`mtest` : un outil de simulation du comportement mécanique d'un point matériel

T. Helfer, J.M. Proix[(a)], I Ramière

Juillet 2015

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

RÉSUMÉ

`mfront` peut générer des lois de comportement mécanique pour différents codes cibles : `Cast3M` (code aux éléments finis généraliste développé par le CEA utilisé comme composant mécanique dans `pleiades`), `Aster` (code aux éléments finis généraliste développé par ÉDF), `cyrano` (code combustible développé par ÉDF) [Helfer 13d]. Chacun de ces codes présente des spécificités que `mfront` doit prendre en compte pour des performances optimales.

Ce document présente un outil de simulation du comportement mécanique d'un point matériel nommé `mtest`. La motivation initiale du développement de `mtest` était de disposer d'un outil *autonome* permettant de maîtriser le développement de `mfront` : `mtest` peut se comporter comme chacun des différents codes, c'est à dire adopter les mêmes conventions ou utiliser les mêmes algorithmes numériques.

`mtest` permet de piloter le chargement mécanique d'un point matériel dont le comportement est décrit en petites ou en grandes transformations ou par un modèle de zones cohésives. Il est possible d'imposer certaines composantes des contraintes et/ou certaines composantes des déformations (ou le gradient de la déformation en grandes transformations, ou le saut de déplacement pour les modèles de zones cohésives) au cours d'un historique (de contraintes, de température, de fluence, etc.).

`mtest` permet de simuler de nombreux essais mécaniques simples (sans effet de structure) :

- essais de traction/compression de type fluage (charge constante), écrouissage (vitesse de déformation imposée) ou relaxation (déformation totale constante) et les essais de SATOH (pièce chauffée dont la dilatation est empêchée). Ces essais peuvent présenter des cycles ou des variations de chargement ;
- essais en pression interne sur tube présentant les mêmes variantes que les essais de traction/compression.

Du fait de sa restriction à un unique point matériel, `mtest` se distingue par ses temps d'exécution, généralement d'un à plusieurs ordres de grandeurs inférieur à ceux des codes mentionnés. La part relative prise par l'intégration de la loi de comportement est très significative, ce qui implique que toute régression des lois générées en termes de performances peut être automatiquement détectée.

Les résultats obtenus par `mtest` peuvent être comparés à des solutions analytiques ou à des résultats de référence contenus dans un fichier externe. Ces tests génèrent des fichiers qui peuvent être interprétés par l'automate `jenkins`, au centre de la politique d'intégration continue mise en place par la plate-forme `pleiades`. Plus de 600 cas tests unitaires ont été écrits à l'aide de `mtest`.

Au delà de l'aspect test unitaire, cet outil s'est aussi avéré très utile et apprécié des utilisateurs de `mfront` qui l'emploient pour :

- valider le développement d'une nouvelle loi de comportement ;
- recaler ses paramètres, notamment avec `ADAO`, un module `Salomé` pour l'Assimilation de Données et l'Aide à l'Optimisation ;
- tester son comportement numérique, en particulier lorsque l'algorithme d'intégration local échoue.

Ce document présente tout d'abord les principales fonctionnalités de cet outil. Son utilisation en ligne de commande et l'ensemble des commandes disponibles sont alors décrits. Nous détaillons ensuite quelques tests intégrés à la gestion de configuration de `mfront`, dont certains sont des contributions externes.

SOMMAIRE

1 INTRODUCTION

1.1 CONTEXTE

1.1.1 Rappel

mfront est un outil de générateur de code développé et maintenu au sein de la plate-forme pleiades. Il s'agit d'un outil essentiel dans la stratégie de mutualisation, de capitalisation et de pérennisation des connaissances matériau utilisées au sein des logiciels de simulation du DEC/SESC [Michel 09]. Son but est également de permettre à des utilisateurs non développeurs d'enrichir les applications de la plate-forme, en premier lieu l'application de conception licos : les fichiers d'entrée de mfront se focalisent sur le contenu physique, les détails informatiques et numériques étant gérés par mfront [Helfer 13c, Helfer 13d].

1.1.2 Une utilisation de mfront croissante en dehors de la plate-forme pleiades

Récemment, l'utilisation de mfront s'est accrue :
- au sein de la plate-forme pleiades, l'application cyrano, développée par ÉDF utilise des lois générées par mfront ;
- au sein du CEA, mfront est utilisé par le DER [d'Arrigo 12] et le DMN [Helfer 11, Milliard 14] dans le cadre de simulations Cast3M ;

Par ailleurs, mfront a été évalué par les partenaires industriels du CEA :
- AREVA pour ses activités combustible [Olagnon 13] ;
- ÉDF pour son code aux éléments finis généraliste Aster [Proix 13].

En dehors de la plate-forme pleiades, l'utilisation de mfront s'est pour l'instant essentiellement portée sur l'écriture de lois de comportement mécanique. Différentes interfaces ont été développées : elles permettent d'utiliser les lois de comportement mécanique écrites en mfront dans différents codes cibles tout en exploitant au mieux leurs caractéristiques propres (portant à la fois sur les conventions qu'ils utilisent, sur leurs capacités de modélisation ainsi que sur les algorithmes numériques dont ils disposent). Ces interfaces sont aujourd'hui au nombre de 3 :
- l'interface umat, dédiée au code Cast3M [Verpeaux 98, CEA 13] et le solveur TMFFT de la plate-forme pleiades [Castelier 09, Jérôme 10] ;
- l'interface aster, dédiée au code Aster [EDF 13b] ;
- l'interface cyrano, dédiée au code combustible cyrano [Thouvenin 10].

1.2 MOTIVATION INITIALE DU DÉVELOPPEMENT DE MTEST

Cette utilisation croissante de mfront risquait de complexifier sa maintenance, d'autant que les lois générées ne pouvaient être testées qu'avec l'aide des codes cibles tiers, dont ceux cités précédemment.

Pour éviter cet écueil, nous avons donc développé un outil permettant de simuler le comportement d'un point matériel, similaire en possibilités à ce qu'offre la commande SIMU_POINT_MAT d'Aster [EDF 13a] : il est possible de piloter le chargement mécanique d'un point matériel, en imposant certaines composantes des contraintes et/ou certaines composantes des déformations au cours d'un historique défini par l'utilisateur.

L'utilisateur peut simuler plusieurs essais mécaniques simples (au moins tant qu'ils n'impliquent pas d'effets de structure, striction notamment), par exemple :
- la plupart des essais multi-axés ;
- essais de traction/compression ;
- essais de cisaillement ;
- essais de fluage ;

— essais de relaxation ;
— essais de SATOH ;
— essais en pression interne sur tube.

Les résultats obtenus peuvent être comparés à des solutions analytiques ou à des solutions de référence. En sortie, deux fichiers sont créés :

— un fichier texte contenant l'évolution de l'ensemble des variables internes de la loi ;
— un fichier `xml` qui contient le résultat des différentes comparaisons. Ce fichier peut être interprété par l'automate d'intégration continue `jenkins`[1] utilisé pour le développement de `mfront`.

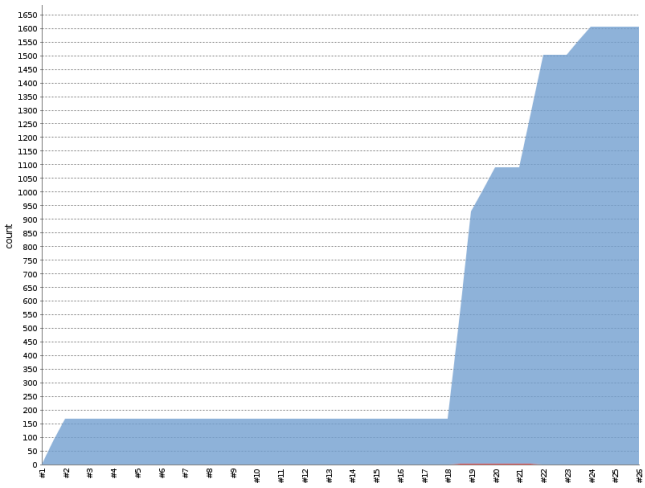

FIGURE 1 : Évolution du nombre de cas tests enregistrée par l'outil `jenkins`. Le nombre de cas tests unitaires est actuellement supérieur à 400. Les chiffres présentés par `jenkins` correspondent à la somme des exécutions sur plusieurs plate-formes.

Cet outil a permis de tester de nombreuses lois de comportement mécanique (plus de 60 actuellement) dans diverses situations avec l'ensemble des interfaces, ce qui conduit à plus de 400 tests unitaires. L'évolution du nombre de cas-tests peut être visualisée sur la figure 1. Notons qu'une part significative de ces cas tests est due à des contributions externes au projet `pleiades`.

1.3 UN OUTIL APPRÉCIÉ PAR LES UTILISATEURS DE MFRONT

1.3.1 Une utilisation croissante

Si `mtest` a été initialement développé pour des besoins internes au développement de `mfront`, `mtest` semble apprécié des utilisateurs de `mfront`, au sein du projet `pleiades` ou à l'extérieur. Il est utilisé :

— comme un outil de validation unitaire des lois de comportement ;
— comme un outil d'optimisation des lois de comportement[2] ;
— comme un outil de recalage des paramètres du comportement. `mtest` a notamment été couplé avec le code `Matlab` pour l'interprétation d'essais sur un acier EM10 [Milliard 14] et avec le module `ADAO` de la plate-forme `Salomé`.

1. http://jenkins-ci.org
2. Il est notamment possible de demander à une loi `mfront` de générer automatiquement un fichier `mtest` en cas de non convergence de l'intégration sur un pas de temps. Il est ainsi possible d'analyser les causes de divergence sans avoir à relancer un calcul éléments finis beaucoup plus coûteux.

1.3.2 Avantages

La principale qualité de `mtest` est sa rapidité, très nettement supérieure à celle d'un code généraliste. À titre de comparaison, un essai de traction uniaxiale de la base des cas tests est réalisé au moins 140 fois plus rapidement avec `mtest` qu'avec `Cast3M`[3]. L'intégralité des cas tests de `mfront` associés aux lois de comportement mécanique (plus de 400) s'exécutent en une trentaine de secondes sur un ordinateur de bureau.

Le second avantage, lié au premier, est le temps pris par l'intégration de la loi de comportement qui peut être noyé dans le bruit de fond pour des essais simples dans des codes généralistes : lors du test précédent, la loi de comportement représentait environ $25\,\%$ du temps de calcul[4] contre $0,02\,\%$ pour le temps `Cast3M`[5]. Dans la plupart des cas, le temps de calcul total, avec `mtest`, permet donc une appréciation de l'efficacité numérique de la loi de comportement. Il est donc possible :

— de comparer simplement différents algorithmes ;
— de vérifier que les évolutions de `mfront` ne conduisent pas à une dégradation des performances.

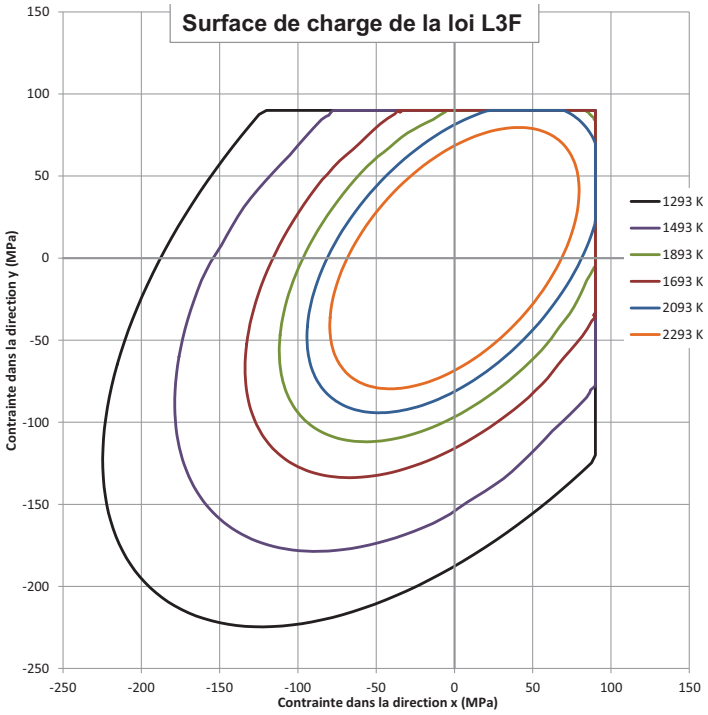

FIGURE 2 : Surface de charge d'un modèle de comportement viscoplastique du dioxyde d'uranium [Salvo 14].

