Le générateur de code `mfront` : présentation générale et application aux propriétés matériau et aux modèles

T. Helfer, É Castelier

Décembre 2013

RÉSUMÉ

L'une des missions de la plate-forme `pleiades` est de capitaliser les connaissances matériau utilisées pour la simulation des éléments combustibles et absorbants.

`mfront` est un générateur de code dédié aux connaissances matériau. Son domaine d'application couvre :

- les propriétés matériau ;
- les lois de comportement ;
- les modèles,

pour lesquels il rassemble un important savoir-faire physique, numérique et informatique.

Couplé à la base de données `sirius`, `mfront` vise à garantir une gestion pérenne, robuste, efficace et évolutive des connaissances matériau dans la plate-forme `pleiades` [Michel 09, Helfer 11b]. Il permet également à des utilisateurs non développeur d'écrire leurs propres connaissances matériau [Helfer 10].

Il s'agit d'un outil mature, qui se veut de qualité industrielle et qui intéresse fortement différents départements du CEA [Helfer 11a, d'Arrigo 12] ainsi que ses partenaires, Areva [Olagnon 13] et ÉDF [Proix 13].

Nous décrivons dans cette note l'utilisation de `mfront` et détaillons comment écrire des propriétés matériau et des modèles. Les lois de comportement mécaniques sont détaillées dans une note à part [Helfer 13c].

SOMMAIRE

1 INTRODUCTION

Cette note donne une vue générale du générateur de code `mfront` et de son application aux propriétés matériau et aux modèles physico-chimiques. Elle est complétée par trois autres notes qui traitent [1] :

— des lois de comportement mécanique [Helfer 13c] ;
— de l'interface `umat` utilisée par le code aux éléments finis `Cast3M` [Helfer 13b] ;
— de l'interface utilisée par le code aux éléments finis `Aster` [Helfer 13a] ;

`mfront` fait partie d'une librairie mathématique généraliste nommée `tfel` très utilisée par `mfront`, mais qui a également de nombreuses autres utilisations dans la plate-forme `pleiades`. Ces notes sont intégrées à la gestion de configuration de `tfel` et évoluent continûment avec les développements de `tfel` [2]. Le présent document a été généré à partir de la révision 727776783.

1.1 OBJECTIF DE MFRONT

Développé dans le cadre de la plateforme `pleiades` du CEA, `mfront` vise à garantir une gestion pérenne, robuste, efficace et évolutive (au sens du suivi des évolutions logicielles) des connaissances matériau et à faciliter leurs écritures par des utilisateurs non développeurs [Michel 09, Helfer 11b] :

— la pérennisation et le caractère évolutif des connaissances matériau est assuré par :
  — l'écriture des connaissances matériau dans un fichier indépendant du contexte logiciel cible (code ou langage de programmation). Ce fichier contient uniquement l'information physique et, éventuellement, des informations numériques. Ce fichier sert à générer des sources C++ qui sont adaptées au contexte par l'utilisation de la notion d'interface (voir section 2.4). Si le code ou le langage cible évolue, il suffit de faire évoluer l'interface, sans modifier le fichier `mfront` [3]
  — l'utilisation du langage C++, actuellement considéré comme l'un des langages les plus utilisés au monde [4] et qui est particulièrement utilisé dans le monde scientifique ;
— les performances, en particulier pour les lois de comportement mécaniques, sont également assurées par l'utilisation du langage C++ qui permet d'utiliser des techniques de programmation extrêmement efficaces. Ces bonnes performances ont été constatées à plusieurs reprises [Michel 09, Helfer 11b, Proix 13, Olagnon 13] ;
— la simplicité d'utilisation est liée à l'utilisation de la génération de code, qui permet de masquer la plupart des détails informatiques. Dans le cas des lois de comportement mécanique, l'écriture d'une bibliothèque de calcul tensorielle permet de décrire les équations régissant l'évolution des variables internes sous une forme quasi-symbolique.

Les fichiers écrits en `mfront` sont capitalisés dans une base de données propre à la plate-forme nommée `sirius`.

1. Un guide de référence de la librairie `tfel` est également en cours de rédaction. Cette note étant assez ambitieuse, sa rédaction s'étalera dans le temps. Elle est cependant disponible en version projet dans les sources de la librairie. Elle contient en particulier le guide d'installation de la librairie.

2. La version la plus à jour est obtenue dans le répertoire de compilation des sources par la commande :

```
make doc-pdf
```

3. Pour illustrer notre propos, donnons quelques exemples :
— l'architecture $V2.0$ fournit une interface aux propriétés matériau qui a permis de porter l'ensemble de propriétés matériau de `MAIA`. Le temps de développement de l'interface est de l'ordre de l'heure ;
— toutes les lois de comportement mécanique écrites avec `mfront` peuvent être utilisées ou dans le code aux éléments finis `Cast3M` [CEA 13] ou dans le code `Aster` [EDF 13]. Le support d'autres codes est envisageable ;
— certains modèles utilisés dans `licos` sont restés inchangés depuis leur écriture pour l'application `celaeno` (les deux codes n'ont plus grand chose de commun) ;
— Les modèles `mfront` sont interchangeables entre les `licos` et `germinal`, ce qui n'est pas le cas des modèles écrits en C++ ;
— Les modèles `mfront` utilisés dans `germinal` ne seront pas modifiés par le passage à l'architecture 2.0.

4. Il s'agit du langage le plus utilisé si on considère le langage `C` comme un sous-ensemble du C++, et du troisième ou quatrième langage si on les distingue [Software 13].

1.2 PLAN DE LA NOTE

La première partie constitue une présentation générale de `mfront` et de son utilisation.

La seconde partie décrit comment écrire des propriétés matériau.

La troisième partie est dédiée à l'écriture des modèles physico-chimiques.

Les annexes traitent des questions suivantes :
- le contrôle de la compilation des fichiers `mfront` en librairies dynamiques ;
- l'interface `C` aux propriétés matériau ;
- l'interface `Cast3M` aux propriétés matériau ;
- l'interface `python` aux propriétés matériau ;
- l'appel de propriétés matériaux externes dans `Cast3M` et des modifications qui ont été effectuées à ce code pour permettre cet appel ;
- l'utilisation des propriétés matériau dans le tableur `Microsoft Office Excel` via l'interface `Cast3M`.

2 PRÉSENTATION GÉNÉRALE DE MFRONT

`mfront` est un générateur de code dédié aux connaissances matériau : partant d'un fichier contenant les informations physiques et numériques utiles, `mfront` génère un ou plusieurs fichiers C++ utilisables dans différents contextes logiciels, ce qui englobe des codes de calculs ou des langages de programmation. `mfront` peut éventuellement gérer leurs compilations.

Pour proposer une solution adaptée aux différentes situations, plusieurs analyseurs syntaxiques sont disponibles. Pour chaque analyseur, différentes interfaces sont disponibles qui permettent d'adapter le code généré au contexte logiciel cible.

Nous commençons par présenter la structure du fichier de donnée (paragraphe 2.1) et les directives communes à tous les analyseurs (paragraphe 2.2).

Puis, nous décrivons les *analyseurs* (paragraphe 2.3) et les interfaces (paragraphe 2.4) disponibles.

2.1 FICHIERS D'ENTRÉE

```
@Parser IsotropicMisesCreep;
@Behaviour SiCCreep;
@Author    É. Brunon;
@Date      06/12/07;

@Description{
Matériaux RCG-T et RCG-R
Un point sur le carbure de Silicium
NT SESC/LIAC 02-024 ind 0 de décembre 2002
J.M. ESCLEINE
§8.4.2 Page 34
}

@ExternalStateVariable real flux;

@StaticVariable real A =  4.4e3;
@StaticVariable real B = 76.0e3;
@StaticVariable real a =  1.0e-37;

@LocalVariable real AF1;
@LocalVariable real AF3;

@InitLocalVariables{
  AF1   = A*exp(-B/(T+theta*dT));
  AF3   = a*(flux+theta*dflux);
}

@FlowRule{
  df_dseq  = AF1+AF3;
  f        = seq*df_dseq;
}
```

FIGURE 1 : Implantation de la loi de comportement viscoplastique du SiC en `mfront`.

La figure 1 donne un premier exemple de fichier `mfront`. Nous décrivons en détail cet exemple dans la note dédiée aux lois de comportement mécanique [Helfer 13c] et nous insistons ici sur sa structure, puis nous décrivons certains mots clés communs à l'ensemble des analyseurs.

Le type `real` Pour représenter les nombres flottants, nous avons introduit un type nommé `real`. Les interfaces doivent définir ce type pour la cible visée. Les connaissances matériaux écrites en `mfront` peuvent ainsi être utilisée en simple, double ou quadruple précision.

Noms de glossaire Pour l'intégration des fichiers générés dans les applications de la plateforme `pleiades`, il est possible d'affecter aux variables d'entrée et de sortie des connaissances matériau des noms de glossaire. Ces noms de glossaire servent à assurer le bon échange d'information.

L'annexe **??** décrit les noms de glossaire les plus fréquemment utilisés dans le code `licos`.

2.1.1 Structure du fichier d'entrée

Un fichier d'entrée de `mfront` représente une unique connaissance matériau (une propriété matériau, une loi de comportement, un modèle).

Il se présente sous la forme d'une liste de mots clés, appelés également directives dans la suite du document, commençant par une arobase `@`. Dans la mesure du possible, nous avons recherché à nous rapprocher le plus possible de la syntaxe utilisée par le langage C++ et certaines parties du fichier sont directement écrites en C++. La surcouche apportée par `mfront` a cependant été créée pour minimiser les détails propres à ce langage. Le retour d'expérience montre que l'écriture d'un fichier `mfront` est accessible, dans les cas usuels, à des personnes n'ayant pas ou peu de compétences en développement informatique.

Ordre des directives L'ordre des directives n'est pas imposé. Il faut cependant souligner que `mfront` analyse le fichier de manière séquentielle et n'autorisera pas qu'une variable soit utilisée avant d'avoir été déclarée.

Par exemple, il est impossible de préciser la plage de validité en température d'une propriété de matériau avant de déclarer que la propriété dépend de la température.

Commentaires Les deux types de commentaires introduits par le langage C++ sont supportés :
- les commentaires commençant par les caractères `/*` et finissant par les caractères `*/`. Ces commentaires peuvent s'étendre sur plusieurs lignes ;
- les commentaires commençant par les caractères `//` s'étendent jusqu'à la fin de la ligne courante ;

2.2 Directives communes à l'ensemble des analyseurs

Nous décrivons dans ce paragraphe différentes directives communes à l'ensemble des analyseurs.

La directive `@Parser` La directive `@Parser` permet de préciser l'analyseur utilisé. Cette directive est suivie du nom de l'analyseur à utiliser et d'un point virgule. La liste des analyseurs disponible est donnée au paragraphe 2.3. Par défaut, l'analyseur `DefaultParser` est utilisé.

