Présentation du générateur de code `mfront`

Thomas Helfer, É. Castelier

26 novembre 2009

- **gérer** les connaissances matériau :
  - les propriétés matériau (module d'Young, etc...) ;
  - les lois de comportements mécaniques (viscoplasticité, plasticité, endommagement) ;
  - les modèles (gonflement, évolution physico-chimique) ;

- **gérer** les connaissances matériau :
  - les propriétés matériau (module d'Young, etc...) ;
  - les lois de comportements mécaniques (viscoplasticité, plasticité, endommagement) ;
  - les modèles (gonflement, évolution physico-chimique) ;
- **mutualiser** ces connaissances matériau :
  - entre les différentes études `Cast3M` ;
  - avec d'autres codes :
    - METEOR, EXCEL ...
    - quelque soit leur langage (fortran,C,C++, etc..) ;

- **gérer** les connaissances matériau :
  - les propriétés matériau (module d'Young, etc...) ;
  - les lois de comportements mécaniques (viscoplasticité, plasticité, endommagement) ;
  - les modèles (gonflement, évolution physico-chimique) ;
- **mutualiser** ces connaissances matériau :
  - entre les différentes études `Cast3M` ;
  - avec d'autres codes :
    - METEOR, EXCEL ...
    - quelque soit leur langage (fortran,C,C++, etc..) ;
- **simplifier** le travail des utilisateurs :
  - numérique ;
  - informatique ;
  - minimiser le risque d'erreur ;

- **gérer** les connaissances matériau :
  - les propriétés matériau (module d'Young, etc...) ;
  - les lois de comportements mécaniques (viscoplasticité, plasticité, endommagement) ;
  - les modèles (gonflement, évolution physico-chimique) ;
- **mutualiser** ces connaissances matériau :
  - entre les différentes études `Cast3M` ;
  - avec d'autres codes :
    - METEOR, EXCEL ...
    - quelque soit leur langage (fortran,C,C++, etc..) ;
- **simplifier** le travail des utilisateurs :
  - numérique ;
  - informatique ;
  - minimiser le risque d'erreur ;
- Deux développements complémentaires :
  - appels externes ;
  - générateur de code `mfront` ;

- conductivité thermique du combustible *UPuC* :

$$k\left(T, p, \tau\right)$$

  - T est la température ;
  - p est la porosité ;
  - τ est le taux de combustion ;

- conductivité thermique du combustible *UPuC* :

$$k\left(T, p, \tau\right)$$

  - T est la température ;
  - p est la porosité ;
  - τ est le taux de combustion ;
- introduction en 3 étapes :
  - écriture d'une fonction `UPuC_ThermalConductivity` ;
  - création d'une librairie `libUPuCMaterialProperties.so` ;
  - appel depuis `Cast3M` ;

- deux possibilités :
  - écrire directement la loi en `C/C++/fortran` ;
  - utiliser un générateur de code ;

```
#ifdef __cplusplus
extern "C"{
#endif /* __cplusplus */

unsigned short
UPuC_ThermalConductivity_nargs = 3u ;

double
UPuC_ThermalConductivity(const double * const castem_params)
{
  using namespace std ;
  const double T = castem_params[0] ;
  const double p = castem_params[1] ;
  const double Bu = castem_params[2] ;
  double k ;
  if( T <= 773.15){
    k = ( 8.14e-6 * T - 0.010096882) * T + 19.65063040915 ;
  } else{
    k = ( -1.88e-6 * T + 0.009737044) * T + 10.2405949657 ;
  }
  k *= ( 1. - p) /( 1. + 2. * p) ;
  k *=  1. -( 0.02 * Bu) ;
  return k ;
} // end of UPuC_ThermalConductivity

#ifdef __cplusplus
} // end of extern "C"
#endif /* __cplusplus */
```

```
#ifdef __cplusplus
extern "C"{
#endif /* __cplusplus */

unsigned short
UPuC_ThermalConductivity_nargs = 3u ;

double
UPuC_ThermalConductivity(const double * const castem_params)
{
  using namespace std ;
  const double T = castem_params[0] ;
  const double p = castem_params[1] ;
  const double Bu = castem_params[2] ;
  double k ;
  if( T <= 773.15){
    k = ( 8.14e-6 * T - 0.010096882) * T + 19.65063040915 ;
  } else{
    k = ( -1.88e-6 * T + 0.009737044) * T + 10.2405949657 ;
  }
  k *= ( 1. - p) /( 1. + 2. * p) ;
  k *=  1. -( 0.02 * Bu) ;
  return k ;
} // end of UPuC_ThermalConductivity

#ifdef __cplusplus
} // end of extern "C"
#endif /* __cplusplus */
```