Un troisième avantage est la possibilité d'utiliser `mtest` au travers d'une bibliothèque `C++` [Stroustrup 04] ou d'une bibliothèque `python` [Von Rossum 07]. Cette possibilité permet :

— d'intégrer simplement `mtest` dans des chaînes de calculs complexes, incluant le recalage de paramètres par rapport à des données expérimentales ;
— de permettre de réaliser rapidement des essais paramétriques. À titre d'exemple, la figure 2 représente la surface de charge du modèle de comportement viscoplastique du dioxyde d'uranium développé au cours de la thèse de Maxime Salvo [Salvo 14]. Pour obtenir cette surface de charge, il a été nécessaire de réaliser, pour chaque température et chaque direction de l'espace des contraintes un essai de

3. Nous avons retenu le temps le plus pénalisant mesuré par la commande `time` pour `mtest` qui est assez imprécis : cette mesure varie entre 4 microsecondes et 0 (!). Cette imprécision est partiellement due au fait que le chargement des librairies dynamiques sur lesquelles s'appuie `mtest` représente pratiquement 90% du temps d'exécution. Ces librairies peuvent être mises en cache lors d'un second appel.

4. Il s'agit de la part prise par l'intégration dans la fonction principale du programme : le temps de chargement des librairies mentionné plus haut n'est pas pris en compte.

5. La loi testée étant très efficace, il s'agit pratiquement d'une borne inférieure du temps passé dans l'intégration de la loi de comportement.

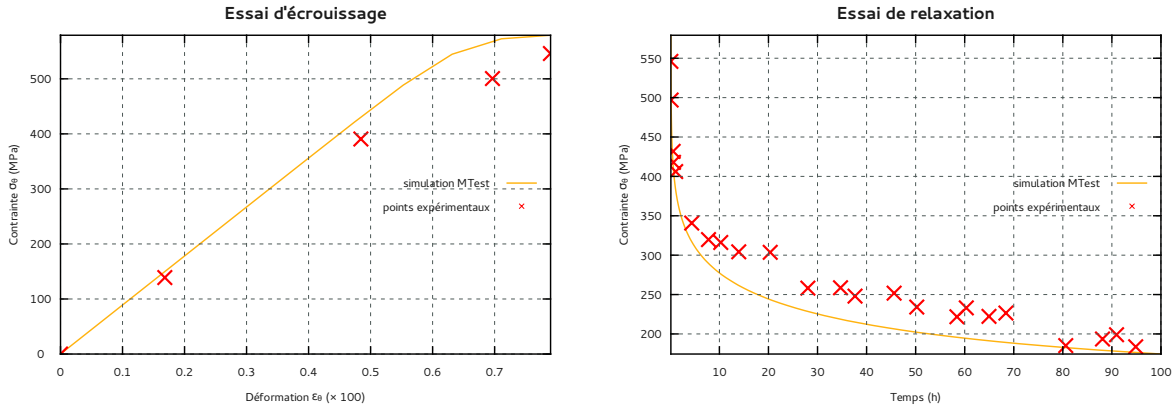

FIGURE 3 : Exemple de simulation avec `mtest` d'un essai d'écrouissage puis relaxation d'un tube en Zircaloy4 irradié. Comparaison des contraintes prédites par la loi standard de l'application `cyrano` (réimplantée avec `mfront`) aux mesures expérimentales.

traction, soit plusieurs milliers de simulations ;

— de simuler des essais relativement complexes tels qu'un tube en pression interne asservi de telle sorte que la vitesse de dilatation diamétrale soit tout d'abord constante (phase d'écrouissage durant laquelle la pression interne augmente) puis nulle (phase de relaxation durant laquelle la pression interne diminue). Un exemple d'une telle simulation, réalisée par l'équipe de développement du code `cyrano`, est représenté en figure 3.

Le dernier avantage est l'ergonomie : un essai peut être simulé avec une dizaine de lignes de mise en données, bien moins que ce qui est nécessaire avec un code tel que `Cast3M` (qui nécessite par ailleurs de définir un maillage).

PLAN DE CETTE NOTE

Nous présentons tout d'abord le principe de la simulation d'un point matériel et les algorithmes de résolution disponibles dans `mtest`.

Nous détaillons ensuite l'utilisation de `mtest` en ligne de commandes et les différents fichiers générés.

Dans un troisième temps, nous décrivons la structure d'un fichier d'entrée et donnons la liste de mots clés disponibles et leurs significations.

Enfin, nous donnons des exemples de tests utilisés pour assurer la non régression des lois de comportement mécanique générées par `mfront`.

2 SIMULATION D'UN POINT MATÉRIEL : PRINCIPE ET ASPECTS NUMÉRIQUES

`mtest` est un outil de simulation du comportement d'un point matériel. Il permet de tester des lois écrites en petites déformations, en grandes déformations et des modèles de zones cohésives.

Pour simplifier la présentation, nous avons restreint la présentation faite ici au cas des petites déformations. Les cas des lois écrites en grandes déformations ou des modèles de zones cohésives s'en déduisent sans difficulté majeure [6].

2.1 RAPPEL SUR LES LOIS DE COMPORTEMENT MÉCANIQUE ET SUR LEURS RÔLES DANS L'ÉQUILIBRE MÉCANIQUE D'UNE STRUCTURE

Comme nous l'avons décrit en introduction, la motivation initiale de `mtest` était de permettre de tester les différentes interfaces des lois de comportement sans recourir à des codes tiers. `mtest` doit donc permettre de tester tous les aspects liés à une loi de comportement. De ce fait, l'algorithme de résolution de `mtest`, restreint à un point matériel et décrit dans la section suivante, doit offrir la même richesse que les algorithmes des codes de calcul de structure. Il nous est donc apparu utile de rappeler comment se place la loi de comportement dans un calcul d'équilibre mécanique d'un calcul de structure. Cette approche nous permet de rappeler des notions importantes, telles que celles de matrice de raideur ou de matrice de prédiction.

Les lois de comportement mécanique décrites dans ce document permettent de calculer d'une part le tenseur des contraintes $\underline{\sigma}$ en un point d'une structure mécanique, et par ailleurs l'évolution microstructurelle du matériau, représentée par un jeu de variables internes noté de manière symbolique Y. Nous en faisons une présentation volontairement succincte, nous renvoyons à la documentation de `mfront` relative aux lois de comportement pour les détails [Helfer 13d].

Intégration locale Plus précisément, les lois de comportement mécanique travaillent de manière incrémentale. Sur un pas de temps dt entre deux instants t et $t + dt$, et connaissant :

- la valeur d'un certain nombre de propriétés matériau, calculées par le code appelant. Ces propriétés matériaux peuvent être de natures diverses (module d'YOUNG, exposant de la loi de NORTON, limite initiale du domaine d'élasticité, etc...) ;
- la valeur en début de pas et l'incrément sur le pas d'un certain nombre de variables dites externes, fournies par le code appelant. Dans `Aster`, ils correspondent aux variables de contrôle. Dans `Cast3M`, l'utilisateur est libre de définir différentes variables externes qui évoluent indépendamment de la résolution mécanique ;
- le tenseur des déformations totales du matériau à l'instant t, noté $\underline{\epsilon}^{to}|_t$;
- la valeur des variables internes à l'instant t, noté $Y|_t$;
- l'incrément du tenseur des déformations totales sur le pas de temps, noté $\Delta\,\underline{\epsilon}^{to}$;
- l'incrément de temps $\Delta\,t$.

Les lois de comportement mécanique déterminent le tenseur des contraintes et les variables internes en fin de pas de temps, notés respectivement $\underline{\sigma}|_{t+\Delta\,t}$ et $Y|_{t+\Delta\,t}$. Pour cela, elles sont généralement amenées à intégrer un système différentiel, et l'on parle généralement d'*intégration locale de la loi de comportement*.

Recherche de l'équilibre Dans la plupart des codes de mécanique (formulés en déplacements), l'appel à la loi de comportement s'inscrit dans un algorithme de recherche d'un incrément de déplacement tel que

6. Tout au plus pouvons nous signaler une difficulté inhérente aux lois écrites en grandes transformations. Ces lois relient le gradient de la déformation, qui a 9 composantes en $3D$ au tenseur de CAUCHY qui en a 6. Il y a donc une indétermination qu'il faut lever en imposant trois conditions cinématiques fixant trois composantes indépendantes du gradient de la transformation.

l'équilibre mécanique de la structure soit assuré en fin de pas de temps. Cet algorithme est dit global, par opposition à l'intégration locale de la loi de comportement.

L'algorithme global détermine en chaque point une estimation de l'incrément du tenseur des déformations totales $\Delta \underline{\epsilon}^{to}$ sur le pas de temps qui est donné à la loi de comportement mécanique. La réaction locale du matériau, le tenseur des contraintes $\underline{\sigma}$, est alors comparée aux chargements mécaniques subis par la structure. Si cette réaction locale n'assure pas l'équilibre de la structure, alors une nouvelle estimation est proposée et ce processus est répété jusqu'à convergence.

Matrices de raideur Ce processus utilise une variante plus ou moins sophistiquée de l'algorithme de NEWTON : à chaque itération, un problème linéaire est résolu, dont la matrice, dite de raideur, se construit à partir de matrices élémentaires déduites du comportement du matériau.

Différents choix de matrice de raideur (et autant de variantes de l'algorithme de résolution) sont donc possibles, citons :

- la matrice élastique ;
- la matrice élastique endommagée (dans le cas d'une loi décrivant un endommagement), dite matrice sécante ;
- la matrice tangente au comportement, définie dans une formulation en vitesse par la dérivée $\left.\frac{\partial \dot{\underline{\sigma}}}{\partial \dot{\underline{\epsilon}}^{to}}\right|_{t+\Delta t}$;
- la matrice tangente cohérente, définie par la dérivée $\frac{\partial \underline{\sigma}|_{t+\Delta t}}{\partial \Delta \underline{\epsilon}^{to}}$.

Initialisation de la recherche Certains codes aux éléments finis (`Aster` notamment), peuvent s'appuyer sur la loi de comportement pour démarrer de manière plus efficace la recherche de l'équilibre. Dans ce cas, la loi de comportement est appelée pour fournir une prédiction *a priori* de la matrice de raideur, sans effectuer l'intégration de la loi. Différentes prédictions de la matrice de raideur sont envisageables :

- la matrice élastique initiale ;
- la matrice élastique endommagée, dite matrice sécante ;
- la matrice tangente au comportement, définie dans une formulation en vitesse par la dérivée $\left.\frac{\partial \dot{\underline{\sigma}}}{\partial \dot{\underline{\epsilon}}^{to}}\right|_{t}$ calculée en début de pas de temps.

L'interface `umat` utilisée par le code aux éléments finis `Cast3M` présente une particularité qu'il peut être intéressant d'utiliser quand la loi ne fournit pas d'opérateur de prédiction : le code `Cast3M` exige que des propriétés élastiques fassent systématiquement partie de la liste de propriétés matériau et `mtest` peut s'en servir pour fournir un opérateur élastique par défaut[7, 8].

2.2 ALGORITHMES DE RÉSOLUTION

Pour une loi de comportement donnée, `mtest` calcule comment un point matériel réagit à un historique de chargement mécanique au cours duquel certaines composantes des contraintes et/ou des déformations peuvent être imposées. Ce paragraphe présente les différents algorithmes de résolution disponibles.

Équilibre mécanique en l'absence de déformations imposées En l'absence de déformations imposées, l'équilibre mécanique se traduit par l'égalité du tenseur des contraintes $\underline{\sigma}$, calculé par la loi de comportement

7. Notons que la loi peut ne pas utiliser les propriétés matériau qu'exige `Cast3M`. Dans ce cas, ces propriétés doivent être considérées comme des paramètres numériques permettant de construire une matrice de raideur effective servant à la recherche de l'équilibre mais n'influençant pas le résultat final.
8. Voir le mot clé `@PredictionPolicy`

mécanique $\underline{\sigma}$, et du tenseur des contraintes imposées $\underline{\sigma}^{\text{ext}}$:

$$\underline{\sigma}|_{t+\Delta t}\left(\Delta \underline{\epsilon}^{to}\right) = \underline{\sigma}^{\text{ext}}|_{t+\Delta t} \tag{1}$$

Pour chaque pas de temps, l'inconnue de ce problème est l'incrément des déformations totales $\Delta \underline{\epsilon}^{to}$.

Recherche de l'équilibre, algorithme de NEWTON La loi de comportement est généralement non linéaire, un algorithme de NEWTON est utilisé pour déterminer $\Delta \underline{\epsilon}^{to}$.

L'algorithme de NEWTON cherche à annuler une fonction $r\left(\Delta \underline{\epsilon}^{to}\right)$, nommée le résidu, définie par :

$$r\left(\Delta \underline{\epsilon}^{to}\right) = \underline{\sigma}|_{t+\Delta t}\left(\Delta \underline{\epsilon}^{to}\right) - \underline{\sigma}^{\text{ext}}|_{t+\Delta t} \tag{2}$$

Pour cela, l'algorithme de NEWTON procède de manière itérative. Soit $\Delta \underline{\epsilon}^{to}_n$ l'estimation de la solution obtenue à la n$^{\text{e}}$ itération, l'estimation suivante $\Delta \underline{\epsilon}^{to}_{n+1}$ est obtenue en linéarisant la fonction r à l'aide d'un développement limité à l'ordre 1 :

$$r\left(\Delta \underline{\epsilon}^{to}_{n+1}\right) \approx r\left(\Delta \underline{\epsilon}^{to}_{n}\right) + \frac{\partial\, \underline{\sigma}|_{t+\Delta t}}{\partial \Delta \underline{\epsilon}^{to}}.\left(\Delta \underline{\epsilon}^{to}_{n+1} - \Delta \underline{\epsilon}^{to}_{n}\right)$$

où apparaît la matrice tangente cohérente $\dfrac{\partial\, \underline{\sigma}|_{t+\Delta t}}{\partial \Delta \underline{\epsilon}^{to}}$ [9].

