La librairie `tfel` : guide de référence

T. Helfer

2013

RÉSUMÉ

Cette note constitue la notice de référence de la librairie `tfel`.

La librairie `tfel` vise à se doter d'outils, notamment mathématiques, basés sur des paradigmes de programmation avancés. Cette librairie a été découpée en modules, dont les principaux sont `Math`, `Material`, `System` et `Utilities`. Les principales fonctionnalités de ces différents modules sont décrits.

Nous décrivons ensuite certaines techniques de programmation qui sous-tendent la librairie et permettent d'atteindre des performances élevées.

La liste des plate-formes et des compilateurs supportés permet de juger de la qualité logicielle de la librairie et de sa portabilité.

La dernière partie décrit les procédures d'installation de `tfel`.

SOMMAIRE

1 INTRODUCTION

Ce note constitue le guide de référence de la bibliothèque `tfel`.

La bibliothèque `tfel` vise à se doter d'outils, notamment mathématiques, basés sur des paradigmes de programmation avancés, dont nous présenterons certains en section 7.

Elle est aujourd'hui intégrée à l'architecture `pleiades` même si il est possible de la distribuer indépendamment, en particulier pour des utilisations du générateur de code `mfront` hors du cadre `pleiades` [Helfer 11, d'Arrigo 12, Olagnon 13, Proix 13].

Le langage C++ a été choisi pour sa grande disponibilité, son caractère libre et pérenne, son interaction avec les langages `C` et `fortran`, et son support de tels paradigmes.

2 VUE D'ENSEMBLE

Nous décrivons dans cette section le contenu de la bibliothèque. Nous commençons par décrire les quelques exécutables fournis par `tfel`, puis nous détaillons les différentes librairies produites.

2.1 EXÉCUTABLES

Actuellement, quatre exécutables sont produits :

1. `tfel-config` qui permet de récupérer des informations sur :
   (a) le répertoire d'installation de `tfel` ;
   (b) les options de compilation (et d'optimisation) que nous conseillons d'utiliser avec le compilateur qui a servi à construire `tfel` ;
   (c) les drapeaux d'avertissement à utiliser que nous conseillons d'utiliser avec le compilateur qui a servi à construire `tfel` ;
2. `tfel-doc` est un utilitaire de description de cas test et de génération d'un guide en LaTeX. Il a notamment été utilisé pour décrire les cas tests de la bibliothèque `tfel` et de l'application `licos` [Helfer 12] ;
3. le générateur de code `mfront` qui fait l'objet d'une documentation particulière [Helfer 13a, Helfer 13b] ;
4. un outil de test élémentaire de lois de comportement mécanique nommé `mtest` [Helfer 14].

2.2 MODULES DISPONIBLES ET LIBRAIRIES PRODUITES

`tfel` est un projet complexe séparé en différents modules (packages en anglais). Nous en dressons ici la liste accompagnée d'une description succincte. Les principaux modules seront chacun décrits dans une section spécifique.

1. le module `TFEL/Math` est de loin le module le plus important de la librairie. Il contient de nombreuses classes représentant les objets mathématiques usuels et différents algorithmes numériques. La plupart de ces classes et de ces algorithmes sont implantés à l'aide de `template`. Ce module fournit plusieurs librairies :
   (a) `libTFELMath.so` contient quelques classes de base pour les autres librairies du module ;
   (b) `libTFELMathCubicSpline.so` contient la gestion des erreurs des classes d'interpolation de données par des splines cubiques ;
   (c) `libTFELMathKriging.so` contient certaines spécialisations des classes d'interpolation de données multi-dimensionnelles par des méthodes de krigeage et la gestion d'erreur associée ;
   (d) `libTFELMathParser.so` contient des classes dédiées à l'interprétation de formules mathématiques complexes à partir de chaînes de caractères ;
2. TFELMaterial contient certains classes utilitaires pour l'écriture de propriétés matériau ou loi de comportement mécanique.
3. le module `TFEL/Exception` contient les bases des classes d'exceptions utilisées dans `tfel`. Les sources de ce module sont compilées dans la librairie `libTFELException.so` ;
4. le module `TFEL/Tests` propose un framework de test similaire à `CppUnit`. Les sources de ce module sont compilées dans la librairie `libTFELTests.so` ;
5. le module `TFEL/Utilities` contient différentes classes utilitaires. Il fournit la librairie `libTFELUtilities.so` ;
6. le module `TFEL/System` propose :

(a) un enrobage C++ de diverses fonctionnalités : `POSIX` [Blaess 11] :
   i. créations et interaction avec des processus ;
   ii. gestion des signaux ;
   iii. créations de répertoires, etc.. ;
   iv. traduction des erreurs `POSIX` en exceptions ;

(b) des facilités d'appels de fonctions définies dans des librairies externes ;

Modules internes D'autres modules existent qui ont essentiellement vocation à être utilisés en interne :

1. le module `TFEL/Metaprogramming` contient des classes utilitaires simplifiant la « méta-programmation à base de template » [Alexandrescu 00, Abrahams 04].
2. le module `TFEL/Typetraits` contient un ensemble de classes permettant de donner de manière non intrusive des informations sur des classes d'objets.
3. le module `TFEL/FSAlgorithm` contient la réécriture de la plupart des algorithmes de la librairie standard pour des objets dont la taille est (raisonnablement) petite et connue à la compilation. L'idée de ce module est tirée des travaux réalisés pour bâtir la librairie `MTL` (Matrix Template Library) [Siek 98]. Son rôle est essentiel pour les performances de la librairie.

Ces modules sont constitués de fichiers d'entête uniquement.

Les modules `TFEL/Metaprogramming` et `TFEL/Typetraits` permettent de faciliter l'emploi de certaines techniques de programmation avancées sur la base du standard C++ de 1998. Elles contiennent des parties très ardues et potentiellement les plus difficiles à maintenir. Leur documentation a été de ce fait été l'objet d'une attention particulière. Nous les avons également construites en surveillant les travaux du comité de standardisation. Ainsi, leur taille devrait largement diminuer avec le portage de `tfel` sur le standard de 2011 dont le support par la compilateur standard est maintenant quasi-complet.

3 DESCRIPTION DU MODULE TFEL/Math

Nous décrivons dans cette section les principales fonctionnalités proposées par le module TFEL/Math.

3.1 OBJETS MATHÉMATIQUES

Dans ce paragraphe, nous détaillons les objets mathématiques qui ont été implantés dans le module TFEL/Math.

3.1.1 Quantités physiques

Nom	Masse	Longueur	Temps	Ampère	Température	Candela	Mole
NoUnit	0	0	0	0	0	0	0
Mass	1	0	0	0	0	0	0
Length	0	1	0	0	0	0	0
Time	0	0	1	0	0	0	0
Ampere	0	0	0	1	0	0	0
Temperature	0	0	0	0	1	0	0
Kelvin	0	0	0	0	1	0	0
Candela	0	0	0	0	0	1	0
Mole	0	0	0	0	0	0	1
InvLength	0	-1	0	0	0	0	0
InvTemperature	0	0	0	0	-1	0	0
Frequency	0	0	-1	0	0	0	0
Velocity	0	1	-1	0	0	0	0
Acceleration	0	1	-2	0	0	0	0
Momentum	1	1	-1	0	0	0	0
Force	1	1	-2	0	0	0	0
Newton	1	1	-2	0	0	0	0
Stress	1	-1	-2	0	0	0	0
StressRate	1	-1	-3	0	0	0	0
Pressure	1	-1	-2	0	0	0	0
Energy	1	2	-2	0	0	0	0
EnergyDensity	1	-1	-2	0	0	0	0
Density	1	-3	0	0	0	0	0
ThermalConductivity	1	1	-3	0	-1	0	0
HeatFluxDensity	1	0	-3	0	0	0	0

TABLEAU 1 : Grandeurs prédéfinies et décomposition suivant les grandeurs de base du système international.

La classe qt représente des nombres réels représentant des quantités physiques. L'idée d'une telle classe a été discutée à plusieurs reprises [Abrahams 04], mais nous sommes particulièrement inspirés du livre de BARTON et NACKMAN [Barton 94].

Il s'agit d'une classe template prenant deux arguments : une classe représentant le type de grandeur représentée et le type numérique utilisé.

Les grandeurs usuelles ont été prédéfinies et sont regroupées au tableau 1 qui donne également leurs décompositions suivant les grandeurs de base du système international. Il est possible de définir d'autres grandeurs par des puissances rationnelles des unités de base à l'aide de la méta-fonction GenerateUnit.

Toutes les opérations mathématiques usuelles ont été définies. Le compilateur autorisera toutes les opérations conservant le sens physique. Ainsi le produit d'une force par une distance à pour résultat une énergie. L'addition d'une masse et d'une température n'est pas autorisée.

