Écriture de lois de comportement avec `mfront` : tutoriel

T. Helfer, J.M. Proix[(a)]

Décembre 2014

[(a)] Électricité de France, R&D - Département Analyses Mécaniques et Acoustique

RÉSUMÉ

Ce document est un tutoriel sur l'écriture des lois de comportement mécanique avec `mfront`.

Deux lois sont traitées :

- une loi élasto-(visco)-plastique typique des métaux ;
- une loi élasto-viscoplastique isotrope compressible typique du combustible nucléaire.

La rédaction de ce document fait partie de la démarche de mise en open-source de `mfront`.

SOMMAIRE

1 INTRODUCTION

L'objet de ce document est de compléter la documentation de référence `mfront`. Celle-ci se compose aujourd'hui de :

- une présentation générale et un document de référence sur l'écriture de propriétés matériau et de modèles physico-chimiques [Helfer 13a] ;
- un document de référence sur l'écriture de lois de comportement mécanique [Helfer 13b] ;
- différents documents, distribués avec `mfront`, qui décrivent les interfaces à différents codes aux éléments finis (`Cast3M`, `Code_Aster` et `ZeBuLoN` notamment) ;
- différents supports de formation, eux aussi distribués avec `mfront`.

L'ensemble des ces documents est disponible sur le site officiel de `mfront` :

http://tfel.sourceforge.net

Par ailleurs, les utilisateurs de `Code_Aster` disposent déjà de documentation propre décrivant l'utilisation de `mfront` [EDF 14d, EDF 14a, EDF 14b, EDF 14c].

L'objet du présent document est de permettre aux nouveaux utilisateurs de découvrir pas à pas comment une loi de comportement mécanique peut être implantée avec `mfront`. Pour les applications, nous utiliserons dans un premier temps des simulations sur point matériel avec `mtest` (logiciel livré avec `mfront`) puis, dans un second temps, des simulations par éléments finis, en utilisant soit `Code_Aster`, soit `Cast3M`.

1.1 COMMENT ÉCRIRE UNE LOI DE COMPORTEMENT : UN EXEMPLE TRÈS SIMPLE

Dans la suite, les tenseurs d'ordre 2 symétriques seront notés $\underline{a}$. Les opérateurs linéaires agissant sur les tenseurs d'ordre 2 symétriques seront notés $\underline{\underline{\mathbf{D}}}$.

1.1.1 La loi de comportement de NORTON

En guise de préliminaire, prenons par exemple la loi de NORTON. Elle est définie par :

$$\begin{cases} \underline{\epsilon}^{\mathrm{to}} = \underline{\epsilon}^{\mathrm{el}} + \underline{\epsilon}^{\mathrm{vis}} \\ \underline{\sigma} = \underline{\underline{\mathbf{D}}} : \underline{\epsilon}^{\mathrm{el}} \\ \dot{\underline{\epsilon}}^{\mathrm{vis}} = \dot{p}\,\underline{n} \\ \dot{p} = A\,\sigma_{\mathrm{eq}}^{m} \end{cases}$$

où :
- $\underline{\epsilon}^{\mathrm{to}}$, $\underline{\epsilon}^{\mathrm{el}}$, $\underline{\epsilon}^{\mathrm{vis}}$ sont des variables tensorielles symétriques représentent respectivement les déformations totale, élastique et visqueuse ;
- $\underline{n} = \frac{3}{2}\,\frac{\underline{s}}{\sigma_{\mathrm{eq}}}$ est la direction d'écoulement ;
- $\underline{s}$ est le déviateur des contraintes ;
- σ_{eq} est la norme de VON MISES.

L'opérateur d'élasticité $\underline{\underline{\mathbf{D}}}$ est calculé à partir du module d'YOUNG E et du coefficient de POISSON ν.

1.1.2 Discrétisation implicite de la loi de NORTON

Pour intégrer cette loi dans un calcul de structure, il faut procéder à une discrétisation en temps, ce qui revient à définir une suite d'instants de calcul $\{t_i\}_{1 \leq i \leq I}$.

Pour utiliser un algorithme implicite, il suffit d'écrire toutes les quantités à l'instant t_i et de remplacer les dérivées en temps par leurs incréments sur l'intervalle $\Delta t = t_i - t_{i-1}$:

$$\begin{cases} \Delta \underline{\epsilon}^{\mathrm{el}} - \Delta \underline{\epsilon}^{\mathrm{to}} + \Delta p \, \underline{n} = 0 \\ \Delta p - \Delta t \, A \, \sigma_{\mathrm{eq}}^{m} = 0 \end{cases}$$

avec : — $\underline{\sigma} = \underline{\underline{\mathbf{D}}} : \underline{\epsilon}^{\mathrm{el}}$;
— $\underline{n} = \dfrac{3}{2} \dfrac{\underline{s}\,(t_i)}{\sigma_{\mathrm{eq}}\,(t_i)}$.

On obtient ainsi un système de 7 équations : 6 équations relatives à la décomposition additive du tenseur des déformations (en 3D), et une équation relative à l'écoulement visco-plastique. Les 7 inconnues sont les 6 composantes de $\Delta \underline{\epsilon}^{\mathrm{el}}$ et Δp.

La résolution implicite de ce système est effectuée par une méthode de NEWTON.

1.1.3 Première implantation

Voici un exemple très simple d'intégration implicite de ce modèle avec `mfront` :

```
1  @Parser Implicit;
2  @Behaviour Norton;
3  @Algorithm NewtonRaphson_NumericalJacobian ;
4
5  @RequireStiffnessTensor;
6
7  @MaterialProperty real A;
8  @MaterialProperty real m;
9
10 @StateVariable real p ;
11
12 @ComputeStress{
13   sig = D*eel ;
14 }
15
16 @Integrator{
17   real seq = sigmaeq(sig) ;
18   Stensor n = Stensor(0.) ;
19   if(seq > 1.e-12){
20     n = 1.5*deviator(sig)/seq ;
21   }
22   feel += dp*n-deto ;
23   fp -= dt*A*pow(seq,m) ;
24 } // end of @Integrator
25
26 @TangentOperator{
27   Stensor4 Je ;
28   getPartialJacobianInvert(Je) ;
29   Dt = D*Je ;
30 }
```

Description ligne par ligne Un fichier `mfront` commence généralement par une partie déclarative décrivant l'algorithme utilisé pour la résolution, le nom du comportement (ici NORTON),puis la liste des propriétés matériau utilisées. On fournit ensuite le nom des variables internes, et la description des équations du système à résoudre.

- la ligne 1 précise que nous allons utiliser une méthode d'intégration implicite ;
- la ligne 2 donne le nom de la loi ;
- la ligne 3 précise que l'on utilise un algorithme de NEWTON avec jacobienne numérique ;
- la ligne 5 demande le calcul de la matrice d'élasticité ;
- les lignes 7 et 8 définissent les propriétés matériau de la loi ;
- la ligne 10 déclare la variable interne p. Signalons que la variable interne `eel` (tenseur déformation élastique $\underline{\epsilon}^{\mathrm{el}}$) est prédéfinie par `mfront` ;
- les lignes 16 à 24 définissent les équations à résoudre ; la convention de nom est la suivante : pour chaque variale interne `x`, l'incrément est noté `dx`, et l'équation correspondante `fx` ;
- la ligne 17 demande le calcul de la norme de Von Mises ;
- les lignes 18 à 21 correspondent au calcul de la direction d'écoulement ;
- la ligne 22 définit l'équation $\Delta \underline{\epsilon}^{\mathrm{el}} - \Delta \underline{\epsilon}^{\mathrm{to}} + \Delta p \, \underline{n} = 0$. En effet, `feel` est initialisé à `deel` et l'opérateur `+=` cumule les valeurs (comme en langage `c`) ;

— la ligne 23 définit l'équation $\Delta p - \Delta t\, A\, \sigma_{\text{eq}}^{m} = 0$. Comme précédemment, `fp` est initialisé à `dp`, et `fp -= xx` est équivalent à `fp = fp - xx`.
— les lignes 26 à 29 permettent de calculer automatiquement l'opérateur tangent à partir de l'inverse de la matrice jacobienne, comme explicité ci-dessous.

On constate que l'écriture de la loi se limite quasiment à la description des équations. De plus on bénéficie d'une écriture compacte, utilisant des variables tensorielles.

Différentes méthodes d'intégration sont diponibles dans `mfront`([Helfer 13b]). L'algorithme d'intégration utilisé ici (`NewtonRaphson_NumericalJacobian`) permet une écriture rapide, et est donc tout à fait adapté au test de modèles.

L'implantation fournit un code beaucoup plus rapide que celle d'un algorithme explicite, mais peut toutefois être optimisée en termes de performances. Pour cela, il suffit d'ajouter les termes de la matrice jacobienne (dérivées des équations par rapport aux inconnues).

De plus, la matrice tangente en sortie de l'intégration est calculée automatiquement à partir de la matrice jacobienne, ce qui permet d'économiser un temps de développement important et conduit à une matrice tangente cohérente de très bonne qualité [Helfer 13b]. Tout ceci conduit, en très peu de temps, à une intégration robuste, et une convergence très bonne. On voit qu'il est possible de profiter de cette simplicité d'écriture pour effectuer des variantes, des tests de modèles, etc.

`mfront` gère la compilation de la loi, il suffit de taper dans un terminal :

```
$ mfront --obuild --interface=aster norton.mfront
```

ou

```
$ mfront --obuild --interface=castem norton.mfront
```

suivant le code de résolution que l'on souhaite utiliser : l'interface `aster` correspond au `Code-Aster`, tandis que l'interface `castem` correspond au code `Cast3M`.

Ceci génère deux répertoires : `src` et `include`.

Selon l'interface, le répertoire src contient en particulier une bibliothèque dynamique :

```
src/libAsterBehaviour.so
```

pour une exécution avec `Code_Aster`, ou bien :

```
src/libUmatBehaviour.so
```

pour une exécution avec `Cast3M`. Le nom `Umat` est historique et correspond au nom de la routine que l'utilisateur `Cast3M` devait surcharger avant le support des appels dynamiques.

Dans le cas du code `Cast3M`, un répertoire nommé `castem` est créé dans la plupart des cas. Ce répertoire contient une mise en données type de la loi de comportement pour les différentes hypothèses de modélisation supportées.

1.1.4 Premier test

L'outil `mtest` permet d'effectuer très facilement des simulations sur point matériel, afin de calculer la réponse de la loi de comportement à des sollicitations en contraintes ou en déformations.

`mtest`, couplé à un logiciel d'optimisation comme `Adao`, permet de plus d'effectuer le recalage des propriétés matériau. Le fichier de données de `mtest` (nommé ici `norton.mtest`) se présente de la façon suivante :

```
1 @Behaviour<aster> './src/libAsterBehaviour.so' 'asternorton' ;
2 @MaterialProperty<constant> 'young' 178600.0E6 ;
3 @MaterialProperty<constant> 'nu' 0.3 ;
4 @MaterialProperty<constant> 'A' 8.e-67 ;
5 @MaterialProperty<constant> 'n' 8.2 ;
6 @ExternalStateVariable 'Temperature' 293.15 ;
7 @ImposedStress 'SXX' { 0. : 0., 30. : 40.e6};
8 @ImposedStress 'SXY' { 0. : 0., 30. : 40.e6};
9 @Times {0.,30. in 100};
```

— la ligne 1 définit la bibliothèque à utiliser et le nom du comportement ;
— les lignes 2 à 5 définissent les valeurs des propriétés matériau de la loi ;
— la ligne 6 déclare la valeur de la température (inutile dans le cas présent) ;
— les lignes 7 à 8 spécifient le chargement, sous la forme de composantes de contraintes ou de déformations en fonction du temps ;
— la ligne 9 définit la discrétisation en temps, donc les instants calculés.

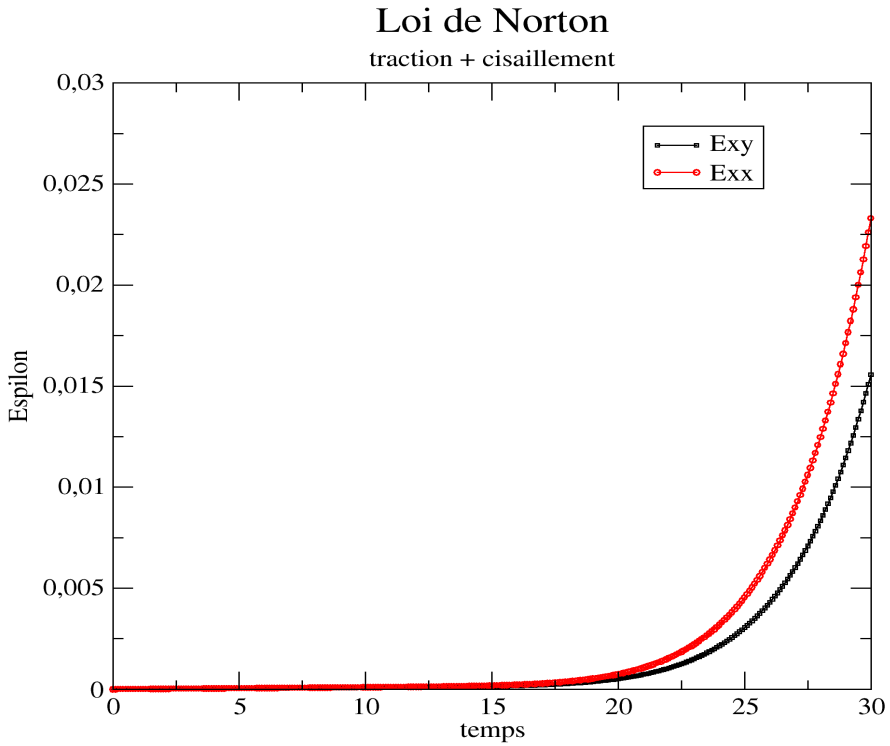


FIGURE 1 : Réponse d'une loi de NORTON à un essai de fluage en traction-cisaillement

Il suffit alors de lancer le calcul par :

```
mtest norton.mtest
```

Ceci produit un fichier résultat `norton.res` contenant les composantes des tenseurs de déformation, de contrainte, et les variables internes en fonction du temps. La réponse en déformation est représentée en figure 1.

1.2 PLAN DE CE TUTORIEL

Ce tutoriel présente en détail comment développer des lois de comportement et ce que l'on peut attendre de `mfront`.

La section 2 a pour but de situer l'intégration de la loi de comportement dans le processus de calcul d'un code par éléments finis.

La section 3 décrit pas à pas l'implantation en MFront d'une loi élasto-plastique de Chaboche, puis de sa variante visco-plastique, relativement simple, et de quelques extensions.

Enfin, la section 4 fournit davantage de précisions sur l'intégration, sur la gestion des propriétés matériau, et différentes hypothèses de calcul, en décrivant l'implantation d'une loi viscoplastique compressible typique du combustible nucléaire.

2 QUELQUES RAPPELS ET DÉFINITIONS SUR L'INTÉGRATION DES LOIS DE COMPORTEMENT : OÙ SE PLACE MFRONT ?

2.1 ALGORITHME DE RÉSOLUTION D'UN PROBLÈME NON LINÉAIRE QUASI-STATIQUE

Quand le comportement des matériaux composant une structure n'est pas linéaire, la résolution de l'équilibre global de la structure à chaque instant est un problème non linéaire.

La formulation du problème quasi-statique discrétisé consiste à exprimer l'équilibre de la structure pour une suite d'instants de calcul $\{t_i\}_{1\leq i\leq I}$ qui paramètrent le chargement :

$$\mathrm{L}^{\text{int}}(\mathrm{u}_i, t_i) = \mathrm{L}_i^{\text{ext}}(t_i)$$

avec :

- t_i représente la variable d'instant ;
- u_i est le champ de déplacement à l'instant t_i qui est l'inconnu du problème ;
- $\mathrm{L}^{\text{ext}}(t)$ est le chargement mécanique extérieur auquel est soumis la structure ;
- L^{int} représente les forces internes, reliées au champ de contraintes $\underline{\sigma}$.

Ce problème non linéaire est, à chaque instant d'intérêt t_i, généralement résolu par une méthode de type NEWTON.

En pratique, on distingue deux phases dans l'algorithme :

- une phase de prédiction, optionnelle qui estime au mieux la solution avec les données de début de pas ;
- une phase de correction itérative.

Pour la phase de correction, on obtient un système à résoudre, qui s'écrit alors, pour chaque itération n [1] :

$$\mathrm{K}_i^n.\delta\mathrm{u}_i^{n+1} = \mathrm{L}_i^{\text{méca}} - \mathrm{L}_i^{\text{int},n}$$

Le vecteur des forces internes $\mathrm{L}_i^{\text{int},n}$ est calculé à partir des contraintes $\underline{\sigma}_i^n$, celles-ci étant calculées à partir des déplacements u_i^n par l'intermédiaire de la relation de comportement du matériau.

En fait, dans le cas des comportements incrémentaux, $\underline{\sigma}_i^n$ est calculé à partir de la connaissance des contraintes $\underline{\sigma}_{i-1}$ et des éventuelles variables internes α_{i-1} à l'instant précédent et de l'incrément de déformation $\underline{\epsilon}^{\text{to}}(\Delta\,\mathrm{u}_i^n)$ induit par l'incrément de déplacement depuis le début du processus itératif.

La figure 2 donne une vision très globale de cet algorithme.

La figure 3 précise la place de la loi de comportement dans la recherche de l'équilibre.

2.2 BUT ET RÉSULTAT DE L'INTÉGRATION D'UNE LOI DE COMPORTEMENT

Au pas de temps i et à l'itération de Newton n, à partir des contraintes et variables internes à l'équilibre précédent $(\underline{\sigma}_{i-1}, \alpha_{i-1})$ et de l'incrément de déformation $\underline{\epsilon}^{\text{to}}(\Delta\,\mathrm{u}_i^n)$ (et éventuellement avec la température, hydratation..), il faut calculer, en chaque point de GAUSS de chaque élément fini :

- les contraintes et variables internes :

$$\left(\underline{\sigma}_{i-1}, \alpha_{i-1}, \underline{\epsilon}^{\text{to}}\left(\Delta\,\mathrm{u}_i^n\right)\right) \rightarrow \underline{\sigma}_i^n, \alpha_i^n$$

- l'opérateur tangent :

$$\left(\underline{\sigma}_{i-1}, \alpha_{i-1}, \underline{\epsilon}^{\text{to}}(\Delta\,\mathrm{u}_i^n)\right) \rightarrow \left(\frac{\partial\underline{\sigma}}{\partial\underline{\epsilon}^{\text{to}}}\right)_i^n$$

1. Pour simplifier, nous omettons les équations de LAGRANGE relatives aux conditions aux limites.

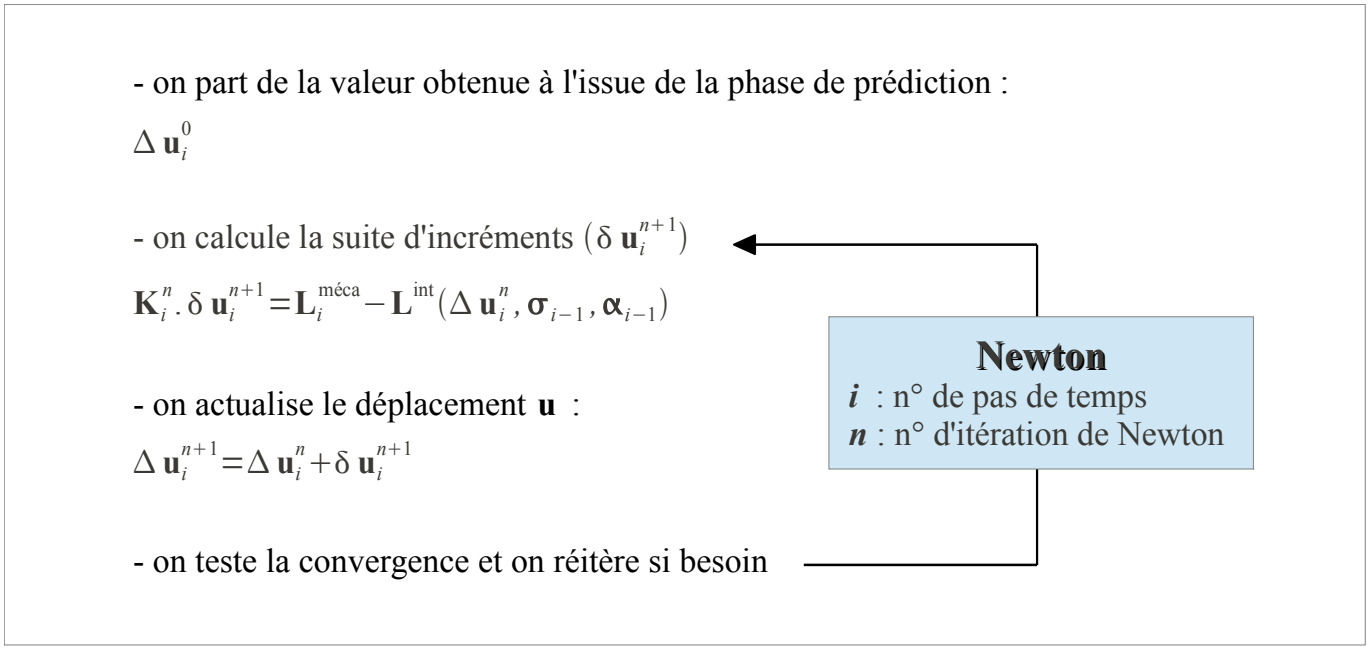


FIGURE 2 : Schéma général de résolution d'un problème non linéaire en mécanique des structures par la méthode de Newton-Raphson.