En annulant cette estimation linéarisée de r, nous obtenons la nouvelle estimation $\Delta \underline{\epsilon}^{to}_{n+1}$:

$$\Delta \underline{\epsilon}^{to}_{n+1} = \Delta \underline{\epsilon}^{to}_{n} - \left(\frac{\partial\, \underline{\sigma}|_{t+\Delta t}}{\partial \Delta \underline{\epsilon}^{to}}\right)^{-1}.r\left(\Delta \underline{\epsilon}^{to}_{n}\right) \tag{3}$$

Comme nous l'avons évoqué plus haut, l'utilisation de la matrice tangente cohérente, même si elle apparaît naturelle, n'est qu'une possibilité parmi d'autres et l'équation (3) s'écrit en pratique :

$$\Delta \underline{\epsilon}^{to}_{n+1} = \Delta \underline{\epsilon}^{to}_{n} - K^{-1}.r\left(\Delta \underline{\epsilon}^{to}_{n}\right) \tag{4}$$

La matrice de raideur K doit être bien choisie pour permettre la convergence de l'algorithme, mais ce choix n'influence pas la solution trouvée.

La matrice de raideur peut-être [10] :

- la matrice élastique (non endommagée, si la loi de comportement décrit l'endommagement du matériau) ;
- la matrice élastique endommagée (si la loi de comportement décrit l'endommagement du matériau), dite matrice sécante ;
- la matrice tangente au comportement, définie dans une formulation en vitesse par la dérivée $\left.\dfrac{\partial \dot{\underline{\sigma}}}{\partial \dot{\underline{\epsilon}}^{to}}\right|_{t+\Delta t}$;
- la matrice tangente cohérente, définie par la $\dfrac{\partial\, \underline{\sigma}|_{t+\Delta t}}{\partial \Delta \underline{\epsilon}^{to}}$;

L'interface `UMAT` présente une particularité déjà évoquée pour l'opérateur de prédiction permettant de proposer une matrice de raideur par défaut quand la loi n'en implante pas : le code `Cast3M` exige que des propriétés élastiques fassent systématiquement partie de la liste de propriétés matériau [11]

9. L'utilisation du symbole de dérivation partielle est conventionnelle. Elle vient du fait que la contrainte peut également dépendre d'autres variables internes, telle que la température, dans le cas de problème couplé.

10. Voir le mot clé `@StiffnessMatrixType`.

11. Notons que la loi peut ne pas utiliser les propriétés matériau qu'exige `Cast3M`. Dans ce cas, ces propriétés doivent être considérées comme des paramètres numériques permettant de construire une matrice de raideur effective utilisée pour la recherche de l'équilibre : la convergence de l'algorithme d'équilibre est garantie si cette matrice effective conduit à une réponse plus raide que celle de la loi de comportement.

Critères de convergence L'algorithme de NEWTON est répété jusqu'à ce que les deux critères de convergence suivants soient simultanément satisfaits :

— un critère sur la valeur du résidu, qui doit être inférieure à une certaine valeur notée ε_σ :

$$\left|\left| \underline{\sigma}|_{t+\Delta t} - \underline{\sigma}^{\text{ext}}|_{t+\Delta t} \right|\right| < \varepsilon_\sigma \tag{5}$$

— un critère sur la stationnarité des estimations de l'incrément des déformations totales : la différence entre deux estimations successives doit être inférieure à une certaine valeur notée ε_ϵ :

$$\left|\left| \Delta \underline{\epsilon}^{to}_{n+1} - \Delta \underline{\epsilon}^{to}_{n} \right|\right| < \varepsilon_\epsilon \tag{6}$$

Certaines conditions de chargement rajoutent leurs propres critères de convergence (contrainte imposée par exemple).

Vitesse de convergence Dans le cas où la matrice tangente cohérente $\frac{\partial\, \underline{\sigma}|_{t+\Delta t}}{\partial \Delta \underline{\epsilon}^{to}}$ est exacte, la convergence de l'algorithme de NEWTON est quadratique (ordre 2). Si une matrice approchée est utilisée, notamment la matrice d'élasticité, la convergence est en général linéaire (ordre 1). `mtest` fournit une estimation de l'ordre de convergence donnée à partir de quatre estimations successives de la solution $\Delta \underline{\epsilon}^{to}_{n+3}$, $\Delta \underline{\epsilon}^{to}_{n+2}$,$\Delta \underline{\epsilon}^{to}_{n+1}$ et $\Delta \underline{\epsilon}^{to}_{n}$ [Brezinski 06] :

$$o \approx \frac{\log\left(\frac{\left|\left| \Delta \underline{\epsilon}^{to}_{n+3} - \Delta \underline{\epsilon}^{to}_{n+2} \right|\right|}{\left|\left| \Delta \underline{\epsilon}^{to}_{n+2} - \Delta \underline{\epsilon}^{to}_{n+1} \right|\right|}\right)}{\log\left(\frac{\left|\left| \Delta \underline{\epsilon}^{to}_{n+2} - \Delta \underline{\epsilon}^{to}_{n+1} \right|\right|}{\left|\left| \Delta \underline{\epsilon}^{to}_{n+1} - \Delta \underline{\epsilon}^{to}_{n} \right|\right|}\right)}$$

Cette estimation est cependant assez souvent imprécise.

Accélération de convergence Dans le cas d'une convergence linéaire, il est possible d'utiliser un algorithme d'accélération de convergence introduit par les opérateurs de résolution de `Cast3M` [Pascal 05] [12].

Nous utilisons dans `mtest` la première version de cet algorithme. L'idée de cet algorithme est de :

— conserver les 3 derniers résidus, notés r_n, r_{n-1}, r_{n-2} et les 3 dernières estimations de la solution, notées $\Delta \underline{\epsilon}^{to}_{n}$, $\Delta \underline{\epsilon}^{to}_{n-1}$, $\Delta \underline{\epsilon}^{to}_{n-2}$.

— projeter le vecteur nul (la solution recherchée !) sur l'hyperplan construit à partir des trois derniers résidus. La figure 4 illustre géométriquement cette projection. Pour cela :

  — on choisit un des résidus comme origine du plan, dans notre implantation : r_{n-2} ;

  — on définit trois vecteurs $\tilde{r}_n$, $\tilde{r}_{n-1}$ et $\tilde{r}_0$ par :

$$\begin{cases} \tilde{r}_n = r_n - r_{n-2} \\ \tilde{r}_{n-1} = r_{n-1} - r_{n-2} \\ \tilde{r}_0 = -r_{n-2} \end{cases}$$

  Le vecteur $\tilde{r}_0$ est l'expression du vecteur nul dont nous recherchons la projection dans ce nouveau système de coordonnées.

  — on utilise le procédé d'orthogonalisation de SCHMIDT pour construire deux vecteurs normaux dans le plan à partir des vecteurs $\tilde{r}_{n-1}$ et $\tilde{r}_n$. Deux cas se présentent :

    — ces deux vecteurs sont colinéaires ;

    — ces deux vecteurs sont indépendants ;

  — si ces deux vecteurs sont colinéaires, nous pouvons définir un vecteur normal par :

$$n_1 = \frac{\tilde{r}_{n-1}}{||\tilde{r}_{n-1}||}$$

12. Voir le mot clé `@UseCastemAccelerationAlgorithm`.

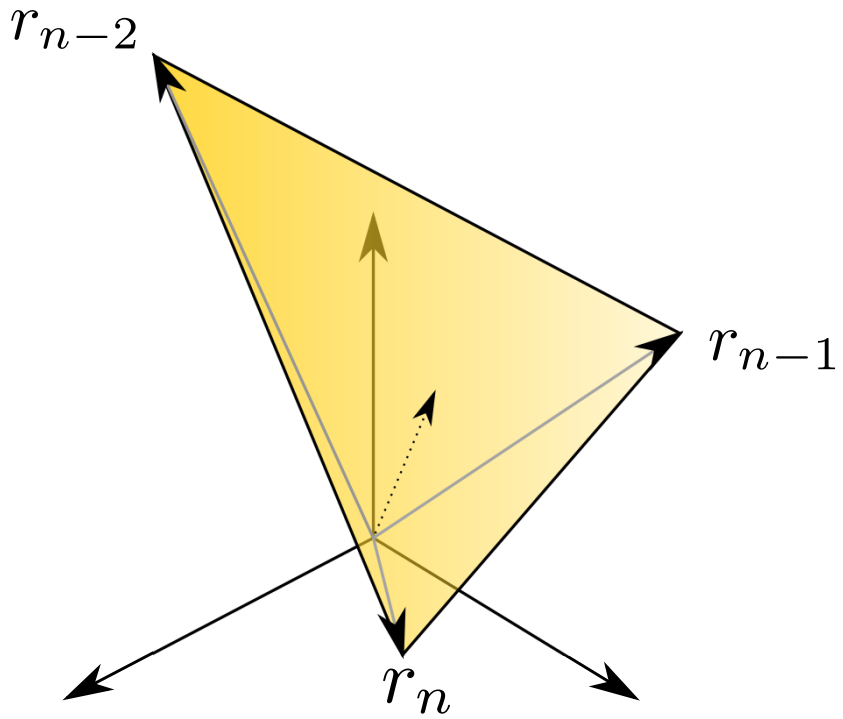

FIGURE 4 : Illustration géométrique de la méthode d'accélération de `Cast3M` : projection du vecteur nul sur l'hyperplan formé par les trois derniers résidus.

En notant c_1 le produit scalaire $\tilde{r}_0 \,.\, n_1$, la projection $\tilde{r}_\perp$ du vecteur $\tilde{r}_0$ s'écrit :

$$\tilde{r}_\perp = c_1 n_1 = \frac{c_1}{||\tilde{r}_{n-1}||} \tilde{r}_{n-1}$$

Dans le repère initial, nous avons :

$$r_\perp = \left(1 - \frac{c_1}{||\tilde{r}_{n-1}||}\right) r_{n-2} + \frac{c_1}{||\tilde{r}_{n-1}||} r_{n-1}$$

La nouvelle estimation des inconnues sera donnée par la même combinaison linéaire :

$$\Delta \underline{\epsilon}^{to}_\perp = \left(1 - \frac{c_1}{||\tilde{r}_{n-1}||}\right) \Delta \underline{\epsilon}^{to}_{n-2} + \frac{c_1}{||\tilde{r}_{n-1}||} \Delta \underline{\epsilon}^{to}_{n-1}$$

— si ces deux vecteurs sont indépendants, une nouvelle estimation des inconnues peut être fournie d'une manière tout à fait similaire au cas précédent.

La nouvelle estimation des inconnues peut ne pas vérifier les conditions cinématiques imposées, que nous traitons au paragraphe suivant, une nouvelle itération après l'étape d'accélération sera toujours systématiquement effectuée [13].

La méthode d'accélération peut s'avérer très efficace quand la matrice tangente cohérente n'est pas disponible. À titre d'exemple, la figure 5 compare comment les estimations d'équilibre convergent en fonction des options choisies pour la résolution : avec l'accélération de convergence, la vitesse de convergence est superlinéaire (approximativement $1,7$ d'après nos essais).

Cette méthode ne doit pas être utilisée avec une matrice tangente cohérente. En effet, dans ce cas, la vitesse de convergence serait dégradée (on perd la convergence quadratique).

13. Les dernières versions des opérateurs de résolution de `Cast3M` [Pascal 05] contournent cette difficulté en utilisant une stratégie légèrement différente : ils calculent le résidu accéléré et calculent une correction des inconnues en effectuant une itération supplémentaire de l'algorithme (4). L'estimation obtenue vérifie alors nécessairement les conditions cinématiques imposées.