Ces vérifications sont faites à la compilation et l'utilisation des `template` et de l'`inlining` fait que les quantités ainsi définies n'ont aucun coût à l'exécution.

Des résultats de type `NoUnit` peuvent être convertis automatiquement dans le type numérique sous-jacent. Ils peuvent être ainsi utilisés dans toutes les fonctions usuelles (logarithmes, exponentielle, etc...).

Note Les quantités ainsi définies sont aujourd'hui de peu d'utilité pratique [1], mais nous pouvons espérer que leur usage se répande à terme.

L'introduction des quantités a cependant eu un grand rôle sur la conception des classes mathématiques de `tfel`. En particulier, nous vérifions que toutes les opérations mathématiques sont valides avec des conventions très strictes [2].

3.1.2 Tenseurs d'ordre 2 symétriques

La classe `stensor` représente des tenseurs d'ordre 2 symétriques. Il s'agit de la principale implantation du concept `StensorConcept` [3]. Ces classes sont particulièrement utilisée pour l'implantation des lois de comportement mécanique.

Cette classe `template` est paramétrée par :

1. la dimension d'espace ;
2. le type numérique utilisé ;
3. une classe de stockage des valeurs.

Stockage des valeurs L'idée d'utiliser différentes classes de stockage s'est avérée avoir peu d'intérêt pratique : seul le stockage par défaut, fourni par la classe `StensorStatic`, est utilisée. Cette classe stocke les valeurs du tenseur dans un tableau de taille fixe et permet d'y accéder directement.

La classe `stensor` n'utilisant aucune méthode virtuelle, l'utilisation de la classe `StensorStatic` assure également que la taille d'un tenseur symétrique est exactement la taille de son tableau de valeur. Il est donc licite de convertir un pointeur de taille fixe vers un pointeur de tenseur symétrique, ce qui est très utilisé en pratique, par exemple dans la gestion de champs de l'architecture `pleiades`.

1. En effet, la plupart des corrélations expérimentales font apparaître des coefficients numériques dont il est difficile de donner l'unité. Il s'agit d'une difficulté pratique que nous allons rapidement illustrer et dont nous donnons une solution acceptable. Cette solution présente certains avantages numériques.

   Par exemple, une loi de NORTON s'écrit :

   $$\dot{p} = A\,\sigma_{eq}^{E}$$

   où apparaissent la déformation plastique cumulée p , la contrainte équivalente σ_{eq}, et deux coefficients A et E. L'unité du coefficient A est pour le moins exotique.

   En fait, la loi de NORTON devrait être écrite ainsi :

   $$\dot{p} = \dot{\varepsilon}_0 \left(\frac{\sigma_{eq}}{\sigma_0}\right)^{E}$$

   où apparaît une contrainte de normalisation σ_0. Cette forme est `a priori` plus complexe, mais en pratique elle peut s'avérer numériquement beaucoup plus précise et efficace. En effet, dans la première version, on évalue une puissance d'un nombre de l'ordre de 10^8 alors que dans la seconde, on évalue la puissance d'un nombre proche de l'unité : les risques de divergence sont bien moindre.

   Cette remarque se généralise assez bien : si l'utilisation de quantité demande un travail supplémentaire aux développeurs de loi, ce travail peut avoir des avantages numériques.

2. Par exemple, il n'est pas possible d'affecter un objet dont le type numérique est en double précision à un objet dont le type numérique est en double précision. De manière générale, il n'est pas possible d'affecter un résultat basé sur un certain type numérique à un résultat sur un type plus petit. La classification des types numériques standard est assuré par une classe nommée `Promote`

3. La notion de concept dans `tfel` est décrite au paragraphe 7.4

Premiers éléments d'optimisation L'utilisation de la classe `StensorStatic` présente plusieurs avantages [4] :

1. aucune allocation dynamique n'est faite si l'objet est défini sur la pile (les allocations dynamiques sont coûteuses) ;
2. les accès aux valeurs sur la pile permettent au compilateur d'effectuer des optimisations très poussées du code ;
3. le compilateur peut contrôler à la compilation si l'accès aux valeurs est valide.

Convention La classe `stensor` utilise une convention particulière, adoptée par certains codes aux éléments finis [Doghri 00, Bornert 01, EDF 13]. Un tenseur d'ordre 2 symétrique est représenté par un vecteur. Les trois premières composantes du vecteur représentent les composantes diagonales du tenseur et les suivants les composantes extra-diagonales multipliées par un facteur $\sqrt{2}$:

$$\begin{pmatrix} s_{00} & s_{01} & s_{02} \\ s_{00} & s_{11} & s_{12} \\ s_{02} & s_{12} & s_{22} \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} s_{00} \\ s_{11} \\ s_{22} \\ \sqrt{2}\, s_{01} \\ \sqrt{2}\, s_{02} \\ \sqrt{2}\, s_{12} \end{pmatrix}$$

Représentation matricielle — Représentation vectorielle

Cette représentation vectorielle est telle que le produit contracté de deux tenseurs d'ordre 2 symétriques est égal au produit scalaire de leurs représentations vectorielles. Dans la suite, nous ne ferons plus de distinction entre la représentation matricielle et la représentation tensorielle d'un vecteur [5].

Dimension	Nombre de composantes
1	3
2	4
3	6

TABLEAU 2 : Nombre de composantes des tenseurs d'ordre 2 symétriques en fonction de la dimension.

Nombre de composantes En fonction de la dimension d'espace, certaines composantes extradiagonales peuvent être nulles et ne sont pas stockées. Le tableau 2 donne le nombre de composantes des tenseurs d'ordre 2 symétriques en fonction de la dimension d'espace.

Accès aux composantes du tenseur L'accès à une composante d'un tenseur d'ordre 2 symétrique se fait par l'opérateur `[]` ou par l'opérateur `()`. Ces opérateurs prennent le numéro de la composante en argument.

Opérations mathématiques Les opérations suivantes sont permises sur les tenseurs d'ordre 2 symétriques :

1. la négation ;
2. multiplication à droite et à gauche par un scalaire ;

4. Pour pouvoir bénéficier de ces avantages, les lois de comportement générées par `mfront` sont implantées par des classes `template` paramétrées par l'hypothèse de modélisation qui fixe la dimension de l'espace. Elles sont instanciées un fois par hypothèse de modélisation supportée.

5. Notons que le code aux éléments finis utilise une autre convention, celle de VOIGT. Cette convention représente également les tenseurs d'ordre 2 symétriques par des vecteurs mais différencie les contraintes et les déformations : les composantes extradiagonales des contraintes ne sont affectées d'aucun facteur, et celles des déformations sont affectées d'un facteur 2. Cette dissymétrie rend pratiquement impossible une écriture systématique des opérations tensorielles.

3. division à droite par un scalaire ;
4. addition et soustraction de deux tenseurs ;
5. produit contracté de deux tenseurs. Pour cette opération, l'opérateur | est utilisé ;
6. produit tensoriel de deux tenseurs. Pour cette opération, l'opérateur ^ est utilisé. Le résultat est une application linéaire sur les tenseurs d'ordre 2 symétriques décrite au paragraphe 3.1.3.

Les règles de priorité du C++ ne permettent pas de reproduire les règles de priorité naturelle pour les produits contracté et tensoriel, l'emploi de parenthèse est parfois nécessaire.

Ces opérations sont implantées par des techniques particulières, appelées `expression templates` [Veldhuizen 95, Vandevoorde 02], qui permettent au compilateur de générer un code optimisée. Ces techniques sont partiellement décrites au paragraphe 7.

Importation et exportation Les codes aux éléments finis utilisent différentes représentations des tenseurs. En particulier, le code aux éléments finis `Cast3M` utilise la convention de VOIGT [CEA 13]. Deux méthodes assurent les conversions, si nécessaire :

1. la méthode `importVoigt` permet de convertir des valeurs utilisant la convention de VOIGT pour les déformations ;
2. la méthode `importTab` permet de convertir des valeurs utilisant la convention de VOIGT pour les contraintes.

Calcul des valeurs propres Les tenseurs d'ordre 2 symétriques sont toujours diagonalisables. La méthode `computeEigenValues` permet de calculer les valeurs propres d'un tenseur d'ordre 2 symétrique. En $1D$, aucun calcul n'est fait. En $2D$, la troisième composante est nécessairement une valeur propre et les deux autres s'obtiennent en trouvant les racines d'un polynômes d'ordre 2. En $3D$, le polynôme caractéristique a trois racines que l'on calcule par la méthode de CARDAN.

