A step-by-step implementation of the plastic-viscoplastic behaviour of Aimery Assire

Thomas Helfer, Maxence Wangermez

21/11/2021

Abstract

This report describes how to implement the plastic-viscoplastic behaviour of Aimery Assire in `MFront`.

Contents

Introduction

This tutorial shows how to implement the behaviour proposed by Aimery Assire in his PhD thesis [1] using an implicit scheme.

In this tutorial, plasticity and viscoplasticity will be considered as uncoupled.

1 Some introductory material

This document is meant to help new users to find their ways in the existing documentation of `MFront` and summarize the most important points.

Material properties and models
As most users are focused on mechanical behaviours, this document will not discuss *material properties* and *models*.
Although this document is meant to be self-contained, we highly recommend that the reader first starts reading the tutorial on material properties, which are conceptually much simplier than mechanical behaviours.

`MFront` has been designed to help the user writing complex mechanical behaviours using a domain specific language that are:

- numerically efficient.
- portable from one solver to another.
- easy to read (by your colleagues) and maintain.

Before writing your own behaviour, a few advices are helpful:

- You must have some knowledge of this behaviour and of its physical meaning. See the following classical textbooks for a good introduction [2–5]. **If you are new to finite strain single crystal plasticity, do not try to directly study the examples found in the `MFront` tests base.** It highly recommended to have some basic understanding of the classical algorithms used to integrate constitutive equations (see [2, 4, 6, 7]).
- You must develop your behaviour *step by step*. Do **not** neglect unit testing. This is what the `MTest` tool has been made for.
- You must have some knowledge about your finite element solver, in particular when using "advanced" features (finite strain, non local models, etc…).

The final advice is the more important: as every open-source project, `MFront` has its community of users. **Do not hesitate to ask questions**:

- on the forum.
- to the authors.

1.1 Understand how your mechanical solver works and the role of the mechanical behaviour in the resolution of the mechanical equilibrium

For the sake of simplicity, we consider in this paragraph a behaviour written in the realm of infinitesimal transformations.

1.1.1 Main role of mechanical behaviour

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

Figure 1: Main role of the mechanical behaviour

Given a strain increment $\Delta\,\underline{\varepsilon}^{\mathrm{to}}$ over a time step $\Delta\,t$, a mechanical behaviour must compute (see Figure 1):

- The value of the stress $\left.\underline{\sigma}\right|_{t+\Delta\,t}$ at the end of the time step.
- The value of internal state variables $\left.\vec{Y}\right|_{t+\Delta\,t}$ at the end of the time step. Those internal state variables describe the microstructural evolution of the material.
- The consistent tangent operator, for many solvers (see below), which is defined as the derivative of the stress at the end of the time step with respect to the total strain increment $\dfrac{\partial\left.\underline{\sigma}\right|_{t+\Delta\,t}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{to}}}$.
- For specific cases, the mechanical behaviour shall also provide:
  - The speed of sound
  - A prediction operator (`code_aster`)
  - The elastic operator (`Abaqus-Explicit`, `Europlexus`)
  - The estimation of the stored and dissipated energies (`Abaqus-Explicit`)

Advanced features

- Provide a estimation of the next time step for time step automatic adaptation
- Check bounds:
  - Physical bounds
  - Standard bounds
- Clear error messages
- 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 (MGIS/MTest/Aster/Europlexus):
  - Numbers Types (scalar, tensors, symmetric tensors)
  - Entry names /Glossary names…

1.1.1.1 Interfaces available for mechanical behaviours

The list of available interfaces for mechanical behaviours can be retrieved using the following command:

```
$ mfront --list-behaviour-interfaces
```

At the time this document is written, the following interfaces are avaiable: `Cast3M` (also used by `TMFTT` and `AMITEX\_FFTP`), `code_aster`, Abaqus Standard, Ansys, `CalculiX`, `Europlexus`, `dianafea`.

A special interface, named `generic` is also available. This special interface is meant to work with the `MGIS` project [8], which is used by many solvers to support the usage of`MFront`behaviours, including`MANTA`,`OpenGeoSys`,`MoFEM`, etc.

1.1.2 External state variables

In practice, Figure 1 is incomplete and mechanical behaviours may require to know the evolution some other state variables of the material, i. e. state variables whose evolution is not handled by the mechanical behaviour. Such state variables are depicted as **external** (to the mechanical behaviour) in `MFront`.

A typical example of external state variable is the temperature. Other examples include phase fraction, chemical composition, moisture, etc…

Note that if most solvers support the definition of the temperature as an external state variable, some, such as `Ansys`, don't support other ones.

1.1.3 Material properties and parameters

`MFront` implementations can be classified in two main categories:

- **self-contained**, which denotes implementations that contain all the physical information (e.g., model equations and material coefficients).
- **generic**, which denotes implementations for which the solver is required to provide additional physical information about the material treated, e.g. the values of certain coefficients. Those "generic" implementations are usually shipped with solvers as ready-to-use behaviours.

An alternative way of expressing the distinction between self-contained and generic implementations is to consider that generic implementations only describe a set of constitutive equations while self-contained implementations describes a set of constitutive equations **and** the material coefficients specific identified on a well-defined set of experiments for a particular material.

About material coefficients
In practice, the physical information described by self-contained implementations may be more complex than a set of material coefficients. For example, the Young modulus of a material may be defined by an analytical formula and cannot thus be reduced to a set of constants. This analytical formula shall be part of a self-contained mechanical behaviour implementation. Of course, this analytical formula could be included in the set of constitutive equations and parametrized to retrieve a certain degree of generality. In our experience, such a hybrid approach is fragile, less readable and cumbersome. Moreover, it does not address the main issue of generic behaviours which is the management of the physical information in a reliable and robust way.

In the authors' experience, self-contained behaviours allows to **decouple the material knowledge management from the development (source code) of the solvers of interest** and allows to set up a strigent material knowledge management process. This statement is described in depth in the description of the `MFrontGallery` project.

1.1.3.1 Material properties

In `MFront`' wording, the material coefficients that must be provided to a generic behaviour by the calling solver are called **material properties**.

It is worth emphasing that some solvers only support to define uniform material properties, while other allow to evaluate the material properties at each integration points. For example, the latter solvers thus allow to evaluate a Young modulus as a function of the temperature outside of the mechanical behaviour.

1.1.3.2 Parameters On the other hand, self-contained implementations may sometimes too restrictive, typically for identification or for uncertainty quantification.

To solve this issue, `MFront` introduces the concept of *parameters*. A parameter is a variable which must be declared with a default value.

This variable is stored in some global memory and `MFront`' interfaces can provide ways to modify those global variables.

1.2 Understand how `MFront` works

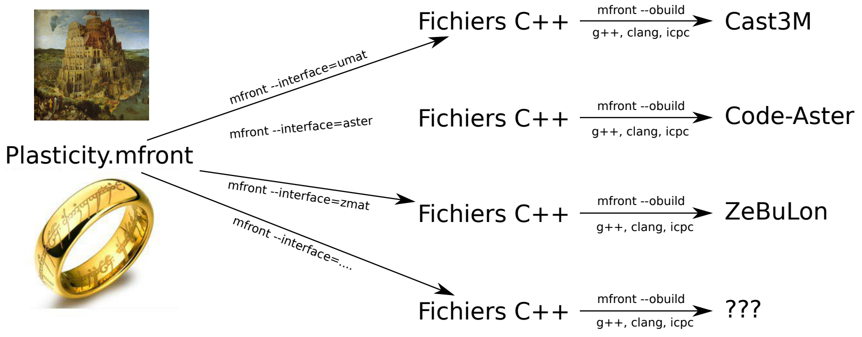

Figure 2: Illustration of `MFront` code generation: from an unique implementation file, `MFront` generates `C++` code which is specific to the targeted solver.

`MFront` translates a set of input files into `C++` sources and compiles them as depicted in Figure 2. A basic knowledge of the `C` and `C++` syntax is then advisable.

`MFront` first introduces the notion of `Domain Specific Language` (`DSL`), which gives a context about what the input file describes, mainly the type of the material knowledge treated (material property, model, finite strain behaviour, strain based behaviour, cohesive zone moel) and/or the algorithm used to integrate the state variables' evolution.

The list of available DSLs can be retrieved as follows:

```
$  mfront --list-dsl
available dsl:
- DefaultCZMDSL                  : this parser is the most generic one as
it does not make any restriction on the behaviour or the integration method
that may be used.
- DefaultDSL                     : this parser is the most generic one as
it does not make any restriction on the behaviour or the integration method
that may be used.
....
```

The `Implicit` DSL
For most mechanical behaviours, the `Implicit` DSL is generally recommended for moderate to complex behaviours: the behaviour integration is performed using a generalised mid-point rule [4, 6, 7] and allows the computation of the consistent tangent operator [9], which is required by most finite element solvers to have decent numerical performances.
The `Implicit` DSL is documented on this page

Each `DSL` has its own conventions, keywords and automatically declared variables. `MFront` specific keywords begins with the `@` letter.

Once the appropriate `DSL` choosen, the list of available keywords can be retrieved from the command line:

```
$ mfront --help-keywords-list=Implicit
```

where `Implicit` is the name of the `DSL`.

The help of for a specific keyword can be retrieved as follows:

```
$ mfront --help-keywords-list=Implicit:@Author
```

where `Implicit` is the name of the `DSL` and `@Author` the name of the keyword.

A good way to get familiar with a specific DSL is to look in the `MFront` gallery if some examples describe its usage.

Otherwise, one may want to look at the tests distributed with `MFront` in the `mfront/tests/behaviours` directory (online access on github here).

About the tests cases
Beware that the implementations in `mfront/tests/behaviours` may not reflect the best way to implement a specific behaviour: `MFront` evolves a lot and each version introduces new features which simplifies the implementation of behaviours. A typical example is the `StandardElastoViscoPlasticity` brick which is described in Section 1.3.1. Some implementations in `mfront/tests/behaviours` are thus out-dated, but kept to ensure the backward compatibility of `MFront`.

Detailed documentation
This page references most of the available documentations.
For example, the complete description of the `DSL`s available can be found here and here, in French. However, such detailed documentation is not required for most users. We even consider that beginning with `MFront` from these documents can be overwhelmingly difficult. Reading the detailed example of the gallery is much easier: `Longum iter est per praecepta, breve et efficax per exempla` (It's a long way by the rules, but short and efficient with examples).
For French users, a (quite out-dated) tutorial can be found here

1.3 Before writing your first behaviour

1.3.1 Look if your behaviour can be implemented using the `StandardElastoViscoPlasticity` brick

The `StandardElastoViscoPlasticity` brick can be used to implement a large class of strain based elasto-visco-plastic behaviours implementations.

It is worth to look if your behaviour falls in that category, or a least if part of your behaviour falls in that category:

- In the first case, implementing the behaviour using the `StandardElastoViscoPlasticity` brick can be straightforward. Even if you do not want to use the `StandardElastoViscoPlasticity` brick, it is worth to have a reference implementation to which one can compare its own implementation in terms of performance, robustness, etc.
- In the second case, implementing a complex behaviour is only tractable step by step. So it is worth to implement first the part which can be implemented using the `StandardElastoViscoPlasticity` brick and validate this part.

1.3.2 Have a look at the `MFront` gallery

The `MFront` gallery is meant to document how to describe in depth and step by step the implementation of some behaviours. If one of the described behaviour is close to the one you plan to implement, it is highly recommend to carefully read the associated page and start by modifying the one described in the gallery.