2.2.1 Directives informatives

Nous décrivons dans ce paragraphe quatre directives optionnelles dont le but est uniquement informatif [5]. Elles devraient également servir à terme lors de la génération automatique d'une documentation à partir d'un fichier `mfront`.

La directive `@Description` La directive `@Description` permet de documenter le fichier `mfront`. Cette documentation est donnée dans un bloc commençant par une accolade ouvrante `{` et se terminant par une accolade fermante `}`. Elle doit idéalement contenir la référence d'où est extraite la connaissance matériau traitée, les éventuelles adaptations faites et le cadre d'utilisation.

La directive `@Author` Le mot clé `@Author` donne le nom de la personne qui a écrit le fichier `mfront`. Ce mot clé est suivi du nom jusqu'à ce qu'un point virgule soit rencontré.

La directive `@Date` Le mot clé `@Date` donne la date d'écriture du fichier `mfront`. Ce mot clé est suivi de cette date jusqu'à ce qu'un point virgule soit rencontré.

La directive `@Material` La directive `@Material` précise le nom du matériau auquel se réfère la connaissance matériau. Ce mot clé est suivi du nom du matériau et d'un point virgule.

Ce nom de matériau est généralement utilisé pour construire le nom des fonctions ou des classes générées, il est donc soumis à certaines restrictions : un nom de matériau ne peut commencer par un chiffre et ne peut contenir que des lettres alphabétiques ou des chiffres. Le nom du matériau peut également être utilisé pour construire le nom de la librairie générée.

2.2.2 Appel aux propriétés matériau écrites en `mfront`

Afin de minimiser la duplication du code (et donc le risque d'erreur), il est naturel que les différentes connaissances matériau puissent utiliser des propriétés matériau définies par ailleurs [6].

Nous avons introduit la directive `@MaterialLaw` pour traiter ces cas. Cette directive est suivie d'un tableau de chaînes de caractères désignant des fichiers `mfront` et d'un point virgule.

Les fichiers désignés sont analysés par `mfront` avec deux effets :
- ces fichiers vont être analysés par une interface spéciale nommée `mfront` [7] et les fichiers générés vont être ajoutés à la liste des fichiers à compiler [8] ;
- l'interface `mfront` rend disponibles des fonctions dont le nom est donné par :
  - le nom du matériau, suivi du caractère tiret-bas (underscore) _ et par le nom de la propriété, donné par la directive `@Law`, si le nom du matériau est défini par la directive `@Material` (voir section 4) ;
  - le nom de la propriété, donné par la directive `@Law`, si le nom du matériau n'est pas défini.

5. Dans le cadre de la plateforme `pleiades`, ces informations sont utilisées lors de l'insertion d'un fichier `mfront` dans la base de données `sirius`.

6. Les propriétés matériau ont un statut particulier :
   - elles n'ont de sens que relativement à une loi de comportement ou à un modèle. Ainsi, un module d'YOUNG n'a de sens que si le comportement élastique du matériau est supposé représenté par une loi de HOOKE.
   - les propriétés matériau peuvent être liées. Par exemple, pour un matériau dont le comportement mécanique est basé sur une loi d'élasticité linéaire isotrope, le module d'YOUNG E, le coefficient de POISSON ν et le module de cisaillement G sont reliés par :

$$\nu = \frac{E}{2\,G} - 1$$

7. L'utilisation de cette interface évite d'éventuels conflits de nom.

8. La gestion de la compilation des fichiers est décrite en annexe A.

Ces différentes fonctions sont accessibles dans les différents blocs de code définis par l'utilisateur. Les arguments de ces fonctions sont ceux déclarés dans le fichier par la directive `@Input` (voir section 4) et doivent être fournis dans l'ordre de déclaration.

```
@Parser   MaterialLaw;
@Law      CreepExponent;
@Material A316TiHyperTrempe;
@Author   Thomas Helfer;
@Date     21 Jan. 2010;

@Description{
  Dossier de caractérisation du
  matériau de gainage des éléments
  absorbant 1ère et 2ème charge SPX1
  J.M. Escleine
  DEC/SDC/  85-2077
  DEC/SMPA/ 85-1449
  Décembre 1985
}

// changing the name of output
@Output n;

@Input T;
T.setGlossaryName("Temperature");

// temperature bounds
@PhysicalBounds T in [0:*[;
@Bounds T in [733.15:873.15];

@Function{
  n =  18571/T-12.861;
} // end of function
```

FIGURE 2 : Exemple de définition d'une propriété matériau.

Un exemple d'utilisation de la directive `@MaterialLaw` est illustrée par les figures 2 et 3. Deux propriétés matériau externes sont déclarées. Le code de la première d'entre elles est donné en figure 2. Elles sont utilisées en figure 3 pour initialiser des variables dans un bloc commençant par la directive `@InitLocalVariables`.

Note Il est important de noter que le nom du fichier `mfront` n'intervient pas dans le nom des fonctions générées. Il est cependant souhaitable et pratique de garder une certaine cohérence entre le nom du fichier et le nom des fonctions générées.

Chemins de recherche La directive `@MaterialLaw` recherche le fichier à inclure :

- dans le répertoire courant ;
- dans les répertoires précisés en ligne de commandes par l'option `--search-path` (voir la section 3) ;
- dans les répertoires pointés par la variable d'environnement `MFRONT_INCLUDE_PATH`. Cette variable fournit une liste de chemins séparés par le caractère `':'`.

```
@MaterialLaw{"A316TiHyperTrempe_CreepExponent.mfront",
             "A316TiHyperTrempe_CreepFactor.mfront"};

@InitLocalVariables{
  nc = A316TiHyperTrempe_CreepExponent(T + theta*dT);
  C  = A316TiHyperTrempe_CreepFactor(T + theta*dT);
```

FIGURE 3 : Exemple d'utilisation de la directive `MaterialLaw`. Utilisation de la propriété matériau définie en figure 2.

2.2.3 Inclusion d'autres fichiers `mfront`

La directive `@Import` permet d'inclure un fichier `mfront` dans le fichier courant.

Cette directive présente un intérêt particulier pour les lois de comportement mécanique[9]. En effet, beaucoup de lois de comportement utilisent le même formalisme et ne se différencient que par leurs coefficients. Une solution classique consiste à rajouter des propriétés matériau à la liste d'appel de la loi de comportement. Cette solution présente plusieurs inconvénients :

- on s'éloigne du choix `pleiades` de proposer des lois de comportement mécanique spécifiques aux matériaux ;
- il augmente le risque d'erreur de la part de l'utilisateur, qui doit correctement renseigner son jeu de données dans le code appelant ;
- il est difficile de passer d'un coefficient indépendant de la température à un coefficient dépendant de la température[10] alors que cela se fait naturellement dans les schémas d'intégration implicite[11].
- dans la plupart des codes appelant, la gestion des propriétés matériau a un coût non négligeable.

La solution qu'apporte la directive `@Import` est la suivante :

- on écrit un fichier maître définissant les différents coefficients. La façon de définir ces coefficients (propriété matériau[12], variable locale[13], etc..) et éventuellement la façon de les initialiser[14] dépend du cas traité.
- ce fichier maître inclut un fichier patron (« `template` » en anglais) contenant tout le détail de l'intégration.

Chemins de recherche La directive `@Import` recherche le fichier à inclure :

- dans le répertoire courant ;
- dans les répertoires précisés en ligne de commandes par l'option `--search-path` (voir la section 3) ;
- dans les répertoires pointés par la variable d'environnement `MFRONT_INCLUDE_PATH`. Cette variable fournit une liste de chemins séparés par le caractère `':'`.

Une limitation Il est aujourd'hui nécessaire de préciser l'analyseur à utiliser dans le fichier principal[15].

9. Pour comprendre ce paragraphe, une lecture préalable de la note dédiée aux lois de comportement mécanique peut s'avérer nécessaire [Helfer 13c].
10. Les propriétés matériau fournies par le code appelant sont constantes sur le pas de temps et l'instant auquel elles sont évaluées n'est pas nécessairement cohérent avec le schéma d'intégration utilisé par la loi de comportement.
11. Pour des raisons intrinsèques à ces schémas d'intégration, il est possible de substituer une propriété matériau variable dans le temps par sa valeur « en milieu de pas de temps ». Cette remarque n'est absolument pas valable pour les schémas d'intégration explicite de type RUNGE-KUTTA.
12. Voir le mot clé `@MaterialProperty`.
13. Voir le mot clé `@LocalVariable`.
14. Voir le mot clé `@InitLocalVariable`.
15. Voir le mot clé `@Parser`.

2.2.4 Directives de compilation

Trois directives sont liées à la compilation des sources. Les directives `@Includes` et `@Link` ont une importance particulière pour une utilisation avancée de `mfront` : ces directives permettent d'utiliser des fonctionnalités définies dans des librairies C++ externes.

La directive `@Library` La directive spécifie le nom de la librairie générée.

Cette information est optionnelle.

La directive `@Includes` La directive `@Includes` permet de déclarer des lignes de codes C++ introduites en tête des fichiers générés. Ces lignes sont données dans un bloc commençant par une accolade ouvrante `{` et se terminant par une accolade fermante `}`.

Le nom de cette directive est due au fait que ces instructions se résument généralement à une suite d'inclusions de fichiers d'entête par le mot clé `#include` du préprocesseur C++.

```
@Includes{
#include<TFEL/Material/Lame.hxx>
#include<TFEL/Material/Hill.hxx>
}
```

FIGURE 4 : Exemple d'utilisation de la directive `Includes`.

La figure 4 donne un exemple d'utilisation de la directive `@Includes`.

La directive `@Link` La directive `@Link` est utilisée pour lier les librairies générées à des librairies externes.

Elle est suivie d'une chaîne de caractères (entre doubles quotes) contenant des instructions à fournir à l'éditeur de liens. L'utilisation des doubles quotes permet l'utilisation de caractères spéciaux.

```
@Link {"-lTFELMaterialSingleCristals"};
```

FIGURE 5 : Exemple d'utilisation de la directive `Link`.

La figure 5 donne un exemple d'utilisation de la directive `@Link`.

2.2.5 Gestion des bornes

Les variables d'états peuvent posséder des bornes intrinsèques : une température ne peut être négative, une porosité supérieure à 1. Ces bornes intrinsèques sont dénommées dans la suite « bornes physiques ». Par ailleurs, la corrélation définissant la propriété matériau a souvent été établie sur un certain domaine : les limites de ce domaine expérimental seront dénommées « bornes de validité » (de la corrélation).

```
// temperature bounds
@PhysicalBounds T in [0:*[;
@Bounds T in [733.15:873.15];
```

FIGURE 6 : Exemple d'utilisation des directives `Bounds` et `@PhysicalBounds`.

