of the project is given in the recently published paper in the Journal Of Opensource Software [2]
- Used in the FailureCriteria project (see J. Hure's talk)

MFEM/MGIS

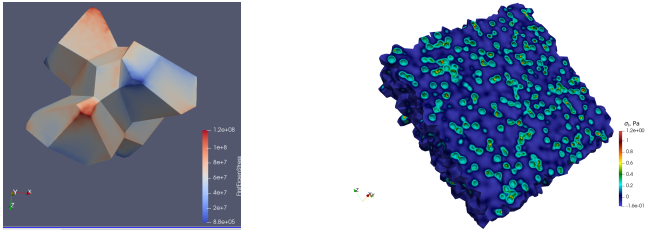


Figure 1: Von Mises equivalent stress of the Cauchy stress in a polycristal (left). Calculation of the first principal stress σ_I induced by a unitary internal pressure in spherical porosities (right).

- The `MFEM/MGIS` project provides an HPC thermomechanical solver for mesocale simulations.
  - `https://thelfer.github.io/mfem-mgis/index.html`
  - `https://github.com/thelfer/mfem-mgis`
  - This solver is developed in the framework of the European Opera HPC project.
- An overview of the project is given in the recently published paper in the Journal Of Opensource Software [1]

Outline

3. Evolutions and new features in TFEL libraries

Some evolutions of the TFEL/Math library

- A static method named `zero` is now available to create fixed-size tensorial objects.

```
constexpr auto s = stensor<2u, double>::zero();
```

- The `View` class allows to interpret a contiguous memory area as a tensorial object. Such views are not efficient on GPUs where coalescent memory access are preferable. The newly introduced `CoalescedView` class offers a solution to this issue. A coalesced view is initialized by an array of pointers to the components of the mapped object, as illustrated in the following snippet:

```
int values[8] = {1, 10, 2, 20, 3, 30, 4, 40};
std::array ptrs{&values[0], &values[2], &values[4], &values[6]};
auto s1 = map<stensor<2u, int>>(ptrs);
```

New library TFELMFrontDatabase

- The TFELMFrontDatabase library provides the MFrontDatabase class which allows to analyse libraries generated by MFront.

```
import mfront.database
db = mfront.database.MFrontDatabase()
r = db.analyseDirectory('/tmp/test-lib/')
epts = db.getEntryPoints(type = 'behaviour',
                         interface = 'generic',
                         material = 'concrete'):
for e in epts:
    print(f"{e.library}: {e.name}")
```

- Entry points can be filtered by name, material, interface and type (behaviour, material property or point-wise model).
- The database can be built from a list of directories or using an environment variable.
- See usage in MGIS later in this talk.

TFEL/Material

- `TFEL/Material`'s evolutions were mainly driven by the developments of the `TFEL/Material/Homogenization` library
  - See A. Martin's talk

Outline

4. Evolutions and new features in MFront

Warnings

- Many warnings have been added to detect potential misuses of `MFront` or known bad practices. A typical example of this feature is the following:

```
$ mfront --interface=generic Plasticity.mfront
[warning]: using the default value for the convergence threshold.
This value is generally considered too loose.
You may want to consider a more stringent value (1e-14 is a good choice).
See the `@Epsilon` keyword for details.
```

- The user has many ways of controlling how warnings are treated, or how to activate/deactivate them.
  - Reporting warnings is activated by default and can be disabled by passing the command line argument `--report-warnings=false` to `MFront`.
  - Warnings are associated with keywords and code blocks. Warnings can be disabled by appending the `safe` option to them:

```
@Epsilon<safe> 1;
```

  - This `safe` option can be ignored by passing the `--ignore-safe` command line argument to `MFront`. This argument is useful when analysing an existing file (written by another person) to question implementation choices.
- **Proposals for new warnings are welcomed !**

Behaviour variables

```
@BehaviourVariable b1 {
  file: "Plasticity.mfront",
  variables_suffix: "1",
  external_names_prefix: "FirstPhase",
  store_gradients: true,
  store_thermodynamic_forces: true,
  shared_external_state_variables: {".+"}
};
```

- Behaviour variables are variables associated with another behaviour.
  - https://thelfer.github.io/tfel/web/behaviour-variable.html
  - `mfront --help-keyword=Default:@BehaviourVariable`
- Behaviour variable is useful in many contexts:
  - It allows switching from a numerical implementation to another, for instance from a fast implementation to a robust one if the fast one diverges.
  - It allows easy implementation of various non-linear homogenization schemes. Tutorials are available on the TFEL website for:
    - Implementing the Sachs/Reuss homogenization scheme.
    - **Implementing the Taylor/Voigt homogenization scheme**.
    - Implementing the β-rule homogenization scheme.
  - Thanks to behaviour variables, the `@Model` keyword now supports point-wise models based on the following DSL: `DefaultModel`, `RungeKuttaModel`, `ImplicitModel`.

