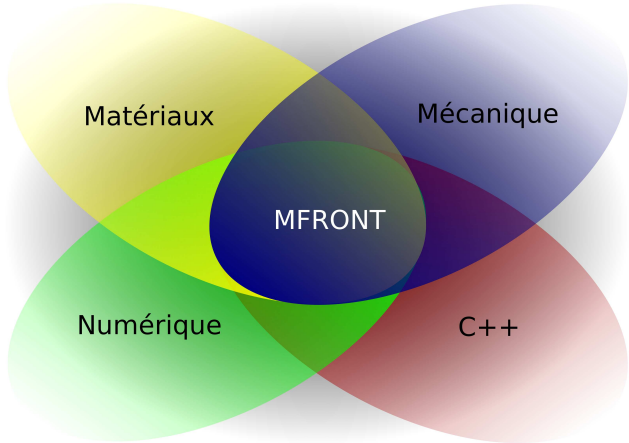
DE LA RECHERCHE À L'INDUSTRIE

cea

www.cea.fr

Lois de comportement mécanique avec `mfront`



— T. Helfer

Sommaire

Lois de comportement mécanique

Exemples

Généralités

Analyseurs spécifiques

Intégration par une méthode explicite (Runge-Kutta)

Intégration implicite

L'interface `umat`

Algorithmes numériques alternatifs

Conclusions

Annexes

Mécanique non linéaire quasi-statique

- équilibre mécanique, $\Delta\vec{\mathbb{U}}$ tel que :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

Mécanique non linéaire quasi-statique

- équilibre mécanique, $\Delta\vec{\mathbb{U}}$ tel que :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

- force interne élémentaire :

$$\begin{aligned}\vec{\mathbb{F}}_i^e &= \int_{V^e} \underline{\sigma}_{t+\Delta t}\left(\Delta\,\underline{\epsilon}^{to}, \Delta\, t\right) : \underline{\mathbf{B}}\, \mathrm{d}V \\ &= \sum_{i=1}^{N^G} \left(\boxed{\underline{\sigma}_{t+\Delta t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_i\right), \Delta\, t\right)} : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right)\right) w_i\end{aligned}$$

où $\underline{\mathbf{B}}$ permet de calculer $\Delta\,\underline{\epsilon}^{to}$ à partir de $\Delta\vec{\mathbb{U}}$

Mécanique non linéaire quasi-statique

- équilibre mécanique, $\Delta\vec{\mathbb{U}}$ tel que :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

- force interne élémentaire :

$$\vec{\mathbb{F}}_i^e = \sum_{i=1}^{N^G} \left(\underline{\sigma}_{t+\Delta t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_i\right), \Delta\, t\right) : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right) \right) w_i$$

- résolution par Newton-Raphson :

$$\Delta\vec{\mathbb{U}}^{n+1} = \Delta\vec{\mathbb{U}}^n - \left(\left.\frac{\partial\vec{\mathbb{R}}}{\partial\Delta\vec{\mathbb{U}}}\right|_{\Delta\vec{\mathbb{U}}^n} \right)^{-1} .\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^n\right) = \Delta\vec{\mathbb{U}}^n - \underline{\underline{\mathbb{K}}}^{-1} .\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^n\right)$$

Mécanique non linéaire quasi-statique

- équilibre mécanique, $\Delta\vec{\mathbb{U}}$ tel que :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

- force interne élémentaire :

$$\vec{\mathbb{F}}_i^e = \sum_{i=1}^{N^G} \left(\boxed{\underline{\sigma}_{t+\Delta t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_i\right), \Delta\, t\right)} : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right) \right) w_i$$

- résolution par Newton-Raphson :

$$\Delta\vec{\mathbb{U}}^{n+1} = \Delta\vec{\mathbb{U}}^{n} - \underline{\underline{\mathbb{K}}}^{-1}.\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^{n}\right)$$

- calcul de la raideur élémentaire :

$$\underline{\underline{\mathbb{K}}}^e = \sum_{i=1}^{N^G} {}^t\underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right) : \boxed{\frac{\partial\Delta\underline{\sigma}}{\partial\Delta\underline{\epsilon}^{to}}\left(\vec{\eta}_i\right)} : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right) w_i$$

où $\dfrac{\partial\Delta\underline{\sigma}}{\partial\Delta\underline{\epsilon}^{to}}$ est la *matrice tangente cohérente*.

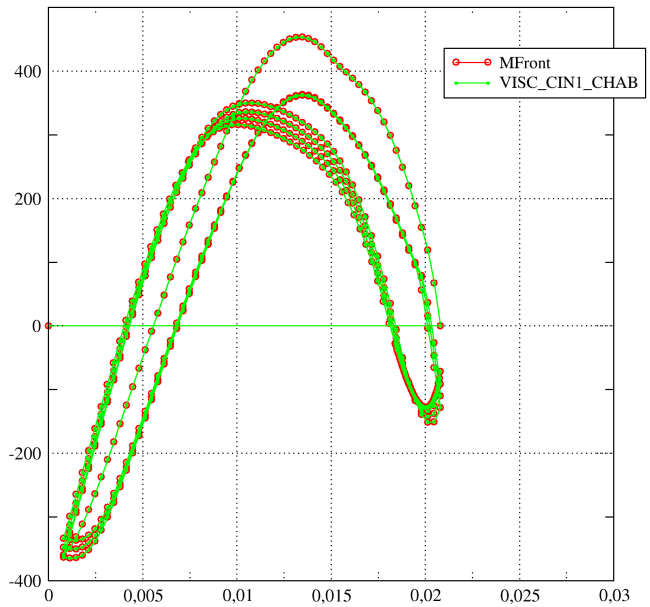
Rôle de la loi de comportement mécanique

- la loi de comportement mécanique doit :
  - calculer les contraintes $\underline{\sigma}$ en fin de pas de temps pour un incrément de déformations $\Delta\,\underline{\epsilon}^{to}$ donné ;
  - calculer l'évolution de variables d'état (dits variables internes) décrivant l'état microstructural du matériau ;
  - fournir une approximation de la matrice tangente cohérente $\dfrac{\partial \Delta \underline{\sigma}}{\partial \Delta \underline{\epsilon}^{to}}$;

Exemples

Exemples - hsnv125d

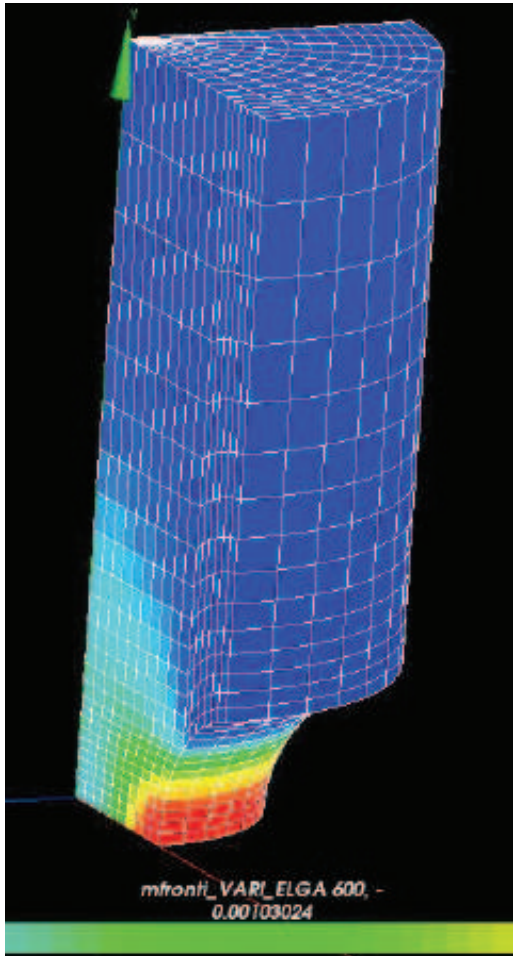
	`mfront` implicite (19 équations)	`VISC_CIN1_CHAB` implicite aster optimisé (1 équation)
Nombre de pas de temps	76	76
Nombre d'itérations de NEWTON	160	151
Temps CPU	4,67s	4,19s



- loi viscoplastique avec deux variables d'écrouissage cinématique non linéaire ;
- cas test très sollicitant :
  - chargement cyclique ;
  - variations importantes de la température dont dépend la plupart des coefficient ;
- intégrée dans `mfront` par une méthode implicite avec jacobienne par perturbation.