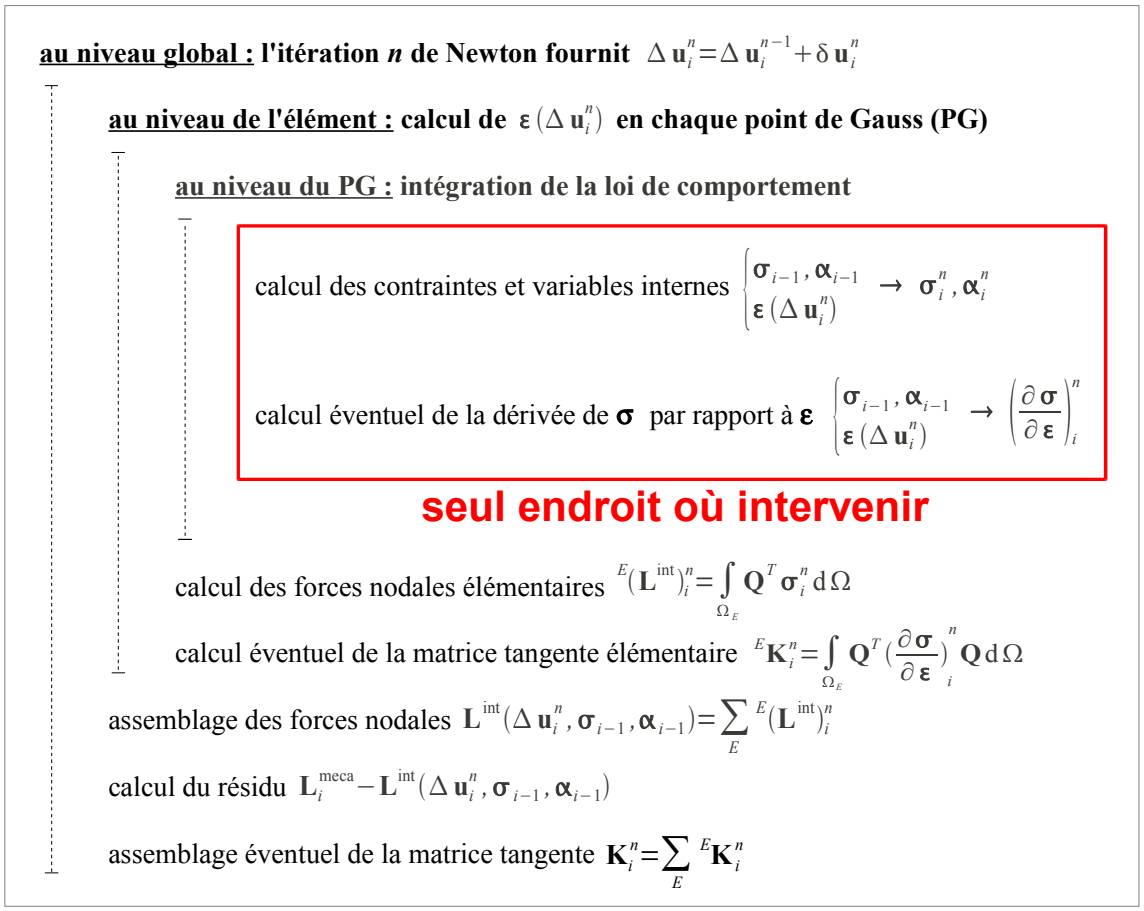


FIGURE 3 : Place de la loi de comportement dans le schéma itératif de recherche de l'équilibre.

À l'itération 0 du pas de temps i (initialisation de l'algorithme de Newton), on choisit comme matrice tangente de prédiction la matrice tangente à l'équilibre précédent $(i-1)$, soit :

$$\mathrm{K}_i^0 = \mathrm{K}_{i-1}$$

À partir des contraintes et variables internes à l'équilibre précédent $(\underline{\sigma}_{i-1}, \alpha_{i-1})$, il faut donc calculer en chaque point de GAUSS de chaque élément fini :

— l'opérateur tangent en prédiction : $\underline{\sigma}_{i-1}, \alpha_{i-1} \rightarrow \left(\frac{\partial \underline{\sigma}}{\partial \underline{\underline{\epsilon}}^{to}}\right)_i^0$

2.3 DIFFÉRENTS TYPES DE MATRICES TANGENTES

2.3.1 Matrice tangente dite cohérente ou consistante

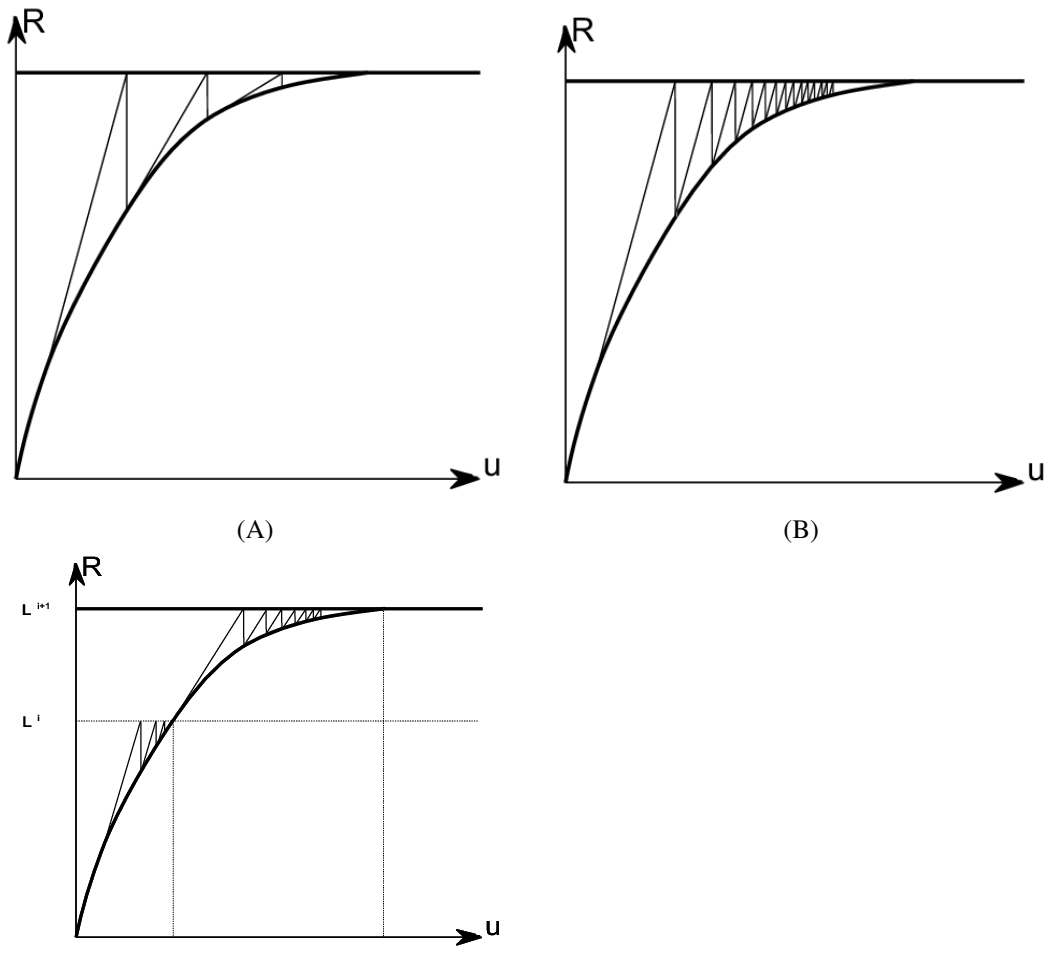


FIGURE 4 : Matrices tangentes (A) cohérente, (B) élastique, (C) prédiction

Réactualisée à chaque itération, elle assure la meilleure vitesse de convergence (quadratique) à l'algorithme global de Newton (figure 4-A). Mais pour de grands incréments de chargement, ou des décharges, la matrice tangente cohérente peut conduire à des divergences de l'algorithme.

2.3.2 Autres matrices « tangentes »

Les approximations dans le calcul de la matrice « tangente » dégradent la vitesse de convergence par rapport à la matrice tangente cohérente mais la solution obtenue reste juste tant que le résidu est calculé de manière exacte. Il existe plusieurs variantes possibles (dites méthodes de quasi-Newton). Citons les plus simples :

- la matrice élastique :
  - calculée une seule fois (économique) à partir des paramètres d'élasticité ;
  - recommandée en cas de décharge ;
  - convergence lente mais assurée (figure 4-B).
- matrice tangente réactualisée tous les i_0 incréments de charge (figure 4-C) ou toutes les n_0 itérations de NEWTON
  - coût moindre
  - direction moins bien évaluée ;
  - diverge parfois dans les zones de forte non linéarité ;

Pour tous ces algorithmes, il est possible, en cas de non convergence, de procéder à un re-découpage local du pas de temps. Pour cela, on adopte une interpolation linéaire du déplacement et des variables de commande (par exemple la température) au cours de l'intervalle de temps, ce qui conduit, pour tout incrément de temps à :

$$\forall \tau \in [t_{\text{i-1}}, t_i]\, T(\tau) = T_{\text{i-1}} + \frac{\tau - t_{\text{i-1}}}{\Delta t}[T - T_{\text{i-1}}] \qquad \varepsilon_{\text{ij}}^k(\tau) = \varepsilon_{\text{ij}}(\text{u}_{\text{i-1}}) + \frac{\tau - t_{\text{i-1}}}{\Delta\, t}[\varepsilon_{\text{ij}}^k(\text{u}_i^n) - \varepsilon_{\text{ij}}(\text{u}_{\text{i-1}})]$$

2.4 MÉTHODES D'INTÉGRATION MISES EN ŒUVRE DANS MFRONT

`mfront` permet de réaliser ces calculs au niveau du point d'intégration. Plusieurs algorithmes de résolution sont offerts à l'utilisateur :

- des algorithmes adaptés aux lois simples (viscoplastiques et plastiques isotropes, etc.) par intégration implicite avec réduction à une équation scalaire, ce qui est équivalent à ce qui est fait dans `Code_Aster` pour les lois élasto-plastiques de VON MISES, les lois de LEMAITRE, de CHABOCHE ;
- l'intégration explicite par des algorithmes de type RUNGE-KUTTA (d'ordres variés) avec contrôle d'erreur ;
- l'intégration implicite par une θ-méthode et des algorithmes de NEWTON, (avec des variantes de type BROYDEN) ;
- l'intégration directe (si la résolution est analytique) ;

2.5 NOMENCLATURE ET MOTS-CLÉS RÉSERVÉS

`mfront` définit par défaut un certain nombre de mots clés qui représentent les quantités utiles aux différents algorithmes de résolution. Pour éviter les conflits, ces noms ne peuvent pas être utilisés pour désigner d'autres variables.

- `Dt` représente la matrice tangente cohérente qu'il faut calculer ;
- `sig` représente la contrainte qu'il faut calculer ;
- `eto` représente la déformation totale en début de pas ;
- `deto` représente l'incrément de déformation totale (constante sur le pas) ;
- `T` représente la valeur de la température en début de pas ;
- `dT` représente l'incrément de changement de température (constante sur le pas) ;
- pour toute variable interne `Y`, `Y` représente sa valeur en début de pas ;
- pour toute variable interne `Y`, `dY` représente l'incrément de cette variable sur le pas, incrément qu'il faut calculer ;
- pour toute variable externe `V`, `V` représente sa valeur en début de pas ;
- pour toute variable externe `V`, `dV` représente son incrément de variation sur le pas de temps (constante sur le pas).

3 DÉVELOPPEMENT PAS À PAS D'UNE LOI ÉLASTO-(VISCO)-PLASTIQUE

Nous allons suivre pas à pas le développement en `mfront` d'une loi élasto-plastique, puis visco-plastique (modèle de J.L. CHABOCHE).

3.1 LOI ÉLASTO-PLASTIQUE DE CHABOCHE

Il s'agit d'une loi de comportement élasto-plastique à écrouissage isotrope et cinématique non linéaires. Les équations du modèle sont résumées brièvement :

— relation contraintes déformations élastiques :

$$\underline{\sigma} = \underline{\underline{\mathbf{D}}} : \left(\underline{\epsilon}^{\mathrm{to}} - \underline{\epsilon}^{\mathrm{p}}\right)$$

— Critère de plasticité :

$$F\left(\underline{\sigma}, \underline{X}\right) = \left(\sigma - \underline{X}\right)_{\mathrm{eq}} - R(p) \leq 0$$

avec, pour tout tenseur $\underline{A}$:

$$A_{\mathrm{eq}} = \sqrt{\frac{3}{2}\underline{\tilde{A}} : \underline{\tilde{A}}}$$

où $\underline{\tilde{A}}$ est le déviateur de $\underline{A}$.

— l'évolution de la déformation plastique est gouvernée par une loi d'écoulement normale au critère de plasticité :

$$\underline{\dot{\epsilon}}^{\mathrm{p}} = \dot{p}\,\underline{n} \quad \text{avec} \quad \underline{n} = \frac{3}{2}\frac{\underline{\tilde{\sigma}} - \underline{X}}{\left(\sigma - \underline{X}\right)_{\mathrm{eq}}}$$

— $\underline{X}$ représente l'écrouissage cinématique non linéaire. Il peut résulter d'une combinaison de plusieurs écrouissages cinématiques $\underline{X} = \underline{X}_1 + \underline{X}_2 + ...$;

— L'évolution de chaque variable d'écrouissage $\underline{X}_i$ est donnée par :

$$\underline{X}_i = \frac{2}{3} C_i \underline{\alpha}_i$$

$$\underline{\dot{\alpha}}_i = \underline{\dot{\epsilon}}^{\mathrm{p}} - \gamma_i\,\underline{\alpha}_i\,\dot{p}$$

— La fonction d'écrouissage isotrope $R\left(p\right)$ est définie par :

$$R\left(p\right) = R^{\infty} + \left(R^{0} - R^{\infty}\right)\exp\left(-b\,p\right)$$

Les propriétés matériau de ce modèle sont donc E, ν, R^0, R^∞, b, C_1, C_2, γ_1, γ_2.

La discrétisation implicite (par une θ-méthode) de ces équations conduit à résoudre un système d'équations où les inconnues sont :

— le tenseur $\Delta\,\underline{\epsilon}^{\mathrm{el}}$;

— le scalaire, $\Delta\,p$;

— les tenseurs $\Delta\,\underline{\alpha}_i$ (pour $i = 1, n$ variables cinématiques) ;

et où chaque équation d'évolution temporelle de la forme $\dot{y} = f\left(y, z, ...\right)$ est remplacée par :

$$\Delta y - \Delta\,t\,f\left(y + \theta\Delta y, z + \theta\Delta z, , ...\right) = 0$$

Dans le cas de la plasticité, on n'écrit pas d'équation d'évolution de la variable p, mais directement le respect du critère de plasticité.

Deux cas se présentent. L'évolution peut être élastique. Il est possible de tester cela en calculant un tenseur de test $\underline{\sigma}^{tr}$ tel que :

$$\underline{\sigma}^{tr} = \underline{D} : \left(\underline{\epsilon}^{\mathrm{el}}\big|_{t} + \Delta\,\underline{\epsilon}^{\mathrm{to}}\right)$$

Si le critère plastique évalué avec cette contrainte de test est négatif, c'est à dire si :

$$F^{el}\left(\underline{\sigma},\underline{X}\right)=\left(\underline{\sigma}^{tr}-\underline{X}|_t\right)_{\text{eq}}-R\left(p|_t\right)<0$$

alors la solution cherchée est triviale et donnée par :

$$\Delta\,\underline{\epsilon}^{\text{p}}=0\quad\Delta\,p=0\quad\Delta\,\underline{\alpha}_i=\underline{0}$$

Sinon, il faut résoudre le système suivant :

$$F(\sigma,X)=0\quad\Leftrightarrow\quad\begin{cases}(\underline{\sigma}|_{t+\Delta\,t}-\underline{X}|_{t+\Delta\,t})_{\text{eq}}-R(p(t+\Delta\,t)) & =0\\ \Delta\,\underline{\alpha}_i-\Delta\,\underline{\epsilon}^{\text{p}}+\gamma_i(\underline{\alpha}_i+\theta\Delta\underline{\alpha}_i)\Delta\,p & =\underline{0}\\ \Delta\,\underline{\epsilon}^{\text{el}}-\Delta\,\underline{\epsilon}^{\text{to}}+\Delta\,\underline{\epsilon}^{\text{p}} & =\underline{0}\end{cases}$$

où $\Delta\,\underline{\epsilon}^{\text{p}}=\Delta\,p\ \underline{n}|_{t+\Delta\,t}$.

Dans le cas de 2 variables cinématiques, (le système est alors de taille $6+1+2\times6=19$), la « traduction » de ces équations en `mfront` est décrite ici pas à pas :

1. choix de l'algorithme avec matrice jacobienne numérique dans un premier temps.

```
@Parser Implicit;
@Behaviour Chaboche;
@Algorithm NewtonRaphson_NumericalJacobian;
@Theta 1.;
```

   En plasticité, on choisit en général $\theta=1$ pour que le critère soit vérifié en fin de pas de temps (à l'instant t_i)

2. définition des propriétés matériau (modifiables depuis le jeu de commandes `mtest` ou `Code_Aster`) :

```
@MaterialProperty real young;
young.setGlossaryName("YoungModulus");
@MaterialProperty real nu;
young.setGlossaryName("PoissonRatio");
@MaterialProperty real R_inf;
@MaterialProperty real R_0;
@MaterialProperty real b;
@MaterialProperty real C[2];
@MaterialProperty real g[2];
```

   Les appels à `setGlossaryName` sont importants pour `Cast3M` car ils permettent à `mfront` de faire le lien entre les propriétés matériau nécessaires à la loi et ceux définis par `Cast3M` pour ses propres besoins.

3. définition des variables internes (l'écriture ci-dessus utilise des tableaux, ce qui permet de gérer facilement un nombre quelconque de variables cinématiques. Ici on en choisit 2) :

```
@StateVariable real     p;
@StateVariable Stensor a[2];
```

   De plus, il faut compter en tant que variable interne (donc inconnue du problème à résoudre) le tenseur des déformations élastiques (ou plutôt son incrément `deel`). L'utilisation de `deel` plutôt que `dsig` (utilisée habituellement dans `Code-Aster` permet d'obtenir un système bien conditionné et permet de traiter sans précaution particulière le cas de propriétés élastiques variables dans le temps.

4. définition et initialisation des variables locales à l'algorithme (l'initialisation est faite une seule fois avant l'appel de l'algorithme implicite ce qui permet de gagner du temps de calcul) :

```
@LocalVariable stress lambda;
@LocalVariable stress mu;
@LocalVariable stress Fel;

/* Initialize Lame coefficients */
@InitLocalVariables{
  lambda = computeLambda(young,nu);
  mu     = computeMu(young,nu);
  // prediction elastique
  StresssStensor sigel(lambda*trace(eel+deto)*Stensor::Id()+2*mu*(eel+deto));
  for(unsigned short i=0;i!=2;++i){
    sigel -= C[i]*a[i]/1.5 ;
  }
  const stress seqel = sigmaeq(sigel);
  const stress Rpel  = R_inf + (R_0-R_inf)*exp(-b*p) ;
  Fel   = seqel - Rpel ;
}
```

On calcule dans cette section le critère F^{el}, qui permet d'éviter de lancer l'algorithme de NEWTON si la solution reste élastique.

5. calcul des contraintes. À chaque itération de l'algorithme implicite et après convergence, on calcule :

```
39 @ComputeStress{
40   sig = lambda*trace(eel)*Stensor::Id()+2*mu*eel;
41 }
```

6. Calcul des différents termes du système d'équations :

```
41 @Integrator{
42   if(Fel > 0){
43     // solution plastique
44     // Les variables suivies de _ sont en t+theta*dt
45     const real p_           = p +theta*dp ;
46     const real Rp_          = R_inf + (R_0-R_inf)*exp(-b*p_) ;
47     Stensor a_[2];
48     // calcul du tenseur Sigma-X
49     Stensor sr = deviator(sig);
50     for(unsigned short i=0;i!=2;++i){
51       a_[i]       = a[i]+theta*da[i];
52       sr          -= C[i]*a_[i]/1.5;
53     }
54     const stress  seq = sigmaeq(sr);
55     Stensor n = 1.5*sr/seq;
56     feel = deel - deto + dp*n ;
57     fp    = (seq-Rp_)/young;
58     for(unsigned short i=0;i!=2;++i){
59       fa[i] = da[i] - dp*(n-g[i]*a_[i]);
60     }
61   } else {
62     // solution elastique
63     feel = deel - deto;
64   }
65 }
```

— la valeur de p est actualisée en $t_i + \theta\Delta\, t$ à la ligne 45, ce qui permet d'actualiser l'écrouissage isotrope $R\,(p)$;
— de même, pour les variables d'écrouissage cinématique α_i en ligne 51. Ceci permet de calculer le tenseur $(\underline{\sigma}|_{t+\Delta\, t} - \underline{X}|_{t+\Delta\, t})$ à la ligne 52 ;
— la direction d'écoulement $\underline{n} = \dfrac{3}{2}\dfrac{\tilde{\sigma} - \underline{X}}{(\sigma - \underline{X})_{\text{eq}}}$;
— la ligne 56 décrit simplement la décomposition additive des déformations ;
— la ligne 57 traduit le critère de plasticité (normalisé par le module d'Young pour conserver un système d'équations portant sur des grandeurs de la même dimension que les déformations ;
— les lignes 58 à 60 décrivent les évolutions des variables cinématiques :

$$\Delta\,\underline{\alpha}_i - \Delta p\,(\underline{n} - \gamma_i \underline{\alpha}_i) = \underline{0}$$

— enfin la ligne 63 correspond au cas où l'incrément est élastique.

7. calcul de l'opérateur tangent cohérent : il utilise directement l'inverse de la matrice jacobienne [Helfer 13b] :

```
67 @TangentOperator{
68   if((smt==ELASTIC)||(smt==SECANTOPERATOR)){
69     computeElasticStiffness<N,Type>::exe(Dt,lambda,mu);
70   } else if (smt==CONSISTENTTANGENTOPERATOR){
71     StiffnessTensor De;
72     Stensor4 Je;
73     computeElasticStiffness<N,Type>::exe(De,lambda,mu);
74     getPartialJacobianInvert(Je);
75     Dt = De*Je;
76   } else {
77     return false;
78   }
79 }
```

La variable `smt` permet de connaître le type d'opérateur tangent demandé par le code appelant.

Cette écriture avec `mfront` et l'algorithme implicite (et matrice jacobienne numérique) permettent d'obtenir une résolution efficace (plus efficace que son équivalent avec l'algorithme explicite de RUNGE-KUTTA, qui s'écrit de façon similaire, en remplaçant « `feel` », « `fp` », par « `deel` », « `dp` », *etc.*).

Avec cet exemple, on, peut déjà tester le développement sur un point matériel à l'aide de `mtest`, puis sur un maillage éléments finis (par exemple avec `Code_Aster`).

Pour une sollicitation uniaxiale de 10 cycles en traction - compression, on obtient très rapidement (0,024s) le résultat donné en figure 5.