1.3.3 Write your constitutive equations as a system of ordinary differential equations using tensorial notations

Constitutive equations can be expressed as a system of ordinary differential equations which allows to determine the evolution of the state variables of the material. As the state variables are tensorial objects, those differential equations can be written using tensorial operations: this allows to implement the integration of the behaviour for all supported modelling hypothesis. Do not write tensorial operations using indices and loops.

1.3.4 Select your integration algorithm

`MFront` DSLs provides built-in support for various integration algorithms:

- DSLs of the `Default` family let the user write its own integration algorithm from scratch.
- DSLs of the `Runge-Kutta` family is based on explicit Runge-Kutta algorithms to integrate the constitutive equations. Thoses algorithms are seldom used as they do not provide a simple way to compute the consistent tangent operator.
- DSLs of the `Implicit` family perform the behaviour integration using a generalised mid-point rule [4, 6, 7].

1.3.5 Write your discretized equations by hand using tensorial notations

Apart from DSLs of the `Runge-Kutta` family which allow a direct transcription of the system of ordinary differential equations driving the state variable evolution, the other algorithms requires to discretize the constitutive equations.

1.3.6 Look at the documentation of the tensor operations

Second and fourth orders tensors are the basic mathematical tools used to describe mechanical behaviours. `MFront` is based on the `TFEL/Math` library which provides many standards operations and functions to handle them. See here for an introduction.

1.4 A basic example: isotropic elasticity in small strain

The following code shows how to implement the behaviour describing isotropic elasticity in small strain using the `Default` DSL:

```
@Behaviour Elasticity;

@MaterialProperty stress young;
@MaterialProperty real nu;

@Integrator{
  // second order identity for symmetric tensors
  constexpr auto id = StrainStensor::Id();
  // Lamé coefficients
  const auto lambda = computeLambda(young, nu);
  const auto mu = computeMu(young, nu);
  // strain at the end of the time step
  const auto e = eto + deto;
  // Hooke law
  sig = lambda * trace(e) * id + 2 * mu * e;
}

@TangentOperator{
  const auto lambda = computeLambda(young, nu);
  const auto mu = computeMu(young, nu);
  Dt = lambda * Stensor4::IxI() + 2 * mu * Stensor4::Id();
}
```

We won't describe this file in depth in this paragraph. We simply note that:

- A `MFront`' source file is a simple text file.
- `MFront`' keywords starts with the `@` letter. Some keywords are associated with metadata (such as the `@Behaviour` keyword), others to variable declaration (such as the `@MaterialProperty` declaration) and others introduces code blocks (`@Integrator` and `@TangentOperator` in this example). Code blocks are written using standard `C++` (`MFront` only automatically declares some types, such as `stress, strain`,

etc. and some namespaces to have access to the functions and objects provided by the `TFEL/Math` library (tensors and operations on tensors) and `TFEL/Material` library).

- The code uses tensorial objects, in order to mimic the mathematical expressions. Tensors and operations on tensors provided by are described on this page.

About editing `MFront` files
Most text editors have special modes for `C++` highlighting and we strongly recommand to associate files with the extension `.mfront` with `C++` mode.
Note that an experimental text editor called `tfel-editor` with specific support for `MFront` is also available here: https://github.com/thelfer/tfel-editor

1.4.1 Compilation

This implementation, if written in a file named `Elasticity.mfront`, can be compiled as follows, for the `generic` interface:

```
$ mfront --obuild --interface=generic Elasticity.mfront
Treating target : all
The following library has been built :
- libBehaviour.so :  Elasticity_AxisymmetricalGeneralisedPlaneStrain Elasticity_Axisymmetrical
  Elasticity_PlaneStrain Elasticity_GeneralisedPlaneStrain Elasticity_Tridimensional
```

The output shows that `MFront` has generated a shared library several functions implementing the behaviour, one per supported modelling hypothesis.

1.4.2 Testing

The previous shared library can be tested using `MTest` at the level of a material point using the following input file:

```
@Behaviour<generic> "src/libBehaviour.so" "Elasticity";
// material properties
@MaterialProperty<constant> "young" 150e9;
@MaterialProperty<constant> "nu" 0.3;
// external state variable
@ExternalStateVariable "Temperature" 293.15;
// loadings
@ImposedStrain "EXX" {0 : 0, 1 : 1.e-2};
// time discretisation
@Times {0, 1};
```

By default, `MTest` assumes that the tridiemensional modelling hypothesis is used and selects automatically the appropriate function.

This test imposes the value of the component ε_{xx} of the strain (named `EXX` by convention in `MTest`) from 0 at time 0 to 10^{-2} at time 1. `MTest` is meant to find the evolution of the strain tensor which satisfies this condition and lead to stress components egal to 0, except for the component σ_{xx}.

As described by the `@Times` keyword, only one time step is performed in this example, from time 0 to time 1.

Assumed that the previous input file is named `Elasticity.mtest`, the test can be run as follows:

```
$ mtest Elasticity.mtest
```

This command will generated a file named `Elasticity.res` which contains the evolution of the strain, the stress and internal state variables during the test. On our machine, the stress tensor at time 1 has the following value (using a vector notation):

$$\underline{\sigma} = \left(\sigma_x x, \sigma_{yy}, \sigma_{zz}, \sqrt{2}\,\sigma_{xy}, \sqrt{2}\,\sigma_{xz}, \sqrt{2}\,\sigma_{yz}\right)^T = (1.5e+09, -2.34182e-08, 2.66219e-08, 0, 0, 0)^T$$

About conventions used to store tensors in `TFEL/Math`
The conventions used to store tensors in `TFEL/Math` are described on this page.

One can readily check that σ_{xx} is equal to $E\,\varepsilon_{xx}$ where E is the Young's modulus. The other components are null up to the machine accuracy, i.e. $\sigma_{yy}/\sigma_{xx} \approx 1,5\,10^{-17}$.

2 Constitutive equations of the plastic-viscoplastic behaviour of Aimery Assire

2.1 Additive split of the total strain

The total strain $\underline{\varepsilon}^{\mathrm{to}}$ is assumed to be the sum of an elastic strain $\underline{\varepsilon}^{\mathrm{el}}$, a plastic strain $\underline{\varepsilon}^{\mathrm{p}}$ and a viscoplastic strain $\underline{\varepsilon}^{\mathrm{vp}}$ as follows:

$$\underline{\varepsilon}^{\mathrm{to}} = \underline{\varepsilon}^{\mathrm{el}} + \underline{\varepsilon}^{\mathrm{p}} + \underline{\varepsilon}^{\mathrm{vp}} \tag{1}$$

2.2 Hooke law

The elasticity is assumed linear and isotropic, i.e. given by the Hooke law:

$$\underline{\sigma} = \lambda\,\mathrm{tr}\left(\underline{\varepsilon}^{\mathrm{el}}\right)\underline{I} + 2\,\mu\,\underline{\varepsilon}^{\mathrm{el}} \tag{2}$$

where λ and μ are the first and second Lamé parameters.

2.3 Viscoplastic part

The viscoplastic part of the the strain follows a modified associated Norton behaviour where the exponent and the coefficient depend on the equivalent viscoplastic strain p_{vp}.

Both isotropic hardening rule and kinematic hardening are taken into account:

- Isotropic hardening is described using a Voce law. The size of the elastic domain R_v is thus given by the following function of the equivalent viscoplastic strain:

$$R_v\left(p_{\mathrm{vp}}\right) = R_{v_0} + \left(R_{v_\infty} - R_{v_0}\right)\left(1 - \exp\left(-b_v\,p_{\mathrm{vp}}\right)\right)$$

- Kinematic hardening is described using two back-stresses denoted respectively $\underline{X}_{\mathrm{vp}}^{(1)}$ and $\underline{X}_{\mathrm{vp}}^{(2)}$.

The stress criterion is given by the von Mises equivalent stress σ_{eq} of the effective stress $\underline{\sigma} - \underline{X}_{\mathrm{vp}}^{(1)} - \underline{X}_{\mathrm{vp}}^{(2)}$:

$$\sigma_{\mathrm{eq}} = J_2\left(\underline{\sigma} - \underline{X}_{\mathrm{vp}}^{(1)} - \underline{X}_{\mathrm{vp}}^{(2)}\right)$$

The von Mises stress and its derivatives
Let $\underline{v}$ be an arbitrary symmetric second order tensor.
$J_2\left(\underline{v}\right)$ can be written as:

$$J_2\left(\underline{v}\right) = \sqrt{\underline{v}\,:\,\underline{\underline{\mathbf{M}}}\,:\,\underline{v}}$$

where $\underline{\underline{\mathbf{M}}}$ is the projector on the deviatoric space multiplied by $\dfrac{3}{2}$, i.e.:

$$\underline{\underline{\mathbf{M}}} = \frac{3}{2}\left(\underline{\underline{\mathbf{I}}} - \frac{1}{3}\underline{I}\otimes\underline{I}\right)$$

The first derivative of the $J_2\left(\underline{v}\right)$ with respect to $\underline{v}$ is normal to the isovalue of $J_2\left(\underline{v}\right)$. Its value is given by:

$$\frac{\partial J_2}{\partial \underline{v}} = \frac{1}{J_2\left(\underline{v}\right)}\underline{\underline{\mathbf{M}}}\,:\,\underline{v} = \frac{3}{2\,J_2\left(\underline{v}\right)}\underline{\tilde{v}}$$

where $\underline{\tilde{v}}$ is the deviatoric part of $\underline{v}$.
In the following, the second derivative of $J_2\left(\underline{v}\right)$ with respect to $\underline{v}$ will be useful. It can readily be computed as follows:

$$\frac{\partial^2 J_2}{\partial^2 \underline{v}} = \frac{1}{J_2\left(\underline{v}\right)}\left(\underline{\underline{\mathbf{M}}} - \frac{\partial J_2}{\partial \underline{v}}\otimes\frac{\partial J_2}{\partial \underline{v}}\right) \tag{3}$$

The yield function f_{vp} is thus given by:

$$f_{\mathrm{vp}} = \sigma_{\mathrm{eq}} - R_v\left(p_{\mathrm{vp}}\right)$$

As the behaviour is associated, the flow direction $\underline{n}_{\mathrm{vp}}$ is given by:

$$\underline{n}_{\mathrm{vp}} = \frac{\partial \sigma_{\mathrm{eq}}}{\partial \underline{\sigma}} = \frac{3}{2\,\sigma_{\mathrm{eq}}}\,\underline{s}$$

where $\underline{s}$ is the deviatoric part of the effective stress $\underline{\sigma} - \underline{X}_{\mathrm{vp}}^{(1)} - \underline{X}_{\mathrm{vp}}^{(2)}$.