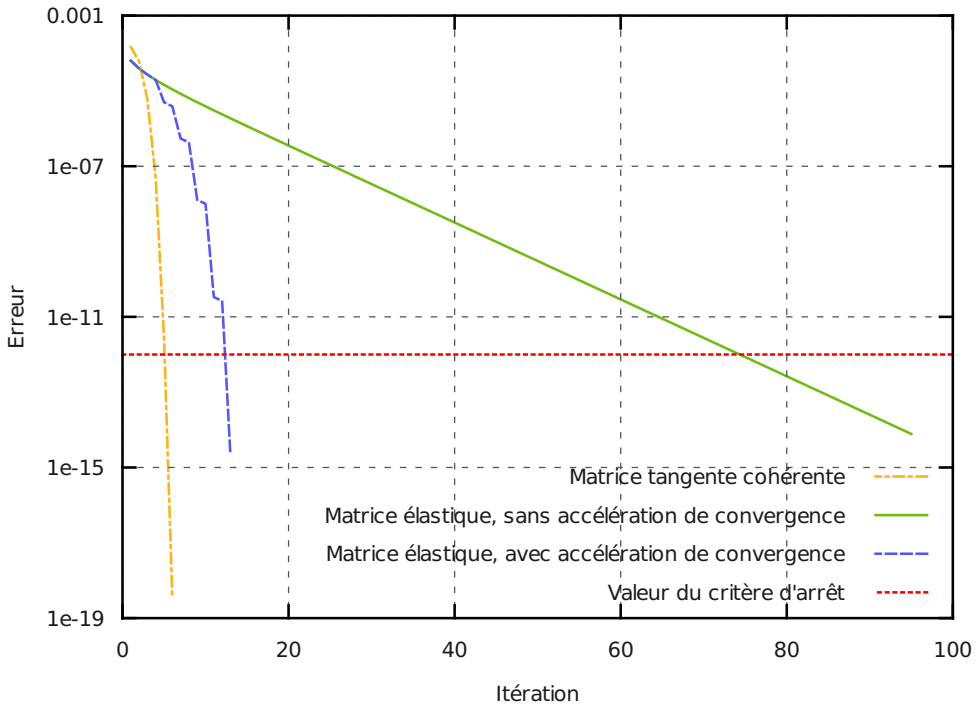

FIGURE 5 : Effet de la méthode d'accélération de `Cast3M` sur la convergence vers l'équilibre.

Par défaut l'accélération de convergence ne se déclenche qu'à la quatrième itération [14] et est répétée toutes les deux itérations [15].

Traitement des déformations imposées L'algorithme précédent doit être modifié pour tenir compte des déformations imposées. Pour cela, comme dans la plupart des codes éléments finis, de nouvelles variables nommées multiplicateurs de LAGRANGE sont introduites. Le principe de ces multiplicateurs est expliqué dans la documentation du code `Aster` (qui reprend ce principe de `Cast3M`) [Pellet 01, Abbas 13], à laquelle nous renvoyons le lecteur intéressé pour des détails complémentaires. Pour simplifier, nous traitons le cas d'une seule composante ϵ_i^{to} de la déformation dont la valeur est imposée à $\varepsilon^{\text{imp}}\left(t+\Delta t\right)$:

$$\epsilon_i^{to}\big|_t + \Delta\,\epsilon_i^{to} = \varepsilon^{\text{imp}}\left(t+\Delta t\right) \tag{7}$$

Pour cela, nous introduisons une nouvelle variables λ_i, appelée multiplicateur de LAGRANGE, telle que le résidu est :

$$r\left(\Delta\,\underline{\epsilon}^{to}, \Delta\,\lambda_i\right) = \begin{pmatrix} \underline{\sigma}\big|_{t+\Delta t}\left(\Delta\,\underline{\epsilon}^{to}\right) - \underline{\sigma}^{\text{ext}}\big|_{t+\Delta t} + a\,\left(\lambda_i|_t + \Delta\,\lambda_i\right)\begin{pmatrix}\vdots\\1\\\vdots\end{pmatrix} \\ a\,\left(\epsilon_i^{to}\big|_t + \Delta\,\epsilon_i^{to} - \varepsilon^{\text{imp}}\left(t+\Delta t\right)\right) \end{pmatrix} \tag{8}$$

où $\left(\ldots \quad 1 \quad \ldots\right)$ est un tenseur dont seule la i^{e} composante est non nulle et où a est un paramètre de normalisation tel que les différents termes de la matrice jacobienne soient du même ordre de grandeur. La jacobienne

14. Voir le mot clé `@CastemAccelerationTrigger`.
15. Voir le mot clé `@CastemAccelerationPeriod`.

est alors :

$$\frac{\partial r}{\partial \left(\Delta \underline{\epsilon}^{to}, \Delta \lambda_i\right)} = \begin{pmatrix} & \vdots & \\ \dfrac{\partial \sigma}{\partial \Delta \underline{\epsilon}^{to}} & a & \\ & \vdots & \\ \ldots \quad a \quad \ldots & 0 & \end{pmatrix}$$

On applique alors l'algorithme de NEWTON décrit jusqu'à présent. L'équation (8) montre qu'à convergence, nous aurons bien la condition imposée :

$$\epsilon_i^{to}\big|_t + \Delta\, \epsilon_i^{to} = \varepsilon^{\text{imp}}\left(t + \Delta\, t\right)$$

Si il n'y a pas de contraintes imposées sur la i^{e} composante, l'équation (8) montre que $a\ (\lambda_i|_t + \Delta\, \lambda_i)$ s'interprète comme la contrainte nécessaire pour imposer la contrainte (7).

Dans la suite, nous continuerons d'omettre les détails liés aux multiplicateurs de LAGRANGE [16].

Dilatation thermique La dilatation thermique peut être prise en compte pour les lois en petites déformations, de manière similaire à ce que font la plupart des codes. Pour cela, l'incrément des dilatations thermiques est tout simplement déduit de l'incrément de déformation fourni à la loi de comportement.

Pour définir une dilatation thermique orthotrope, il est nécessaire de définir les coefficients de dilatation thermique associés à chacune des trois directions principales, notés respectivement α_1, α_2 et α_3, par des propriétés matériau, nommées respectivement `ThermalExpansion1`, `ThermalExpansion2` et `ThermalExpansion3` [17]. Dans le repère propre du matériau, la dilatation thermique s'écrit alors :

$$\underline{\epsilon}^{th} = \begin{pmatrix} \alpha_1\ (T - T_{\text{ref}}) \\ \alpha_2\ (T - T_{\text{ref}}) \\ \alpha_3\ (T - T_{\text{ref}}) \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

où T_{ref} est la température de référence. Ce tenseur est exprimé dans le repère du calcul par un changement de repère [18].

Pour définir une dilatation thermique isotrope, il est nécessaire de définir un coefficient de dilatation thermique, noté α, par une propriété matériau, nommée `ThermalExpansion` [19]. La déformation thermique s'écrit alors :

$$\underline{\epsilon}^{th} = \begin{pmatrix} \alpha\ (T - T_{\text{ref}}) \\ \alpha\ (T - T_{\text{ref}}) \\ \alpha\ (T - T_{\text{ref}}) \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

où T_{ref} est la température de référence.

Par défaut, la température de référence est égale à $293.15\, K$. Cette valeur peut être changée en définissant une variable réelle nommée `ThermalExpansionReferenceTemperature` [20].

16. En particulier, nous confondons dans cette note la matrice de raideur donnée par le comportement et la matrice utilisée pour la résolution de l'équilibre.
17. Voir le mot clé `@MaterialProperty`.
18. Voir le mot clé `@RotationMatrix`.
19. Voir le mot clé `@MaterialProperty`.
20. Voir le mot clé `@Real`.

Prédiction initiale Par défaut, la solution trouvée au pas de temps précédent est utilisée comme estimation initiale de la solution en fin de pas de temps.

D'autres stratégies peuvent cependant être utilisées [21] :

— utiliser une extrapolation linéaire des précédents pas de temps. Si $\Delta\, t_p$ et $\Delta\, \underline{\epsilon}_p^{to}$ désignent respectivement l'incrément de temps et l'incrément de déformation du pas précédent, l'estimation initiale $\underline{\epsilon}_0^{to}$ de la solution est donnée par :

$$\underline{\epsilon}_0^{to} = \underline{\epsilon}^{to}\big|_t + \frac{\Delta\, t}{\Delta\, t_p}\Delta\, \underline{\epsilon}_p^{to}$$

— effectuer une première résolution utilisant une matrice de raideur K adaptée. Il est possible d'utiliser :
  — la matrice élastique non endommagée ;
  — la matrice élastique endommagée, dite matrice sécante ;
  — la matrice tangente au comportement, définie dans une formulation en vitesse par la dérivée $\left.\frac{\partial \dot{\underline{\sigma}}}{\partial \dot{\underline{\epsilon}}^{to}}\right|_t$ calculée en début de pas de temps.

  L'estimation initiale $\underline{\epsilon}_0^{to}$ de la solution est alors donnée par :

$$K\left(\underline{\epsilon}_0^{to} - \underline{\epsilon}^{to}\big|_t\right) = \underline{\sigma}|_t - \underline{\sigma}^{\text{ext}}\big|_{t+\Delta\, t} + K\left(\underline{\epsilon}^{th}\big|_{t+\Delta\, t} - \underline{\epsilon}^{th}\big|_t\right)$$

  On remarque que par rapport à l'équation (4), les dilatations thermiques doivent être explicitement prises en compte.

2.3 INTERFACES AUX LOIS DE COMPORTEMENT

`mtest` supporte les mêmes interfaces que `mfront` [22] :

— l'interface `umat` est utilisée pour une adhérence des lois au code `Cast3M` et aux différentes applications de la plate-forme `pleiades` [Helfer 13b] ;
— l'interface `cyrano` est utilisée pour une adhérence à l'application combustible d'ÉDF [Thouvenin 10].
— l'interface `Aster` est utilisée pour une adhérence au code aux éléments finis du même nom [Helfer 13a].

2.4 HYPOTHÈSES DE MODÉLISATION DISPONIBLES

Différentes hypothèses de modélisation peuvent être testées. Ces hypothèses se différencient par la dimension de l'espace ($1D$, $2D$ ou $3D$), et le nom des composantes des tenseurs.

La dimension d'espace ne suffit pas à distinguer les différentes hypothèses. Le cas des contraintes planes est un premier exemple et cette hypothèse demande généralement un traitement spécial au niveau de la loi de comportement.

Mais les lois de comportement peuvent également réagir différemment suivant l'hypothèse de modélisation, notamment les lois orthotropes. En particulier, nous avons montré que dans `Cast3M` les restrictions sur la définition des axes d'orthotropie nécessitaient de modifier la définition du tenseur de HILL de différentes manières en fonction de l'hypothèse utilisée [Helfer 13b].

Le tableau 1 donne la liste des hypothèses supportées par `mtest` et leurs caractéristiques.

Cas de l'interface `Aster` L'interface `Aster` ne permet pas aujourd'hui de distinguer les hypothèses de modélisation et ne se base que sur la dimension d'espace [23]. En $2D$, l'interface utilise la version plan généralisé de la loi de comportement.

21. Voir le mot-clé `@PredictionPolicy`.
22. Voir le mot clé `@Behaviour`.
23. Les contraintes planes sont traitées dans `Aster` par l'algorithme global par une méthode due à DE BORST [Proix 12].

Hypothèse	Dimension de l'espace	Nombre de composantes du tenseur des déformations	Nom des composantes du tenseur des déformations
AxisymmetricalGeneralisedPlaneStrain	1	3	ERR EZZ ETT
Axisymmetrical	2	4	ERR EZZ ETT ERZ
PlaneStress	2	4	EXX EYY EZZ EXY
PlaneStrain	2	4	EXX EYY EZZ EXY
GeneralisedPlaneStrain	2	4	EXX EYY EZZ EXY
Tridimensional	3	6	EXX EYY EZZ EXY EXZ EYZ

TABLEAU 1 : Liste des hypothèses de modélisation supportées et caractéristiques.

3 UTILISATION DE MTEST

Nous décrivons dans ce paragraphe l'utilisation de `mtest` en ligne de commande.

`mtest` s'utilise ainsi :

```
mtest [options] fichier1.mtest fichier2.mtest ...
```

La structure des fichiers d'entrée ainsi que l'ensemble des mots clés sont décrits au paragraphe 4.

`mtest` s'inspire de l'utilitaire `cmake` et permet de consulter sa documentation depuis la ligne de commandes. Par exemple, la documentation associée au mot clé `@Behaviour` est accessible ainsi :

```
$ mtest -help-keyword=@Behaviour
```

```
The @Behaviour keyword declares the behaviour used for the test. This
keyword must be followed by an option specifying the interface used by the
behaviour.
The umat, cyrano and aster interfaces are supported.
Two strings are then expected:
- the library in which the behaviour is implemented;
- the name of the function.
Example:
@Behaviour<umat> 'libMFrontCastemBehaviours.so' 'umatnorton';
```

3.1 OPTIONS DE MTEST

Nous détaillons dans ce paragraphe les options disponibles à la ligne de commandes.

L'option `--version` affiche la version de `mtest` utilisée.

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

L'option `--verbose` permet de régler le degré de verbosité de `mtest`. Plusieurs niveaux sont possibles :
- `quiet` correspond à un affichage minimal :
- `level1`, `level2` et `level3` désignent des niveaux de verbosité croissants.
- `debug` et `full` font apparaître des sorties qui ne sont généralement utiles qu'au développeur.

L'option `--help-keywords-list` affiche tous les mots clés utilisables dans le fichier d'entrée.

L'option `--help-keyword` affiche la documentation associée au mot clé donné en argument.

L'option `--help-commands-list` est synonyme de l'option `--help-keywords-list`. Elle a été introduite par compatibilité avec `cmake`.

L'option `--help-command` est synonyme de l'option `--help-keyword`. Elle a été introduite par compatibilité avec `cmake`.

