Analysis of a new integrator for finite element code for the calculation of fuel rod thermal-mechanical behaviour:

MFRONT

Julien Olagnon

Master of Science Thesis
School of Engineering Sciences
Department of Physics
Division of Nuclear Reactor Technology
Stockholm, Sweden, 2013
Abstract

This master thesis deals with the study of MFRONT. This is a new integrator for finite element code for the calculation of fuel rod thermal-mechanical behaviour.

The paper describes its utilization with classical finite element code and then with more formal programming languages. The creation of new law in MFRONT has been tested and successfully implemented. The paper shows in a precise way the first steps to create these laws and the tests which has been done.

The paper also discusses the utility of MFRONT and its advantages. The main point is that calculations are much faster with the use of this integrator. This integrator also allows to create precise library for the law and an easy use of them.
# CONTENTS

**INTRODUCTION:** ...................................................................................................................................................... 7

1. BACKGROUND OF THE THESIS ............................................................................................................................... 8

   1.1. DESCRIPTION OF THE NUCLEAR FUEL .................................................................................................................. 8

2. BEHAVIOUR OF THE NUCLEAR FUEL IN THE REACTOR ......................................................................................... 11

   2.1. THE IRRADIATION HISTORY ................................................................................................................................... 11

   2.2. THE PELLET BEHAVIOUR .................................................................................................................................... 11

   2.3. THE CLADDING BEHAVIOUR ............................................................................................................................. 12

   2.4. THE PELLET/CLADDING INTERACTION ............................................................................................................... 13

2.5. CALCULATION MODEL OF THERMO-MECHANICAL BEHAVIOUR OF THE NUCLEAR FUEL IN A SIMULATION CODE ......................................................................................................................... 13

   2.5.1. THE DIFFERENT KINDS OF CODES .................................................................................................................... 13

   2.5.2. STRUCTURE OF FUEL ROD CODES .................................................................................................................. 14

   2.5.3. HEAT EQUATION ............................................................................................................................................. 14

   2.5.4. MECHANICAL EQUATION ................................................................................................................................ 14

3. MFRONT ........................................................................................................................................................................ 16

   3.1. GENERAL PRESENTATION .................................................................................................................................... 16

   3.2. MFRONT FILE ..................................................................................................................................................... 16

   3.2.1. MATERIAL PROPERTY ..................................................................................................................................... 16

   3.2.2. BEHAVIOUR LAW .......................................................................................................................................... 18

   3.3. MFRONT WITH CASTEM .................................................................................................................................... 20

   3.3.1. MATERIAL PROPERTIES .................................................................................................................................. 20

   3.3.2. BEHAVIOUR LAW .......................................................................................................................................... 21

   3.4. MFRONT WITH GALILEO .................................................................................................................................... 21

   3.5. MFRONT WITH ALCYONE .................................................................................................................................... 22

4. VALIDATION OF MFRONT WITH CASTEM .................................................................................................................. 23

   4.1. MATERIAL LAW .................................................................................................................................................. 23

   4.2. BEHAVIOUR LAW .............................................................................................................................................. 23

   4.2.1. MFRONT FILE ............................................................................................................................................. 23

   4.2.2. TESTS ............................................................................................................................................................ 25

4.3. CONCLUSION .......................................................................................................................................................... 34

5. APPLICATIONS OF AREVA LAWS WITH MFRONT .................................................................................................... 35

   5.1. MATERIAL PROPERTY: YOUNG MODULUS ............................................................................................................. 35

   5.2. BEHAVIOUR LAW ............................................................................................................................................... 36

   5.2.1. CLADDING CREEP LAW ................................................................................................................................. 36

   5.2.2. VALIDATION OF THE LAW ............................................................................................................................. 37
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.3.</td>
<td>OTHER CLADDING LAWS</td>
</tr>
<tr>
<td>5.3.</td>
<td>CONCLUSION ABOUT MFRONT</td>
</tr>
<tr>
<td>6.</td>
<td>MFRONT LAWS IN GALILEO</td>
</tr>
<tr>
<td>6.1.</td>
<td>MFRONT IN FORTRAN</td>
</tr>
<tr>
<td>6.1.1.</td>
<td>MATERIAL PROPERTY</td>
</tr>
<tr>
<td>6.1.2.</td>
<td>BEHAVIOUR LAW</td>
</tr>
<tr>
<td>6.1.2.1.</td>
<td>THE UMAT FUNCTION</td>
</tr>
<tr>
<td>6.1.2.2.</td>
<td>THE CALL OF UMAT BY A C-PROGRAM</td>
</tr>
<tr>
<td>6.1.2.3.</td>
<td>THE CALL WITH FORTRAN</td>
</tr>
<tr>
<td>6.1.2.4.</td>
<td>THE COMPILATION AND THE LINKAGE</td>
</tr>
<tr>
<td>6.1.2.5.</td>
<td>DIAGRAM</td>
</tr>
<tr>
<td>6.2.</td>
<td>TEST OF THE CALL OF THE MFRONT LAW WITH FORTRAN</td>
</tr>
<tr>
<td>6.2.1.</td>
<td>MATERIAL PROPERTY</td>
</tr>
<tr>
<td>6.2.2.</td>
<td>BEHAVIOUR LAW</td>
</tr>
<tr>
<td>6.3.</td>
<td>TEST IN GALILEO</td>
</tr>
<tr>
<td>6.3.1.</td>
<td>MATERIAL PROPERTY</td>
</tr>
<tr>
<td>6.3.2.</td>
<td>BEHAVIOUR LAW</td>
</tr>
<tr>
<td></td>
<td>CONCLUSION</td>
</tr>
<tr>
<td></td>
<td>BIBLIOGRAPHY</td>
</tr>
<tr>
<td></td>
<td>APPENDIX</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

FIGURE 1: UO

FIGURE 2: SCHEMATIC OF A FUEL ROD (SOURCE WIKIPEDIA) .......................................................... 8
FIGURE 3: FUEL NUCLEAR ELEMENTS (SOURCE WIKIPEDIA) ............................................................ 9
FIGURE 4: EXAMPLE OF A POWER HISTORY DURING A POWER RAMP .............................................. 11
FIGURE 5: THE CRACKED PELLET ...................................................................................................... 12
FIGURE 6: THE HOURGLASSING PELLET .......................................................................................... 12
FIGURE 7: EXAMPLE OF A MATERIAL PROPERTY .............................................................................. 16
FIGURE 8: EXAMPLE OF A BEHAVIOUR LAW .................................................................................... 18
FIGURE 9: MFRONT FILE OF THE NORTON LAW .............................................................................. 23
FIGURE 10: MFRONT FILE OF THE ORTHOTROPIC CREEP LAW ..................................................... 24
FIGURE 11: DIAGRAM OF THE LOADING OF THE PLATE. IN RED THE TRACTION LOADING, IN GREEN THE SHEAR LOADING AND IN BLUE THE FIXED PART IN BOTH CASES ............................................................... 25
FIGURE 12: \( \sigma_{YY} \) ALONG YY DEPENDING ON THE TIME FOR A TRACTION LOADING. THE RED CURVE IS WITH THE MFRONT-NORTON-LAW AND THE BLUE WITH CASTEM .................................................................................. 26
FIGURE 13: \( \varepsilon_{YY} \) DEPENDING ON THE TIME FOR A TRACTION LOADING. THE RED CURVE IS WITH THE MFRONT-NORTON-LAW AND THE BLUE WITH CASTEM .................................................................................. 27
FIGURE 14: \( \sigma_{YZ} \) DEPENDING ON THE TIME FOR A SHEAR LOADING. THE RED CURVE IS WITH THE MFRONT-NORTON-LAW AND THE BLUE WITH CASTEM .................................................................................. 27
FIGURE 15: \( \varepsilon_{YZ} \) DEPENDING ON THE TIME DURING A SHEAR LOADING STRESS. THE RED CURVE IS WITH THE MFRONT-NORTON-LAW AND THE BLUE WITH CASTEM .................................................................................. 28
FIGURE 16: \( \sigma_{YY} \) DEPENDING ON THE TIME FOR A TRACTION LOADING. THE RED CURVE IS WITH THE MFRONT-ORTHOTROPIC CREEP-LAW AND THE BLUE WITH CASTEM ......................................................... 29
FIGURE 17: \( \varepsilon_{YY} \) DEPENDING ON THE TIME FOR A TRACTION LOADING. THE RED CURVE IS WITH THE MFRONT-ORTHOTROPIC CREEP-LAW AND THE BLUE WITH CASTEM ......................................................... 29
FIGURE 18: \( \sigma_{YZ} \) DEPENDING ON THE TIME FOR A SHEAR LOADING. THE RED CURVE IS WITH THE MFRONT-ORTHOTROPIC-LAW AND THE BLUE WITH CASTEM .................................................................................. 30
FIGURE 19: \( \varepsilon_{YZ} \) DEPENDING ON THE TIME DURING A SHEAR LOADING. THE RED CURVE IS WITH THE MFRONT-ORTHOTROPIC CREEP-LAW AND THE BLUE WITH CASTEM .................................................................................. 30
FIGURE 20: \( \sigma_{YY} \) DEPENDING ON THE TIME FOR A TRACTION LOADING. THE RED CURVE IS WITH THE MFRONT-IMPLICITORTHOTROPIC CREEP-LAW AND THE BLUE WITH CASTEM .................................................................................. 31
FIGURE 21: \( \varepsilon_{YY} \) DEPENDING ON THE TIME FOR A TRACTION LOADING. THE RED CURVE IS WITH THE MFRONT-IMPLICITORTHOTROPIC CREEP-LAW AND THE BLUE WITH CASTEM .................................................................................. 32
FIGURE 22: \( \sigma_{YZ} \) DEPENDING ON THE TIME FOR A SHEAR LOADING. THE RED CURVE IS WITH THE MFRONT-IMPLICITORTHOTROPIC-LAW AND THE BLUE WITH CASTEM .................................................................................. 32
FIGURE 23: \( \varepsilon_{YZ} \) DEPENDING ON THE TIME DURING A SHEAR LOADING. THE RED CURVE IS WITH THE MFRONT-ORTHOTROPIC CREEP-LAW AND THE BLUE WITH CASTEM .................................................................................. 33
FIGURE 24: MFRONT FILE OF THE YOUNG MODULUS OF THE CLADDING ............................................. 35
FIGURE 25: COMPARISON OF THE VARIATION OF THE YOUNG MODULUS LAW IN FUNCTION OF THE TEMPERATURE BETWEEN MFRONT (YELLOW) AND CASTEM (RED) .................................................................................. 35
FIGURE 26: MFRONT FILE FOR THE PRIMARY THERMAL CREEP ................................................................ 36
FIGURE 27: MESH OF THE ANNUlus (LEFT) AND PRESSURE FORCE APPLIED ON IT (RIGHT). IN RED THE INTERNAL PRESSURE, IN PINK THE EXTERNAL .................................................................................. 37
FIGURE 28: MESH OF THE 3D-CYLINDER .......................................................................................... 37
FIGURE 29: COMPARISON OF RESULTS FOR THE CREEP OF THE 3D-CLADDING BETWEEN GALILEO (RED) AND CASTEM+MFRONT (BLUE) .................................................................................. 38
FIGURE 30: COMPARISON OF RESULTS FOR THE CREEP OF THE 3D-CLADDING BETWEEN GALILEO (RED) AND CASTEM+MFRONT (BLUE) .................................................................................. 38
FIGURE 31: MFRONT FILE OF AN ORTHOTROPIC ELASTIC LAW ....................................................... 39
FIGURE 32: MFRONT FILE OF AN ORTHOTROPIC CREEP LAW FOR THE CLADDING ...................... 40
FIGURE 33: COMPARISON OF RESULTS FOR THE ORTHOTROPIC CREEP OF THE 2D-CLADDING BETWEEN GALILEO (RED) AND CASETM+MFRONT (BLUE) ................................................................. 41
FIGURE 34: THE CALLING OF MFRONT MATERIAL PROPERTY VIA FORTRAN ................................................. 42
FIGURE 35: MFRONT UMAT PROTOTYPE ................................................................. 43
FIGURE 36: C-PROGRAM CALLING THE MFRONT LAW ................................................................. 45
FIGURE 37: THE CALLING OF UMAT VIA FORTRAN ................................................................. 46
FIGURE 38: DIAGRAM EXPLAINING THE LINK BETWEEN THE DIFFERENT CODES ................................................. 47
FIGURE 39: COMPARISON BETWEEN MFRONT LAW WITH FORTRAN AND EXCEL ................................................. 48
FIGURE 40: DIAGRAM OF THE LOADING OF THE ROD. IN BLUE THE STRAIN LOADING (NODE 2) AND IN GREEN THE FIXED NODE (NODE 1). ................................................................. 48
FIGURE 41: COMPARISON OF THE STRESS GIVEN BY THE CREEP LAW BETWEEN MFRONT+FORTRAN (RED) AND MFRONT+CASETM (BLUE) ................................................................. 49
FIGURE 42: COMPARISON OF ELASTIC STRAIN IN RADIAL DIRECTION WITH GALILEO (RED) AND MFRONT (BLUE) ................................................................. 49
FIGURE 43: COMPARISON OF POISSON RATIO WITH GALILEO (RED) AND MFRONT (BLUE) ................................................................. 50
FIGURE 44: COMPARISON OF THE YOUNG MODULUS WITH GALILEO (RED) AND MFRONT (BLUE) ................................................................. 50
FIGURE 45: COMPARISON OF THE THERMAL CONDUCTIVITY WITH GALILEO (RED) AND MFRONT (BLUE) ................................................................. 51
FIGURE 46: COMPARISON OF THE TEMPERATURE WITH GALILEO (RED) AND MFRONT (BLUE) ................................................................. 51
FIGURE 47: COMPARISON OF THERMAL STRAIN IN RADIAL DIRECTION WITH GALILEO (RED) AND MFRONT (BLUE) ................................................................. 52
FIGURE 48: DIAGRAM SHOWING HOW A FINITE ELEMENTS CODE WORK ................................................................. 52
INTRODUCTION:

This master thesis is realized in the Fuel Rod Code And Database group of the Materials and Thermo-mechanics department in the AREVA company. This group is dedicated to the development of the AREVA’s simulation code which calculates the thermo-mechanical behaviour of the fuel rods used in Pressured Water Reactor (PWRs). The code is called GALILEO.

The fuel rods exhibit significant strains, when they are in the reactor, so it is very important to guarantee that the rod will not break in a normal utilization or during small incidents. The rod acts as the first barrier of the nuclear fuel. The complex phenomena occurring in the rod imply to use numerical simulation to predict its behaviour. For the company it is essential to prove to the safety authorities and to the customers that the rods are safe and to define the limits they can sustain in case of accidents.

In the same time, a public institute, the CEA (Nuclear Energy Commission) develops its own code ALCYONE. This code is more accurate but exhibits a longer running time. The code ALCYONE is developed in a platform of codes dedicated to fuel simulation called PLEIADES. PLEIADES is actually a partnership between the CEA, AREVA and EDF (the French electricity company). Within this code, the CEA is creating a mechanical solver, MFRONT, which simplifies the generation and the application of the mechanical behaviour laws in codes developed in PLEIADES. The aim of this thesis is to analyze the opportunity of implementing MFRONT in GALILEO. First I will analyze how this solver works and make the verification of some examples given by the CEA. Then I will write AREVA laws in MFRONT and verify that they work. Finally I will study how MFRONT can work with GALILEO.
1. BACKGROUND OF THE THESIS

1.1. DESCRIPTION OF THE NUCLEAR FUEL

The fissile materials of reactors are shaped as cylindrical pellets, around 13.5 mm high and 8.2 mm of diameter. They are made of uranium oxide $\text{UO}_2$, with around 3-5% in fissile uranium $\text{U}^{235}$ (Figure 1).

![Figure 1: UO$_2$ Pellets (source AREVA)](image)

Around 300 pellets are vertically pilled up in a cylindrical tube of zirconium alloy, called clad, around 4 m high, 9.5 mm of diameter and 0.57 mm thick. This alloy exhibits a good resistance to corrosion, a high mechanical resistance and good neutronic properties. The most common alloys are the Zircaloy 4 (that contains tin, iron and chrome) and the alloy M5®, which is more resistant to corrosion through the adding of Niobium.

The pellets are maintained in contact in the rod by mean of a spring. The rod is finally pressurized with helium, in order to minimize the pressure difference between the inside and the outside of the rod, when it will be in the reactor (Figure 2).

![Figure 2: Schema of a Fuel Rod (source Wikipedia)](image)

The fuels rods are inserted together in a nuclear fuel element. A nuclear fuel element is constituted by 264 rods, in order to form a parallelepiped of around 4 m high and 30 cm width (Figure 3). Inside the heart of the 900 MWe reactor, there are 157 nuclear fuel elements, which adds up around 41 000 rods. This fact shows that the resistance of the rod is very important.
1.2. SIMULATION CODES

1.2.1. GALILEO

GALILEO is a simulation code developed by AREVA that predicts the thermal-mechanical behaviour of a single rod (the pellets and the cladding). It contains a consistent set of physical models for the analysis of the fuel in normal and off-normal conditions with regards to thermal, mechanical and fission-gas aspects. The code of a modular structure includes a set of stand-alone subprograms, each describing a single phenomenon. The subprograms are called by a driver program that controls the overall progress of the analysis. Special numerical subroutines control the time step and accelerate the convergence of the iterative processes.