Calcul des vecteurs propres La méthode `computeEigenVectors` permet de calculer les valeurs propres et les vecteurs propres d'un tenseur d'ordre 2 symétrique.

Changement de base La méthode `changeBasis` de changer un tenseur de base. Elle prend en argument une matrice de rotation. En $1D$, aucune opération n'est faite (seule matrice identité est valide). En $2D$, on suppose que la troisième direction est invariante. Pour des raisons de performances, aucune vérification de ces hypothèses n'est faite.

Tenseur identité Le tenseur identité, noté $\underline{\mathbf{I}}$, est accessible via la méthode statique `Id`.

Trace La fonction `trace` permet de calculer la trace du tenseur passé en argument, c'est à dire la somme de ses termes diagonaux :

$$trace\,(a) = \sum_{i=0}^{3} a_{ii} = \underline{\mathbf{a}} \colon \underline{\mathbf{I}}$$

Contrainte équivalente au sens de VON MISES La fonction `sigmaeq` retourne la contrainte équivalente σ_{eq} au sens de VON MISES du tenseur passé en argument :

$$\sigma_{eq}\,(\underline{\sigma}) = \sqrt{\frac{3}{2}\,\underline{\mathbf{s}} \colon \underline{\mathbf{s}}} \quad \text{avec} \quad \underline{\mathbf{s}} = \underline{\sigma} - \frac{1}{3}\,trace\,(\underline{\sigma})\,\underline{\mathbf{I}}$$

Contrainte équivalente au sens de TRESCA La fonction `tresca` retourne la contrainte équivalente σ_{eq}^{T} au sens de TRESCA du tenseur passé en argument :

$$\sigma_{eq}^{T}\left(\underline{\sigma}\right) = \max_{i,j i!=j} \left|s_i^p - s_j^p\right|$$

où s_i^p est la i^{e} valeur propre du tenseur $\underline{\sigma}$.

Compatibilité avec la librairie standard La classe `stensor` présente toutes les caractéristiques d'un container de la librairie standard du C++ [Stroustrup 04]. Elle définit :

1. les alias nécessaires, dont les principaux sont `iterator`, `const_iterator`, et `value_type` ;
2. les méthodes `begin`, `end`, `rbegin`, `rend` et `size`.

3.1.3 Applications linéaires sur les tenseurs d'ordre 2 symétriques

L'écriture des lois de comportements mécaniques fait intervenir des tenseurs d'ordre 4, c'est à dire des applications linéaires qui s'appliquent sur des tenseurs d'ordre 2.

Dans ce paragraphe, nous nous intéressons plus particulièrement aux tenseurs d'ordre 4 qui préservent la symétrie des tenseurs d'ordre 2 symétriques. Ils sont représentés par la classe `st2tost2`.

La classe `st2tost2` est une classe `template` paramétrée par :

1. la dimension d'espace ;
2. le type numérique utilisé.

Représentation matricielle Les tenseurs d'ordre 4 sont caractérisés par des composantes à quatre indices. En s'appuyant sur la représentation vectorielle des tenseurs d'ordre 2, une représentation matricielle est également possible : le résultat de l'application d'un tenseur d'ordre 4 sur un tenseur d'ordre 2 est égal au produit de leur représentation matricielle et vectorielle respectivement.

Stockage des valeurs La classe `st2tost2` stocke ses valeurs dans un tableau de taille fixe. La classe `st2tost2` n'utilise aucune méthode virtuelle : sa taille est exactement celle de son tableau de valeurs.

Ces choix présentent les mêmes avantages que pour les tenseurs d'ordre 2 symétriques :

1. aucune allocation dynamique n'est faite si l'objet est défini sur la pile ;
2. les accès aux valeurs sur la pile permettent au compilateur d'effectuer des optimisations très poussées du code ;
3. il est licite de convertir un pointeur vers un tableau de valeurs vers un pointeur d'objet de type `st2tost2` et inversement ;
4. le compilateur peut contrôler à la compilation si l'accès aux valeurs est valide.

3.1.4 Tenseurs d'ordre 2

La classe `tensor` implémente des tenseurs d'ordre 2 non symétriques.

Fonctions utiles Différentes fonctions sont les tenseurs d'ordre 2 ont été implantées :

1. la fonction `det` calcule le déterminant d'un tenseur ;
2. la fonction `computeRightCauchyGreenTensor` calcule le tenseur de CAUCHY-GREEN droit C d'un tenseur F :
$$C = F^T \cdot F$$
Le résultat est un tenseur d'ordre 2 symétrique de type `stensor`.
3. la fonction `computeLeftCauchyGreenTensor` calcule le tenseur de CAUCHY-GREEN gauche B d'un tenseur F :
$$B = F \cdot F^T$$
Le résultat est un tenseur d'ordre 2 symétrique de type `stensor`.
4. la fonction `syme` symétrise un tenseur. Le résultat est un tenseur d'ordre 2 symétrique.

3.1.5 Vecteurs

Deux types de vecteurs sont implantés :

1. des vecteurs de petite taille connue à la compilation. Ils sont implantés par la classe `tvector` ;
2. des vecteurs de taille quelconque. Ils sont implantés dans la classe `vector`

Ces deux classes implémentent le concept `VectorConcept`[6].

Vecteurs de petite taille connue

Vecteurs quelconques

3.1.6 Matrices

Deux types de matrices sont implantés :

1. des matrices de petite taille connue à la compilation. Ils sont implantés par la classe `tmatrix` ;
2. des matrices de taille quelconque. Ils sont implantés dans la classe `matrix`

Ces deux classes implémentent le concept `MatrixConcept`[7].

Matrices de petite taille connue

Matrices quelconques

6. La notion de concept dans `tfel` est décrite au paragraphe 7.4
7. La notion de concept dans `tfel` est décrite au paragraphe 7.4

3.1.7 Tableaux multi-dimensionnels

3.1.8 Objets composites

3.2 ALGORITHMES MATHÉMATIQUES

3.2.1 Résolution de systèmes linéaires par un décomposition `LU`

Décomposition `LU`

Résolution par descente/remontée

Calcul de l'inverse d'une matrice

3.2.2 Résolution de systèmes linéaires par un décomposition `QR`

Décomposition `QR` Toute matrice A de format (M, N) admet une décomposition de la forme :

$$A = Q\,R$$

où :
1. Q est une matrice orthogonale de format (M, M) ;
2. R est une matrice triangulaire supérieure de format M, N.

Une telle décomposition peut être obtenue par une série de transformations de HOUSEHOLDER. De telles transformations sont représentées par des matrices de la forme :

(1) $$H_i = I + \frac{1}{\beta_i}\, v_i^T\ v_i$$

où β est un coefficient de normalisation.

À la fin de la décomposition, la matrice Q est le produit ces matrices :

(2) $$Q = H_1 \cdots H_n$$

L'algorithme utilisé pour la décomposition de la matrice est extrait du livre de P. Lascaux et R. Théodor [Lascaux 94].

La classe `QRDecomp` réalise cette factorisation par la méthode de HOUSEHOLDER et propose différentes méthodes pour travailler avec cette décomposition :

1. la méthode `exe` réalise la décomposition de la matrice. Cette méthode s'appuie sur un vecteur auxiliaire qui contient les éléments diagonaux de la matrice R (voir paragraphe précédent). La décomposition est faite sur place, c'est à dire que la matrice A est utilisée pour stocker les matrices Q et R. Deux vecteurs supplémentaires, correctement dimensionnés, sont nécessaires pour stocker les termes diagonaux de R et les coefficients de normalisation qui interviennent dans la normalisation des vecteurs β définissant les transformations de HOUSEHOLDER ;
2. la méthode `tq_product` réalise le produit de la matrice Q^T (transposée de la matrice Q) et d'un vecteur v.
3. la méthode `back_substitute` résout le problème :

$$R\,x = b$$

Stockage de la décomposition Les termes diagonaux de R sont stockés dans un vecteur annexe. Les termes non situés sur la diagonale de matrice R est stockés dans la partie triangulaire de la matrice A. La matrice Q n'est pas stockée explicitement, mais on stocke dans partie inférieure de A les vecteurs v_i qui apparaissent dans les transformations de HOUSEHOLDER (1). Les coefficients de normalisation $beta_i$ sont stockés dans un vecteur annexe.

Application de la décomposition QR à la résolution de systèmes linéaires . Si A est une matrice, b un vecteur. A se décompose en une matrice orthogonale Q et matrice triangulaire supérieure R.

La matrice Q étant orthogonale, son inverse est sa égale à sa transposée.