L'option `--floating-point-exceptions` demande à ce que certaines exceptions mathématiques ne soient plus ignorées. Par défaut, les opérations mathématiques invalides conduisent à l'apparition de nombres particuliers nommés `NaN` ou `Inf` et n'arrêtent pas l'exécution du programme. Si l'option `--floating-point-exceptions` est utilisée, une opération mathématique invalide conduit le système d'exploitation à envoyer à `mtest` le signal `SIGFPE` qui finit son exécution (sauf si l'option `--backtrace` est utilisée).

L'option `--backtrace` affiche la pile d'appel en cas d'erreur de segmentation (signal `SIGSEGV`) ou en cas d'exception mathématique (signal `SIGFPE`).

L'option `--result-file-output` doit être suivi d'un argument valant `true` ou `false` suivant que l'utilisateur veut activer ou désactiver la génération d'un fichier de résultat.

3.2 Fichiers générés

Par défault, `mtest` génère deux fichiers de sortie :

- un fichier texte organisé en colonnes. La première colonne donne l'instant considéré. Les colonnes suivantes contiennent les différentes composantes de la déformation totale, les composantes de la contrainte et les variables internes de la loi de comportement. Par défaut, si le fichier d'entrée se nomme `xxx.mtest`, ce fichier se nomme `xxx.res` ;
- le second fichier est un fichier au format `xml` donnant les résultats des comparaisons à des résultats de référence (donnés par une formule analytique ou par un fichier externe) et le temps d'exécution du test. Ce fichier est directement analysable par le gestionnaire d'exécution `jenkins`. Par défaut, si le fichier d'entrée se nomme `xxx.mtest`, ce fichier se nomme `xxx.xml`.

4 DESCRIPTION DU FICHIER D'ENTRÉE

Cette section s'intéresse au fichier d'entrée de `mtest`. Nous décrivons tout d'abord sa structure puis la liste des mots clés disponibles.

```
@Author Helfer Thomas;
@Date   09 avril 2013;
@Description{
  "Test simple en traction uniaxiale imposant que la composante EXX
  de la déformation varie dans le temps de manière sinusoïdale. La
  contrainte SXX est égale à cette déformation multipliée par le
  module d'Young. Les déformations EYY et EZZ sont égales à cette
  déformation multipliée par l'opposé du coefficient de Poisson."
};

@Behaviour<aster> 'libMFrontAsterBehaviours.so' 'asterelasticity';
@MaterialProperty<constant> 'YoungModulus'     150.e9;
@MaterialProperty<constant> 'PoissonRatio'       0.3;
@MaterialProperty<constant> 'MassDensity'        0.;
@MaterialProperty<constant> 'ThermalExpansion'   0.;

@ExternalStateVariable 'Temperature' {0:293.15,3600.:800};

@Real 'e0' 1.e-3;
@ImposedStrain<function> 'EXX' 'e0*sin(t/900.)';

@Times {0.,3600 in 20};

// stresses
@Test<function> 'SXX' 'YoungModulus*EXX'  1.e-3;
@Test<function> 'SYY' '0.'                1.e-3;
@Test<function> 'SZZ' '0.'                1.e-3;
@Test<function> 'SXY' '0.'                1.e-3;
@Test<function> 'SXZ' '0.'                1.e-3;
@Test<function> 'SYZ' '0.'                1.e-3;
// strains
@Test<function> 'EYY' '-PoissonRatio*EXX' 1.e-12;
@Test<function> 'EZZ' '-PoissonRatio*EXX' 1.e-12;
@Test<function> 'EXY' '0.'                1.e-12;
@Test<function> 'EXZ' '0.'                1.e-12;
@Test<function> 'EYZ' '0.'                1.e-12;
```

FIGURE 6 : Premier exemple de fichier `mtest`.

Premier exemple La figure 6 donne un premier exemple de fichier d'entrée de `mtest`.

4.1 STRUCTURE

Le fichier d'entrée de `mtest` reprend la syntaxe des fichiers `mfront` : il se présente sous la forme d'une liste de mots clés commençant par une arobase `@`. Ces mots clés peuvent éventuellement être suivis par des options, données entre chevrons ouvrant < et fermant >.

Commentaires Les deux types de commentaires introduits par le langage C++ sont supportés :

- les commentaires commençant par les caractères `/*` et finissant par les caractères `*/`. Ces commentaires peuvent s'étendre sur plusieurs lignes ;
- les commentaires commençant par les caractères `//` s'étendent jusqu'à la fin de la ligne courante ;

Lecture d'un nombre réel À plusieurs endroits du fichier d'entrée, un nombre réel est attendu. S'il est donné par une chaîne de caractères, le contenu de cette chaine est interprété comme une formule mathématique et il est possible d'y utiliser les variables définies précédemment, à condition que celles-ci soient constantes dans le temps.

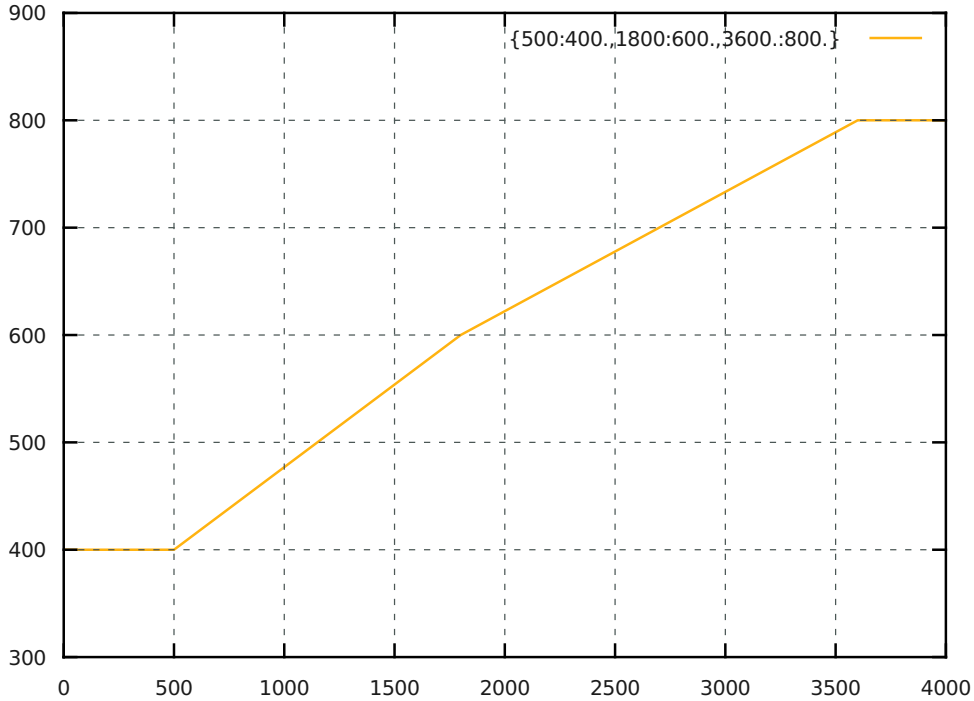

FIGURE 7 : Évolution temporelle correpondant à la déclaration `{500:400.,1800:600.,3600.:800.}`.

Description d'une évolution temporelle Les évolutions dans le temps de certaines variables sont des données essentielles qui doivent être renseignées par l'utilisateur. Ces évolutions peuvent être déclarées de deux manières :

- par une description discrète. Une évolution constante est donnée par une unique valeur réelle. Des évolutions plus complexes peuvent être rentrées sous la forme de tableaux associatifs associant à des temps la valeur des évolutions. Entre deux de ces temps, l'évolution est linéaire. Au delà des temps extrémaux, l'évolution est constante. Par exemple, l'extrait suivant correspond à l'évolution tracée en figure 7 :

```
{500:400.,1800:600.,3600.:800.}
```

Dans l'exemple de la figure 6, la température, en tant que variable externe, est définie de cette manière ;
— par une fonction explicite du temps, donnée comme une chaîne de caractères. Il est possible de réutiliser les évolutions déclarées précédemment ou d'utiliser explicitement le temps auquel correspond la variable `t`. Dans l'exemple de la figure 6, l'évolution de la composante `EXX` du tenseur des déformations est définie de cette manière.

4.2 LISTE DES MOTS CLÉS

Le mot clé `@Author` permet de donner le nom de la personne ayant écrit le fichier de test.

Tout ce qui suit ce mot clé jusqu'au premier point virgule est pris en compte.

Le mot clé `@Behaviour` déclare quelle loi de comportement mécanique est utilisée. Il est suivi du type d'interface, donné entre chevrons ouvrant `<` et fermant `>`. Deux interfaces sont actuellement supportées : l'interface `Aster` [Helfer 13a] et l'interface `umat` [Helfer 13b] correspondant au code `Cast3M`.

Deux arguments sont ensuite attendus :
— le nom de la librairie dans laquelle est implantée la loi ;
— le nom de la fonction qui implante la loi.

Le mot clé `@CastemAccelerationPeriod` permet de préciser la fréquence d'utilisation de la méthode d'accélération présentée au paragraphe 2.2.

Le mot clé `@CastemAccelerationTrigger` permet de préciser à partir de quelle période se déclenche la méthode d'accélération présentée au paragraphe 2.2.

Le mot clé `@Date` permet de préciser la date d'écriture du fichier de test.

Tout ce qui suit ce mot clé jusqu'au premier point virgule est pris en compte.

Le mot clé `@Description` permet de donner une description du test. Il est suivi d'un bloc contenant la description.

Le mot clé `@Evolution` permet de définir une fonction du temps (évolution).

Ce mot clé prend comme option le type d'évolution souhaitée. Si l'option `evolution` est utilisée, une évolution discrète sera attendue. Si l'option `function` est utilisée, la définition d'une fonction du temps sera attendue. Par défaut, une évolution discrète est attendue.

Le mot clé `@Evolution` prend deux arguments :
— le nom de la variable externe, sous la forme d'une chaîne de caractères ;
— l'évolution, dont la forme dépend du type choisi.

Le mot clé `@ExternalStateVariable` permet de spécifier l'évolution d'une variable externe à la loi de comportement. Dans la terminologie `Aster`, les variables externes s'appellent des variables de pilotage. Dans `Cast3M`, ces variables sont définies par des chargements. La température est considérée comme une variable externe particulière qui est obligatoire.

Ce mot clé prend comme option le type d'évolution souhaitée. Si l'option `evolution` est utilisée, une évolution discrète sera attendue. Si l'option `function` est utilisée, la définition d'une fonction du temps sera attendue. Par défaut, une évolution discrète est attendue.

Comme le mot clé `@Evolution`, le mot clé `@ExternalStateVariable` prend deux arguments :
- le nom de la variable externe, sous la forme d'une chaîne de caractères ;
- l'évolution, dont la forme dépend du type choisi.

Le mot clé `@ImposedStrain` permet de spécifier l'évolution d'une des composantes du tenseur des déformations.

Comme pour le mot clé `@ExternalStateVariable`, le type d'évolution est donné en option.

Hypothèse	Nom des composantes du tenseur des déformations					
AxisymmetricalGeneralisedPlaneStrain	ERR	EZZ	ETT			
Axisymmetrical	ERR	EZZ	ETT	ERZ		
PlaneStress	EXX	EYY		EXY		
PlaneStrain	EXX	EYY		EXY		
GeneralisedPlaneStrain	EXX	EYY	EZZ	EXY		
Tridimensional	EXX	EYY	EZZ	EXY	EXZ	EYZ

TABLEAU 2 : Liste des composantes du tenseur des déformations qu'il est possible d'imposer en fonction de l'hypothèse de modélisation.

Le mot clé `@ImposedStrain` prend deux arguments :
- le nom de la composante, sous la forme d'une chaîne de caractères. Les noms de composantes disponibles dépendent de l'hypothèse de modélisation choisie (voir le mot clé `@ModellingHypothesis`). Le tableau 2 donne la liste des composantes qu'il est possible d'imposer en fonction de l'hypothèse de modélisation ;
- l'évolution, dont la forme dépend du type choisi.

Le mot clé `@ImposedStress` permet de spécifier l'évolution d'une des composantes du tenseur des contraintes.

Comme pour le mot clé `@ExternalStateVariable`, le type d'évolution est donné en option.

Les tenseurs sont représentés en utilisant la convention en vigueur dans `Aster` et `mfront` : les composantes extra-diagonales des tenseurs de déformations sont affectées d'un $\sqrt{2}$.

Hypothèse	Nom des composantes du tenseur des contraintes					
AxisymmetricalGeneralisedPlaneStrain	SRR	SZZ	STT			
Axisymmetrical	SRR	SZZ	STT	SRZ		
PlaneStress	SXX	SYY		SXY		
PlaneStrain	SXX	SYY		SXY		
GeneralisedPlaneStrain	SXX	SYY	SZZ	SXY		
Tridimensional	SXX	SYY	SZZ	SXY	SXZ	SYZ

TABLEAU 3 : Liste des composantes du tenseur des contraintes qu'il est possible d'imposer en fonction de l'hypothèse de modélisation.