Exemples - Hayhurst

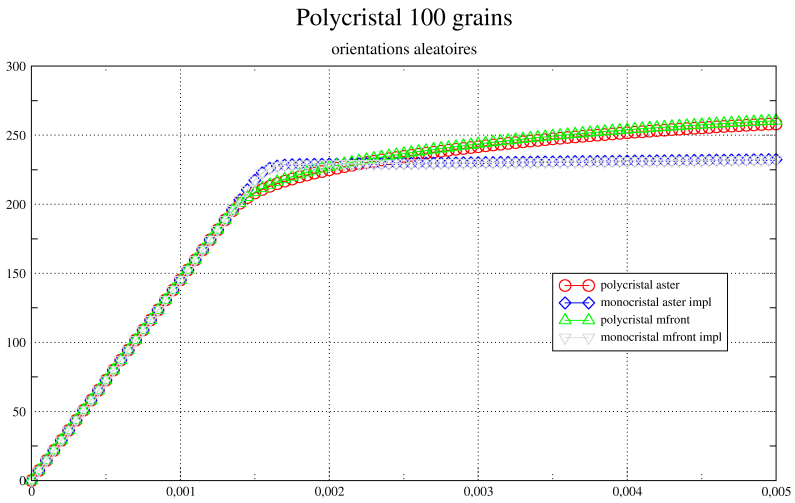
	MFRONT Implicite	MFRONT implicite avec jacobienne numérique	Aster avec jacobienne numérique
Nombre de pas de temps	11	11	11
Nombre d'itérations de NEWTON	37	36	44
Temps CPU	8mn11s	7mn58s	17mn43s
Temps d'intégration de la loi de comportement	2mn26s	2mn21s	10mn51s
Temps d'inversion de la matrice	5mn11s	5mn5s	6mn11s



- loi de fluage tertiaire (avec endommagement)

Exemples - PolyCristal 100 grains

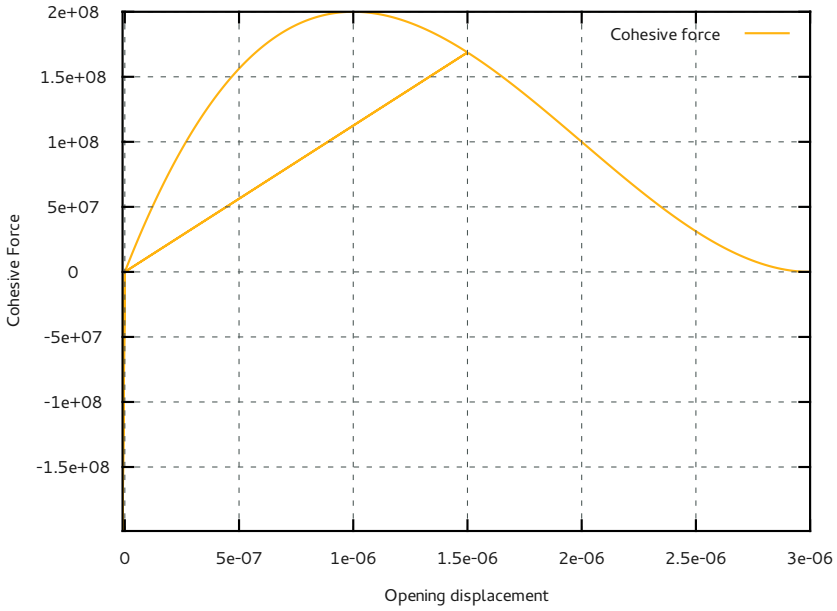
	`mfront` (RK42)	`Aster`
Temps CPU	90s	136s



- 4200 variables internes (!) :
  - 12 systèmes de glissement ;
  - 100 grains ;
- l'intégration implicite est inenvisageable ;
- on voit que l'ordre du Runge-Kutta a un rôle important :
  - l'implantation `Aster` utilise un Runge-Kutta d'ordre 2/1

Modèles de zones cohésives (en cours)

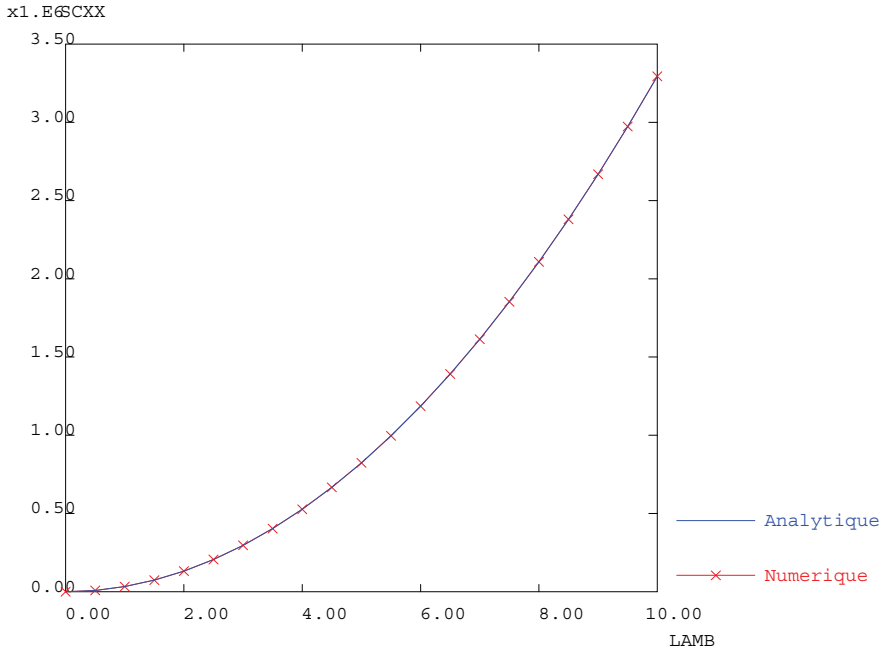
```
@Integrator{
  const real C        = real(27)/real(4);
  // tagential behaviour
  t_t = ks*(u_t+du_t);
  if(computeTangentOperator_){
    Dt_tt = ks*tmatrix<N-1,N-1,real>::Id();
    Dt_tn = Dt_nt = tvector<N-1,real>(real(0));
  }
  // normal behaviour
  if(u_n+du_n<0){
    // compression
    t_n = kn*(u_n+du_n);
    if(computeTangentOperator_){
      Dt_nn = kn;
    }
  } else {
    // traction
    // reduced opening displacement
    const real rod = (u_n+du_n)/delta;
    // previous damage
    const real d_1 = d;
    d   = min(max(d,rod),0.99);
    // damage indicator
    di  = ((d_1>d)&&(du_n>0)) ? 1. : 0.;
    // initial stiffness
    const real K1 = C*smax/delta;
    // secant stiffness
    const real K  = K1*(1-d)*(1-d);
    t_n = K*(u_n+du_n);
    if(computeTangentOperator_){
      if(smt==ELASTIC){
 Dt_nn = K1;
      } else if(smt==SECANTOPERATOR){
 Dt_nn = K;
      } else if(smt==CONSISTENTTANGENTOPERATOR){
 if(d>=0.99){
   Dt_nn = K1*(1-d)*(1-3*d);
 } else {
   Dt_nn = K;
 }
      }
    }
  }
} // end of @Integrator
```



- modèle de Tvergaard ;
- distinction explicite entre le comportement normal et tangentiel ;
- utilisable uniquement avec l'analyseur par défaut ;

Transformations finies (en cours)

```
@Parser DefaultFiniteStrainParser;
@Behaviour IncompressiblePlaneStressMooneyRivlinBehaviour;
@Author T. Helfer;
@Date 19/10/2013;

@ModellingHypothesis PlaneStress;

@MaterialProperty stress C1;
@MaterialProperty stress C2;

@Integrator{
  static const real cste = sqrt(real(2));
  // right Cauch Green Tensor
  Stensor c = computeRightCauchyGreenTensor(F1);
  // specific treatment du to plane stress and incompressibility
  c(2) = 1.; // PlaneStress
  // ("in plane") inverse of b
  const Stensor ci = invert(c);
  // volume change in the plane
  const real Jp = det(c);
  // incompressibility
  c(2) = 1/Jp;
  // checks
  // first invariant of the left Cauchy-Green deformation tensor
  const real i1 = trace(c);
  // second Piola-Kirchhoff stress
  const stress zz = C1 + C2*i1;
  const stress pr = 2*(zz-C2*c(2))*c(2);
  StressStensor s = 2*(zz*Stensor::Id()-C2*c)-pr*ci;
  s(2) = 0.; // Plane stress
  // Cauchy stress
  sig(0) = s(0)*F1(0)*F1(0) + F1(3)*F1(3)*s(1) + cste*F1(0)*F1(3)*s(3);
  sig(1) = s(0)*F1(4)*F1(4) + F1(1)*F1(1)*s(1) + cste*F1(4)*F1(1)*s(3);
  sig(3) = (F1(4)*F1(3)+F1(1)*F1(0))*s(3)+ (F1(4)*F1(0)*s(0) +
F1(1)*F1(3)*s(1)) *cste;
  sig(2) = 0.;
  // no tangent operator yet
  if(computeTangentOperator_){
    string msg("tangent operator not yet available");
    throw(runtime_error(msg));
```