- `g++ -O2 -fPIC -DPIC -shared UPuC_ThermalConductivity.cxx -o libUPuCMaterialProperties.so`

```
@Parser   MaterialLaw ;
@Law      ThermalConductivity ;
@Material UPuC ;
@Author   Thomas Helfer ;

@Output k ;  //< changing the name of output

@Input T,p,Bu ;                        //< inputs of the law

@Function{
  if (T<=773.15){
    k = (8.14e-6*T-0.010096882)*T+19.65063040915 ;
  } else {
    k = (-1.88e-6*T+0.009737044)*T+10.2405949657 ;
  }
  k *= (1.-p)/(1.+2.*p) ;
  k *= 1.-(0.02*Bu) ;
} // end of function
```

```
@Parser   MaterialLaw ;
@Law      ThermalConductivity ;
@Material UPuC ;
@Author   Thomas Helfer ;

@Output k ;  //< changing the name of output

@Input T,p,Bu ;                        //< inputs of the law
T.setGlossaryName("Temperature") ;     //< pleiades name
p.setGlossaryName("Porosity") ;        //< pleiades name
Bu.setGlossaryName("BurnUp") ;         //< pleiades name

@PhysicalBounds T in [0 :*[ ;  //< temperature physical bounds
@Bounds T in [0 :2573.15] ;    //< temperature bounds
@PhysicalBounds p in [0 :1] ;  //< porosity physical bounds
@PhysicalBounds Bu in [0 :*[ ; //< burn-up   physicalbounds

@Function{
  if (T<=773.15){
    k = (8.14e-6*T-0.010096882)*T+19.65063040915 ;
  } else {
    k = (-1.88e-6*T+0.009737044)*T+10.2405949657 ;
  }
  k *= (1.-p)/(1.+2.*p) ;
  k *= 1.-(0.02*Bu) ;
} // end of function
```

```
@Parser   MaterialLaw ;
@Law      ThermalConductivity ;
@Material UPuC ;
@Author   Thomas Helfer ;

@Output k ;  //< changing the name of output

@Input T,p,Bu ;                        //< inputs of the law
T.setGlossaryName("Temperature") ;     //< pleiades name
p.setGlossaryName("Porosity") ;        //< pleiades name
Bu.setGlossaryName("BurnUp") ;         //< pleiades name

@PhysicalBounds T in [0 :*[ ;  //< temperature physical bounds
@Bounds T in [0 :2573.15] ;    //< temperature bounds
@PhysicalBounds p in [0 :1] ;  //< porosity physical bounds
@PhysicalBounds Bu in [0 :*[ ; //< burn-up  physicalbounds

@Function{
  if (T<=773.15){
    k = (8.14e-6*T-0.010096882)*T+19.65063040915 ;
  } else {
    k = (-1.88e-6*T+0.009737044)*T+10.2405949657 ;
  }
  k *= (1.-p)/(1.+2.*p) ;
  k *= 1.-(0.02*Bu) ;
} // end of function
```

- `mfront -obuild -interface=castem UPuC_ThermalConductivity.mfront`

- un fichier clair (avis subjectif) ;

- un fichier clair (avis subjectif) ;
- interfaces disponibles :
  - castem (!) ;
  - Excel (Visual Basic) ;
  - C/C++/fortran ;
  - python ;
  - octave ;
  - gnuplot ;
  - etc...