Le mot clé `@ImposedStress` prend deux arguments :
- le nom de la composante, sous la forme d'une chaîne de caractères. Les noms de composantes disponibles dépendent de l'hypothèse de modélisation choisie (voir le mot clé `@ModellingHypothesis`).

Le tableau 3 donne la liste des composantes qu'il est possible d'imposer en fonction de l'hypothèse de modélisation ;
— l'évolution, dont la forme dépend du type choisi.

Les tenseurs sont représentés en utilisant la convention en vigueur dans `Aster` et `mfront` : les composantes extra-diagonales des tenseurs de déformations sont affectées d'un $\sqrt{2}$.

Le mot clé `@InternalStateVariable` permet de donner la valeur initiale d'une variable interne.

Ce mot clé prend deux arguments :
— le nom de la variable interne ;
— une valeur réelle si la variable interne est scalaire, un tableau de valeurs réelles si la variable interne est tenseur d'ordre 2 symétrique. La taille du tableau dépend de l'hypothèse de modélisation (voir le tableau 1). Dans ce dernier cas, l'utilisateur doit respecter les conventions utilisées dans `mfront` : les termes extra-diagonaux sont affectés d'un facteur $\sqrt{2}$;

Dans le cas de tableau de variables internes, il y trois possibilités :
— `@InternalStateVariable 'omega' 6.1504e-09;` affecte à chacune des entrées du tableau la même valeur.
— `@InternalStateVariable 'omega[0]' 6.1504e-09;` affecte spécifiquement à la première entrée du tableau la valeur spécifiée.
— `@InternalStateVariable 'omega' 6.1504e-09,6.1504e-09;` affecte des valeurs distinctes à chacune des entrées du tableau.

Le mot clé `@MaterialProperty` permet de définir une des propriétés matériau.

Le type de propriété matériau est donné en option. Deux types sont supportés, `constant` et `castem`.

L'interface `castem` correspond à l'interface du même nom proposée par `mfront` et utilisée par certaines applications de la plate-forme `pleiades`. Il s'agit d'une extension apportée au code `Cast3M` permettant d'appeler des fonctions dans des libraires externes.

Le mot clé `@MaterialProperty` prend deux arguments :
— le nom de la propriété matériau ;
— une valeur réelle si la propriété matériau est constante, le nom de la librairie et de la fonction si elle est de type `castem` ;

Le mot clé `@Parameter` permet de spécifier la valeur d'un des paramètres réels de la loi. Ce mot clé est suivi du nom du paramètre et de sa valeur.

Le mot clé `@IntegerParameter` permet de spécifier la valeur d'un des paramètres entiers de la loi. Ce mot clé est suivi du nom du paramètre et de sa valeur.

Le mot clé `@UnsignedIntegerParameter` permet de spécifier la valeur d'un des paramètres entiers positifs de la loi. Ce mot clé est suivi du nom du paramètre et de sa valeur.

Le mot clé `@MaximumNumberOfIterations` permet de spécifier le nombre maximal d'itérations de l'algorithme global, nombre qui est le seul argument attendu.

Le mot clé `@MaximumNumberOfSubSteps` permet de spécifier le nombre maximal de sous-découpages en temps en cas de non convergence de l'algorithme global, nombre qui est le seul argument attendu.

Le mot clé `@ModellingHypothesis` permet de spécifier l'hypothèse de modélisation à utiliser.

Les hypothèses de modélisation valides sont listées au tableau 1.

Ce mot clé doit être l'un des premiers du fichier, car d'autres mots clés peuvent avoir besoin d'accéder à l'hypothèse de modélisation et si celle-ci n'est pas encore spécifiée, l'hypothèse par défaut sera déclarée. Dans ce cas, le traitement du mot `@ModellingHypothesis` conduira à une erreur.

Le mot clé `@OutputFile` permet de spécifier la base de noms des fichiers de sortie. `mtest` ajoute à cette base les extensions appropriées, par exemple `.xml` pour la sortie contenant le résultat des tests (voir à ce sujet le paragraphe 3.2).

Le mot clé `@OutputFilePrecision` permet de préciser le nombre de chiffres significatifs utilisés pour l'affichage des résultats. Ce nombre est l'unique argument attendu.

Le mot clé `@PredictionPolicy` permet de choisir comment est prédite l'estimation de la solution (voir le paragraphe 2.2).

Ce mot clé est suivi d'une chaîne de caractères. Les valeurs suivantes sont admises :

- `NoPrediction`, qui est le choix par défaut ;
- `LinearPrediction`, qui demande à ce que la prédiction se fasse par une interpolation linéaire ;
- `ElasticPrediction`, qui demande à ce que la prédiction se fasse par une résolution utilisant la matrice élastique non endommagée ;
- `ElasticPredictionFromMaterialProperies`, qui demande à ce que la prédiction se fasse à partir d'une matrice de raideur élastique calculée non par la loi mais sur la base des propriétés matériau. Cette option est uniquement supportée par l'interface UMAT : nous tirons parti du fait que le code `Cast3M` exige que des propriétés élastiques fassent systématiquement partie de la liste de propriétés matériau ;
- `SecantPrediction`, qui demande à ce que la prédiction se fasse par une résolution utilisant la matrice élastique éventuellement endommagée ;
- `TangentOperatorPrediction`, qui demande à ce que la prédiction se fasse par une résolution utilisant la matrice tangente en début de pas de temps.

Le mot clé `@Real` permet de déclarer une constante. Deux arguments sont attendus, le nom de la constante et sa valeur.

Le mot clé `@RotationMatrix` permet de spécifier les axes d'orthotropie du matériau. Il n'est valide que si la loi traitée est orthotrope.

Le mot clé `@StiffnessMatrixType` permet de spécifier quelle matrice doit être utilisée pour la recherche de l'équilibre.

Ce mot clé est suivi d'une chaîne de caractères. Les valeurs suivantes sont admises :

- `Elastic`, qui demande à ce que la recherche de l'équilibre se fasse en utilisant la matrice élastique non endommagée ;
- `SecantOperator`, qui demande à ce que la recherche de l'équilibre se fasse en utilisant la matrice élastique endommagée ;
- `TangentOperator`, qui demande à ce que la recherche de l'équilibre se fasse en utilisant la matrice tangente ;

— `ConsistentTangentOperator`, qui demande à ce que la recherche de l'équilibre se fasse en utilisant la matrice tangente cohérente.

Le type de matrice utilisée par défaut dépend de l'interface à la loi de comportement utilisée.

Le mot clé `@Strain` permet de donner la valeur initiale des déformations totales.

Ce mot clé prend un argument, un tableau de valeurs réelles. La taille de ce tableau dépend de l'hypothèse de modélisation (voir le tableau 1). L'utilisateur doit respecter les conventions utilisées dans `mfront` : les termes extra-diagonaux sont affectés d'un facteur $\sqrt{2}$;

Le mot clé `@Stress` permet de donner la valeur initiale des contraintes.

Ce mot clé prend un argument, un tableau de valeurs réelles. La taille de ce tableau dépend de l'hypothèse de modélisation (voir le tableau 1). L'utilisateur doit respecter les conventions utilisées dans `mfront` : les termes extra-diagonaux sont affectés d'un facteur $\sqrt{2}$;

Le mot clé `@StrainEpsilon` permet de spécifier la valeur de critère à utiliser pour tester la stationnarité des prédictions de l'incrément des déformations totales (voir équation (6)).

Le mot clé `@StressEpsilon` permet de spécifier la valeur de critère à utiliser pour tester si l'équilibre mécanique est atteint (voir équation (5)).

Le mot clé `@Test` permet d'introduire de tests sur les déformations totales, les contraintes ou les variables internes. Ce mot clé n'a pas d'intérêt pour un utilisateur standard. Il est utilisé pour créer des cas de non régressions qui contient de nombreux exemples d'utilisations.

Dans ce paragraphe, on appellera résultat :
— la valeur d'une variable interne si elle est scalaire ;
— la valeur d'une des composantes d'une variable interne (si elle est tensorielle), des contraintes ou des déformations.

Ce mot clé prend en option le type de test à effectuer, `function` ou `file`.

Les tests de type `function` comparent les résultats à des solutions analytiques. Dans ce cas, deux syntaxes sont possibles :
— la première attend le nom d'un résultat et la fonction du temps à laquelle le résultat est comparé ;
— la seconde syntaxe attend un dictionnaire associant à un résultat la fonction du temps à laquelle il est comparé.

Les deux syntaxes attendent un second argument qui est la valeur du critère de comparaison à utiliser.

Les tests de type `file` comparent les résultats à des solutions de référence. Le premier argument attendu est le nom de fichier contenant les résultats de référence. Deux syntaxes sont alors possibles :
— la première attend le nom d'un résultat et le numéro de colonne contenant le résultat de référence auquel le résultat est comparé ;
— la second syntaxe attend un dictionnaire associant à un résultat le numéro de colonne contenant le résultat de référence auquel le résultat est comparé.

Les deux syntaxes attendent un troisième argument qui est la valeur du critère de comparaison à utiliser.

Dans tous les cas, la comparaison entre un résultat et le résultat attendu se fait de manière absolue.

Les tests sont effectués à chaque pas de temps, une fois l'équilibre atteint.

Le mot clé `@Times` précise les temps de calculs. Ces temps sont donnés par intervalles. Un intervalle peut être découpé un certain nombre de fois à l'aide du mot clé `in`.

Le mot clé `@UseCastemAccelerationAlgorithm` précise si l'on veut utiliser l'algorithme d'accélération de convergence présenté en paragraphe 2.2

Ce mot clé est suivi de la valeur `true` si l'on veut activer cet algorithme, `false` sinon.

5 QUELQUES EXEMPLES

Nous décrivons quelques exemples de mise en œuvre de `mtest`.

5.1 ESSAIS ÉLÉMENTAIRES

Il est intéressant de préciser comment simuler quelques essais élémentaires.

5.1.1 Essai de traction uniaxiale en déplacement imposé

Il est possible de simuler un essai de traction uniaxiale en déplacement imposé en ne spécifiant que la déformation dans l'axe de traction. L'équilibre imposera que les autres composantes des contraintes soient nulles.

Il est possible de se ramener à des données expérimentales de type « force-déplacement » par les équivalences classiques :

- le déplacement est égal à la déformation multipliée par la longueur de l'éprouvette dans l'axe de traction ;
- la force est égale à la contrainte multipliée par la surface de la section de l'éprouvette transverse à l'axe de traction.

Nous avons déjà recontré en figure 6 un exemple de simulation d'un essai de traction uniaxiale en déplacement imposé.

5.1.2 Essai de traction uniaxiale en force imposée

Il est possible de simuler un essai de traction uniaxiale en force imposée en ne spécifiant que la contrainte dans l'axe de traction. L'équilibre imposera que les autres composantes des contraintes soient nulles.

Il est possible de se ramener à des données expérimentales de type « force-déplacement « par les mêmes équivalences qu'au paragraphe précédent.

Ce type d'essai permet en particulier une première modélisation rapide d'essais de fluage (charge constante). Pour ces essais, il est intéressant de ne pas simuler la phase de mise en charge en fournissant des valeurs initiales des déformations, contraintes et variables internes adéquates [24].

Le cas test `strainhardeningcreep.mtest`, reporté en figure 8, donne un exemple de simulation d'un essai de fluage pour une loi d'écoulement viscoplastique avec écrouissage. Le résultat de ce cas test est comparé à la solution analytique en figure 9.

5.1.3 Essai de cisaillement

L'essai de cisaillement est un essai numérique pratique permettant de tester simplement des lois.

5.1.4 Tube sous pression

Dans le cas d'un point matériel, nous pouvons simuler un tube sous pression en imposant les contraintes à l'aide des solutions classiques.

Le système d'axes naturel pour l'analyse des contraintes dans un tube est donné par le repère cylindrique illustré en figure 10.

24. Voir les mots clés `@Strain`, `@Stress` et `@InternalStateVariable`

```
@Author Helfer Thomas;
@Date   09 avril 2013;
@Description{
  "Essai de fluage sur un matériau modélisé"
  "par une loi de type Lemaître (loi viscoplastique "
  "avec écroussage)."
  "Vérification de la réponse fournie par "
  "une loi générée par l'analyseur "
  "IsotropicStrainHardeningMisesCreep de mfront."
  "Comparaison à la solution analytique."
};

@ModellingHypothesis 'Axisymmetrical';
@UseCastemAccelerationAlgorithm true;

@Behaviour<umat> 'libMFrontCastemBehaviours.so' 'umatstrainhardeningcreep';
@MaterialProperty<constant> 'YoungModulus'     150.e9;
@MaterialProperty<constant> 'PoissonRatio'       0.3;
@MaterialProperty<constant> 'MassDensity'        0.;
@MaterialProperty<constant> 'ThermalExpansion'   0.;
@MaterialProperty<constant> 'A'  8.e-40;
@MaterialProperty<constant> 'Ns' 4.2;
@MaterialProperty<constant> 'Np' 1.37;

@Real 'srr' -20e6;
@ImposedStress 'SRR' 'srr';
// Initial value of the elastic strain
@Real 'EELRR0' 0.00013333333333333333;
@Real 'EELZZ0' -0.00004;
@InternalStateVariable 'ElasticStrain' {'EELRR0','EELZZ0','EELZZ0',0.};
// Initial value of the total strain
@Strain {'EELRR0','EELZZ0','EELZZ0',0.};
// Initial value of the total stresses
@Stress {'srr',0.,0.,0.};

@ExternalStateVariable 'Temperature' 293.15;

@Times {0.,10. in 200, 100. in 30,3600 in 20};

@Real 'p0' 1.e-6;
@Test<function> 'EquivalentViscoplasticStrain'
     '(p0**(Np+1)-(Np+1)*A*srr**Ns*t)**(1/(Np+1))-p0' 1.e-5;
```

FIGURE 8 : Simulation d'un essai de fluage.