Le problème linéaire $A\,x = b$ est donc équivalent au problème :

$$R\,x = y \quad \text{avec} \quad y = Q^T\ b \tag{3}$$

La solution du problème (3) peut être obtenue par les opérations suivantes :

1. décomposition de la matrice A par la méthode `exe` ;
2. calcul du vecteur y par la méthode `tq_product` ;
3. inversion du système triangulaire $R\,x = y$ par la méthode `back_substitute`.

À la fin de ses opérations, le vecteur b contient la solution du problème (3) et la matrice A contient les matrices Q et R.

3.2.3 Résolutions de systèmes d'équations non linéaires

Algorithmes de NEWTON

Algorithmes de BROYDEN

3.2.4 Intégration des systèmes différentiels par des méthodes de RUNGE-KUTTA

3.2.5 Interpolation de données par splines cubiques

3.2.6 Interpolation de données par krigeage

3.2.7 Identifications de paramètres par la méthode de LEVENBERG-MARQUARDT

3.3 ÉVALUATION D'EXPRESSIONS MATHÉMATIQUES

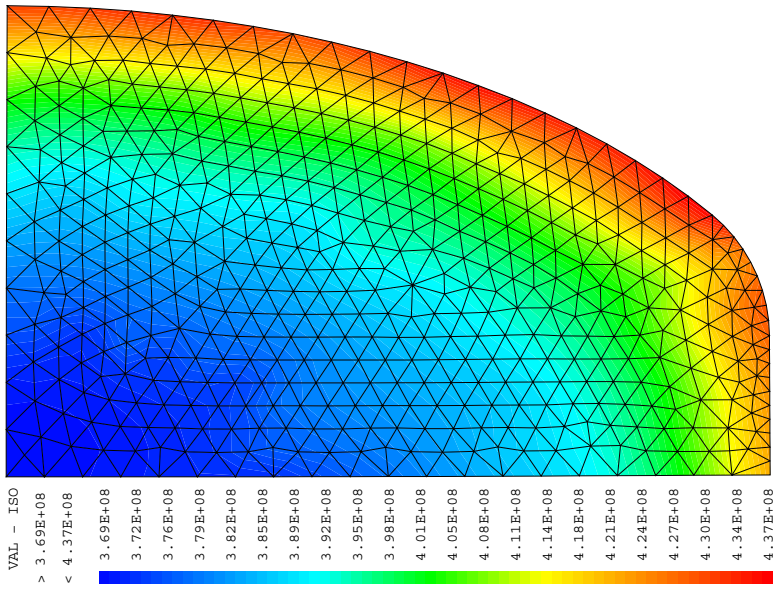

FIGURE 1 : Interpolation des puissances calculées par le code `TRIPOLI` sur un maillage utilisé par un calcul thermo-mécanique effectué avec le code `licos`.

4 DESCRIPTION DU MODULE TFEL/Material

5 DESCRIPTION DU MODULE TFEL/System

5.1 GESTION DES LIBRAIRIES EXTERNES

5.2 GESTION DES PROCESSUS

5.3 GESTION DES SIGNAUX

6 DESCRIPTION DU MODULE TFEL/Utilities

6.1 LA CLASSE GenType

6.2 LECTURE DE FICHIER TEXTE

6.3 LECTURE D'UN JEU DE DONNÉES

7 QUELQUES TECHNIQUES DE PROGRAMMATION SOUS-TENDANT LA LIBRAIRIE DE CALCULS TENSORIELS DE TFEL

Nous décrivons ici quelques techniques de programmation sous-tendant la bibliothèque de calculs tensoriels de `tfel`. Ces techniques, aux quels de nombreux travaux ont été consacrés, se différencient notamment de techniques plus traditionnelles et ont été développées pour combiner simplicité d'utilisation par la surcharge des opérateurs mathématiques usuels et performances optimales, au moins égales à celles obtenues en `fortran`.

7.1 UTILISATION DE LA MÉMOIRE

Le nombre de composantes d'un tenseur, et donc la taille mémoire qu'il utilise, dépend *a priori* de la dimension : un tenseur a 3 composantes en $1D$, 4 composantes en $2D$ et 6 composantes en $3D$.

Étant de petite taille, la zone mémoire associée peut être initialisée sur la pile (nommée `stack` en anglais), ou sur le tas (nommé `heap`). L'allocation sur le tas est coûteuse et ne peut être utilisée pour des petits calculs, tels que ceux utilisés pour l'intégration des lois de comportement, sans réduire considérablement les performances. L'allocation sur la pile est en comparaison sans coût. Elle permet par ailleurs une meilleure localisation des données (et donc des optimisations par le compilateur) et est compatible sans précautions particulières avec une utilisation des exceptions.

L'allocation sur la pile peut se faire en stockant de manière systématique un tableau de taille 6 qui est la taille maximale des tenseurs quelque soit la dimension. Il est alors nécessaire de stocker la dimension d'espace dans le tenseur. Toutes les opérations tensorielles doivent faire appel à des boucles sur allant de 0 à la taille du tenseur. Cette méthode présente plusieurs désavantages :

1. la définition d'un champ de tenseur, où l'on attribue un tenseur à chaque point de discrétisation d'un domaine géométrique, conduit à une gaspillage de mémoire ;
2. il n'est pas possible au compilateur de dérouler ces boucles et d'optimiser les opérations tensorielles ;
3. le compilateur ne peut vérifier que l'accès aux données est valide, c'est à dire qu'un indice trop grand n'est pas utilisé. Une telle vérification peut être faite par des tests, mais cela alourdit le code et ralentit l'exécution ;
4. il est nécessaire de passer la dimension d'espace au constructeur des objets, ce qui alourdit l'écriture.

Nous avons choisi une solution qui nous est apparue plus avantageuse et consistant à paramétrer les tenseurs par la dimension d'espace : un tenseur $1D$ est un objet différent d'un objet $2D$. Ce choix permet d'utiliser un ensemble de techniques de programmation générique permettant par exemple de « dérouler les boucles » à la compilation et au compilateur de vérifier notre code[8].

Ce paramétrage par la dimension d'espace est masqué dans `mfront` : une loi de comportement est en fait paramétrée par la dimension d'espace et est spécialisée pour chaque dimension d'espace, produisant un code optimisé dans chaque cas (au détriment du temps de compilation et de la taille du code généré, mais ces aspects ne sont pas problématiques pour les lois de comportement).

7.2 CHOIX DU TYPE NUMÉRIQUE

L'ensemble des objets mathématiques de `tfel` sont également paramétrés par le type numérique utilisé.

8. De manière générale, nous avons privilégié dans `tfel` des techniques de programmation permettant de détecter au plus tôt, c'est à dire dès la phase de compilation, des erreurs qui ne sont détectables dans d'autres langages qu'à l'exécution.

Ce paramétrage du type numérique est masqué dans `mfront`. `mfront` propose d'utiliser un type opaque `real` pour les opérations mathématiques. Dans le cas de l'interface `umat`, la loi de comportement est spécialisée en utilisant un nombre flottant en double précision. D'autres codes peuvent faire des choix différents : le code de résolution par transformées rapides développé au sein du projet `pleiades` permet de choisir le type numérique à utiliser à la compilation.

7.3 ÉLIMINATION DES OBJETS TEMPORAIRES

Il est classique de vanter, pour des applications scientifiques, la possibilité dans le langage C++ de surcharger les opérateurs mathématiques afin de rapprocher l'écriture du code de l'écriture mathématique usuelle.

Une implantation « naïve » de la surcharge des opérateurs consiste à écrire, dans le cas particulier de l'addition, que la somme de deux tenseurs est un tenseur. Ainsi, la somme `d` de trois tenseurs `a`, `b` et `c` s'écrit :

$$\texttt{d = a + b + c}$$

Cette expression est interprétée par le compilateur en introduisant des objets temporaires, résultats des opérations prises deux à deux :

$$\texttt{d = a + b + c} \quad \Rightarrow \quad \left\{ \begin{array}{ll} \texttt{tmp1} & = \texttt{a} + \texttt{b} \\ \texttt{tmp2} & = \texttt{tmp1} + \texttt{c} \\ \texttt{d} & = \texttt{tmp3} \end{array} \right.$$

L'évaluation et l'affectation des ces temporaires conduit à trois boucles (qui peuvent être éventuellement déroulées), là où une doit suffire.

Une solution à cela est d'utiliser une évaluation paresseuse des opérations : le résultat de la somme de deux tenseurs étant un objet particulier « résultat de la somme de deux tenseurs » dont le rôle est de porter l'information des opérations à effectuer jusqu'au moment de leur évaluation effective, c'est à dire jusqu'au moment de l'affectation au tenseur `d`. Le résultat de la somme de trois tenseurs est un objet du type « résultat de la somme du résultat de la somme de deux tenseurs et d'un tenseur ».