2.3.1 Evolution of the equivalent viscoplastic strain

The rate of viscoplastic deformation is finally given by:

$$\underline{\dot{\varepsilon}}^{\mathrm{vp}} = \dot{p}_{\mathrm{vp}}\,\underline{n}_{\mathrm{vp}} \tag{4}$$

where the viscoplastic equivalent strain rate is given by:

$$\dot{p}_{\mathrm{vp}} = \left(\frac{f_v}{K_{\mathrm{vp}}\left(p_{\mathrm{vp}}\right)}\right)^{E_{\mathrm{vp}}\left(p_{\mathrm{vp}}\right)} = \left(\frac{\sigma_{\mathrm{eq}} - R_v\left(p_{\mathrm{vp}}\right)}{K_{\mathrm{vp}}\left(p_{\mathrm{vp}}\right)}\right)^{E_{\mathrm{vp}}\left(p_{\mathrm{vp}}\right)} \tag{5}$$

with:

- $E_{\mathrm{vp}}\left(p_{\mathrm{vp}}\right) = A_{E_{\mathrm{vp}}}\,p_{\mathrm{vp}}^{C_{E_{\mathrm{vp}}}} + B_{E_{\mathrm{vp}}}$
- $K_{\mathrm{vp}}\left(p_{\mathrm{vp}}\right) = A_{K_{\mathrm{vp}}}\,p_{\mathrm{vp}} + B_{K_{\mathrm{vp}}}$

2.3.2 Evolution of the back strains

Let $\underline{\alpha}_{\mathrm{vp}}^{(1)}$ and $\underline{\alpha}_{\mathrm{vp}}^{(2)}$ be the back-strains associated respectively with the back-stresses $\underline{X}_{\mathrm{vp}}^{(1)}$ and $\underline{X}_{\mathrm{vp}}^{(2)}$ as follows:

$$
\left\{
\begin{aligned}
\underline{X}^{(1)}_{\mathrm{vp}} &= \frac{2}{3}\,C^{(1)}_{\mathrm{vp}}\,\underline{\alpha}^{(1)}_{\mathrm{vp}}\\
\underline{X}^{(2)}_{\mathrm{vp}} &= \frac{2}{3}\,C^{(2)}_{\mathrm{vp}}\,\underline{\alpha}^{(2)}_{\mathrm{vp}}
\end{aligned}
\right.
$$

The evolution of the first back-strain follows a modified Armstrong-Frederik law as follows [10]:

$$
\underline{\dot{\alpha}}^{(1)}_{\mathrm{vp}} = \dot{p}_{\mathrm{vp}}\,\underline{n}_{\mathrm{vp}} - D^{(1)}_{\mathrm{vp}}\,\dot{p}_{\mathrm{vp}}\,\underline{\alpha}^{(1)}_{\mathrm{vp}} - \left(\frac{\left|\underline{X}^{(1)}_{\mathrm{vp}}\right|}{M}\right)^{m}\frac{\underline{X}^{(1)}_{\mathrm{vp}}}{\left|\underline{X}^{(1)}_{\mathrm{vp}}\right|} \tag{6}
$$

The evolution of the second back-strain follows the classical Armstrong-Frederik law as follows:

$$
\underline{\dot{\alpha}}^{(2)}_{\mathrm{vp}} = \dot{p}_{\mathrm{vp}}\,\underline{n}_{\mathrm{vp}} - D^{(2)}_{\mathrm{vp}}\,\dot{p}_{\mathrm{vp}}\,\underline{\alpha}^{(2)}_{\mathrm{vp}} \tag{7}
$$

2.4 Plastic part

The plastic part of the strain takes into account both isotropic hardening and kinematic hardening:

- Isotropic hardening is described using as the sum of two Voce laws. The size of the elastic domain R_p is thus given by the following function of the equivalent plastic strain:

$$
R_p\left(p_{\mathrm{p}}\right) = R_{p_0} + \left(R^{(1)}_{p_\infty} - R_{p_0}\right)\left(1 - \exp\left(-b^{(1)}_p\,p_{\mathrm{p}}\right)\right) + \left(R^{(2)}_{p_\infty} - R_{p_0}\right)\left(1 - \exp\left(-b^{(2)}_p\,p_{\mathrm{p}}\right)\right)
$$

- Kinematic hardening is described using two back-stresses denoted respectively $\underline{X}^{(1)}_{\mathrm{p}}$ and $\underline{X}^{(2)}_{\mathrm{p}}$.

The stress criterion is given by the von Mises equivalent stress σ_{eq} of the effective stress $\underline{\sigma} - \underline{X}^{(1)}_{\mathrm{p}} - \underline{X}^{(2)}_{\mathrm{p}}$:

$$
\sigma_{\mathrm{eq}} = J_2\left(\underline{\sigma} - \underline{X}^{(1)}_{\mathrm{p}} - \underline{X}^{(2)}_{\mathrm{p}}\right)
$$

The yield function f_{p} is thus given by:

$$
f_{\mathrm{p}}\left(\underline{\sigma}, \underline{X}^{(1)}_{\mathrm{p}}, \underline{X}^{(2)}_{\mathrm{p}}, p_{\mathrm{p}}\right) = \sigma_{\mathrm{eq}} - R_p\left(p_{\mathrm{p}}\right)
$$

As the behaviour is associated, the flow direction $\underline{n}_{\mathrm{p}}$ is given by:

$$
\underline{n}_{\mathrm{p}} = \frac{\partial \sigma_{\mathrm{eq}}}{\partial \underline{\sigma}} = \frac{3}{2\,\sigma_{\mathrm{eq}}}\,\underline{s}
$$

where $\underline{s}$ is the deviatoric part of the effective stress $\underline{\sigma} - \underline{X}^{(1)}_{\mathrm{p}} - \underline{X}^{(2)}_{\mathrm{p}}$.

Finally, the plastic strain rate is given by:

$$
\underline{\dot{\varepsilon}}^{\mathrm{p}} = \dot{p}_{\mathrm{p}}\,\underline{n}_{\mathrm{p}} \tag{8}
$$

2.4.1 Evolution of the equivalent plastic strain

The evolution of the equivalent plastic strain follows the classic Karush-Kuhn-Tucker (KKT) conditions:

$$
\left\{
\begin{aligned}
\dot{p}_{\mathrm{p}}\,f_{\mathrm{p}} &= 0\\
\dot{p}_{\mathrm{p}} &\geq 0\\
f_{\mathrm{p}} &\leq 0
\end{aligned}
\right. \tag{9}
$$

2.4.2 Evolution of the back strains

Let $\underline{\alpha}_{\mathrm{p}}^{(1)}$ and $\underline{\alpha}_{\mathrm{p}}^{(2)}$ be the back-strains associated respectively with the back-stresses $\underline{X}_{\mathrm{p}}^{(1)}$ and $\underline{X}_{\mathrm{p}}^{(2)}$ as follows:

$$\begin{cases}\underline{X}_{\mathrm{p}}^{(1)} = \dfrac{2}{3}\,C_{\mathrm{p}}^{(1)}\,\underline{\alpha}_{\mathrm{p}}^{(1)} \\ \underline{X}_{\mathrm{p}}^{(2)} = \dfrac{2}{3}\,C_{\mathrm{p}}^{(2)}\,\phi_c\left(p_{\mathrm{p}}\right)\,\underline{\alpha}_{\mathrm{p}}^{(2)} \quad \text{with} \quad \phi_c\left(p_{\mathrm{p}}\right) = \phi_{p_0}^{(c)} + \left(\phi_{p_\infty}^{(c)} - \phi_{p_0}^{(c)}\right)\,\left(1 - \exp\left(-\omega_p^{(c)}\,p_{\mathrm{p}}\right)\right)\end{cases}$$

The evolution of the first back-strain follows the classical Armstrong-Frederik law as follows:

$$\underline{\dot{\alpha}}_{\mathrm{p}}^{(1)} = \dot{p}_{\mathrm{p}}\,\underline{n}_{\mathrm{p}} - D_{\mathrm{p}}^{(1)}\,\dot{p}_{\mathrm{p}}\,\underline{\alpha}_{\mathrm{p}}^{(1)} \tag{10}$$

The evolution of the second back-strain follows a modified Armstrong-Frederik law as follows:

$$\underline{\dot{\alpha}}_{\mathrm{p}}^{(2)} = \dot{p}_{\mathrm{p}}\,\underline{n}_{\mathrm{p}} - D_{\mathrm{p}}^{(2)}\,\phi_d\left(p_{\mathrm{p}}\right)\,\dot{p}_{\mathrm{p}}\,\underline{\alpha}_{\mathrm{p}}^{(2)} \quad \text{with} \quad \phi_d\left(p_{\mathrm{p}}\right) = \phi_{p_0}^{(d)} + \left(\phi_{p_\infty}^{(d)} - \phi_{p_0}^{(d)}\right)\,\left(1 - \exp\left(-\omega_p^{(d)}\,p_{\mathrm{p}}\right)\right) \tag{11}$$

2.5 Evolution of the elastic strain

The evolution of the elastic strain is simply obtained by the time derivative of the split of the strains (Equation (1)) as follows:

$$\underline{\dot{\varepsilon}}^{\mathrm{el}} = \underline{\dot{\varepsilon}}^{\mathrm{to}} - \underline{\dot{\varepsilon}}^{\mathrm{p}} - \underline{\dot{\varepsilon}}^{\mathrm{vp}} = \underline{\dot{\varepsilon}}^{\mathrm{to}} - \dot{p}_{\mathrm{vp}}\,\underline{n}_{\mathrm{vp}} - \dot{p}_{\mathrm{p}}\,\underline{n}_{\mathrm{p}} \tag{12}$$

where Equations (4) and (8) were used.

3 Implicit scheme

3.1 Choice of the state variables

In this tutorial, the following set of state variables is selected:

$$\vec{Y} = \begin{pmatrix} \underline{\varepsilon}^{\mathrm{el}} \\ p_{\mathrm{vp}} \\ \underline{\alpha}_{\mathrm{vp}}^{(1)} \\ \underline{\alpha}_{\mathrm{vp}}^{(2)} \\ p_{\mathrm{p}} \\ \underline{\alpha}_{\mathrm{p}}^{(1)} \\ \underline{\alpha}_{\mathrm{p}}^{(2)} \end{pmatrix} \tag{13}$$

3.2 Behaviour integration

With a little work, Equations (12), (5) (6), (7), (9) (10) and (11) can be used to express the evolution of the state variables as a system of ordinary differential equations:

$$\dot{\vec{Y}} = G\left(\vec{Y}\right) \tag{14}$$

Treatment of the plastic part
In this paragraph, we will ignore that plasticity deserves a special treatment. Of course, you may use the consistency equation to derive a ordinary differential equation satisfied by the plastic multiplier, but this is, in practice, a very bad idea. As we will see later, it is much better to replace this differential equation by an implicit equation stating that the system must be on the yield surface at the end of the time step.

Many algorithms exist to solve this ordinary differential equations over a finite time step $\Delta\, t$.

3.3 Implicit scheme. Definition of the residual

An implicit scheme turns this ordinary system of differential equations into a system of non linear equations whose unknown is the increment $\Delta\,\vec{Y}$ of the state variables $\vec{Y}$ over the time step $\Delta\, t$.

Let us introduce the following variables and notations:

- t denotes the time at the beginning of the time step.
- θ denotes a numerical parameter between 0 and 1.
- $\left.\vec{Y}\right|_{t}$ denotes the value of the state variables at the beginning of the time step.
- $\left.\vec{Y}\right|_{t+\theta\,\Delta\,t}$ denotes the value of the state variables at an intermediate time $t+\theta\,\Delta\,t$:

$$\left.\vec{Y}\right|_{t+\Delta\,t} = \left.\vec{Y}\right|_{t} + \theta\,\Delta\,\vec{Y}$$

- $\left.\vec{Y}\right|_{t+\Delta\,t}$ denotes the value of the state variables at the end of the time step:

$$\left.\vec{Y}\right|_{t+\Delta\,t} = \left.\vec{Y}\right|_{t} + \Delta\,\vec{Y}$$

Using those notations, we can straightforwardly express the implicit system derived from Equation (14) as follows:

$$\Delta\,\vec{Y} - G\left(\left.\vec{Y}\right|_{t} + \theta\,\Delta\,\vec{Y}\right)\,\Delta\,t = \vec{0} \tag{15}$$

Finally, the increment $\Delta\,\vec{Y}$ of the state variables satisfies the following implicit system of non linear equations:

$$\vec{F}\left(\Delta\,\vec{Y}\right) = \vec{0} \tag{16}$$

with $\vec{F}\left(\Delta\,\vec{Y}\right) = \Delta\,\vec{Y} - G\left(\left.\vec{Y}\right|_{t} + \theta\,\Delta\,\vec{Y}\right)\,\Delta\,t$.