- modèle de Mooney Rivlin incompressible en contraintes planes ;
- exemples inspirés des tests de Laurent Gornet (Centrale Nantes) ;
- utilisable uniquement avec l'analyseur par défaut ;

Guide des cas tests

- loi de Norton : `*Norton*.mfront`
- loi de Lemaître : `StrainHardeningCreep.mfront`
- endommagement fragile : `Mazars.mfront`, `Lorentz.mfront`, `DDIF2.mfront` ;
- endommagement ductile (fluage tertiaire) : `Burger.mfront`
- plasticité avec variables d'écrouissage cinématique n : `Chaboche.mfront`
- viscoplasticité avec variables d'écrouissage cinématique : `Chaboche2.mfront`
- lois monocristallines `MonoCristal.mfront` `MonoCristalNewtonRaphson.mfront` `MonoCristal_DD_CFC.mfront`
- lois polycristallines `PolyCristal.mfront` `PolyCristal_DD_CFC.mfront`

Guide des cas tests

- loi de Norton : `*Norton*.mfront`
- loi de Lemaître : `StrainHardeningCreep.mfront`
- endommagement fragile : `Mazars.mfront`, `Lorentz.mfront`, `DDIF2.mfront` ;
- endommagement ductile (fluage tertiaire) : `Burger.mfront`
- plasticité avec variables d'écrouissage cinématique n : `Chaboche.mfront`
- viscoplasticité avec variables d'écrouissage cinématique : `Chaboche2.mfront`
- lois monocristallines `MonoCristal.mfront` `MonoCristalNewtonRaphson.mfront` `MonoCristal_DD_CFC.mfront`
- lois polycristallines `PolyCristal.mfront` `PolyCristal_DD_CFC.mfront`

environ 50 exemples dans la base de cas test !

Généralités

Définitions

- propriété matériau `@MaterialProperty` :
  - fournie par le code appelant !
- variable local `@LocalVariable` :
  - permet de réduire le nombre de calculs en précalculant certains termes avant l'intégration (exemple de termes d'Arrhenius) ;
- variable interne `@StateVariable` ;
- variable interne auxiliaire `@AuxiliaryStateVariable` :
  - permet de réduire la taille des systèmes à intégrer ;
- variable externe `@ExternalStateVarible` ;

Généralités

- les lois de comportement mécanique sont par nature complexes :
  - **intégrer** une équation différentielle ... dans le meilleur des cas ! ;

Généralités

- les lois de comportement mécanique sont par nature complexes :
  - **intégrer** une équation différentielle ... dans le meilleur des cas ! ;
  - variables scalaires ou tensorielles ;

Généralités

- les lois de comportement mécanique sont par nature complexes :
  - **intégrer** une équation différentielle ... dans le meilleur des cas ! ;
  - variables scalaires ou tensorielles ;
  - cas très différents :
    - plasticité, endommagement, surfaces de charges ;
    - orthotropie ;

Généralités

- les lois de comportement mécanique sont par nature complexes :
  - **intégrer** une équation différentielle ... dans le meilleur des cas ! ;
  - variables scalaires ou tensorielles ;
  - cas très différents :
    - plasticité, endommagement, surfaces de charges ;
    - orthotropie ;
- l'algorithme d'intégration doit être :
  - fiable (donner le bon résultat) ;
  - robuste (converger) ;
  - efficace (temps de calcul) ;

Généralités

- les lois de comportement mécanique sont par nature complexes :
  - **intégrer** une équation différentielle ... dans le meilleur des cas ! ;
  - variables scalaires ou tensorielles ;
  - cas très différents :
    - plasticité, endommagement, surfaces de charges ;
    - orthotropie ;
- l'algorithme d'intégration doit être :
  - fiable (donner le bon résultat) ;
  - robuste (converger) ;
  - efficace (temps de calcul) ;
- travail long et pénible ;

Analyseurs disponibles

- quatre intégrateurs spécifiques (80% des besoins) :
  - écoulement viscoplastique isotrope $\dot{\epsilon}_{eq} = f\left(\sigma_{eq}\right)$;
  - écoulement viscoplastique isotrope avec écrouissage $\dot{\epsilon}_{eq} = f\left(\sigma_{eq}, \epsilon_{eq}\right)$;
  - écoulement plastique isotrope $f\left(\sigma_{eq}, \epsilon_{eq}\right) <= 0$;
  - une somme des différents écoulements précédents ;

Analyseurs disponibles

- quatre intégrateurs spécifiques (80% des besoins) :
  - écoulement viscoplastique isotrope $\dot{\epsilon}_{eq} = f(\sigma_{eq})$;
  - écoulement viscoplastique isotrope avec écrouissage $\dot{\epsilon}_{eq} = f(\sigma_{eq}, \epsilon_{eq})$;
  - écoulement plastique isotrope $f(\sigma_{eq}, \epsilon_{eq}) <= 0$;
  - une somme des différents écoulements précédents ;
- un intégrateur de type RUNGE-KUTTA :
  - différents algorithmes disponibles (correcteur prédicteur 5/4 par défaut) ;

Analyseurs disponibles

- quatre intégrateurs spécifiques (80% des besoins) :
  - écoulement viscoplastique isotrope $\dot{\epsilon}_{eq} = f(\sigma_{eq})$;
  - écoulement viscoplastique isotrope avec écrouissage $\dot{\epsilon}_{eq} = f(\sigma_{eq}, \epsilon_{eq})$;
  - écoulement plastique isotrope $f(\sigma_{eq}, \epsilon_{eq}) <= 0$;
  - une somme des différents écoulements précédents ;
- un intégrateur de type RUNGE-KUTTA :
  - différents algorithmes disponibles (correcteur prédicteur 5/4 par défaut) ;
- un intégrateur de type IMPLICITE :
  - différents algorithmes disponibles (NEWTON-RAPHSON par défaut) ;

Analyseurs disponibles

- quatre intégrateurs spécifiques (80% des besoins) :
  - écoulement viscoplastique isotrope $\dot{\epsilon}_{eq} = f(\sigma_{eq})$;
  - écoulement viscoplastique isotrope avec écrouissage $\dot{\epsilon}_{eq} = f(\sigma_{eq}, \epsilon_{eq})$;
  - écoulement plastique isotrope $f(\sigma_{eq}, \epsilon_{eq}) <= 0$;
  - une somme des différents écoulements précédents ;
- un intégrateur de type RUNGE-KUTTA :
  - différents algorithmes disponibles (correcteur prédicteur 5/4 par défaut) ;
- un intégrateur de type IMPLICITE :
  - différents algorithmes disponibles (NEWTON-RAPHSON par défaut) ;
- un intégrateur par défaut laissant l'utilisateur libre de faire ce qu'il veut !

Choix de l'analyseur à utiliser

- si un intégrateur spécifique existe, l'utiliser :
  - réduction du nombre de variables d'intégration et méthode implicite ;

Choix de l'analyseur à utiliser

- si un intégrateur spécifique existe, l'utiliser :
  - réduction du nombre de variables d'intégration et méthode implicite ;
- si l'on doit recourir à un intégrateur spécifique, **préférer l'intégration implicite**, surtout s'il s'agit de lois indépendantes du temps (plasticité, endommagement) :
  - l'équation différentielle pour la plasticité ou l'endommagement doit être remplacée par la nullité du critère en fin de pas de temps ;
  - il vaut mieux savoir calculer la jacobienne :
    - la bibliothèque tensorielle livrée avec `mfront` est alors très utile ;
  - les temps de calculs sont souvent **très** avantageux ;
  - on a (plus facilement) la **tangente cohérente** ;

Choix de l'analyseur à utiliser

- si un intégrateur spécifique existe, l'utiliser :
  - réduction du nombre de variables d'intégration et méthode implicite ;
- si l'on doit recourir à un intégrateur spécifique, **préférer l'intégration implicite**, surtout s'il s'agit de lois indépendantes du temps (plasticité, endommagement) :
  - l'équation différentielle pour la plasticité ou l'endommagement doit être remplacée par la nullité du critère en fin de pas de temps ;
  - il vaut mieux savoir calculer la jacobienne :
    - la bibliothèque tensorielle livrée avec `mfront` est alors très utile ;
  - les temps de calculs sont souvent **très** avantageux ;
  - on a (plus facilement) la **tangente cohérente** ;
- utiliser une méthode de Runge-Kutta si :
  - **vraiment rien d'autre** n'est possible (impossibilité de calculer la jacobienne, nombre de variables très grands) ;
  - si le temps de développement est limité ;
  - si le temps d'intégration de la loi de comportement ne pose pas de problème de performance ;
  - attention aux variations des paramètres externes pour les conditions de consistance (en particulier la température) !