Le fichier `mtest` pour cet exemple est le suivant :

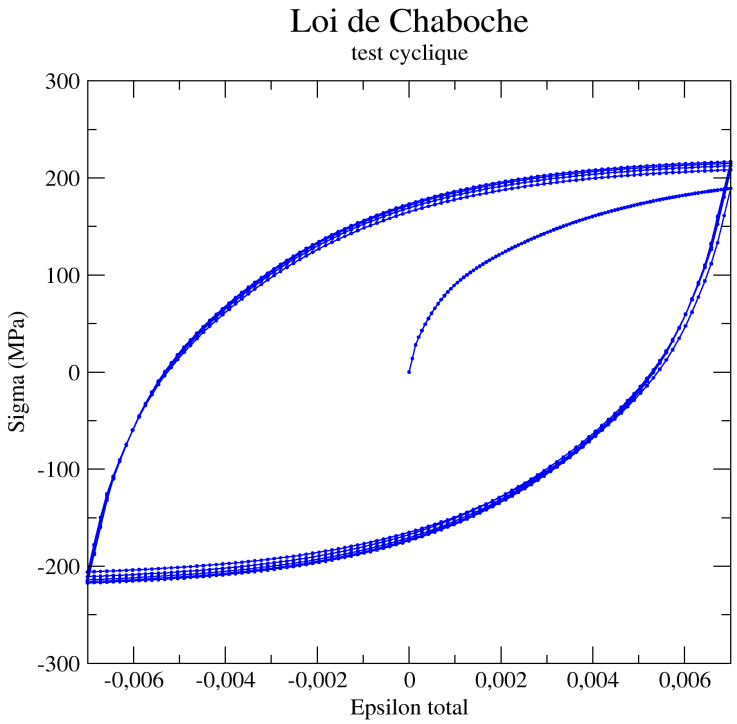


FIGURE 5 : Vérification de l'implantation de la loi élasto-plastique.

```
@Behaviour<aster> 'src/libAsterBehaviour.so' 'asterchaboche';
@MaterialProperty<constant> 'young' 200000. ;
@MaterialProperty<constant> 'nu'   0.33 ;
@MaterialProperty<constant> 'R_inf' 50. ;
@MaterialProperty<constant> 'R_0' 30. ;
@MaterialProperty<constant> 'b'   20. ;
@MaterialProperty<constant> 'C[0]' 187000.;
@MaterialProperty<constant> 'C[1]' 45000.;
@MaterialProperty<constant> 'g[0]'   4460. ;
@MaterialProperty<constant> 'g[1]'   340. ;

@ExternalStateVariable 'Temperature' 0.;

@ImposedStrain 'EYY' {0.: 0., 1.: 0.007, 2.: -0.007, 3.: 0.007, 4.: -0.007,  5.: 0.007,  6.: -0.007,  7.: 0.007,  8.: -0.007,  9.: 0.007, 10.: -0.007,11.: 0.007,
        12.: -0.007, 13.: 0.007, 14.: -0.007, 15.: 0.007, 16.: -0.007,  17.: 0.007, 18.: -0.007, 19.: 0.007, 20.: -0.007, 21.: 0.007, 22.: -0.007};

@Times {0.,10. in 1000};
```

3.2 LOI ÉLASTO-VISCOPLASTIQUE DE CHABOCHE

Le modèle diffère peu du précédent, car seul le critère de plasticité : $F(\sigma, X) = (\underline{\sigma} - \underline{X})_{\text{eq}} - R(p) \leq 0$ est remplacé par une loi d'écoulement en puissance :

$$\dot{p} = \langle \frac{F}{K} \rangle^m$$

où $\langle F \rangle$ désigne la partie positive de F définie ci-dessus, soit :

$$\langle F \rangle = \max(0, F)$$

Dans le fichier mfront, seule l'équation donnant l'évolution de $\Delta\, p$ change :

```
fp      = (seq_-Rp_)/young;
```

devient :

```
vp  = pow(F*UNsurK,m) ;
fp -= vp*dt;
```

UNsurK et m ont bien sûr été ajoutés dans la définition des propriétés matériau de la loi :

```
@MaterialProperty real m;
@MaterialProperty real UNsurK;
```

Un nouveau test effectué à l'aide de mtest permet de montrer la capacité du modèle dans le cas d'un chargement cyclique (cycles en température en déformation), issu du test hsnv125 de Code_Aster. La durée du calcul est de 0,58s. Le fichier mtest est le suivant :

```
@Behaviour<aster> 'src/libAsterBehaviour.so' 'asterviscochab';

@Real 'ThermalExpansionReferenceTemperature' 0.;

@MaterialProperty<function> 'young' '2.e5 - (1.e5*((TC - 100.)/960.)**2)';
@MaterialProperty<constant> 'nu' 0.3 ;
@MaterialProperty<function> 'ThermalExpansion' '1.e-5 + (1.e-5 * ((TC - 100.)/960.) ** 4)';

@MaterialProperty<constant> 'R_inf' 100. ;
@MaterialProperty<constant> 'R_0' 200. ;
@MaterialProperty<constant> 'b' 20. ;
@MaterialProperty<function> 'C[0]' '(1.e6 - (98500. * (TC - 100.) / 96.))';
@MaterialProperty<constant> 'C[1]' 0.;
@MaterialProperty<function> 'g[0]' '(5000.-5.*(TC-100.))';
@MaterialProperty<constant> 'g[1]' 0.;
@MaterialProperty<function> 'm' '7. - (TC - 100.) / 160.' ;
@MaterialProperty<function> 'UNsurK' '4900./(4200.*(TC+20.)-3.*(TC+20.0)**2)';

@ExternalStateVariable 'Temperature' {  0.:  0.,   1.:1060.,
                                       61.:100., 121.:1060.,
                                      181.:100., 241.:1060.,
                                      301.:100., 361.:1060.,
                                      421.:100., 481.:1060.
                    };
@Evolution<function> 'TC' 'Temperature';

@Real 'DY1' -0.0208 ;
@Real 'DY2' -0.0008 ;
@Real 'Tau' '100.0*sqrt(2.0)' ;

@ImposedStrain 'EXX' { 0.: 0.0, 0.1 :'DY1', 1.0:'DY1',
                      61.:'DY2',121.:'DY1',181.:'DY2',241.:'DY1',
                     301.:'DY2',361.:'DY1',421.:'DY2',481.:'DY1'};

@ImposedStress 'SXY' {0.0: 0.0, 0.1 : 0.0 , 1.0: 'Tau', 1000. : 'Tau'};

@Times { 0.,  0.1 in 60,
           1.  in 60,   61.  in 60,  121.  in 60, 181.  in 60,
         241.  in 60,  301.  in 60,  361.  in 60, 421.  in 60,
       449.8   in 29, 465.4  in 15, 473.8  in 9,  481.  in 45 };
```

La figure 6 donne l'évolution de différentes variables d'intérêt au cours du test.

La figure 7 montre que la comparaison de la loi développée en mfrontavec son équivalent dans Code_Aster fournit des résultats identiques.

3.3 TRAITEMENT DES ERREURS, DÉVERMINAGE

Et si, au cours du développement, on rencontre des erreurs, que faire ?

Lors de la compilation avec mfront, les erreurs sont la plupart du temps explicites :

- `Viscochab.mfront:94: error: 'F' was not declared in this scope`
- `Viscochab.mfront:74: warning: unused variable 'Rp_'`
- `Viscochab.mfront:91: error: expected ',' or ';' before 'if'` (oubli d'un « ; » en fin de ligne
- etc...

Lors de l'exécution, il est possible de localiser les erreurs numériques ou autres via les options de compilation :

- le plus simple est de compiler avec `--debug` (détails sur l'intégration) :
  `mfront --obuild --interface=aster --debug chaboche.mfront`

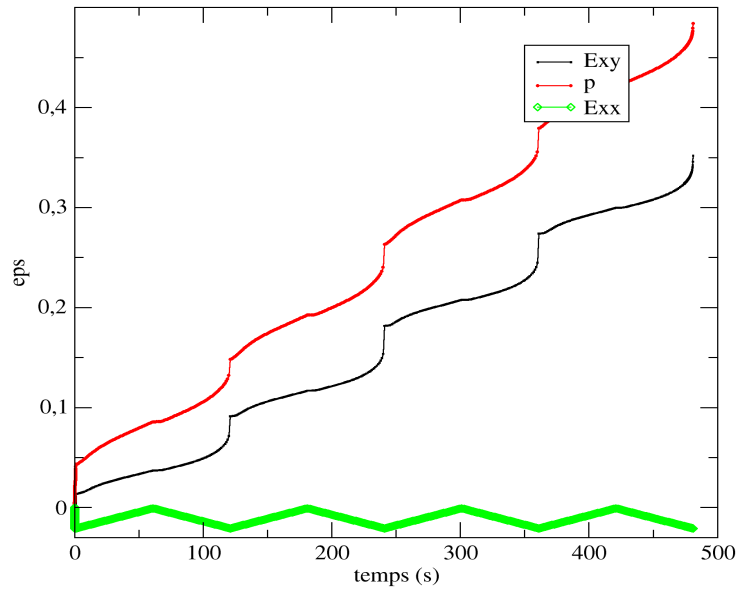


FIGURE 6 : Vérification de l'implantation de la loi élasto-visco-plastique.

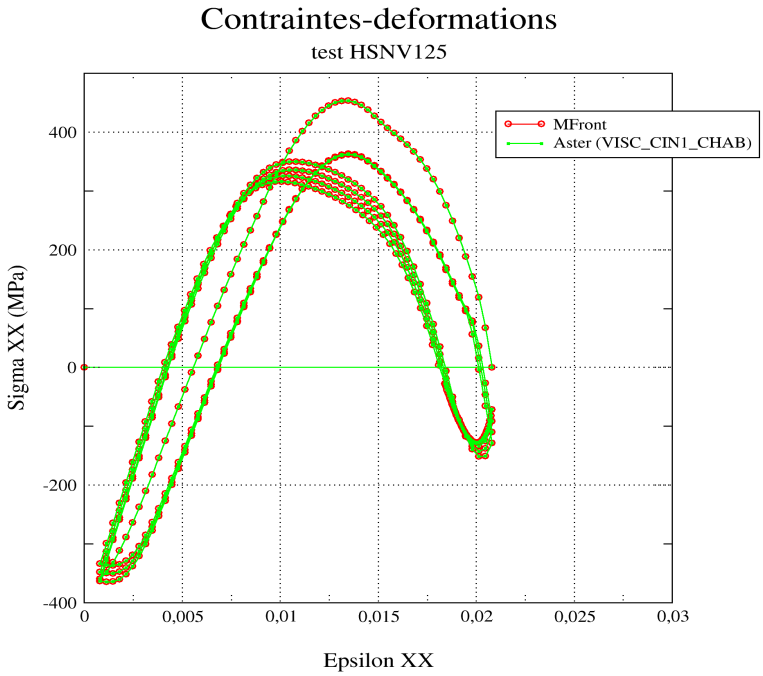


FIGURE 7 : test hsnv125 pour comparer la loi élasto-visco-plastique à celle de `Code_Aster`.

— un « classique » du développement, les impressions locales :
  — pour imprimer une variable locale, il suffit d'écrire :
    `cout << " seq calculé " << seq << endl;`
  — pour afficher l'état courant de l'intégration :
    `cout << *this << endl ;`
— pour trouver l'endroit où une erreur d'exécution se produit, on peut compiler en mode debug :
  `CXXFLAGS='-g'` `mfront --obuild --interface=aster chaboche.mfront`
— enfin, dans un calcul `Code_Aster`, on peut générer des fichiers mtest en cas d'échec d'intégration :
  `mfront --obuild --interface=aster --@AsterGenerateMTestFileOnFailure=true chaboche.mfront`

3.4 ZOOM SUR LES PERFORMANCES

En peu de lignes, nous avons construit un modèle élasto-visco-plastique qui fournit de bons résultats. Ceci montre la souplesse de `mfront`. Mais qu'en est-il des performances ? De nombreux benchmarks ont été effectués [Proix 13], mais on peut examiner ce qu'il en est sur le comportement que nous venons de développer :

— le test prend 0,578s avec `mtest` ;
— dans `Code_Aster`, la résolution sur un point matériel avec le comportement `mfront` est moins rapide qu'avec `mtest` : 2,03s (car le temps cpu fixe relatif à l'environnnement d'exécution est de l'ordre de 1s).
— toujours dans `Code_Aster`, la résolution avec le comportement VISC_CIN2_CHAB prend 1.41s, ce qui est logique puisque l'intégration du comportement résout (par une méthode de type « sécante ») une seule équation au lieu de 19 pour l'implantation proposée ici.
— si maintenant on utilise le comportement VISCOCHAB dans `Code_Aster`, pour lequel la résolution est similaire à celle de `mfront`, sur un système de 21 équations, la résolution échoue en implicite. En explicite (méthode de Runge-Kutta d'ordre 2), le temps CPU pour ce test est d'environ 3mn.

La convergence avec mtest est très rapide, par exemple, au dernier pas de temps :

```
iteration 1 : 0.00439197 94.3891
iteration 2 : 0.00196436 67.3256
iteration 3 : 0.000240731 3.14792
iteration 4 : 2.22909e-06 0.0282276
iteration 5 : 1.88601e-10 0.000232322
iteration 6 : 5.82316e-15 1.49916e-08
convergence, after 6 iterations, order 1.1075
```

On peut constater que les résidus au cours des itérations montrent une très bonne convergence. La matrice tangente cohérente est donc précise.

Remarque Il est possible dans `mfront` d'intégrer de façon encore plus efficace des comportements élasto-visco-plastiques à écoulement normal (comportements standard généralisés). Ici le comportement est non standard, et l'écriture est suffisamment simple pour permettre de réaliser facilement des modifications du modèle tout en restant efficace.

3.5 AJOUT DE LA MATRICE JACOBIENNE

Les essais précédents montrent la souplesse de l'intégration d'une loi de comportement à l'aide de `mfront` : il suffit de décrire les équations du système à intégrer. La matrice jacobienne, nécessaire à la résolution de ce système par la méthode de Newton, est estimée par perturbation, ce qui s'avère en pratique de bonne qualité, mais prend un temps machine plus important qu'une matrice directement programmée.

Pour illustrer cela, nous allons introduire la matrice jacobienne dans le comportement précédent. Il faut donc supprimer (ou commenter //) la ligne :

// @Algorithm NewtonRaphson_NumericalJacobian ;

La matrice jacobienne est composée des différentes dérivées croisées de chaque équation f_x par rapport à chaque inconnue dy. Par convention, la dérivée correspondante se nommera `dfx_ddy`. De plus, la matrice jacobienne est initialisée à l'identité.

Listons les différentes dérivées à calculer :

$$\text{dfeel\_deel} = \frac{\partial}{\partial \Delta \underline{\epsilon}^{\text{el}}} \left(\Delta \underline{\epsilon}^{\text{el}} + \Delta\, \underline{\varepsilon}^{p} - \Delta\, \underline{\varepsilon} \right) = \underline{\underline{\mathbf{I}}} + \Delta\, p \frac{\partial \underline{n}}{\partial \Delta\, \underline{\epsilon}^{\text{el}}} \quad \text{avec} \quad \tilde{I}_{ijkl} = \frac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk})$$

$$\text{dfeel\_ddp} = \underline{n} + \Delta\, p \frac{\partial \underline{n}}{\partial \Delta\, p}$$

$$\text{dfeel\_dda1} = \Delta p \frac{\partial \underline{n}}{\partial \Delta\, \alpha_1}$$

$$\text{dfeel\_dda2} = \Delta\, p \frac{\partial \underline{n}}{\partial \Delta\, \alpha_2}$$

$$\text{dfp\_ddp} = 1 + \Delta\, t \frac{m}{K} \left\langle \frac{F}{K} \right\rangle^{m-1} \frac{\partial R}{\partial \Delta\, p}$$

$$\text{dfp\_dda1} = -\Delta\, t \frac{m}{K} \left\langle \frac{F}{K} \right\rangle^{m-1} \frac{\partial (\sigma - X)_{\text{eq}}}{\partial \Delta\, \alpha_1}$$

$$\text{dfp\_ddeel} = -\Delta\, t \frac{m}{K} \left\langle \frac{F}{K} \right\rangle^{m-1} \frac{\partial (\sigma - X)_{\text{eq}}}{\partial \Delta\, \underline{\epsilon}^{\text{el}}}$$

$$\text{dfa1\_dda1} = \underline{\underline{\mathbf{I}}} - \Delta\, p \frac{\partial \underline{n}}{\partial \Delta\, \alpha_1} + \Delta\, p \gamma_1 \theta \underline{\underline{\mathbf{I}}}$$

$$\text{dfa1\_dda2} = -\Delta p \frac{\partial \underline{n}}{\partial \Delta\, \alpha_2}$$

$$\text{dfa2\_dda2} = \underline{\underline{\mathbf{I}}} - \Delta\, p \frac{\partial \underline{n}}{\partial \Delta\, \alpha_2} + \Delta\, p \gamma_2 \theta \underline{\underline{\mathbf{I}}}$$

$$\text{dfa2\_dda1} = -\Delta p \frac{\partial \underline{n}}{\partial \Delta\, \alpha_1}$$

$$\text{dfa1\_ddeel} = \text{dfa2\_ddeel} = -\Delta\, p \frac{\partial \underline{n}}{\partial \Delta\, \varepsilon^{e}}$$

$$\text{dfa1\_ddp} = -\underline{n} - \Delta\, p \frac{\partial \underline{n}}{\partial \Delta\, p} + \left(\gamma_1 + \Delta p \frac{\partial \gamma_1}{\partial \Delta p} \right) \underline{\alpha}_1$$

Les dérivées de la normale $\underline{n}$ peuvent être calculées de la façon suivante :

$$\frac{\partial \underline{n}}{\partial \Delta \underline{\epsilon}^{\text{el}}} = \frac{2\,\mu\,\theta}{(\underline{\sigma} - \underline{X})_{\text{eq}}} \left(\underline{\underline{\mathbf{M}}} - \underline{n} \otimes \underline{n} \right) \qquad \frac{\partial \underline{n}}{\partial \Delta\, \alpha_1} = \frac{-2\, C_1\, \theta}{3\, (\underline{\sigma} - \underline{X})_{\text{eq}}} \left(\underline{\underline{\mathbf{M}}} - \underline{n} \otimes \underline{n} \right)$$

Le tenseur $\underline{\underline{\mathbf{M}}}$ a été utilisé pour simplifier les expressions : $\underline{\underline{\mathbf{M}}} = \frac{3}{2}\underline{\underline{\mathbf{I}}} - \frac{1}{2}\underline{I} \otimes \underline{I}$

Ces expressions peuvent être écrites directement dans le fichier `mfront` :

```
const real Fexp   = UNsurK*pow(F*UNsurK,m-1) ;
const Stensor4 Jmn  = Stensor4::M() - (n_^n_);
dfeel_ddeel += 2.*mu*theta*dp*Jmn*inv_seq ;
dfeel_ddp    = n_;     // ajouter termes dn/dp si C fonction de p
dfp_ddeel    = - Fexp*m*dt* 2.*mu*theta*(n_| Stensor4::M() )/1.5;
dfp_ddp      = 1 + theta* Fexp *m *dt*b*(Rinf-Rp);
for(unsigned short i=0;i!=2;++i){
   dfeel_dda(i)   = -C[i]*dp*theta* inv_seq/1.5*Jmn ;
   dfp_dda(i)     =  Fexp*m*dt*C[i]*theta/1.5*n;
   dfa_ddeel(i)   = -2.*mu*theta*dp*Jmn *inv_seq;
   dfa_ddp(i)     = -n_ + g[i]*a_[i];
```

```
12          dfa_dda(i,i)   =  (1+dp*g[i]*theta)*Stensor4::Id()+C[i]*dp*theta*inv_seq/1.5*Jmn;
13      }
14      dfa_dda(0,1)   =  C[1]*dp*theta*inv_seq/1.5*Jmn;
15      dfa_dda(1,0)   =  C[0]*dp*theta*inv_seq/1.5*Jmn;
```

On constate donc que la seule difficulté consiste à calculer les expressions analytiques de ces dérivées à la main ! Une fois écrite cette matrice jacobienne, on peut vérifier sa précision en la comparant à la matrice jacobiennne numérique :

```
@CompareToNumericalJacobian true;
@JacobianComparisonCriterium 1.e-6;
```

Ces instructions peuvent être ajoutées dans le fichier `mfront` , ou mieux, sur la ligne de commande à la compilation (ce qui évite de « polluer » le fichier mfront) :

```
mfront --obuild --@CompareToNumericalJacobian=true
--@JacobianComparisonCriterium=1.e-6 chaboche.mfront
```

On mesure ensuite son efficacité sur les tests. Le test précédent prend cette fois $0,43s$ (au lieu de $0,578$ avec matrice jacobienne numérique). Sur un point matériel, cela est sans conséquence, mais le gain peut être imporant dans un calcul de structure.

Une fois que le comportement développé à l'aide de `mfront` est validé sur point matériel, avec `mtest`, et que la matrice jacobienne est correcte, il peut ensuite être utilisé dans des calculs de structure, aux grandes déformations (par exemple en utilisant `GDEF_LOG` dans `Code_Aster` ou d'autres codes). Les benchmarks effectués montrent l'efficacité de ce type d'intégration (même pour des comportements complexes, par exemple en plasticité cristalline, cf. [Proix 13]).

3.6 AJOUT DE LA NON RADIALITÉ ET DE LA MÉMOIRE D'ÉCROUISSAGE

Un des avantages de l'écriture très compacte des lois de comportement avec `mfront` réside dans la facilité d'intervention (amélioration, développement, maintenance) dans le fichier `mfront`.

Si, par exemple, on souhaite ajouter des effets de non-radialité au comportement précédent (ici, pour simplifier l'exposé, avec matrice jacobienne numérique), il faut modifier légèrement l'évolution de l'écrouissage cinématique :

$$\dot{\underline{\alpha}}_i = \dot{\varepsilon}^p - \gamma_i \alpha_i \dot{p}$$

devient :

$$\dot{\underline{\alpha}}_i = \dot{\varepsilon}^p - \gamma_i \left[\delta_i \underline{\alpha}_i + (1-\delta_i)(\underline{\alpha}_i : n) n\right] \dot{p}$$

où δ_i sont des coefficients compris entre 0 et 1 (1 revient à annihiler l'effet de non-radialité).