$\vec{F}$ is called the residual of the implicit system.

The `zeros` and `fzeros` variables. Views
Vector $\Delta\,\vec{Y}$ can be accessed through a variable called `zeros` and vector $\vec{F}$ can be accessed through a variable called `fzeros`.
Those variables are seldom used done in practice, except for debugging.
It is much easier to manipulate directly the increments of the state variables and to compute the blocks resulting from the decomposition of $\vec{F}$, as detailled in the Section 3.3.2.
Internally, the increments of the state variables are views to blocks in the `zeros` variable and the blocks resulting from the decomposition of $\vec{F}$ are view to the `fzeros` variables. More details about views are given in the documentation of the `TFEL/Math` library.

3.3.1 Initialisation of the residual

Equation (15) shows that the residual $\vec{F}$ can naturally be initialized to $\Delta\,\vec{Y}$. This is done automatically by `MFront` before each evaluation of the residual.

3.3.2 Block decomposition of the residual

Following Equation (13), the residual can be decomposed as follows

$$\vec{F} = \begin{pmatrix} f_{\underline{\varepsilon}^{\mathrm{el}}} \\ f_{p_{\mathrm{vp}}} \\ f_{\underline{\alpha}_{\mathrm{vp}}^{(1)}} \\ f_{\underline{\alpha}_{\mathrm{vp}}^{(2)}} \\ f_{p_{\mathrm{p}}} \\ f_{\underline{\alpha}_{\mathrm{p}}^{(1)}} \\ f_{\underline{\alpha}_{\mathrm{p}}^{(2)}} \end{pmatrix} \tag{17}$$

Each block is associated with a state variable and can be deduced from the ordinary differential equation associated with this state variable.

For example, the block associated with the elastic strain is associated with the slip of the strain associated written in incremental form:

$$f_{\underline{\varepsilon}^{\mathrm{el}}} = \Delta\,\underline{\varepsilon}^{\mathrm{el}} - \Delta\,\underline{\varepsilon}^{\mathrm{to}} + \Delta\,p_{\mathrm{vp}}\,\underline{n}_{\mathrm{vp}}\Big|_{t+\theta\,\Delta\,t} + \Delta\,p_{\mathrm{p}}\,\underline{n}_{\mathrm{p}}\Big|_{t+\theta\,\Delta\,t} \tag{18}$$

3.3.3 Newton-Raphson algorithm and jacobian matrix

Equation (16) is solved using an iterative algorithm. In this tutorial, the standard Newton-Raphson algorithm will be used.

Let $\Delta\,\vec{Y}^{(n)}$ be the estimation at the n^{th} iteration of the Newton algorithm. $\Delta\,\vec{Y}^{(n+1)}$ is then given by:

$$\Delta\,\vec{Y}^{(n+1)} = \Delta\,\vec{Y}^{(n)} + \delta\,\Delta\,\vec{Y}^{(n)} \quad \text{with} \quad \delta\,\Delta\,\vec{Y}^{(n)} = -J^{-1}\left(\Delta\,\vec{Y}^{(n)}\right)\,\vec{F}\left(\Delta\,\vec{Y}^{(n)}\right)$$

where J is the jacobian matrix given by:

$$J\left(\Delta\,\vec{Y}^{(n)}\right) = \frac{\partial \vec{F}}{\partial \Delta\,\vec{Y}}\left(\Delta\,\vec{Y}^{(n)}\right)$$

In `MFront`, the jacobian matrix can be computed analytically or evaluated numerically using a centered finite difference scheme.

The `jacobian` variable
The jacobian matrix is stored internally in a matrix called `jacobian`. This variable is seldom used in pratice, except for debuggin.

3.3.4 Initialisation of the jacobian

Equation (15) shows that the jacobian J can naturally be initialized to the identity matrix I_d as it can be written:

$$J = I_d + \Delta\,t\,\frac{\partial \vec{G}}{\partial \Delta\,\vec{Y}}$$

This is done automatically by `MFront` before each evaluation of the residual.

Small time steps
It is interesting to note that the jacobian matrix naturally tends to the identity matrix I_d as the time step tends toward zero.

A bad evaluation of the derivative $\dfrac{\partial \vec{G}}{\partial \Delta\,\vec{Y}}$ can thus be compensated by using (very) small time steps.
A contrario, if an implicit behaviour implementation only converges for very small time steps, it may indicate that the jacobian matrix is incorrect (or very approximate).

3.3.5 Block decomposition of the jacobian

Using a block decomposition of the increment of the state variables derived from Equation (17), and the Block Decomposition (17), the jacobian matrix can also be decomposed by blocks as follows:

$$J = \begin{pmatrix} \dfrac{\partial f_{\underline{\varepsilon}^{\mathrm{el}}}}{\partial \Delta\, \underline{\varepsilon}^{\mathrm{el}}} & \cdots & \cdots & \cdots & \dfrac{\partial f_{\underline{\varepsilon}^{\mathrm{el}}}}{\partial \Delta\, \underline{\alpha}_{\mathrm{p}}^{(2)}} \\ \vdots & \ddots & \cdots & \ddots & \vdots \\ \vdots & \vdots & \dfrac{\partial f_{y_i}}{\partial \Delta\, y_j} & \vdots & \vdots \\ \vdots & \ddots & \cdots & \ddots & \vdots \\ \dfrac{\partial f_{\underline{\alpha}_{\mathrm{p}}^{(2)}}}{\partial \Delta\, \underline{\varepsilon}^{\mathrm{el}}} & \cdots & \cdots & \cdots & \dfrac{\partial f_{\underline{\alpha}_{\mathrm{p}}^{(2)}}}{\partial \Delta\, \underline{\alpha}_{\mathrm{p}}^{(2)}} \end{pmatrix}$$

3.3.6 Nonlinear solvers delivered with `MFront`

By default, the standard Newton-Raphson algorithm is used. Other algorithms can be selected using the `@Algorithm` keyword.

```
$ mfront --help-keyword=Implicit:@Algorithm
The `@Algorithm` keyword is used to select a numerical algorithm.

The set of available algorithms depends on the domain specific
language. As the time of writting this notice, the following
algorithms are available:

- `euler`, `rk2`, `rk4`, `rk42` , `rk54` and `rkCastem` for the
  `Runge-Kutta` dsl.
- `NewtonRaphson`, `NewtonRaphson_NumericalJacobian`,
  `PowellDogLeg_NewtonRaphson`,
  `PowellDogLeg_NewtonRaphson_NumericalJacobian`, `Broyden`,
  `PowellDogLeg_Broyden`, `Broyden2`, `LevenbergMarquardt`,
  `LevenbergMarquardt_NumericalJacobian` for implicit dsls.
```

Those nonlinear solvers are implemented in the `TFEL/Math` library. The documentation of this library provides a comprehensive description of the structure of those algorithms.

In practice, as illustrated on Figure 3, `MFront` allows the user to write the most important steps of the non linear algorithm, which are:

- The computation of a prediction of the increment of the state variables in the `Predictor` code blocks. By default, thoses increments are initialized to zero.
- The update of the stress at $t+\theta\,\Delta\,t$ in the `@ComputeStress` code block. As discussed in the next paragraph, this part can be generated automatically by `MFront` by the `StandardElasticity` brick (see Section 3.4.1). This brick also handles the computation of the stress at the end of the time step that is normally deduced implicitly from the code given in the `@ComputeStress` code block or explicitely given in the `@ComputeFinalStress` code block.
- The computation of the residual $\vec{F}$ and jacobian matrix J in the `@Integrator` code block.
- The definition of additional convergence checks in the `@AdditionalConvergenceChecks` code block. This code block is very useful in multi-surface plasticity, or combining laws with thresholds.

After the resolution of the implicit system, the user may compute the consistent tangent operator in the `@TangentOperator` code block. In this document, this code block will be generated automatically by the `StandardElasticity` brick (see Section 3.4.1). This brick will also handle the computation of the prediction operator that is normally introduced by the `@PredictionOperator` keyword.

The other keywords appearing in Figure 3 (such as `@InitLocalVariables`, `@UpdateAuxiliaryStateVariables`) will be detailled later.

3.3.6.1 Default stopping criterion

By default, `MFront` will stop the iterations of the non linear solver if the norm of the residual is below a given threshold:

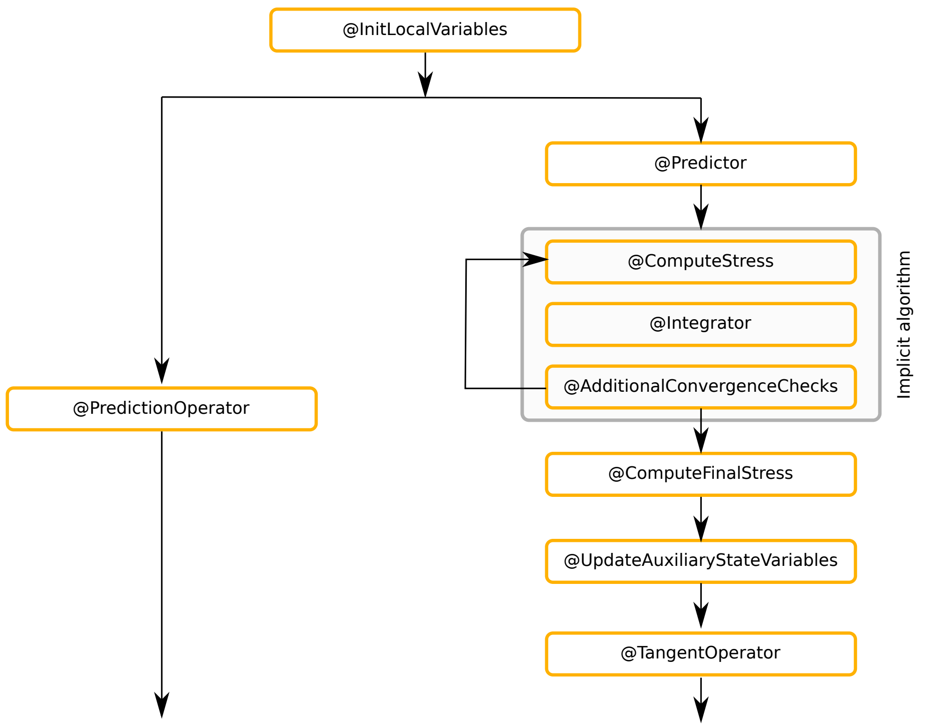

Figure 3: Description of the main steps of the implicit scheme and the associated code blocks.

$$\|\vec{F}\left(\Delta\,\vec{Y}^{(n)}\right)\| < \varepsilon_{\mathrm{NR}}$$

The default value for $\varepsilon_{\mathrm{NR}}$ is 10^{-8} which can be a bit loose. In particular, this value may lead to an inaccurate evaluation of the consistent tangent operator if its computation follows the method described in Section 3.4.1.2. We strongly advice to change this default value to a more strictier value using the `@Epsilon` keyword, as follows:

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

Examples from the `MFront` gallery
Many examples in the `MFront` gallery sets the default value of the stopping criterion at 10^{-16} which is a very strigent value in practice.
Such low value is however very important in unit testing and non regression tests.