Variables prédéfinies

- la déformation élastique `eel` est déclarée automatiquement pour tous les intégrateurs (sauf celui par défaut) ;
- la température `T` et son incrément `dT` est déclarée automatiquement ;
- pour une variable interne `X`, la variable `dX` est automatiquement défini pour définir l'incrément de la variable `X` sur le pas :
  - attention aux substitutions de variables (voir transparents suivants) ;

Variables prédéfinies

- la déformation élastique `eel` est déclarée automatiquement pour tous les intégrateurs (sauf celui par défaut) ;
- la température `T` et son incrément `dT` est déclarée automatiquement ;
- pour une variable interne `X`, la variable `dX` est automatiquement défini pour définir l'incrément de la variable `X` sur le pas :
  - attention aux substitutions de variables (voir transparents suivants) ;
- pour une variable externe `X`, la variable `dX` est automatiquement défini pour définir l'incrément de la variable `X` sur le pas ;

Variables prédéfinies

- la déformation élastique `eel` est déclarée automatiquement pour tous les intégrateurs (sauf celui par défaut) ;
- la température `T` et son incrément `dT` est déclarée automatiquement ;
- pour une variable interne `X`, la variable `dX` est automatiquement défini pour définir l'incrément de la variable `X` sur le pas :
  - attention aux substitutions de variables (voir transparents suivants) ;
- pour une variable externe `X`, la variable `dX` est automatiquement défini pour définir l'incrément de la variable `X` sur le pas ;
- attention aux substitutions de variables (voir transparents suivants) :
  - pour les intégrateurs explicites, `dX` est remplacée par la valeur de la vitesse dans les parties `@ComputeStress` et `@Derivative`

Transformation du code

- `mfront` modifie largement le code fourni par l'utilisateur :
  - pour le rendre compatible avec le `C++` :
    - `mfront` utilise des fonctionnalités très avancées `C++` (`template` et `template metaprogramming`) dont la syntaxe est difficile (*a minima*) ;
    - ajout du pointeur `this` devant toutes les variables internes ;
    - ajout de code de sortie implicite ;
  - pour remplacer les valeurs des variables par les variables actualisées dans le système différentiel :

    `exp(-Q/(R*T))` devient

    `exp(-Q/(R*(this->T+(Behaviour::Theta)*this->dT)))` en implicite

    `exp(-Q/(R*(this->T_courant)))` en Runge-Kutta
  - pour évaluer les contraintes en cours de résolution et en fin du pas

    `sig = D*eel` devient :

    `this->sig = (this->D)*(this->eel_courant)` ou

    `(this->D)*(this->eel_final)`

@InitLocalVariables

- Il s'agit d'une partie du code :
  - dédiée à l'initialisation de variables locales (données membre de la classe C++) ;
  - qui n'est pas « transformée » par les règles précédentes ;
- Les variables locales permettent d'économiser des calculs coûteux en les effectuant en dehors de l'algorithme d'intégration ;

Limitations de `mfront`

- les variables internes sont ou des scalaires ou des tenseurs d'ordre 2 symétriques :
  - **il y a un $\sqrt{2}$ sur les termes extradiagonaux** pour les tenseurs d'ordre 2 symétriques ;
  - les « tenseurs d'ordre 4 » sont utilisables au cours de l'intégration et permettent de définir certains blocs de la jacobienne en implicite ;
- la possibilité d'utiliser des vecteurs comme variables internes (modèles de zones cohésives) est partiellement implantée ;
- la possibilité d'utiliser des tenseurs non symétriques comme variables internes (grandes transformations) est partiellement implantée ;

Limitations de `mfront`

- les variables internes sont ou des scalaires ou des tenseurs d'ordre 2 symétriques :
  - **il y a un $\sqrt{2}$ sur les termes extradiagonaux** pour les tenseurs d'ordre 2 symétriques ;
  - les « tenseurs d'ordre 4 » sont utilisables au cours de l'intégration et permettent de définir certains blocs de la jacobienne en implicite ;
- la possibilité d'utiliser des vecteurs comme variables internes (modèles de zones cohésives) est partiellement implantée ;
- la possibilité d'utiliser des tenseurs non symétriques comme variables internes (grandes transformations) est partiellement implantée ;
- `mfront` ne supporte (aujourd'hui) que les modélisations $1D$ (trois composantes du tenseurs des déformations), $2D$ axisyimétriques, déformations planes (généralisées ou non) et contraintes planes, et $3D$:
  - le support des contraintes planes (généralisées ou non) est partiel (on doit écrire une implantation spécifique ou laisser l'interface au code éléments finis faire, voir l'interface `umat`) ;

Limitations de `mfront`

- les variables internes sont ou des scalaires ou des tenseurs d'ordre 2 symétriques :
  - **il y a un $\sqrt{2}$ sur les termes extradiagonaux** pour les tenseurs d'ordre 2 symétriques ;
  - les « tenseurs d'ordre 4 » sont utilisables au cours de l'intégration et permettent de définir certains blocs de la jacobienne en implicite ;
- la possibilité d'utiliser des vecteurs comme variables internes (modèles de zones cohésives) est partiellement implantée ;
- la possibilité d'utiliser des tenseurs non symétriques comme variables internes (grandes transformations) est partiellement implantée ;
- `mfront` ne supporte (aujourd'hui) que les modélisations $1D$ (trois composantes du tenseurs des déformations), $2D$ axisyimétriques, déformations planes (généralisées ou non) et contraintes planes, et $3D$:
  - le support des contraintes planes (généralisées ou non) est partiel (on doit écrire une implantation spécifique ou laisser l'interface au code éléments finis faire, voir l'interface `umat`) ;
- `mfront` supporte des lois isotropes et orthotropes (intégrateurs génériques) :

Optimisation du code

- la loi de comportement est « instanciée » une fois par hypothèse de modélisation (voir `@ModellingHypothesis`) :
  - les opérations tensorielles sont « optimisées » pour chaque dimension :
    - déroulement des boucles à la compilation ;
  - allocation de la mémoire sur la pile ;

Analyseurs spécifiques

Généralités

- quatre intégrateurs spécifiques :
  - `IsotropicMisesCreep`, écoulement viscoplastique isotrope $\dot{\epsilon}_{eq} = f\left(\sigma_{eq}\right)$;
  - `IsotropicStrainHardeningMisesCreep`, écoulement viscoplastique isotrope avec écrouissage $\dot{\epsilon}_{eq} = f\left(\sigma_{eq}, \epsilon_{eq}\right)$;
  - `IsotropicPlasticMisesFlow`, écoulement plastique isotrope $f\left(\sigma_{eq}, \epsilon_{eq}\right) <= 0$;
  - `MultipleIsotropicMisesFlows`, une somme des différents écoulements précédents ;

Généralités

- quatre intégrateurs spécifiques :
  - `IsotropicMisesCreep`, écoulement viscoplastique isotrope $\dot{\epsilon}_{eq} = f\left(\sigma_{eq}\right)$;
  - `IsotropicStrainHardeningMisesCreep`, écoulement viscoplastique isotrope avec écrouissage $\dot{\epsilon}_{eq} = f\left(\sigma_{eq}, \epsilon_{eq}\right)$;
  - `IsotropicPlasticMisesFlow`, écoulement plastique isotrope $f\left(\sigma_{eq}, \epsilon_{eq}\right) <= 0$;
  - `MultipleIsotropicMisesFlows`, une somme des différents écoulements précédents ;
- l'élasticité est élastique isotrope :
  - les coefficients d'élasticité sont donnés par le code aux éléments finis ;