The GALILEO code is therefore designed to cope with the following coupled physical phenomena:
- heat production and conduction in the fuel rod
- fission gas release
- mechanical behaviour of the pellet and cladding structures
- growth of a corrosion layer at the cladding external surface.

With the irradiation and thermal-hydraulic condition and the power histories, GALILEO is able to calculate and predict the behaviour of a fuel rod in a reactor. It allows to be sure that the behaviour of the rod is secure, during its life in the reactor.

For the calculation the fuel rod is subdivided into discrete axial regions (slices). These slices are further subdivided into around 26 discrete radial regions of concentric rings. This slice and ring composite forms the numerical framework for the mathematical analysis. At each time step the axial slices are first individually analyzed and then coupled and quantities such as the internal pressure are determined. This general mathematical calculation sequence, which is performed at each time step, produces fuel rod predictions that accurately simulate fuel rod behaviour.

1.2.2. CASTEM

CASTEM is a simulation code developed by the CEA, used to make thermal mechanical calculations for solids and fluids (www-cast3m.cea.fr). It applies a finite element method for solving scientific problems. It allows creating its own mesh and defining the physical model(s) applied on this mesh. The code solves then the problem and allows an easy post-treatment of the key-parameters. It is a code dedicated to the research. In point of fact, every parameter can be controlled and nothing is hidden in the code.
1.2.3. **ALCYONE**

ALCYONE is the reference simulation fuel rod code of the CEA. It uses the finite element code CASTEM, developed by the CEA. This code is able to use four kinds of rod models. One is in one dimension (1D), which allows the model of the whole rod, as GALILEO. Other models are in 2D or 3D. They allow to model one pellet or several pellets of the rod, where the power (and so the deformation) is maximized.

1.2.4. **MFRONT**

MFRONT is a new mathematical integrator from the CEA for fuel rod behaviour calculation. It is supposed to be used for every kind of materials properties:
- The material property
- The behaviour law
- Other strain models (corrosion, swelling ...)

As a matter of fact MFRONT is an interface, which creates libraries that can be used by finite element codes.

The advantages of MFRONT are:
- The use of clear inputs files, with only physical information
- The integrator is easily operated (very few computer skills needed)
- It creates an optimized materials library.

The aim of the CEA is to use it with ALCYONE and other codes. It is presently operational, but still in a testing phase. The integrator gets a new algorithm, which is much faster than the previous integrator already used now.

AREVA is keen to use this integrator with GALILEO.

1.3. **PROBLEMATIC AND STAKES OF THE THESIS**

The fuel rod is submitted to many strains during its period in the reactor. Considering the cladding of the rod being the first protection of the fission products, it is very important that it remains in a good shape during the whole process. So it is beneficial to predict the behaviour of the rod during the irradiation, in normal use and during incidents.

The fuel rod codes are therefore vital for AREVA. The use of GALILEO should show to the customers the characteristics of AREVA fuel rods and their limits. In addition to this aspect of the rod, there is also an economical stake for the fuel rod. In point of fact the more we take caution, the less the reactor is efficient. It is therefore important to consider reasonable but not excessive limits. The more precisely we know the behaviour of fuel rods, the better will be the outcome.

Another problem is the time that the calculations take. These kinds of calculation are very long and the amounts of calculations which are needed are extremely large. The new calculator MFRONT should be both more accurate and faster. So the aim of the thesis is:

- To understand the principle of this integrator.
- To validate the functioning of it in simple case.
- To write AREVA’s behaviour laws with MFRONT and validate the results.
- To try to connect it with the AREVA simulation code GALILEO and check its efficiency.
2. **BEHAVIOUR OF THE NUCLEAR FUEL IN THE REACTOR**

2.1. **THE IRRADIATION HISTORY**

The aim of the fuel rod simulation codes is to reproduce the behaviour of the fuel and the cladding, during a given history of irradiation. These histories represent either normal condition of utilisation or incidental condition with unusual increase of power, which can lead to the failure of the cladding. A typical shape of a history in a reactor is shown in Figure 4. The variation of power that the rods receive follows most of the time that shape. There are a few cycles of irradiation (usually 3) in normal use (around 200 W/m). Each cycle is supposed to last one year.

![Diagram of normal cycles over time](image)

**Figure 4: Example of a Power History during a power ramp**

2.2. **THE PELLET BEHAVIOUR**

At the beginning of the irradiation, the pellet gets denser. But quickly due to the accumulation of the fission product, the pellet density decreases. In point of fact the number of atoms (and the same do the volume) increases, but the mass remains the same. This is called the pellet swelling.

The temperature increase leads to the thermal expansion of the pellet. As there is a temperature gradient (the temperature is higher in the centre of the pellet than in the border) it induces internal stresses inside the pellet, because the heart of the pellet tries to expand more than the border. According to the brittle behaviour of ceramics, the UO$_2$ pellet gets some cracking. This happens in the beginning of irradiation history (Figure 5).
Figure 5: The Cracked Pellet

The ruptures happen mostly in the radial direction and we usually get around six to ten fragments. The displacement of these fragments due to the vibration and the thermal expansion leads to a small increase of the pellet diameter. This is the relocation phenomenon. One other main consequence of the important temperature gradient is a thermo-elastic deformation of the cracked pellet, which leads to a hourglassing shape (Figure 6).

Figure 6: The Hourglassing Pellet

Finally there are two important phenomena which occur during very high power level. On one hand we observe an important swelling in the middle plan of the pellet. This is due to the accumulation of gaseous fission products. On the other hand there is a creep strain which will lead to the filling of the dishing hole of the pellet.

2.3. THE CLADDING BEHAVIOUR

The temperature in the cladding is much lower than in the pellet, so the thermal expansion in the cladding is very low. The main deformation of the cladding is due to the radial creep. This is linked to the pressure difference between the inside (gap pellet/cladding) and the outside of the cladding. The creep deformation is irreversible and increases to a large extent with temperature and neutron flux.

Owing to the increase of temperature the pressure in the gap increases at the beginning to a lower value than the external pressure. So the cladding exhibits compression stresses, which deforms the cladding in order to suppress the gap between the cladding and the pellet.
Initially the cladding presents a small ovality. In the reactor this ovality increases until the cladding reaches the pellet at the point of the small axis of the ellipse. Then the ovalization gets stabilized and even decreases as the cladding reaches the whole pellet.

Another important effect within the cladding which occurs is the corrosion of its external surface. The small layer of corrosion must not exceed 100 μm. If the corrosion is a problem with the alloy Zircaloy 4, this problem is solved with the M5® alloy.

2.4. THE PELLET/CLADDING INTERACTION

The initial gap existing between the pellet and the cladding decreases during the irradiation. This is due to the strains described above. After a certain amount of utilization, the closure of the gap happens. This is called the pellet-cladding interaction (PCI). It creates tensile stresses and local strains in the cladding and a relocation of the pellet fragments in the inside of the rod. As far as the pellet keeps on swelling, it pushes on the cladding, whose diameter increases.

The pellet-cladding contacts begin in the inter-pellets planes, according to the hourglassing shape of the pellets. The cladding strains due to the pellet shape occur first on this level (called primary ridges). During a high power level, the important swelling in the middle plane of the pellets leads also to cladding strains. These are called secondary ridges, and are often more important than the first ones.

Shortly after the contact, stresses in the cladding are stabilized to a value depending on the external pressure, the pellet strains and the power level. It is said that the cladding is conditioned to a power level. If the power increases and overtakes the power of the conditioned rods, the stresses also increase. In accident cases, some cracking in the cladding can occur in front of pellet crack and the cladding can break. When this occurs, this is due to the combined effect of the stresses and the apparition of the corrosive fission products. The power level in reactor is limited in order to never be in such cases.

2.5. CALCULATION MODEL OF THERMO-MECHANICAL BEHAVIOUR OF THE NUCLEAR FUEL IN A SIMULATION CODE

The analytic model of the thermo-mechanical behaviour of the fuel rod is very difficult for the following reasons:

- the non-linearity of the major part of the laws
- a partial understanding of the behaviour of complex materials during the irradiation
- the presence of chemical effects, leading to the corrosion of the cladding and the transport of species in the fuel
- an important interaction between all these phenomena

Such a complexity leads to a specific distinction of different problems and the use of simulation codes.

2.5.1. THE DIFFERENT KINDS OF CODES

There are a few kinds of modes for fuel rods simulations. The literature (mostly [3] and [7]) show a review of the available codes. The easiest models consider that the rod is a fulfilled pipe of infinite length and they express that the parameters of the problems (temperature, strains, stresses ...) only depend on the time \( t \) and the distance from the axis (the radius of the cylinder), \( r \). This is so clearly axisymmetric. The 1D model is considered when this axisymmetric hypothesis is used to model the rod behaviour in a specific height. Some 1D models can be easily coupled along the axis to model the whole rod. The models are commonly called 1.5D, and often follow the hypothesis of plane strain \( \varepsilon_z = \text{Constant} \).
There are two different kinds of 2D models. First the 2DRT models ignore the axisymmetric hypothesis and just make the calculation at a specific position along the height. The parameters only depend on \( r, \theta \) and the time \( t \). The second models are the 2DRZ models that keep the axisymmetric hypothesis. They model a part of the rod or a fragment of the pellet. The parameters only depend on \( r, z \) and the time \( t \).

The 3D models represent a whole fragment of the pellet or the whole pellet.

Although there are many changes in these codes, some observations are still true:
- Most of the codes are 1D or 1.5D.
- These codes are the only ones that can model the whole rod, because of calculation times and data storage (except some recent exceptions).
- Every code has at least three parts: a thermal calculation, a mechanical one and a gas behaviour part.

### 2.5.2. STRUCTURE OF FUEL ROD CODES

The inputs of fuel rod codes are basically always the same. They are the geometry, the materials, the external pressure and the external temperature of the rod and the irradiation history ([4]).

The elements are analyzed in a double-loop on the thermal-mechanical increment first and then on the time increment. During this analysis, the thermal calculation is done first. It gives the temperature field in the rod. Then the mechanical calculation is done, which determines the strains and the stresses using the temperature field. As seen before, other models (such as corrosion, gas swelling...) are involved in the process, but they will not be explained in the following part.

### 2.5.3. HEAT EQUATION

The temperature field is calculated with the heat equation (1) and the Fourier law (2):

\[
\begin{align*}
P_v - \text{div}(\tilde{\phi}) &= C_p \cdot \rho \cdot \frac{\partial T}{\partial t} & (1) \\
\tilde{\phi} &= -\lambda \cdot \text{grad}(T) & (2)
\end{align*}
\]

where \( P_v \) is the volume power given by internal sources (fissions), \( \tilde{\phi} \) the heat flux, \( \lambda \) the thermal conductivity, \( C_p \) the specific heat, \( \rho \) the density, \( T \) the temperature and \( t \) the time.

That explains the need of a numerical resolution. At steady state, an example of the temperature field is

\[
T_{\text{heart}} - T(r) = \frac{P_i}{4 \cdot \pi \cdot \lambda} \left( \frac{r}{R} \right)^2,
\]

With \( P_i \), given by the neutronic codes. \( \lambda, \ C_p \) and \( \rho \) are constant. The \( \left( \frac{r}{R} \right)^2 \) term shows the temperature gradient is close to a parabolic shape.

The boundary condition is always the same for the external part of the cladding. It states that there is a heat transfer between the cladding and the water. The boundary condition in the heart is the nullity of the temperature gradient in the heart.

### 2.5.4. MECHANICAL EQUATION
The main equations of the mechanical calculation for the rod are the partition of strains (1), the equilibrium equation (2) and the Hooke’s law (3):

\begin{align*}
\varepsilon &= \varepsilon^{\text{elastic}} + \sum \varepsilon^{\text{inelastic}} \quad (1) \\
div(\sigma) &= 0 \quad (2) \\
\varepsilon^{\text{elastic}} &= C \cdot \sigma \quad (3)
\end{align*}

The principle of the partition of strains is that every physical phenomenon is uncoupled. This is physically not completely true, but it gives a very good approximation. So it is commonly admitted. Each phenomenon has effects either on the geometry or on the temperature. An effect on the temperature will also lead to an effect on the geometry. So each inelastic strain is calculated with the temperature field, with semi-empirical models. Then the finite element method leads the entire problem to a single equation. So the problem is to find all the inelastic deformations.

For the pellets, there are:
- Thermal expansion
- Densification
- Relocation
- Gaseous swelling
- Solid swelling
- Creep
- Cracking.

For the cladding:
- Thermal expansion
- Irradiation swelling
- Creep
- Plasticity.
3. **MFRONT**

3.1. **GENERAL PRESENTATION**

MFRONT is a code generator dedicated to mechanical laws. With a file containing the mathematical and physical information of the law, it creates C++ files, which are used to solve mechanical calculations. It uses a mathematical library of the CEA, called TFEL, which is included in the PLEIADES platform. MFRONT operates with GNU compilers and with LINUX.

MFRONT can actually create libraries for the material properties (i.e. the relevant coefficient of thermo-mechanics such as the Young-Modulus, the thermal conductivity ...), the behaviour laws (elastic law, creep law, plastic law ...) and for other models (gaseous swelling, corrosion ...). Those libraries can be used with FORTRAN, C, C++, CASTEM ... Concerning the behaviour law, MFRONT uses the UMAT interface. The UMAT interface is a subroutine which allows making mechanical calculation. It is used in many finite element codes (such as ABAQUS, CASTEM ...). The library created with UMAT can then be used with FORTRAN, C, CASTEM ...

A typical MFRONT library is supposed to make faster calculation than the previous ones. In point of fact MFRONT allows choosing a specific (and so an optimized) calculation method for each law. The choice of the method is controlled via a parser. The parser is the basis of MFRONT principle. The choice of the parser actually allows to choose the numerical method to solve the equation. (Runge-Kutta, Newton-Raphson ...). It also fixes the form of the rest of the MFRONT file.

3.2. **MFRONT FILE**

A MFRONT file contains the information needed for a law. The writing of this file is rather complicated and depends on the kind of the law (material property, behaviour law and model). This section explains how the MFRONT file is built. This section (and the whole study of MFRONT) will be limited on the material property and the behaviour law. Those files are the examples given by the CEA.

3.2.1. **MATERIAL PROPERTY**

Figure 7 gives an example of the MFRONT-file of a material law.

```plaintext
@Parser MaterialLaw;
@Material Zr
@Law YOUNGMODULUS;
@Author Julien Olagnon
@Date 2013-01-02;
@Library CladYoung
@Output E;
@Input T,p;
@PhysicalBounds T in [0:*[;
@PhysicalBounds p in [0:1];
@Bounds T in [300,1500]
@Function{
    const real E0 = 1.E9 ;
    const real B  = 1.E6 ;
    const real T0 = 300. ;
    E = (E0-(B*T*exp(-T0/T)))*exp(-p);
}
```

Figure 7: Example of a material property
This law gives the Young modulus $E$, function of the temperature ($T$) and the porosity ($\text{poro}$). The file looks like a series of different directives. Each directive has a special function. Table 1 gives the name and the function of the directives used in Figure 7.

<table>
<thead>
<tr>
<th>Directives</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Parser</td>
<td>It gives the parser which will be used. (For the material property it is always MaterialLaw)</td>
</tr>
<tr>
<td>@Law</td>
<td>It gives the name of the law. The user is completely free to choose this name.</td>
</tr>
<tr>
<td>@Material</td>
<td>It gives the name of the material of the law. The user is also completely free to choose this name.</td>
</tr>
<tr>
<td>@Author and @Date</td>
<td>They give the author and the date of the file. This has not much importance except to know and to contact the author of the law.</td>
</tr>
<tr>
<td>@Library</td>
<td>It gives the name of the library created by the MFRONT file. The user is also completely free to choose this name.</td>
</tr>
<tr>
<td>@Input</td>
<td>It gives the input of the law (the temperature and the porosity).</td>
</tr>
<tr>
<td>@Output</td>
<td>It gives the output of the law (here the Young modulus).</td>
</tr>
<tr>
<td>@PhysicalBounds</td>
<td>It shows the admitted value for different parameters (here a porosity registers between 0 and 1 and a temperature is always positive). It shows an error during the calling of the law if these criteria are not respected.</td>
</tr>
<tr>
<td>@Bounds</td>
<td>It shows the admitted value for different parameters for the law (here the temperature is between 300K and 1500K, for this law). It also shows an error during the calling of the law if these criteria are not respected, but the error is a different in this case. The user can deal with the way he wants to.</td>
</tr>
<tr>
<td>@Function</td>
<td>It gives the equation of the law.</td>
</tr>
</tbody>
</table>

**Table 1:** Name and function of the directives used in the material property (Figure 7).

The creation of the library for the file mfrontlaw.mfront can be done under LINUX, with the command:

```
mfront --obuild --interface=castem mfrontlaw.mfront
```

The directive `--obuild` implies an optimized compilation and the directive `--interface=castem` allows choosing the language we want to use the property. The word CASTEM can be replaced by C, C++, FORTRAN...