The value of the $\varepsilon_{\mathrm{NR}}$ can be changed at runtime using the `epsilon` parameter.

Note that the convergence criterion can be changed or extended by defining an `@AdditionalConvergenceChecks` code block.

3.3.6.2 Default value of implicit parameter

The default value of the implicit parameter θ is 0.5. This value has been recommended by various authors for viscoplastic behaviour as it leads to a quadratic convergence in time.

In practice, a fully implicit resolution is generally more satisfying, so we strongly recommend to change this default value using the `@Theta` keyword as follows:

```
@Theta 1;
```

The value of the implicit parameter can be changed at runtime using the `theta` parameter.

3.3.6.3 Maximum number of iterations

The algorithms must fail if they did not converge after a maximum number of iterations.

The default value of the maximum number of iterations is 100. This default value can be changed using the `@IterMax` keyword as follows:

```
@IterMax 50;
```

The value of the maximum number of iterations can be changed at runtime using the `itermax` parameters.

3.3.6.4 Handling failures to the residual evaluation

The evaluation of the residual may fail for various reasons.

For example, an incorrect prediction of the stress may lead to infinite viscoplastic rate.

In this case, as described in the documentation of the `TFEL/Math` library, `MFront` will take the latest correction computed by the solver and divide it by two. This leaves the search direction unchanged but decrease its intensity by two. This can be though as a sort of basic line-search.

The user may indicate a failure in the evaluation of the jacobian by returning `false` in the `@Integrator` code block based on some physical criteria:

- for a plastic behaviour, one may want to reject a correction leading to a stress estimate well above the yield surface.
- for a viscoplastic plastic behaviour, one may want to reject stress estimates leading to unphysical viscoplastic strain rate.

This can lead to very powerful algorithms.

3.4 Some helpful features of `MFront`

3.4.1 The `StandardElasticity` brick

The plastic-viscoplastic behaviour of Aimery Assire falls in the realms of the so-called `StandardElasticity` brick which automatically:

- Generates the code for the computation of the stress at $t+\theta\,\Delta\,t$ (i.e. the code normally generated by the `@ComputeStress` code block) and the computation of the stress at $t+\Delta\,t$ (i.e. the code normally generated by the `@ComputeFinalStress` code block).
- Generates the code for the computation of the prediction operator, i.e. the code that is normally generated by the `@PredictionOperator` code block.
- Generates the code for the computation of the consistent tangent operator, i.e. the code that is normally generated by the `@TangentOperator` code block.
- Generates the code for the support for plane stress and generalized plane stress modelling hypotheses.

This brick assumes that:

- The total strain increment $\underline{\varepsilon}^{\mathrm{to}}$ only appears in the implicit system inside the split of strain (see Equation (1)).
- The stress is computed by the Hooke law (see Equation (2)).
- The elastic strain increment is stored at the beginning of $\vec{Y}$ vector and the implicit equation $f_{\underline{\varepsilon}^{\mathrm{el}}}$ associated with the elastic strain, is stored at the beginning of the residual $\vec{F}$.

The usage of the `StandardElasticity` brick is declared as follows:

```
@Brick StandardElasticity;
```

3.4.1.1 Automatic support of the plane stress and axisymmetrical generalised plane stress hypotheses

The axial strain $\varepsilon^{\mathrm{to}}_{zz}$ is defined as an additional state variable and the associated equation in the implicit system is added to enforce the plane stress condition.

$$f_{\varepsilon^{\mathrm{to}}_{zz}} = \frac{1}{E}\,\sigma_{zz}\big|_{t+\Delta\,t} \tag{19}$$

where E is the Young modulus.

$\Delta\,\varepsilon^{\mathrm{to}}_{zz}$ is added to the implicit equation $f_{\underline{\varepsilon}^{\mathrm{el}}}$ (see Equation (18)) as follows:

$$f_{\underline{\varepsilon}^{\mathrm{el}}} - = \Delta\,\varepsilon^{\mathrm{to}}_{zz}\,\vec{e}_z \otimes \vec{e}_z$$

3.4.1.2 Automatic derivation of the consistent tangent operator The consistent tangent operator is by definition the derivative $\dfrac{\partial\,\underline{\sigma}\big|_{t+\Delta\,t}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{to}}}$ of the stress at the end of the time step with respect to the total strain increment.

According to the Hooke law (2), this derivative can be expressed as:

$$\frac{\partial\,\underline{\sigma}\big|_{t+\Delta\,t}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{to}}} = \underbrace{\frac{\partial\,\underline{\sigma}\big|_{t+\Delta\,t}}{\partial\,\underline{\varepsilon}^{\mathrm{el}}\big|_{t+\Delta\,t}}}_{\underline{\underline{\mathbf{D}}}\big|_{t+\Delta\,t}} : \underbrace{\frac{\partial\,\underline{\varepsilon}^{\mathrm{el}}\big|_{t+\Delta\,t}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{el}}}}_{\underline{\underline{\mathbf{I}}}} : \frac{\partial\Delta\,\underline{\varepsilon}^{\mathrm{el}}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{to}}} = \underline{\underline{\mathbf{D}}}\big|_{t+\Delta\,t} : \frac{\partial\Delta\,\underline{\varepsilon}^{\mathrm{el}}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{to}}} \tag{20}$$

with $\underline{\underline{\mathbf{D}}}\big|_{t+\Delta\,t} = \lambda\big|_{t+\Delta\,t}\,\underline{I} \otimes \underline{I} + 2\,\mu\big|_{t+\Delta\,t}\,\underline{\underline{\mathbf{I}}}$.

3.4.1.2.1 Formal derivation of $\dfrac{\partial\Delta\,\underline{\varepsilon}^{\mathrm{el}}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{to}}}$ The Implicit System (17) can be differentiated with respect to the total strain by virtue of the implicit function theorem:

$$\frac{\mathrm{d}\vec{F}}{\mathrm{d}\Delta\,\underline{\varepsilon}^{\mathrm{to}}} = \frac{\partial\vec{F}}{\partial\Delta\,\vec{Y}}\,\frac{\mathrm{d}\Delta\,\vec{Y}}{\mathrm{d}\Delta\,\underline{\varepsilon}^{\mathrm{to}}} + \frac{\partial\vec{F}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{to}}} = 0$$

The derivative $\dfrac{\partial\vec{F}}{\partial\Delta\,\vec{Y}}$ is the jacobian J of the implicit system, so the previous equation can be rewritten as:

$$\frac{\mathrm{d}\Delta\,\vec{Y}}{\mathrm{d}\Delta\,\underline{\varepsilon}^{\mathrm{to}}} = -J^{-1}\,\frac{\partial\vec{F}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{to}}}$$

Under the assumptions of the `StandardElasticity` framework, the derivative $\dfrac{\partial\vec{F}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{to}}}$ has the form:

$$\frac{\partial\vec{F}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{to}}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{21}$$

Thanks to Equation (21), one sees that the product $-J^{-1}\,\dfrac{\partial\vec{F}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{to}}}$ only contains the the 6 first columns of the J^{-1}. This allows identifying $\dfrac{\mathrm{d}\Delta\,\underline{\varepsilon}^{\mathrm{el}}}{\mathrm{d}\Delta\,\underline{\varepsilon}^{\mathrm{to}}}$ with the 6×6 upper-left submatrix of J^{-1}. Let $J^{-1}_{\underline{\varepsilon}^{\mathrm{el}}}$ this submatrix:

$$J^{-1}_{\underline{\varepsilon}^{\mathrm{el}}} = \frac{\mathrm{d}\Delta\,\underline{\varepsilon}^{\mathrm{el}}}{\mathrm{d}\Delta\,\underline{\varepsilon}^{\mathrm{to}}}$$

Thanks to Equation (20), the consistent tangent operator finally reads:

$$\frac{\mathrm{d}\underline{\sigma}}{\mathrm{d}\Delta\,\underline{\varepsilon}^{\mathrm{to}}} = \underline{\underline{\mathbf{D}}}\big|_{t+\Delta\,t} \cdot J^{-1}_{\underline{\varepsilon}^{\mathrm{el}}} \tag{22}$$

Extension to more complex behaviours
The derivation of the consistent tangent operator presented here can be extended to more complex situations, when the assumptions of the `StandardElasticity` framework don't hold. See the documentation of `MFront`' `Implicit` DSLs for details: https://thelfer.github.io/tfel/web/implicit-dsl.html

3.4.2 Numerical computation of the jacobian

Most nonlinear algorithms delivered with `MFront` can use a numerical jacobian. For example, the `Newton-Raphson` algorithm with a numerical jacobian can be used using the following line:

```
@Algorithm NewtonRaphson_NumericalJacobian;
```

The jacobian matrix is computed using a centered finite difference scheme. This may have a significant impact on the numerical performances of the implementation.

In the case of the behaviour of A. Assire, the number of state variables is, i.e. the size of $\vec{Y}$ vector is 32 in $3D$ (5 symmetric tensors with 6 components in $3D$ and 2 scalars). This means that the residual will be evaluated 65 times if a numerical jacobian is used:

- 1 for the computation of the residual.
- 2×32 for the computation of the numerical approximation of the jacobian matrix.

The perturbation value used to compute the numerical jacobian also have a strong numerical impact. By default, the perturbation value is chosen a one tenth of the default stopping value of the convergence criterion. We recommend to change this value using the `@PerturbationValueForNumericalJacobianComputation` keyword as follows:

```
@PerturbationValueForNumericalJacobianComputation 1.e-7;
```

Values between 10^{-8} and 10^{-6} usually give satisfactory results.

3.4.3 Numerical computation of blocks of the jacobian

For algorithms based on the analytical definition of the jacobian, `MFront` can evaluate only some blocks numerically using the `@NumericallyComputedJacobianBlocks` as follows:

```
@NumericallyComputedJacobianBlocks {dfpp_ddeel,dfeel_ddeel};
```

All the other blocks shall be computed analytically.

3.4.4 Numerical verification of the jacobian

The `@CompareToNumericalJacobian` keyword can be used to request a comparison of the jacobian blocks computed analytically to a numerical approximation.

If the norm of the difference between the analytical and numerical value is beyond a given threshold, `MFront` displays on the terminal the value of the analytical block, the values of the numerical block and the value of their difference.

The default value of this threshold can be changed using the `@JacobianComparisonCriterion` value.

3.4.5 A step by step implementation of the jacobian

Combining the ability to compute numerically certain blocks of the jacobian matrix and the comparison of the analytical jacobian matrix to a numerical approximation allows to gradually implement an analytical jacobian.

First, one may declare all the jacobian blocks to be computed numerically. Then, one may start by the most obvious blocks and remove them from the list of blocks computed numerically.

Then, one may consider a given block, try to implement it, remove it from the list of the blocks computed numerically and see of the analytical value matches the numerical approximation. If the analytical value is correct, one may consider another code block. If not, one must correct the implementation of this term.