Généralités

- quatre intégrateurs spécifiques :
  - `IsotropicMisesCreep`, écoulement viscoplastique isotrope $\dot{\epsilon}_{eq} = f\left(\sigma_{eq}\right)$;
  - `IsotropicStrainHardeningMisesCreep`, écoulement viscoplastique isotrope avec écrouissage $\dot{\epsilon}_{eq} = f\left(\sigma_{eq}, \epsilon_{eq}\right)$;
  - `IsotropicPlasticMisesFlow`, écoulement plastique isotrope $f\left(\sigma_{eq}, \epsilon_{eq}\right) <= 0$;
  - `MultipleIsotropicMisesFlows`, une somme des différents écoulements précédents ;
- l'élasticité est élastique isotrope :
  - les coefficients d'élasticité sont donnés par le code aux éléments finis ;
- il suffit de donner la (ou les) fonction(s) f ;

Généralités

- quatre intégrateurs spécifiques :
  - `IsotropicMisesCreep`, écoulement viscoplastique isotrope $\dot{\epsilon}_{eq} = f(\sigma_{eq})$;
  - `IsotropicStrainHardeningMisesCreep`, écoulement viscoplastique isotrope avec écrouissage $\dot{\epsilon}_{eq} = f(\sigma_{eq}, \epsilon_{eq})$;
  - `IsotropicPlasticMisesFlow`, écoulement plastique isotrope $f(\sigma_{eq}, \epsilon_{eq}) <= 0$;
  - `MultipleIsotropicMisesFlows`, une somme des différents écoulements précédents ;
- l'élasticité est élastique isotrope :
  - les coefficients d'élasticité sont donnés par le code aux éléments finis ;
- il suffit de donner la (ou les) fonction(s) f ;
- algorithme optimisé avec réduction à une équation scalaire par écoulement ;

Généralités

- quatre intégrateurs spécifiques :
  - `IsotropicMisesCreep`, écoulement viscoplastique isotrope $\dot{\epsilon}_{eq} = f(\sigma_{eq})$;
  - `IsotropicStrainHardeningMisesCreep`, écoulement viscoplastique isotrope avec écrouissage $\dot{\epsilon}_{eq} = f(\sigma_{eq}, \epsilon_{eq})$;
  - `IsotropicPlasticMisesFlow`, écoulement plastique isotrope $f(\sigma_{eq}, \epsilon_{eq}) <= 0$;
  - `MultipleIsotropicMisesFlows`, une somme des différents écoulements précédents ;
- l'élasticité est élastique isotrope :
  - les coefficients d'élasticité sont donnés par le code aux éléments finis ;
- il suffit de donner la (ou les) fonction(s) f ;
- algorithme optimisé avec réduction à une équation scalaire par écoulement ;
- pour des raisons de performances, il faut penser à précalculer les termes dépendants de la température (ou d'autres variables externes) :
  - variables locales et `@InitLocalVariables`

Loi viscoplastique isotrope

- comportement viscoplastique du *NbZrC* :
  - écoulement thermique :

$$\dot{\epsilon}_{eq} = A(T)\,\sigma_{eq}^{n}$$

  - écoulement athermique :

$$\dot{\epsilon}_{eq} = B\frac{\mathrm{d}D}{\mathrm{d}t}\sigma_{eq}$$

Loi viscoplastique isotrope

```
@Parser    MultipleIsotropicMisesFlows;
@Material  NbZrC;
@Behaviour Creep;
@Author    É. Brunon;

@LocalVar real A;
@ExternalStateVar real dpa;
dpa.setGlossaryName("IrradiationDamage");

@InitLocalVars{
  A = 7.1687e-31*exp(-4.1722e+04/(T+theta*dT));
}

// Fluage Thermique
@FlowRule Creep{
  real tmp = A*pow(seq,3.9);
  df_dseq  = 4.9*tmp;
  f        = seq*tmp;
}

// Fluage d'irradiation
@FlowRule Creep{
  df_dseq = 0.5e-12*ddpa/dt;
  f       = df_dseq*seq;
}
```

Loi viscoplastique isotrope

```
@Parser    MultipleIsotropicMisesFlows;
@Material  NbZrC;
@Behaviour Creep;
@Author    É. Brunon;

@LocalVar real A;
@ExternalStateVar real dpa;
dpa.setGlossaryName("IrradiationDamage");

@InitLocalVars{
  A = 7.1687e-31*exp(-4.1722e+04/(T+theta*dT));
}

// Fluage Thermique
@FlowRule Creep{
  real tmp = A*pow(seq,3.9);
  df_dseq  = 4.9*tmp;
  f        = seq*tmp;
}

// Fluage d'irradiation
@FlowRule Creep{
  df_dseq = 0.5e-12*ddpa/dt;
  f       = df_dseq*seq;
}
```

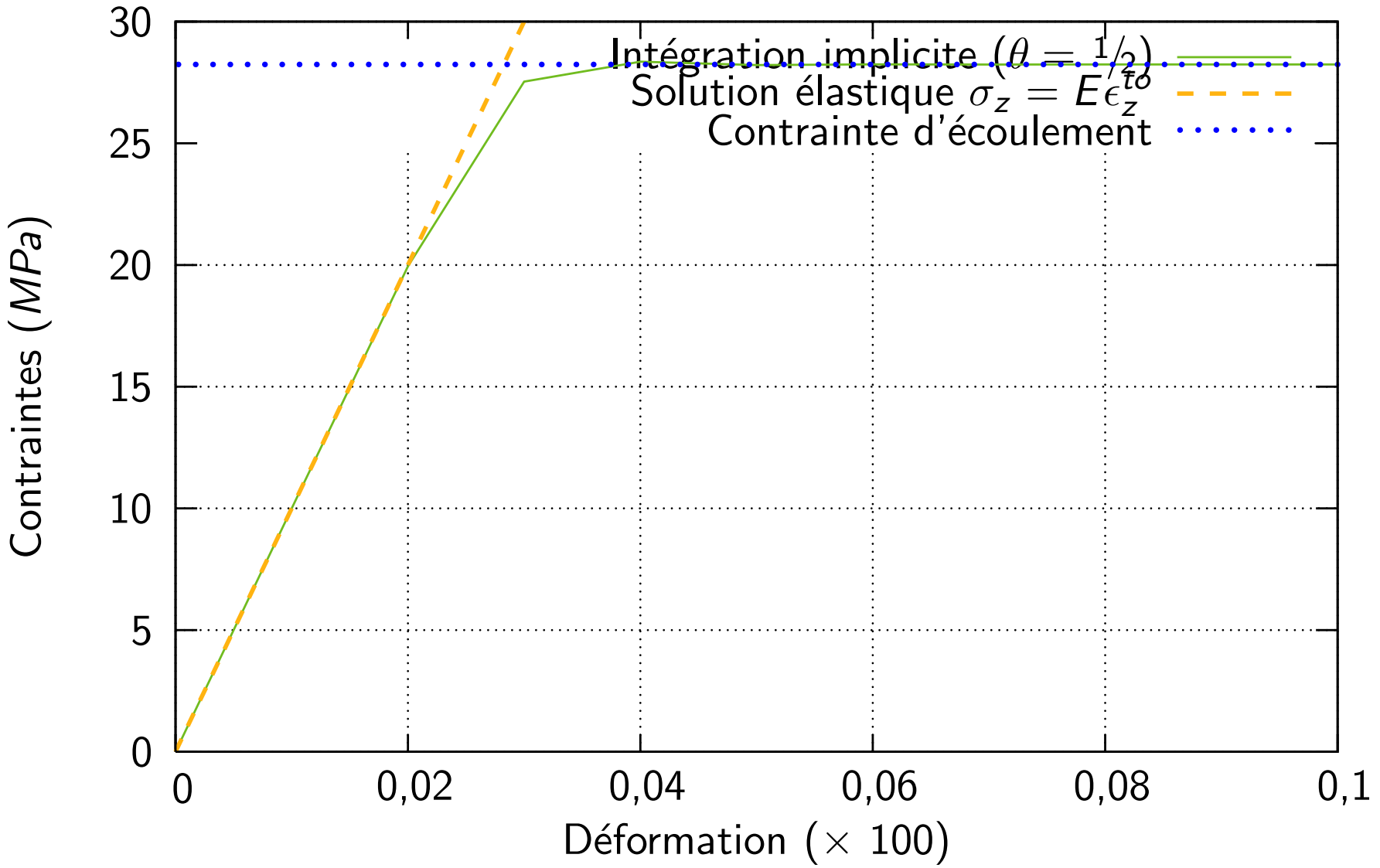
```
mfront --obuild --interface=umat NbZrC_CreepBehaviour.mfront
```

Loi viscoplastique isotrope

```
@Parser    MultipleIsotropicMisesFlows;
@Material  NbZrC;
@Behaviour Creep;
@Author    É. Brunon;

@LocalVar real A;
@ExternalStateVar real dpa;
dpa.setGlossaryName("IrradiationDamage");

@InitLocalVars{
  A = 7.1687e-31*exp(-4.1722e+04/(T+theta*dT));
}

// Fluage Thermique
@FlowRule Creep{
  real tmp = A*pow(seq,3.9);
  df_dseq  = 4.9*tmp;
  f        = seq*tmp;
}

// Fluage d'irradiation
@FlowRule Creep{
  df_dseq = 0.5e-12*ddpa/dt;
  f       = df_dseq*seq;
}
```

- `mfront --obuild --interface=umat NbZrC_CreepBehaviour.mfront`
- sous Cast3M : attention à l'incrément de temps qui est nul en cas de convergence forcée !