The name of the created library is “lib” + “the name of the chosen interface” + “the name given after the directive @Library” (in this case it is `libCastemCladYoung.so`). The name of the created law is “the name given after the directive @Material” + “_” + “the name given after the directive @Law” (in this case it is `Zr_YOUNGMODULUS`). Those names are useful for the calling of the law.
3.2.2. BEHAVIOUR LAW

Figure 8 gives an example of the MFRONT file for a very simple behaviour law:

```plaintext
@Parser DefaultParser;
@Behaviour Elastic;
@Author Julien;
@Date 2012/03/11;
@Library UO2
@RequireStiffnessTensor;
@Integrator{
  sig = D*(eto+deto);
}
```

*Figure 8: Example of a Behaviour Law*

It is the law of an elastic behaviour. Some directives can be different than the ones of the material law. Table 2 gives the function of the used directives of Figure 8. Other important directives used in the following parts are given Table 3. The parameters $\text{sig}$ is the stress and $\text{eto}$, the total strain and $\text{deto}$ the increment of total strain. Those names are fixed in MFRONT.

<table>
<thead>
<tr>
<th>Directives</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Parser, @Author and @Date</td>
<td>They are the same as the ones described above. The different parsers used in this report are DefaultParser, RungeKutta, Implicit and IsotropicMisesCreep. The other parsers are given in Appendix</td>
</tr>
<tr>
<td>@Behaviour</td>
<td>It has the same role as the directive @Law in the material law MFRONT-file. The user is completely free to choose this name.</td>
</tr>
<tr>
<td>@Library</td>
<td>It gives the name of the library created by the MFRONT file. The user is also completely free to choose this name.</td>
</tr>
<tr>
<td>@RequireStiffnessTensor</td>
<td>It creates the parameter $D$, which contains the stiffness matrix.</td>
</tr>
<tr>
<td>@Integrator</td>
<td>It contains the equation the integrator should solve at each step. $\text{sig}$ is the stress tensor, $\text{eto}$ the strain tensor and $\text{deto}$ the increment of the strain tensor, during the step. This word and the form of the equation depends on the chosen parser.</td>
</tr>
</tbody>
</table>

*Table 2: Names and function of the directives used in the behaviour law (Figure 8).*

The directive containing the equation could have a different name considering the kind of laws and the parser. The creation of the library for the file mfrontlaw.mfront is also done via LINUX, with the command:

```
mfront --obuild --interface=umat mfrontlaw.mfront
```

The directive --obuild implies an optimized compilation and the directive --interface=umat shows the library is only done for UMAT. This library can then be called with the other language.
The name of the created library is the same as seen previously (in this case it is `libUmatUO2.so`). Furthermore in those cases the name of the interface is always `umat`. The name of the created law is “`umat`” + “the name given after the directive `@Library`” + “the name given after the directive `@Behaviour`” (in this case it is `umatUO2Elastic`).

<table>
<thead>
<tr>
<th>Directives</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>@FlowRule</td>
<td>This is the directive containing the equation of the law, according to the Parser <code>IsotropicMisesCreep</code>.</td>
</tr>
<tr>
<td>@Algorithm</td>
<td>It precises the utilised algorithm used with the parser (only for <code>RungeKutta</code> and <code>Implicit</code> parser).</td>
</tr>
<tr>
<td>@StateVariable</td>
<td>It gives the state variables of the problem (for example the viscoplastic strain and the equivalent strain). Those variables describe the local mechanical state of the material. The behaviour law characterized the evolution of these parameters. It is important to notice that the elastic strain is implicitly given.</td>
</tr>
<tr>
<td>@AuxiliaryStateVariable</td>
<td>It gives other state variables, which are not required for the integration but can be useful for the post-treatment.</td>
</tr>
<tr>
<td>@LocalVariable and @InitLocalVariable</td>
<td>They are directives which allow to give the local variable and to initialize them. A local variable is a variable which can be calculated before the integration, in order to prevent useless calculation. Here it is the Hill-Tensor, which defines the orthotropic material.</td>
</tr>
<tr>
<td>@ComputeStress</td>
<td>It calculates the stress tensor.</td>
</tr>
<tr>
<td>@Integrator</td>
<td>This is the directive containing the equation of the law, according to the Parser <code>Implicit</code> and <code>DefaultParser</code>.</td>
</tr>
<tr>
<td>@Derivative</td>
<td>This is the directive containing the equation of the law, according to the Parser <code>RungeKutta</code>.</td>
</tr>
<tr>
<td>@UpdateAuxiliaryStateVariable</td>
<td>It allows updating the auxiliary state variables after the calculation.</td>
</tr>
<tr>
<td>@Coef</td>
<td>It declares the coefficients of the law that can be adjusted by the user.</td>
</tr>
<tr>
<td>@OrthotropicBehaviour</td>
<td>It allows the use of the orthotropic laws.</td>
</tr>
</tbody>
</table>

**Table 3: Names and function of the directives used in other behaviour law**

In addition of the directive, specific key-words are used for the writing of the equations for the physical parameters. Table 4 gives the most important ones. Those keywords can be implicitly declared among the parser.
### Table 4: List of the main MFRONT keywords

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Physical parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>eto</td>
<td>Total strain tensor</td>
</tr>
<tr>
<td>eel</td>
<td>Elastic strain tensor</td>
</tr>
<tr>
<td>evp</td>
<td>Viscoplastic strain tensor</td>
</tr>
<tr>
<td>sig</td>
<td>Stress tensor</td>
</tr>
<tr>
<td>seq or sigeq</td>
<td>Equivalent stress</td>
</tr>
<tr>
<td>T</td>
<td>Temperature</td>
</tr>
<tr>
<td>H</td>
<td>Hill matrix</td>
</tr>
<tr>
<td>p</td>
<td>Equivalent strain</td>
</tr>
<tr>
<td>f</td>
<td>The flow function</td>
</tr>
<tr>
<td>d+another keyword (as deel)</td>
<td>Increment of the parameter (in this case increment of elastic strain).</td>
</tr>
<tr>
<td>d Keyword +_+ d another keyword (as deel_dseq)</td>
<td>Derivative of those keywords (in this case derivation of the elastic strain by the equivalent stress).</td>
</tr>
</tbody>
</table>

### 3.3. MFRONT WITH CASTEM

MFRONT law can be used with CASTEM. The section shows how to connect the MFRONT law with the code. An easy way of using the libraries is to create a new folder and copy all the MFRONT libraries in this folder and to add this folder in the environment variable `LIBRARY_PATH` of LINUX. With this solution, I am sure to have all the libraries and that the codes find them easily.

#### 3.3.1. MATERIAL PROPERTIES

The use of these laws begins by the creation of a table with three inputs:

```plaintext
TLoi = 'TABLE' ;
TLoi.'LIBRAIRIE' = 'libCastemCladYoung.so' ;
TLoi.'MODELE' = 'Zr_YOUNGMODULUS' ;
TLoi.'VARIABLES' = 'MOTS' 'T' 'PORO';
```

The inputs of the laws are `MODELE` the name of law, `LIBRAIRIE` the name and the location of the library CASTEM will use for applying the law. The third input is `VARIABLES`, it describes the name of the inputs of the law (here the temperature and the porosity).

After describing the library the law has to be processed by the program. For the material law, it occurs in the `MATE` function (with `MOD1`, the CASTEM model the law is applied on):

```plaintext
MAT1=MATE MOD1 YOUN TLOI NU 0.3 ;
```
3.3.2. BEHAVIOUR LAW

For the behaviour law, the calling of the law begins in the same way, with the creation of a table, which has only two inputs this time:

\[
\begin{align*}
T\text{Loi} &= 'TABLE' ; \\
T\text{loi}.'MODELE' &= 'umatUO2Elastic'; \\
T\text{loi}.'LIBRAIRIE' &= 'libUmatUO2.so';
\end{align*}
\]

The calling of behaviour laws is a little bit different, it occurs in the MODE function:

\[
\begin{align*}
\text{Coel3D} &= 'MOTS' 'YOUN' 'NU' 'RHO' 'ALPH'; \\
\text{Stav3D} &= 'MOTS' 'EEXX' 'EEYY' 'EEZZ' 'EEXY' 'EEXZ' 'EEYZ' 'P'; \\
\text{Para3D} &= 'MOTS' 'T';
\end{align*}
\]

\[
\text{Model} = \text{MODE mail1 'MECANIQUE' 'ELASTIQUE' 'NON_LINEAIRE'}
\]

\[
'\text{UTILISATEUR'} '\text{DESC_LOI'} T\text{Loi}
\]

\[
'\text{C\_MATERIAU'} \text{coel3D}
\]

\[
'\text{C\_VARINTER'} \text{stav3D}
\]

\[
'\text{PARA\_LOI'} \text{para3D};
\]

First three lists are created, with the name of the material parameters (Coel3D), the name of internal parameters (Stav3D) and the name of external parameters (Para3D). The material parameters are the usual thermo-mechanical coefficients (such as Young modulus, Poisson’s ratio ...). The internal variables describe the mechanical state of the material. They are calculated with the behaviour law. In this case they are the elastic strain tensor (EEXX, EEYY ...) and the equivalent stress (P). The external variables describe other variables whose evolution is given by another. They are calculated with other model. In this case this is only the temperature (T). Then we create the model with these lists and the table with the law. The lists depend on the dimension which the calculation is doing and the kind of the law. So it is very important to have a good understanding of the law before using it. During the MFRONT file compilation a CASTEM file is generated which shows how the law should be called.

Then at the end, after the calculation we can find the output in the table solution in:

- table.'DEFORMATIONS’, for the total strain.
- table.'CONTRAINTE’, for the total strain.
- table.'VARIABLES INTERNES’, for the internal variables.

The utilisation of MFRONT with CASTEM may be seen as quite difficult, but actually, after understanding the principle, it is very easy to use. For implementing it in some existing CASTEM example (which can be found on the CASTEM website: www-cast3m.cea.fr/index.php?page=exemples), only two steps are required: first creating the table of the law and then modifying the CASTEM directives MODE and MATE according to the law.

3.4. MFRONT WITH GALILEO

The use of MFRONT with GALILEO is far more complicated. A whole part of the report is dedicated to this explanation (see Section 6).
3.5. MFRONT WITH ALCYONE

The use of MFRONT with ALCYONE is for the moment rather complicated. First we have to modify the UTILPROC file and replace an existing law, by the new MFRONT law. Then we should recompile ALCYONE again. MFRONT is not completely linked with ALCYONE for a proper use. For those reasons this report will not speak about the use of MFRONT with ALCYONE.
4. VALIDATION OF MFRONT WITH CASTEM

We got from the CEA three simple behaviour laws and one material law. Before using it, the aim is that we should assure that they work. So simple cases are realized in CASTEM (a CEA finite element code) and we make the calculation with the MFRONT laws and the same laws of CASTEM. Then we compare it. Furthermore will be evaluated how much time is gained with the use of MFRONT.

The laws we have have:
- A material property which gives the evolution of the Young modulus as function of porosity and temperature for a SiC material,
- An isotropic creep of Norton,
- An orthotropic creep of Norton,
- An orthotropic creep of Norton, which is solved by an implicit integration method (supposed to be much faster).

4.1. MATERIAL LAW

For the test of the SiC Young modulus (Figure 7), I used a simulation of cylinder submitted to a force. The bottom of cylinder is fixed and a force of 25 000 N is applied at one point of the top, along the cylinder axis. This test is inspired of an example available on the CASTEM website, called elas2.dgibi. I created two different inputs. In the first one, I rewrote the Young modulus law inside CASTEM, and with the second one, I called the law with MFRONT. The code gives the displacement along the axis of the point where the force is applied. It can be found in Appendix 3.

For both, we found 1.56069*10^-3 m. The values are identical to 6 digits, suggesting that the law is correctly described by MFRONT. It validates the MFRONT file of the law and the parser ‘MaterialLaw’ for AREVA, given the fact MFRONT is validated internally at CEA. Furthermore it confirms the successful implementation of MFRONT in AREVA.

4.2. BEHAVIOUR LAW

4.2.1. MFRONT FILE

Figure 9 and Figure 10 show the MFRONT files of behaviour law, which has been tested (the ImplicitOrthotropicCreep is given in Appendix 2).

```plaintext
@Parser IsotropicMisesCreep;
@Behaviour Norton;
@Library Norton;
@FlowRule{
    real A = 8.e-67;
    real tmp = A*pow(seq,7.2);
    f = tmp*seq;
    df_dseq = 8.2*tmp;
}
```

Figure 9: MFRONT file of the Norton Law
Figure 10: MFRONT file of the OrthotropicCreep Law
The understanding of the Norton file (Figure 9) is not difficult. The file of OrthotropicCreep is a little bit more complicated (Figure 10). The directives of these files are given in Table 2 and Table 3. The parser is IsotropicMisesCreep for the Norton law and RungeKutta for the OrthotropicCreep law. For the first file the expression of both variables $f$ (the flow function) and $df\_dseq$ (the derivative of the flow function by the equivalent stress) are required for this parser. As its name implies, the second one uses a Runge-Kutta method. It uses an algorithm called rk54 (which combines Runge-Kutta methods of order 4 and order 5). The expressions of the increments the equivalent strain ($dp$), the creep strain ($devp$) and the elastic strain ($deel$) are needed according to the RungeKutta parser.

4.2.2. TESTS

For those validations, I used tests of a plate submitted first to a tensile loading, then to a shear loading. Those are inspired by the CASTEM examples creep01_trayY and creep01_cisYZ. It consists of a thin square plate. The length is 1 m and the width is 1 cm. The first example is an example of pure traction and pure shear stress. The simulation time is one hour. The plate is submitted to a force of 5.0e8 N in the traction case and 2.0e7 N for the shear stress (Figure 11). The force grows up linearly from 0 to its final value during the first sixty seconds and then remains constant. The integration step is 30 s for the traction and 10 s for the shear stress. The CASTEM code is given in Appendix 4.

![Figure 11: Diagram of the loading of the plate. In red the traction loading, in green the shear loading and in blue the fixed part in both cases.](image-url)
In all cases, the plots of the stresses and the strains in the important direction for both laws are displayed (MFRONT in red and CASTEM in blue).

**-Norton:**

In this case, this is a Norton law. Its expression is:

$$\varepsilon_{eq} = A \cdot \sigma_{eq}^{6.2}$$

$A = 8.0 \cdot 10^{-6.7}$ is a constant, $\varepsilon_{eq}$ is the equivalent creep strain and $\sigma_{eq}$ is the Von-Mises equivalent stress. So we have, for the traction loading:

![Graph showing $\sigma_{yy}$ along YY depending on the time for a traction loading. The red curve is with the MFRONT-Norton-Law and the blue with CASTEM.](image)

**Figure 12:** $\sigma_{yy}$ along YY depending on the time for a traction loading. The red curve is with the MFRONT-Norton-Law and the blue with CASTEM.
Figure 13: $\varepsilon_{yy}$ depending on the time for a traction loading. The red curve is with the MFRONT-Norton-Law and the blue with CASTEM.

The strains and the stresses in the other directions are equal to zero. And for the shear loading:

Figure 14: $\sigma_{yz}$ depending on the time for a shear loading. The red curve is with the MFRONT-Norton-Law and the blue with CASTEM.
Figure 15: $\varepsilon_{YZ}$ depending on the time during a shear loading stress. The red curve is with the MFRONT-Norton-Law and the blue with CASTEM.

Figure 12, Figure 13, Figure 14 and Figure 15 show that the Norton law of CASTEM and MFRONT give identical results. It validates the Norton law of MFRONT and the corresponding parser (here “IsotropicMisesCreep”).

- OrthotropicCreep:

The OrthotropicCreep law presents the same law as the Norton law, but it is coded for an orthotropic material. The parser is different and its name is RungeKutta. As its name implies, it uses a Runge-Kutta method. The used algorithm is defined in the directive @Algorithm. For this file it is rk54 (it combines Runge-Kutta methods of order 4 and order 5).

Nevertheless there is no orthotropic creep law, in CASTEM. Consequently I only tested the OrthotropicCreep in isotropic cases. The isotropic case is obtained if the Young modulus and the Poisson’s ratios are equal along the three directions and the Hill coefficients are equal to 0.5 or 1.5.

The traction loading is seen in Figure 16 and Figure 17:
Figure 16: $\sigma_{yz}$ depending on the time for a traction loading. The red curve is with the MFRONT-OrthotropicCreep-Law and the blue with CASTEM.

Figure 17: $\varepsilon_{yz}$ depending on the time for a traction loading. The red curve is with the MFRONT-OrthotropicCreep-Law and the blue with CASTEM.
And the shear loading example is shown in Figure 18 and Figure 19:

![Graph showing shear loading example](image)

**Figure 18:** $\sigma_{yz}$ depending on the time for a shear loading. The red curve is with the MFRONT-Orthotropic-Law and the blue with CASTEM.

![Graph showing shear loading example](image)

**Figure 19:** $\varepsilon_{yz}$ depending on the time during a shear loading. The red curve is with the MFRONT-OrthotropicCreep-Law and the blue with CASTEM.
The curves by the CASTEM law and the MFRONT law are the same. These tests allow showing the good function of the parser RungeKutta. The validation of the orthotropic materials will be seen later when we write our own orthotropic laws.

**ImplicitOrthotropicCreep:**

The ImplicitOrthotropicCreep law is the same law as the OrthotropicCreep law, but it uses an implicit parser (called Implicit), which means that some parameters are not directly calculated (here it is the viscoplastic strains). This method is supposed to be faster and more robust. For the same reasons, the name of the algorithm has to be given. If it is not given, as in this example (Appendix 2), the Newton-Raphson integration method is used.

For the same reasons as above, the law is only tested on isotropic cases, but it allows validating the Implicit parser.

So the traction loading is seen in Figure 20 and Figure 21:

![Graph](image)

**Figure 20:** \(\sigma_{yy}\) depending on the time for a traction loading. The red curve is with the MFRONT-ImplicitOrthotropicCreep-Law and the blue with CASTEM.
Figure 21: $\varepsilon_{yz}$ depending on the time for a traction loading. The red curve is with the MFRONT-ImplicitOrthotropicCreep-Law and the blue with CASTEM.