Le mot clé `@Bounds` Le mot clé `@Bounds` définit les bornes de validité de la corrélation utilisée pour chacune des différentes variables d'entrée. Le traitement du dépassement d'une de ces bornes dépend de l'interface utilisée.

Le mot clé `@PhysicalBounds` Le mot clé `@PhysicalBounds` définit les valeurs admissibles des différentes variables d'entrée. Le dépassement d'une de ces bornes doit arrêter le calcul.

La figure 6 donne un exemple d'utilisation des directives `@Bounds` et `@PhysicalBounds`.

2.2.6 Notion de paramètre

Les études réalisées dans le cadre de la plate-forme `pleiades` sont souvent paramétriques : il s'agit de mesurer l'impact d'une incertitude sur une quantité (conductivité thermique, coefficient d'échange) ou l'impact d'une valeur numérique (critère de convergence) sur les résultats du code.

Il est donc intéressant de pouvoir paramétrer une connaissance matériau. Plusieurs solutions sont possibles.

La plus évidente est de prévoir une ou plusieurs entrées supplémentaire. Par exemple, si l'on veut introduire un coefficient d'incertitude sur une conductivité thermique, il est toujours possible de passer ce coefficient comme un argument supplémentaire de la fonction qui réalise le calcul de cette conductivité.

Cette approche est très utilisée dans le code `licos` : une connaissance matériau livrée avec le code est reprise par l'utilisateur qui rajoute le ou les paramètres d'intérêts, recompile sa connaissance matériau dans une bibliothèque qui lui est propre et relance son calcul en utilisant cette bibliothèque. Cette même approche est utilisée par les utilisateurs du code `Cast3M`.

Cette approche ne peut que difficilement être utilisée dans les applications filières qui n'ont pas l'algorithme de gestion de dépendances de `licos` : en plus des étapes précédentes, il faudrait que l'utilisateur fasse les « branchements » à la main dans le code source de l'application, ce qui peut s'avérer complexe.

Le second désavantage de cette méthode est d'alourdir l'appel aux connaissances matériau, ce qui peut avoir un coût non négligeable [16].

Pour contourner ces difficultés, `mfront` introduit une notion de *paramètre*, c'est à dire une valeur, considérée comme constante au cours d'un calcul, mais pouvant être modifiée par l'utilisateur au moment de l'initialisation. Pour les propriétés, les paramètres sont des valeurs réelles. Pour les lois de comportements, les paramètres peuvent être des valeurs réelles ou entières [17]. Pour les modèles, les paramètres peuvent avoir différents types (valeurs réelles ou entières, tableaux, etc..).

Les interfaces qui le souhaitent peuvent mettre en place une gestion appropriée de ces paramètres [18]. Par exemple, l'interface `Cast3M` aux propriétés matériau et l'interface `umat` aux loi de comportement créent automatiquement une fonction permettant de modifier les différents paramètres qui sont stockés dans des variables globales. Il est facile de mettre en place un mécanisme générique, c'est à dire indépendant de toute connaissance matériau, permettant la modification des valeurs des paramètres depuis le jeu de données [19].

Cette solution corrige les défauts de la première approche. En particulier, l'accès aux valeurs des paramètres n'a pratiquement aucun coût numérique. Cette solution introduit cependant un inconvénient : tous les matériaux utilisant cette connaissance sont affectés alors que dans la première solution la modification pouvait être faite matériau par matériau. Nous pouvons donner deux cas pouvant poser problème :

16. Dans le cas de l'application `licos`, ce surcoût est généralement négligeable, mais il peut être nettement sensible sur des calculs $1D$.
17. Nous avons distingué deux types d'entier correspondant respectivement aux types C `int` et `unsigned short` (entiers positifs).
18. Par défaut, la plupart des interfaces décrites plus bas se contentent de traiter les paramètres de la même manière que des valeurs constantes.
19. Par exemple, dans `licos`, le mot clé `Parameter` a été introduit. Ce mot clé est suivi du nom de la bibliothèque matériau, du nom de la connaissance matériau, du nom du paramètre et de sa valeur. Cette solution devrait être généralisée à toutes les applications de la plate-forme.

— dans le cas d'une aiguille absorbante, il arrive que la partie inférieure de la colonne absorbante soit constituée de B_4C à un enrichissement en ^{10}B donné tandis que la partie supérieure soit constituée de B_4C à un enrichissement différent. Il est naturel de définir deux matériaux distincts pour décrire cette situation que l'ont décrira à l'aide des mêmes propriétés matériaux. Dans ce cas la notion de paramètre ne permet pas distinguer les parties basses et hautes.
— pour des crayons combustibles mélangeant des parties oxydes UO_2 et MOX, la notion de paramètre ne permet pas de modifier une propriété ou un modèle sur le combustible oxyde uniquement ;

Dans le cas de l'application `licos`, l'utilisateur pourra choisir la solution la plus adaptée à son besoin. Dans `Cast3M`, seule la première solution est possible.

Définition de paramètres Pour les propriétés et les lois de comportements, la directive `@Parameter` a été définie. Elle est suivie du nom du paramètre, du signe égal, de sa valeur par défaut et d'un point-virgule.

Pour les modèles, les paramètres propres aux modèles, utilisés pour leur spécialisation [20], sont distingués des paramètres dits globaux qui peuvent être mis en commun avec plusieurs modèles. Les premiers sont définis par la directive `@LocalParameter`, les seconds par la directive `@GlobalParameter`. Ces mots clés sont suivis du type du paramètre, de son nom et d'un point-virgule. Une valeur par défaut peut être fournie par la méthode `setDefaultValue`.

Paramètres automatiquement déclarés Les lois de comportement mécaniques introduites par `mfront` peuvent déclarer automatiquement un certain nombre de paramètres. La valeur du critère de convergence ou le nombre maximum d'itération sont des exemples de tels paramètres.

Nous détaillerons dans la documentation propre à chaque analyseur les paramètres déclarés automatiquement.

2.3 ANALYSEURS DISPONIBLES

Les connaissances matériaux supportées par `mfront` sont classées en trois catégories :
— les propriétés matériau ;
— les lois de comportement mécanique ;
— les modèles.

Pour traiter ces connaissances, différents analyseurs sont disponibles :
— l'analyseur `MaterialLaw` qui traite de propriétés matériau. La section 4 lui est consacrée ;
— l'analyseur `DefaultParser` qui permet de traiter tous types de lois de comportements ;
— l'analyseur `RungeKutta` qui propose la résolution d'une loi de comportement mécanique quelconque à l'aide de l'une des méthodes de RUNGE-KUTTA ;
— l'analyseur `IsotropicMisesCreep` qui gère exclusivement des lois de type NORTON, c'est à dire des comportements viscoplastiques incompressibles sans écrouissage de matériaux isotropes ;
— l'analyseur `IsotropicStrainHardeningMisesCreep` qui gère exclusivement les lois de type LEMAÎTRE [Chaboche 09], c'est à dire des comportements mécaniques viscoplastiques incompressibles avec écrouissage des matériaux isotropes ;
— l'analyseur `IsotropicPlasticMisesFlow` qui gère exclusivement les lois de comportement mécanique plastique incompressible des matériaux isotropes ;
— l'analyseur `MultipleIsotropicMisesFlows` qui gère une combinaison arbitraire d'écoulements des trois types précédents. Les différents écoulements sont alors supposés non couplés ;
— les analyseurs `Implicit` et `ImplicitII` qui simplifient la résolution d'une loi de comportement mécanique quelconque à l'aide d'une intégration implicite ;
— l'analyseur `Model` qui permet de traiter des modèles. Cet analyseur fait l'objet de la section 5.

20. Nous renvoyons à la documentation des codes de la plate-forme `pleiades` pour une utilisation concrète de ces paramètres.

Les analyseurs dédiés aux lois de comportement mécaniques sont largement majoritaires. Ils sont décrits dans une note spécifique [Helfer 13c], mais la lecture de la présente section est nécessaire pour l'aborder.

L'écriture de propriétés matériau est décrite en section 4, celle des modèles physico-chimiques en section 5.

La directive `@Parser` (voir figure 1) est utilisée pour préciser quel analyseur doit être utilisé pour interpréter le fichier considéré [21].

2.4 INTERFACES

`mfront` permet d'écrire les connaissances matériau de manière indépendante du code ou du langage cible [22].

Pour que celles-ci soient utilisables dans un code particulier, `mfront` introduit la notion d'*interface* : le code source généré dépend de l'interface choisie par l'utilisateur et cherche à s'intégrer au mieux dans le contexte du code cible et à en tirer le meilleur parti.

Insistons sur l'originalité de cette démarche. Dans des solutions plus habituelles, un code source fixe est généré puis une couche d'enrobage permet d'utiliser ce code dans des cibles distinctes. Cette solution conduit souvent à des conversions de données et des allocations mémoires inutiles. Dans l'approche `mfront`, le code source fixe est limité au maximum : il se limite souvent au code fourni par l'utilisateur (s'il n'est pas transformé) afin de créer un code qui s'utilise de manière optimale dans le contexte logiciel cible.

2.4.1 Interfaces disponibles

Par exemple, `mfront` est livré avec différentes interfaces pour les propriétés matériau :
- les interfaces `c`, `fortran`, C++ et `python` pour pouvoir utiliser les propriétés dans les langages du même nom ;
- l'interface `castem` pour pouvoir utiliser les propriétés dans le code aux éléments finis `Cast3M` ;
- l'interface `gnuplot` pour une visualisation des courbes d'évolution des propriétés matériau ;

Pour les lois de comportement, deux interfaces sont actuellement disponibles :
- l'interface `umat` pour une utilisation des lois générées dans le code aux éléments finis `Cast3M` [CEA 13]. Cette interface est décrite dans une note dédiée [Helfer 13b] ;
- l'interface `aster` pour une utilisation des lois générées dans le code aux éléments finis `Aster` [EDF 13]. Cette interface est décrite dans une note dédiée [Helfer 13a].

2.4.2 Ajout d'interfaces

Il est possible de rajouter dynamiquement de nouvelles interfaces par l'utilisation de la variable d'environnement `MFRONT_ADDITIONAL_LIBRARIES` qui doit désigner une liste de librairies automatiquement chargées au démarrage de `mfront` [23].

Ce mécanisme permet donc de créer des interfaces sans modifier `mfront`. Il est principalement utilisé par les applications de la plate-forme `pleiades` pour que `tfel` puisse être considéré comme un pré-recquis.

21. Cette directive est traditionnellement placée en tête du fichier `mfront` bien que cela ne soit pas nécessaire.
22. Précisons enfin si une interface vise à fournir un code utilisable dans un langage de programmation particulier, *le code généré est toujours du C++*. Il est ainsi possible d'utiliser toute la richesse de ce langage.
23. Ce mécanisme est utilisé par exemple par l'architecture `pleiades` pour générer l'interface `pleiades` aux propriétés matériau, interface qui évolue avec les versions de l'architecture.