Qualification



- essai de traction à vitesse imposée

Intégration par une méthode explicite (Runge-Kutta)

Généralités

- il faut résumer l'écriture de la loi de comportement à un système différentiel :

$$\dot{Y} = G\left(Y, t\right)$$

où t représente symboliquement l'évolution des variables externes et de la déformation totale ;

Généralités

- il faut résumer l'écriture de la loi de comportement à un système différentiel :

$$\dot{Y} = G\left(Y, t\right)$$

où t représente symboliquement l'évolution des variables externes et de la déformation totale ;

- le système différentiel s'écrit dans un bloc `@Derivative` ;

Généralités

- il faut résumer l'écriture de la loi de comportement à un système différentiel :

$$\dot{Y} = G\left(Y, t\right)$$

  où t représente symboliquement l'évolution des variables externes et de la déformation totale ;
- le système différentiel s'écrit dans un bloc `@Derivative` ;
- pour toute variable interne et externe, `dX` est remplacée par la valeur de la vitesse dans les parties `@ComputeStress` et `@Derivative`
  - ce n'est pas l'incrément !

Généralités

- il faut résumer l'écriture de la loi de comportement à un système différentiel :

$$\dot{Y} = G\left(Y, t\right)$$

  où t représente symboliquement l'évolution des variables externes et de la déformation totale ;
- le système différentiel s'écrit dans un bloc `@Derivative` ;
- pour toute variable interne et externe, `dX` est remplacée par la valeur de la vitesse dans les parties `@ComputeStress` et `@Derivative`
  - ce n'est pas l'incrément !
- le code du bloc `@UpdateAuxiliaryStateVariables` peut être appelé plusieurs fois. Il faut utiliser la variable locale `dt_` pour connaître le pas de temps effectivement utilisé (`dt` désigne toujours le pas de temps total)

Loi viscoplastique orthotrope

- exemple d'une loi viscoplastique orthotrope ;
- utilisation de l'intégrateur générique RUNGE-KUTTA ;
- intégration du système :

$$\begin{cases} \dot{\underline{\varepsilon}}^{el} &= \dot{\underline{\varepsilon}}^{to} - \dot{\underline{\varepsilon}}^{vis} \\ \dot{\underline{\varepsilon}}^{vis} &= f\left(\sigma_{eq}\right)\underline{\mathbf{n}} \end{cases}$$

Loi viscoplastique orthotrope

```
@Parser    RungeKutta;
@Behaviour GPLS;
@Author    Helfer Thomas;
@Algorithm rk54;
@Date 27/03/09;

@Includes{
#include<MaterialLaw/Lame.hxx>
#include<MaterialLaw/Hill.hxx>
}

@Epsilon 1.e-6;

@OrthotropicBehaviour;
@RequireStiffnessTensor;

@StateVar real p;      /* Equivalent viscoplastic strain */
@StateVar Stensor evp; /* Viscoplastic strain            */
```

Loi viscoplastique orthotrope

```
@ComputeStress{
  sig = D*eel;
}

@Derivative{
  /* coefficients d'orthotropie */
  real Hrr = ...;
  real Htt = ...;
  real Hzz = ...;
  real Hrt = ...;
  real Hrz = ...;
  real Htz = ...;
  /* tenseur de Hill */
  st2tost2<N,real> H = hillTensor<N,real>(Hzz,Hrr,Htt,
                                          Hrz,Hrt,Htz);
  real sigeq = sqrt(sig|H*sig);
  if(sigeq>1.e9){
    return false;
  }
  Stensor  n(0.);
  if(sigeq > 10.e-7){
    n    = H*sig/sigeq;
  }
  /* Système différentiel */
  dp   = ...;
  devp = dp*n;
  deel = deto - devp;
}
```

Critère de convergence

- la directive `@Epsilon` permet de préciser la valeur du critère utilisé par les différentes méthodes d'intégration :
  - par défaut, toutes les variables contribuent de la même manière au calcul de l'erreur ;
  - la valeur par défaut est adaptée si toutes les variables internes sont de l'ordre de grandeur des déformations ;

Critère de convergence

- la directive `@Epsilon` permet de préciser la valeur du critère utilisé par les différentes méthodes d'intégration :
  - par défaut, toutes les variables contribuent de la même manière au calcul de l'erreur ;
  - la valeur par défaut est adaptée si toutes les variables internes sont de l'ordre de grandeur des déformations ;
- il est possible d'affecter un poids aux différentes variables par la méthodes `setErrorNormalisationFactor` :

```
p.setErrorNormalisationFactor(young)
```

Intégration implicite

Analyseurs disponibles

- Deux analyseurs :
  - `Implicit` qui déclare automatiquement la déformation élastique ;
  - `ImplicitII` qui ne déclare pas automatiquement la déformation élastique ;
- différents algorithmes :
  - `NewtonRaphson` (jacobienne calculée par l'utilisateur) ;
  - `NewtonRaphson_NumericalJacobian` (jacobienne calculée par différence finie centrée) ;
  - `Broyden` (jacobienne partielle) ;

Intégration implicite

- le système différentiel est remplacé par un système non-linéaire :

$$F(\Delta Y) = \Delta Y - \Delta t\, G(Y_t + \theta \Delta Y, t + \theta\, \Delta t) = 0$$

Intégration implicite

- le système différentiel est remplacé par un système non-linéaire :

$$F(\Delta Y) = \Delta Y - \Delta t\, G(Y_t + \theta \Delta Y, t + \theta\, \Delta t) = 0$$

- pour les lois indépendantes du temps, on annule directement la surface de charge !

Intégration implicite

- le système différentiel est remplacé par un système non-linéaire :

$$F(\Delta Y) = \Delta Y - \Delta t\, G(Y_t + \theta \Delta Y, t + \theta\, \Delta t) = 0$$

- pour les lois indépendantes du temps, on annule directement la surface de charge !
- on résout ce système par un Newton-Raphson
  - il faut la jacobienne $J = \dfrac{\partial F}{\partial \Delta Y}$

Intégration implicite

- le système différentiel est remplacé par un système non-linéaire :

$$F(\Delta Y) = \Delta Y - \Delta t\, G(Y_t + \theta \Delta Y, t + \theta\, \Delta t) = 0$$

- pour les lois indépendantes du temps, on annule directement la surface de charge !
- on résout ce système par un Newton-Raphson
  - il faut la jacobienne $J = \dfrac{\partial F}{\partial \Delta Y}$
- la jacobienne peut être calculée par blocs :

$$J = \frac{\partial F}{\partial Y} = \begin{pmatrix} \frac{\partial f_{y_1}}{\partial y_1} & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \frac{\partial f_{y_i}}{\partial y_j} & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & \cdots & \frac{\partial f_{y_N}}{\partial y_N} \end{pmatrix}$$

Intégration implicite

- le système différentiel est remplacé par un système non-linéaire :

$$F(\Delta Y) = \Delta Y - \Delta t\, G(Y_t + \theta \Delta Y, t + \theta\, \Delta t) = 0$$

- pour les lois indépendantes du temps, on annule directement la surface de charge !
- on résout ce système par un Newton-Raphson
  - il faut la jacobienne $J = \dfrac{\partial F}{\partial \Delta Y}$
- la jacobienne peut être calculée par blocs :

$$J = \frac{\partial F}{\partial Y} = \begin{pmatrix} \frac{\partial f_{y_1}}{\partial y_1} & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \frac{\partial f_{y_i}}{\partial y_j} & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & \cdots & \frac{\partial f_{y_N}}{\partial y_N} \end{pmatrix}$$