And the shear loading is shown in Figure 22 and Figure 23:

Figure 22: $\sigma_{yz}$ depending on the time for a shear loading. The red curve is with the MFRONT-ImplicitOrthotropic-Law and the blue with CASTEM.
Figure 23:  $\varepsilon_{YZ}$ depending on the time during a shear loading. The red curve is with the MFRONT-OrthotropicCreep-Law and the blue with CASTEM.

The curves obtained from the CASTEM law and the MFRONT law are the same. So we also validate the Implicit Parser.

As these different laws are the same, it seems interesting to compare numerically them to each other. For the three MFRONT and the CASTEM laws, the maximum of stress and strain at the end has been calculated (Table 5).

<table>
<thead>
<tr>
<th></th>
<th>MFRONT</th>
<th>CASTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum Strain</td>
<td>5.00E+10</td>
<td>5.00E+10</td>
</tr>
<tr>
<td>Maximum Stress</td>
<td>0.3089</td>
<td>0.3088</td>
</tr>
<tr>
<td>Shear</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum Strain</td>
<td>2.00E+07</td>
<td>2.00E+07</td>
</tr>
<tr>
<td>Maximum Stress</td>
<td>0.3284</td>
<td>0.3289</td>
</tr>
</tbody>
</table>

Table 5: Comparison of the results of the different laws.

The values of all the cases are almost the same. The most distant value is the strain calculated by the MFRONT-Norton law, but it still is very similar (less than 0.15% error). It can be explained by the integration method, which is supposed to be easier (and so faster).

Now, it could be interesting to compare the running time of these different laws and see the utility of MFRONT. For a better accuracy, the mesh has been divided. For the traction the square is divided into 4 squares and for the shear stress into 25 squares. Results have been shown Table 6.
### Table 6: Comparison of the results of the different laws.

<table>
<thead>
<tr>
<th>Running Time (s)</th>
<th>MFRONT</th>
<th>CASTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FlowRules</td>
<td>RungeKutta</td>
</tr>
<tr>
<td>Traction</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>Shear</td>
<td>32</td>
<td>40</td>
</tr>
<tr>
<td>Saving time (%)</td>
<td>37%</td>
<td>15%</td>
</tr>
</tbody>
</table>

The saving time percentage seems important for these small examples, especially for the parser IsotropicMisesCreep and Implicit. The saving time could be very large with MFRONT. Those tests prompt the use of MFRONT.

**Remark:** Some small differences can appear between CASTEM and MFRONT, for very short tests. It is due to the precise instant in the time step the calculation is made. This can be changed in the MFRONT file by the directive @Theta for some parsers. But it has actually no importance, because the calculations are the same. The only important thing is to know which calculation is made.

### 4.3. CONCLUSION

Those different examples show that the utilization of MFRONT is reliable. It gives consistent results with at least the same precision as other integrators. Furthermore it seems to have a better running time. Its use with CASTEM is accessible and the way to code new law is very intuitive.

The aim now is to create a MFRONT file of each behaviour law used in GALILEO, in order to generate the library of those laws and to make the calling of them easier.
5. APPLICATIONS OF AREVA LAWS WITH MFRONT

The coefficients are standardized in the part of the report.

5.1. MATERIAL PROPERTY: YOUNG MODULUS

I wrote the Young modulus law for the cladding. Figure 24 gives the MFRONT files.

```plaintext
@Parser MaterialLaw;
@Material M5;
@Law YoungModulus;
@Output E;
@Input T;
T.setGlossaryName("Temperature");

@PhysicalBounds T in [0:*[;
@Bounds T in [273.15:1600];
@Function
{
    real E1 = 100.-0.012*T;
    E = E1*1.e9;
}
```

Figure 24: MFRONT file of the Young Modulus of the Cladding

![Graph](image)

Figure 25: Comparison of the variation of the Young Modulus law in function of the temperature between MFRONT (Yellow) and CASTEM (Red)
The verification of the law is simple. I calculated the value of Young modulus for a few temperatures by CASTEM once with MFRONT once directly with CASTEL and compared them. Figure 25 shows the result. We can see that the curves fit perfectly. Hence I validate the MFRONT file. It is ready to be used by AREVA.

5.2. BEHAVIOUR LAW

5.2.1. CLADDING CREEP LAW

AREVA needed primarily the Low Stress Creep laws for the clad, for a new study. This is the reason why these laws have been coded first. The Low Stress Creep in the clad can be divided into two creeps: the thermal creep and the irradiation creep. The thermal creep is also divided into two parts (the primary and the secondary creep).

\[ \varepsilon_{\text{creep}} = \varepsilon_{\text{therm}} + \varepsilon_{\text{irr}} \]

With: \( \varepsilon_{\text{therm}} = \varepsilon_{\text{th1}} + \varepsilon_{\text{th2}} \)

These equations are in an isotropic material and can be expressed as a flow rule. The utilised parser is IsotropicMisesCreep for all of them. The creation of some external parameters is required. They are the temperature, the time and the neutronic flux. Figure 26 shows the MFRONT file of the Primary Thermal Creep, whose equation is:

\[ \dot{\varepsilon} = A_1 \cdot \exp\left(-\frac{150}{T}\right) \cdot \exp\left(-B_3 \cdot t\right) \cdot \exp\left(A_2 \cdot \sigma_{eq}\right) \]

with \( T \), the temperature, \( t \) the time and \( \sigma_{eq} \) the equivalent stress.

```plaintext
@Parser    IsotropicMisesCreep;
@Library   M5ThermPrim;
@ExternalStateVariable real temp;
@StaticVariable real B3 = 1.0;
@StaticVariable real A2 = 0.1;
@StaticVariable real A1 = 0.001;
@LocalVariable real Eecr;
@LocalVariable real Eexp;
@InitLocalVariables{
    Eecr = exp(-150/(T+theta*dT));
    Eexp = exp(-B3*(temp+theta*dtemp));
}
@FlowRule{
    df_dseq = Eexp*Eecr*exp(A2*seq)*A2*A1;
    f = df_dseq/A2;
}
```

Figure 26: MFRONT file for the Primary Thermal Creep
The file is very similar to the Norton file showed in §IV. The parser is IsotropicMisesCreep and the law can be given as a flow rule. \texttt{@StaticVariable} gives the constant parameters of the problem. \texttt{@ExternalStateVariable} gives the external variables. Those are given by the simulation code, which will call the MFRONT file. Here it is the time (\texttt{temp}) and the temperature (\texttt{T}), which is implicitly given by MFRONT. The two other MFRONT files are given in the Appendix 5.

5.2.2. VALIDATION OF THE LAW

The tests of this law are done with CASTEM. Two simulations have been provided one in two dimensions and one in three dimensions. The 2D-mesh (Figure 27) is an annulus divided into quadrilateral elements. It represents the 2D section of a fuel rod. Internal and external pressure stresses (respectively 15.5MPa and 7MPa) are applied on the annulus, as shown in Figure 27. The 3D-mesh (Figure 28) is a quarter of an empty cylinder divided in cubic elements. It represents a part of the fuel rod’s cladding. The same pressures are applied on it. The simulations have been done over 6000 hours for the 2D-case and 1000 hours for the 3D case. The CASTEM code is given in Appendix 6.

![Figure 27: Mesh of the annulus (left) and pressure force applied on it (right). In red the internal Pressure, in pink the external](image)

![Figure 28: Mesh of the 3D-cylinder](image)
The results have been compared to the simulation of the same calculation done by GALILEO (Figure 29 and Figure 30), which integrates this law by its own solver.

**Figure 29:** Comparison of results for the Creep of the 2D-cladding between GALILEO (red) and CASTEM+MFRONT (blue)

**Figure 30:** Comparison of results for the Creep of the 3D-cladding between GALILEO (red) and CASTEM+MFRONT (blue)
The curves are exactly the same in both cases. So the MFRONT-creep-law is working perfectly and can be used by AREVA.

5.2.3. OTHER CLADDING LAWS

The previous law was an isotropic law and the cladding acts actually more as an orthotropic material. So it is still interesting to write them and to understand how the orthotropic behaviour works with MFRONT. I wrote two other MFRONT laws for the cladding. One is a normal orthotropic elastic law and the other one is for the orthotropic creep. The MFRONT files of those laws are given in Figure 31 and Figure 32.

```plaintext
@Parser  DefaultParser;
@Behaviour MS;
@Library  OrthoElas;

@OrthotropicBehaviour;
@RequireStiffnessTensor;

@Integrator{
    sig = D*(eto+deto);
}
```

**Figure 31: Mfront file of an orthotropic elastic law**

The main difference is the directive @OrthotropicBehaviour. It allows defining the orthotropic behaviour of the materials. It has three main effects:

- The law will require the orthotropic directions and three values for the Young modulus and the Poisson coefficient (one for each orthotropic direction).
- It will express the main input state variables (the stress tensor, the strain tensor and the increment strain tensor) in the orthotropic axis system.
- It will express the output stress tensor in the initial axis system.

The Hill tensor needs to be defined as well (in case of non-elastic behaviour). For that we use the 4-order-tensor type that can be found in the TFEL library. The parser is also different because we cannot use an equivalent strain law (and so a one dimension parser) anymore, as the behaviour is not isotropic.

These laws are also tested with CASTEM on the same 2D-example as the creep law ($\text{\$}$. The test is done with the isotropic coefficient as I have not obtained results with the orthotropic coefficients for comparison. The results are given in Figure 33 for the orthotropic creep and in Appendix 7 for the orthotropic elastic behaviour. As it is the same test as before, the CASTEM sources are not given. The small changes can be done without difficulty.
@Parser RungeKutta;
@Behaviour M5;
@Algorithm rk54;
@Library LowCreep;
@OrthotropicBehaviour;
@RequireStiffnessTensor;
@ExternalStateVariable real temp;
@StateVariable Stensor evp;
  evp.setGlossaryName("ViscoplasticStrain");
@StateVariable real p;
  p.setGlossaryName("EquivalentViscoplasticStrain");
@AuxiliaryStateVariable real seq;
  seq.setGlossaryName("HillStress");

@StaticVariable real B3 = 1.;
@StaticVariable real aD = 1.;
@StaticVariable real airr = 1.e-10;
@StaticVariable real ni = 5/3;
@StaticVariable real phi = 3.e12;
@StaticVariable real Qi = 10.;
@StaticVariable real A1 = 0.001;
@StaticVariable real A2 = 0.1;
@LocalVariable real H_F,H_G,H_H,H_L,H_M,H_N;
@LocalVariable real Eirr,B1,B2;
@Includes
  #include<TFEL/Material/Hill.hxx>
@InitLocalVariables
  H_F = 0.5;
  H_G = 1-H_F;
  H_H = 0.5;
  H_L = 1.5;
  H_M = 1.5;
  H_N = 1.5;
  H = hillTensor<N,real>(H_F,H_G,H_H,H_L,H_M,H_N);
  Eirr = airr*phi*exp(-T/Qi);
  B2 = A1*exp(-150/T)*exp(-B3*temp);
  B1 = aD*exp(-T/500);
}
@ComputeStress
  sig = D*eel;
}
@Derivative
  real sigeq = sqrt(sig|H*sig);
  Stensor n(0.);
  if(sigeq > 10.e-7){
    n = H*sig/sigeq;
  }
  dp =
    B1*exp(-0.01*sigeq)
    + Eirr*pow(sigeq,ni)
    + B2*exp(-A2*sigeq);
  devp = dp*n;
  deel = deto - devp;
}

Figure 32: Mfront file of an orthotropic creep law for the cladding
Figure 33: Comparison of results for the Orthotropic Creep of the 2D-cladding between GALILEO (red) and CASETM+MFRONT (blue)

The curves are the same between GALILEO and MFRONT. Therefore those MFRONT laws are validated. Even if some verifications remain to be done on the behaviour of non isotropic Hill tensor.

5.3. CONCLUSION ABOUT MFRONT

After this work on MFRONT, I used it regularly and therefore it seems fitting to render my observations considering MFRONT. I am now able to explain the advantages and answer the following questions: Does it work well? Is it easy to use? Is it efficient? These questions will be evaluated in this chapter.

The first point addresses the creation of new laws. The writing of MFRONT laws is initially difficult. One of the challenging parts is to choose the most efficient parser for a given law. Secondly the shape of the MFRONT law changes as function of the parser. For example the directive where the equation is given and the shape the equations are different according to the parser.

Those difficulties are nevertheless quickly overcome. At this point writing with MFRONT becomes simple and intuitive. It consists only in giving the equation of the law and the value of its parameters. Once the functioning of MFRONT (especially the functioning of the parsers) is clearly understood, the writing of the law becomes very fast and simple. Furthermore MFRONT is very precise and complete. Many parameters can be controlled with MFRONT (for example the directive @Theta can control the moment in the time step the calculation is done). Moreover on behalf on the selection of the parser, each law has the most efficient way to be calculated (in term of precision and running time). That is the main strength of MFRONT: MFRONT is able to solve each law in the best way.

Another interesting part is that the writer has the control of the law. He decides for example how his law can be modified: all the coefficients of the law can be fixed with the directive @StaticVariable or the user can have to choose the value of some coefficients declared with @Coef. He can also decide to add some interesting outputs with @ExternalStateVariable.

As previously shown, the results of MFRONT are very precise and correct, even with more complicated laws. It is running perfectly and it can be used, without any problems or doubts. The main goal is now to write all the laws with it. Concerning its efficiency, I have so far not been able to estimate the running time, but it can be done after the MFRONT laws are connected to GALILEO.
6. MFRONT LAWS IN GALILEO

GALILEO is the AREVA fuel rod code. It is written in FORTRAN. So I first call MFRONT library in FORTRAN and test it. Then I connect it with GALILEO.

6.1. MFRONT IN FORTRAN

6.1.1. MATERIAL PROPERTY

This part is very easy. The creation of the MFRONT file has to be done with the FORTRAN interface. Then the name of the function can be called in any FORTRAN file with the required input. With this function another function is created (called name of MFRONT function + “_CHECK_BOUNDS”). It allows verifying the respect of the bounds defined in the MFRONT file. Figure 34 gives an example of the structure.

```
PROGRAM TEST
  REAL*8 ZR_YOUNGMODULUS
  INTEGER*4 ZR_YOUNGMODULUS_CHECKBOUNDS
  REAL*8 Temp
  REAL*8 poro
  Poro = 0.1
  temp = 200.
  print,* "Young modulus = ", ZR_YOUNGMODULUS(temp,poro)
  print,* "CkeckBounds   = ", ZR_YOUNGMODULUS_CHECKBOUNDS (temp,poro)
END
```

Figure 34: The calling of MFRONT material property via FORTRAN

It is important to notice that the result of the property function is a “real*8” and the result of the checkbounds function is an “integer*4”. The value of the checkbounds function is 0, if the bounds of each parameter are respected. It is a negative number if the value of one of the parameter is out of the physical bounds (directive @PhysicalBounds). It is a positive number if the value of one of the parameter is out of the bounds of the law (directive @Bounds).

6.1.2. BEHAVIOUR LAW

This part was challenging. In point of fact, MFRONT creates UMAT-library for the behaviour law and UMAT is C-solver. So the point was to call a C-function with FORTRAN. That is a little touchy. The set of codes I wrote is composed of a C-program including subroutines, which called the UMAT-function and a FORTRAN program which called the C subroutines. An other unsuccessful method is given in Appendix 8.

6.1.2.1. THE UMAT FUNCTION

The prototype of UMAT function with MFRONT is shown in Figure 35. The UMAT prototype makes the calculation for one time step on one node of the mesh. It takes as inputs the properties of the materials (such as the Young-Modulus, Poisson coefficient ...), the total strains, the elastic strains, the temperature and other
external variables (such as the time, the fluence…) in the beginning of the time step. The increments of external variables, temperature and total strains are also required. Additionally the program needs the calculation condition. It is an integer which fixes the dimension of the calculation and its hypothesis (axisymmetric, plane strains …). For GALILEO it will be equal to 14 (1D, plane strains). The other possible values are given in Appendix 12. These inputs are found in the prototype with the word const before their declaration (which means they remain constant during the calling of the function).

The shear stress and the internal variables (such as elastic strains, the equivalent stress …) at the beginning of the time step are also input. They are not declared with the word const, because they are updated during the calculation (at the end of the time step). They are also actually part of the outputs of the function. The other outputs are the tangential matrix and an output number. This number is equal to zero if the calculation was running correctly, and to another integer if the calculation had some problems (Appendix 12).

Finally, the number have a typical kind, it is the UMATReal and the UMATInt. As matter of fact, the UMAT integers are rather touchy. If the computer we are working on is on 32bits, the integers are simple precision (int), but if the computer is on 64bits, they are double precision (long). The UMAT reals remain independent of the computer.

```c
typedef void (*UMATfctPtr)(const UMATInt *const, /* number of shear stress components */
                           const UMATReal *const, /*time increment */
                           const UMATReal *const, /*matrix of changing axis system */
                           UMATReal *const, /*tangential matrix */
                           const UMATReal *const, /*total strain tensor*/
                           const UMATReal *const, /*increment strain tensor */
                           const UMATReal *const, /*temperature*/
                           const UMATReal *const, /*temperature increment*/
                           const UMATReal *const, /*materials property*/
                           const UMATInt *const, /* number of materials property */
                           const UMATReal *const, /*external variables*/
                           const UMATReal *const, /*increment external variables */
                           UMATReal *const, /*internal variables */
                           const UMATInt *const, /* number of internal variables */
                           UMATReal *const, /*stress tensor*/
                           const UMATInt *const, /*calculation condition*/
                           UMATInt *const); /*output number*/
```

**Figure 35: MFRONT UMAT prototype**

### 6.1.2.2. THE CALL OF UMAT BY A C-PROGRAM

The principle of this program is to create two subroutines. The first one creates a pointer to the UMAT function of the MFRONT law. The second one uses the pointer and the required parameters of the function to call the UMAT function.