Qu'est-ce que l'effet de non-radialité ? Il se traduit par un écrouissage supplémentaire lors de chargements cycliques où les composantes du tenseur des contraintes en un point ne sont pas proportionnelles. Pour illustrer ce phénomène l'essai de traction-torsion permet de contrôler l'aspect non-radial et de mettre en évidence le sur-écrouissage dans le cas où la traction et la torsion sont déphasées [Shamsaei 11].

La discrétisation implicite donne pour cette équation :

$$\text{fa}_i = \Delta\,\alpha_i - \Delta\,\varepsilon^p + \gamma_i(p+\theta\Delta\,p)\left[\delta_i(\underline{\alpha}_i + \theta\Delta\,\alpha_i) + (1-\delta_i)\left((\underline{\alpha}_i + \theta\Delta\alpha_i) : n\right) n\right]\Delta\,p = 0$$

ce qui modifie peu de choses dans le fichier `mfront` :

1. au début du fichier, la déclaration des propriétés matériau supplémentaires :

```
21  @MaterialProperty real delta1;
22  @MaterialProperty real delta2;
```

2. et dans l'intégration :

```
86        fa1   = da1 -vp*rac32*n + g1_*vp*(deltal*a1_+(1-deltal)*a1n*n);
87        fa2   = da2 -vp*rac32*n + g2_*vp*(deltal*a2_+(1-delta2)*a2n*n);
88        feel += vp*rac32*n-deto;
```

On peut voir l'effet de non-radialité par exemple sur un test de traction-torsion où les déformations axiales et celles de cisaillement suivent un trajet en étoile (cf. figure 8). Chaque branche correspond à un aller -retour dans le diagramme gamma-epsilon.

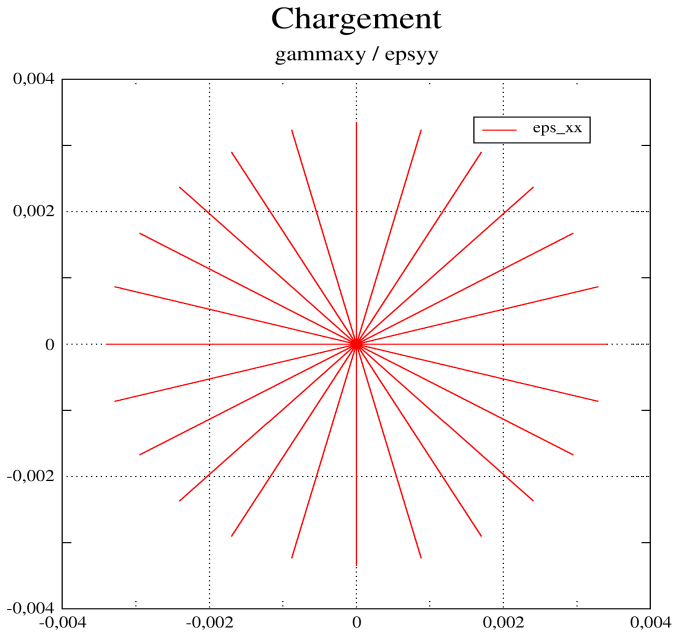


FIGURE 8 : chargement en étoile en traction-torsion.

La réponse obtenue avec un modèle sans effet de non-radialité (courbe verte sur la figure 9) est très différente de celle obtenue avec le modèle qui tient compte de cet effet (courbes rouge et bleue, superposées).

De même, l'ajout d'un effet de mémoire d'écrouissage ne pose pas de problème. Il est important de prendre en compte cet effet, lié à la plus grande amplitude de déformation plastique, pour des chargements cycliques. Cela revient à modifier la définition de l'écrouissage isotrope : la fonction $R(p)$ n'est plus connue explicitement, mais par l'intermédiaire du système d'équations :

$$R(p) = R_0 + R|_t + \Delta R \quad \text{avec} \quad \Delta R = b\,(Q - R)\;\Delta p$$

et

$$Q = Q_0 + (Q_m - Q_0)\left(1 - e^{-2\mu\left(q|_t + \Delta q\right)}\right)$$

On introduit donc une variable interne supplémentaire q, déterminée par le critère :

$$f\left(\underline{\epsilon}^{\mathrm{p}}, \underline{\xi}, q\right) = \frac{2}{3}\left(\underline{\epsilon}^{\mathrm{p}} - \underline{\xi}\right)_{\mathrm{eq}} - q = \frac{2}{3}\sqrt{\frac{3}{2}\left(\underline{\epsilon}^{\mathrm{p}} - \underline{\xi}\right) \;:\; \left(\underline{\epsilon}^{\mathrm{p}} - \underline{\xi}\right)} - q \leq 0$$

ce qui correspond à un domaine convexe dans l'espace des déformations plastiques de centre $\underline{\xi}$ et de rayon q. L'évolution de ce centre est donné par la loi de normalité :

$$\Delta\,\underline{\xi} = \frac{(1-\eta)}{\eta}\,\Delta\,q\,\underline{n}^{\star} \quad \text{avec} \quad \underline{n}^{\star} = \frac{3}{2}\frac{\underline{\epsilon}^{\mathrm{p}} - \underline{\xi}}{\left(\underline{\epsilon}^{\mathrm{p}} - \underline{\xi}\right)_{\mathrm{eq}}}$$

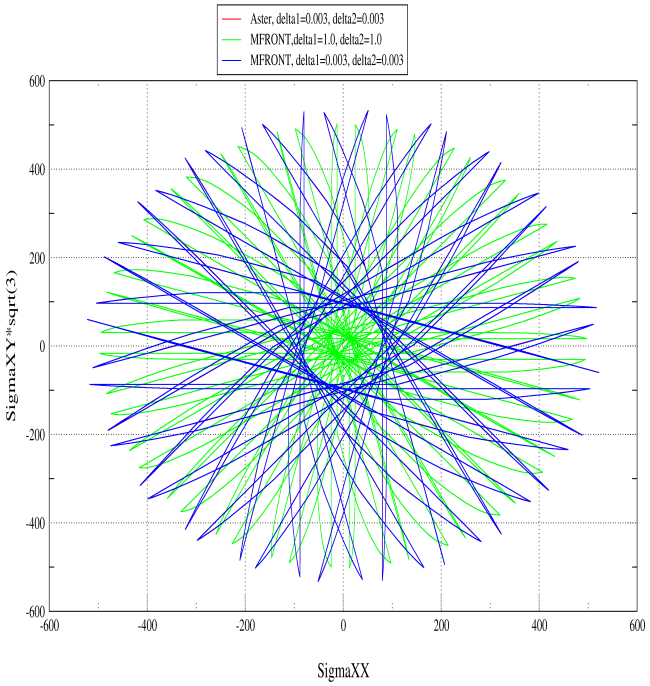


FIGURE 9 : simulation du chargement en étoile - modèles avec et sans non-radialité

Introduisons ces équations dans notre comportement `mfront` :

1. il faut ajouter les variables internes $\underline{\xi}$ et q :

```
@StateVariable real     q;
@StateVariable Stensor Ksi;
```

2. nous aurons besoin de mémoriser la valeur de l'écrouissage isotrope R. Mais il n'est pas nécessaire de définir R en tant que variable interne car ce n'est pas vraiment une inconnue du système d'équations (ΔR peut être déterminé explicitement à partir de Δp et Δq). On choisit donc de la définir comme variable auxiliaire, actualisée après convergence de l'algorithme d'intégration implicite :

```
@AuxiliaryStateVariable real R;
```

   En fin d'intégration, on trouvera donc :

```
@UpdateAuxiliaryStateVars{
  // valeur de milieu de pas, q ayant
  // deja ete mis a jour
  const real q_ = q-(1-theta)*dq;
  const real Q = Q0 + (Qm - Q0) * (1 - exp(-2 * Mu * q_) );
  R+=b*(Q-R)*dp;
}
```

3. il reste à ajouter les équations (`fq` et `fksi`) relatives à l'effet de mémoire :

```
    feel -= deto;
  }
}

@UpdateAuxiliaryStateVars{
  // valeur de milieu de pas, q ayant
  // deja ete mis a jour
  const real q_ = q-(1-theta)*dq;
  const real Q = Q0 + (Qm - Q0) * (1 - exp(-2 * Mu * q_) );
  R+=b*(Q-R)*dp;
}
```

4. Et il reste une petite modification sur le calcul du critère de plasticité, puisque $R(p)$ doit être maintenant déterminé à partir de Δp et Δq :

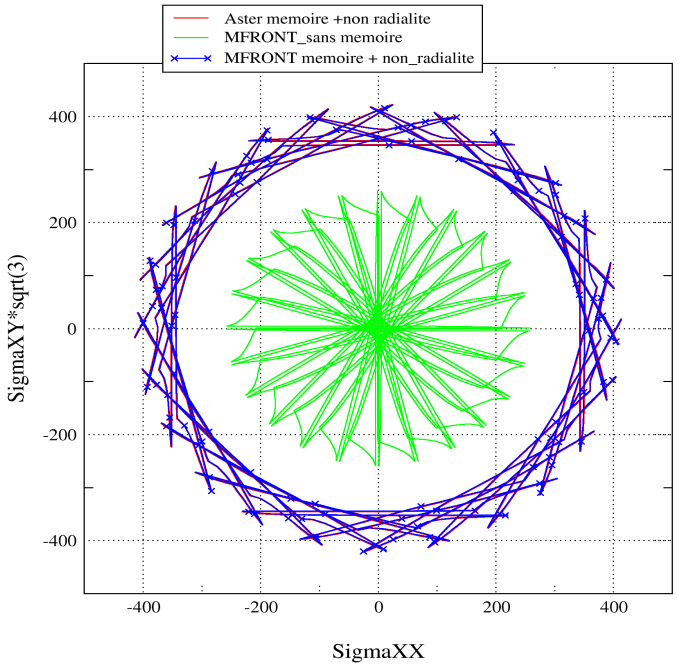


FIGURE 10 : chargement en étoile - modèles avec et sans mémoire-sigma-xy — sigma-xx

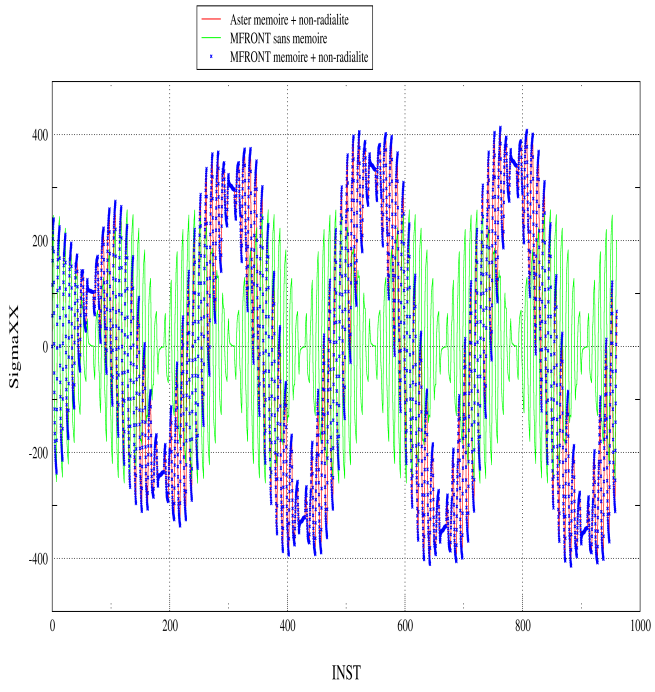


FIGURE 11 : chargement en étoile - modèles avec et sans mémoire- sigma-xx – temps

```
76    const real Q_     = Q0 + (Qm-Q0)*(1-exp(-2*Mu*(q+theta*dq)));
77    const real Rp     = R0 + R + b*(Q_-R)*dp ;
78    const real F      = seq - Rp ;
```

On obtient donc rapidement un modèle capable de représenter très correctement des essais de traction-torsion sur plusieurs cycles 10 et 11 (courbes rouge : `Code_Aster` et bleue : `mfront` confondues), alors que le modèle initial (sans mémoire ni effet de non radialité, courbe verte) est loin de la solution.

Il est possible, après avoir vérifié que le modèle convient, et fournit la solution attendue, de chercher à l'optimiser pour diminuer le temps de résolution ([EDF 14e]), en réduisant la taille du système et en introduisant la matrice jacobienne. Mais cet exemple d'un comportement déjà sophistiqué montre qu'il est possible très rapidement de tester des lois de comportement de façon opérationnelle.

Loi	Algorithme `mfront` et test
Chaboche	Implicite
HAYHURST (fluage et endommagement des aciers)	Implicite, jacobienne , GDEF_LOG, Theta=0.5 et Explicite, Runge-Kutta
MAZARS (béton, endommagement)	Avec séchage et hydratation
BETON_BURGER_FP	Fluage propre du béton avec retrait
MONOCRISTAL (MC)	Implicite, jacobienne, petites déformations
POLYCRISTAL (MC)	Runge-kutta , 30 grains
MONOCRISTAL (DD_CFC)(+ irradiation)	Implicite, jacobienne, petites déformations
POLYCRISTAL (DD_CFC)	Runge-Kutta , 30 grains
MONOCRISTAL (DD_CC)(+ irradiation)	Implicite, jacobienne, petites déformations
META_LEMA_ANI	Implicite, jacobienne numérique, avec effet des phases métallurgiques
MONOCRISTAL (MC)	Implicite, jacobienne numérique, grandes déformations

TABLEAU 1 : Quelques exemples de lois de comportement dévelopées avec `mfront`.

Plusieurs autres lois de comportement ont été développées et sont inclues dans les tests de `mfront` et de `Code_Aster`. Nous en donnons une liste partielle au tableau 1. Pour chacun, le temps de développement n'a pas dépassé une demi-journée.

3.7 IDENTIFICATION DES PROPRIÉTÉS MATÉRIAU À L'AIDE D'ADAO

Une fois le modèle validé, il est souvent nécessaire de procéder à l'identification de ses propriétés matériau, souvent à partir de données expérimentales ou provenant d'autres modèles.

Nous avons vu que `mtest` permet d'effectuer très efficacement la simulation de la réponse d'un point matériel à une sollicitation en contraintes ou en déformations. Ce sera le « moteur » de l'identification, en utilisant l'outil `Adao` [Argaud 14], intégré à la plate-forme `Salomé` [Salome 14].

`Adao` permet de trouver, par des algorithmes d'optimisation, les valeurs optimales d'un ensemble de paramètres noté X, qui minimisent une fonctionnelle $F = \left|\left|Y^{obs} - H(X)\right|\right|$, représentant l'écart entre des valeurs observées Y^{obs} et la simulation $H(X)$.

Le code de simulation (ici `mtest`, mais cela peut-être `Code_Aster`) calcule pour chaque ensemble de paramètres X la réponse $H(X)$.

Les valeurs Y^{obs} « observées » peuvent être expérimentales ou issues d'autres modèles. Sur la figure 12 : Y^{obs} correspond aux ordonnées de la courbe rouge, qui est une courbe expérimentale.

La courbe verte représente le résultat de la simulation pour un jeu de paramètres optimum $H(X^{final})$.

Une copie d'écran de l'interface de `Adao` dans `Salomé` est présentée sur la figure 13.

On peut voir sur cette figure :

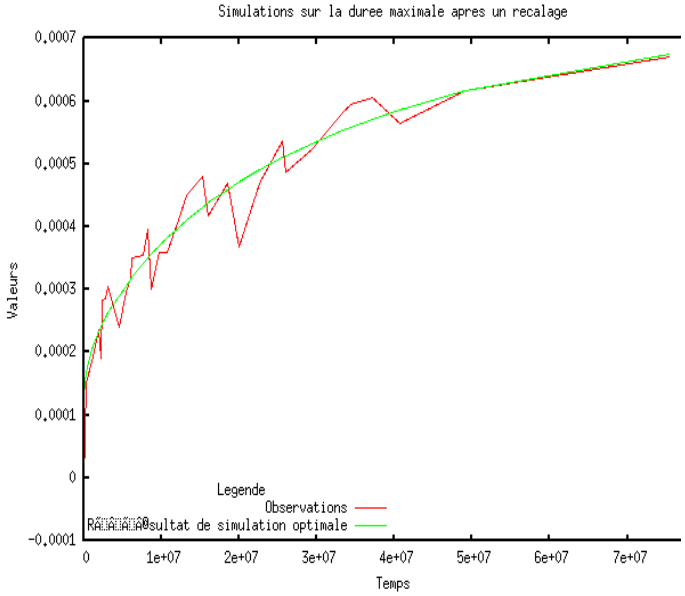


FIGURE 12 : courbe expérimentale et recalée

— en haut à gauche la décroissance de la fonctionnelle F au cours du processus ;
— en haut à droite les courbes expérimentales et calculées ;
— en bas les valeurs des paramètres finaux.

Recalage de la loi élasto-visco-plastique Nous allons utiliser `mtest` (associé au comportement élasto-visco-plastique précédent) comme moteur de simulation $H(X)$ pour chercher les valeurs des propriétés matériau de cette loi permettant de simuler les 4 courbes contraintes-déformation représentées en figure 14, pour différentes amplitudes de déformation imposée :

Il faut définir, pour `Adao`, *a minima* :
— les paramètres à identifier X ;
— leurs valeurs initiales X^{ini} ;
— la fonction $H(X)$.
Ici, nous choisissons de recaler les 5 propriétés matériau : C_1, C_2, γ_1, γ_2, b, dont les valeurs initiales sont :

```
[180000., 45000., 4460., 340., 10.]
```

La fonction $H(X)$ est définie par un appel direct à `mtest`, via une fonction `Python`, à partir des valeurs de X fournies par l'algorithme d'optimisation d'`Adao`.

Cette fonction `Python` permet d'appeler `mtest` directement sans passer par des fichiers intermédiaires (on retrouve les principales commandes de `mtest` décrites précédemment) :

L'opérateur $H(X)$ est défini par la fonction python qui a pour nom DirectOperator par convention dans `Adao`.

La transcription des instructions de `mtest` définissant les données dans la fonction `Courbe` est directe :

`@MaterialProperty<constant> 'Para' valeur` devient :

```
m.setMaterialProperty('Para',valeur)
```

`m` étant une instance de la classe `Mtest`.

Les arguments d'entrée de la fonction `Courbe` sont ici :
— les paramètres en cours d'identification `xx` ;

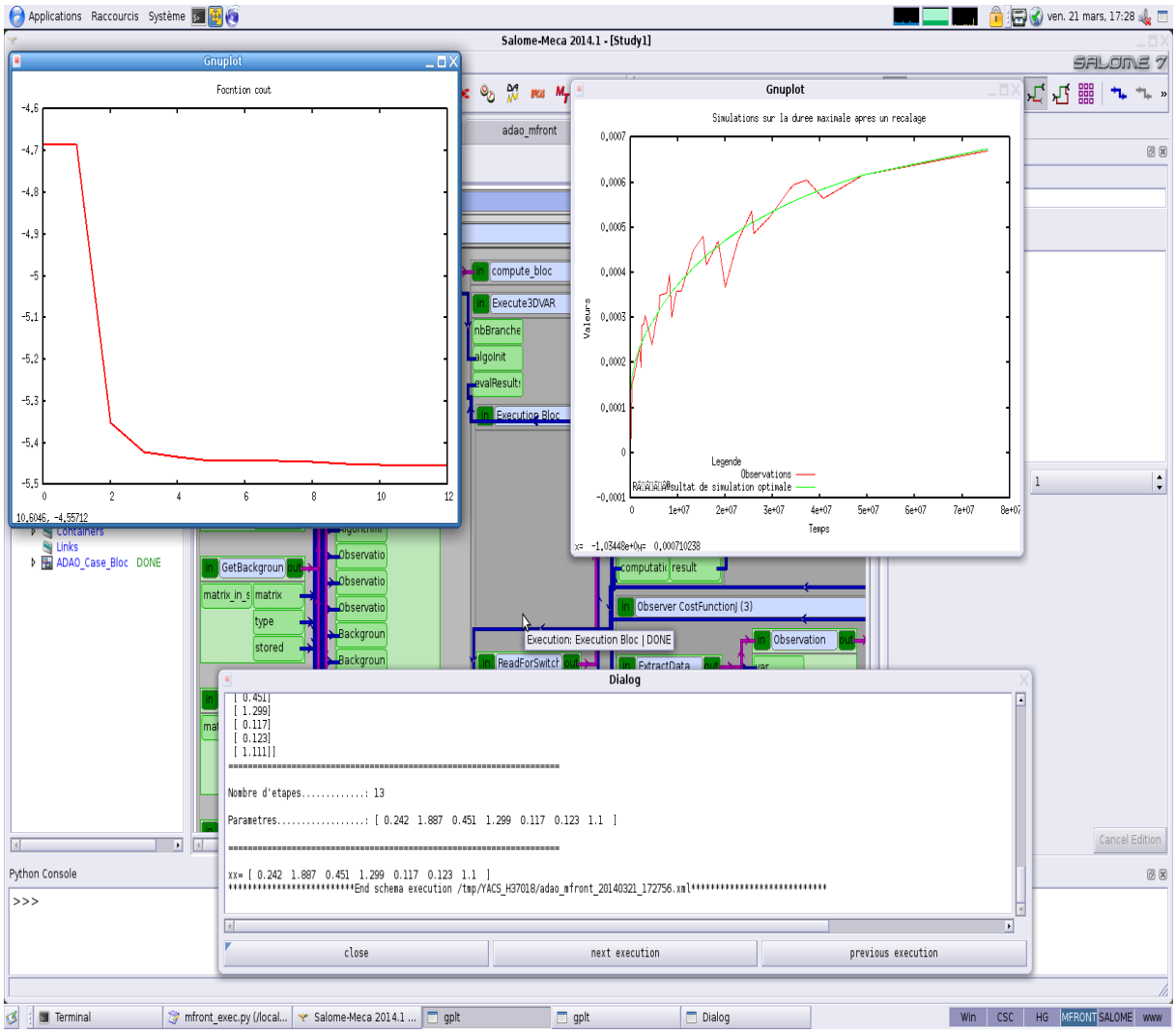


FIGURE 13 : recalage à l'aide de `Adao`

— la liste d'instants `linst` ;
— le dictionnaire python `dico` contenant la description du chargement.

Le chargement correspondant à chaque essai est en effet décrit par un dictionnaire dont les clés sont les instants et les valeurs sont les valeurs des déformations imposées.

Ces valeurs sont extraites ici d'un fichier annexe (`char.py`).

Les lignes 6 à 19 permettent de mettre ces données (déformations imposées en fonction du temps pour les 4 essais) sous forme de 4 dictionnaires python.