- un fichier clair (avis subjectif) ;
- interfaces disponibles :
  - castem (!) ;
  - Excel (Visual Basic) ;
  - C/C++/fortran ;
  - python ;
  - octave ;
  - gnuplot ;
  - etc...
- gestion facilitée des bornes des propriétés matériau ;

- un fichier clair (avis subjectif) ;
- interfaces disponibles :
  - castem (!) ;
  - Excel (Visual Basic) ;
  - C/C++/fortran ;
  - python ;
  - octave ;
  - gnuplot ;
  - etc...
- gestion facilitée des bornes des propriétés matériau ;
- gestion facilitée des dépendances entre propriétés matériau ;

- un fichier clair (avis subjectif) ;
- interfaces disponibles :
  - castem (!) ;
  - Excel (Visual Basic) ;
  - C/C++/fortran ;
  - python ;
  - octave ;
  - gnuplot ;
  - etc...
- gestion facilitée des bornes des propriétés matériau ;
- gestion facilitée des dépendances entre propriétés matériau ;
- interaction avec la base de données matériau `sirius` :
  - en entrée (en développement) ;
  - en sortie ;

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

* 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
* dans laquelle cette fonction est définie
* - 'VARIABLES' contient la liste des paramètres dont dépend
* la fonction appelée
Tmat = 'TABLE' ;
Tmat. 'MODELE' = 'UPuC_ThermalConductivity' ;
Tmat. 'LIBRAIRIE' = 'libUPuCMaterialProperties.so' ;
Tmat. 'VARIABLES' = 'MOTS' 'T' 'PORO' 'FIMA' ;

* Création du matériau.
MatT1 = 'MATERIAU' ModT1 'K' Tmat ;
```

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

* 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
* dans laquelle cette fonction est définie
* - 'VARIABLES' contient la liste des paramètres dont dépend
* la fonction appelée
Tmat = 'TABLE' ;
Tmat. 'MODELE' = 'UPuC_ThermalConductivity' ;
Tmat. 'LIBRAIRIE' = 'libUPuCMaterialProperties.so' ;
Tmat. 'VARIABLES' = 'MOTS' 'T' 'PORO' 'FIMA' ;

* Création du matériau.
MatT1 = 'MATERIAU' ModT1 'K' Tmat ;
```

- utilisation transparente dans les procédures classiques (PASAPAS) ;

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

* 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
* dans laquelle cette fonction est définie
* - 'VARIABLES' contient la liste des paramètres dont dépend
* la fonction appelée
Tmat = 'TABLE' ;
Tmat. 'MODELE' = 'UPuC_ThermalConductivity' ;
Tmat. 'LIBRAIRIE' = 'libUPuCMaterialProperties.so' ;
Tmat. 'VARIABLES' = 'MOTS' 'T' 'PORO' 'FIMA' ;

* Création du matériau.
MatT1 = 'MATERIAU' ModT1 'K' Tmat ;
```

- utilisation transparente dans les procédures classiques (PASAPAS) ;
- utilisation simple de lois multi-variables :
  - les paramètres doivent être définies par des « chargements » ;

- Les lois de comportements sont par nature complexes :
  - **intégrer** une équation différentielle ;

- Les lois de comportements sont par nature complexes :
  - **intégrer** une équation différentielle ;
  - variables scalaires ou tensorielles ;

- Les lois de comportements sont par nature complexes :
  - **intégrer** une équation différentielle ;
  - variables scalaires ou tensorielles ;
  - cas très différents :
    - plasticité, endommagement, surfaces de charges ;
    - orthotropie ;

- Les lois de comportements sont par nature complexes :
  - **intégrer** une équation différentielle ;
  - 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) ;

- Les lois de comportements sont par nature complexes :
  - **intégrer** une équation différentielle ;
  - 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 ;

- 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 ;

- 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) ;

- 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) ;

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

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

  - écoulement athermique :

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