- on peut demander une vérification numérique !

```
@CompareToNumericalJacobian true;
```

Calcul de la jacobienne

```
@Integrator{
  ...
  const real sigeq = sqrt(sig|H*sig);
  const real tmp = A*pow(sigeq,E-1.);
  real inv_sigeq(0);
  Stensor  n(0.);
  if(sigeq > 1.){
    inv_sigeq = 1/sigeq;
  }
  n  = (H*sig)*inv_sigeq;
  feel += dp*n-deto;
  fp   -= tmp*sigeq*dt;
  dfeel_ddeel += theta*dp*(H-(n^n))*D*inv_sigeq;
  dfeel_ddp    = n;
  dfp_ddeel    = -theta*tmp*E*dt*(n|D);
}
```

- les variables `fX` sont des « images » du vecteur inconnu, et les dérivées `dfX_ddY` des images de la matrice jacobienne ;

L'interface `umat`

L'interface umat

- l'interface umat a pour rôle :
  - d'assurer le sous-découpage du pas de temps en cas de non convergence :
    - `@UMATUseTimeSubStepping[umat] true;`
    - `@UMATMaximumSubStepping[umat] 20;`

L'interface umat

- l'interface umat a pour rôle :
  - d'assurer le sous-découpage du pas de temps en cas de non convergence :
    - `@UMATUseTimeSubStepping[umat] true;`
    - `@UMATMaximumSubStepping[umat] 20;`
  - d'effectuer les rotations dans le repère matériau pour les lois de comportement orthotrope :
    - `@OrthotropicBehaviour;`

L'interface `umat`

- l'interface umat a pour rôle :
  - d'assurer le sous-découpage du pas de temps en cas de non convergence :
    - `@UMATUseTimeSubStepping[umat] true;`
    - `@UMATMaximumSubStepping[umat] 20;`
  - d'effectuer les rotations dans le repère matériau pour les lois de comportement orthotrope :
    - `@OrthotropicBehaviour;`
  - de calculer les matrices d'élasticité et de coefficients de dilatation thermique *à la demande* :
    - `@RequireStiffnessTensor;`
    - `@RequireThermalExpansionCoefficientTensor;`

L'interface `umat`

- l'interface umat a pour rôle :
  - d'assurer le sous-découpage du pas de temps en cas de non convergence :
    - `@UMATUseTimeSubStepping[umat] true;`
    - `@UMATMaximumSubStepping[umat] 20;`
  - d'effectuer les rotations dans le repère matériau pour les lois de comportement orthotrope :
    - `@OrthotropicBehaviour;`
  - de calculer les matrices d'élasticité et de coefficients de dilatation thermique *à la demande* :
    - `@RequireStiffnessTensor;`
    - `@RequireThermalExpansionCoefficientTensor;`
  - de générer des fichiers `mtest` en cas d'échec pour analyse :
    - `@UMATGenerateMTestFileOnFailure[umat] true;`

L'interface `umat`

- l'interface umat a pour rôle :
  - d'assurer le sous-découpage du pas de temps en cas de non convergence :
    - `@UMATUseTimeSubStepping[umat] true;`
    - `@UMATMaximumSubStepping[umat] 20;`
  - d'effectuer les rotations dans le repère matériau pour les lois de comportement orthotrope :
    - `@OrthotropicBehaviour;`
  - de calculer les matrices d'élasticité et de coefficients de dilatation thermique *à la demande* :
    - `@RequireStiffnessTensor;`
    - `@RequireThermalExpansionCoefficientTensor;`
  - de générer des fichiers `mtest` en cas d'échec pour analyse :
    - `@UMATGenerateMTestFileOnFailure[umat] true;`
  - de traiter la condition de contraintes planes par une variable interne supplémentaire représentant la déformation totale axiale et une boucle externe pour assurer $\sigma_z = 0$;

Gestion des contraintes planes

- introduction d'une variable interne supplémentaire (déformation axiale totale) ;
- utilisation de la loi $2D$;
- boucle pour trouver un incrément de déformation totale tel que la contrainte soit nulle en fin de pas :

$$\Delta\,\epsilon_z^{to\,(0)} = -\frac{1}{E}\,\sigma_z{}^{(0)} - \nu\left(\Delta\,\epsilon_x^{to} + \Delta\,\epsilon_y^{to}\right)$$

$$\Delta\,\epsilon_z^{to\,(1)} = \Delta\,\epsilon_z^{to\,(0)} - \frac{1}{E}\,\frac{1}{E}\sigma_z{}^{(1)}$$

$\Delta\,\epsilon_z^{to\,(n)}$ donné par la méthode de la sécante

- critère d'arrêt :

$$\left|\sigma_z{}^{(n)}\right| < E\,\varepsilon \quad \text{avec} \quad \varepsilon = 10^{-12}$$

Appel externe de loi de comportement

```
* Création d'une table contenant les données relatives
* à la propriété externe :
* - 'MODELE' contient le nom de la fonction appelée
* - 'LIBRAIRIE' contient le nom de la librairie externe
Tloi = 'TABLE';
Tloi. 'MODELE' = 'NbZrCCreep' ;
Tloi. 'LIBRAIRIE' = 'libNbZrCBehaviours.so' ;
* Création du modèle mécanique
ModM1 = 'MODELISER' s1 'MECANIQUE' 'ELASTIQUE'
'ISOTROPE' 'NON_LINEAIRE' 'UTILISATEUR'
''DESC_LOI' Tloi
'C_MATERIAU' CLOI
'C_VARINTER' VLOI
'PARA_LOI' PLOI;
```

Appel externe de loi de comportement

```
* Création d'une table contenant les données relatives
* à la propriété externe :
* - 'MODELE' contient le nom de la fonction appelée
* - 'LIBRAIRIE' contient le nom de la librairie externe
Tloi = 'TABLE';
Tloi. 'MODELE' = 'NbZrCCreep' ;
Tloi. 'LIBRAIRIE' = 'libNbZrCBehaviours.so' ;
* Création du modèle mécanique
ModM1 = 'MODELISER' s1 'MECANIQUE' 'ELASTIQUE'
'ISOTROPE' 'NON_LINEAIRE' 'UTILISATEUR'
''DESC_LOI' Tloi
'C_MATERIAU' CLOI
'C_VARINTER' VLOI
'PARA_LOI' PLOI;
```

- utilisation du mot clé 'DESC_LOI' au lieu de 'NUME_LOI' ;

Algorithmes numériques alternatifs

Algorithmes de Broyden

- Les algorithmes de Broden sont des algorithmes de type quasi-Newton :

$$Y_n = Y_{n-1} - \underset{\sim}{J}_{n-1}^{-1} . F\left(Y_{n-1}\right)$$

- premier algorithme de Broyden (mise à jour de rang 1) :

$$\underset{\sim}{J}_n = \underset{\sim}{J}_{n-1} + \frac{\Delta F_n - \underset{\sim}{J}_{n-1} . \Delta Y_n}{||\Delta Y_n||^2} \otimes \Delta Y_n$$

  - avantage : on peut donner certains termes du jacobien (les plus faciles à calculer ou les moins coûteux) ;
  - avantage : on peut facilement partir sur une estimation numérique du jacobien ou en calculer une en cours de résolution si la qualité de $\underset{\sim}{J}_n$ se dégrade ;

Algorithmes de Broyden

- Les algorithmes de Broden sont des algorithmes de type quasi-Newton :

$$Y_n = Y_{n-1} - \underset{\sim}{J}_{n-1}^{-1} \,.\, F\left(Y_{n-1}\right)$$

- premier algorithme de Broyden (mise à jour de rang 1) :

$$\underset{\sim}{J}_n = \underset{\sim}{J}_{n-1} + \frac{\Delta F_n - \underset{\sim}{J}_{n-1} \,.\, \Delta Y_n}{||\Delta Y_n||^2} \otimes \Delta Y_n$$