Dans la fonction `Courbe`, l'instruction `m.setImposedStrain('EXX ',dico)` permet d'imposer l'évolution de la composante XX des déformations en fonction du temps.

Quelques instructions supplémentaires permettent d'exécuter le calcul pas à pas, pour chaque instant, en gérant la reprise d'un instant à l'autre :

— `s = MTestCurrentState()` définit l'état courant ;
— `wk = MTestWorkSpace()` définit l'espace de travail ;
— `m.completeInitialisation()` initialise le calcul ;
— `m.initializeCurrentState(s)` initialise l'état courant ;

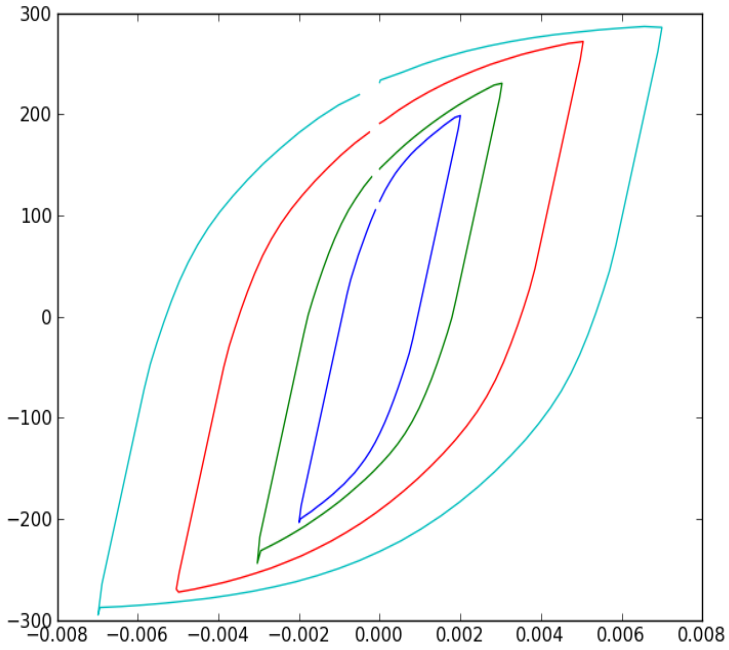


FIGURE 14 : courbes expérimentales contrainte (MPa)—déformation pour quatre amplitudes de déformation

— `m.initializeWorkSpace(wk)` initialise l'espace de travail ;
— `m.execute(s,wk,linst[i],linst[i+1])` calcule un pas de temps.

Le résultat de la fonction DirectOperator est un vecteur Y=H(X) qui est transmis à l'algorithme d'optimisation pour évaluer la fonctionnelle, et ses gradients par différences finies.

La figure 16 présente une copie d'écran illustrant les résultats de l'optimisation. On peut y voir :
— en haut à gauche la décroissance de la fonctionnelle F au cours du processus ;
— en haut à droite les courbes $\sigma = f(t)$ expérimentales et calculées ;
— en bas à droite les valeurs des paramètres finaux ;
— en bas à gauche le résultat de l'optimisation (courbes expérimentales et simulées quasi-confondues)

3.8 UTILISATION DANS CODE_ASTER

3.8.1 Modification du fichier de commandes

Une fois le comportement défini dans `mfront`, et compilé de façon habituelle :

```
mfront --obuild --interface=aster Chaboche.mfront
```

une bibliothèqe dynamique est produite :

```
libAsterBehabviour.so
```

Avant de l'utiliser dans un calcul de structures avec `Code_Aster`, il est conseillé d'effectuer les premiers tests sur un point matériel à l'aide de `mtest`.

Ensuite, tout est prêt pour une étude avec `Code_Aster`. Pour une description complète, on pourra consulter la notice d'utilisation [EDF 14d].

```python
# -*- coding: utf-8 -*-
def Courbe(xx, linst, diceps):
    m = MTest()
    setVerboseMode(VerboseLevel.VERBOSE_QUIET)
    m.setPredictionPolicy(PredictionPolicy.LINEARPREDICTION)
    m.setBehaviour('aster','src/libAsterBehaviour.so','asterviscochab');
    m.setImposedStrain('EXX',diceps)
    m.setMaterialProperty('nu',0.33)
    m.setMaterialProperty('young',184000.0)
    m.setMaterialProperty('rho',0.)
    m.setMaterialProperty('ThermalExpansion',0.)
    m.setMaterialProperty('C1',xx[0])
    m.setMaterialProperty('C2', xx[1])
    m.setMaterialProperty('g1' ,xx[2])
    m.setMaterialProperty('g2',xx[3])
    m.setMaterialProperty('R_0',30.)
    m.setMaterialProperty('R_inf',50.)
    m.setMaterialProperty('b',xx[4])
    m.setMaterialProperty('m',18.96)
    m.setMaterialProperty('UNsurK','1./120.')
    m.setExternalStateVariable('Temperature',0.)
    s  = MTestCurrentState()
    wk = MTestWorkSpace()
    m.completeInitialisation()
    m.initializeCurrentState(s)
    m.initializeWorkSpace(wk)
    YY1 = [0]
    n=len(linst)-1
    for i in range(n):
        m.execute(s,wk,linst[i],linst[i+1])
        YY1.append(s.s1[0])
    return YY1

def DirectOperator( XX ):
    xx = numpy.ravel(XX)
    import char
    exp=char.chargements
    colonne=0
    t1=exp[0].take([colonne],axis=1).ravel().tolist()
    t2=exp[1].take([colonne],axis=1).ravel().tolist()
    t3=exp[2].take([colonne],axis=1).ravel().tolist()
    t4=exp[3].take([colonne],axis=1).ravel().tolist()
    colonne=1
    e1=exp[0].take([colonne],axis=1).ravel().tolist()
    e2=exp[1].take([colonne],axis=1).ravel().tolist()
    e3=exp[2].take([colonne],axis=1).ravel().tolist()
    e4=exp[3].take([colonne],axis=1).ravel().tolist()
    diceps1=dict(zip(t1,e1))
    diceps2=dict(zip(t2,e2))
    diceps3=dict(zip(t3,e3))
    diceps4=dict(zip(t4,e4))
    YY1 = Courbe(xx, t1, diceps1)
    YY2 = Courbe(xx, t2, diceps2)
    YY3 = Courbe(xx, t3, diceps3)
    YY4 = Courbe(xx, t4, diceps4)
     # on ne conserve que le dernier cycle
    YY=YY1[410-82:410]+YY2[410-82:410]+YY3[410-82:410]+YY4[410-82:410]
    return numpy.array(YY)
```

FIGURE 15 : fonction python d'appel à `mfront`

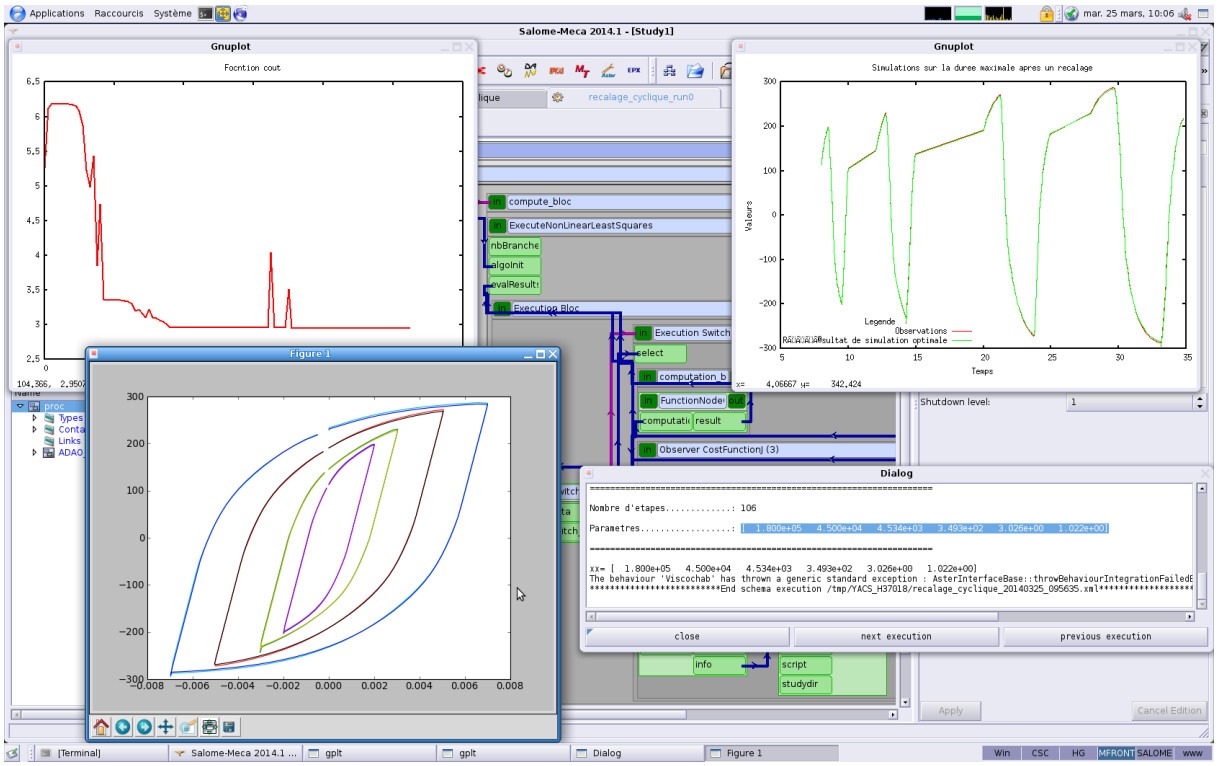


FIGURE 16 : Copie d'écran `Adao` du recalage sur 4 courbes cycliques

Pour cela, il suffit de fournir comme donnée de l'étude le fichier `libAsterBehabviour.so` (fichier de type "nom" dans `astk`).

Dans le fichier de commandes, il suffit de modifier deux points :

Dans la commande `DEFI_MATERIAU`, il faut ajouter :

```
.......=DEFI_MATERIAU( UMAT=_F( C1 = ... , C2 = ... , C3 = ... , ),
```

On fournit les propriétés matériau `Ci` dans l'ordre où elles sont définies dans le fichier `mfront`.

Par exemple pour le comportement élastoplastique de Chaboche, on donnera :

```
UMAT = _F(
      C1=Young,
      C2=Poisson,
      C3=R_Inf,
      C4=R_0,
      C5=B,
      C6=C1,
      C7=C2,
      C8=g1,
      C9=g2,
      NB_VALE=9,
)
```

Le mot clé `NB_VALE` permet d'optimiser le temps de lecture des propriétés.

Le deuxième endroit à modifier est le mot-clé `COMPORTEMENT` dans les commandes globales de calcul (`STAT_NON_LINE`, `DYNA_NON_LINE`, `SIMU_POINT_MAT`, ...),

Le comportement a pour nom « MFRONT ». On spécifie donc dans ces commandes :

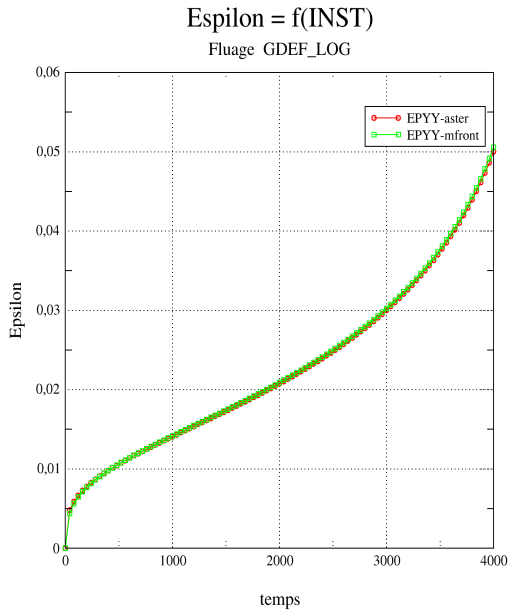
```
COMPORTEMENT=_F(RELATION='MFRONT',
                LIBRAIRIE='LibAsterBehaviour.so',
                NOM_ROUTINE='asterchaboche', (nom du comportement)
                NB_VARI=19, (nombre de variables internes)
                DEFORMATION='GDEF_LOG',)
```

Éprouvette entaillée 2D axi	MFRONT impl J prog	MFRONT impl J num	MFRONT expl RK4/5	Aster impl J prog	Aster impl J num	Aster expl RK2/1
Nb de pas de temps	601	601	4011	601	601	4011
Nb itérations Newton	1818	1810	17493	2479	1832	17495
Temps CPU	3mn5s	3mn19s	32mn21s	8mn40s	8mn53s	46mn37s

TABLEAU 2 : comparaison des temps de calcul

Comme dans l'exemple ci-dessus, il est possible d'ajouter le mot-clé `DEFORMATION` pour effectuer un calcul en grandes déformations logarithmiques ou le mot-clé `ITER_INTE_PAS` par exemple pour permettre un redécoupage local du pas de temps.

3.8.2 Exemple de résultats



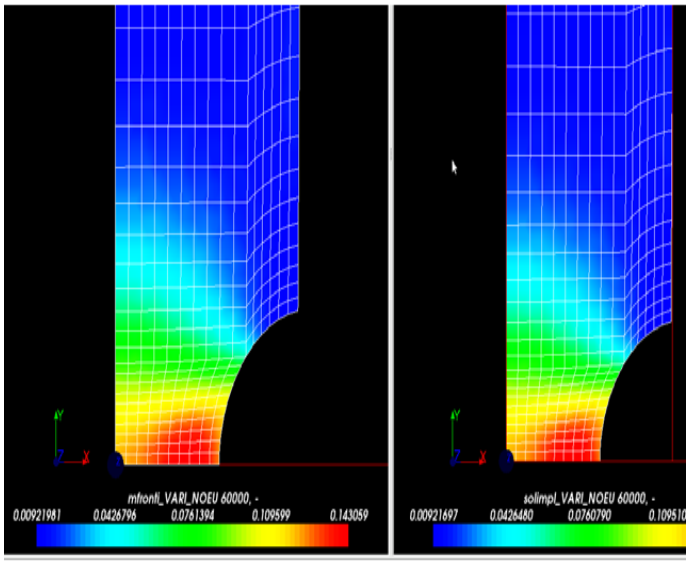


FIGURE 17 : déformation maxi— temps

FIGURE 18 : `Code_Aster`— `mfront`

En guise d'illustration, les résultats d'un calcul de fluage sur une éprouvette entaillée (maillage axisymétrique, 3300 DDL) avec `Code_Aster` sont donnés sur les figures 17 et 18.

La comparaison des temps CPU est nettement en faveur de `mfront`, détaillés au tableau 3.8.2. Notons que même l'intégration implicite avec jacobienne numérique est efficace.

4 IMPLANTATION D'UNE LOI VISCOPLASTIQUE ISOTROPE COMPRESSIBLE

Cette section décrit les étapes de l'implantation d'une loi de comportement élasto-viscoplastique typique du combustible nucléaire :

- nous commençons par décrire la loi de comportement en la décomposant en parties élémentaires (élasticité, viscoplasticité) ;
- nous détaillons ensuite comment implanter les propriétés élastiques de la loi (module d'YOUNG, coefficient de POISSON, coefficient de dilatation thermique) ;
- une première implantation est alors proposée. Elle se base sur une méthode de résolution par perturbation qui ne nécessite pas de fournir le jacobien du système. L'implantation obtenue n'est pas optimale mais permet de réaliser rapidement des tests ;
- nous montrons ensuite comment réaliser des premiers tests de vérifications ;
- nous reprenons ensuite le travail d'implantation de la loi pour obtenir une implantation optimisée de la loi :
  - par le calcul des termes du jacobien du système implicite ;
  - par le calcul de la matrice tangente cohérente.

4.1 DESCRIPTION DE LA LOI DE COMPORTEMENT

Nous considérons ici l'implantation d'une loi de comportement viscoplastique isotrope et compressible. Cette loi couple l'écoulement viscoplastique à l'évolution de la porosité du matériau.

4.1.1 Partition des déformations

On fait l'hypothèse classique en petites transformations que la déformation totale $\underline{\epsilon}^{\mathrm{to}}$ se répartit additivement en une déformation élastique $\underline{\epsilon}^{\mathrm{el}}$, une déformation viscoplastique $\underline{\epsilon}^{\mathrm{vis}}$ et une déformation thermique $\underline{\epsilon}^{\mathrm{th}}$:

$$\underline{\epsilon}^{\mathrm{to}} = \underline{\epsilon}^{\mathrm{el}} + \underline{\epsilon}^{\mathrm{vis}} + \underline{\epsilon}^{\mathrm{th}} \tag{1}$$

Soulignons dès à présent que cette relation, d'apparence anodine, sera en réalité centrale dans l'implantation que nous allons proposer :

- elle permet une grande modularité. Nous décrivons ailleurs qu'il « suffit » d'ajouter un nouveau terme de déformation pour représenter un nouveau phénomène (écoulement viscoplastique sous irradiation, plasticité dilatante représentant la décohésion des joints de grains, fissuration macroscopique) sans que cela impacte l'implantation de la partie viscoplastique (voir [Helfer 14a]).
- elle permet un calcul quasi-automatique de la matrice tangente cohérente. Si la matrice tangente cohérente n'est pas utilisée par le code éléments finis `Cast3M`, c'est un élément important pour les performances de la loi dans l'application combustible Cyrano [Thouvenin 10] ;
- elle permet un support simple et non intrusif des hypothèses en contraintes planes, notamment l'hypothèse d'axisymétrie en contrainte plane généralisée utilisée par l'application combustible Cyrano [Thouvenin 10].

Ces différents points justifient largement de conserver cette équation dans le système implicite que nous allons étudier dans la suite. L'implantation proposée pourra apparaître comme moins optimisée que ce que nous avons proposé par ailleurs [Helfer 05].

4.1.2 Élasticité

Le comportement élastique relie les déformations élastiques $\underline{\epsilon}^{\mathrm{el}}$ aux contraintes $\underline{\sigma}$ par la loi de HOOKE isotrope :

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

où nous avons utilisé les notations suivantes :

— $\underline{\underline{\mathbf{D}}}(T,p) = \lambda\,\underline{I} \otimes \underline{I} + 2\,\mu\,\underline{\underline{\mathbf{I}}}$ est la matrice de raideur, supposée dépendre de la température T et de la porosité ;
— $\lambda(T,p)$ et $\mu(T,p)$ sont respectivement les premier et second coefficients de LAMÉ du matériau ;
— $\mathrm{tr}\left(\underline{\epsilon}^{\mathrm{el}}\right)$ désigne la trace du tenseur $\underline{\epsilon}^{\mathrm{el}}$, c'est à dire la somme de ses termes diagonaux ;
— $\underline{I}$ est le tenseur identité d'ordre 2 ;
— $\underline{\underline{\mathbf{I}}}$ est le tenseur identité d'ordre 4 ;
— $\underline{I} \otimes \underline{I}$ est le tenseur d'ordre 4 tel que pour tout tenseur $\underline{s}$:

$$\underline{I} \otimes \underline{I}\,\underline{s} = \mathrm{tr}(s)\ \underline{I}$$

Propriétés élastiques En pratique, les coefficients de LAMÉ se déduisent du module d'YOUNG $E(T,p)$, fonction de la température et de la porosité p du matériau, et du coefficient de POISSON ν, supposé constant :

$$\lambda(T,p) = \frac{\nu\,E(T,p)}{(1+\nu)(1-2\,\nu)} \qquad \mu(T,p) = \frac{E(T,p)}{2\ (1+\nu)}$$

Pour illustrer notre propos, nous utiliserons une loi qui n'est pas celle des applications combustible CEA, mais une loi librement accessible [Fink 81] :

$$E(T,p) = 2,26\,10^{11}\ (1-2,62\,p)\ \left(1-1,31\,10^{-4}\ (T-273,15)\right)$$

Cette même référence préconise un coefficient de POISSON constant d'une valeur de $0,316$.

4.1.3 Dilatation thermique

Dans le cas des petites transformations, la dilatation thermique d'un corps est généralement traitée par le code appelant, en amont de la loi de comportement qui reçoit en entrée une déformation totale parfois appelée « mécanique ». Nous avons choisi de traiter explicitement la dilatation thermique pour pouvoir :
— traiter de manière exacte la dilatation thermique, la plupart des codes réalisant une approximation que nous décrirons plus loin ;
— utiliser notre loi en grandes déformations.

L'annexe A donne des précisions sur le traitement de la dilatation thermique. Dans la suite, le coefficient de dilatation thermique α sera supposé constant et égal à $10^{-5}\,K^{-1}$.

4.1.4 Écoulement viscoplastique

Nous supposons qu'il existe un potentiel de dissipation Φ fonction des contraintes dont dérive l'écoulement viscoplastique [Moreau 70, Besson 01, Chaboche 09].

La vitesse de déformation viscoplastique $\underline{\dot{\epsilon}}^{\mathrm{vis}}$ s'exprime alors :

$$\underline{\epsilon}^{\mathrm{vis}} = \frac{\partial \Phi}{\partial \underline{\sigma}}$$

Contrainte équivalente Dans la suite, nous faisons l'hypothèse que le potentiel de dissipation Φ est en fait une fonction d'une contrainte équivalente (scalaire) s_{eq} telle que :

$$s_{eq} = \sqrt{A(f)\ p^2 + B(f)\ \sigma_{\mathrm{eq}}^2}$$

où p est la pression hydrostatique et σ_{eq} est la contrainte équivalente au sens de VON MISES :

$$p = \frac{1}{3}\mathrm{tr}(\sigma)$$

$$\sigma_{\mathrm{eq}} = \sqrt{\frac{3}{2}\,dev(\underline{s})\ :\ dev(\underline{s})}$$

où $dev\left(\underline{s}\right)$ est la partie déviatorique des contraintes :

$$dev\left(\underline{s}\right) = \sigma - \frac{1}{3}\,\mathrm{tr}\left(\sigma\right)\underline{I}$$

Par définition, $dev\left(\underline{s}\right)$ est de trace nulle.