Nous utilisons le compilateur pour générer en arrière plan de tels objets (en nous appuyant sur la notion de concept développée au paragraphe suivant) et les éliminer au moment de l'évaluation effective du résultat. Cette technique, appelée `expression template`, couplée aux techniques de déroulement de boucles, permet de transformer l'addition de trois tenseurs à du code équivalent à :

$$\texttt{d = a + b + c} \quad \Rightarrow \quad \left\{ \begin{array}{ll} d[0] & = \texttt{a}[0] + \texttt{b}[0] + \texttt{c}[0] \\ d[1] & = \texttt{a}[1] + \texttt{b}[1] + \texttt{c}[1] \\ d[2] & = \texttt{a}[2] + \texttt{b}[2] + \texttt{c}[2] \end{array} \right.$$

si les tenseurs `a`, `b`, `c` et `d` sont des tenseurs $1D$.

Le moteur d'`expression template` de `tfel` se compare favorablement, en terme de fonctionnalités, à toutes les implantations publiées à notre connaissance, en particulier car nous avons développé une notion de concept qui est maintenant décrite.

7.4 NOTION DE CONCEPT

Nous avons décrit brièvement le moteur d'`expression template` qui sous-tend les opérations mathématiques dans `tfel`. Chaque opération conduit à des objets dont le type est généré automatiquement par le compilateur. D'un point de vue pratique, il est nécessaire que ses objets se comportent comme des tenseurs

« normaux » afin de pouvoir les utiliser dans des fonctions tensorielles usuelles (trace, contrainte équivalente, etc..).

Il pourrait être tenté d'utiliser une classe mère abstraite et des méthodes virtuelles associées. Cette approche est incompatible avec les choix de conception précédent :

1. il est nécessaire d'allouer les objets sur le tas et de manipuler des pointeurs ce qui est à la fois inefficace et incompatible avec une écriture proche de l'écriture mathématique usuelle ;
2. l'utilisation de méthodes virtuelles ruine toutes les optimisations que nous avons cherchées à mettre en place dans les paragraphes précédents.

Une autre technique d'abstraction est donc nécessaire. De nombreuses recherches ont permis de dégager la notion de « concept ». Cette notion aurait dû être un des piliers du futur standard du langage C++ mais son introduction a été différée par manque de maturité.

`tfel` propose une notion de « concept » beaucoup moins étendue et générale que celle proposée pour le futur standard mais qui est basée sur le standard actuel. La notion de concept dans `tfel` repose sur la combinaison de différentes techniques de programmation relativement pointues :

1. une classe `S` implante la notion de concept en dérivant de la classe `StensorConcept<S>` : `S` hérite d'une classe dont elle est paramètre. Cette technique, appelée `Curiously Recurring Template Pattern`, a été énormément utilisée en programmation générique pour obtenir une flexibilité équivalente aux méthodes virtuelles sans en payer le coût ;
2. une classe à deux paramètres `Implements` qui contient une variable statique `cond`. Si une `S` implante le concept `StensorConcept`, la variable statique `Implements<S,StensorConcept>::cond` vaut `true`, `false` dans le cas contraire. L'implantation effective de la classe `Implements` repose sur les capacités d'introspection de l'opérateur `sizeof` ;
3. le recours au principe `SFINAE`, `Substitution Failure Is Not An Error` pour filtrer les paramètres valides d'une fonction. Ce principe stipule que lorsque le compilateur examine la définition d'une fonction paramétrée pour savoir si elle applicable dans une contexte particulier, si la substitution du paramètre conduit à un échec, alors cette fonction est simplement *éliminée* de la liste des candidats *sans causer d'erreur*.

   L'application de ce principe se fait dans `tfel` par la classe `EnableIf` qui paramétrée par une variable booléenne et une type `T`. La classe `EnableIf<true,T>` est implantée et contient un alias nommé `type` égal à `T`. La classe `EnableIf<false,T>` n'est jamais définie. Ainsi, l'alias `EnableIf<cond,T>::type` n'est défini que si la variable `cond` vaut `true`.

   Ces préliminaires permettent de comprendre la déclaration de la fonction `trace` dans `tfel` :

```
template<class T>
TFEL_MATH_INLINE
typename tfel : :meta : :EnableIf<
  tfel : :meta : :Implements<T,StensorConcept> : :cond,
  typename StensorTraits<T> : :NumType
> : :type
trace(const T&) ;
```

   L'utilisation de la classe `EnableIf` conduit à ce que cette fonction ne puisse être appliquée que si la classe `T` implante le concept `StensorConcept`. Ce filtre permet d'éviter un conflit possible avec la déclaration de la fonction `trace` d'une matrice :

```
template<class T>
TFEL_MATH_INLINE
typename tfel : :meta : :EnableIf<
  tfel : :meta : :Implements<T,MatrixConcept> : :cond,
  typename MatrixTraits<T> : :NumType
> : :type
trace(const T&) ;
```

L'application de toutes ces techniques permet au compilateur de traduire la trace de la somme de deux tenseurs `s1` et `s2` en un code équivalent à :

$$\texttt{trace(s1+s2)} \quad \Rightarrow \quad \texttt{s1}[0] + \texttt{s2}[0] + \texttt{s1}[1] + \texttt{s2}[1] + \texttt{s1}[2] + \texttt{s2}[2]$$

8 DOCUMENTATION INFORMATIQUE

tfel est développé en C++ sur la base du standard de 1998. Nous détaillons dans ce paragraphe quels sont les plate-formes et les compilateurs supportés.

Ces différents points permettent d'apprécier la qualité logicielle de tfel.

8.1 PLATE-FORMES SUPPORTÉES

8.1.1 Systèmes unix

Sous unix, nous nous sommes appuyés sur la norme POSIX pour réaliser les appels système [Blaess 11]. Le principal utilisé est linux et des tests de portabilité ont été effectués sur les systèmes Solaris et FreeBSD.

La compatibilité POSIX permet également de compiler tfel dans un environnement cygwin[9]. Pour différentes raisons, un portage natif sous Windows est préférable et l'utilisation de tfel dans un environnement cygwin est amené à tomber en désuétude.

8.1.2 Systèmes Windows

Il est possible de compiler tfel pour les systèmes Windows à l'aide de la suite msys.

8.2 COMPILATEURS SUPPORTÉES

Compilateur GNU Les compilateurs de la suite gcc et développé dans le cadre du projet GNU sont les compilateurs disponibles par défaut sur les distributions linux.

Toutes les versions du compilateur depuis la version 3.4 sont supportées. Les versions antérieures ne supportent pas de manière satisfaisantes la norme 98.

Compilateur Clang Le compilateur

Compilateur Intel

Compilateur Pathscale

8.3 COMPILATEURS NON SUPPORTÉES

Différents tests ont été menés avec des compilateurs

SunStudio dans la version 12.3. Visual C++

9. cygwin fournit une surcouche POSIX au-dessus des systèmes Windows

9 NOTICE D'INSTALLATION

tfel propose aujourd'hui deux méthodes de compilation et d'installation. La première est basée sur le « moteur de production » cmake. Cette méthode est aujourd'hui conseillée. La seconde est basée sur les traditionnels autotools.

9.1 COMPILATION, INSTALLATION ET GÉNÉRATION DE BINAIRES À L'AIDE DE CMAKE

L'installation de tfel nécessite une version supérieure à 2.8 de cmake.

À partir du répertoire racine des sources, il est conseillé de créer un sous-répertoire pour la construction des binaires :

```
$ mkdir build
$ cd build
```

Préparation de la compilation La commande cmake est invoquée pour préparer la compilation :

```
$ cmake ../ -DCMAKE_INSTALL_PREFIX=... -DCMAKE_BUILD_TYPE="Release" [options]
```

La variable CMAKE_INSTALL_PREFIX permet de préciser le répertoire d'installation.

La variable CMAKE_BUILD_TYPE précise le type de compilation souhaitée. Deux valeurs sont possibles : Release (version de production) et Debug (version de développement).

Pour certains systèmes, il est également possible de préciser la variable LIB_SUFFIX pour modifier le nom du répertoire d'installation des librairies. Ainsi, pour être compatible avec les conventions des distributions Mandriva (entre autres), on utilisera l'option -DLIB_SUFFIX=64.