  - avantage : on peut donner certains termes du jacobien (les plus faciles à calculer ou les moins coûteux) ;
  - avantage : on peut facilement partir sur une estimation numérique du jacobien ou en calculer une en cours de résolution si la qualité de $\underset{\sim}{J}_n$ se dégrade ;
- second algorithme de Broyden (formule de Sherman-Morrison) :

$$\underset{\sim}{J}_n^{-1} = \underset{\sim}{J}_{n-1}^{-1} + \frac{\Delta Y_n - \underset{\sim}{J}_{n-1}^{-1} \,.\, \Delta F_n}{\Delta Y_n \mid \underset{\sim}{J}_{n-1}^{-1} \,.\, \Delta F_n} \otimes \left(\Delta Y_n \mid \underset{\sim}{J}_{n-1}^{-1}\right)$$

  - avantage : on gagne l'inversion de la matrice ;

Algorithmes de Broyden

- Les algorithmes de Broden sont des algorithmes de type quasi-Newton :

$$Y_n = Y_{n-1} - \underset{\sim}{J}_{n-1}^{-1} . F\left(Y_{n-1}\right)$$

- premier algorithme de Broyden (mise à jour de rang 1) :

$$\underset{\sim}{J}_n = \underset{\sim}{J}_{n-1} + \frac{\Delta F_n - \underset{\sim}{J}_{n-1} . \Delta Y_n}{||\Delta Y_n||^2} \otimes \Delta Y_n$$

  - avantage : on peut donner certains termes du jacobien (les plus faciles à calculer ou les moins coûteux) ;
  - avantage : on peut facilement partir sur une estimation numérique du jacobien ou en calculer une en cours de résolution si la qualité de $\underset{\sim}{J}_n$ se dégrade ;
- second algorithme de Broyden (formule de Sherman-Morrison) :

$$\underset{\sim}{J}_n^{-1} = \underset{\sim}{J}_{n-1}^{-1} + \frac{\Delta Y_n - \underset{\sim}{J}_{n-1}^{-1} . \Delta F_n}{\Delta Y_n \mid \underset{\sim}{J}_{n-1}^{-1} . \Delta F_n} \otimes \left(\Delta Y_n \mid \underset{\sim}{J}_{n-1}^{-1}\right)$$

  - avantage : on gagne l'inversion de la matrice ;
- Les formules précédentes ont un coût !

Initialisation de la jacobienne

- il faut initialiser la jacobienne (ou son inverse) ;
- l'identité, une bonne approximation ?

$$\dot{Y} = G\left(Y\right) \quad \Rightarrow \quad \Delta\, Y - \Delta\, t\, G\left(Y + \theta\, \Delta\, Y\right) = 0$$

$$\Rightarrow \quad J = I - \theta \Delta\, t \frac{G}{\Delta\, Y}$$

- choix par défaut dans `mfront` ;

Initialisation de la jacobienne

Algorithme	Coût par rapport à Newton
Broyden	$4,29$
Broyden, $\underset{\sim 0}{J}$ numérique	$2,02$
Broyden II	$4,15$

- il faut initialiser la jacobienne (ou son inverse) ;
- l'identité, une bonne approximation ?

$$\dot{Y} = G(Y) \quad \Rightarrow \quad \Delta Y - \Delta t\, G(Y + \theta \Delta Y) = 0$$

$$\Rightarrow \quad J = I - \theta \Delta t \frac{G}{\Delta Y}$$

- choix par défaut dans `mfront` ;
- en pratique les résultats peuvent être mitigés :
  - il peut être intéressant de démarrer par une approximation numérique du jacobien ;

Jacobienne partielle

Variante	Nombre de cycles	Ratio par rapport à NEWTON
Défaut	20 197 423	4, 29
J exact	6 120 862	1, 3
$\frac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta \underline{\epsilon}^{el}}$ exact	36 821 766	7, 82
$\frac{\partial f_p}{\partial \Delta p}$ exact	33 826 368	7, 186
$\frac{\partial f_p}{\partial \Delta \underline{\epsilon}^{el}}$ exact	19 577 698	4, 15
$\frac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta p}$ exact	12 132 956	2, 58
$\frac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta p}$ et $\frac{\partial f_p}{\partial \Delta \underline{\epsilon}^{el}}$ exacts	4 686 228	0, 995

- le premier algorithme de BROYDEN laisse la possibilité de donner certains termes du jacobien ;
  - ça peut être très intéressant !

Amélioration de la robustesse

- déjà testé :
  - initialisation par prédiction explicite :
    - peu ou pas efficace ;
  - accélération de `Cast3M` :
    - plus stable, très peu efficace (perte de la convergence quadratique) ;
    - très très utile dans `mtest` !
  - relaxation :
    - plus stable, peu employé ;
- en cours de développement : méthode de Powell :
  - méthode à région de confiance avec transition continue entre une minimisation (plus grande pente) et un quasi-Newton (convergence locale) ;

Conclusions

Perspectives

- ajouts de nouveaux analyseurs spécfiques :
  - plasticité isotrope à écrouissage linéaire ;
  - plasticité/viscoplasticité isotrope compressible ;
- matrice tangente cohérente :
  - facile pour algorithmes spécifiques ;
  - pour l'intégration explicite :
    - matrice tangente ?
  - pour l'intégration implicite par NEWTON :
    - calcul automatique ;
    - surcharge par l'utilisateur ;
  - pour l'intégration implicite par BROYDEN :
    - ? ? ?
- retourner un estimation du pas de temps :
  - validation de l'hypothèse de linéarité du chargement ;
  - pas de temps adaptatif ;

Perspectives

- algorithme de Powell ;
- résolution en contraintes planes optimisée :
  - algorithmes spécifiques ;
  - prise en compte dans dans les algorithmes génériques ;
- support de lois mécaniques générales :
  - transformations finies ;
  - modèles de zones cohésives (intégration implicite) ;
  - lois à gradient ;
- à terme, réutilisation du formalisme « petites déformations » en grandes déformations (interface `umat`) :
  - grandes rotations, petites déformations (facile)
  - déformations logarithmiques ?
  - corotationnel ?

Annexes

Lois de comportement orthotropes

- il n'est pas possible d'assurer une cohérence entre les différentes hypothèses de modélisation ;
- pour les tubes, on choisit :
  - pour les modélisations $1D$, $2D\,(r,z)$ et $3D$, r est la première direction d'orthotropie, z la seconde et θ la troisième (imposé par le $1D$) ;
  - pour les modélisations $2D$ planes, r est la première direction d'orthotropie, θ la seconde et z la troisième ;
  - conventions particulières pour le tenseur de Hill ;

Lois de comportement orthotropes

```
st2tost2<N,real> H;
if((getModellingHypothesis()==ModellingHypothesis::PLANESTRESS)||
   (getModellingHypothesis()==ModellingHypothesis::PLANESTRAIN)||
   (getModellingHypothesis()==ModellingHypothesis::GENERALISEDPLANESTRAIN)){
   H = hillTensor<N,real>(Hzz,Hrr,Htt,
                          Hrt,Hrz,Htz);
} else {
   H = hillTensor<N,real>(Htt,Hrr,Hzz,
                          Hrz,Hrt,Htz);
}
```

- il n'est pas possible d'assurer une cohérence entre les différentes hypothèses de modélisation ;
- pour les tubes, on choisit :
  - pour les modélisations $1D$, $2D\,(r,z)$ et $3D$, r est la première direction d'orthotropie, z la seconde et θ la troisième (imposé par le $1D$) ;
  - pour les modélisations $2D$ planes, r est la première direction d'orthotropie, θ la seconde et z la troisième ;
  - conventions particulières pour le tenseur de Hill ;

Tableaux de variables internes

```
@MaterialProperty real A[2];      /* Norton coefficient    */
@MaterialProperty real E[2];      /* Norton exponant       */
...
@StateVariable real p[2];       /* Equivalent viscoplastic strain */
@StateVariable Stensor evp[2]; /* Viscoplastic strain             */
...
@Derivative{
  ...
  for(unsigned short i=0;i!=2;++i){
    dp[i]    = A[i]*pow(sigeq,E[i]);
    devp[i]  = dp[i]*n;
    deel    -= devp[i];
  }
}
```

- regroupement des équations de même formalisme :
  - plasticité cristalline, NTFA ;
  - aujourd'hui limité aux algorithmes de Runge-Kutta ;

- il est possible de « compléter » l'interface `umat` :
  - noms et nombre des propriétés matériaux ;
  - noms et nombre des variables internes ;
  - noms et nombre des variables externes ;
  - nom du fichier source ;
- définition de paramètres qui ne passent pas par l'appel standard :
  - paramètres physiques ;
  - valeur des critères de convergence ;
  - nombre maximal d'itérations ;
  - fonctions supplémentaires pour fixer la valeur des paramètres ;