4 Implementation of the viscoplastic part

The implementation can be decomposed into various steps of growing complexity:

- As a first step, one may consider a simple Norton behaviour without isotropic and kinematic hardenings with constant normalisation factor K_{vp} and exponent E_{vp}. This is done in Section 4.1.
- As a second step, one may consider to add isotropic hardening.
- As a third step, one may consider to add kinematic hardenings.
- As a fourth step, one may implement the whole viscoplastic part by adding the dependency of the normalisation factor K_{vp} and exponent E_{vp} to the equivalent viscoplastic strain p_{vp}.

One shall always consider comparing the prediction of its implementation to the results of a reference one. In many cases, the `StandardElastoViscoplasticity` brick may provide such a reference implementation.

4.1 Simple Norton behaviour

4.1.1 Constitutive equations

It is always worth rewritting the constitutive equations, even in simple case.

The state variables are the elastic strain $\underline{\varepsilon}^{\mathrm{el}}$ and the equivalent viscoplastic strain p_{vp}.

The constitutive equation driving the evolution of the elastic strain is derived from the split of the strain:

$$\underline{\dot{\varepsilon}}^{\mathrm{el}} = \underline{\dot{\varepsilon}}^{\mathrm{to}} - \dot{p}_{\mathrm{vp}}\, \underline{n}_{\mathrm{vp}} \tag{23}$$

The constitutive equation driving the evolution of the equivalent viscoplastic strain is:

$$\dot{p}_{\mathrm{vp}} = \left(\dfrac{\sigma}{K_{\mathrm{vp}}}\right)^{E_{\mathrm{vp}}} \tag{24}$$

where the stress $\underline{\sigma}$ is related to the elastic strain by the Hooke Law (2).

4.1.2 Implementation using the `StandardElastoViscoplasticity` brick

Listing 1 Implementation of the Norton behaviour using the StandardElastoViscoplasticity brick.

```
@DSL Implicit;
@Behaviour ImplicitNorton2;
@Author Thomas Helfer;
@Date 04 / 10 / 2021;

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

@Brick StandardElastoViscoPlasticity{
  stress_potential : "Hooke" {young_modulus : 150e9, poisson_ratio : 0.3},
  inelastic_flow : "Norton" {
    criterion : "Mises",
    K : 100e6,  // Stress normalisation factor
    n : 4.5     // Norton exponent
  }
};
```

As illustrated by Listing 1, the implementation of the Norton behaviour using the `StandardElastoViscoplasticity` brick is straightforward. It can be decomposed in three parts:

1. The meta-data which declare:

- the type of DSL used,

- the name of the behaviour,
- the author of the implementation
- and the date. Those meta-data may also contain a description using the `@Description` keyword.

2. The numerical parameters, including:

- the choice of the non linear solver,
- the default value of the criterion value,
- the default value of the implicit parameter θ.

3. The call to the `StandardElastoViscoPlasticity` brick. The reader can refer to the documentation of the brick for an in-depth explantations.

A few remark can be made at this stage:

- The Newton-Raphson solver being the default, its declaration is optional.
- The declaration of the elastic properties in the brick is optional. They can also be declared using the `@ElasticMaterialProperties` keyword. If they are not declared, two material properties with the external names `YoungModulus` and `PoissonRatio` are automatically declared because the material is isotropic (the default).

  Since the elastic properties are defined using values, two parameters named with the external names `YoungModulus` and `PoissonRatio` are automatically declared. Rather than using a value, a formula or an external `MFront` file could have been used.
- No state variable is explicitly declared. The `Implicit` DSL automatically declares the elastic strain as the first state variable. The name of this variable inside `MFront` is eel and its external name is `ElasticStrain`. The `StandardElastoViscoPlasticityBrick` automatically declares the equivalent plastic strain with the `EquivalentViscoplasticStrain` external name.
- The stress normalisation factor `K` and value of the Norton exponent `n` are declared using numerical values. This automatically declares two parameters named respectively `K` and `E` (`n` is used internally to define the normal). Implicitly, the Norton coefficient `A` is also defined to 1. Rather than a value, a formula or an external `MFront` file could also have been used.
- The consistent tangent operator is computed automatically using the technique described in Section 3.4.1.2.

External names
External names are an important concept in `MFront`.
The external name of a variable is the name of this variable from the calling solver.
The user may associate an external name to a variable in two ways:

- using the `setGlossaryName` method. In this case, the name passed to the `setGlossaryName` method must be declared in the `TFEL`' glossary
- using the `setEntryName` method. In this case, any name can be used.

An example of the declaration of the Norton coefficient as a function of the temperature
The following listing uses formulae to define the viscoplastic properties using previously declared parameters:

```
//! activation temperature for the Norton behaviour
@Parameter temperature Ta = 273.15;
Ta.setEntryName("NortonActivationTemperature");
//! activation temperature for the Norton behaviour
@Parameter temperature A0 = 2e-36;
A0.setEntryName("NortonCoeffcient");

@Brick StandardElastoViscoPlasticity{
  stress_potential : "Hooke" {young_modulus : 150e9, poisson_ratio : 0.3},
  inelastic_flow : "Norton" {
    criterion : "Mises",
    A : "A0 * exp(-Ta / T)", //
    n : 4.5,                 //
    K : 1
  }
};
```

Note the `doxygen` like comments above the declaration of the parameters `Ta` and `A0` that can be used by the `mfront-doc` and `mfront-query` tools (See Appendix A).

4.1.2.1 Compilation

The compilation of the `MFront` file in a shared library can be done as follows:

```
$ mfront --obuild --interface=aster ImplicitNorton2.mfront
Treating target : all
The following library has been built :
- libAsterBehaviour.so :  asterimplicitnorton2
```

4.1.2.2 First test using the `MTest`

`MTest` allows to study the behaviour on a single material point by imposing for a given component either the strain history or the stress history.

Listing 2 Uniaxial tensile test on the simple Norton behaviour implemented using the StandardElastoViscoplasticity brick.

```
@Author Thomas Helfer;
@Date   17/09/2021;

@ModellingHypothesis "Tridimensional";
@Behaviour<aster> "src/libAsterBehaviour.so" "asterimplicitnorton2";
@ExternalStateVariable "Temperature" 293.15;
@ImposedStrain "EXX" {0 : 0, 1 : 1e-2};
@Times{0, 1 in 10};
```

4.1.2.3 Adding support for plane stress hypotheses

By default, plane stress hypotheses, i.e. the standard plane stress hypothesis and the axisymmetrical generalized plane stress hypothesis used by `Cyrano` fuel performance code, are not supported.

Support for those hypotheses is be added automatically by the `StandardElastoViscoPlasticity` brick following the technique described in Section 3.4.1.1 if we explicitly declare that those modelling hypotheses shall be supported. The most straightforward way to do this is to use the following declaration (generally at the beginning of the `MFront` implementation, by convention):

```
@ModellingHypotheses {".+"};
```

4.1.2.4 Integration in `code_aster` with the `mtest` python module https://thelfer.github.io/tfel/web/mtest-python.html

```
>>> b = mtest.Behaviour('src/libAsterBehaviours.so', 'asterimplicitnorton', 'Tridimensional')
>>> print(b.getInternalStateVariablesNames())
['ElasticStrain']
>>> b.expandInternalStateVariablesNames()
['ElasticStrainXX', 'ElasticStrainYY', 'ElasticStrainZZ', 'ElasticStrainXY', 'ElasticStrainXZ',
 'ElasticStrainYZ']
```

4.1.3 Implementation with the `StandardElasticity` brick

As the whole plastic-viscoplastic behaviour of Aimery Assire can't be implemented using the `StandardElastoViscoPlasticity`, one shall explicitly implement the computation of the residual and the computation of the jacobian matrix. By chance, this behaviour respects the assumpations of the `StandardElasticity` brick, described in Section 3.4.1), which considerably eases its implementation.

We now show how to use this brick to implement the simple Norton behaviour. We first start using a numerical jacobian and then implement the computation of this jacobian.

4.1.3.1 Variables automatically declared by the `Implicit` DSL

At this stage, it is worth detailling the variables automatically declared by the `Implicit` DSL:

- As every DSL defining a strain based behaviour, the total strain $\underline{\varepsilon}^{\mathrm{to}}$ and its increment $\Delta\,\underline{\varepsilon}^{\mathrm{to}}$ are automatically declared as the `eto` and `deto` variables respectively.
- As every DSL defining a strain based behaviour, the variable `sig` is also automatically declared.
  - This variable is initialized with the value of the stress at the beginning of the time step $\left.\underline{\sigma}\right|_{t}$. This value can be used in the `@InitLocalVariables` and `@Predictor` code blocks.
  - After the behaviour integration, this variable must contain the value of the stress at the end of the time step $\left.\underline{\sigma}\right|_{t+\Delta\,t}$. This is generally done in the `@ComputeFinalStress` code block.
  - During the Newton algorithm, this variable is generally used to store the current estimate of the stress at the middle of the time step $\left.\underline{\sigma}\right|_{t+\theta\,\Delta\,t}$. This estimate is generally computed in the `ComputeStress` code block.
- As in every DSL, the temperature is defined as an external state variable with the external name `Temperature`:
  - The variable `T` contains the value of the temperature at the beginning of the time step.
  - The variable `dT` contains the value of the increment of the temperature over the time step.
- The convergence threshold $\varepsilon_{\mathrm{NR}}$ is associated with the `epsilon` parameter. The default value of the parameter can be changed using the `@Epsilon` keyword.
- The implicit parameter θ is associated with the `theta` parameter. The default value of the parameter can be changed using the `@Theta` keyword.
- As in every DSL, the tangent operator is associated with a variable named `Dt`. This variable is only accessible in the `@TangentOperator` and `@PredictionOperator` code blocks.
- As in every DSL, the variable `D` is reserved for the stiffness matrix. This variable is only defined if one of the keywords `@ComputeStiffnessTensor` or the `@RequireStiffnessTensor` is used.

Variables in `MFront`

A variable in `MFront` is characterised by:

- its category (i.e. its role). The main categories of variables are:
  - state variable (see `@StateVariable` keyword),
  - auxiliary state variable (see `@AuxiliaryStateVariable` keyword),
  - local variable (see `@LocalVariable` keyword),
  - parameter (see `@Parameter` keyword),
  - integration variable (see `@IntegrationVariable` keyword),
  - material property (see `@MaterialProperty` keyword).
- its type.
- its name inside `MFront`.
- its symbolic name. This is used for unicode support.
- its external name.

4.1.3.2 Implementation using a numerical jacobian

Following Section 3.3, the residual of the implicit system to be solved can be obtained by discretizing Equations (23) and (24).