3 UTILISATION DE MFRONT EN LIGNE DE COMMANDE

Nous décrivons dans ce paragraphe l'utilisation de `mfront` sous `unix`. Les commandes présentées restent valides sous `Windows` à condition d'utiliser l'un des environnements `msys` ou `cygwin`.

3.1 APPEL À MFRONT EN LIGNE DE COMMANDE

`mfront` s'utilise ainsi :

```
mfront [options] fichier1.mfront fichier2.mfront ...
```

L'option la plus utilisée est `--interface` qui permet de choisir la cible de la compilation.

Les fichiers générés se trouvent généralement dans un sous-répertoire nommé `src` pour les fichiers source et `include` pour les fichiers d'entête.

Cette commande conduit à la génération :
- de fichiers sources ;
- de fichiers de dépendances ;
- de fichiers de cibles.

Ces derniers fichiers permettent d'utiliser `mfront` de manière séquentielle. `mfront` peut d'abord être appelé sur plusieurs fichiers pour générer les fichiers sources puis lancer la compilation des librairies ainsi :

```
# Génération des fichiers sources
mfront -interface=?? fichier1.mfront
mfront -interface=?? fichier2.mfront
...
# Génération d'un fichier Makefile et lancement
de la compilation
mfront -obuild
```

Dans une utilisation avancée, l'utilisateur peut avoir un contrôle assez fin sur le processus de compilation. Nous renvoyons le lecteur à l'annexe A pour plus de détails.

Note sur la génération des fichiers en parallèle La génération de fichiers `mfront` en parallèle est possible. Pour cela, un verrou global est posé au moment de l'écriture des fichiers afin que les différents processus `mfront` ne modifient pas les mêmes fichiers en même temps (fichiers cibles et fichiers de dépendances notamment).

Cela pourrait conduire à des blocages si `mfront` se termine sans libérer ce verrou[24]. Dans ce cas, il est nécessaire de retirer ce verrou global à la main. Sous `linux`, il s'agit de supprimer le fichier :

```
/dev/shm/sem-mfront.$UID
```

Un moyen plus brutal consiste à redémarrer la machine.

3.2 OPTIONS DE LA LIGNE DE COMMANDE

L'option `--help` L'option `--help` liste l'ensemble des options disponibles et quitte le programme.

24. Nous devons cependant préciser que ce problème ne s'est présenté qu'une unique fois.

L'option `--list-parsers` L'option `--list-parsers` affiche la liste des analyseurs disponibles et quitte le programme.

L'option `--help-keywords-list` L'option `--help-keywords-list` liste l'ensemble des mots clés d'une interface et quitte le programme. Par exemple :

```
mfront --help-keywords-list=MaterialLaw
```

L'option `--help-keyword` L'option `--help-keyword` affiche la documentation d'un des mots clés d'une interface et quitte le programme. Par exemple :

```
mfront --help-keyword=MaterialLaw:@Function
```

L'option `--version` L'option `--version` affiche la version de `mfront` utilisée et quitte le programme.

L'option `--interface` L'option `--interface` spécifie la ou les interfaces à utiliser pour générer les fichiers sources. Les différentes interfaces sont séparées par une virgule.

L'option `--verbose` L'option `--verbose` demande l'affichage de messages supplémentaires.

L'option `--warning` L'option `--warning` demande l'affichage de messages d'avertissement supplémentaires.

L'option `--debug` L'option `--debug` modifie la génération des fichiers de compilation et son effet dépend de l'analyseur choisi. [25].

L'option `--make` L'option `--make` conduit à la génération d'un fichier `Makefile.mfront`.

L'option `--omake` L'option `--omake` conduit à la génération d'un fichier `Makefile.mfront` avec des directives de compilation optimisées.

Les options de compilation optimisées sont données par le résultat de la commande

```
tfel-config --oflags
```

L'option `--clean` L'option `--clean` conduit à la génération d'un fichier `Makefile.mfront` et lance la commande :

```
make -C src -f Makefile.mfront clean
```

25. Dans un mode de fonctionnement normal de `mfront` insère des directives `#line` pour repérer la position des erreurs dans le fichier `mfront` d'origine (le compilateur indique l'erreur dans le fichier `mfront` d'origine et non dans le fichier généré). L'option `--debug` désactive l'insertion de ces directives `#line`.

L'option `--build` L'option `--build` conduit à la génération d'un fichier `Makefile.mfront` et lance la commande :

```
make -C src -f Makefile.mfront
```

L'option `--obuild` L'option `--obuild` conduit à la génération d'un fichier `Makefile.mfront` avec des directives de compilation optimisées et lance la commande :

```
make -C src -f Makefile.mfront
```

L'option `--search-path` Un fichier `mfront` peut inclure d'autres fichiers grâce à la directive `@Import`. L'option `--search-path` permet d'ajouter un ou plusieurs chemins de recherche pour ces fichiers. Plusieurs chemins peuvent être précisés en les séparant par le caractère `':'`.

L'option `--include` L'option `--include` est synonyme de l'option `--search-path`.

L'option `--nodeps` L'option `--nodeps` empêche l'inclusion des règles de dépendances dans le fichier `Makefile.mfront` généré. Cette option a été introduite pour éviter certains problèmes sous `Windows`.

L'option `--silent-build` L'option `--silent-build` est spécifiée avec l'option `false`, les lignes de compilation utilisées seront affichées.

L'option `--otarget` L'option `--otarget` permet de ne compiler qu'une des librairies de manière optimisée. Cette option est suivie du nom de la cible.

L'option `--target` L'option `--target` permet de ne compiler qu'une des librairies.

L'option `--win32` L'option `--win32` est utilisée pour permettre la compilation croisée sous `linux` de librairies dynamiques pour `Windows`. Cette possibilité est décrite en annexe A.2.

Ajout de directives Il est possible d'ajouter des directives à un fichier depuis la ligne de commande.

La ligne de commande suivante :

```
mfront -@AsterCompareToNumericalTangentOperator=true implicit.mfront
```

donnera le même résultat que si on avait rajouter la ligne suivante en tête du fichier `implicit.mfront` :

```
@AsterCompareToNumericalTangentOperator true ;
```

La règle générale est la suivante : toute option de la ligne de commande commençant par une arobase est interprétée comme une directive à ajouter en début de l'analyse du fichier.

4 PROPRIÉTÉS MATÉRIAUX

Les propriétés matériau sont l'un des éléments de connaissance essentiel des matériaux. Une propriété matériau est une fonction d'un ensemble de variables d'état thermodynamique du matériau.

4.1 LES DIFFÉRENTS TYPES DE VARIABLES

Ce paragraphe décrit les différents types de variables manipulés lorsque l'on écrit des propriétés matériau.

4.1.1 Variable de sortie

La variable de sortie est par défaut nommée `res`. Ce nom peut être changé par la directive `@Output`.

4.1.2 Variables d'entrée

Les variables d'entrée des propriétés matériau sont définies par la directive `@Input`.

Ces variables d'entrée peuvent utiliser la méthode `setGlossaryName` pour définir son nom de glossaire. Si aucun nom de glossaire adapté n'existe, ces variables peuvent utiliser la méthode `setEntryName` pour être utilisable dans l'architecture `pleiades`.

Nous avons vu au paragraphe 2.2.5 comment affecter des bornes aux variables d'entrée.

4.1.3 Variables statiques

La directive `@StaticVariable` permet d'introduire une variable statique au sens du langage C++.

4.1.4 Paramètres

La directive `@Parameter` a été décrite au paragraphe 2.2.6.

4.1.5 Constantes

La directive `@Constant` est synonyme de la directive `@StaticVariable` pour des variables réelles.

4.2 MOTS CLÉS SPÉCIFIQUES

La directive `@Namespaces` a été introduite pour inclure les fonctions ou les classes générées dans un espace de nom distinct. Ce mot clé n'est utilisé que par l'interface C++ et peut éventuellement être utilisée par l'interface `pleiades`.

La directive `@UseTemplate` demande la génération de fonctions ou de classes paramétrées par le type numérique utilisé (par défaut, les interfaces utilisent généralement des valeurs réelles en double précision). Cette directive n'est pour l'instant utilisée que par l'interface C++.

4.3 INTERFACES DISPONIBLES

Les interfaces disponibles par défaut pour les propriétés matériaux sont :

- l'interface `c` pour une utilisation dans des codes écrits dans ce langage. Cette interface est décrite en annexe B ;
- l'interface `fortran` pour une utilisation dans des codes écrits dans ce langage ;
- l'interface `c++` pour une utilisation dans des codes écrits dans ce langage ;
- l'interface `castem` pour une utilisation dans le code aux éléments finis `Cast3M`. Cette interface est décrite en annexe C ;
- l'interface `python` pour une utilisation dans des codes écrits dans ce langage. Cette interface est décrite en annexe D ;
- l'interface `gnuplot` pour une visualisation des courbes d'évolution des propriétés matériau en fonction de leur paramètres ;

4.4 EXEMPLE DU MODULE D'YOUNG DU SiC

SNEAD a proposé une corrélation donnant le module d'YOUNG du SiC en fonction de la température et de la porosité sous la forme :

$$E\left(T,p\right)=\left(E_0-B\,T\,\exp\left(-\frac{T_0}{T}\right)\right)\exp\left(-C\,p\right) \tag{1}$$

où :
- T est la température ;
- p est la porosité ;
- E_0, B, T_0 C sont des coefficients.

La figure 7 reproduit le fichier `mfront` implémentant cette corrélation.

Le mot clé `@Parser` La première ligne, commençant par le mot clé `@Parser`, décrit le type d'analyseur utilisé, ici `MaterialLaw`.

Le mot clé `@Law` La second ligne, commençant par le mot clé `@Law`, donne le nom de la propriété matériau. Ce nom sera utilisé pour l'appel à la propriété matériau (nom de la fonction ou de la classe générée suivant l'interface utilisée).

Le mot clé `@Material` Le mot clé `@Material`, non utilisé ici, sert à regrouper plusieurs propriétés d'un même matériau dans une même bibliothèque.

Les mots clé `@Author` et `@Date` La troisième et la quatrième ligne renseignent respectivement l'auteur du fichier et la date de création à l'aide des mots clés `@Author` et `@Date`.

Le mot clé `@Description` Le mot clé `@Description` permet de donner les références bibliographiques d'où la corrélation est extraite ou tout commentaire sur son origine.