The C-program is given in Figure 36. In the beginning of the code, the needed includes are given. Following there is the definition of the UMAT types to be done. At that point the prototypes of UMAT function are defined (as show previous paragraph). We recognize then the name of the function given by MFRONT.

Then there is the function which creates the pointer (CreatePointer). This function needs in inputs the name of the behaviour of the law (it is the name given after the directive @Library of the MFRONT file + the name given after the directive @Behaviour) as a string and the length of this string. It gives the pointer to the UMAT function and a verification integer as outputs (this integer is equal to 0, if everything was
going well and -1 otherwise). The pointer is initially given in a union which contains a pointer and a UMAT integer. This union allows the pointer to have the required precision.
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

#ifdef GALILEO_32
typedef int  UMATInt;
typedef double UMATReal;
#else /* GALILEO_32 */
typedef long  UMATInt;
typedef double UMATReal;
#endif
typedef void (*UMATFctPtr)(const UMATInt  *const,
const UMATReal *const, UMATReal *const,
const UMATReal *const, const UMATReal *const,
const UMATReal *const, const UMATReal *const,
const UMATReal *const, const UMATReal *const,
const UMATReal *const, const UMATInt  *const,
const UMATReal *const, const UMATInt  *const,
const UMATReal *const, const UMATInt  *const,
const UMATReal *const, const UMATInt  *const,
const UMATReal *const, const UMATInt  *const,
const UMATReal *const, const UMATInt  *const,
const UMATReal *const, const UMATInt  *const);

void umatlowcreepflowm51d(const UMATInt *const, const UMATReal *const,
const UMATReal *const, const UMATReal *const,
const UMATReal *const, const UMATInt  *const,
const UMATReal *const, const UMATInt  *const,
const UMATReal *const, const UMATInt  *const,
const UMATReal *const, const UMATInt  *const);

void CreatePointer_(UMATInt  *ptr, const char* const behaviour,
UMATInt* ret, const int behaviour_length)
{
union {
    UMATInt  ivalue;
    UMATFctPtr fptr;
} fct;
if(strcmp("lowcreepflowm51d",behaviour,behaviour_length)==0){
    fct.fptr = umatlowcreepflowm51d;
} else {
    *ret = -1;
    return;
}
*ptr = fct.ivalue;
*ret = 0;
}

void callmechanicalbehaviour_(UMATInt  *ptr,
const UMATReal *const NTENS,
const UMATReal *const DTIME, const UMATReal *const DROT,
const UMATReal *const DDSDDE, const UMATReal *const STRAN,
const UMATReal *const DSTRAN, const UMATReal *const TEMP,
const UMATReal *const DTEMP , const UMATReal *const PROPS,
const UMATInt  *const NPROPS, const UMATReal *const PREDEF,
const UMATReal *const DPRED, UMATReal *const STATEV,
const UMATInt  *const NSTATV, UMATReal *const STRESS,
const UMATInt  *const NDI, UMATInt  *const KINC)
{
union {
    UMATInt  ivalue;
    UMATFctPtr fptr;
} fct;
    fct.ivalue = *ptr;
    (fct.fptr)(NTENS,DTIME,DROT,DDSDDE,STRAN,DSTRAN,TEMP,DTEMP,PROPS,NPROPS,
PREDEF,DPRED,STATEV,NSTATV,STRESS,NDI,KINC);
}

Figure 36: C-program calling the MFRONT law
Adding further laws is simple. We just have to add a prototype of the new UMAT function and an `else if` condition, which compares the string `behaviour` with the new name. At the end the subroutine `callmechanicalbehaviour` is created. It is a classic program: it takes the pointer and the key parameters required for the UMAT function as inputs. The outputs are the output parameters of the UMAT function. The union is also used for the same reason.

6.1.2.3. THE CALL WITH FORTRAN

The FORTRAN program (Figure 37) is only an example. It explains how the UMAT function is called. To begin with the variables which will be used in the UMAT function (stress tensor, strains ...) are declared, such as two integers for the pointer. The subroutine `CreatePointer` is then called with the name of the law. At the end the subroutine `CallMechanicalBehaviour` is called, with the pointer we just got and the good parameters of the law. For a real utilization, we have to initialize the parameters and add loops on the number of time step and the number of elements. (A complete example is given in Appendix 13).

```fortran
program Test
  INTEGER ptr;
  INTEGER ret;
  INTEGER NTENS
  REAL*8 DTIME
  REAL*8 DROT
  REAL*8 DDSDDE
  REAL*8 STRAN
  REAL*8 DSTRAN
  REAL*8 TEMP
  REAL*8 DTEMP
  REAL*8 PROPS
  INTEGER NPROPS
  REAL*8 PREDEF
  REAL*8 DPRED
  REAL*8 STATEV
  INTEGER NSTATV
  REAL*8 STRESS
  INTEGER NDI
  INTEGER KINC
  CALL CreatePointer(ptr,"norton",ret)
  print *, "ptr : ", ptr
  CALL CALLMECHANICALBEHAVIOUR(PTR,NTENS,DTIME,DROT,DDSDDE,STRAN,
& DSTRAN,TEMP,DTEMP,PROPS,NPROPS,PREDEF,DPRED,STATEV,
& NSTATV,STRESS,NDI,KINC)
end program Test
```

**Figure 37: The calling of UMAT via FORTRAN**

It is interesting to notice that the C-functions end with an underscore (`callmechanicalbehaviour_ ...`) and they are called without it in the FORTRAN (`CALL callmechanicalbehaviour(...)`) program. The reason is simple: the underscore in the C-function means that the functions are able to be called with FORTRAN.
6.1.2.4. THE COMPILATION AND THE LINKAGE

The compilation of these programs and the linkage with the library are rather complicated. A specific makefile is required for it which is given in Appendix 10. Two main choices are required for the compilation: a dynamic or static compilation of MFRONT and an internal or external call for the law. The information for the makefile is given in the file make.rules (also given in Appendix 9).

The advantage of the static compilation of MFRONT for GALILEO is that the executable contains everything he needs to run. It allows that every GALILEO user have the same version of the MFRONT laws. This is not the case with a dynamic compilation. In point of fact, with a dynamic compilation, GALILEO searches the last version of the law, which could have been modified. So it does not give the required guarantee for the customer. Nevertheless, in case of an updating of MFRONT law, all the programs (and so GALILEO) should be compiled again. This is rather complicated.

Finally, I decided to make a dynamic compilation in the beginning. In point of fact it is now a development process. So many points are expected to be modified. A static compilation will occur when MFRONT works perfectly with GALILEO and all the laws are written.

For an internal call the function knows directly where it can find the library whereas an external use the dlopen function, which search the library in the LibraryPATH. The external call is easier for the development part, so I choose this one for the moment.

This makefile calls another makefile which allows the compilation of the MFRONT file. It is also given in Appendix 11. As this is specifically done for GALILEO, the name of the library begins by libGalileo.

6.1.2.5. DIAGRAM

A diagram summarizes the links for the calling of MFRONT.

![Diagram explaining the link between the different codes](image-url)

**Figure 38: Diagram explaining the link between the different codes**
6.2. TEST OF THE CALL OF THE MFRONT LAW WITH FORTRAN

6.2.1. MATERIAL PROPERTY

The test for the material law with FORTRAN is given in Figure 39. It has been compared with an EXCEL calculation. It shows that the MFRONT works perfectly with FORTRAN. It is ready to be used with GALILEO.

![Figure 39: Comparison between MFRONT law with FORTRAN and EXCEL](image)

6.2.2. BEHAVIOUR LAW

The set-up for the example is the following: there is a one dimension rod, with only two nodes. The first node is fixed and there is an imposed strain on the second node along the rod. This displacement increases from 0 to 1% during the first ten seconds and is constant then. The simulation is calculated on 100s, each time step is one second. A diagram of the test and a curve of the imposed strain are given in Figure 40.

![Figure 40: Diagram of the loading of the rod. In blue the strain loading (node 2) and in green the fixed node (node 1).](image)

There are two FORTRAN codes (given in Appendix 13), which call the MFRONT law. The results are compared with a CASTEM result. The code is also given in Appendix 14. As the one dimension calculation is complicated with CASTEM, I compare it to a thin 2D-plate, which result is supposed to be the same.
As the strains are imposed, I compared the result of the stress along the rod for the second nodes. The results are given in Figure 41. The curves are fitting perfectly. Thus I can validate the C/FORTRAN codes to call MFRONT behaviour law and the MFRONT creep law is available to be added in GALILEO.

6.3. TEST IN GALILEO

6.3.1. MATERIAL PROPERTY

I wrote the laws of the Young modulus and the Poisson ratio, for the cladding and the fuel. I implemented them in GALILEO. I run GALILEO once using the MFRONT law and a second with the coefficients calculated with GALILEO.
I compare different parameters (elastic strain, the Young Modulus and the Poisson ratio) for both methods. The results for the fuel are given in Figure 42, Figure 43 and Figure 44, and for the cladding in Appendix 15. It is a simple case, which is supposed to show interesting variations for the Poisson ratio and the Young modulus.

![Graph of Poisson ratio comparison](image1)

**Figure 43: Comparison of Poisson ratio with GALILEO (red) and MFRONT (blue)**

![Graph of Young modulus comparison](image2)

**Figure 44: Comparison of the Young modulus with GALILEO (red) and MFRONT (blue)**

The results are exactly the same. So I validate the good implementation of MFRONT in GALILEO. Nevertheless it could be interesting also to test an example with a very influent parameter to show that MFRONT is also very efficient with precise calculation. This is the case with the thermal conductivity. I test it with a very energy-producing example. Results are given in Figure 45, Figure 46 and Figure 47.
Figure 45: Comparison of the thermal conductivity with GALILEO (red) and MFRONT (blue)

Figure 46: Comparison of the temperature with GALILEO (red) and MFRONT (blue)
Figure 47: Comparison of thermal strain in radial direction with GALILEO (red) and MFRONT (blue)

The curves are exactly the same as well. So the precision of MFRONT can be insured.

6.3.2. BEHAVIOUR LAW

The case of the behaviour law is far more complicated. As point of fact the result of each law is linked to the result of the other ones. After running the calculation for the different laws, the calculation has to be compared with the other results and to refine, usually with a Newton-Raphson method. As it was a complex problem, I was not able to solve it during this master thesis. Nevertheless, the principle of adding the creep law in GALILEO is quite easy. The file `efluag_gear.f90` of GALILEO has to be modified, adding the creep law as seen §6.1.2.3. Then the convergence method has to be added.

Figure 48 gives a diagram explaining the way a finite elements codes works and where MFRONT is really applied.
CONCLUSION

The aim of this thesis was to discover MFRONT, to test it and to study the possibility of using it in AREVA (especially with CASTEM). I tested the example laws given by the CEA. Then I wrote some AREVA laws and tested them, with CASTEM and in FORTRAN. Finally I work to connect them with GALILEO. The main target of this report is a synthesis of the use of MFRONT and a discussion of its efficiency.

The writing of MFRONT is rather intuitive. As point of fact the writing of the law itself simply consists of writing it in the same way as on a paper. The declaration of the different variables and parameters is also easy and let an important degree of freedom (for example it can be chosen or not to give the control to the user of the law). The only difficult aspect is to know the best parser for the law. MFRONT is very efficient because it allows to select the best way to solve an equation. But this best way has to be known. So it is interesting and important to know the advantages of each parser to use MFRONT the most efficiently.

The use of MFRONT with different codes is simple and efficient. Even if it is little less intuitive for the behavior laws, once the method is seen and understood, it is rather easy. One of the main advantages of MFRONT is that the laws are clearly identified and managed with a clever way. The whole code is much more readable and slight. The laws are also clearly stored as they are part of a large library. It simplifies its use. Another important aspect is the running time. In point of fact it is supposed to be far better with MFRONT. I was able to prove it, at least with CASTEM. I can say with this calculation with CASTEM that the running time seems better. I have a gain of an average of 30%. This seems great especially as the measured running time contains the whole codes and not only the part of the calculation of the law. So this can be a great improvement for the calculation of fuel rod behavior.

This is the first work for MFRONT in AREVA. Before using it successfully in AREVA, it remains some important things to do. First, all the laws have to be written in MFRONT, with the good parser. It also means that some tests have to be done, to understand the advantages of each parser. Their validity has to be tested and then the laws have to be added in GALILEO.
BIBLIOGRAPHY


## Appendix 1 - The different parsers

<table>
<thead>
<tr>
<th>Parser</th>
<th>Name of the equation directive</th>
<th>Name of the Algorithm</th>
<th>Utilization of the parser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material property</td>
<td>MaterialLaw</td>
<td>@Function</td>
<td>If all the other parser doesn’t work.</td>
</tr>
<tr>
<td>DefaultParser</td>
<td></td>
<td>@Integrator</td>
<td>None</td>
</tr>
<tr>
<td>RungeKutta</td>
<td></td>
<td>@Derivative</td>
<td>This parser should be used when the Implicit parser does not work (for example if we are not able to calculate the tangential matrix).</td>
</tr>
<tr>
<td>Implicit</td>
<td></td>
<td>@Integrator</td>
<td>NewtonRaphson, Broyden, Broyden2</td>
</tr>
<tr>
<td>IsotropicMisesCreep</td>
<td></td>
<td>@FlowRules</td>
<td>Parser for isotropic plastic or viscoplastic laws. They are the best for flow function cases.</td>
</tr>
<tr>
<td>IsotropicStrain</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HardeningMisesCreep</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IsotropicPlasticMisesFlow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MultipleIsotropicMisesFlows</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix 2- MFRONT-file of the ImplicitOrthotropicCreep

```plaintext
@Parser Implicit;
@Behaviour ImplicitOrthotropicCreep;

@OrthotropicBehaviour;
@RequireStiffnessTensor;
@Theta 1.;

@StateVariable real p;
p.setGlossaryName("EquivalentViscoplasticStrain");
@AuxiliaryStateVariable real seq;
seq.setGlossaryName("HillStress");
@LocalVariable real H_F;
@LocalVariable real H_G;
@LocalVariable real H_H;
@LocalVariable real H_L;
@LocalVariable real H_M;
@LocalVariable real H_N;
@InitLocalVariables{
    H_F = 0.5;
    H_G = 1-H_F;
    H_H = 0.5;
    H_L = 1.5;
    H_M = 1.5;
    H_N = 1.5;
}

@ComputeStress{
    sig = D*eel;
}

@Integrator{
    const real A = 8.e-67;
    const real E = 8.2;
    const st2tost2<N,real> H = hillTensor<N,real>(H_F,H_G,H_H,
                                              H_L,H_M,H_N);
    const real sigeq = sqrt(sig|H*sig);
    const real tmp = A*pow(sigeq,E-1.);
    real inv_sigeq(0);
    Stensor n(0.);
    n = (H*sig)*inv_sigeq;
    feel += dp*n-deto;
    fp = tmp*sigeq*dt;
    dfeel_ddeel += theta*dp*(H-(n^n))*D*inv_sigeq;
    dfeel_ddp = n;
    dfp_ddeel = -theta*tmp*E*dt*(n|D);
}

@UpdateAuxiliaryStateVars{
    const st2tost2<N,real> H = hillTensor<N,real>(H_F,H_G,H_H,
                                              H_L,H_M,H_N);
    seq = sqrt(sig|H*sig);
}
```