This residual is readily given by:

$$\left\{\begin{aligned} f_{\underline{\varepsilon}^{\mathrm{el}}} &= \Delta\,\underline{\varepsilon}^{\mathrm{el}} - \Delta\,\underline{\varepsilon}^{\mathrm{to}} + \Delta\,p_{\mathrm{vp}}\,\underline{n}_{\mathrm{vp}}\Big|_{t+\theta\,\Delta\,t} \\ f_p &= \Delta\,p - \Delta\,t\left(\dfrac{\sigma\big|_{t+\theta\,\Delta\,t}}{K_{\mathrm{vp}}}\right)^{E_{\mathrm{vp}}} \end{aligned}\right.$$

Listing 3 Implementation of the simple Norton behaviour implemented using the StandardElasticity brick with a numerical jacobian.

```
@DSL Implicit;
@Behaviour ImplicitNorton_NumericalJacobian;
@Author Thomas Helfer;
@Date 17 / 09 / 2021;

@Algorithm NewtonRaphson_NumericalJacobian;
@PerturbationValueForNumericalJacobianComputation 1.e-8;
@Epsilon 1.e-14;
@Theta 1;

@Brick StandardElasticity{young_modulus : 150e9, poisson_ratio : 0.3};

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

@Parameter stress Kv = 100e6;
Kv.setEntryName("NortonNormalisationFactor");
@Parameter strainrate de0 = 1;
de0.setEntryName("NortonReferenceStrainRate");
@Parameter real Evp = 4.5;
Evp.setEntryName("NortonExponent");

@Integrator {
  constexpr auto eeps = 1.e-14;
  const auto seps = young * eeps;
  const auto seq = sigmaeq(sig);
  const auto iseq = 1 / (max(seq, seps));
  const auto n = 3 * deviator(sig) * (iseq / 2);
  const auto vp = de0 * pow(seq / Kv, Evp);
  // implicit equation associated with elasticity
  feel += dpvp * n;
  // implicit equation associated with the equivalent plastic strain
  fpvp -= dt * vp;
}
```

The implementation of the residual must be done in the `@Integrator` code block. The whole implementation of the behaviour is reported on Listing 3.

A few remarks can be made:

- As for the use of the `StandardElastoViscoplasticity` brick, the elastic properties can be defined by the options `young_modulus` and `poisson_ratio` options passed to the `StandardElasticity` brick.
- The normalisation factor K_{vp} and exponent E_{vp} are automatically declared as parameters named respectively `Kvp` and `Evp`. The `setEntryName` method is used to associate the external names `NortonNormalisationFactor` and `NortonExponent` to those parameters.

- The `StandardElasticity` automatically computes the current estimation of stress $\underline{\sigma}\left.\right|_{t+\theta\,\Delta\,t}$ in the variable `sig` before each evalution of the residual in the `@Integrator` code block. If the `StandardElasticity` brick is not used, the user may use the `@ComputeStress` code block to perform this computation.
- After convergence of the local Newton algorithm, the `StandardElasticity` automatically computes the stress $\underline{\sigma}\left.\right|_{t+\Delta\,t}$ at the end of the step. If the `StandardElasticity` brick is not used, the user may use the `@ComputeFinalStress` code block to perform this computation, although this is seldom required for strain based behaviour as the code in `@ComputeFinalStress` is generally automatically deduced from the code defined by `@ComputeStress`.
- The computation of `feel` takes into account the fact that this variable has been initialized to $\Delta\,\underline{\varepsilon}^{\mathrm{el}} - \Delta\,\underline{\varepsilon}^{\mathrm{to}}$ by the `StandardElasticity` brick.
- The computation of `fpvp` takes into account the fact that this variable has been initialized to $\Delta\,p_{\mathrm{vp}}$ by the `Implicit` DSL as explained in Section 3.3.1.

Material properties vs parameters
In most MFront tutorials, we usually prefer to use parameters, which are declared using the `@Parameter` keyword.
Contrary to material properties, declared using the `@MaterialProperty` keyword, parameters have default values. Parameters are thus useful to build behaviours specific to one particular material. Such behaviours are self-contained, which considerably eases building a strict material knowledge management policy.
Parameters are also stored in a global structure. This means that parameters are uniform. In some codes, material properties may be defined on a per integration point basis.

4.1.3.3 Using quantities

A quantity in the `TFEL/Math` library are a mathematical object associated with an unit type. Quantities are meant to allow dimensional analysis and prevent illegal operations.

Quantities are enable by the `@UseQt` as follows:

```
@UseQt true;
```

4.1.3.4 Elimination of the implicit equation associated with the equivalent plastic strain

The increment $\Delta\,p_{\mathrm{vp}}$ can easily be eliminated from the implicit system since there is no isotropic hardening. However, its value is generally required for post-processings.

Auxiliary state variables are saved from one time step to the other but are not part of the implicit system: there is no implicit equations associated with those variables and their increments are not declared.

Auxiliary state variables are generally updated after the convergence of the implicit scheme in the `@UpdateAuxiliaryStateVariables` code block. At this stage of the behaviour integration (i.e. in the `@UpdateAuxiliaryStateVariables` code block), the stress have been updated to their values at the end of the time step and the state variables have been updated. Note that the gradients (the total strain for strain based behaviours) and the external state variables are never updated.

Listing 4 shows how to modify Implementation 3 to eliminate the implicit equation associated with the equivalent viscoplastic strain and updating it in the `@UpdateAuxiliaryStateVariables` code block.

4.1.3.5 Implementation using a analytical jacobian

The jacobian matrix only contains the $\dfrac{\partial f_{\underline{\varepsilon}^{\mathrm{el}}}}{\partial \Delta\,\underline{\varepsilon}^{\mathrm{el}}}$ derivative which can be computed as follows:

$$\dfrac{\partial f_{\underline{\varepsilon}^{\mathrm{el}}}}{\partial \Delta\,\underline{\varepsilon}^{\mathrm{el}}} = \underline{\underline{\mathbf{I}}} - \Delta\,t\,\underline{n}\left.\right|_{t+\theta\,\Delta\,t} \otimes \dfrac{\partial v_{p_{\mathrm{vp}}}}{\partial \Delta\,\underline{\varepsilon}^{\mathrm{el}}} - \Delta\,t\,v_{p_{\mathrm{vp}}}\,\dfrac{\partial\,\underline{n}_{\mathrm{vp}}\left.\right|_{t+\theta\,\Delta\,t}}{\partial \Delta\,\underline{\varepsilon}^{\mathrm{el}}}$$

with $v_{p_{\mathrm{vp}}} = \left(\dfrac{\sigma\left.\right|_{t+\theta\,\Delta\,t}}{K_{\mathrm{vp}}}\right)^{E_{\mathrm{vp}}}$.

$\dfrac{\partial v_{p_{\mathrm{vp}}}}{\partial \Delta\,\underline{\varepsilon}^{\mathrm{el}}}$ can be computed by the chain rule as follows:

$$\frac{\partial v_{p_{\mathrm{vp}}}}{\partial \Delta\,\underline{\varepsilon}^{\mathrm{el}}} = \underbrace{\frac{\partial v_{p_{\mathrm{vp}}}}{\partial\,\sigma_{\mathrm{eq}}\big|_{t+\theta\,\Delta\,t}}}_{\frac{E_{\mathrm{vp}}}{K_{\mathrm{vp}}}\left(\frac{\sigma|_{t+\theta\,\Delta\,t}}{K_{\mathrm{vp}}}\right)^{E_{\mathrm{vp}}-1}} : \underbrace{\frac{\partial\,\sigma_{\mathrm{eq}}\big|_{t+\theta\,\Delta\,t}}{\partial\,\underline{\sigma}\big|_{t+\theta\,\Delta\,t}}}_{\underline{n}_{\mathrm{vp}}\big|_{t+\theta\,\Delta\,t}} : \underbrace{\frac{\partial\,\underline{\sigma}\big|_{t+\theta\,\Delta\,t}}{\partial\,\underline{\varepsilon}^{\mathrm{el}}\big|_{t+\theta\,\Delta\,t}}}_{\lambda|_{t+\theta\,\Delta\,t}\,\underline{I}\otimes\underline{I}+2\,\mu|_{t+\theta\,\Delta\,t}\,\underline{\underline{\mathbf{I}}}} : \underbrace{\frac{\partial\,\underline{\varepsilon}^{\mathrm{el}}\big|_{t+\theta\,\Delta\,t}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{el}}}}_{\theta\,\underline{\underline{\mathbf{I}}}}$$

Finally, using the fact that $\underline{n}_{\mathrm{vp}}\big|_{t+\theta\,\Delta\,t}$ is a deviatoric tensor to eliminate the term proportional to $\lambda|_{t+\theta\,\Delta\,t}$, $\dfrac{\partial v_{p_{\mathrm{vp}}}}{\partial \Delta\,\underline{\varepsilon}^{\mathrm{el}}}$ is given by:

$$\frac{\partial v_{p_{\mathrm{vp}}}}{\partial \Delta\,\underline{\varepsilon}^{\mathrm{el}}} = 2\,\mu|_{t+\theta\,\Delta\,t}\,\theta\left[\frac{E_{\mathrm{vp}}}{K_{\mathrm{vp}}}\left(\frac{\sigma|_{t+\theta\,\Delta\,t}}{K_{\mathrm{vp}}}\right)^{E_{\mathrm{vp}}-1}\right]\,\underline{n}_{\mathrm{vp}}\big|_{t+\theta\,\Delta\,t}$$

$\dfrac{\partial\,\underline{n}_{\mathrm{vp}}\big|_{t+\theta\,\Delta\,t}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{el}}}$ can also be computed by chain rule in a similar way:

$$\frac{\partial\,\underline{n}_{\mathrm{vp}}\big|_{t+\theta\,\Delta\,t}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{el}}} = \underbrace{\frac{\partial\,\underline{n}_{\mathrm{vp}}\big|_{t+\theta\,\Delta\,t}}{\partial\,\underline{\sigma}\big|_{t+\theta\,\Delta\,t}}}_{\frac{1}{\sigma_{\mathrm{eq}}|_{t+\theta\,\Delta\,t}}\left(\underline{\underline{\mathbf{M}}}-\underline{n}_{\mathrm{vp}}|_{t+\theta\,\Delta\,t}\otimes\underline{n}_{\mathrm{vp}}|_{t+\theta\,\Delta\,t}\right)} : \underbrace{\frac{\partial\,\underline{\sigma}\big|_{t+\theta\,\Delta\,t}}{\partial\,\underline{\varepsilon}^{\mathrm{el}}\big|_{t+\theta\,\Delta\,t}}}_{\lambda|_{t+\theta\,\Delta\,t}\,\underline{I}\otimes\underline{I}+2\,\mu|_{t+\theta\,\Delta\,t}\,\underline{\underline{\mathbf{I}}}} : \underbrace{\frac{\partial\,\underline{\varepsilon}^{\mathrm{el}}\big|_{t+\theta\,\Delta\,t}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{el}}}}_{\theta\,\underline{\underline{\mathbf{I}}}}$$

where we used Equation (3).

Finally,

$$\frac{\partial\,\underline{n}_{\mathrm{vp}}\big|_{t+\theta\,\Delta\,t}}{\partial\Delta\,\underline{\varepsilon}^{\mathrm{el}}} = \frac{2\,\mu|_{t+\theta\,\Delta\,t}\,\theta}{\sigma_{\mathrm{eq}}\big|_{t+\theta\,\Delta\,t}}\left(\underline{\underline{\mathbf{M}}}-\underline{n}_{\mathrm{vp}}\big|_{t+\theta\,\Delta\,t}\otimes\underline{n}_{\mathrm{vp}}\big|_{t+\theta\,\Delta\,t}\right)$$

The implementation of the Norton behaviour with an analytical jacobian is reported in Listing 5. If we compare Listing 5 and Listing 4, we may notice that:

- The name of the algorithm was changed from `NewtonRaphson_NumericalJacobian` to `NewtonRaphson`.
- The definition of the default value for the perturbation value used for the computation of the numerical jacobian was removed.
- The computation of the jacobian has been added. As discussed in Section 3.3.4, we took into account the fact that `dfeel_ddeel` has automatically been initialized to the identity before the call to the `@Integrator` code block.‘

Priority of operator ^
In the `TFEL/Math` library, the tensorial product is associated to the ^ operator. In `C++`, this operator has a low priority contrary to its mathematical counterpart. Use of parenthesis is strongly adviced.