Le mot clé `@Output` Le mot clé `@Output` change le nom de la propriété matériau calculée. Par défaut, celle-ci s'appelle `res`.

```
@Parser MaterialLaw;
@Law    SIC_YOUNGMODULUS_SNEAD;
@Author Thomas Helfer;
@Date   2007-12-06;

@Description{
  Journal of Nuclear Materials 371 ( 2007 ) 329-377
  Handbook of SiC properties for fuel performance modeling
  L.L. SNEAD et al.
  Pages 339 et 340 - équations (17) et (18)
}

// changing the name of output
@Output E;

// input of the law
@Input T,p;

// variables bounds
@PhysicalBounds T in [0:*[;
@PhysicalBounds p in [0:1];

@Function{
  const real E0 =  460.00E9 ;
  const real B  =    0.04E9 ;
  const real T0 =  962.00   ;
  const real C  =    3.57   ;
  E = (E0-(B*T*exp(-T0/T)))*exp(-C*p);
} // end of function
```

FIGURE 7 : Implantation du module d'YOUNG du SiC en `mfront`.

Le mot clé `@Input` Le mot clé `@Input` donne la liste des variables d'état dont dépend la propriété. Les variables peuvent être déclarées en une fois ou par plusieurs utilisations consécutives du mot clé `@Input`. L'ordre de déclaration des variables peut être important suivant l'interface utilisé : ainsi la fonction générée par les interfaces `c` ou `castem` prendront leurs arguments dans l'ordre de déclaration utilisé dans le fichier `mfront`.

La méthode `setGlossaryName` La méthode `setGlossaryName`, non utilisée ici, permet de préciser un nom associé à une variable d'entrée. Ce nom est utilisé par les applications de la plateforme `pleiades` pour assurer l'échange d'informations.

Le mot clé `@PhysicalBounds` Le mot clé `@PhysicalBounds` définit les bornes physiques des variables d'état définies plus haut. Ici nous indiquons qu'une température ne peut être négative et qu'une porosité est comprise entre 0 et 1. Les cas de dépassement de ces bornes sont très souvent liés à des problèmes graves des codes (erreur de programmation, solutions inacceptables). Pour ces raisons, la plupart des interfaces signaleront une erreur au code appelant, conduisant généralement à son arrêt.

Le mot clé `@Function` Le mot clé `@Function` permet de définir le code (en `c++`) utilisé pour définir la propriété matériau. Le fait que le code soit en fait du `c++` est ici anecdotique tant le code est proche de l'équation (1).

5 MODÈLES

Deux types de modèles sont usuellement distingués :

- les modèles demandant une résolution globale sur l'ensemble du domaine considéré. Ces modèles concernent généralement la diffusion (thermique ou chimique) et l'équilibre mécanique. Ce type de modèle étant coûteux à développer, nécessitant le recours à des techniques de discrétisation telles que les éléments finis, ils sont généralement écrits de manière générique, la spécialisation au problème concret à traiter se faisant en renseignant ou des propriétés matériau ou des lois de comportement ;
- les modèles dits « point » décrivant l'évolution locale du matériau en fonction de son état thermodynamique local. Ce type de modèle regroupe les modèles de production de gaz de fission, de concentration isotopique (en l'absence de diffusion), de corrosion, de gonflement, etc.. ;

`mfront` ne permet pas de traiter les modèles demandant une résolution globale. Nous nous intéresserons uniquement aux modèles « point ».

Pour traiter ces modèles, `mfront` propose l'analyseur `Model`.

5.1 GÉNÉRALITÉS SUR LES MODÈLES « POINTS »

Les modèles « point » s'appuient sur un ou plusieurs domaines. Les valeurs des différentes variables d'états du matériau sont représentées par des champs aux points de discrétisation du ou des domaines considérés.

Comme les lois de comportement, les modèles nécessitent *a priori* le recours à des méthodes d'intégration. En pratique, pour les modèles courants, notamment les modèles de gonflement, une estimation par une méthode des trapèzes (les valeurs des champs d'entrée étant prises en milieu de pas de temps) est généralement suffisante.

5.2 MODÈLE DE GONFLEMENT SOLIDE DE L'$(U, Pu)\,C$

Nous décrivons ici l'introduction du modèle de gonflement solide de l'$(U, Pu)\,C$ [Pelletier 04].

Le code source est donné en figure 8.

Le mot clé `@Parser` La première ligne, commençant par le mot clé `@Parser`, décrit le type d'analyseur utilisé, ici `Model`.

Le mot clé `@Model` La second ligne, commençant par le mot clé `@Model`, donne le nom du modèle.

Les mots clé `@Author` et `@Date` La troisième et la quatrième ligne renseignent respectivement l'auteur du fichier et la date de création à l'aide des mots clés `@Author` et `@Date`.

Le mot clé `@Description` Le mot clé `@Description` permet de donner les références bibliographiques d'où le modèle est extrait.

Le mot clé `@LocalParameter` Le mot clé `@LocalParameter` permet de déclarer une variable locale, propre au modèle.

```
@Parser   Model;
@Model    SolidSwellingModel;
@Material UPuC;
@Author   Helfer Thomas;
@Date     06 Déc. 2007;

@Output s;
s.setGlossaryName("SolidSwelling");
s.setDefaultInitialValue(0.);
s.setDepth(1);

@Input Bu;
Bu.setGlossaryName("BurnUp");
Bu.setDepth(1);

@Input p;
p.setGlossaryName("Porosity");
p.setDepth(1);

@Function compute
{
  const real coef1 = 8.e-3;
  const real coef2 = 4.e-2;
  const real p_    = 0.5*(p+p_1);
  s = s_1 + coef1*exp(coef2-p_)*(Bu-Bu_1);
} // end of function compute
```

FIGURE 8 : Implantation du modèle de gonflement solide de l'$(U, Pu)\,C$ en `mfront`.

Le mot clé `@GlobalParameter` De la même manière, le mot clé `@GlobalParameter` permet de déclarer un paramètre dont la valeur sera lue dans le fichier d'entrée.

Le mot clé `@Output` Le mot clé `@Output` permet de préciser un des champs de sortie du modèle.

Le mot clé `@Input` Le mot clé `@Input` permet de préciser un des champs d'entrée du modèle.

La méthode `setDepth` La méthode `setDepth` permet de préciser la profondeur d'un champ, c'est à dire le nombre de pas de temps antérieurs auquel l'on a accès. Une profondeur de `1` permet d'avoir accès à la valeur courante (en fin de pas de temps) du champ et à sa valeur au début du pas.

La méthode `setGlossaryName` La méthode `setGlossaryName` permet de préciser le nom d'un champ dans `pleiades`.

Le mot clé `@Function` Le mot clé `@Function` permet d'introduire l'implantation du modèle. Un même modèle pouvant avoir plusieurs fonctions, un identifiant, ici `compute`, doit être précisé. L'implantation du modèle est faite en C++. Dans le code présenté, la variable `s_1` représente la valeur du champ s au début du pas de temps et la variable s sa valeur en fin de pas de temps.

5.3 INTERFACE DISPONIBLE

À ce jour, aucune interface pour les modèles n'est disponible par défaut. Certaines applications de la plateforme `pleiades` fournissent leurs propres interfaces dédiées (`licos`, `germinal`).

RÉFÉRENCES

[CEA 13] CEA . *Site Cast3M*, 2013.

[Chaboche 09] CHABOCHE JEAN-LOUIS, LEMAÎTRE JEAN, BENALLAL AHMED et DESMORAT RODRIGUE. *Mécanique des matériaux solides*. Dunod, Paris, 2009.

[d'Arrigo 12] D'ARRIGO JOSÉ et GENTET DAVID. *Notice d'utilisation de la base de données matériau du LE2S*. Rapport technique, DER/SESI/LE2S, 2012.

[EDF 13] EDF . *Site du Code_Aster*, 2013.

[Foundation 06] FOUNDATION FREE SOFTWARE. *GNU coding standards*, Novembre 2006. http ://www.gnu.org/prep/standards/http ://www.gnu.org/prep/standards/.

[GNU 07] GNU LE PROJET. *Documentation du logiciel Autoconf*, 2007.

[Helfer 10] HELFER THOMAS, CASTELIER ÉTIENNE, BRUNO ÉRIC, GOHIER ÉRIC et BONHOMME CHRISTIAN. *Ajout de connaissances matériau (propriétes matériau, lois de comportement et modèles - au code Celaeno à l'aide générateur code MFront*. Rapport technique 10-002, DEC/SESC/LSC, 2010.

[Helfer 11a] HELFER THOMAS. *Présentation de mfront*, Juin 2011.

[Helfer 11b] HELFER THOMAS, MICHEL BRUNO, BOUINEAU VINCENT et COSTOMIRIS AUDREY. *Bilan de l'optimisation de l'intégration des lois de comportement mécanique dans Alcyone et Germinal V2*. Note technique 11 - 005, CEA DEN/DEC/SESC/LSC, Février 2011.

[Helfer 13a] HELFER THOMAS. *L'interface aster aux lois de comportement mécanique de MFront*. Note technique, CEA DEN/DEC/SESC/LSC, 2013. En cours de rédaction.

[Helfer 13b] HELFER THOMAS. *L'interface umat aux lois de comportement mécanique de MFront*. Note technique, CEA DEN/DEC/SESC/LSC, 2013. En cours de rédaction.

[Helfer 13c] HELFER THOMAS, CASTELIER ÉTIENNE, BLANC VICTOR et JULIEN JÉRÔME. *Le générateur de code mfront : écriture de lois de comportement mécanique*. Note technique 13-020, CEA DEN/DEC/SESC/LSC, 2013.

[Michel 09] MICHEL BRUNO. *Étude de faisabilité de la génération automatique des lois de comportement mécanique non linéaires dans la plate-forme pleiades*. Note technique 09-028, DEC/SESC/LSC, Novembre 2009.

[Olagnon 13] OLAGNON JULIEN et GARNIER CHRISTOPHE. *Analysis of a new integrator for finite element code for the the calculation of fuel rods thermal-mechanical beahviour : mfront*. Rapport technique FS1-0010103, Areva-NP, 2013.

[Pelletier 04] PELLETIER M. *Recueil de données sur le combustible carbure mixte UPuC*. Note technique 02-018 Indice 1, CEA DEN/DEC/SESC/LSC, Avril 2004.

[Proix 13] PROIX JEAN-MICHEL. *Intégration des lois de comportement à l'aide de MFront : bilan des tests réalisés pour l'utilisation avec Code Aster*. Rapport technique H-T64-2013-00922-FR, EDF-R&D/AMA, 2013.

[Software 13] SOFTWARE TIOBE. *Programming Community Index for June 2013*, 2013.

[Von Rossum 07] VON ROSSUM G. *Python Library Reference*, 2007.

[Wikipédia 09] WIKIPÉDIA . *Name mangling*. 2009.