Appendix 3 - The CASTEM code for testing the material property

```plaintext
* fichier : elas2.dgibi
OPTION DIME 3 ELEM TRI3;
OPTION ECHO 1 ;
TITRE 'CYLINDRE PINCE A BORDS ENCASTRES' ;
TEMS ;
*-----------------------------------------------------
* GEOMETRY
P1=7 0 0 ;P2=0 0 7 ;C1=0 0 0 ;VECT=0 70 0 ;
LIG1=C 6 P1 C1 P2 ;SURF=LIG1 TRAN 30 VECT ;
LIG1 LIG2 LIG3 LIG4=COTE SURF ;P3=LIG2 POIN FINA ;
CHPZ = COOR 2 SURF ;
CHP4 = 5*CHPZ + 400 ;
*-----------------------------------------------------
* MODEL DEFINITION and MFRONT LAW
*-----------------------------------------------------
TLoi = 'TABLE' ;
TLoi.'LIBRAIRIE' = 'src/libCastemSiC.so' ;
TLoi.'MODELE' = 'SiC_YoungModulus_GFR' ;
TLoi.'VARIABLES' = 'MOTS' 'T' 'PORO' ;
MOD1=MODL SURF MECANIQUE ELASTIQUE DKT ;
cht = 'CHANGER' CHP4 'COMP' 'T' ;
vel = ('CHANGER' 'CHAM' cht MOD1) 'ET' (MANUEL 'CHML' MOD1 'PORO' 0.2) ;
chaY = MANUEL 'CHML' MOD1 'Y' TLoi ;
chy = 'VARI' 'NUAG' MOD1 chaY vel ;
MAT1=MATR MOD1 YOUN TLoi NU 0.3 ;
EPEE=CARB MOD1 EPAI 0.1 ;
MAT1=MAT1 ET EPEE ;
RIG1=RIGI MOD1 MAT1 ;
*-----------------------------------------------------
* BOUNDARY CONDITIONS
ENC1=BLOQ LIG4 UZ ;ENC2=BLOQ LIG4 RX ;ENC3=BLOQ LIG4 RY ;
ENC4=BLOQ LIG2 UX ;ENC5=BLOQ LIG2 RX ;ENC6=BLOQ LIG2 RZ ;
ENC7=BLOQ LIG3 UY ;ENC8=BLOQ LIG3 RX ;ENC9=BLOQ LIG3 RZ ;
ENC10=BLOQ LIG1 DEPL ROTA ;
ENC11=ENC1 ET ENC2 ET ENC3 ET ENC4 ET ENC5 ET ENC6 ET
ENC7 ET ENC8 ET ENC9 ET ENC10 ;
F1=0 0 -25000 ;
EFOR=FORC F1 P3 ;
RIG2=RIG1 ET ENC11 ;
*-----------------------------------------------------
* RESULTS
DE1=RESOU RIG2 EFOR ;
D1X = EXTR DE1 UX P1 ;
D3Z = EXTR DE1 UZ P3 ;
defo1 = DEFO SURF DE1 0.0 ;
defo2 = DEFO SURF DE1 200 ROUG ;
trac (defo1 'ET' defo2) ;
SAUT PAGE ;
MESS ' DEPLACEMENT EN X DE LA BASE VALEUR THEORIQUE : 0.00' ;
MESS ' DEPLACEMENT EN X DE LA BASE VALEUR CALCULEE : ' D1X ;
SAUT 2 LIGNE ;
MESS ' DEPLACEMENT EN Z THEORIQUE : -1.140' ;
MESS ' DEPLACEMENT EN Z CALCULE : ' D3Z ;
TEMS ;
FIN ;
```
Appendix 4 - The CASTEM code for testing the first MFRONT law. Traction-Loading. Norton Law

* fichier : creep01_traYY.dgibi
* 0 - Maillage
* PLAQUE CARREE COTE : 1 m EPAISSEUR : 1 cm

opti dime 3 elem cub8 mode tridim ;

l1 = 1.0 ;
l2 = 1.0 ;
ep0 = 0.01 ;

p01 = 0.0 0.0 0.0 ;
p02 = ep0 0.0 0.0 ;
p03 = ep0 12 0.0 ;
p04 = 0.0 12 0.0 ;

p10 = 0.0 0.0 l1 ;
p20 = ep0 0.0 l1 ;
p30 = ep0 12 l1 ;
p40 = 0.0 12 l1 ;

li01 = 'DROIT' 1 p01 p02 ;
li02 = 'DROIT' 1 p02 p03 ;
li03 = 'DROIT' 1 p03 p04 ;
li04 = 'DROIT' 1 p04 p01 ;

li10 = 'DROIT' 1 p10 p20 ;
li20 = 'DROIT' 1 p20 p30 ;
li30 = 'DROIT' 1 p30 p40 ;
li40 = 'DROIT' 1 p40 p10 ;

sur01 = 'DALL' li01 li02 li03 li04 ;
sur10 = 'DALL' li10 li20 li30 li40 ;
mail1 = sur01 VOLU 1 sur10 ;

li2 = 'DROIT' 1 p02 p20 ;
li3 = 'DROIT' 1 p20 p10 ;
li4 = 'DROIT' 1 p10 p01 ;
sbloq = 'DALL' li01 li2 li3 li4 ;
li5 = 'DROIT' 1 p04 p03 ;
li6 = 'DROIT' 1 p03 p30 ;
li7 = 'DROIT' 1 p30 p40 ;
li8 = 'DROIT' 1 p40 p04 ;
schar = 'DALL' li5 li6 li7 li8 ;
ELIM (mail1 ET sbloq ET schar) 1.e-5 ;
TRAC mail1 ;

* 1.1 the first model with CASTEM
mo_flupo = MODE mail1 MECANIQUE ELASTIQUE ISOTROPE FLUAGE POLYNOMIAL ;
densiu = 0.95 ;
dgrain = 8.0E-6 ;
tuo2 = 1500.0 ;
tauxfisk = 1.5E19 ;

xyoun = (2.2693E11 - (1.5399E7*tuo2)) - (9.35973E3 * (tuo2**2.0)) ;
xcis = (8.583E10 - (5.157E6*tuo2)) - (3.747E3 * (tuo2**2.0)) ;
xnu = (xyoun/(2.0*xcis)) - 1.0 ;
xrho = 10950.0 ;
xalph = ( (1.1833E-5 - (5.013E-9*tuo2)) + (3.756E-12 * (tuo2**2.0)) )
- (6.125E-17 * (tuo2**3.0)) ;

xsmax = xyoun / 1000.0 ;
xf0 = 0;
xf1 = 8.0e-67;
xf2 = 8.2;
xf3 = 0.0;
xf4 = 0.0;
xf5 = 0.0;
xf6 = 0.0;

ma_flupo = MATE mo_flupo 'YOUN' xyoun 'NU ' xnu   'RHO ' xrho
 'ALPH' xalph
 'SMAX' xsmax 'AF0 ' xaf0 'AF1 ' xaf1
 'AF2 ' xaf2 'AF3 ' xaf3 'AF4 ' xaf4
 'AF5 ' xaf5 'AF6 ' xaf6 ;

c11 = BLOQ mai11 'UX' 'UZ' ;
c12 = BLOQ sbloq 'UY' ;
c10 = c11 ET c12 ;

depmax = 5.0E8 ;
dep0 = FORCE schar 'FX' 0.0 'FY' depmax 'FZ' 0.0 ;
ev0 = EVOL MANU temps (PROG 0.0 60. 3600.0)
 y (PROG 0.0 1.0  1.0 ) ;
cha0 = CHAR MECA dep0 ev0 ;

pro0 = PROG 0.0  PAS  30. 3600.;
pro1 = PROG 0.0 30.  60. 180. PAS 180. 3600.;

tabflup = TABLE ;
tabflup.'MODELE' = mo_flupo ;
tabflup.'CHARACTERISTIQUES' = ma_flupo ;
tabflup.'CHARGEMENT' = cha0 ;
tabflup.'BLOCAGES_MECANIQUES' = c10 ;
tabflup.'TEMPS_CALCULES' = pro0 ;
tabflup.'TEMPS_SAUVES' = pro1 ;
TMASAU=table;
tabflup . 'MES_SAUVEGARDES'=TMASAU;
TMASAU . 'DEFTO'=VRAI;
TMASAU . 'DEFIN'=VRAI;
TEMPS 'ZERO' ;
PASAPAS tabflup ;
CPUint = TEMPS 'NOEC' ;

* 2 - 2d Model MFRONT
Tloi = 'TABLE';
Tloi.'MODELE'='umatnortonorton';
Tloi.'LIBRAIRIE' = 'libUmatNorton.so';
coe12D = 'MOTS' 'YOUN' 'NU' 'RHO' 'ALPH';
stav2D = 'MOTS' 'EEXX' 'EEYY' 'EEZZ' 'EEXY' 'EEXZ' 'EEYZ' 'EEXZ' 'EEYX' 'P';
para2D = 'MOTS' ;
mo_util = MODE mai11 'MECANIQUE' 'ELASTIQUE' 'NON_LINEAIRE'
 'UTILISATEUR' 'DESC_LOI' Tloi
 'C_MATERIAU' coe12D
 'C_VARINTER' stav2D
 'PARA_LOI' para2D;
ma_util = MATE mo_util 'YOUN' xyoun 'NU ' xnu 'RHO ' xrho 
'ALPH' xalph ;

tabutil = TABLE ;
tabutil.'MODELE' = mo_util ;
tabutil.'CARACTERISTIQUES' = ma_util ;
tabutil.'CHARGEMENT' = cha0 ;
tabutil.'BLOCAGES_MECAJNIQUES' = c10 ;
tabutil.'TEMPS_CALCULES' = pro0 ;
tabutil.'TEMPS_SAUVES' = pro1 ;
TMASAU=table ;
tabutil. 'MES_SAUVESGEDSES'=TMASAU ;
TMASAU .'DEFTO'=VRAI ;
TMASAU .'DEFIN'=VRAI ;
TEMPS 'ZERO' ;
PASAPAS tabutil ;
CPUext = TEMPS 'NOEC' ;

*--------------------------------------------------------------------------
<table>
<thead>
<tr>
<th>3 - Post-treatment :</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbgau = 8 ;</td>
</tr>
<tr>
<td>netat = DIME (tabflup.'TEMPS') ;</td>
</tr>
<tr>
<td>ietat = -1 ;</td>
</tr>
<tr>
<td>'OUBLIER' tabflup 'DEFORMATIONS' ;</td>
</tr>
<tr>
<td>'OUBLIER' tabutil 'DEFORMATIONS' ;</td>
</tr>
<tr>
<td>ietat = -1 ;</td>
</tr>
<tr>
<td>tabflup. 'DEFORMATIONS' = 'TABLE' ;</td>
</tr>
<tr>
<td>tabutil. 'DEFORMATIONS' = 'TABLE' ;</td>
</tr>
<tr>
<td>REPETER BECART netat ;</td>
</tr>
<tr>
<td>ietat = ietat + 1 ;</td>
</tr>
<tr>
<td>tabflup. 'DEFORMATIONS'. ietat = 'EPSI' (tabflup.'MODELE') (tabflup.'DEPLACEMENTS'. ietat) ;</td>
</tr>
<tr>
<td>tabutil. 'DEFORMATIONS'. ietat = 'EPSI' (tabutil.'MODELE') (tabutil.'DEPLACEMENTS'. ietat) ;</td>
</tr>
<tr>
<td>'FIN' BECART ;</td>
</tr>
<tr>
<td>ifin = netat-1 ;</td>
</tr>
<tr>
<td>epyycmax = 0.0 ;</td>
</tr>
<tr>
<td>igau = 0 ;</td>
</tr>
<tr>
<td>REPETER BEPYYFC nbgau ;</td>
</tr>
<tr>
<td>igau = igau + 1 ;</td>
</tr>
<tr>
<td>epyy = EXTR (tabflup.'DEFORMATIONS'.ifin) 'EPYY' 1 1 igau ;</td>
</tr>
<tr>
<td>SI (epyy &gt; epyycmax) ; epyycmax = epyy ; FINSI ;</td>
</tr>
<tr>
<td>FIN BEPYYFC ;</td>
</tr>
<tr>
<td>epyyumax = 0.0 ;</td>
</tr>
<tr>
<td>igau = 0 ;</td>
</tr>
<tr>
<td>REPETER BEPYYFU nbgau ;</td>
</tr>
<tr>
<td>igau = igau + 1 ;</td>
</tr>
<tr>
<td>epyy = EXTR (tabutil.'DEFORMATIONS'.ifin) 'EPYY' 1 1 igau ;</td>
</tr>
<tr>
<td>SI (epyy &gt; epyyumax) ; epyyumax = epyy ; FINSI ;</td>
</tr>
<tr>
<td>FIN BEPYYFU ;</td>
</tr>
<tr>
<td>Message 'Deformation totale finale (maxi sur points de Gauss)' ;</td>
</tr>
<tr>
<td>Message 'EPYY max avec modele interne = ' epyycmax ;</td>
</tr>
<tr>
<td>Message 'EPYY max avec modele externe = ' epyyumax ;</td>
</tr>
<tr>
<td>Message ;</td>
</tr>
<tr>
<td>elyycmax = 0.0 ;</td>
</tr>
<tr>
<td>igau = 0 ;</td>
</tr>
</tbody>
</table>
REPETER BEIYYFC nbgau;
  igau = igau + 1;
  eiyy = EXTR (tabflup.'DEFORMATIONS_INELASTIQUES'.ifin)
   'EIYY' 1 l igau;
  SI (eiyy > eiyyymax) ; eiyyymax = eiyy ; FINSI;
FIN BEIYYFC;
  eiyyumax = 0.0 ;
REPETER BEIYYFU nbgau;
  igau = igau + 1;
  epyy = EXTR (tabutil.'DEFORMATIONS'.ifin)
   'EPTY' 1 l igau;
  eeyy = EXTR (tabutil.'VARIABLES_INTERNES'.ifin)
   'EEYY' 1 l igau;
  eixx = epyy - eeyy;
  SI (eeyy > eiyyumax) ; eiyyumax = eeyy ; FINSI;
FIN BEIYYFU;
Message 'Deformation de fluage finale (maxi sur points de Gauss)'
Message 'EIYY max avec modele interne = ' eiyyymax
Message 'EIYY max avec modele externe = ' eiyyumax
smyyymax = 0.0 ;
igau = 0 ;
REPETER BSMYYFC nbgau;
  igau = igau + 1;
  smyy = EXTR (tabflup.'CONTRAINTES'.ifin)
   'SMYY' 1 l igau;
  SI (smyy > smyyymax) ; smyyymax = smyy ; FINSI;
FIN BSMYYFC;
smyyumax = 0.0 ;
igau = 0 ;
REPETER BSMYYFU nbgau;
  igau = igau + 1;
  smyy = EXTR (tabutil.'CONTRAINTES'.ifin)
   'SMYY' 1 l igau;
  SI (smyy > smyyymax) ; smyyymax = smyy ; FINSI;
FIN BSMYYFU;
Message 'Contrainte finale (maxi sur points de Gauss)'
Message 'SMYY max avec modele interne = ' smyyymax
Message 'SMYY max avec modele externe = ' smyyumax
Message ;
peivyyu = 'PROG';
peivyyu = 'PROG';
peivyyu = 'PROG';
peivzzu = 'PROG';
psigu = 'PROG';
psigzu = 'PROG';
psigxu = 'PROG';
psigtu = 'PROG';
peivyyf = 'PROG';
peivvyxf = 'PROG';
peivyyf = 'PROG';
peivzzf = 'PROG';
psigf = 'PROG';
psigtf = 'PROG';
netat = DI ME (tabutil.'TEMPS') ;
metat = netat-1;
ietat = -1;
getat = ietat+1;
REPETER BECART netat;
  ietat = ietat + 1;
sigu = tabutil.'CONTRAINTES'.ietat;
sigtu = EXTR (tabutil.'CONTRAINTES'.ietat) 'SMYY' 1 l l;
epxxu = EXTR (tabutil.'DEFORMATIONS'.ietat) 'EPXX' 1 1 1;
epyyu = EXTR (tabutil.'DEFORMATIONS'.ietat) 'EPYY' 1 1 1;
epzzu = EXTR (tabutil.'DEFORMATIONS'.ietat) 'EPZZ' 1 1 1;
epxyu = EXTR (tabutil.'DEFORMATIONS'.ietat) 'GAXY' 1 1 1;
psigtu = psigtu 'ET' ('PROG' sigtu);
peivxxu = peivxxu 'ET' ('PROG' epxxu);
peivyyu = peivyyu 'ET' ('PROG' epyyu);
peivzzu = peivzzu 'ET' ('PROG' epzzu);
peivxyu = peivxyu 'ET' ('PROG' epxyu);
sigf = tabflup.'CONTRAINTES'.ietat;
sigtf = EXTR (tabflup.'CONTRAINTES'.ietat) 'SMYY' 1 1 1;
epxxf = EXTR (tabflup.'DEFORMATIONS'.ietat) 'EPXX' 1 1 1;
epyyf = EXTR (tabflup.'DEFORMATIONS'.ietat) 'EPYY' 1 1 1;
epzzf = EXTR (tabflup.'DEFORMATIONS'.ietat) 'EPZZ' 1 1 1;
epxyf = EXTR (tabflup.'DEFORMATIONS'.ietat) 'GAXY' 1 1 1;
psigtf = psigtf 'ET' ('PROG' sigtf);
peivxxf = peivxxf 'ET' ('PROG' epxxf);
peivyyf = peivyyf 'ET' ('PROG' epyyf);
peivzzf = peivzzf 'ET' ('PROG' epzzf);
peivxyf = peivxyf 'ET' ('PROG' epxyf);

'FIN' BECART;

evsigtu = 'EVOL' 'ROUGE' 'MANU' 'TEMPS' pro1 'SMYY' psigtu;
evxxu = 'EVOL' 'ROUGE' 'MANU' 'TEMPS' pro1 'EIPRR' peivxxu;
evyyu = 'EVOL' 'ROUGE' 'MANU' 'TEMPS' pro1 'EIPTT' peivyyu;
evzzu = 'EVOL' 'ROUGE' 'MANU' 'TEMPS' pro1 'EIPZZ' peivzzu;
evxyu = 'EVOL' 'ROUGE' 'MANU' 'TEMPS' pro1 'EIPXY' peivxyu;
evsigtf = 'EVOL' 'BLEU' 'MANU' 'TEMPS' pro1 'SMYY' psigtf;
evxf = 'EVOL' 'BLEU' 'MANU' 'TEMPS' pro1 'EIPXX' peivxxf;
evyyf = 'EVOL' 'BLEU' 'MANU' 'TEMPS' pro1 'EIPYY' peivyyf;
evzzf = 'EVOL' 'BLEU' 'MANU' 'TEMPS' pro1 'EIPZZ' peivzzf;
evxyf = 'EVOL' 'BLEU' 'MANU' 'TEMPS' pro1 'EIPXY' peivxyf;
*evsf = 'EVOL' 'BLEU' 'MANU' 'TEMPS' pro0 'SIGEQ' psigf;
croi = 'TABLE';
croi.1 = 'MARQ CROI';