Les fonctions $A\left(f\right)$ et $B\left(f\right)$ dépendent de la porosité. Un choix possible pour ces fonctions est :

$$A\left(f\right) = \frac{9}{4}\left(E\,\left(f^{-1/E} - 1\right)\right)^{-\frac{2\,E}{E+1}}$$
$$B\left(f\right) = \left(1 + \frac{2}{3}\,f\right)\,\left(1-f\right)^{-\frac{2\,E}{E+1}}$$

où E est une constante. Nous verrons dans la suite qu'il est intéressant d'introduire un coefficient C_A tel que :

$$A\left(f\right) = \frac{9\,C_A}{4}\left(E\,\left(f^{1/E} - 1\right)\right)^{-\frac{2\,E}{E+1}}$$

Si C_A est nul, on retrouve une loi viscoplastique isotrope incompressible (sans effet de la porosité), ce qui est utile lors de l'étape de vérification de la loi.

Cette forme de potentiel de dissipation provient de l'homogénisation du problème d'une matrice poreuse incompressible isotrope supposée obéir à une loi viscoplastique de NORTON [Michel 92]. Elle a été utilisée à plusieurs reprises pour décrire le combustible nucléaire [Monerie 06]. La constante E apparaissant dans la définition des fonctions $A\left(f\right)$ et $B\left(f\right)$ est l'exposant de la loi de de NORTON de la matrice.

Potentiel elliptique Les mêmes résultats d'homogénéisation conduisent à proposer comme potentiel de dissipation un potentiel dit « elliptique » qui généralise l'expression d'une loi de NORTON :

$$\Phi\left(s_{eq}\right) = \frac{A_\Phi}{E+1}\,s_{eq}^{E+1}$$

Expression usuelle de l'écoulement plastique Développons maintenant l'expression de la vitesse d'écoulement viscoplastique :

$$\underline{\epsilon}^{\mathrm{vis}} = \frac{\partial\phi}{\partial\underline{\sigma}} = \frac{\partial\phi}{\partial s_{eq}}\frac{\partial s_{eq}}{\partial\underline{\sigma}} = \frac{\partial\phi}{\partial s_{eq}}\left[\frac{\partial s_{eq}}{\partial p}\frac{\partial p}{\partial\underline{\sigma}} + \frac{\partial s_{eq}}{\partial\sigma_{\mathrm{eq}}}\frac{\partial\sigma_{\mathrm{eq}}}{\partial\underline{\sigma}}\right] = \frac{\partial\phi}{\partial s_{eq}}\left[\frac{1}{3}\frac{\partial s_{eq}}{\partial p}\underline{I} + \frac{\partial s_{eq}}{\partial\sigma_{\mathrm{eq}}}\underline{n}\right]$$

où nous avons introduit le classique tenseur normal $\underline{n}$ égal à la dérivée de la contrainte équivalente de VON MISES par rapport au tenseur des contraintes :

$$\underline{n} = \frac{\partial\sigma_{\mathrm{eq}}}{\partial\underline{\sigma}} = \frac{3}{2\,\sigma_{\mathrm{eq}}}\,dev\left(\underline{s}\right)$$

où s est le déviateur des contraintes introduit plus haut. Le tenseur n est donc également déviatorique (de trace nulle).

Nous pouvons alors donner les expressions des différentes dérivées apparaissant dans cette expression :

$$\frac{\partial\phi}{\partial s_{eq}} = s_{eq}^{E} \qquad \frac{\partial s_{eq}}{\partial p} = \frac{1}{s}\,A\left(f\right)\,p \qquad \frac{\partial s_{eq}}{\partial\sigma_{\mathrm{eq}}} = \frac{1}{s}\,B\left(f\right)\,\sigma_{\mathrm{eq}}$$

4.1.5 Évolution de la porosité

L'évolution de la porosité est donnée par :

$$\dot{f} = \left(1-f\right)\,\mathrm{tr}\left(\underline{\dot{\epsilon}}^{\mathrm{vis}}\right)$$

4.2 PROPRIÉTÉS MATÉRIAU

Telle que nous l'avons présentée, notre loi de comportement s'appuie sur différentes propriétés du matériau :

- le module d'YOUNG ;
- le coefficient de POISSON ;
- le coefficient de dilatation thermique.

Afin de traiter la question de leur implantation en `mfront`, prenons le temps d'examiner deux questions :

- ces propriétés doivent-elles être évaluées par la loi de comportement ou par le code appelant ?
- vaut-il mieux utiliser des coefficients matériaux ou des paramètres ?

4.2.1 Évaluation par la loi ou par le code appelant ?

Ces propriétés doivent-elles être implantées dans la loi de comportement ou implantées à part, par exemple pour être accessibles dans d'autres lois de comportement ou, dans le cas du code aux éléments finis `Cast3M`, pour permettre la construction d'une matrice de raideur ?

La première solution nous apparaît comme la plus satisfaisante sur le principe : l'identification de la loi se fait pour des propriétés élastiques données qui font donc partie intégrante de la loi. En pratique, la seconde solution a pour l'instant toujours été retenue car elle minimisait le travail d'implantation d'une propriété.

Dans le cas du code `Cast3M`, cette propriété était souvent écrite en langage `gibiane` dans le jeu de données de la simulation, ce qui n'était pas sans poser de nombreux problèmes :

- l'écriture du code était fastidieuse pour les propriétés complexes et d'une mise en œuvre délicate dès lors qu'une propriété dépendait de plusieurs paramètres ;
- le code était « copier/coller » d'un jeu de données à un autre. Des tentatives de mutualisation ont bien été menées en se basant sur un fichier « UTILPROC » commun, mais force est de constater que malgré les efforts déployés, ces tentatives ne se sont pas révélées pérennes :
- l'implantation ne peut être réutilisée dans d'autres solveurs. Or, nous verrons dans la suite qu'il est intéressant et pratique de réaliser des tests unitaires sur la base du logiciel `mtest` puis de réaliser les calculs de structure à l'aide de `Cast3M`.

`mfront` apporte une solution quelque peu différente qui concilie les deux solutions évoquées plus haut : un même fichier source peut servir à générer une bibliothèque appelée depuis `Cast3M` (ou `Code_Aster`) ou directement être intégrée à l'implantation de la loi.

```
1  @Parser     MaterialLaw;
2  @Material UO2;
3  @Law        PoissonRatio_Fink1981;
4  @Author     T. Helfer;
5  @Date       04/04/2014;
6
7  @Description
8  {
9    Thermodynamic properties of Uranium dioxide
10   Fink, J.K. and Chasanov, M.G. and Leibowitz, L.
11   Argonne National Laboratory - ANL-CEN-RSD-80-3
12   04/1981
13 }
14
15 @Output nu;
16
17 @Function
18 {
19   nu=0.316;
20 }
```

FIGURE 19 : Implantation du coefficient de POISSON du dioxide d'uranium.

4.2.2 Implantation du coefficient de POISSON

La figure 19 présente l'implantation du coefficient de Poisson de la loi. Nous renvoyons à la documentation générale de `mfront` pour les détails liés à l'implantation d'une propriété matériau [Helfer 13a].

Il peut sembler lourd d'écrire un fichier source pour une propriété constante, à la fois en terme d'implantation, mais aussi en terme d'efficacité numérique (les codes de calcul savent gérer des valeurs constantes !). Cela présente cependant plusieurs avantages :

1. il est possible de stocker ce fichier dans une base de données, la base de données `sirius` de la plate-forme `pleiades` notamment ;
2. cette valeur n'est pas dans les jeux de données des codes appelant, mais conservée à un endroit unique : on ne duplique pas d'information physique à chaque nouveau calcul ;
3. certains codes permettent d'optimiser l'évaluation d'une fonction sans argument (le code `licos` notamment [Helfer 11]) et l'appel ne se fait qu'une fois.

Choix du nom du fichier `mfront` Le nom de la fonction générée sera `UO2_PoissonRatio_Fink1981` (combinaison du nom du matériau donné en ligne 2 et du nom de la loi donnée en ligne 3). Les noms des sources `C++` générées commenceront par ce nom de fonction. Pour simplifier l'intégration de ce fichier `mfront` dans un environnement de compilation, il est recommandé de nommer le fichier source `mfront` « `UO2_PoissonRatio_Fink1981.mfront` » car cela permet d'utiliser des règles de compilation génériques.

```
1  @Parser     MaterialLaw;
2  @Material UO2;
3  @Law        YoungModulus_Fink1981;
4  @Author     T. Helfer;
5  @Date       04/04/2014;
6
7  @Description
8  {
9    Thermodynamic properties of Uranium dioxide
10   Fink, J.K. and Chasanov, M.G. and Leibowitz, L.
11   Argonne National Laboratory - ANL-CEN-RSD-80-3
12   04/1981
13 }
14
15 @Input T,p;
16 T.setGlossaryName("Temperature");
17 p.setGlossaryName("Porosity");
18
19 @Output E;
20
21 @PhysicalBounds T in [0:*[;
22 @PhysicalBounds p in [0:1.];
23
24 @Bounds T in [0.:1600.];
25
26 @Function
27 {
28   E=2.26e11*(1-2.62*p)*(1-1.31e-4*(T-273.15));
29 }
```

FIGURE 20 : Implantation du module d'YOUNG du dioxide d'uranium.

4.2.3 Implantation du module d'YOUNG

La figure 20 montre comment est implanté le module d'YOUNG du dioxide d'uranium. Notons quelques nouveautés :

— la définition de bornes physiques de la loi en lignes 21 et 22 (mot clé `@PhysicalBounds`). En dehors de ces bornes, les arguments ont des valeurs non physiques ;
— la définition de bornes de la loi en ligne 24. Cette borne indique la limite de validité de la corrélation (mot clé `@Bounds`) ;
— la définition de noms de glossaire pour les variables d'entrée en lignes 16 et 17. Ces noms de glossaire permettent à des codes comme `mtest` et `licos` d'évaluer la propriété.

Implantation du coefficient de dilatation thermique L'implantation du coefficient de dilatation thermique est similaire à celle du coefficient de Poisson. Elle est indiquée en figure 21. Notons que nous avons explicitement indiqué la température de référence T_α dans une constante nommée `ReferenceTemperature`. Cette constante sera utilisée par la loi de comportement pour le calcul de la dilatation thermique.

```
@Parser     MaterialLaw;
@Material UO2;
@Law        ThermalExpansion;
@Author     T. Helfer;
@Date       04/04/2014;

@Description
{
  Material Property Correlations: Comparisons between
  FRAPCON-3.4, FRAPTRAN 1.4, and MATPRO
  W.G. Luscher and K.J. Geelhood
  Aout 2010
  Division of System Analysis
  Office of Nuclear Regulatory Research
  U.S. Nuclear Regulatory Commission
  NUREG/CR-7024 PNNL-19417
}

@Constant ReferenceTemperature = 300.;

@Output a;

@Function
{
  a=1.e-5;
}
```

FIGURE 21 : Implantation du coefficient de dilatation thermique du dioxide d'uranium.

Compilation Pour pouvoir être utilisé avec `mtest`, l'outil de test unitaire livré avec `mfront`, les propriétés matériau précédentes doivent être compilées avec l'interface `Cast3M` :

```
$ mfront --obuild --interface=castem UO2_PoissonRatio_Fink1981.mfront
UO2_YoungModulus_Fink1981.mfront UO2_ThermalExpansion_MATPRO.mfront
Treating target : all
The following library has been build :
- libCastemUO2.so
```

Cette commande crée une librairie nommée `libUO2Castem.so` dans un sous-répertoire nommé `src`.

Dans une phase de vérification, d'autres interfaces peuvent être intéressantes. Par l'exemple, l'interface `Python` crée un module qui peut être utilisé pour la vérification.

```
set grid
set term png

import<castem> UO2_YoungModulus_Fink1981(T,p) 'src/libCastemUO2.so'

print UO2_YoungModulus_Fink1981(293.15,0.05)

set output "UO2_YoungModulus_Fink1981.png"
set key right top
set xlabel "Temperature (K)"
set ylabel "Module d'Young (GPa)"

plot [293.15:1600] 1e-9*UO2_YoungModulus_Fink1981(x,0.05) t "Module d'Young, 5% de porosite" linewidth 6, \
     1e-9*UO2_YoungModulus_Fink1981(x,0.025) t "Module d'Young, 2,5% de porosite" linewidth 6, \
     2.26e2*(1-2.62*0.05)*(1-1.31e-4*(x-273.15)) with dots t "Expression analytique, 5% de porosite", \
     2.26e2*(1-2.62*0.025)*(1-1.31e-4*(x-273.15)) with dots t "Expression analytique, 2,5% de porosite"

k1 = 1.e-5
k2 = 3.e-3
k3 = 4.e-2
E  = 6.9e-20
k  = 1.38e-23
d1_11(T)=k1*T-k2+k3*exp(-E/(k*T))
d1_12(T)=k1*(T-k2/k1)

plot [300:1573.15] d1_11(x)  lw 4, d1_12(x) w l

print d1_11(300)
print k2/k1
```

FIGURE 22 : Script `tplot` permettant de tracer l'évolution du module d'YOUNG de l'uranium au travers de son implantation `mfront` et de la comparer à une expression analytique.

Vérification Il existe différentes façons de vérifier l'implantation d'une loi `mfront`. `licos` propose un utilitaire graphique nommée `tplot` qui propose un langage de script proche de celui du classique `gnuplot` et qui permet de comparer les valeurs retournées par la fonction générée avec `mfront` à l'expression analytique. La figure 22 présente le script

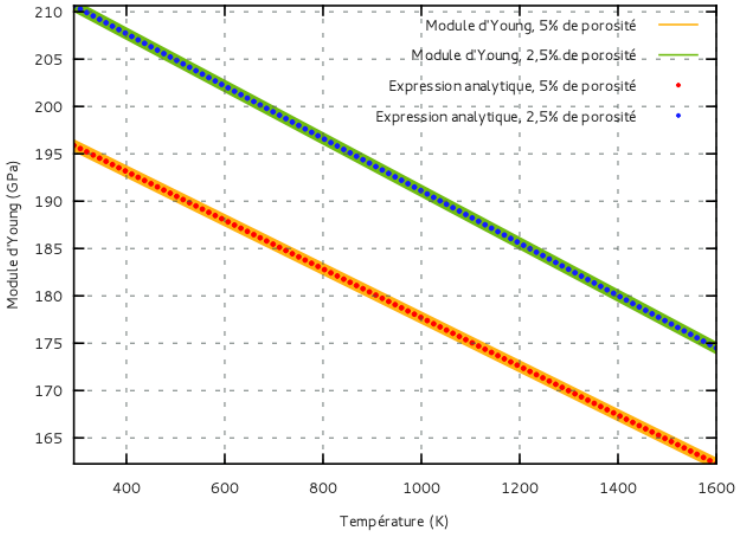


FIGURE 23 : Évolution du module d'YOUNG de l'UO_2. Données libres [Fink 81]

utilisé pour tracer la figure 23 : la seule spécificité par rapport à `gnuplot` est la ligne 4 (mot clé `import`) qui permet d'utiliser la fonction générée par `mfront`.

Pour une vérification systématique, les applications de la plate-forme `pleiades` peuvent utiliser l'utilitaire `mpplot` qui permet de sortir des fichiers colonnes qui peuvent être comparés à des fichiers de référence à l'aide de l'utilitaire `pleiades-check`.

4.2.4 Les coefficients de la loi : propriété matériau ou paramètre ?

En plus des propriétés thermoélastiques, la loi de comportement qui nous intéresse dans cette section présente de nombreux coefficients :

- exposant E de la loi de NORTON ;
- intensité A_ϕ de l'écoulement ;
- le coefficient C_A de la loi de comportement.

Par rapport aux propriétés thermoélastiques traitées au paragraphes précédents, ces coefficients sont propres à la loi et ont peu de chances d'être réutilisés dans d'autres lois.

Ces coefficients peuvent être déclarés soit comme des propriétés matériau *que le code appelant doit fournir* soit comme des *paramètres*. Chacune de ces modalités présente des avantages et des inconvénients que nous allons maintenant détailler : le fait de déclarer un coefficient comme étant un paramètre ou une propriété matériau est une affaire de compromis et d'habitude de travail.

Propriétés matériau Afin de pouvoir modifier les valeurs des coefficients d'une loi depuis un jeu de données `Cast3M` ou `Code_Aster`, l'interface `castem` « standard » impose de déclarer ces coefficients comme des propriétés matériau (mot clé `@MaterialProperty`) que doit fournir le code appelant.

Le principal avantage de cette méthode est de permettre l'écriture de lois de comportement génériques. En effet, l'écriture d'une loi de comportement avec un langage de bas niveau (`fortran` ou `C` par exemple) est une tâche longue, difficile et fastidieuse. L'usage de `mfront` limite considérablement la portée de cet argument.

Le second avantage de cette méthode est que l'utilisateur a une grande liberté et peut faire des tests en dehors de la loi de comportement, par exemple faire varier A_ϕ avec le taux de combustion.

Or, cet avantage est à notre avis le principal défaut de cette méthode puisqu'une partie de l'information physique est

dissociée de l'implantation de la loi : nous considérons préférable de rassembler toutes les informations physiques dans un unique fichier `mfront` (ou éventuellement dans des fichiers dédiés comme pour le module d'YOUNG) et cela d'autant plus que l'on souhaite capitaliser les lois dans des bases de données (la base `sirius` de la plate-forme `pleiades` par exemple).

L'autre défaut de cette méthode est que la gestion des propriétés matériau par le code appelant peut être coûteuse : une implantation d'une loi utilisant de nombreuses propriétés matériau est significativement plus lente à l'exécution que l'implantation de la même loi utilisant des propriétés matériau « en dur ».

La notion de paramètres Le besoin de modifier à l'exécution des coefficients de la loi demeure cependant, par exemple dans le cadre d'études de sensibilité. Dans ce cas, `mfront` propose une alternative efficace aux propriétés matériau : la notion de paramètre (mot clé `@Parameter`). Un paramètre se comporte comme une variable globale dont la valeur par défaut est spécifiée dans le fichier `mfront` (ce qui est impossible avec une propriété matériau). La valeur d'un paramètre peut être modifiée à l'exécution sans passer par l'appel standard à la loi de comportement et donc sans induire de coût supplémentaire, si le code appelant le permet. Seul le code `Cast3M` ne le permet pas, mais les applications de la plate-forme `pleiades`, le code `Code_Aster` ou `mtest`, peuvent modifier les paramètres de la loi depuis le langage `Python` ou le `C++`.

Insistons sur le fait qu'un paramètre est alors une caractéristique propre à une loi et non à un matériau : si deux matériaux distincts utilisent la même loi, la modification d'un paramètre affecte le comportement des deux matériaux.

Dans la suite, nous déclarerons les coefficients E, A_ϕ et C_A comme des paramètres.

4.3 DISCRÉTISATION ET IMPLANTATION PAR UNE θ MÉTHODE

Nous avons choisi d'intégrer notre loi de comportement par une θ-méthode pour différentes raisons :
- c'est la méthode la plus performante ;
- on a accès à la matrice tangente cohérente ;
- les autres lois du combustible (modèle de fissuration DDIF2 notamment) sont déjà implantées dans ce formalisme.

4.3.1 Gestion de la dilatation thermique

Nous utilisons le mot clé `@ComputeThermalExpansion` pour que `mfront` gère la dilatation thermique. Celle-ci sera gérée différemment selon que la loi est utilisée en petites déformations ou en grandes déformations. Les variables `eto` et `deto` qui apparaîtront dans l'implantation sont respectivement la déformation totale mécanique (déformation totale moins la dilatation thermique) en début de pas et son incrément.

4.3.2 Réduction du nombre de variables internes

Telle que présentée au paragraphe précédent, notre loi possède 3 variables internes :
- la déformation élastique $\underline{\epsilon}^{\mathrm{el}}$;
- la déformation viscoplastique $\underline{\epsilon}^{\mathrm{vis}}$;
- la porosité ;

Cas de l'écoulement viscoplastique Commençons par traiter le cas de la déformation viscoplastique $\underline{\epsilon}^{\mathrm{vis}}$. Il serait assez simple de retirer complètement cette variable du système et économiser ainsi 6 variables internes dans le système. Pour simplifier l'écriture du système implicite et pour simplifier les post-traitements, nous avons choisi dans ce tutoriel un choix moins optimal qui introduit deux variables internes supplémentaires. L'origine de ces variables provient du fait qu'il est possible de décomposer la vitesse d'écoulement viscoplastique suivant deux directions orthogonales dans l'espace des

contraintes : notons respectivement $\dot{p}_v$ et $\dot{p}_d$ la trace de $\underline{\epsilon}^{\text{vis}}$ et la projection de $\underline{\epsilon}^{\text{vis}}$ suivant $\underline{n}$. Nous avons :

$$\dot{p}_v = \frac{\partial \phi}{\partial s_{eq}} \frac{\partial s_{eq}}{\partial p}$$
$$\dot{p}_d = \frac{\partial \phi}{\partial s_{eq}} \frac{\partial s_{eq}}{\partial \sigma_{\text{eq}}}$$

Ces équations introduisent les deux variables internes recherchées :

— la variable p_v qui représente le changement de volume dû à l'écoulement viscoplastique (liés à la croissance ou la diminution de la porosité) ;

— la variables p_d qui représente la déformation viscoplastique déviatorique cumulée.

Traitement de la porosité En utilisant la variable p_v, l'équation d'évolution de la porosité est donnée par :

$$\dot{f} = (1 - f)\, \dot{p}_v$$

Cette relation explicite entre p_v et f va nous permettre d'exclure la porosité du système implicite. Dans `mfront`, ce type de variable interne est qualifiée d'auxiliaire.

4.3.3 Écriture du système implicite

Connaissant les valeurs des variables internes à l'instant t, nous cherchons à connaître leurs valeurs en fin de pas de temps, à l'instant $t + \Delta\, t$.

Quelques notations Dans la suite, pour chacune des variables internes x, nous noterons :

1. Δx son incrément sur le pas de temps ;
2. $x|_t$ sa valeur en début de pas de temps ;
3. $x|_{t+\theta\,\Delta\,t} = x|_t + \theta\,\Delta\, x$ l'estimation de sa valeur au temps intermédiaire $t + \theta\,\Delta\, t$. Nous parlerons souvent, de manière abusive, de la valeur en milieu de pas de temps de la variable x (en pratique, θ est souvent pris égal à $0,5$, sauf pour le traitement des lois plastiques) ;
4. $x|_{t+\Delta\,t} = x|_t + \Delta\, x$ sa valeur en fin de pas de temps ;

Équation relative à la déformation élastique La partition des déformations s'écrit, sous forme incrémentale :

$$\Delta\underline{\epsilon}^{\text{el}} + \Delta\underline{\epsilon}^{\text{vis}} - \Delta\underline{\epsilon}^{\text{to}} = 0$$

En introduisant les incréments des variables p_v et p_d, nous obtenons l'équation $f_{\underline{\epsilon}^{\text{el}}}$ associée à la déformation élastique :

$$f_{\underline{\epsilon}^{\text{el}}} = \Delta\underline{\epsilon}^{\text{el}} + \frac{1}{3}\,\Delta\, p_v\, \underline{I} + \Delta\, p_d\, \underline{n}|_{t+\theta\,\Delta\,t} - \Delta\underline{\epsilon}^{\text{to}}$$

où $\underline{n}|_{t+\theta\,\Delta\,t}$ est l'évaluation du tenseur normal à l'instant $t + \theta\Delta\, t$.