LISTE DES FIGURES

ANNEXE A GESTION DE LA COMPILATION

En plus de générer des fichiers source, `mfront` peut gérer leurs compilations sous forme de librairies dynamiques. Cette compilation se fait en plusieurs étapes :

- génération d'un fichier `Makefile.mfront` dans le sous-répertoire des sources `src`. La génération du fichier Makefile se fait en analysant le répertoire `src` qui contient (après la phase de génération des sources) différents fichiers précisant les noms des libraires à générer, la liste de leurs sources et de leurs dépendances, etc.. ;
- le lancement du processus de compilation. Il s'agit essentiellement d'un appel à l'utilitaire `make`.

ANNEXE A.1 VARIABLES AFFECTANT LE PROCESSUS DE COMPILATION

Bien que cela ne soit généralement pas nécessaire, l'utilisateur `mfront` peut affecter le processus de compilation à l'aide de variables d'environnement. Cela est utile dans les situations suivantes :

- utilisation d'une librairie tierce (variables `INCLUDES` et `LDFLAGS`) ;
- utilisation d'un compilateur différent du compilateur système (variables `CC` et `CXX`) ;
- utiliser des options de compilation particulières (variables `CFLAGS` et `CXXFLAGS`) ;
- génération de binaire pour un système différent (cross-compilation) (variables `CC`, `CXX` `RANLIB`, `DLLTOOL` et `AR`) ;

Le fichier `Makefile.mfront` suit les règles des fichiers `Makefile` : il est sensible aux variables d'environnement suivantes :

- `CC` qui désigne le compilateur `C` à utiliser ;
- `CXX` qui désigne le compilateur C++à utiliser. Ce compilateur est également utilisé pour l'édition de liens ;
- `INCLUDES` qui désigne des directives de préprocesseur à utiliser en plus des directives définies par les sources ;
- `CFLAGS` qui désigne les directives de compilation à utiliser pour les sources `c`. Si cette variable n'est pas définie, `mfront` utilisera les valeurs suivantes :
  - la sortie de la commande `tfel-config --oflags` si l'une des options `--obuild` ou `--omake` a été utilisée ;
  - `-O2` sinon.
- `CXXFLAGS` qui désigne les directives de compilation à utiliser pour les source C++. Si cette variable n'est pas définie, `mfront` utilisera les valeurs suivantes :
  - la sortie de la commande `tfel-config --oflags` si l'une des options `--obuild` ou `--omake` a été utilisée ;
  - `-O2` sinon.
- `LDFLAGS` qui désigne des directives à passer à l'éditeur de liens ;
- `RANLIB`, `DLLTOOL` et `AR` qui ont été introduits pour la compilation croisée de librairies dynamiques pour `Windows` depuis `linux`.

ANNEXE A.2 COMPILATION CROISÉE DE LIBRAIRIES DYNAMIQUES POUR WINDOWS DEPUIS LINUX

À partir d'une version `linux` de `mfront`, il est possible de générer des librairies dynamiques pour `Windows` dans le cas des propriétés matériau. Il faut cependant que ces propriétés ne fassent pas appel à des librairies externes, ou, si tel est le cas de disposer des versions `Windows` de ces librairies [26].

26. Pour compiler des lois de comportements, il serait nécessaire de disposer des librairies mathématiques de TFEL pour `Windows` : il faut donc compiler l'ensemble de `TFEL` et `mfront` en cross-compilation et utiliser la version `Windows` de `mfront`. Ceci est effectivement possible via l'émulateur `wine`, mais cette manière de procéder sort du cadre de ce paragraphe.

Pour générer des librairies dynamiques pour `Windows` depuis `linux`, il est tout d'abord nécessaire de disposer d'un cross-compilateur. Sous `Debian`, l'utilisateur pourra installer les paquets `mingw32` (`Windows` 32 bits) ou `mingw64` (`Windows` 32 et 64 bits).

Il est alors nécessaire de définir les variables d'environnement `CC`, `CXX` `RANLIB`, `DLLTOOL` et `AR` et d'utiliser l'option `--win32`.

Voici un exemple de génération de librairie dynamique pour `Windows` depuis `linux` :

```
AR=i586-mingw32msvc-ar RANLIB=i586-mingw32msvc-ranlib
DLLTOOL=i586-mingw32msvc-dlltool CXX=i586-mingw32msvc-g++ ../../src/mfront
-win32 -obuild -interface=c YoungModulus.mfront
```

ANNEXE B L'INTERFACE C

Cette annexe est dédiée à l'interface `c` aux propriétés matériau.

ANNEXE B.1 NOMS DES FONCTIONS GÉNÉRÉES

Par propriété matériau, une ou deux fonctions sont générées.

La première calcule effectivement la valeur de la propriété matériau en reprenant le code fourni après la directive `@Function`. Un nom de propriété doit nécessairement avoir été défini par la directive `@Law`. Deux cas se présentent alors :

- si un nom de matériau a été spécifié, le nom de la fonction sera `mat_law`, où `mat` est le nom du matériau et `law` le nom de la propriété ;
- si aucun nom de matériau n'a été spécifié, le nom de la fonction sera `law`, où `law` est le nom de la propriété.

Cette fonction prend des réels en double précision en arguments, dans l'ordre de déclaration des entrées de la propriété matériau, introduites par la directive `@Input`.

ANNEXE B.2 GESTION DES BORNES

Une deuxième fonction est générée si des bornes ont été fournies. Cette deuxième fonction renvoie :

- un nombre positif si une borne de validité de la loi est violée ;
- un nombre négatif si une borne physique l'est ;
- `0` si aucune borne n'est violée.

La valeur absolue du nombre est l'ordre de déclaration de la variable mise en cause.

Cette fonction a le nom de la première fonction suivi de `_checkBounds`. Elle prend les mêmes arguments, donnés dans le même ordre, que la première fonction.

ANNEXE B.3 NOMS DE LA LIBRAIRIE GÉNÉRÉE

Différents cas se présentent :

- si un nom de librairie a été défini par la directive `@Library`, le nom de la librairie sera `libXXX.so` où `XXX` est le nom fourni par l'utilisateur ;
- si aucun nom de librairie n'est spécifié mais qu'un nom de matériau a été fourni, le nom de la librairie sera `libMMM.so` où `MMM` est le nom du matériau fourni par l'utilisateur ;
- si aucun nom de librairie ni aucun nom de matériau n'ont été spécifiés, la librairie s'appellera `libMaterialLaw.so`.

ANNEXE C L'INTERFACE CASTEM

Cette annexe est dédiée à l'interface `castem` aux propriétés matériau. L'utilisation de cette interface est décrite dans l'annexe E. Une utilisation directe de la librairie générée nécessite une version modifiée de `Cast3M` utilisée et maintenue au sein de la plateforme `pleiades`. L'utilisation dans la version officielle est possible, mais il faut passer par une modification de la méthode `compute` et une recompilation partielle de `Cast3M`.

ANNEXE C.1 NOM DE LA FONCTION GÉNÉRÉE

Par propriété matériau, une fonction est générée en reprenant le code fourni après la directive `@Function`.

Un nom de propriété doit nécessairement avoir été défini par la directive `@Law`. Deux cas se présentent alors :

- si un nom de matériau a été spécifié, le nom de la fonction sera `mat_law`, où `mat` est le nom du matériau et `law` le nom de la propriété ;
- si aucun nom de matériau n'a été spécifié, le nom de la fonction sera `law`, où `law` est le nom de la propriété.

Cette fonction prend un tableau de réels en double précision en argument. Dans ce tableau, les valeurs des entrées de la propriété matériau, introduites par la directive `@Input`, doivent être fournies dans l'ordre de déclaration.

ANNEXE C.2 GESTION DES BORNES

Si une borne physique est violée, la fonction renvoie la valeur spéciale `NaN` (pour `not a number`), définie dans la norme IEEE 754. Cette valeur de retour conduit à un arrêt des calculs.

Le traitement d'une violation des bornes de validité expérimentale dépend de la politique définie par l'utilisateur à l'aide de la variable d'environnement `CASTEM_OUT_OF_BOUND_POLICY`. Trois politiques sont sont possibles :

- `STRICT`, qui traite un dépassement comme une erreur, avec un traitement similaire à ce qui est fait pour les bornes physiques ;
- `WARNING`, qui conduit à afficher un message d'avertissement sans génération d'erreur ;
- `NONE`, qui ignore le dépassement ;

Si la variable `CASTEM_OUT_OF_BOUND_POLICY` n'est pas définie, les dépassements sont ignorés.

ANNEXE C.3 NOMS DE LA LIBRAIRIE GÉNÉRÉE

Différents cas se présentent :

- si un nom de librairie a été défini par la directive `@Library`, le nom de la librairie sera `libCastemXXX.so` où `XXX` est le nom fourni par l'utilisateur ;
- si aucun nom de librairie n'est spécifié mais qu'un nom de matériau a été fourni, le nom de la librairie sera `libCastemMMM.so` où `MMM` est le nom du matériau fourni par l'utilisateur ;
- si aucun nom de librairie ni aucun nom de matériau n'ont été spécifiés, la librairie s'appellera `libCastemMaterialLaw.so`.

ANNEXE D L'INTERFACE PYTHON

Cette annexe est dédiée à l'interface `python` aux propriétés matériau.

Par propriété matériau, deux fonctions sont générées :
- la première implante effectivement la propriété matériau ;
- la seconde fonction est en fait propre à la librairie générée et déclare les différentes informations implantées dans cette librairie. Elle ne sera pas décrite plus avant dans cette annexe.

ANNEXE D.1 NOM DE LA FONCTION GÉNÉRÉE

La première fonction générée reprend le code fourni après la directive `@Function`.

Un nom de propriété doit nécessairement avoir été défini par la directive `@Law`. Deux cas se présentent alors :
- si un nom de matériau a été spécifié, le nom de la fonction sera `mat_law`, où `mat` est le nom du matériau et `law` le nom de la propriété ;
- si aucun nom de matériau n'a été spécifié, le nom de la fonction sera `law`, où `law` est le nom de la propriété.

Cette fonction prend un tableau de réels en double précision en argument. Dans ce tableau, les valeurs des entrées de la propriété matériau, introduites par la directive `@Input`, doivent être fournies dans l'ordre de déclaration.

ANNEXE D.2 GESTION DES BORNES

Si une borne physique est violée, la fonction renvoie une exception du type `RuntimeError`.

Le traitement d'une violation des bornes de validité expérimentale dépend de la politique définie par l'utilisateur à l'aide de la variable d'environnement `PYTHON_OUT_OF_BOUND_POLICY`. Trois politiques sont possibles :
- `STRICT`, qui traite un dépassement comme une erreur, avec un traitement similaire à ce qui est fait pour les bornes physiques ;
- `WARNING`, qui conduit à afficher un message d'avertissement sans génération d'erreur ;
- `NONE`, qui ignore le dépassement ;

Si la variable `PYTHON_OUT_OF_BOUND_POLICY` n'est pas définie, les dépassements sont ignorés.