Les options suivantes sont disponibles :

1. -DENABLE-STATIC=ON/OFF, qui demande la compilation de librairies statiques en plus de librairies dynamiques. Par défaut, cette option est désactivée ;
2. -Denable-fortran=ON/OFF, qui permet d'activer ou de désactiver la compilation de l'interface fortran. Par défaut, cette option est désactivée ;
3. -Denable-python=ON/OFF, qui permet d'activer ou de désactiver la compilation de l'interface python. Par défaut, cette option est activée si un interpréteur python adéquat est trouvée ;
4. -Denable-aster=ON/OFF, qui permet d'activer ou de désactiver la compilation de l'interface aster. Par défaut, cette option est désactivée ;
5. -Dlocal-castem-header=ON/OFF, qui permet d'activer ou de désactiver la compilation des interfaces castem (propriétés matériau) et umat (loi de comportement) sans utiliser une installation de Cast3M. tfel fournira alors sa propre version du fichier d'entête castem.h. Par défaut, cette option est désactivée ;

Cette phase de préparation va automatiquement rechercher :

1. un installation valide de Cast3M, si l'option -Dlocal-castem-header n'a pas été spécifiée. Cette recherche peut être facilitée en définissant la variable CASTEM_INSTALL_PATH par l'option -DCASTEM_INSTALL_PATH=... ou en définissant une variable d'environnement CASTEMHOME. Si la recherche réussie, les interfaces castem (propriétés matériau) et umat (loi de comportement) seront construites ;
2. une installation valide de gnuplot. Si la recherche réussie, l'interface gnuplot (propriétés matériau) est construite ;
3. une installation valide de doxygen ;

4. une installation valide de LaTeX2e ;

Il est possible de préciser le compilateur à utiliser par les variables d'environnement `CC` (compilateur `c`), `CXX` (compilateur C++), `LD` (éditeur de liens).

Enfin, cette phase de préparation va tenter de trouver les options de compilation optimales ainsi que des drapeaux de compilation assez contraignants. Ces mêmes options seront utilisées par `mfront` pour la compilation des sources générées [10].

Compilation La compilation de `tfel` est lancée par la commande `make` :

```
$ make -j x
```

où `x` est le nombre de processeurs affectés à cette

Installation des binaires Les binaires sont installés par la commande :

```
$ make install
```

Exécution des tests La base des cas test de `tfel` peut être lancée par la commande suivante :

```
$ make check
```

Génération de la documentation La documentation de `tfel` est générée par la commande :

```
$ make doc
```

Cette documentation se décompose en deux parties :

1. une documentation informatique générée par `doxygen` (si cet outil est disponible). Cette documentation peut être générée indépendamment par la commande :
   ```
   $make doc-html
   ```
2. la présente documentation, au format `pdf`, si LaTeX2e est disponible. Cette documentation peut être générée indépendamment par la commande :
   ```
   $make doc-pdf
   ```

Installation de la documentation La documentation de `tfel` est installée par la commande :

```
$ make doc-install
```

La documentation informatique générée par `doxygen` peut être installée par la commande :

```
$ make doc-html-install
```

Les différentes documentations `pdf` peuvent être installées par la commande :

```
$ make doc-pdf-install
```

10. Pour connaître les options retenues, il est possible d'interroger la commande `tfel-config` :

```
$ #retourne les avertissements retenus
$ tfel-config -warning
$ #retourne les options d'optimisation retenues
$ tfel-config -oflags
```

Installation des fichiers de tests Il est possible d'installer les fichiers de tests par la commande suivante :

```
$ make tests-install
```

Génération de paquets RPM Des paquets RPM, utilisés par les distributions `linux Red Hat` et `Mandriva` (entre autres) peuvent être générés par la commande :

```
$cpack -G RPM
```

Génération de paquets DEB Des paquets DEB, utilisés par les distributions `linux Debian` et `Ubuntu` (entre autres) peuvent être générés par la commande :

```
$cpack -G DEB
```

Génération d'installateur `Windows` Il est possible de créer un installateur pour `Windows` à l'aide de l'outil NSIS[11] :

```
$cpack -G NSIS
```

9.2 COMPILATION, INSTALLATION ET GÉNÉRATION DE BINAIRES À L'AIDE DES AUTOTOOLS

Si les sources ont été récupérées à partir de la gestion de configuration, il est nécessaire d'initialiser l'environnement de construction des binaires. Pour cela, il faut taper, dans le répertoire racine des sources, la commande suivante :

```
$ ./bootstrap.sh
```

Il est conseillé de créer un sous-répertoire pour la construction des binaires :

```
$ mkdir build
$ cd build
```

Préparation de la compilation La commande `cmake` est invoquée pour préparer la compilation :

```
$ ../configure --prefix=... [options]
```

L'option `--prefix` permet de préciser le répertoire d'installation.

Les options suivantes sont disponibles :

1. `--enable-production`, qui permet de produire une version optimisée (choix par défaut) ;
2. `--enable-debug`, qui permet de produire une version de développement ;
3. `--enable-tests`, qui permet d'activer la compilation des cas test ;
4. `--enable-fortran`, qui permet d'activer ou de désactiver la compilation de l'interface `fortran`. Par défaut, cette option est désactivée ;

11. `Nullsoft Scriptable Install System` est un produit open-source disponible à l'adresse :

`http://nsis.sourceforge.net/`

5. `--enable-python`, qui permet d'activer ou de désactiver la compilation de l'interface `python`. Par défaut, cette option est activée si un interpréteur `python` adéquat est trouvée ;
6. `--enable-aster`, qui permet d'activer ou de désactiver la compilation de l'interface `aster`. Par défaut, cette option est désactivée ;
7. `--local-castem-header`, qui permet d'activer ou de désactiver la compilation des interfaces `castem` (propriétés matériau) et `umat` (loi de comportement) sans utiliser une installation de `Cast3M`. `tfel` fournira alors sa propre version du fichier d'entête `castem.h`. Par défaut, cette option est désactivée ;
8. `--with-castem=xxx` qui permet de préciser le répertoire d'installation de `Cast3M` ;

Cette phase de préparation va automatiquement rechercher :

1. un installation valide de `Cast3M`, si l'option `--local-castem-header` n'a pas été spécifiée. Cette recherche peut être facilitée en utilisant l'option `--with-castem=xxx` ou ou en définissant une variable d'environnement `CASTEMHOME`. Si la recherche réussie, les interfaces `castem` (propriétés matériau) et `umat` (loi de comportement) seront construites ;
2. une installation valide de `gnuplot`. Si la recherche réussie, l'interface `gnuplot` (propriétés matériau) est construite ;
3. une installation valide de `doxygen` ;
4. une installation valide de LaTeX2e ;

Il est possible de préciser le compilateur à utiliser par les variables d'environnement `CC` (compilateur `c`), `CXX` (compilateur C++), `LD` (éditeur de liens).

Enfin, cette phase de préparation va tenter de trouver les options de compilation optimales ainsi que des drapeaux de compilation assez contraignants. Ces mêmes options seront utilisées par `mfront` pour la compilation des sources générées [12].

Compilation La compilation de `tfel` est lancé par la commande `make` :

```
$ make -j x
```

où `x` est le nombre de processeurs affectés à cette

Installation des binaires Les binaires sont installés par la commande :

```
$ make install
```

Exécution des tests La base des cas test de `tfel` peut être lancée par la commande suivante :

```
$ make check
```

12. Pour connaître les options retenues, il est possible d'interroger la commande `tfel-config` :

```
$ #retourne les avertissements retenus
$ tfel-config -warning
$ #retourne les options d'optimisation retenues
$ tfel-config -oflags
```

Installation de la documentation La documentation de `tfel` est installée par la commande :

```
$ make doc-install
```

Cette documentation se décompose en deux parties :

1. une documentation informatique générée par `doxygen` (si cet outil est disponible). Cette documentation peut être générée indépendamment par la commande :
   ```
   $ make doc-html
   ```
2. la présente documentation, au format `pdf`, si LaTeX2e est disponible. Cette documentation peut être générée indépendamment par la commande :
   ```
   $ make doc-pdf
   ```

9.3 Description du répertoire d'installation

Le répertoire d'installation contient quatre répertoires :

1. le répertoire `bin` contient les exécutables produits ;
2. le répertoire `lib` contient les librairies produites ;
3. le répertoire `include` contient les fichiers d'entête des différentes librairies de `tfel`
4. le répertoire `share` contient des tests et des fichiers d'exemple.

Le répertoire `share/mfront/tests/behaviours` contient un ensemble de fichiers de tests `mfront` qui peut servir d'exemple.