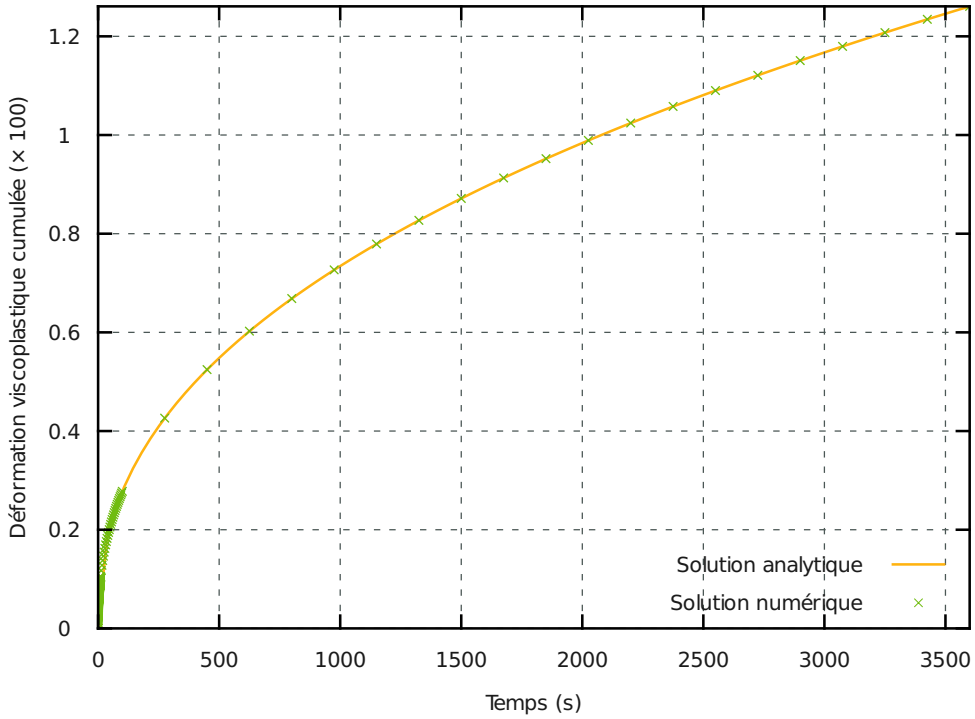

FIGURE 9 : Comparaison entre la solution obtenue par la simulation `mtest` et la solution analytique.

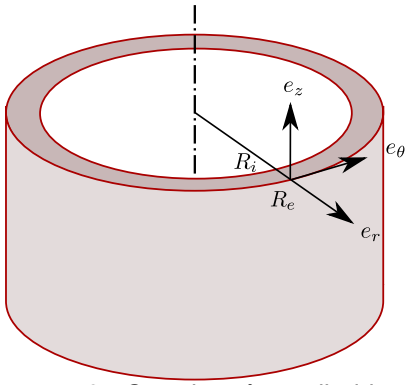

FIGURE 10 : Coordonnées cylindriques.

Nous considérons donc un tube sous pression fermé de rayon interne R_i et de rayon externe R_e. La pression à l'intérieur du crayon est notée P_i et la pression externe P_e.

La contrainte axiale est donnée par l'effet de fond :

$$\sigma_z = \frac{1}{R_e^2 - R_i^2}\left(R_i^2 P_i - R_e^2 P_e\right)$$

Les composantes radiales et tangentielles sont données par les expressions classiques suivantes :

$$\sigma_r = K_1 + \frac{K_2}{r^2}$$
$$\sigma_\theta = K_1 - \frac{K_2}{r^2}$$

où K_1 et K_2 sont deux paramètres choisis pour vérifier les conditions aux limites $\sigma_r\left(R_i\right) = -P_i$ et $\sigma_r\left(R_e\right) = -P_e$. Ces paramètres valent :

$$K_2 = \frac{R_i^2 R_e^2}{R_e^2 - R_i^2}\left(P_e - P_i\right)$$
$$K_1 = -P_i - \frac{K_2}{R_i^2}$$

La mise en données d'un tel test est donnée en figure 11.

5.2 UN CAS TEST COMPLEXE

Nous décrivons dans ce paragraphe un test, proposé par l'IPSI (Insititut pour la Promotion des Sciences de l'Ingénieur) pour la journée $\Phi^2 AS$ [25] du 30 Mars 2000 sur les comportements non linéaires, qui permet de valider la bonne prise en compte de la variation des coefficients avec la température [Proix 11]. Il s'agit d'un test particulièrement sollicitant pour les lois de comportement.

5.2.1 Chargements

Deux chargements mécaniques sont imposés au point matériel :
- une déformation ε_{xx} variable dans le temps ;
- une contrainte de cisaillement σ_{xy} qui est constante dans le temps à $100MPa$.

Une température T évoluant de manière cyclique est également prise en compte.

Les évolutions de la déformation ε_{xx} et de la température au cours du temps sont reportées en figure 12.

5.2.2 Application

Le cas test `chaboche2.mtest` applique les conditions aux limites décrites au paragraphe précédent à une loi viscoplastique à écrouissage cinématique non linéaire donnée en figure 13.

Le fichier d'entrée de ce cas test est donné en figure 14.

La figure 15 compare la solution obtenue à une simulation effectuée avec la macro-commande `SIMU_POINT_MAT` d'`Aster` [EDF 13a].

25. Formation IPSI pour la Formation et l'Information en Analyse de Structures.

```
@Author Helfer Thomas;
@Date   09 avril 2013;
@Description{
  "Modélisation d'un tube sous pression"
};

@ModellingHypothesis 'Axisymmetrical';
@Behaviour<aster> 'libMFrontAsterBehaviours.so' 'asterelasticity';
@MaterialProperty<constant> 'YoungModulus'     150.e9;
@MaterialProperty<constant> 'PoissonRatio'       0.3;
@MaterialProperty<constant> 'MassDensity'        0.;
@MaterialProperty<constant> 'ThermalExpansion'   0.;

@ExternalStateVariable 'Temperature' 293.15;

// Phenix cladding radii
@Real 'Ri' 2.825e-3;
@Real 'Re' 3.250e-3;
@Real 'Rm' '0.5*(Re+Ri)';
// inner and outer pressures
@Evolution 'Pi' {0:0,1:1.e5};
@Evolution 'Pe' {0:0,1:2.e5};

@Evolution<function> 'K2' 'Ri**2*Re**2/(Re**2-Ri**2)*(Pe-Pi)';
@Evolution<function> 'K1' '-Pi-K2/Ri**2';

@ImposedStress<function> 'SRR' 'K1+K2/Rm**2';
@ImposedStress<function> 'STT' 'K1-K2/Rm**2';
@ImposedStress<function> 'SZZ' '1/(Re**2-Ri**2)*(Ri**2*Pi-Re**2*Pe)';

@Times {0.,1.};
```

FIGURE 11 : Exemple de fichier `mtest` simulant un tube en pression.

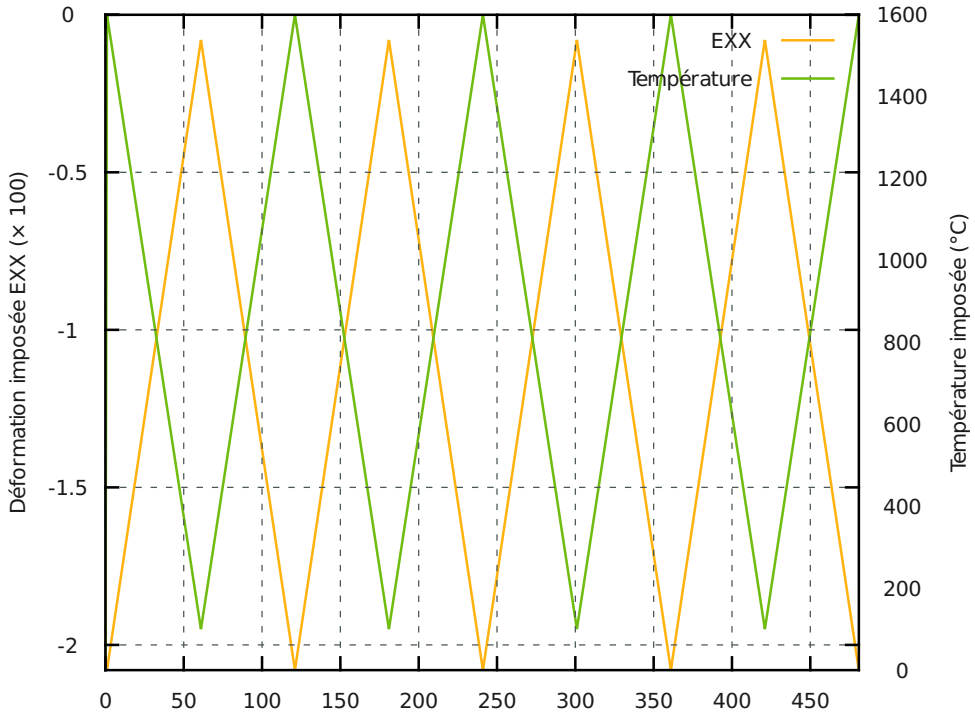

FIGURE 12 : Évolution de la déformation ε_{xx} et de la température au cours de l'essai `HSNV125` [Proix 11].

```
@Author jmp;
@Date   13 02 2013;

@Parser    Implicit;
@Behaviour Chaboche2;

@Algorithm NewtonRaphson_NumericalJacobian;
@Theta 1.0 ;
@Epsilon 1.e-12 ;
@IterMax 200 ;

@MaterialProperty real young; /* mandatory for castem */
young.setGlossaryName("YoungModulus");
@MaterialProperty real nu;    /* mandatory for castem */
nu.setGlossaryName("PoissonRatio");
@MaterialProperty real rho;   /* mandatory for castem */
rho.setGlossaryName("MassDensity");
@MaterialProperty real alpha; /* mandatory for castem */
alpha.setGlossaryName("ThermalExpansion");
@MaterialProperty real Rinf,R0,b,k,w;
@MaterialProperty real C1inf,g1;
@MaterialProperty real C2inf,g2,E,NsurK;

@Includes{
#include"TFEL/Material/Lame.hxx"
}

@StateVariable real    p;
@StateVariable Stensor a1,a2;
@LocalVariable real    lambda,mu;

/* Initialize Lame coefficients */
@InitLocalVars{
  using namespace tfel::material::lame;
  lambda = computeLambda(young,nu);
  mu = computeMu(young,nu);
}

@TangentOperator{
  using namespace tfel::material::lame;
  if((smt==ELASTIC)||(smt==SECANTOPERATOR)){
    computeElasticStiffness<N,Type>::exe(Dt,lambda,mu);
  } else if (smt==CONSISTENTTANGENTOPERATOR){
    StiffnessTensor De;
    Stensor4 Je;
    computeElasticStiffness<N,Type>::exe(De,lambda,mu);
    getPartialJacobianInvert(Je);
    Dt = De*Je;
  } else {
    return false;
  }
}

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

@Integrator{
  Stensor n = Stensor(0.);
  const Stensor a1_ = (a1+theta*da1) ;
  const Stensor a2_ = (a2+theta*da2) ;
  const Stensor X1_ = C1inf*(a1_)/1.5 ;
  const Stensor X2_ = C2inf*(a2_)/1.5 ;
  const real p_ = (p +theta*dp) ;
  const Stensor scin = sig - X1_ - X2_ ;
  const real seq = sigmaeq(scin);
  const real Rp = Rinf + (R0-Rinf)*exp(-b*p_) ;
  const real F = seq - Rp ;
  real vp=0.;
  if(F > 0){
     vp  = pow(F*UNsurK,E) ;
     const real inv_seq = 1/seq;
     n       = 1.5*deviator(scin)*inv_seq;
     feel += vp*dt*n-deto;
     fp   -= vp*dt;
     fa1  = da1 -vp*dt*n + g1*vp*dt*a1_;
     fa2  = da2 -vp*dt*n + g2*vp*dt*a2_;
  } else {
    feel -= deto;
  }
}
```

FIGURE 13 : Exemple d'une loi viscoplastique à écrouissage cinématique non linéaire.

```
@Author jmp;
@Date   25 avril 2013;
@Description{
  "Test d'une loi viscoplastique de Chaboche sans effet de memoire"
};

@Behaviour<aster> 'libMFrontAsterBehaviours.so' 'asterchaboche2';
@MaterialProperty<function> 'YoungModulus' '2.e5 - (1.e5*((TC - 100.)/960.)**2)';
@MaterialProperty<constant> 'PoissonRatio'  0.3 ;
@MaterialProperty<constant> 'MassDensity'   0.;
@MaterialProperty<function> 'ThermalExpansion' '1.e-5 + (1.e-5  * ((TC - 100.)/960.) **
4)';
@Real 'ThermalExpansionReferenceTemperature' 0.;

@MaterialProperty<constant> 'Rinf' 100. ;
@MaterialProperty<constant> 'R0' 200. ;
@MaterialProperty<constant> 'b'  20. ;
@MaterialProperty<constant> 'k'  1.0;
@MaterialProperty<constant> 'w'  0. ;
@MaterialProperty<function> 'C1inf' '(1.e6 - (98500. * (TC - 100.) / 96.))';
@MaterialProperty<constant> 'C2inf' 0.;
@MaterialProperty<function> 'g1'   '(5000. - 5. * (TC - 100.) )' ;
@MaterialProperty<constant> 'g2'   0. ;
@MaterialProperty<function> 'E'   '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 ;
@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: '100.0*sqrt(2.0)', 1000. :
'100.0*sqrt(2.0)'};

@Times { 0.0, 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 };

@Test<file> '@top_srcdir@/mfront/tests/behaviours/references/chaboche2-aster.ref'
            'EYY' 3 1.e-5 ;
@Test<file> '@top_srcdir@/mfront/tests/behaviours/references/chaboche2-aster.ref'
            {'SXX':8,'SXY':36} 3.;
```

FIGURE 14 : Test d'une loi viscoplastique à écrouissage cinématique non linéaire.