ANNEXE D.3 NOMS DE LA LIBRAIRIE GÉNÉRÉE

Différents cas se présentent :
- si un nom de librairie a été défini par la directive `@Library`, le nom de la librairie sera `XXX.so` où `XXX` est le nom fourni par l'utilisateur ;
- si aucun nom de librairie n'est spécifié mais qu'un nom de matériau a été fourni, le nom de la librairie sera `MMM.so` où `MMM` est le nom du matériau fourni par l'utilisateur ;
- si aucun nom de librairie ni aucun nom de matériau n'ont été spécifiés, la librairie s'appellera `MaterialLaw.so`.

ANNEXE E APPEL DE PROPRIÉTÉS MATÉRIAUX EXTERNES DANS CAST3M

Pour les besoins du projet `pleiades`, nous avons modifié le code aux éléments finis `Cast3M` pour pouvoir appeler des propriétés matériau définies dans des librairies externes. Cette section décrit les modifications faites au code aux éléments finis `Cast3M` et l'utilisation de librairies externes depuis le `gibiane`.

ANNEXE E.1 FICHIERS MODIFIÉS ET IMPLÉMENTATION

Modification des sources de `Cast3M` L'évolution proposée demande de modifier deux fichiers source du code `Cast3M` :

- `varinu.eso`, implémentation `esope` de l'opérateur `VARI` chargé de l'appel aux lois externes ;
- `excha1.eso`, implémentation `esope` de l'opérateur `EXTR` pour l'extraction des composantes variables d'un champ par élément et des paramètres dont dépendent les composantes variables. Cette fonctionnalité est utilisée par les procédures `TRANSNON` et `PASAPAS` ;

Ces modifications ont pour objet de pouvoir définir une propriété de matériau à partir d'une table. La syntaxe utilisée est décrite au paragraphe suivant.

Portabilité de la solution proposée L'appel aux fonctions externes se fait par l'appel aux fonctions `dlopen`, `dlsym` et `dlclose` déclarées dans le fichier d'entête `dlfcn.h`. Ces fonctions ont été définies dans la norme POSIX [27] et sont de ce fait utilisables sur la plupart des systèmes dérivés d'UNIX actuels. Sous `Windows`, les fonctions système `LoadLibrary` et `GetProcAddress` sont utilisées.

Nous faisons également trois hypothèses qui peuvent limiter la portabilité de la solution proposée :

- nous supposons que le type double précision du fortran (`REAL*8`) et le type double précision du C/C++(`double`) correspondent.
- nous supposons que le type entier du fortran (`INTEGER`) correspond au type entier du C/C++(`int`) ;
- nous supposons que la taille d'une variable entière de type (`int`) est plus grande que la taille d'un pointeur ;

Les deux premières hypothèses sont généralement vérifiées, mais aucune norme ne l'impose à notre connaissance. La dernière hypothèse est sans doute la plus restrictive : cette pratique de transférer des pointeurs par des entiers a longtemps été utilisée en C, mais il semble que cela ne soit plus le cas sur les architectures 64 bits actuelles [Foundation 06]. Il est intéressant de noter qu'une hypothèse similaire est faite en python pour l'implémentation de l'appel aux fonctions externes via le module `dl` [Von Rossum 07].

La nécessité de convertir un pointeur en entier est liée à un échange nécessaire entre le fortran et le C++. Cette restriction peut *a priori* être levée en échangeant entre le fortran et le C++non plus un entier mais un tableau de taille adéquate et de recopier « caractère par caratère » le pointeur dans ce tableau et effectuer l'opération inverse lorsque le pointeur doit être récupéré [28].

Coût de la solution proposée L'utilisation de la solution proposée peut avoir un coût car les bibliothèques externes sont chargées et déchargées pour chaque évaluation d'une propriété de matériau sur un domaine géométrique. *A priori*, ce coût est faible, sauf pour des domaines petits. Nous avons prévu la possibilité de « pré-lier » la bibliothèque `Cast3M` avec une bibliothèque de propriétés de matériau, ce qui éviterait le chargement et le déchargement de cette bibliothèque.

27. POSIX est l'acronyme de Portable Operating System Interface for uniX. Cette norme a pour objet d'établir une interface de programmation supportée par l'ensemble des systèmes d'exploitation dérivés d'UNIX.
28. Ceci se base sur le fait que toute structure C à une taille multiple de la taille d'un caractère.

ANNEXE E.2 UTILISATION DANS CAST3M

Nous détaillons ici comment déclarer un modèle et un matériau utilisant une propriété définie dans une fonction externe. Nous avons choisi de prendre pour exemple un modèle thermique isotrope dont la conductivité thermique est évaluée par une fonction externe.

Le code `gibiane` utilisé est décrit en figure 9.

```
* Création d'un modèle thermique isotrope
ModT1 = 'MODELISER' s1 'THERMIQUE' 'ISOTROPE' ;

* Création d'une table contenant les données relatives
* à la loi externe :
* - 'LOI' contient le nom de la fonction appelée
* - 'LIBRAIRIE' contient le nom de la librairie externe
* dans laquelle cette fonction est définie
* - 'PARAMETRES' contient la liste des paramètres dont dépend
* la fonction appelée
Tmat = 'TABLE';
Tmat. 'LOI' = 'UO2_THERMALCONDUCTIVITY_METEOR' ;
Tmat. 'LIBRAIRIE' = 'libinterface.so' ;
Tmat. 'PARAMETRES' = 'MOTS' 'T' 'FIMA' 'PORO';

* Création du matériau associé.
* La propriété de matériau 'K' (diffusivité thermique) est évaluée
* par appel à la fonction 'UO2_THERMALCONDUCTIVITY_METEOR' de
* la librairie 'libinterface.so'
MatT1 = 'MATERIAU' ModT1 'K' Tmat;
```

FIGURE 9 : Déclaration d'un modèle et d'un matériau utilisant une propriété de matériau externe.

Nous commençons par déclarer un modèle thermique isotrope. Nous déclarons ensuite une table contenant trois indices :

- `'LOI'`, nom du symbole appelé dans la bibliothèque externe ;
- `'LIBRAIRIE'`, nom de la bibliothèque externe ;
- `'PARAMETRES'`, liste des paramètres dont dépend la fonction.

Les noms des ces indices ont été choisis en cohérence avec les conventions usuelles du code `Cast3M`.

La fonction externe utilisée ici dépend de la température 'T', du taux de combustion 'FIMA' et de la porosité 'PORO'. Conformément aux limitations de `Cast3M`, ces noms de variables ne peuvent avoir plus de quatre caractères.

Le matériau associé est alors défini. La conductivité thermique est initialisée par le tableau précédent.

Chargements externes associés aux paramètres de la loi Les objets modèle et matériau définis précédemment s'utilisent de manière classique. Il est cependant nécessaire de définir l'évolution au cours du pas de temps des paramètres externes à la résolution, dans notre cas les paramètres 'FIMA' et 'PORO'. Si l'on utilise les procédures `PASAPAS` ou `TRANSNON`, ceci est fait en définissant des chargements associés à ces paramètres contenant leur évolution sur le pas de temps et en passant ces chargements à l'indice `'CHARGEMENTS'` de la table utilisée en entrée de ces procédures.

Précautions relatives au "name mangling" des symboles L'indice 'LOI' doit faire l'objet d'une attention particulière : il correspond au nom du symbole de la bibliothèque et peut être différent du nom de la fonction

défini dans le fichier source ayant servi à générer la bibliothèque.

Le langage C est le seul pour lequel le nom de la fonction et le nom du symbole sont identiques.

En C++, des fonctions ayant des arguments différents peuvent porter le même nom. Or un symbole ne peut faire référence qu'à une fonction aux arguments déterminés. Le C++utilise alors une technique appelée "name mangling" pour résoudre les conflits et assurer une relation biunivoque entre le symbole et une fonction [Wikipédia 09]. La même technique est également utilisée pour distinguer des fonctions portant le même nom mais définies dans des espaces de noms différents. Le langage a cependant prévu cette difficulté et a introduit une syntaxe particulière pour éviter le "name mangling" : une fonction déclarée dans un bloc commençant par la directive `extern "C"` est soumise aux règles du langage C et se voit associée un symbole du même nom qu'elle.

Le "name mangling" des fonctions est également nécessaire en fortran, mais pour des raisons différentes : le fortran a une syntaxe insensible à la casse (les lettres majuscules et miniscules ne sont pas distinguées) alors que les symboles font cette distinction. Il n'y a jamais eu de consensus entre les vendeurs de compilateur fortran autour d'un choix unique de "name mangling". Ceci constitue un souci de portabilité majeur des codes mélangeant des sources fortran et des sources C ou C++ : les appels aux fonctions fortran dépendent du compilateur utilisé [29]. En pratique, le symbole associé à une fonction fortran peut être le nom de cette fonction écrit uniquement en majuscules ou en minuscules, éventuellement suivi du caractère '_' (*underscore*). Dans le cas du compilateur gnu fortran (g77 ou gfortran suivant la version utilisée), utilisé en général par les applications de la plateforme `Pléiades`, une fonction appelée `THERMALCONDUCTIVITY` dans le fichier source aura un symbole associé du nom `thermalconductivity_`. Ces difficultés nous ont amenés à *exclure* la possibilité de définir des lois externes en fortran.

Dans la suite, nous supposerons que les noms de fonctions et les noms de symboles associés sont identiques, c'est à dire que la fonction externe a été écrite en C ou en C++à l'intérieur d'un bloc `extern "C"`.

Précautions relatives aux déclarations de paramètres La déclaration des paramètres dans l'indice 'PARAMETRES' doit également faire l'objet d'une attention particulière. L'ordre de leur déclaration doit correspondre à l'ordre des arguments de la fonction. Le nombre des arguments est vérifié automatiquement (voir paragraphe suivant).

Notons enfin qu'il est impossible dans l'opérateur `PASAPAS` d'utiliser des propriétés de matériau externes ne dépendent d'aucun paramètre externe (propriétés constantes). En effet, la procédure teste s'il est nécessaire d'utiliser l'opérateur `VARI` en comptabilisant le nombre de dépendances à des paramètres des propriétés du matériau : si ce nombre est nul, l'opérateur `VARI` n'est pas appelé et l'appel à `PASAPAS` échoue avec un message d'erreur peu explicite. En fait, une tel cas n'a pas été prévu car la déclaration de propriété constante peut et doit se faire explicitement dans l'opérateur `'MATE'`.

ANNEXE E.3 UNE PROCÉDURE DE TRACÉ DES PROPRIÉTÉS MATÉRIAU DANS CAST3M

L'usage a montré qu'il était très pratique de pouvoir tracer les propriétés matériau dans `Cast3M`, pour vérification. Nous nous basons sur l'exemple du module d'YOUNG du SiC, détaillé au paragraphe 4.4 pour illustrer notre propos.