Équation relative au changement de volume viscoplastique L'équation f_{p_v} associée à la variable p_v est :

$$f_{p_v} = \Delta\, p_V - \Delta\, t\, \left.\frac{\partial \phi}{\partial s}\right|_{t+\theta\,\Delta\,t} \left.\frac{\partial s}{\partial p}\right|_{t+\theta\,\Delta\,t}$$

Équation relative à la déformation viscoplastique cumulée L'équation f_{p_d} associée à la variable p_d est :

$$f_{p_d} = \Delta\, p_d - \Delta\, t \left.\frac{\partial \phi}{\partial s}\right|_{t+\theta\,\Delta\, t} \left.\frac{\partial s}{\partial \sigma_{\text{eq}}}\right|_{t+\theta\,\Delta\, t}$$

Mise à jour de la porosité La porosité est mise en jour au cours des itérations de NEWTON par la relation :

$$f|_{t+\theta\,\Delta\, t} = f|_t + \frac{(1-f)\ \theta\,\Delta\, p_v}{1+\theta\,\Delta\, p_v}$$

4.3.4 Implantation

Les équations définissant $f_{\underline{\underline{\varepsilon}}^{\text{el}}}$ f_{p_v} et f_{p_d} forment un système de trois équations pour trois inconnues : elles sont donc suffisantes pour déterminer les incréments des variables internes sur un pas de temps. Reste donc à réaliser l'implantation de la loi. Nous commençons par utiliser un algorithme de NEWTON-RAPHSON dont la jacobienne est calculée numériquement.

```
1  @Parser       Implicit;
2  @Behaviour    EllipticCreep;
3  @Author       Maxime Salvo / Thomas Helfer;
4  @Date         9 Octobre 2013;
5
6  @Includes{
7  #include"TFEL/Material/Lame.hxx"
8  }
9
10 @Algorithm NewtonRaphson_NumericalJacobian;
11
12 // equivalent hydrostatic viscoplastic strain
13 @StateVariable real pv;
14 pv.setEntryName("HydrostaticEquivalentViscoplasticStrain");
15 // equivalent deviatoric viscoplastic strain
16 @StateVariable real pd;
17 pd.setEntryName("DeviatoricEquivalentViscoplasticStrain");
18 // porosity
19 @AuxiliaryStateVariable real f;
20 f.setGlossaryName("Porosity");
21
22 @Parameter A = 8.e-67;
23 A.setEntryName("CreepCoefficient");
24 @Parameter E = 8.2;
25 E.setEntryName("CreepExponent");
26 @Parameter CAf = 1;
27 CAf.setEntryName("CAf");
28
29 // Coefficient de Poisson
30 @LocalVariable real nu;
31 // Premier coefficient de Lame
32 @LocalVariable stress lambda;
33 // Second coefficient de Lame
34 @LocalVariable stress mu;
35 // Porosity at the beginning of the time step
36 @LocalVariable real f_t;
37
38 @MaterialLaw "UO2_YoungModulus_Fink1981.mfront";
39 @MaterialLaw "UO2_PoissonRatio_Fink1981.mfront";
40
41 @ComputeThermalExpansion "UO2_ThermalExpansion_MATPRO.mfront";
```

FIGURE 24 : Première implantation de la loi d'écoulement viscoplastique isotrope incompressible, première partie : déclaration des variables.

La partie déclarative Un fichier `mfront` commence généralement par une partie déclarative reprise en figure 24 :

- la ligne 1 précise que nous allons utiliser une méthode d'intégration implicite ;
- la ligne 2 donne le nom de la loi ;
- les lignes 3 et 4 renseignent les noms des auteurs et la date ;
- les lignes 6 à 7 demandent l'inclusion d'un fichier d'entête de la bibliothèque `TFEL` permettant de calculer les coefficients de LAMÉ et la matrice d'élasticité à partir de ces coefficients ;
- la ligne 10 précise que l'on utilise un algorithme de NEWTON avec jacobienne numérique ;
- les lignes 12 à 20 déclarent les variables internes et leur associent des noms pour l'extérieur. Ces noms de glossaire sont importants pour une adhérence aux applications de la plate-forme `pleiades` et pour `mtest` ;
- les lignes 22 à 27 définissent les paramètres de la loi et les noms à utiliser pour les modifier depuis `mtest`, `Code_Aster` ou les applications de la plate-forme `pleiades` ;

— les lignes 26 à 36 définissent quelques variables locales qui seront utilisables dans l'ensemble des blocs de code. Notons que pour traiter l'évolution de la porosité, nous introduisons une variable interne représentant sa valeur au cours du pas : `mfront` ne définit pas d'incrément pour les variables auxiliaires ;
— les lignes 38 et 39 indiquent dans quels fichiers sont implantés le module d'YOUNG et le coefficient de POISSON ;
— la ligne 41 demande le calcul de la dilatation thermique.

Remarque sur le type des variables Certaines variables sont déclarées de type `stress` et d'autres de type `real`. Il s'agit dans les deux cas d'un alias vers un type de nombre réel (double précision dans toutes les interfaces considérées jusqu'à présent). Le choix d'utiliser le type `stress` est uniquement informatif et vise à rendre le code plus explicite.

```
43 @InitLocalVariables{
44   /* Initialize Poisson coefficient */
45   nu  = UO2_PoissonRatio_Fink1981();
46   /* Porosity at the beginning of the time step */
47   f_t = f;
48 } // end of @InitLocalVars
```

FIGURE 25 : Première implantation de la loi d'écoulement viscoplastique isotrope incompressible, seconde partie : initialisation des variables locales.

Initialisation des variables locales La figure 25 montre comment initialiser les variables locales de la loi. Le coefficient de POISSON étant constant, nous l'évaluons un fois pour l'ensemble de l'intégration. Nous y sauvegardons également la valeur de la porosité en début de pas.

```
50 @ComputeStress{
51   using namespace tfel::material::lame;
52   // porosity at the intermediate time
53   f = f_t + theta*(1-f_t)/(1+theta*dpv)*dpv;
54   // evaluate young modulus to take porosity variation into account
55   const stress young = UO2_YoungModulus_Fink1981(T,min(max(f,real(0)),real(1)));
56   lambda = computeLambda(young,nu);
57   mu = computeMu(young,nu);
58   // Hooke law
59   sig = lambda*trace(eel)*Stensor::Id()+2*mu*eel;
60 } // end of @ComputeStresss
61
62 @ComputeFinalStress{
63   using namespace tfel::material::lame;
64   // porosity at the end of the time step
65   f = f_t + (1-f_t)/(1+theta*dpv)*dpv;
66   // evaluate young modulus to take porosity variation into account
67   const stress young = UO2_YoungModulus_Fink1981(T,min(max(f,real(0)),real(1)));
68   lambda = computeLambda(young,nu);
69   mu     = computeMu(young,nu);
70   // Hooke law
71   sig    = lambda*trace(eel)*Stensor::Id()+2*mu*eel;
```

FIGURE 26 : Première implantation de la loi d'écoulement viscoplastique isotrope incompressible, troisième partie : calcul des contraintes en milieu de pas et en fin de pas.

Évaluation des contraintes en milieu et fin de pas de temps La figure 26 montre comment sont évaluées les contraintes en milieu de pas et en fin de pas. Nous tombons ici sur une spécificité de l'implantation proposée.

En effet, la plupart des lois peuvent se contenter de ne définir que le bloc `@ComputeStress` : `mfront` évalue dans ce cas les différentes variables à l'instant considéré (milieu de pas ou fin de pas).

Ce n'est pas possible ici du fait de l'utilisation de la porosité comme variable auxiliaire : celle-ci doit être mise à jour explicitement avant l'évaluation des propriétés élastiques. Pour bien comprendre ce point, on peut se référer à la figure 27 qui montre l'ordre d'appel des différents blocs de code définis par l'utilisateur.

Le bloc de code `@ComputeStress` sera utilisé pour évaluer les contraintes en milieu de pas et le bloc de code `@ComputeFinalStress` en fin de pas de temps.

Pour se convaincre que les contraintes sont évaluées correctement, il peut être intéressant de regarder quel est le code généré par `mfront` pour ces deux blocs (figure 28).

La figure 29 constitue le cœur de la résolution.

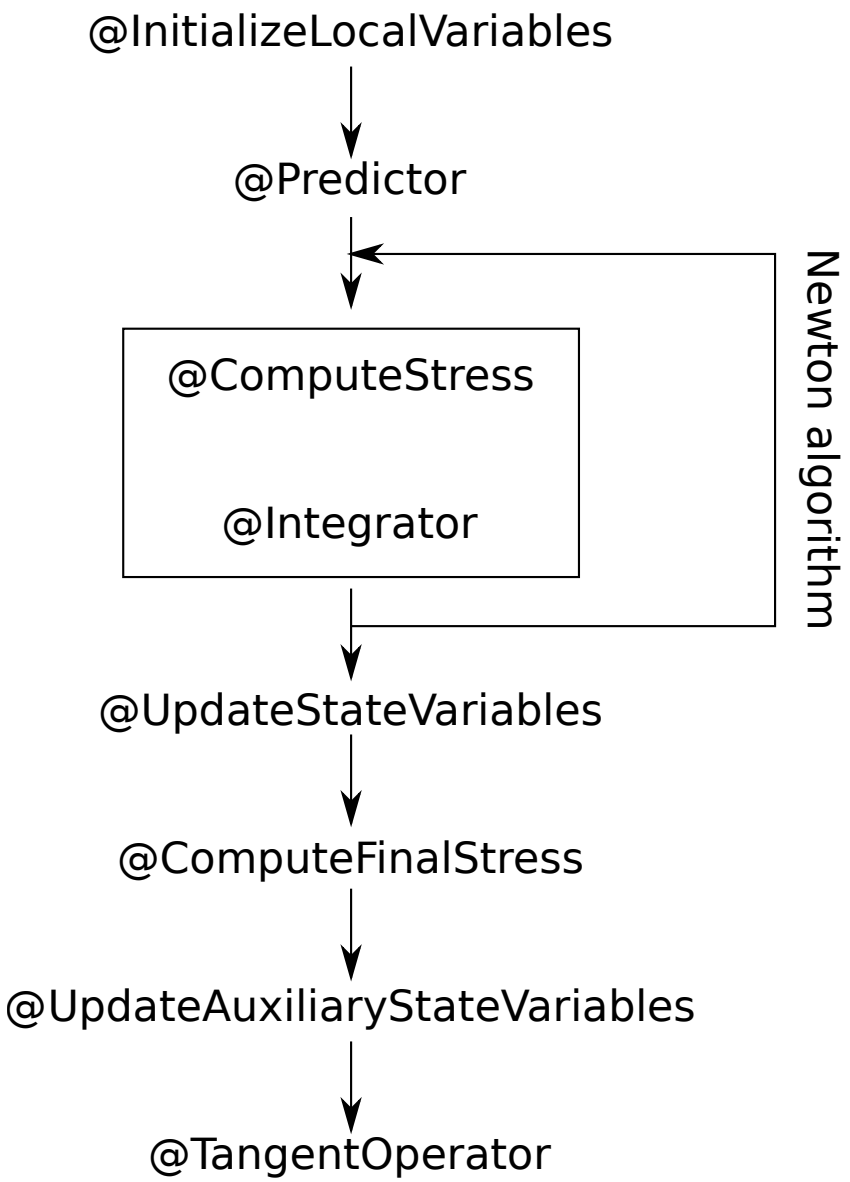


FIGURE 27 : Ordre d'évaluation des différents blocs de code fournis par l'utilisateur dans le cas d'une intégration implicite.

```
304
305       void computeStress(void) {
306         using namespace std;
307         using namespace tfel::math;
308         using mfront::UO2_PoissonRatio_Fink1981;
309         using mfront::UO2_PoissonRatio_Fink1981_checkBounds;
310         using mfront::UO2_ThermalExpansion;
311         using mfront::UO2_ThermalExpansion_checkBounds;
312         using mfront::UO2_YoungModulus_Fink1981;
313         using mfront::UO2_YoungModulus_Fink1981_checkBounds;
314         using std::vector;
315 #line 51 "EllipticCreep-1.mfront"
316         using namespace tfel::material::lame;
317 #line 53 "EllipticCreep-1.mfront"
318         this->f = this->f_t + this->theta * (1 - this->f_t) /
319                                       (1 + this->theta * this->dpv) * this->dpv;
320 #line 55 "EllipticCreep-1.mfront"
321         const stress young =
322             UO2_YoungModulus_Fink1981((this->T + (this->theta) * (this->dT)),
323                                       min(max(this->f, real(0)), real(1)));
324 #line 56 "EllipticCreep-1.mfront"
325         this->lambda = computeLambda(young, this->nu);
326 #line 57 "EllipticCreep-1.mfront"
327         this->mu = computeMu(young, this->nu);
328 #line 59 "EllipticCreep-1.mfront"
329         this->sig = this->lambda *
330                     trace((this->eel + (this->theta) * (this->deel))) *
331                     Stensor::Id() +
332                   2 * this->mu * (this->eel + (this->theta) * (this->deel));
333       }  // end of EllipticCreep::computeStress
334
335       void computeFinalStress(void) {
336         using namespace std;
337         using namespace tfel::math;
338         using mfront::UO2_PoissonRatio_Fink1981;
339         using mfront::UO2_PoissonRatio_Fink1981_checkBounds;
340         using mfront::UO2_ThermalExpansion;
341         using mfront::UO2_ThermalExpansion_checkBounds;
342         using mfront::UO2_YoungModulus_Fink1981;
343         using mfront::UO2_YoungModulus_Fink1981_checkBounds;
344         using std::vector;
345 #line 63 "EllipticCreep-1.mfront"
346         using namespace tfel::material::lame;
347 #line 65 "EllipticCreep-1.mfront"
348         this->f = this->f_t +
349                   (1 - this->f_t) / (1 + this->theta * this->dpv) * this->dpv;
350 #line 67 "EllipticCreep-1.mfront"
351         const stress young = UO2_YoungModulus_Fink1981(
352             (this->T + this->dT), min(max(this->f, real(0)), real(1)));
```

FIGURE 28 : Première implantation de la loi d'écoulement viscoplastique isotrope incompressible, code généré par `mfront` pour l'évaluation des contraintes en milieu et en fin de pas.

```
74  @Integrator{
75    // hydrostatic pressure
76    const stress pr  = trace(sig)/real(3);
77    // Von Mises stress
78    const stress seq = sigmaeq(sig);
79    // equivalent stress
80    const real Af = pow(E*(pow(f,-1/E)-1),-2*E/(E+1));
81    const real Bf = (1+2*f/real(3))*pow(1-f,-2*E/(E+1));
82    const stress s  = sqrt(Af*pr*pr+Bf*seq*seq);
83    if(s>1.e-8*mu){
84      // normal
85      real inv_seq(0);
86      Stensor n(real(0.));
87      if(seq > 1.e-8*mu){
88        inv_seq = 1/seq;
89        n       = 1.5*deviator(sig)*inv_seq;
90      }
91      const real ds_dpr        =  Af*pr/s;
92      const real ds_dseq       =  Bf*seq/s;
93      const real dphi_ds       =  A*pow(s,E);
94      const real dphi_dp       =  dphi_ds*ds_dpr;
95      const real dphi_dseq     =  dphi_ds*ds_dseq;
96      // hydrostatic part
97      const real K = lambda+(2*mu)/3;
98      fpv        -= dphi_dp*dt;
99      // deviatoric part
100     fpd        -= dphi_dseq*dt;
101     // elasticity
102     feel       += (dpv/3)*Stensor::Id()+dpd*n;
103   }
104   feel -= deto;
105 } // end of @Integrator
```

FIGURE 29 : Première implantation de la loi d'écoulement viscoplastique isotrope incompressible, quatrième partie : système implicite.

Le code contenu dans le bloc `@Integrator` est appelé après l'évaluation des contraintes en milieu de pas. La variable f contient l'estimation de la porosité en milieu de pas de temps et la variable sig l'estimation du tenseur des contraintes.

Nous commençons donc par évaluer la pression p, la contrainte équivalente au sens de VON MISES σ_{eq} et la contrainte équivalente s en milieu de pas à l'aide de la contrainte en milieu de pas.

Dans le cas où les contraintes ne sont pas trop faibles, nous définissons les équations associées aux variables viscoplastiques p_v et p_d. Nous utilisons ici les conventions implicites de `mfront` : les variables f_x sont initialisées avant chaque appel à $\Delta\, x$. Ainsi, si les contraintes sont trop faibles, les équations associées aux variables p_v et p_d seront :

$$\begin{aligned} f_{p_v} &= \Delta\, p_v \quad = 0 \\ f_{p_d} &= \Delta\, p_d \quad = 0 \end{aligned}$$

qui se résolvent trivialement en :

$$\Delta\, p_v = \Delta\, p_d = 0$$

4.4 PREMIERS TESTS

4.4.1 Compilation

La première étape des tests est la compilation de la loi. Nous utilisons l'interface `castem` utilisée par le code `Cast3M` :

```
$ mfront --obuild --interface=castem EllipticCreep.mfront
Treating target : all
UO2_ThermalExpansion-mfront.cxx: In function 'double mfront::UO2_ThermalExpansion()':
UO2_ThermalExpansion-mfront.cxx:21:22: warning: unused variable 'ReferenceTemperature'
[-Wunused-variable]
The following libraries have been build :
- libCastemUO2.so
- libMFrontMaterialLaw.so
- libCastemBehaviour.so
```

Le message d'avertissement relatif à la variable `ReferenceTemperature` est sans importance.

Analyse des messages d'erreurs La phase de compilation peut montrer des problèmes de syntaxe. Par exemple, remplaçons la ligne 59 de la figure 26 par celle-ci, contenant une faute de frappe (`lamdba` au lieu de `lambda`) :

```
59  sig = lamdba*trace(eel)*Stensor::Id()+2*mu*eel;
```

Dans ce cas, le compilateur GNU g++ [FSF 14] affichera le message d'erreur suivant :

```
In file included from EllipticCreep.cxx:13:0: EllipticCreep-1.mfront: In member
function 'void tfel::material::EllipticCreep<hypothesis, Type,
false>::computeStress()': EllipticCreep-1.mfront:59:13: error: 'lamdba' was not
declared in this scope compilation terminated due to -Wfatal-errors. make: ***
[EllipticCreep.o] Erreur 1 terminate called after throwing an instance of
'std::runtime_error' what(): MFront::buildLibraries : libraries building went
wrong Abandon
```

La ligne importante ici est la troisième :

```
EllipticCreep-1.mfront:59:13: error: 'lamdba' was not declared in this scope
```

qui indique qu'en ligne 59 à la colonne 13 la variable `lamdba` n'est pas connue. Un effort particulier a été fait pour que le compilateur affiche l'erreur dans le fichier source `mfront` et non dans le fichier généré. Le code étant transformé par `mfront`, le numéro de colonne n'est pas fiable.

4.4.2 Essai de compression hydrostatique

Description Une des spécificités de la loi est d'être compressible. Pour vérifier cet aspect de la loi, nous considérons un essai de compression hydrostatique à pression imposée constante de $70\,MPa$ durant 1 heure. La porosité initiale est de $5\,\%$. La température est de $293,15\,K$.

```
@Author Helfer Thomas;
@Date 08/04/2014;
@Description{
  Hydrostatic compression test
  for the elliptic creep behaviour.
};

@MaximumNumberOfSubSteps 1;

@Behaviour<umat> 'src/libUmatBehaviour.so' 'umatellipticcreep';
@MaterialProperty<constant> 'YoungModulus' 195879447720;
@MaterialProperty<castem> 'PoissonRatio' 'src/libCastemUO2.so' 'UO2_PoissonRatio_Fink1981';

// initial value of the YoungModulus
@Real 'E0' 195879447720;
// Poisson Ratio
@Real 'nu' 0.316;
// bulk modulus
@Real 'K' 'E0/(1-2*nu)';

@Real 'p' -70.e6;
@ImposedStress 'SXX' 'p';
@ImposedStress 'SYY' 'p';
@ImposedStress 'SZZ' 'p';
// Initial value of the elastic strain
@Real 'EELXX0' 'p/K';
@InternalStateVariable 'ElasticStrain' {'EELXX0','EELXX0','EELXX0',0.,0.,0.};
// Initial value of the total strain
@Strain {'EELXX0','EELXX0','EELXX0',0.,0.,0.};
// Initial value of the stresses
@Stress {'p','p','p',0.,0.,0.};

@InternalStateVariable 'Porosity' 0.05;

@ExternalStateVariable 'Temperature' 293.15;

@Times {0.,3600 in 10};
```

FIGURE 30 : Modélisation d'un essai de compression hydrostatique avec `mtest`.

Mise en données La figure 30 présente la mise en donnée utilisée. La principale difficulté est ici la définition de l'état initial (en contraintes, déformations totales et variables internes).

Solution approchée Afin de vérifier la solution obtenue, il est intéressant de disposer d'une solution analytique. Il s'avère que même dans ce cas très simple, les calculs deviennent rapidement rédhibitoires. Nous considérons donc une solution approchée considérant le coefficient $A\left(f\right)$ constant. La contrainte équivalente s est alors également constante et la variable p_v est linéaire en temps :

$$p_v\left(t\right) = \left.\frac{\partial \phi}{\partial s}\right|_{t=0} \left.\frac{\partial s}{\partial p}\right|_{t=0} t$$

Comparaison La figure 31 compare la solution `mtest` à la solution approchée. Les deux solutions coïncident bien au début de l'essai. Les deux solutions s'écartent en fin d'essai car l'approximation faite dans la solution approchée cesse d'être satisfaisante.

Convergence Il est intéressant d'étudier la convergence de l'algorithme d'équilibre de `mtest`. Celui-ci est représenté en figure 32 pour le dernier pas de temps. Il faut environ 50 itérations pour que la norme du résidu soit inférieure à la valeur du critère utilisé par `mtest`. La vitesse de convergence est visiblement linéaire. Cela est dû au fait que `mtest` utilise un opérateur tangent élastique (c'est le choix par défaut pour les lois compilées avec l'interface `castem`).