4.2 Viscoplastic behaviour with isotropic hardening

4.2.1 Implementation using the `StandardElastoViscoplasticity` brick

```
@DSL Implicit;
@Behaviour ImplicitNorton;
@Author Thomas Helfer;
@Date 17 / 06 / 2022;

@Algorithm NewtonRaphson;
@Epsilon 1.e-14;
```

```
@Theta 1;

@Parameter stress Kv = 100e6;
@Parameter real nv = 4.5;
@Parameter stress Rv0 = 150e6;
@Parameter stress Qv = 200e6;
@Parameter stress bv = 10;

@Brick StandardElastoViscoPlasticity{
  stress_potential : "Hooke" {young_modulus : 150e9, poisson_ratio : 0.3},
  inelastic_flow : "Norton" {
    criterion : "Mises",
    isotropic_hardening : "Voce" {R0 : "Rv0", Rinf : "Rv0 + Qv", b : "bv"},
    K : "Kv",
    n : "nv"
  }
};
```

4.2.2 Implementation using the `StandardElasticity` brick

```
@DSL Implicit;
@Behaviour ImplicitNorton2;
@Author Thomas Helfer;
@Date 17 / 09 / 2021;

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

@Brick StandardElasticity{young_modulus : 150e9, poisson_ratio : 0.3};

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

@Parameter stress Kv = 100e6;
@Parameter real nv = 4.5;
@Parameter stress Rv0 = 150e6;
@Parameter stress Qv = 200e6;
@Parameter stress bv = 10;

@LocalVariable bool viscoplastic_loading;

@InitLocalVariables {
  const auto sig_tr = computeElasticPrediction();
  const auto seq_tr = sigmaeq(sig_tr);
  const auto Rv = Rv0 + Qv * (1 - exp(-bv * p));
  viscoplastic_loading = seq_tr - Rv > 0;
}

@Integrator {
  if (!viscoplastic_loading) {
    return true;
  }
  constexpr auto eeps = 1.e-14;
  const auto seps = young * eeps;
  const auto seq = sigmaeq(sig);
  const auto iseq = 1 / (max(seq, seps));
  const auto n = 3 * deviator(sig) * (iseq / 2);
  // equivalent stress
  const auto exp_bv = exp(-bv * (p + theta * dp));
  const auto Rv = Rv0 + Qv * (1 - exp_bv);
  const auto seq_e = seq - Rv;
  const auto iseq_e = 1 / (max(seq_e, seps));
  const auto vp = pow(max(seq_e, stress(0)) / Kv, nv);
  // implicit equation associated with elasticity
  feel += dp * n;
  // implicit equation associated with the equivalent plastic strain
  fp -= dt * vp;
  // jacobian blocks
  const auto dn_ddeel = 2 * mu * theta * iseq * (Stensor4::M() - (n ^ n));
  dfeel_ddeel += dp * dn_ddeel;
  dfeel_ddp = n;
  //
  const auto dvp_dseq_e = nv * vp * iseq_e;
  const auto dvp_ddeel = 2 * mu * theta * dvp_dseq_e * n;
  const auto dRv_ddp = theta * Qv * bv * exp_bv;
  const auto dseq_e_ddp = -dRv_ddp;
  const auto dvp_ddp = dvp_dseq_e * dseq_e_ddp;
  dfp_ddp -= dt * dvp_ddp;
  dfp_ddeel = -dt * dvp_ddeel;
}
```

4.3 Viscoplastic behaviour with isotropic hardening and variable coefficients

5 Implementation of the plastic part

6 Implementation of the plastic-viscoplastic behaviour

A Some useful tools

A.1 The `mfront-doc` tool

The `mfront-doc` tool can be used to generate a markdown file describing the behaviour as follows:

```
$ mfront-doc ImplicitNorton2.mfront -o ImplicitNorton2.md
```

The `ImplicitNorton2.md` markdown file can be parsed by `pandoc` to a generate a PDF file, a word file or an web page.

A.2 The `mfront-query` tool

`mfront-query` is a powerful tool to query information about a behaviour. For example, the following query retrieves the list of the parameters defined by the behaviour:

```
$ mfront-query --parameters ImplicitNorton2.mfront
- ε: value used to stop the iteration of the implicit algorithm
- θ: theta value used by the implicit scheme
- YoungModulus (young): The Young modulus of an isotropic material
- PoissonRatio (nu): The Poisson ratio of an isotropic material
- RelativeValueForTheEquivalentStressLowerBoundDefinition
  (relative_value_for_the_equivalent_stress_lower_bound): Relative value
  used to define a lower bound for the equivalent stress. For isotropic
  parameters, this lower bound will be equal to this value multiplied by
  the Young modulus. For orthotropic materials, this lower bound will be
  this value multiplied by the first component of the stiffness tensor.
- K
- E
- A
- minimal_time_step_scaling_factor: minimal value for the time step
  scaling factor
- maximal_time_step_scaling_factor: maximal value for the time step
  scaling factor
- numerical_jacobian_epsilon: perturbation value used to compute a
  numerical approximation of the jacobian
- iterMax: maximum number of iterations allowed
```

The list of available queries can be retrieved using the `--help-behaviour-queries-list` command line option as follows:

```
$ mfront-query --help-behaviour-queries-list
Usage: mfront-query [options] [files]

Available options are :
--attribute-type                : display an attribute type
--attribute-value               : display an attribute value
--attributes                    : show the list of attributes of the behaviour description
--author                        : show the author name
--auxiliary-state-variables     : show the auxiliary state variables properties for the selected
  modelling hypothesis
--behaviour                     : specify a behaviour identifier (can be a regular expression)
```

References

1. Assire, Aimery. Amorçage et propagation de la fissuration dans les jonctions soudées à haute température. PhD thesis. 2000. Available from: http://www.theses.fr/2000AIX22109

2. Doghri, Issam. Mechanics of deformable solids: Linear, nonlinear, analytical, and computational aspects. Berlin; New York : Springer, 2000. ISBN 3540669604 9783540669609 3642086292 9783642086298.

3. Neto, Djordje, Perić and Owen, D. R. J. Computational methods for plasticity: Theory and applications. Chichester, West Sussex, UK : Wiley, 2008. ISBN 978-0-470-69452-7 0-470-69452-1 978-0-470-69462-6 0-470-69462-9.

4. Besson, Jacques, Cailletaud, Georges, Chaboche, Jean-Louis, Forest, Samuel and Bletry, Marc. Non-linear mechanics of materials. 2010 ed. Dordrecht : Springer, 2009. ISBN 978-90-481-3355-0.

5. Forest, Samuel, Amestoy, Michel, Damamme, Gilles, Kruch, Serge, Maurel, Vincent and Mazière, Matthieu. Mécanique des milieux continus. 2013. École des Mines de Paris. Available from: http://mms2.ensmp.fr/mmc_paris/mmc_paris.php

6. Besson, J. and Desmorat, D. Numerical implementation of constitutive models. In : BESSON, J. [ed.], *Local approach to fracture*. École des Mines de Paris - les presses, 2004.

7. Lemaitre, Jean and Desmorat, Rodrigue. Engineering damage mechanics: Ductile, creep, fatigue and brittle failures. Berlin Heidelberg : Springer-Verlag, 2005. ISBN 978-3-540-21503-5. Available from: //www.springer.com/us/book/9783540215035

8. Helfer, Thomas, Bleyer, Jeremy, Frondelius, Tero, Yashchuk, Ivan, Nagel, Thomas and Naumov, Dmitri. The 'MFrontGenericInterfaceSupport' project. *Journal of Open Source Software*. 2020. Vol. 5, no. 48, p. 2003. DOI 10.21105/joss.02003. Available from: https://doi.org/10.21105/joss.02003
Publisher: The Open Journal

9. Simo, J. C. and Taylor, R. L. Consistent tangent operators for rate-independent elastoplasticity. *Computer Methods in Applied Mechanics and Engineering*. February 1985. Vol. 48, no. 1, p. 101–118. DOI 10.1016/0045-7825(85)90070-2. Available from: http://www.sciencedirect.com/science/article/pii/0045782585900702

10. Armstrong, P. J. and Frederick, C. O. RD/BfN 731: A mathematical representation of the multiaxial Bauschinger effect. Central Electricity Generating Board, 1966.

Listing 4 Implementation of the simple Norton behaviour implemented using the StandardElasticity brick with a numerical jacoabian with the equivalent viscoplastic strain as an auxiliary state variable.

```
@DSL Implicit;
@Behaviour ImplicitNorton_NumericalJacobian2;
@Author Thomas Helfer;
@Date 17 / 09 / 2021;
@UseQt true;

@Algorithm NewtonRaphson_NumericalJacobian;
@PerturbationValueForNumericalJacobianComputation 1.e-8;
@Epsilon 1.e-14;
@Theta 1;

@Brick StandardElasticity{young_modulus : 150e9, poisson_ratio : 0.3};

@AuxiliaryStateVariable strain pvp;
pvp.setGlossaryName("EquivalentViscoplasticStrain");

@Parameter stress Kv = 100e6;
Kv.setEntryName("NortonNormalisationFactor");
@Parameter strainrate de0 = 1;
de0.setEntryName("NortonReferenceStrainRate");
@Parameter real Evp = 4.5;
Evp.setEntryName("NortonExponent");

@LocalVariable strainrate vp;

@Integrator {
  constexpr auto eeps = 1.e-14;
  const auto seps = young * eeps;
  const auto seq = sigmaeq(sig);
  const auto iseq = 1 / (max(seq, seps));
  const auto n = 3 * deviator(sig) * (iseq / 2);
  vp = de0 * pow(seq / Kv, Evp);
  // implicit equation associated with elasticity
  feel += dt * vp * n;
}

@UpdateAuxiliaryStateVariables{
  pvp += dt * vp;
}
```

Listing 5 Implementation of the simple Norton behaviour implemented using the StandardElasticity brick with a numerical jacoabian.

```
@DSL Implicit;
@Behaviour ImplicitNorton;
@Author Thomas Helfer;
@Date 17 / 09 / 2021;
@UseQt true;

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

@Brick StandardElasticity{young_modulus : 150e9, poisson_ratio : 0.3};

@AuxiliaryStateVariable strain pvp;
pvp.setGlossaryName("EquivalentViscoplasticStrain");

@Parameter stress Kv = 100e6;
Kv.setEntryName("NortonNormalisationFactor");
@Parameter strainrate de0 = 1;
de0.setEntryName("NortonReferenceStrainRate");
@Parameter real Evp = 4.5;
Evp.setEntryName("NortonExponent");

@LocalVariable strainrate vp;

@Integrator {
  constexpr auto eeps = 1.e-14;
  const auto seps = young * eeps;
  const auto seq = sigmaeq(sig);
  const auto iseq = 1 / (max(seq, seps));
  const auto n = 3 * deviator(sig) * (iseq / 2);
  vp = de0 * pow(seq / Kv, Evp);
  // implicit equation associated with elasticity
  feel += dt * vp * n;
  // jacobian blocks
  const auto dvp_dseq = Evp * vp * iseq;
  const auto dvp_ddeel = 2 * mu * theta * dvp_dseq * n;
  const auto dn_ddeel = 2 * mu * theta * iseq * (Stensor4::M() - (n ^ n));
  dfeel_ddeel += dt * (n ^ dvp_ddeel) + dt * vp * dn_ddeel;
}

@UpdateAuxiliaryStateVariables{
  pvp += dt * vp;
}
```