Example of usage (Taylor/Voigt homogenization scheme)

```
@Integrator{
  b1.deto=deto;

  constexpr auto b1_smflag = TangentOperatorTraits<MechanicalBehaviourBase::STANDARDSTRAINBASEDBEHAVIOUR>::STANDARDTANGENTOPERATOR;
  const auto r1 = b1.integrate(b1_smflag,CONSISTENTTANGENTOPERATOR);
  StiffnessTensor Dt1 = b1.getTangentOperator();

  b2.deto=deto;

  constexpr auto b2_smflag = TangentOperatorTraits<MechanicalBehaviourBase::STANDARDSTRAINBASEDBEHAVIOUR>::STANDARDTANGENTOPERATOR;
  const auto r2 = b2.integrate(b2_smflag,CONSISTENTTANGENTOPERATOR);
  StiffnessTensor Dt2 = b2.getTangentOperator();

  updateAuxiliaryStateVariables(b1);
  updateAuxiliaryStateVariables(b2);

  sig = f * sig1 + (1 - f) * sig2;
  if (computeTangentOperator_) {
    Dt = f * Dt1 + (1 - f) * Dt2;
  }
}
```

- The simpliest nonlinear homogeneization scheme.
- For the sake of simplicity, error handling has been removed.
- For large number of phases, so-called behaviour variables factories can become handy.
- https://thelfer.github.io/tfel/web/behaviour-variable.html

Stress update algorithm in isotropic DSLs

```
@DSL IsotropicStrainHardeningMisesCreep{
  use_stress_update_algorithm : true
};
```

- Isotropic DSLs all introduced the elastic strain as a state variable. When the elastic material properties are now to be constant in time, then the Hooke law can be written in an incremental form

$$\underline{\sigma}|_{t+\Delta t} = \underline{\sigma}|_t + \lambda \operatorname{tr} \Delta \underline{\epsilon}^{to} + 2\mu \left(\Delta \underline{\epsilon}^{to} - \Delta p \ \ n|_{t+\theta \Delta t} \right)$$

  where λ and μ are the first Lamé's coefficient and the shear modulus, Δp is the increment of the equivalent plastic strain and $n|_{t+\theta \Delta t}$ the flow direction at the middle of the time step.

- The `use_stress_update_algorithm` DSL option prevents the declaration of the elastic strain and switches to this incremental form to compute the stress.

Isotropic harderning rules

```
@DSL IsotropicPlasticMisesFlow{
 use_stress_update_algorithm: true
};
@Behaviour PlasticityWithVoceHardeningRule;
@Epsilon 1e-15;
@ElasticMaterialProperties {
  young_modulus: 150e9,
  poisson_ratio: 0.3
};
@IsotropicHardeningRule "Voce" {R0 : 125e6, Rinf : 500e6, b : 20};
```

- The `@IsotropicHardeningRule` and `@IsotropicHardeningRules` allow to use the isotropic hardening rules available in the `StandardElastoViscoPlasticity` brick.
- If an isotropic hardening rule is defined in the `IsotropicPlasticMisesFlow` DSL, and if no flow rule is defined, the following flow rule is automatically defined:

```
@FlowRule {
  f = seq - R;
  df_dseq = 1;
  df_dp = -dR_dp;
}
```

Removal of the Europlexus interface

- The native Europlexus interface has been removed, as Europlexus now uses the generic interface through MGIS
  - see https://github.com/thelfer/tfel/issues/739 for details.

Outline

5. Evolutions and new features in MGIS library

Overview of MGIS/Function

```
const auto evaluator = pk1_function | as_tensor<3>                    |
                       from_pk1_to_cauchy (F_function | as_tensor<3>) |
                       rotate_backwards (R_function | as_matrix<3,3>);
const auto ok = assign(ctx, sig | as_stensor<3>, evaluator);
```

- This computes the Cauchy stress in the global frame from the first Piola-Kirchhoff stress known in the material frame.
  - This is a typical post-processing for simulation on a polycrystal
- This expression is only valid in `3D`:
  - generated kernels are optimized for specified tensor size
- **Tensorial operations requires to build `MGIS` against the `TFEL`'s libraries**.
- The syntax is strongly inspired by the `std::ranges` library

Overview of MGIS/Function

- MGIS/Function provides a C++-20 interface inspired by std::ranges to create computational kernels which are:
  - composable and easily extensible,
  - constexpr-friendly, multi-thread safe, exception safe, GPU-friendly,
  - adaptable for most data structures, as provided by the targeted solver, language or platform,
  - compatible with various programming models (OpemMP, Kokkos, SYCL, CUDA, parallel STL, hpx, etc.)
  - integrated with the TFEL/Math and TFEL/Material library (Eigen support is feasible),
- MGIS/Function aims at memory safety.
- Two algorithms are provided currently available assign and scalar_reduce.
  - In general, two algorithms are required to build **all** the point-wise operations: std::for_each and std::transform_reduce.
  - Most parallel programming models provides efficient implementations of those algorithms.
  - The efficiency depends on the underlying data structures. In general, the simplier the access to the values, the more efficient will be the implementations.
  - Simple access to values is highly recommended for GPU programming.