Accélération de convergence Il est possible d'activer dans `mtest` l'algorithme d'accélération utilisée par les procédures de résolution de `Cast3M` [Verpeaux 14]. Il s'agit d'une généralisation de la méthode des sécantes, la plus simple à

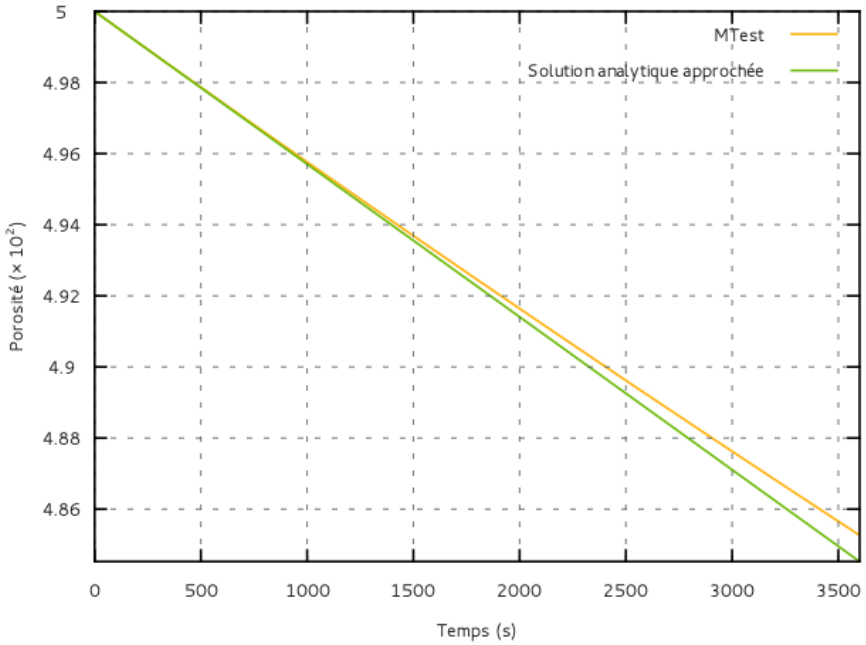


FIGURE 31 : Évolution de la porosité au cours de l'essai de compression hydrostatique.

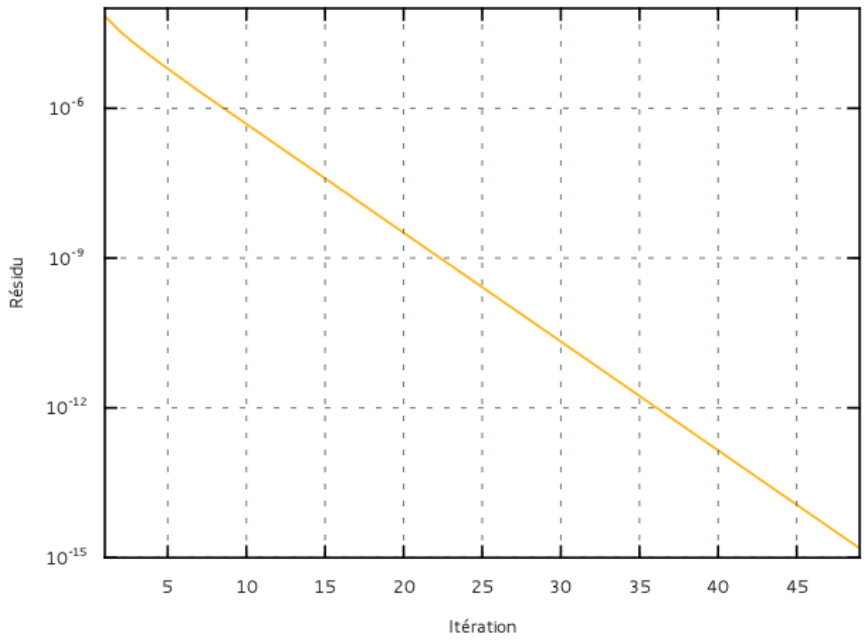


FIGURE 32 : Convergence vers l'équilibre mécanique.

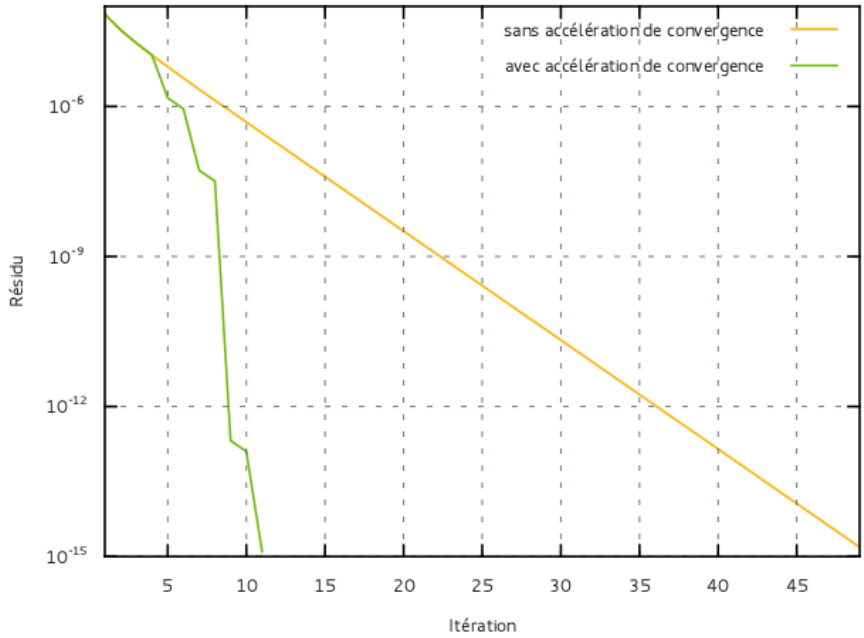


FIGURE 33 : Convergence vers l'équilibre mécanique avec l'algorithme d'accélération de `Cast3M`.

mettre en œuvre et la moins coûteuse (à notre connaissance). Pour cela, il suffit de rajouter la ligne suivante dans le fichier de donnée précédent :

```
@UseCastemAccelerationAlgorithm true;
```

La figure 33 compare les vitesses de convergence avec et sans l'utilisation de l'accélération de convergence. L'effet est net : avec l'accélération de convergence, il ne faut que 11 itérations pour atteindre la précision requise (en fait, dans cet exemple, le résidu est indiqué nul à la précision machine absolue près ($\approx 2,3\,10^{-308}$).

Le mode `debug` Il est également intéressant de regarder comment converge localement l'intégration de la loi. Pour cela, `mfront` dispose d'une option `--debug`. Recompilons la loi :

```
$ mfront --obuild --debug --interface=castem EllipticCreep.mfront
```

Lors de l'exécution du cas test, la loi affiche certains détails sur sa convergence locale :

```
EllipticCreep::integrate() : beginning of resolution
EllipticCreep::integrate() : iteration 1 : 2.14848e-05
EllipticCreep::integrate() : iteration 2 : 1.48821e-10
EllipticCreep::integrate() : convergence after 2 iterations
```

Cet exemple montre que l'algorithme de NEWTON local converge en 2 itérations (la précision par défaut est de 10^{-8}).

Choix des paramètres numériques Nous allons maintenant nous intéresser à deux paramètres numériques :

— la valeur du critère de convergence ε ;

— la valeur de la perturbation ε_p pour le calcul de la jacobienne numérique.

Pour comprendre ce que représente ce dernier paramètre, il faut rappeler comment la valeur de la jacobienne est approchée :

$$J(i,j) = \frac{\partial f_i}{\partial \Delta x_j} \approx \frac{f_i(x_j + \varepsilon_p) - f_i(x_j - \varepsilon_p)}{2\,\varepsilon_p}$$

Par défaut, la valeur du critère de convergence ε est de 10^{-8}. Il est intéressant de remarquer qu'elle est plus lâche que la précision utilisée pour l'équilibre global.

Dans la suite, nous verrons qu'il est nécessaire de baisser ce seuil pour obtenir une matrice tangente cohérente de bonne qualité, typiquement vers 10^{-11}.

La valeur par défaut du second seuil, ε_p, est dix fois inférieure à celle du critère de convergence. Dans notre exemple, nous avons donc une valeur de 10^{-9}, ce qui est une valeur acceptable dans la plupart des cas. Précisons que cette valeur est fixée à la compilation et n'est pas affectée par une modification de la valeur de ε à l'exécution.

Ces deux valeurs peuvent être modifiées par les paramètres `epsilon` et `numerical_jacobian_epsilon`. Il est possible de modifier ces valeurs depuis `mtest` à l'aide du mot clé `@Parameter` :

```
@Parameter 'epsilon' 1.e-13;
@Parameter 'numerical_jacobian_epsilon' 1.e-9;
```

Avec un seuil à 10^{-13} et une perturbation de 10^{-9}, nous obtenons la sortie suivante :

```
EllipticCreep::integrate() : beginning of resolution
EllipticCreep::integrate() : iteration 1 : 2.14849e-05
EllipticCreep::integrate() : iteration 2 : 1.48417e-10
EllipticCreep::integrate() : iteration 3 : 9.80423e-16
EllipticCreep::integrate() : convergence after 3 iterations
```

Avec un seuil à 10^{-13} et une perturbation de 10^{-18}, nous pouvons voir deux effets. En premier lieu, la qualité de la jacobienne numérique est moindre, ce qui augmente le nombre d'itération dans l'algorithme local.

```
EllipticCreep::integrate() : beginning of resolution
EllipticCreep::integrate() : iteration 1 : 2.14849e-05
EllipticCreep::integrate() : iteration 2 : 4.17309e-08
EllipticCreep::integrate() : iteration 3 : 8.06865e-10
EllipticCreep::integrate() : iteration 4 : 6.27906e-11
EllipticCreep::integrate() : iteration 5 : 1.30492e-12
EllipticCreep::integrate() : iteration 6 : 1.78654e-14
EllipticCreep::integrate() : convergence after 6 iterations
```

Dans ce cas, on note que la valeur finale du résidu est moins bonne que dans le cas précédent. Cela a un effet très fort sur la convergence globale, qui nécessite dans ce cas 53 itérations (avec une précision de 10^{-12} le calcul ne converge même pas). Ceci est dû au critère très serré utilisé par `mtest` sur la valeur de contraintes. Qualitativement, nous pouvons dire que la réponse de la loi de comportement est moins « stable » que précédemment, malgré une valeur plus basse du critère de convergence.

La diminution de la qualité de l'approximation de la dérivée avec la valeur de la perturbation est un phénomène numérique connu [Fortin 01].

Nous avons présenté ici un cas favorable où il faut donner une valeur très basse à ε_p pour voir un effet notable. Pour d'autres lois, des effets de ce type peuvent se faire sentir dès 10^{-12}. Nous recommandons donc de toujours explicitement préciser une valeur de l'ordre de 10^{-9} à l'aide du mot clé `@PertubationValueForNumericalJacobianComputation` :

```
@Algorithm NewtonRaphson_NumericalJacobian;
@PerturbationValueForNumericalJacobianComputation 1e-9;
```

5 CONCLUSIONS

Cette note a pour but de guider un utilisateur de `mfront` à développer une loi de comportement, afin de l'utiliser en premier lieu pour des tests unitaires sur point matériel, à l'aide de `mtest`, ce qui permet d'aller jusqu'à l'identification des propriétés matériau à l'aide du logiciel d'acquisition `Adao` développé dans la plate-forme `Salomé` [Argaud 14].

Ce comportement peut ensuite être utilisé avec les codes de calcul par éléments finis interfacés (CAST3M, `Code_Aster`, Cyrano, etc..).

De plus, une réflexion est en cours pour que certaines lois de comportement disponibles dans `Code_Aster` soient intégrées via `mfront`. Le document [Proix 14] établit une liste de lois de comportement candidates à ce remplacement, une fois que la diffusion libre de `mfront` avec `Code_Aster` sera effective.

Pour revenir à ce tutoriel, une bonne façon d'utiliser cette note consiste à expérimenter soi-même le développement d'une loi de comportement de son choix. La plupart des lois simples, élastiques, élasto(-visco-)plastiques peuvent être développées facilement, a priori en moins d'une demi-journée, par exemple avec matrice jacobienne numérique.

Cette note est susceptible d'évoluer en fonction des retours des utilisateurs, afin de préciser certains éléments qui ne seraient pas assez détaillés.

Ce tutoriel met en évidence plusieurs avantages de `mfront` :

- la puissance de ce logiciel pour développer facilement une loi de comportement, avec une syntaxe proche de la formulation « sur le papier » ;
- sa grande souplesse vis-à-vis de l'enrichissement des lois de comportement ;
- les options de vérification proposées ;
- et la robustesse et les performances de l'intégation obtenue.

Pour aller plus loin dans la compréhension du fonctionnement et des possibilités de `mfront`, il est conseillé de consulter la documentation de `mfront` [Helfer 13b, Helfer 13a], et celle de `mtest` [Helfer 14b].

Pour utiliser `mfront` avec `Code_Aster`, on pourra se reporter à la documentation [EDF 14d]. Des exemples peuvent être consultés dans les tests [EDF 14a, EDF 14b, EDF 14c].

6 RÉFÉRENCES

[Argaud 14] ARGAUD JEAN-PHILIPPE. *Documentation d'utilisation, dans la plateforme SALOME 7.3, du module ADAO "Assimilation de Données et Aide à l'Optimisation"*. Rapport technique H-I23-2014-00166-FR, EDF R&D Simulation en Neutronique, Technologies de l'Information et Calcul Scientifique, Analyse et Modèles Numériques, CLAMART, 2014.

[Besson 01] BESSON JACQUES, CAILLETAUD GEORGES et CHABOCHE JEAN-LOUIS. *Mécanique non linéaire des matériaux*. Hermès, Paris, 2001.

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

[EDF 14a] EDF . *MFRONT01 – Test de l'interface Code_Aster-MFRONT avec les lois de Chaboche*. Référence du Code Aster V1.03.126, EDF-R&D/AMA, Avril 2014.

[EDF 14b] EDF . *MFRONT02 – Test de l'interface Code_Aster-MFRONT avec des lois d'endommagement*. Référence du Code Aster V1.03.127, EDF-R&D/AMA, Avril 2014.

[EDF 14c] EDF . *MFRONT03 – Test de l'interface Code_Aster-MFRONT pour des lois avec orthotropie*. Référence du Code Aster V1.03.128, EDF-R&D/AMA, Avril 2014.

[EDF 14d] EDF . *Notice d'utilisation du couplage entre Code_Aster et les modules de lois de comportement Zmat et UMAT.* Référence du Code Aster U2.10.01, EDF-R&D/AMA, Avril 2014.

[EDF 14e] EDF . *Relations de comportement élasto-visco-plastique de Chaboche*. Référence du Code Aster R5.03.04-B, EDF-R&D/AMA, Avril 2014.

[Fink 81] FINK J.K., CHASANOV M.G. et LEIBOWITZ L. *Thermodynamic properties of Uranium dioxide*. Rapport technique ANL-CEN-RSD-80-3, Argonne National Laboratory, Avril 1981.

[Fortin 01] FORTIN ANDRÉ. *Analyse numérique pour ingénieurs*. Presses internationales Polytechnique, [Montréal], 2001.

[FSF 14] FSF (FREE SOFTWARE FOUNDATION). *GNU Compiler Collection (GCC)*, 2014.

[Helfer 05] HELFER THOMAS. *Implantation de lois de comportement viscoplastique de l'UO2 et du modèle de fissuration Mefisto couplé à la viscoplasticité dans Cast3M via l'interface UMAT*. Rapport technique 05-008 PLE 05-008, DEC/SESC/LLCC, Mars 2005.

[Helfer 11] HELFER THOMAS. *Licos 1.0 : notice d utilisation et d'installation, descriptif informatique*. Note technique 11 - 032, DEC/SESC/LSC, Novembre 2011.

[Helfer 13a] HELFER THOMAS et CASTELIER ÉTIENNE. *Le générateur de code mfront : présentation générale et application aux propriétés matériau et aux modèles*. Note technique 13-019, CEA DEN/DEC/SESC/LSC, Juin 2013.

[Helfer 13b] 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.

[Helfer 14a] HELFER THOMAS. *Prise en compte des hypothèses de contraintes planes dans les lois générées par le générateur de code MFront : application à Cyrano3*. Note technique 14-013, CEA DEN/DEC/-SESC/LSC, 2014.

[Helfer 14b] HELFER THOMAS et PROIX JEAN-MICHEL. *MTest : un outil de test unitaire de lois comportement mécanique*. Note technique 14-016, CEA DEN/DEC/SESC/LSC, 2014.

[Michel 92] MICHEL J.C. et SUQUET P. *The constitutive law of nonlinear viscous and porous materials*. Journal of the Mechanics and Physics of Solids, 1992, vol 40, n° 4, p 783 – 812.

[Monerie 06] MONERIE YANN et GATT JEAN-MARIE. *Overall viscoplastic behavior of non-irradiated porous nuclear ceramics*. Mechanics of Materials, Juillet 2006, vol 38, n° 7, p 608–619.

[Moreau 70] MOREAU J. J. *sur les lois de frottement, de plasticité et de viscosité*. Compte Rendu de l'Académie des Sciences série A, Septembre 1970, vol 271, p 608–611.

[Pellet 12] PELLET JACQUES. *Calcul de la déformation thermique*. Référence du Code Aster R4.08.01, EDF-R&D/AMA, Février 2012.

[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.

[Proix 14] PROIX J.M. *Quelles lois de comportement de Code-Aster sont transposables en Mfront ?* Compte-rendu de réunion CR-AMA-14-080-B, EDF/R&D/AMA/T64, 2014.

[Salome 14] SALOME . *The Open Source Integration Platform for Numerical Simulation*, 2014.

[Shamsaei 11] SHAMSAEI N., FATEMI A. et SOCIE D.F. *Multiaxial fatigue evaluation using discriminating strain paths*. International Journal of Fatigue, 2011, vol 33, p 597–609.

[Thouvenin 10] THOUVENIN GILLES, BARON DANIEL, LARGENTON NATHALIE, LARGENTON RODRIGUE et THEVENIN PHILIPPE. *EDF CYRANO3 code, recent innovations*. LWR Fuel Performance Meeting/TopFuel/-WRFPM. Orlando, Florida, USA. Septembre 2010.

[Verpeaux 14] VERPEAUX PIERRE. *Algorithmes et méthodes*, 2014.

ANNEXE A RAPPELS SUR LA DILATATION THERMIQUE

Expérimentalement, on mesure la variation de longueur d'un corps entre une température de référence T_α et une température finale T.

Si l'on note l_{T^α} et l_T les longueurs respectives du corps à ces deux températures, le coefficient de dilatation thermique linéique $\alpha\left(T\right)$ est défini par :

$$\frac{l_T - l_{T^\alpha}}{l_{T^\alpha}} = \alpha\left(T\right)\ \left(T - T^\alpha\right) \tag{2}$$

Dans le cas des petites déformations, l'équation (2) définit une déformation associée à la dilatation thermique :

$$\epsilon^{\text{th}}_{T^\alpha}\left(T\right) = \alpha\left(T\right)\ \left(T - T^\alpha\right)$$

Cette déformation prend comme état de référence la longueur du corps à la température l_{T^α}.

Lors d'un calcul thermo-mécanique, on suppose généralement que la température initiale T_i est supposée être celle à laquelle la géométrie est fournie.

Il est donc nécessaire de modifier la définition de la dilatation thermique pour que l'état de référence soit la géométrie initiale, de longueur l_i.

Pour cela, nous pouvons définir la déformation thermique $\epsilon^{\text{th}}_{T^i}\left(T\right)$ du corps à la température T comme la déformation qu'il aurait si aucune contrainte ne s'exerçait sur lui par la relation :

$$\epsilon^{\text{th}}_{T^i}\left(T\right) = \frac{l_T - l_{T^i}}{l_{T^i}}$$

En introduisant la longueur de référence l_{T^α}, nous obtenons :

$$\begin{aligned}\epsilon^{\text{th}}_{T^i}\left(T\right) &= \frac{l_{T^\alpha}}{l_{T^i}}\,\frac{l_T - l_{T^i}}{l_{T^\alpha}} = \frac{1}{1 + \dfrac{l_{T^i} - l_{T^\alpha}}{l_{T^\alpha}}}\ \left[\frac{l_T - l_{T^\alpha} + l_{T^\alpha} - l_{T^i}}{l_{T^\alpha}}\right] \\ &= \frac{1}{1 + \alpha\left(T^i\right)\left(T^i - T^\alpha\right)}\ \left[\frac{l_T - l_{T^\alpha}}{l_{T^\alpha}} - \frac{l_{T^i} - l_{T^\alpha}}{l_{T^\alpha}}\right]\end{aligned}$$

Nous obtenons finalement la relation :

$$\begin{aligned}\epsilon^{\text{th}}_{T^i}\left(T\right) &= \frac{1}{1 + \alpha\left(T^i\right)\left(T^i - T^\alpha\right)}\ \left[\alpha\left(T\right)\left(T - T^\alpha\right) - \alpha\left(T^i\right)\left(T^i - T^\alpha\right)\right] \\ &= \frac{1}{1 + \epsilon^{\text{th}}_{T^\alpha}\left(T^i\right)}\ \left[\epsilon^{\text{th}}_{T^\alpha}\left(T\right) - \epsilon^{\text{th}}_{T^\alpha}\left(T^i\right)\right]\end{aligned} \tag{3}$$

Dans la procédure `STAT_NON_LINE` de `Code_Aster` [Pellet 12] ou dans les différentes procédures de résolution de `Cast3M`, la relation précédente s'écrit de manière approchée, en négligeant le terme $\epsilon^{\text{th}}_{T^\alpha}\left(T^i\right)$ par rapport à 1 :

$$\epsilon^{\text{th}}_{T^i}\left(T\right) = \epsilon^{\text{th}}_{T^\alpha}\left(T\right) - \epsilon^{\text{th}}_{T^\alpha}\left(T^i\right)$$

Au final, pour pouvoir traiter la dilatation thermique, il est nécessaire d'avoir accès à :

- l'évolution du coefficient de dilatation thermique $\alpha\left(T\right)$ avec la température et la température de référence T_α ;
- la température initiale à laquelle la géométrie du corps a été mesurée ;