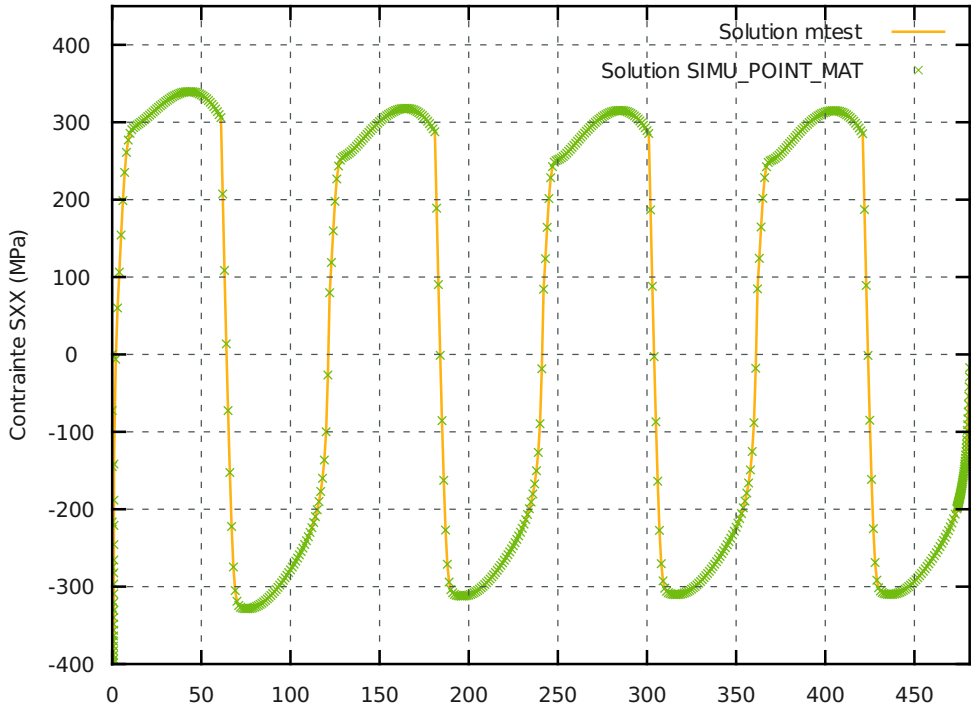

FIGURE 15 : Comparaison de la solution obtenue au cours de l'essai `HSNV125` [Proix 11] à la solution obtenue par la macro-commande `SIMU_POINT_MAT` d'`Aster` [EDF 13a].

6 CONCLUSIONS

Ce document présente un outil de simulation du comportement mécanique d'un point matériel nommé `mtest`. Initialement développé pour des besoins internes permettant d'assurer la qualité du développement de `mfront`, cet outil a su trouver des applications plus larges et est utilisé dans de nombreuses situations où le recours à un calcul de structure n'est pas nécessaire.

Si `mtest` a été initialement développé pour des besoins internes au développement de `mfront`, `mtest` semble apprécié des utilisateurs de `mfront`, au sein du projet `pleiades` ou à l'extérieur. Il est utilisé :

- comme un outil de validation unitaire des lois de comportement ;
- comme un outil d'optimisation des lois de comportement [26] ;
- comme un outil de recalage des paramètres du comportement. `mtest` a notamment été couplé avec le code `Matlab` pour l'interprétation d'essais sur un acier EM10 [Milliard 14] et avec le module `ADAO` de la plate-forme `Salomé`.

26. Il est notamment possible de demander à une loi `mfront` de générer automatiquement un fichier `mtest` en cas de non convergence de l'intégration sur un pas de temps. Il est ainsi possible d'analyser les causes de divergence sans avoir à relancer un calcul éléments finis beaucoup plus coûteux.

RÉFÉRENCES

[Abbas 13] ABBAS MICKAEL. *Algorithme non linéaire quasi-statique STAT_NON_LINE*. Référence du Code Aster R5.03.01 révision : 10290, EDF-R&D/AMA, Janvier 2013.

[Brezinski 06] BREZINSKI CLAUDE et REDIVO ZAGLIA-MICHELA. *Méthodes numériques itératives : algèbre linéaire et non linéaire : niveau M1*. Ellipses Édition, Paris, 2006.

[Castelier 09] CASTELIER ÉTIENNE, HELFER THOMAS et PACULL JULIEN. *Spécifications d'un code de thermomécanique avec résolution par transformées de Fourier rapides*. Note technique 09 - 045, DEC/SESC/LSC, 2009.

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

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

[EDF 13a] EDF . *Macro-commande SIMU_POINT_MAT*. Référence du Code Aster U4.51.12 révision 9069, EDF-R&D/AMA, Avril 2013.

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

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

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

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

[Helfer 13c] HELFER THOMAS 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 13d] 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.

[Jérôme 10] JÉRÔME PASCAL et CASTELIER ÉTIENNE. *TMFFT Software Conception*. Rapport technique, STILOG, 2010.

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

[Milliard 14] MILLIARD FRANCK, COURCELLE ARNAUD, GAVOILLE PIERRE et FALESKOG JONAS. *Mechanical behavior of hexagonal tubes during accidental situation*, Février 2014.

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

[Pascal 05] PASCAL SERGE. *Notice d'utilisation du composant mécanique de PLEIADES : INCREPL*. Rapport technique RT/05-011/A PLE 05-007, SEMT/LM2S, Mars 2005.

[Pellet 01] PELLET JACQUES. *Dualisation des conditions aux limites*. Référence du Code Aster R3.03.01-B, EDF-R&D/MITI/MMN, 2001.

[Proix 11] PROIX JEAN-MICHEL. *HSNV125 - Élément de volume en traction / cisaillement et température variables*. Référence du Code Aster V4.22.125 révision : 7982, EDF-R&D/AMA, 2011.

[Proix 12] PROIX JEAN-MICHEL. *Prise en compte de l'hypothèse des contraintes planes dans les comportements non linéaires*. Référence du Code Aster R5.03.03 révision 10101, EDF-R&D/AMA, 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.

[Salvo 14] SALVO MAXIME. *Caractérisation expérimentale et modélisation du comportement mécanique de l'UO2 en compression à haute vitesse de sollicitation*, Février 2014. Réunion d'avancement de thèse.

[Stroustrup 04] STROUSTRUP BJARNE et EBERHARDT CHRISTINE. *Le langage C++*. Pearson Education, Paris, 2004.

[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 98] VERPEAUX PIERRE, CHARRAS THIERRY et MILLARD ALAIN. *CASTEM2000 : une approche moderne du calcul des structures*. Calcul des structures et intelligence artificielle. Pluralis Editions, 1998.

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

LISTE DES TABLEAUX

LISTE DES FIGURES

ANNEXE A ALGORITHMES D'ACCÉLÉRATION

Les algorithmes d'accélération suivants sont disponibles. Ils sont définis pour accélérer des itérations de point fixe $X_{n+1} = G(X_n)$. La résolution de l'équilibre à l'aide de l'opérateur élastique K rentre dans ce cadre avec

$$G(X) = X - K^{-1} R(X) \tag{9}$$

Ces algorithmes sont détaillés dans [**?**].
Dans la suite on utilisera l'opérateur de différence Δ tel que $\Delta X_n = G(X_n) - X_n$, $\Delta G(X_n) = G(G(X_n)) - G(X_n)$ and $\Delta^2 X_n = \Delta G(X_n) - \Delta X_n$

— La « crossed secant method » ou « crossed 1-δ method » (souvent appelée relaxation d'Aitken ou relaxation dynamique dans la litterature)

$$X_{n+1} = G(X_n) - \frac{(G(X_n) - G(X_{n-1})).(\Delta X_n - \Delta X_{n-1})}{\|\Delta X_n - \Delta X_{n-1}\|^2} \Delta X_n \tag{10}$$

— La « alternate secant method » ou « alternate 1-δ method » (aussi appelée méthodes d'extrapolation d'Anderson avec $M = 1$)

$$X_{n+1} = G(X_n) - \frac{(\Delta X_n - \Delta X_{n-1}).\Delta X_n}{\|\Delta X_n - \Delta X_{n-1}\|^2} (G(X_n) - G(X_{n-1})) \tag{11}$$

— La méthode de Irons and Tuck à laquelle reviennent les 2 méthodes précédentes si elles ne sont appelées qu'une itération de point fixe sur 2.

$$\begin{aligned} X_n &= G(X_{n-1}) \\ X_{n+1} &= G(X_n) - \frac{\Delta X_n.\Delta^2 X_{n-1}}{\|\Delta^2 X_{n-1}\|^2} \Delta X_n \end{aligned} \tag{12}$$

— La « crossed δ^2 method »

$$X_{n+1} = G(X_n) - \frac{(G(X_n) - G(X_{n-1})).(\Delta X_n - 2\Delta X_{n-1} + \Delta X_{n-2})}{\|\Delta X_n - 2\Delta X_{n-1} + \Delta X_{n-2}\|^2} (\Delta X_n - \Delta X_{n-1}) \tag{13}$$

— La « alternate δ^2 method «

$$X_{n+1} = G(X_n) - \frac{(\Delta X_n - 2\Delta X_{n-1} + \Delta X_{n-2}).\Delta X_n}{\|\Delta X_n - 2\Delta X_{n-1} + \Delta X_{n-2}\|^2} (G(X_n) - 2G(X_{n-1}) + G(X_{n-2})) \tag{14}$$

— La « crossed 2-δ method »

$$\begin{aligned} X_{n+1} &= G(X_n) - \lambda_n^1 \Delta X_n - \lambda_n^2 \Delta X_{n-1} \\ &\text{avec } \lambda_n^1, \lambda_n^2 \text{ minimisant} \\ \delta Y_n &= (G(X_n) - G(X_{n-1})) - \lambda_n^1 (\Delta X_n - \Delta X_{n-1}) - \lambda_n^2 (\Delta X_{n-1} - \Delta X_{n-2}) \end{aligned} \tag{15}$$

— La « alternate 2-δ method » (équivalente à la méthode d'extrapolation d'Anderson avec $M = 2$)

$$\begin{aligned} X_{n+1} &= G(X_n) - \lambda_n^1 (G(X_n) - G(X_{n-1})) - \lambda_n^2 (G(X_{n-1}) - G(X_{n-2})) \\ &\text{avec } \lambda_n^1, \lambda_n^2 \text{ minimisant} \\ \delta Y_n &= \Delta X_n - \lambda_n^1 (\Delta X_n - \Delta X_{n-1}) - \lambda_n^2 (\Delta X_{n-1} - \Delta X_{n-2}) \end{aligned} \tag{16}$$

— La « crossed 2-δ bis method » [27]

$$\begin{aligned} X_{n+1} &= G(X_n) - \lambda_n^1 (G(X_n) - X_n) - \lambda_n^2 (G(X_n) - X_{n-1}) \\ &\text{avec } \lambda_n^1, \lambda_n^2 \text{ minimisant} \\ \delta Y_n &= (G(X_n) - G(X_{n-1})) - \lambda_n^1 (\Delta X_n - \Delta X_{n-1}) - \lambda_n^2 (G(X_n) - G(X_{n-1}) - X_{n-1} + X_{n-2}) \end{aligned} \tag{17}$$

27. Attention, dans [**?**], il est montré que cette approche pouvait ne pas converger vers la solution de l'équation de point fixe

INDEX DES DIRECTIVES

U