10 CALCULS DES DÉRIVÉES DES VALEURS ET VECTEURS PROPRES D'UN TENSEUR SYMÉTRIQUE

Un tenseur symétrique $\underline{\mathbf{s}}$ peut toujours être diagonalisé. Soient $\lambda_1, \lambda_2, \lambda_3$ ses trois valeurs propres et $\vec{e}_1, \vec{e}_2, \vec{e}_3$ les vecteurs propres associés. $\underline{\mathbf{s}}$ se décompose ainsi :

$$\underline{\mathbf{s}} = \sum_{i=1}^{3} \lambda_i \, \vec{e}_i \otimes \vec{e}_i$$

Définissons les tenseurs symétriques suivants :

$$\underline{\mathbf{n}}_{ij} = \begin{cases} \vec{e}_i \otimes \vec{e}_i \text{ si } i = j \\ \frac{1}{\sqrt{2}} \left(\vec{e}_i \otimes \vec{e}_j + \vec{e}_j \otimes \vec{e}_i \right) \text{ si } i \neq j \end{cases}$$

Les tenseurs $\underline{\mathbf{n}}_{ij}$ forment une base orthonormée des tenseurs d'ordre 2 symétriques. Le tenseur $\underline{\mathbf{s}}$ se décompose ainsi :

$$\underline{\mathbf{s}} = \sum_{i=1}^{3} \lambda_i \, \underline{\mathbf{n}}_i \tag{4}$$

Les vecteurs propres de $\underline{\mathbf{s}}$ sont orthornomés :

$$\vec{t}_i \, . \, \vec{t}_j = \delta_{ij}$$

Par différentiation :

$$\mathsf{d}\vec{t}_i \, . \, \vec{t}_j + \mathsf{d}\vec{t}_i \, . \, \vec{t}_j = 0$$

Cette propriété nous permet de montrer une propriété importante des différentielles des tenseurs $\underline{\mathbf{n}}_{ij}$:

$$n_{ii} : \mathsf{d}n_{ii} = 0 \tag{5}$$

D'après l'équation (4), la différentielle du tenseur $\underline{\mathbf{s}}$ s'écrit :

$$\mathsf{d}\,\underline{\mathbf{s}} = \sum_{i=1}^{3} \lambda_i \, \mathsf{d}\,\underline{\mathbf{n}}_{ii} + \underline{\mathbf{n}}_{ii} \, \mathsf{d}\,\lambda_i$$

En utilisant la propriété (5), la projection suivant $\underline{\mathbf{n}}_{ii}$ donne :

$$\mathsf{d}\,\lambda_i = \mathsf{d}\,\underline{\mathbf{s}} : \underline{\mathbf{n}}_{ii}$$

En particulier :

$$\frac{\partial \lambda_i}{\partial \underline{\mathbf{s}}} = \frac{\partial \underline{\mathbf{s}}}{\partial \underline{\mathbf{s}}} : \underline{\mathbf{n}}_{ii} = \underline{\mathbf{n}}_{ii}$$

Nous nous intéressons maintenant aux termes $\dfrac{\partial \underline{\mathbf{n}}_{ii}}{\partial \underline{\mathbf{s}}}$.

On montre que :

$$\frac{\partial \underline{\mathbf{n}}_{11}}{\partial \underline{\mathbf{s}}} = \frac{1}{\lambda_1 - \lambda_2} \underline{\mathbf{n}}_{12} \otimes \underline{\mathbf{n}}_{12} + \frac{1}{\lambda_1 - \lambda_3} \underline{\mathbf{n}}_{13} \otimes \underline{\mathbf{n}}_{13} \tag{6}$$

10.1 APPLICATION AUX FONCTIONS ISOTROPES DE TENSEURS SYMÉTRIQUES

Une fonction $\underline{\mathbf{f}}$ de tenseur d'ordre 2 symétrique est dite *isotrope* si elle vérifie :

$$\underline{\mathbf{f}}\left(\tilde{\underline{\mathbf{s}}}\right) = \widetilde{\underline{\mathbf{f}}\left(\underline{\mathbf{s}}\right)}$$

pour changement de base représenté par $\tilde{}$.

Toute fonction définie par une série entière est isotrope. Ainsi les fonctions logarithme $\log$ et exponentielle $\exp$ sont des fonctions isotropes. Dans ce cas, f désigne à la fois une fonction réelle et une fonction de tenseur d'ordre 2 symétrique.

Une telle fonction peut être évaluée à l'aide des vecteurs propres et des tenseurs propres :

$$f\left(\underline{\mathbf{s}}\right) = \sum_{i=1}^{3} f\left(\lambda_i\right) \underline{\mathbf{n}}_i$$

La dérivée de f par rapport à son argument est alors :

$$\begin{aligned}\frac{\partial f}{\partial \underline{\mathbf{s}}} &= \sum_{i=1}^{3} f'\left(\lambda_i\right) \underline{\mathbf{n}}_i \otimes \underline{\mathbf{n}}_i + \sum_{i=1}^{3} f\left(\lambda_i\right) \frac{\partial \underline{\mathbf{n}}_i}{\partial \underline{\mathbf{s}}} \\ &= \sum_{i=1}^{3} f'\left(\lambda_i\right) \underline{\mathbf{n}}_i \otimes \underline{\mathbf{n}}_i + \frac{f\left(\lambda_1\right) - f\left(\lambda_2\right)}{\lambda_1 - \lambda_2} \underline{\mathbf{n}}_{12} \otimes \underline{\mathbf{n}}_{12} + \frac{f\left(\lambda_1\right) - f\left(\lambda_3\right)}{\lambda_1 - \lambda_3} \underline{\mathbf{n}}_{13} \otimes \underline{\mathbf{n}}_{13} + \frac{f\left(\lambda_2\right) - f\left(\lambda_3\right)}{\lambda_2 - \lambda_3} \underline{\mathbf{n}}_{23} \otimes \underline{\mathbf{n}}_{23}\end{aligned}$$

Cas où deux valeurs propres sont égales Supposons que les valeurs propres λ_1 et λ_2 sont égales. Nous avons alors :

$$\frac{\partial f}{\partial \underline{\mathbf{s}}} = \sum_{i=1}^{3} f'\left(\lambda_i\right) \underline{\mathbf{n}}_i \otimes \underline{\mathbf{n}}_i + f'\left(\lambda_1\right) \underline{\mathbf{n}}_{12} \otimes \underline{\mathbf{n}}_{12} + \frac{f\left(\lambda_1\right) - f\left(\lambda_3\right)}{\lambda_1 - \lambda_3} \left[\underline{\mathbf{n}}_{13} \otimes \underline{\mathbf{n}}_{13} + \underline{\mathbf{n}}_{23} \otimes \underline{\mathbf{n}}_{23}\right]$$

Cas où trois valeurs propres sont égales Supposons que les trois propres λ_1, λ_2 et λ_3 sont égales. Nous avons alors :

$$\frac{\partial f}{\partial \underline{\mathbf{s}}} = f'\left(\lambda_1\right) \left[\sum_{i=1}^{3} \underline{\mathbf{n}}_i \otimes \underline{\mathbf{n}}_i + \underline{\mathbf{n}}_{12} \otimes \underline{\mathbf{n}}_{12} + \underline{\mathbf{n}}_{13} \otimes \underline{\mathbf{n}}_{13} + \underline{\mathbf{n}}_{23} \otimes \underline{\mathbf{n}}_{23}\right]$$

Finalement,

$$\frac{\partial f}{\partial \underline{\mathbf{s}}} = f'\left(\lambda_1\right) \underline{\mathbf{I}}$$

10.2 CAS PARTICULIER DES PARTIES POSITIVES ET NÉGATIVES DES TENSEURS

10.2.1 Cas scalaire

La partie positive $\langle x \rangle_+$ d'un scalaire x est définie par :

$$\langle x \rangle_+ = \begin{cases} 0 & \text{si} \quad x < 0 \\ x & \text{si} \quad x \geq 0 \end{cases}$$

De même, la partie négative $\langle x \rangle_-$ est définie par :

$$\langle x \rangle_- = \begin{cases} x & \text{si} \quad x \geq 0 \\ 0 & \text{si} \quad x > 0 \end{cases}$$

Les parties positive et négative sont différentiables partout, sauf à l'orgine qui constitue un point singulier. Nous proposons d'étendre la définition classique de la dérivée ainsi :

$$\frac{\partial \langle x \rangle_+}{\partial x} = \begin{cases} 0 & \text{si} \quad x < 0 \\ \frac{1}{2} & \text{si} \quad x = 0 \\ 1 & \text{si} \quad x > 0 \end{cases} \quad \text{et} \quad \frac{\partial \langle x \rangle_-}{\partial x} = \begin{cases} 1 & \text{si} \quad x < 0 \\ \frac{1}{2} & \text{si} \quad x = 0 \\ 0 & \text{si} \quad x > 0 \end{cases}$$