'DESSIN' (evsigtf 'ET' evsigtu) croi;
'DESSIN' (evxf 'ET' evxxu) croi;
'DESSIN' (evyyf 'ET' evyyu) croi;
'DESSIN' (evzzf 'ET' evzzu) croi;
'DESSIN' (evxyf 'ET' evxyu) croi;
metat = metat-1;
DEF0 = DEFO mail1 tabutil. 'DEPLACEMENTS'.metat 0. 'BLEU';
DEF1 = DEFO mail1 tabutil. 'DEPLACEMENTS'.metat 'ROUGE';
DEF2 = DEFO mail1 tabflup. 'DEPLACEMENTS'.metat 'ROSE';
TRAC (DEF1 'ET' DEF0);
TRAC (DEF2 'ET' DEF0);
TRAC (DEF1 'ET' DEF2);

FIN ;
Appendix 5 - MFRONT File for the secondary thermal creep and the irradiation creep

@Parser IsotropicMisesCreep;
@Behaviour Creep;
@Author Julien;
@Date 23/11/06;
@Library irradiation;

@StaticVariable real aD = 1.;

@LocalVariable real Eth;
@InitLocalVariables{
   Eth = aD*exp(-(T+theta*dT)/500);
}
@FlowRule{
   df_dseq = Eth*exp(1.e-2*seq);
   f = df_dseq*100;
}

@Parser IsotropicMisesCreep;
@Behaviour Creep;
@Author Julien;
@Date 23/11/06;
@Library thermique;

@StaticVariable real air = 1.e-10
@StaticVariable real ni = 5/3;
@StaticVariable real phi = 3.e12;
@StaticVariable real Qi = 10.;

@LocalVariable real Eirr;
@InitLocalVariables{
   Eirr = air*phi*exp(-(T+theta*dT)/Qi);
}
@FlowRule{
   df_dseq = Eirr*ni*pow(seq,ni-1);
   f = seq*df_dseq/ni;
}
OPTION DIME 2 ELEM QU8;
OPTION MODE PLAN DEFO;
*DEFINITION DE LA GEOMETRIE
*---------------------------------------------------------------------
**
**  r1 = 4.745E-3;
r11 = -1.*r1;
r2 = 4.12e-3;
r22 = -1.*r2;
T0 = 643.;

n11 = 20;
n1 = 3*n11/2;
n2 = 5;
O = 0. 0.;

A1 = POINTCYL r1 -45.;
B1 = POINTCYL r1 90.;
C1 = POINTCYL r1 -135.;
C1A = CERC n1 B1 O A1;
C1B = CERC n1 B1 O C1;
C1C = CERC n11 C1 O A1;
A2 = POINTCYL r2 -45.;
B2 = POINTCYL r2 90.;
C2 = POINTCYL r2 -135.;
C2A = CERC n1 A2 O B2;
C2B = CERC n1 B2 O C2;
C2C = CERC n11 C2 O A2;
LA = DROIT n2 A1 A2;
LB = DROIT n2 B2 B1;
LC = DROIT n2 C1 C2;
C12 = C2A 'ET' C2B 'ET' C2C;
C11 = C1A 'ET' C1B 'ET' C1C;
CercA = DALL LA C2A LB C1A;
CercB = DALL LB C1B LC C2B;
CercC = DALL LC C2C LA C1C;
SURF = CercA 'ET' CercB 'ET' CercC;
ELIM SURF 1.e-6;

* TLoi1 = 'TABLE';
TLoi1.'MODELE' = 'umatthermiquem51d';
TLoi1.'LIBRAIRIE' = 'libUmatthermique.so';
coel1 = 'MOTS' 'YOUN' 'NU' 'RHO' 'ALPH';
staw1 = 'MOTS' 'EEXX' 'EEYY' 'EEZZ' 'EEXY'
       'P';
para1 = 'MOTS' 'T';
MOD1=MODE SURF 'MECANIQUE' 'ELASTIQUE'
       'NON_LINEAIRE' 'UTILISATEUR'
       'DESC_LOI' TLoi1
       'C_MATERIAU' coel1
       'C_VARINTER' stav1
       'PARA_LOI' para1;
xyoun = 106.059-0.04764*T0;
xyo = xyoun*1.e9;
xnu = 0.37;
xrho = 6500.;
xalph = 8.5*(T0-273-20);

MAT1 = MATE MOD1
       'YOUN' xyo 'NU' xnu 'RHO' xrho 'ALPH' xalph;
TLoi2 = 'TABLE';
TLoi2.'MODELE' = 'umatirradiationm5ld';
TLoi2.'LIBRAIRIE' = 'libUmatirradiation.so';
coe2 = 'MOTS' 'YOUN' 'NU' 'RHO' 'ALPH';
stav2 = 'MOTS' 'EEXX' 'EEYY' 'EEZZ' 'EEXY'
'P';
para2 = 'MOTS' 'T';
MOD2=MODE SURF 'MECANIQUE' 'ELASTIQUE'
'NON_LINEAIRE' 'UTILISATEUR'
'DESC_LOI' TLoi2
'C_MATRIAU' coe2
'C_VARINTER' stav2
'PARA_LOI' para2;
MAT2 = MATE MOD2
'YOUN' xyo 'NU' xnu 'RHO ' xrho 'ALPH' xalph;
TLoi3 = 'TABLE';
TLoi3.'MODELE' = 'umatm5ecrouecrouissage';
TLoi3.'LIBRAIRIE' = 'libUmatM5ecrou.so';
coe3 = 'MOTS' 'YOUN' 'NU' 'RHO' 'ALPH';
stav3 = 'MOTS' 'EEXX' 'EEYY' 'EEZZ' 'EEXY'
'P';
para3 = 'MOTS' 'T' 'temp';
MOD3=MODE SURF 'MECANIQUE' 'ELASTIQUE'
'NON_LINEAIRE' 'UTILISATEUR'
'DESC_LOI' TLoi3
'C_MATRIAU' coe3
'C_VARINTER' stav3
'PARA_LOI' para3;
MAT3 = MATE MOD3
'YOUN' xyo 'NU' xnu 'RHO ' xrho 'ALPH' xalph;

MODTO = MOD1 'ET' MOD2 'ET' MOD3;
MATTO = MAT1 'ET' MAT2 'ET' MAT3;

**------ CONDITION AU LIMITE ------**

CL1 = SYMT DEPL B1 B2 SURF 1E-6;
CL2 = BLOQ 'ORTHO' O LA;
CL3 = BLOQ SURF UZ;
CL = CL1 'ET' CL2 'ET' CL3;

**------ Définition en Pa de la pression unitaire ------**

FO2 = PRES MASS MODTO 7.0E6 CI2 ;
FO1 = PRES MASS MODTO 15.5E6 CI1 ;
ev0 = EVOL MANU temps (PROG 0.0 3600 3607200)
y (PROG 0.0 1.0 1.0 );
ev1 = EVOL MANU temps (PROG 0.0 3600 3607200)
y (PROG 1.0 1.0 1.0 );
ev2 = EVOL MANU temps (PROG 0.0 PAS 900. 3607200.)
y (PROG 0.0 PAS 900. 3607200.);

CHAR1 = CHAR FO1 ev0;
CHAR2 = CHAR FO2 ev0;
CHARF = CHAR1 'ET' CHAR2;
chttemps = MANU CHPO SURF 'temp' 1.;
CHARto = CHAR 'temp' chttemps ev2;
CT = BLOQ 'T' SURF;
CT0 = DEPI CT T0;
CHT0 = MANU 'CHPO' SURF 1 'T' T0;
CHART = CHAR T CHT0 ev1;
CHARTT = CHARF 'ET' CHART 'ET' CHARto;
pro0 = PROG 0.0 PAS 900. 3607200.;
pro1 = PROG 0.0 PAS 10800. 3607200.;
tabutil = TABLE;
tabutil.'MODELE' = MODTO;
tabutil.'CARACTERISTIQUES' = MATTO;
tabutil.'CHARGEMENT' = CHARTT;
tabutil.'TEMPS_CALCULES' = pro0;
tabutil.'TEMPS_SAUVES' = pro1;
tabutil.'BLOCAGES_MECANIQUES' = CL;
tabutil.'BLOCAGES_THERMIQUES' = CT0;
TMASAU=table;
tabutil.'MES_SAUVEGARDES'=TMASAU;
TMASAU.'DEFTO'=VRAI;
TMASAU.'DEFIN'=VRAI;
TEMPS 'ZERO';
*Post-traitement
peivyyu = 'PROG';
peivxxy = 'PROG';
peivuzz = 'PROG';
psig = 'PROG';
psigt = 'PROG';
netat = DIME (tabutil.'TEMPS');
metat = netat-1;
ietat = -1;
getat = ietat + 1;
REPETER BECART netat ;
ietat = ietat + 1;
tabutil.'DEFORMATIONS'.ietat = 'EPSI' (tabutil.'MODELE')
(tabutil.'DEPLACEMENTS'.ietat);
sig = tabutil.'CONTRAINTES'.ietat;
sigt = EXTR (tabutil.'CONTRAINTES'.ietat) 'SMYY' 1 155 1;
epxx = EXTR (tabutil.'DEFORMATIONS'.ietat) 'EPXX' 1 155 1;
epyy = EXTR (tabutil.'DEFORMATIONS'.ietat) 'EYYY' 1 155 1;
epzz = EXTR (tabutil.'DEFORMATIONS'.ietat) 'EPZZ' 1 155 1;
epxy = EXTR (tabutil.'DEFORMATIONS'.ietat) 'GAXY' 1 155 1;
psigtt = psigt 'ET' ('PROG' sigtt);
psig = psig 'ET' ('PROG' ('MAXIMUM'
(VMIS sig (tabutil. 'MODELE') ) ));
peixxxu = peixxxu 'ET' ('PROG' epxx );
peixyyu = peixyyu 'ET' ('PROG' epyy );
peixzz = peixzz 'ET' ('PROG' epzz );
peixxy = peixxy 'ET' ('PROG' epxy );
'FIN' BECART;
evsigt = 'EVL' 'ROUGE' 'MANU' 'TEMPS' pro1 'SMYY' psigtt;
evx = 'EVL' 'ROUGE' 'MANU' 'TEMPS' pro1 'EIPRR' peixxxu;
evy = 'EVL' 'ROUGE' 'MANU' 'TEMPS' pro1 'EIPPTT' peixyyu;
evz = 'EVL' 'ROUGE' 'MANU' 'TEMPS' pro1 'EIPZZ' peixzzu;
evxy = 'EVL' 'ROUGE' 'MANU' 'TEMPS' pro1 'EIPRZ' peixxyu;
evs = 'EVL' 'ROUGE' 'MANU' 'TEMPS' pro1 'SIGEQ' psig;
'DESSIN' evsigt;
'DESSIN' evx;
'DESSIN' evy;
'DESSIN' evz;
'DESSIN' evxy;
'DESSIN' evs;
metat = netat-1;
DEF0 = DEFO SURF tabutil. 'DEPLACEMENTS'.metat 0. 'BLEU';
DEFI = DEFO SURF tabutil. 'DEPLACEMENTS'.metat 'ROUGE';
TRAC (DEFI 'ET' DEF0);
FIN;
Appendix 7 - Validation curve for the Elastic Orthotropic Behaviour

Appendix 8 - Unsuccessful test for calling UMAT with the interoperability C-Fortran

Before using C-function to call UMAT, I first thought about writing everything in FORTRAN and used the C/FORTRAN interoperability. In point of fact, the ISO_C_BINDING module allows converting C-pointer into FORTRAN directly. The idea was to create a C-pointer with FORTRAN, which points to the UMAT prototype. Then the pointer is converted to a FORTRAN pointer (with the \texttt{C\_F\_POINTEUR} function). This pointer can finally call the required function. There were a few reasons which showed me that was a bad idea. The definition of the prototype is very touchy. I never really succeeded in creating the prototype of a C-function in FORTRAN. The linkage is also complicated. As point of fact the compiler does not do easily a linkage between a FORTRAN code and a C-library. Finally I realized that today FORTRAN calls easily basic C-function.

Those reasons pushed to try another way to call UMAT.
Appendix 9 - Make.rules for the C/FORTRAN codes

# type d'architecture logiciel
GALILEO_ARCH_TYPE=64
# compilation en statique ou en dynamique
# GALILEO_LINKAGE_TYPE=static
GALILEO_LINKAGE_TYPE=dynamic

GALILEO_CPPFLAGS = -Wall -Wfatal-errors -ansi
GALILEO_CPPFLAGS += -D\'F77\_FUNC\(X,Y\)=X\#\#\' -D\'F77\_FUNC\(X,Y\)=X\#\#\'
GALILEO_CXXFLAGS = `tfel-config --oflags``

# drapeaux pour la compilation des fichiers en fonction de l'architecture
ifeq ($(GALILEO_ARCH_TYPE),64)
    GALILEO_FFLAGS = -fdefault-integer-8
else
    GALILEO_FFLAGS = -fdefault-integer-4
    GALILEO_CPPFLAGS += DGALILEO_32
endif

# drapeaux pour la compilation des fichiers en fonction du type de link
ifeq ($(GALILEO_LINKAGE_TYPE),dynamic)
    GALILEO_FFLAGS += -fPIC -DPIC
    GALILEO_CXXFLAGS += -fPIC -DPIC
endif

# drapeaux pour l'éditeur de lien
MFRONT_LDFLAGS = -LUMATInterface `tfel-config --libs --material`
GALILEO_LDFLAGS = $(MFRONT_LDFLAGS) -lstdc++

# pour supporter l'appel de lois externes
# Attention : ne fonctionne que si galileo est compilé en dynamique !
# GALILEO_CPPFLAGS += -DGALILEO_EXTERNAL_FUNCTION
# GALILEO_LDFLAGS += -ldl

###############################################################################
# configuration relative à TFEL/MFRONT
###############################################################################

ifeq ($(GALILEO_ARCH_TYPE),64)
# liste de drapeaux pour mfront en 64 bits
MFRONT_CPPFLAGS:=-DSUN -DUNIX64 -DTHREAD `tfel-config --includes`
else
# liste de drapeaux pour mfront en 32 bits
MFRONT_CPPFLAGS:=-DSUN -DUNIX32 -DTHREAD `tfel-config --includes`
endif

# répertoire d'installation de castem (non nécessaire avec des versions récentes de tfel/mfront)
CASTEMHOME:=-/t1/fftt/public/codes/ALCYONE/1.3_mfront/PREREQUIS/2007/CASTEM/include
MFRONT_CPPFLAGS +=-I$(CASTEMHOME)
Appendix 10 - Makefile for the C/FORTRAN codes

The library is called libGalileoLowCreepFlow. Cfunction.c contains the C-program. The FORTRAN codes (Test_appel_Pointeur.f90 and Test_Interface.f90) use the MFRONT function (by calling the function in cfunction).