Prospects for `MGIS/Function`

- We aim at developing a lot of ready to use modifiers based on `TFEL/Math` and `TFEL/Material`:
  - Hencky strain, Hosford equivalent stress, Hill stress, stress invariants, localization indicators, etc.
  - simple constitutive equations (linear small strain elasticity, Saint Venant-Kirchhoff elasticity, Fourier's law, Euler beam, etc.)
  - homogeneization (see A. Martin's work on `TFEL/Material/Homogeneization`)
- Since most solvers and languages provide simple data structure that are easily mappable with `std::span`, we can provides pre-compiled ready to used post-processings to `MGIS`'s users.
  - Those pre-compiled implementation can be exposed to `python`, `fortran`, `C`, `Julia` (for which bindings already exist).
- The work presented here will be the basis for a first generic interface for GPU's in `MFront`.
  - The behaviour integration will be distinct from the assembly.
  - Such implementation would be directly usage in FFT solvers.
- A nice perspective would be to make `JIT` compilation of kernels.
- Functions and views with coalescent access to data are useful for performances on GPU.

The `integrate_debug` function

- The `integrate_debug` function can be used to generate debug files to analyse integration failures.
  - This remove the need to compile `MFront` behaviours with the `--@GenerateMTestFileOnFailure=true` option.
- This snippet shows that the `integrate_debug` functions can be used as drop-in replacement for the `integrate` function:

```
const auto r = integrate_debug(v, b);
```

- In case of integration failure, a markdown file is generated by default, holding all the information required to reproduce the error.
  - Generating an `MTest` file will require extensions on the `MTest` side to support arbitrary behaviours.
- Even if this shall be superfluous, data on input can be copied to prevent memory corruption.
- If data on input are **not** copied, the cost of using `integrate_debug` shall be negligible compared to `integrate`
- This function is documented and very customizable:
  - https://thelfer.github.io/mgis/web/behaviour-integration-failure-analysis.html

Database of material knowledge

- If `MGIS` is compiled against the `TFEL` libraries, it leverages the `TFELMFrontDatabase` library to provide a function named `loadFromDatabase`:

```
import mgis
import mgis.behaviour as mgis_bv
db = mgis.getDatabase()
db.analyseDirectoriesListedInEnvironmentVariable('MFRONT_GALLERY_LIBRARIES',
                                                 ignore_errors = True)
b = mgis_bv.loadFromDatabase(name='\\w+Elasticity\\w+', material='Wood',
                             hypothesis = mgis.behaviour.Hypothesis.Tridimensional)
```

- Secret dream: see this feature used in `AsterStudy`

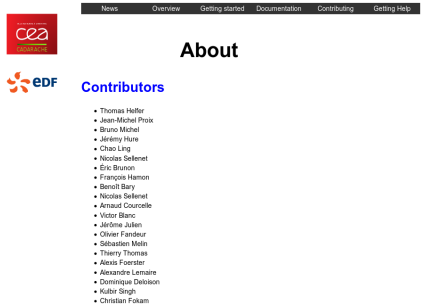
Outline

6. Conclusions and perspectives

Conclusions

- Rewrite code generation in `MFront`
  - Ease extensions of the `StandardElastoViscoPlasticity` brick
- Continue the `MFront` book and improve the documentation.
- Improvement of behaviour variables and behaviour variables factories.
- GPU support:
  - PHD of Tristan Chenaille.
  - Generation of `qfunctions` for `libCEED` and tests in `MFEM` and/or `Ratel`
- Internship dedicated to automatic differentiation and the development of `TFELMathEnzyme`
- Better integration of `MFront`/`MGIS` in `code_aster`:
  - Internship by SIMVIA
- Bindings of `MGIS/Function` in `python`.
- Introduction of `MFrontJIT` in `MGIS`:
  - Integration in `python` for JIT compilations of behaviours described by the `StandardElastoViscoPlasticity` brick

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)

Bibliography I

Thomas Helfer, Guillaume Latu, Raphaël Prat, Maxence Wangermez, and Francesca Cuteri.
MFEM/MGIS, a HPC mini-application targeting nonlinear thermo-mechanical simulations of nuclear fuels at mesoscale.
10(108):7719.

Thomas Helfer, Maxence Wangermez, Eric Simo, Thomas Nagel, Christian B. Silbermann, and Lorenzo Riparbelli.
The MFrontGallery project.
10(109):7742.