La procédure `GETEVOL`, reproduite en figure 15 a été écrite dans ce but.

Un exemple d'utilisation de cette procédure est donnée en figure 10 où est tracée la dépendance du module d'YOUNG du SiC en fonction de la température entre 300 et 1500 Kelvins pour une porosité de $0,1$ avec un échantillonnage de 100 valeurs. La tableau `val` contient ici les valeurs de toutes les variables autres que celle servant au tracé.

La courbe résultante est représentée en figure 11.

29. Voir [GNU 07] pour plus de détails.

```
'OPTION' 'DIME' 3;
'OPTION' 'ELEM' 'QUA4';

TLoi = 'TABLE';
TLoi.'LIBRAIRIE' = 'libCastemMaterialLaw.so';
TLoi.'MODELE' = 'SIC_YOUNGMODULUS_SNEAD';
TLoi.'VARIABLES' = 'MOTS' 'T' 'PORO';

val = 'TABLE';
val.'PORO' = 0.1;

ev = GETEVOL 300. 1500. 100 TLoi 'T' val;
'DESSIN' ev;
```

FIGURE 10 : Utilisation de la procédure GETEVOL.

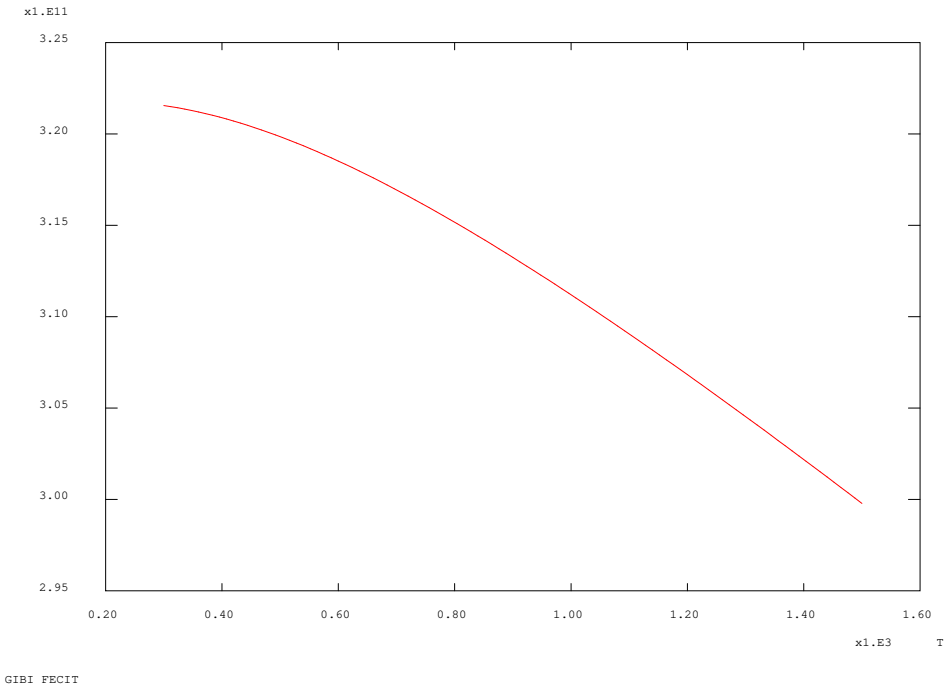

FIGURE 11 : Résultat de la procédure GETEVOL.

ANNEXE F UTILISATION DES PROPRIÉTÉS MATÉRIAU DANS MICROSOFT OFFICE EXCEL VIA L'INTERFACE CAST3M

Bien qu'il existe une interface `Microsoft Office Excel` dédiée, il est pratique de réutiliser dans ce logiciel des librairies matériau générées par `mfront` en utilisant l'interface `Cast3M` (cela évite la génération de deux librairies distinctes).

Nous détaillons dans cette annexe comment procéder. Nous nous basons sur l'exemple du module d'YOUNG du SiC, détaillé au paragraphe 4.4 pour illustrer notre propos.

ANNEXE F.1 DE MFRONT À MICROSOFT OFFICE EXCEL

En premier lieu, les librairies de propriétés matériau générées par `mfront` avec l'interface `Cast3M` peuvent être directement appelées depuis `Microsoft Office Excel`.

```
Declare Function SIC_YOUNGMODULUS_SNEAD Lib "libCastemMaterialLaw"(temp As Double) As
Double

Public Function LoiMat(ParamArray VarTableau())
    Dim v(2) As Double
    v(0) = VarTableau(0)
    v(1) = VarTableau(1)
    LoiMat = SIC_YOUNGMODULUS_SNEAD(v(0))
End Function
```

FIGURE 12 : Appel d'une loi externe depuis `Microsoft Office Excel`.

La figure 12 représente une macro `Visual Basic` permettant cet appel. Elle déclare une fonction `LoiMat` qui peut être directement utilisée dans un classeur sous la forme :

```
=LoiMat($A$1,A3)
```

qui calculera dans cet exemple le module d'YOUNG du SiC en fonction de la valeur de la température, lue dans la cellule `$A$1` et de la valeur de la porosité, lue dans la cellule `A3`.

Le passage par la macro `Visual Basic` décrite en figure 12 est assez lourd et nous pensons être en mesure de nous en passer prochainement, rendant encore plus simple l'utilisation des librairies matériau issue de `mfront` dans `Microsoft Office Excel`.

La figure 13 montre la dépendance du module d'YOUNG du SiC en fonction de la porosité en se basant sur l'exemple traité au paragraphe 4.4.

ANNEXE F.2 DE MICROSOFT OFFICE EXCEL À MFRONT

De plus il est apparu utile (et relativement facile à mettre en œuvre) de pouvoir directement généré des fichiers `mfront` depuis des formules `Microsoft Office Excel`. Une macro `Visual Basic` a été écrite par É. GOHIER dans ce but. La figure 14 montre un exemple de son utilisation.

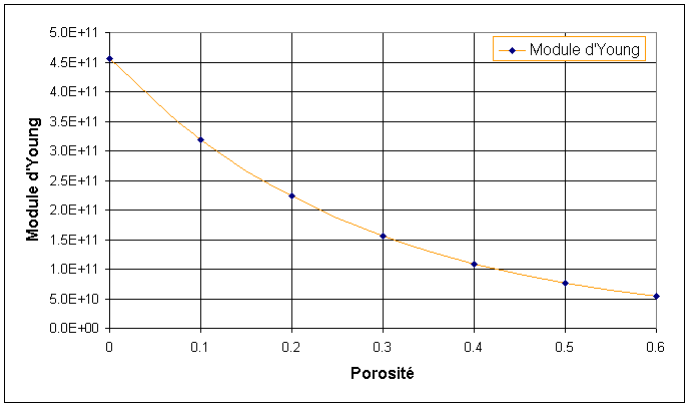

FIGURE 13 : Graphique `Microsoft Office Excel` réalisé à partir d'une bibliothèque dynamique généré par `mfront`.

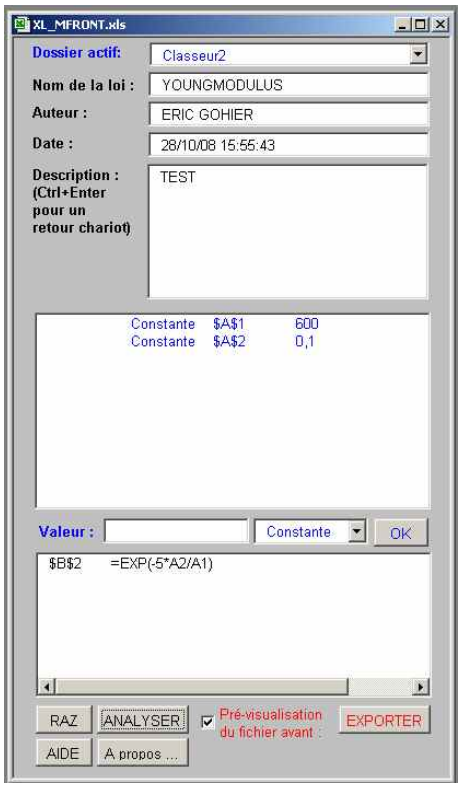

FIGURE 14 : Export d'une formule `Microsoft Office Excel` en `mfront`.

```
'DEBPROC'  GETEVOL        X0*'FLOTTANT'
                             X1*'FLOTTANT'
                             nx*'ENTIER'
                          TLoi*'TABLE'
                          vvar*'MOT'
                       values*'TABLE';
'MESSAGE'  '----------------------'  ;
'MESSAGE'  ' Début : '  X0;
'MESSAGE'  ' Fin : '  X1;
'MESSAGE'  ' Nombre de valeurs : '  nx;
'MESSAGE'  ' Loi : librairie : '  TLoi.'LIBRAIRIE';
'MESSAGE'  ' Loi : Modèle : '  TLoi.'MODELE';
vdime=('VALEUR'  'DIME');
velem=('VALEUR'  'ELEM');
'OPTION'  'DIME'  1;
'OPTION'  'ELEM'  SEG2;
lx  =  'DROIT'  ('POIN'  X0)  ('POIN'  X1)  nx;
modv  =  'MODELISER'  lx  'THERMIQUE'  'ISOTROPE';
xx  =  'COORDONNEE'  1  lx  ;
vpo  =  'CHANGER'  xx  'COMP'  vvar;
vel  =  'CHANGER'  'CHAM'  vpo  modv  ;
nb    =  'DIME'  (TLoi.'VARIABLES');
'REPETER'  bcl  nb;
     vvari  =  'EXTRAIRE'  (Tloi.'VARIABLES')  &bcl;
     'SI'('NEG'  vvari  vvar);
          vel  =  vel  'ET'  ('MANUEL'  'CHML'  modv  vvari  (values.vvari));
     'FINSI';
'FIN'  bcl;
matE  =  'MANUEL'  'CHML'  modv  'Y'  TLoi;
Kel  =  'VARI'  'NUAG'  modv  matE  vel;
Kpo  =  'CHANGER'  'CHPO'  Kel  modv  ;
evabs  =  'EVOL'  'CHPO'  vpo  lx  vvar;
evordo  =  'EVOL'  'CHPO'  Kpo  lx  'Y';
labs  =  'EXTRAIRE'  evabs  'ORDO'  ;
lordo  =  'EXTRAIRE'  evordo  'ORDO'  ;
cou  =  'EVOL'  'MANUEL'  labs  vvar  lordo  ;
cou  =  cou    'COULEUR'  'JAUNE';
'OPTION'  'DIME'  vdime;
'OPTION'  'ELEM'  velem;
'FINPROC'  cou  ;
```

FIGURE 15 : Procédure utilisée pour tracer les valeurs d'une fonction en fonction d'une de ces variables (les autres étant fixées).

INDEX DES VARIABLES D'ENVIRONNEMENT

INDEX DES DIRECTIVES