For the linkage, -lGalileoLowCreepFlow means the compiler links the file with the library name libGalileoLowCreepFlow.so. -Lmfront means the compiler searches the library in the LibraryPATH and in the folder ./mfront.

```makefile
### Makefile ---
include make.rules
ifeq ($(GALILEO_LINKAGE_TYPE),dynamic)
  all: Test_appel_Pointeur.f90 Test_Interface.f90 cfunction.c
      mfront/libGalileoLowCreepFlow.so
  gfortran $(GALILEO_FFLAGS) Test_appel_Pointeur.f90 -o Test_appel_Pointeur.o -c
  gfortran $(GALILEO_FFLAGS) Test_Interface.f90 -o Test_Interface.o -c
  gcc $(GALILEO_CPPFLAGS) cfunction.c -o cfunction.o -c
  gfortran cfunction.o Test_appel_Pointeur.o Test_Interface.o -o Test_Interface
      -Lmfront -lGalileoLowCreepFlow $(GALILEO_LDFLAGS)
  else
    all: Test_appel_Pointeur.f90 Test_Interface.f90 cfunction.c
         mfront/libGalileoLowCreepFlow.a
    gfortran $(GALILEO_FFLAGS) Test_appel_Pointeur.f90 -o Test_appel_Pointeur.o -c
    gfortran $(GALILEO_FFLAGS) Test_Interface.f90 -o Test_Interface.o -c
    gcc cfunction.c -o cfunction.o -c
    gfortran --static cfunction.o Test_appel_Pointeur.o Test_Interface.o
         mfront/libGalileoLowCreepFlow.a -o Test_Interface $(GALILEO_LDFLAGS)
  endif

mfront/libGalileoLowCreepFlow.so:
  make --directory mfront

front/libGalileoLowCreepFlow.a:
  make --directory mfront

clean:
  rm -f *.o
  rm -f Test_Interface
  make --directory mfront clean

### Makefile ends here
```
Appendix 11 - Makefile for the MFRONT file

```makefile
# Makefile generated by mfront.
include ../make.rules

INCLUDES := -Iinclude
CPPFLAGS := $(GALILEO_CPPFLAGS) $(MFRONT_CPPFLAGS)
CXXFLAGS := $(GALILEO_CXXFLAGS)

SRCCXX = src/M51D.cxx src/umatLowCreepFlowM51D.cxx
OBJECTS = $(SRCCXX:.cxx=.o)
MAKEFILES = $(SRCCXX:.cxx=.d)

.PHONY = all clean libGalileoLowCreepFlow.so libGalileoLowCreepFlow.a

ifeq ($(GALILEO_LINKAGE_TYPE),dynamic)
  all : libGalileoLowCreepFlow.so
  libGalileoLowCreepFlow.so : $(OBJECTS)
    $(CXX) -shared $^ -o $@ -L $(MFRONT_LDFLAGS)
  else
    all : libGalileoLowCreepFlow.a
    libGalileoLowCreepFlow.a : $(OBJECTS)
      $(AR) rcs $@ $^ 
  endif

clean :
  @rm -fr $(OBJECTS) src include castem *.so *.d

-include $(MAKEFILES)

src/M51D.cxx src/umatLowCreepFlowM51D.cxx:M51Dtot.mfront
  mfront --interface=umat M51Dtot.mfront

%.o:%.cxx
  $(CXX) $(INCLUDES) $(CPPFLAGS) $(CXXFLAGS) $< -o $@ -c

%.d:%.cxx
  @set -e; rm -f $@;
  $(CXX) -M $(INCLUDES) $(CPPFLAGS) $(CXXFLAGS) $< > $@.$$$$; \
  sed "s,$\n  \$\(.*\)$\n  \"1.o $@ : ,g' < $@.$$$$ > $@; \
  rm -f $@.$$$$
```
## Appendix 12 - Values for different key parameters of the UMAT function

<table>
<thead>
<tr>
<th>Number</th>
<th>Calculation Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Tridimensional</td>
</tr>
<tr>
<td>0</td>
<td>Axissymmetrical</td>
</tr>
<tr>
<td>-1</td>
<td>Plane Strain</td>
</tr>
<tr>
<td>-2</td>
<td>Plane Stress</td>
</tr>
<tr>
<td>-3</td>
<td>Generalised Plane Strain</td>
</tr>
<tr>
<td>14</td>
<td>Axissymmetrical Generalised Plane Strain</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Number</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>-1</td>
<td>The integration failed</td>
</tr>
<tr>
<td>-2</td>
<td>There is an UMATException (Problems between the given arguments and the expected arguments).</td>
</tr>
<tr>
<td>-3</td>
<td>There is a MaterialException (there are bound problems).</td>
</tr>
<tr>
<td>-4</td>
<td>There is a TFELException (Problems with TFEL).</td>
</tr>
<tr>
<td>-5</td>
<td>There is a std::exception (Problems with class).</td>
</tr>
<tr>
<td>-6</td>
<td>There is an unknown exception</td>
</tr>
<tr>
<td>-7</td>
<td>Wrong calculation condition</td>
</tr>
</tbody>
</table>
Subroutine CallPointer(TEMPS,TSTR,DELA,DFLU,DEFOeq,DTOT,DDTOT)
implicit none
real(KIND=8),dimension(3), intent (inout)::TSTR,DELA,DFLU,DDTOT
real(KIND=8),dimension(3), intent (in)::DTOT
real(KIND=8),dimension(9)::DROT,DDSDDE
integer(KIND=8)::NDI,NTENS, NPROP,NSTATV,KINC
real(KIND=8)::DTIME, TEMP, DTEMP
real(KIND=8),dimension(1)::PREDEF,DPRED
real(KIND=8),dimension(3)::STRESS,STRAN,DSTRAN
real(KIND=8),dimension(7)::STATEV
real(KIND=8),dimension(4)::PROPS
real(KIND=8), intent (in)::TEMPS
real(KIND=8), intent (inout)::DEFOeq
integer::ptr
integer::ret
integer::J,II,A
real(KIND=8),dimension(3)::SDEFO

!creation of the pointer
CALL GetPointer(ptr,'libGalileoLowCreepFlow.so','umatlowcreepflowm51d',ret)

NTENS = 3 !number of parameters in the stress tensor (3, because it is 1D)
NSTATV=7 !number of internal variable (7, 3 for elastic strain + 3 for creep strain + 1 for equivalent strain)
NPROP = 4 !number of material property (Young+Poisson+rho+thermical exp)
NDI = 14 ! 1D plane strain
DTIME = 1 !external parameter and its increment (just one, the time)
PREDEF(1) = (TEMPS-DTIME)
DPRED (1) = DTIME !Matrix of changing axis
DROT(1) = 1
DROT (2) = 0
DROT (3) = 0
DROT (4) = 0
DROT (5) = 1
DROT (6) = 0
DROT (7) = 0
DROT (8) = 0
25 DROT(9) = 1

!Stress
STRESS(1) = TSTR(1)
STRESS(2) = TSTR(2)
STRESS(3) = TSTR(3)

!Internal Variable
STATEV(1) = DELA(1)
STATEV(2) = DELA(2)
STATEV(3) = DELA(3)
STATEV(4) = DEFOeq
STATEV(5) = DFLU(1)
STATEV(6) = DFLU(2)
STATEV(7) = DFLU(3)

!total Strain + increment
STRAN(1) = DTOT (1)
STRAN(2) = DTOT (2)
STRAN(3) = DTOT (3)
DSTRAN(1) = DDTOT(1)
DSTRAN(2) = DDTOT(2)
DSTRAN (3) = DDTOT (3)
!Material property
PROPS(1) = 1e11
PROPS(2) = 0.3
PROPS(3) = 1000
PROPS(4) = 1e-6

!>Temperature and increment
TEMP = 640
DTEMP = 0

!>MFRONT calling

CALL CALLMECHANICALBEHAVIOUR(PTR, NTENS, DTIME, DROT, DDSDDE, STRAN, &
DSTRAN, TEMP, DTEMP, PROPS, NPROP, PREDEF, DPRED, STATEV, &
NSTATV, STRESS, ND1, KINC)

SDEFO(1) = STATEV(5) + STATEV(1)
SDEFO(2) = STATEV(6) + STATEV(2)
SDEFO(3) = STATEV(7) + STATEV(3)
if (SDEFO(1) /= (DTOT(1)+DDTOT(1))) then
DDTOT(1) = SDEFO(1) - DTOT(1)
GO TO 25
endif

DFLU(1) = STATEV(5)
DFLU(2) = STATEV(6)
DFLU(3) = STATEV(7)

DELA(1) = STATEV(1)
DELA(2) = STATEV(2)
DELA(3) = STATEV(3)

TSTR(1) = STRESS(1)
TSTR(2) = STRESS(2)
TSTR(3) = STRESS(3)

DEFOeq = STATEV(4)

END subroutine CallPointer
program RodExample
  implicit none
  integer(KIND=8)::time,JJ
  real(KIND=8),dimension(1)::PREDEF,DPRED
  real(KIND=8),dimension(1,2)::DEFOeq
  real(KIND=8),dimension(3,2)::STRESS,STRAIN,DSTRAIN,DELA,DFLU,DTOT,DDTOT
  real(KIND=8),dimension(7)::STATEV
  real(KIND=8),dimension(4)::PROPS
  real(KIND=8),dimension(100)::EFLU1,EFLU2,EELA1,EELA2,ETOT1,ETOT2,STRE1,STRE2
  do JJ=1,2,1
    STRESS(1,JJ) = 0
    STRESS(2,JJ) = 0
    STRESS(3,JJ) = 0
    DELA(1,JJ) = 0
    DELA(2,JJ) = 0
    DELA(3,JJ) = 0
    DFLU(1,JJ) = 0
    DFLU(2,JJ) = 0
    DFLU(3,JJ) = 0
    DEFOeq(1,JJ) = 0
    DTOT(1,JJ) = 0
    DTOT(2,JJ) = 0
    DTOT(3,JJ) = 0
    DDTOT(1,JJ) = 0
    DDTOT(2,JJ) = 0
    DDTOT(3,JJ) = 0
  end do
  DDTOT(1,2) = 0.001
  print *, 'coucou'
  open(unit=27, file='resultats_tests',status='unknown')
  write(27,*) "temps EFLU1 EFLU2 EELA1 EELA2 ETOT1 ETOT2 STRE1 STRE2"
  do time = 1,100,1
    print *,time
    do JJ = 1,2
      call CallPointer
      (time,STRESS(:,JJ),DELA(:,JJ),DFLU(:,JJ),DEFOeq(1,JJ),DTOT(:,JJ),DDTOT(:,JJ))
    end do
    EFLU1(time) = DFLU(1,1)
    EFLU2(time) = DFLU(1,2)
    EELA1(time) = DELA(1,1)
    EELA2(time) = DELA(1,2)
    ETOT1(time) = DFLU(1,1) + DELA(1,1)
    ETOT2(time) = DFLU(1,2) + DELA(1,2)
    STRE1(time) = STRESS(1,1)
    STRE2(time) = STRESS(1,2)
    DTOT(1,1) = DTOT(1,1)+DDTOT(1,1)
    DTOT(1,2) = DTOT(1,2)+DDTOT(1,2)
    if(time == 10) then
      DDTOT(1,2) =0
    end if
    write(27,*
    time,EFLU1(time),EFLU2(time),EELA1(time),EELA2(time),ETOT1(time),ETOT2(time),STRE1(time),STRE2(time)
  end do
end program RodExample
Appendix 14 - CASTEM codes for the Rod-Tests

```
OPTION DIME 2 ELEM QUA4;
OPTION ECHO 1;
OPTION MODE PLAN GENE;
TITRE 'planch';
TEMPS ;
* GEOMETRY
-----------------------------------------------------
T0 = 640.;
O = POIN 0. 0.;
A = POIN 1. 0.;
C = POIN 0. 0.05;
B = POIN 1. 0.05;
OA = DROIT 1 O A;
AB = DROIT 1 A B;
BC = DROIT 1 B C;
CO = DROIT 1 C O;
SURF = DALL OA  AB BC  CO;
PG = POIN 0.5 0.025;
PG = SURF POIN PROC PG;
TRAC SURF;
ELIM SURF 1.e-6;
*
TLoi1 = 'TABLE';
TLoi1.'MODELE' = 'umatlowcreepflowm51d';
TLoi1.'LIBRAIRIE' = 'libUmatLowCreepFlow.so';
coel1 = 'MOTS' 'YOUN' 'NU' 'RHO' 'ALPH';
stav1 = 'MOTS' 'EEXX' 'EEYY' 'EEZZ' 'EEXY';
paral = 'MOTS' 'T' 'temp';
MOD1=MODE SURF 'MECANIQUE' 'ELASTIQUE'
'NON_LINEAIRE' 'UTILISATEUR'
'DESC_LOI' TLoi1
'C_MATERIAU' coel1
'C_VARINTER' stav1
'PARA_LOI' paral
'DPGE' PG;
xyoun = 100;
xyo = xyoun*1.e9;
xnu = 0.3;
xrho = 1000.;
xalph = 1.e-6;
MAT1 = MATE MOD1
'YOUN' xyo 'NU' xnu  'RHO ' xrho 'ALPH' xalph;
MODTO = MOD1;
MATTO = MAT1;
*------ Boundary Condition------
DENS 1e-6;

CL1 = BLOQ CO UX;
CL2 = BLOQ SURF UY;
CL3 = BLOQ AB UX;
CL4 = BLOQ PG U2 RX RY;
CL = CL1 'ET' CL2 'ET' CL3 'ET' CL4;
ev0 = EVOL MANU temps (PROG 0.0 10. 100.)
y (PROG 0 0.01 0.01);
ev1 = EVOL MANU temps (PROG 0.0 10. 100.)
y (PROG 1.0 1.0 1.0 ) ;
ev2 = EVOL MANU temps (PROG 0.0 PAS 1. 100.)
y (PROG 0.0 PAS 1. 100.);
```

ev2 = EVOL MANU temps (PROG 0.0 PAS 1. 100.)
y (PROG 0.0 PAS 1. 100.);
chdepl = ((DEPI CL3 1.) 'ET' (DEPI CL1 0.));
CHARF = CHAR 'DIMP' chdepl ev0;

chtemps = MANU CHPO SURF 'temp' 1.;
CHARto = CHAR 'temp' chtemps ev2;

CT = BLOQ 'T' SURF;
CT0 = DEPI CT T0;
CHT0 = MANU 'CHPO' SURF 1 'T' T0;

CHART = CHAR T CHT0 ev1;

CHARTT = CHARF 'ET' CHART 'ET' CHARto;

pro0 = PROG 0.0 PAS 1. 100.;
pro1 = PROG 0.0 PAS 1. 100.;
tabutil = TABLE;
tabutil.'MODELE' = MODTO;
tabutil.'CARACTERISTIQUES' = MATTO;
tabutil.'CHARGEMENT' = CHARTT;
tabutil.'TEMPS_CALCULES' = pro0;
tabutil.'TEMPS_SAUVES' = pro1;
tabutil.'BLOCAGES_MECA' = CT0;
tabutil.'BLOCAGES_THERMIQUES' = CHT0;

TMASAU=table;
tabutil .'MES_SAUVEGARDES'=TMASAU;
TMASAU .'DEFTO'=VRAI;
TMASAU .'DEFIN'=VRAI;
TEMPS 'ZERO';
PASAPAS tabutil;

netat = DIME (tabutil.'TEMPS');
metat = netat - 1;
ietat = -1;
'REPETER BECART netat ;

ietat = ietat + 1;
REPETER BECART netat ;

  ietat = ietat + 1;

* ************** recul des deformations
  tabutil.'DEFORMATIONS'.ietat = 'EPSI' (tabutil.'MODELE')
  (tabutil.'DEPLACEMENTS'.ietat);

  sig = tabutil.'CONTRAINTES'.ietat;
  defto = tabutil.'DEFORMATIONS'.ietat;
  def = tabutil.'VARIABLES_INTERNES'.ietat;

  sigxx = EXTR sig 'SMXX' 1 1 2;
  sigyy = EXTR sig 'SMYY' 1 1 1;
  sigzz = EXTR sig 'SMZZ' 1 1 1;
  etxx = EXTR defto 'EPXX' 1 1 1;
  etyy = EXTR defto 'EPYY' 1 1 1;
  etzz = EXTR defto 'EPZZ' 1 1 1;
  eexx = EXTR def 'EEXX' 1 1 1;
  eeyy = EXTR def 'EEYY' 1 1 1;
  eezz = EXTR def 'EEZZ' 1 1 1;
  efxx = EXTR def 'EVXX' 1 1 1;
  efyy = EXTR def 'EVYY' 1 1 1;
  efzz = EXTR def 'EVZZ' 1 1 1;

  psigxx = psigxx 'ET' ('PROG' sigxx);
  psigyy = psigyy 'ET' ('PROG' sigyy);
  psigzz = psigzz 'ET' ('PROG' sigzz);
  petxxu = petxxu 'ET' ('PROG' etxx );
  petyyu = petyyu 'ET' ('PROG' etyy);
  petzzu = petzzu 'ET' ('PROG' etzz);
  peexxu = peexxu 'ET' ('PROG' eexx);
  peeyyu = peeyyu 'ET' ('PROG' eeyy);
  peezzu = peezzu 'ET' ('PROG' eezz);
  pefxxu = pefxxu 'ET' ('PROG' efxx);
  pefyyu = pefyyu 'ET' ('PROG' efyy);
  pefzzu = pefzzu 'ET' ('PROG' efzz);

'FIN' BECART;

evsigxx = 'EVOL' 'ROUGE' 'MANU' 'TEMPS' pro1 'SMXX' psigxx;
evsigy = 'EVOL' 'ROUGE' 'MANU' 'TEMPS' pro1 'SMYY' psigyy;
evsigzz = 'EVOL' 'ROUGE' 'MANU' 'TEMPS' pro1 'SMZZ' psigzz;
etx = 'EVOL' 'VERT' 'MANU' 'TEMPS' pro1 'ETXX' petxxu;
ety = 'EVOL' 'VERT' 'MANU' 'TEMPS' pro1 'ETYY' petyyu;
etz = 'EVOL' 'BLU' 'MANU' 'TEMPS' pro1 'ETZZ' petzzu;
eex = 'EVOL' 'ROUGE' 'MANU' 'TEMPS' pro1 'EEXX' peexxu;
eey = 'EVOL' 'VERT' 'MANU' 'TEMPS' pro1 'EEYY' peeyyu;
eez = 'EVOL' 'BLU' 'MANU' 'TEMPS' pro1 'EEZZ' peezzu;
efx = 'EVOL' 'ROUGE' 'MANU' 'TEMPS' pro1 'EFXX' pefxxu;
efy = 'EVOL' 'VERT' 'MANU' 'TEMPS' pro1 'EFYY' pefyyu;
efz = 'EVOL' 'BLU' 'MANU' 'TEMPS' pro1 'EFZZ' pefzzu;

'DESSIN' (evsigxx 'ET' evsigyy 'ET' evsigzz);

'DESSIN' (etx 'ET' ety);
`'DESSIN' (evsigxx 'ET' evsigyy 'ET' evsigzz);

'DESSIN' (etx 'ET' ety);
'DESSIN' (efx 'ET' efy);
'DESSIN' (eex 'ET' eey);

metat = netat-1;
DEF0 = DEFO SURF tabutil. 'DEPLACEMENTS'.metat 0. 'BLEU';
DEF1 = DEFO SURF tabutil. 'DEPLACEMENTS'.metat 'ROUGE';

@EXCEL1 evsigxx resultat_castem;

TRAC (DEF1 'ET' DEF0);
FIN;
Appendix 15 – Comparison of the results for the cladding (out of the report)

Elastic strain, GALILEO (red) MFRONT (blue)

Young Modulus, GALILEO (red) MFRONT (blue)