10.2.2 Définition dans le cas d'un tenseur

La décomposition spectrale d'un tenseur symétrique est défini ainsi :

$$\underline{\epsilon} = \sum_{i=1}^{3} \lambda_i \, \underline{\mathbf{n}}_i$$

où apparaissent les valeurs propres λ_i et les tenseurs propres $\underline{\mathbf{n}}_i$ (voir la documentation de référence de 'TFEL' pour une définition). Ces tenseurs propres vérifient :

$$\underline{\mathbf{n}}_i : \underline{\mathbf{n}}_j = \delta_{ij}$$

La partie positive d'un tenseur est alors définie ainsi :

$$\langle \underline{\epsilon} \rangle_+ = \sum_{i=1}^{3} \langle \lambda_i \rangle_+ \, \underline{\mathbf{n}}_i$$

Les parties positive et négative sont alors définies ainsi :

$$\langle \underline{\epsilon} \rangle_+ = \sum_{i=1}^{3} \langle \lambda_i \rangle_+ \, \underline{\mathbf{n}}_i$$
$$\langle \underline{\epsilon} \rangle_- = \sum_{i=1}^{3} \langle \lambda_i \rangle_- \, \underline{\mathbf{n}}_i$$

Propriétés Les parties positive et négative ont les propriétés suivantes :

$$\underline{\epsilon} = \langle \underline{\epsilon} \rangle_+ + \langle \underline{\epsilon} \rangle_-$$
$$\langle \underline{\epsilon} \rangle_+ \, : \, \langle \underline{\epsilon} \rangle_- = 0$$
$$\underline{\epsilon} : \underline{\epsilon} = \langle \underline{\epsilon} \rangle_+ \, : \, \langle \underline{\epsilon} \rangle_+ + \langle \underline{\epsilon} \rangle_- \, : \, \langle \underline{\epsilon} \rangle_-$$

Calcul de la dérivée Si toutes les valeurs propres sont différentes, la dérivée de la partie positive d'un tenseur s'écrit :

$$\frac{\partial \langle \underline{\epsilon} \rangle_+}{\partial \underline{\epsilon}} = \sum_{i=1}^{3} \underline{\mathbf{n}}_i \otimes \frac{\partial \langle \lambda_i \rangle_+}{\partial \underline{\epsilon}} + \langle \lambda_i \rangle_+ \, \frac{\partial \underline{\mathbf{n}}_i}{\partial \underline{\epsilon}}$$

Avec les résultats établis précédemment, nous avons :

$$\frac{\partial \langle \underline{\epsilon} \rangle_+}{\partial \underline{\epsilon}} = \sum_{i=1}^{3} \frac{\partial \langle \lambda_i \rangle_+}{\partial \lambda_i} \underline{\mathbf{n}}_i \otimes \underline{\mathbf{n}}_i + \sum_{j \neg i} \frac{\langle \lambda_i \rangle_+}{\lambda_i - \lambda_j} \underline{\mathbf{n}}_{ij} \otimes \underline{\mathbf{n}}_{ij}$$

Cette définition admet une limite bien définie quand deux valeurs propres sont égales. Supposons que λ_1 et λ_2 soient proches. En rapprochant les termes en $\underline{\mathbf{n}}_{12} \otimes \underline{\mathbf{n}}_{12}$ (en utilisant la symétrie rappelée plus haut) et en faisant tendre λ_1 vers λ_2, nous obtenons :

$$\frac{\langle \lambda_2 \rangle_+ - \langle \lambda_1 \rangle_+}{\lambda_2 - \lambda_1} \underline{\mathbf{n}}_{12} \otimes \underline{\mathbf{n}}_{12} \rightarrow_{\lambda_1 \rightarrow \lambda_2} \left. \frac{\partial \langle \lambda \rangle_+}{\partial \lambda} \right|_{\lambda = \lambda_2} \underline{\mathbf{n}}_{12} \otimes \underline{\mathbf{n}}_{12}$$

De même, si trois valeurs propres sont égales, nous avons le résultat suivant :

$$\frac{\partial \langle \underline{\epsilon} \rangle_+}{\partial \underline{\epsilon}} = \frac{\partial \langle \lambda \rangle_+}{\partial \lambda} \underline{\underline{\mathbf{I}}}$$

Pour conclure ce paragraphe, rappelons également quelques résultats classiques :

$$\frac{\partial \text{tr } \langle \underline{\epsilon} \rangle_+^2}{\partial \underline{\epsilon}} = 2 \text{ tr } \langle \underline{\epsilon} \rangle_+$$
$$\frac{\partial \langle \underline{\epsilon} \rangle_+ : \langle \underline{\epsilon} \rangle_+}{\partial \underline{\epsilon}} = 2 \langle \underline{\epsilon} \rangle_+$$

10.3 Dérivées première et seconde de fonctions des valeurs propres d'un tenseur symétrique

Considérons une fonction $f(\lambda_1, \lambda_2, \lambda_3)$ des trois valeurs propres $\lambda_1, \lambda_2, \lambda_3$ d'un tenseur symétrique $\underline{\mathbf{s}}$.

10.3.1 Dérivée première

La dérivée première se calcule simplement :

$$\frac{\partial f}{\partial \underline{\mathbf{s}}} = \sum_{i=1}^{3} \frac{\partial f}{\partial \lambda_i} \underline{\mathbf{n}}_{ii}$$

10.3.2 Dérivée seconde

La dérivée seconde se calcule ainsi :

$$\frac{\partial^2 f}{\partial s \, \partial \underline{\mathbf{s}}} = \sum_{i=1}^{3} \sum_{j=1}^{3} \frac{\partial^2 f}{\partial \lambda_i \, \partial \lambda_j} \underline{\mathbf{n}}_{ii} \otimes \underline{\mathbf{n}}_{jj} + \sum_{i=1}^{3} \frac{\partial f}{\partial \lambda_i} \frac{\partial \underline{\mathbf{n}}_{ii}}{\partial \underline{\mathbf{s}}}$$

Cas $\lambda_1 = \lambda_2$ Dans ce cas, apparaît dans l'expression précédente un terme singulier de la forme :

$$\lim_{\lambda_1 \rightarrow 0} \frac{1}{\lambda_1 - \lambda_2} \left(\frac{\partial f}{\partial \lambda_1} - \frac{\partial f}{\partial \lambda_2} \right)$$

RÉFÉRENCES

[Abrahams 04] ABRAHAMS DAVID et GURTOVOY ALEKSEY. *C++ template metaprogramming : concepts, tools, and techniques from boost and beyond*. Addison-Welsley, Boston, 2004.

[Alexandrescu 00] ALEXANDRESCU ANDREI. *Modern C++ design : applied generic programming and design patterns*. Addison-Wesley, Boston, MA ; London, 2000.

[Barton 94] BARTON JOHN J. et NACKMAN LEE R. *Engineering and Scientific C++ : An Introduction with Advanced Techniques and Examples 1st (first) Edition*. Addison Wesley, 1994.

[Blaess 11] BLAESS CHRISTOPHE. *Développement système sous Linux : [ordonnancement multi-tâche, gestion mémoire, communications, programmation réseau]*. Eyrolles, Paris, 2011.

[Bornert 01] BORNERT MICHEL, BRETHEAU THIERRY et GILORMINI PIERRE. *Homogénéisation en mécanique des matériaux 1 : matériaux aléatoires élastiques et milieux périodiques (Traité MIM, série alliages métalliques, Alliages Métalliques)*. Hermes Science, Paris, 2001.

[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.

[Doghri 00] DOGHRI ISSAM. *Mechanics of deformable solids : linear, nonlinear, analytical, and computational aspects*. Springer, Berlin ; New York, 2000.

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

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

[Helfer 12] HELFER THOMAS. *Cas tests unitaires de la version 1.0 de Licos*. Note technique 12-005, CEA DEN/DEC/SESC/LSC, Avril 2012.

[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 14] 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.

[Lascaux 94] LASCAUX PATRICK et THÉODOR RAYMOND. *Analyse numérique matricielle appliquée à l'art de l'ingénieur. Tome 2, Tome 2,*. Masson, Paris ; Milan ; Barcelone, 1994.

[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.

[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.

[Siek 98] SIEK JEREMY G et LUMSDAINE ANDREW. *The matrix template library : A generic programming approach to high performance numerical linear algebra*. Computing in Object-Oriented Parallel Environments, p 59–70. Springer, 1998.

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

[Vandevoorde 02] VANDEVOORDE D. et JOSUTTIS N.M. *C++ Templates : The Complete Guide*. Addison Wesley, 2002.

[Veldhuizen 95] VELDHUIZEN TODD. *Expression Templates*. C++ Report, 1995, vol 7, p 26–31.

LISTE DES TABLEAUX

LISTE DES FIGURES

INDEX DES CLASSES ET DES FONCTIONS

V