```
@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 ;
}
```

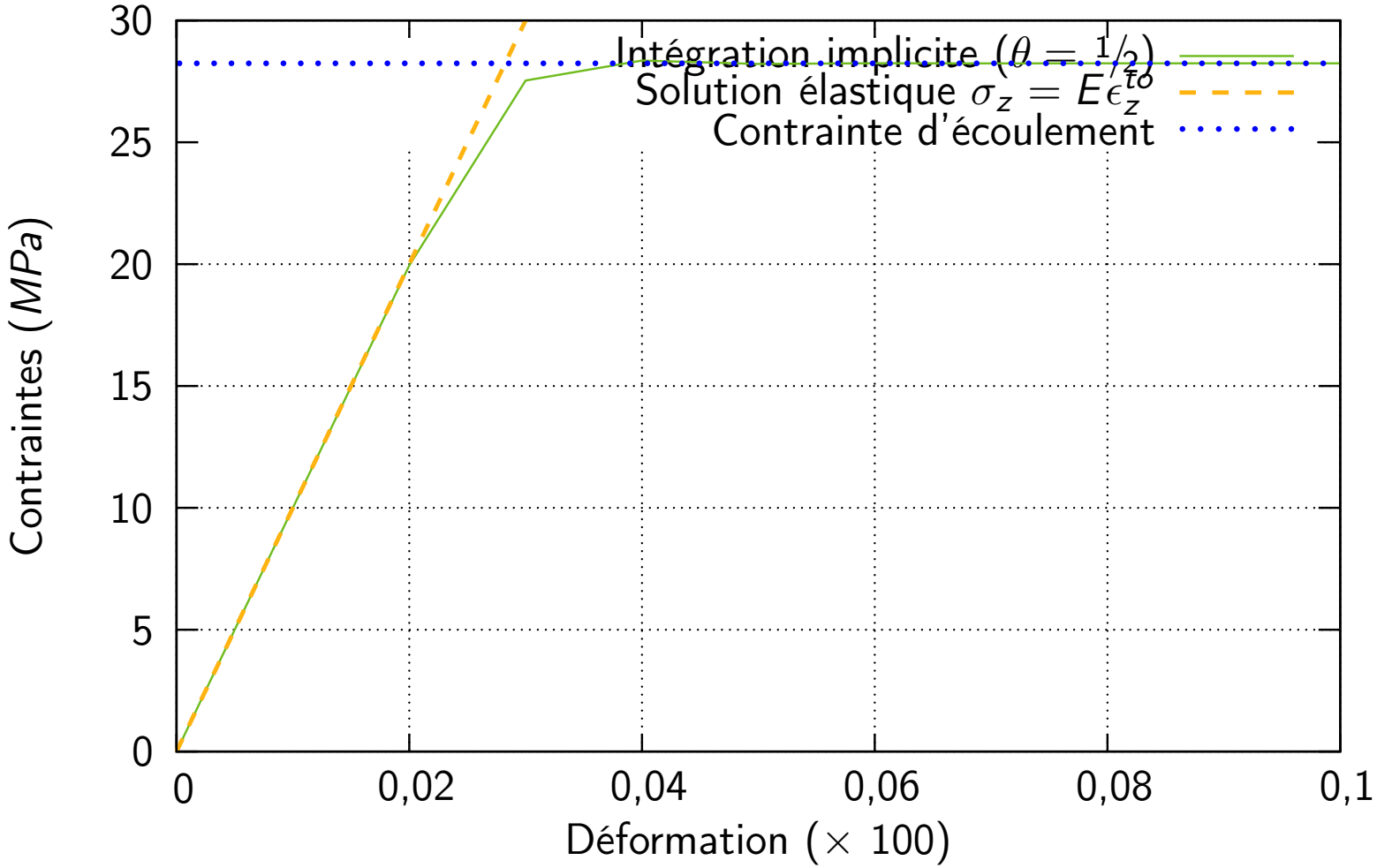
```
@Parser    MultipleIsotropicMisesFlows ;
@Material  NbZrC ;
@Behaviour Creep ;
@Author    É. Brunon ;

@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`

```
* 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;
```

```
* 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' ;



- essai de traction à vitesse imposée

- testés sur :
  - castem 2006, 2007, 2008 ;
  - `LiNuX` 32/64 bits ;
  - `Windows` 32 bits ;
- très bon retour utilisateur ;
- utilisés dans les codes « `pleiades` » :
  - en cours de standardisation ;
- **proposition d'intégration dans** `Cast3M` :
  - propriété matériaux partiellement acceptée :
    - une option à décommenter ;
  - les lois de comportements ?

- version actuelle sous licence libre ;

- version actuelle sous licence libre ;
- lois de comportement générées efficaces :
  - gain d'un facteur 3 à 8 par rapport à `Cast3M` :
    - en fonction des cas ;
    - sur les cas testés (!) ;

- version actuelle sous licence libre ;
- lois de comportement générées efficaces :
  - gain d'un facteur 3 à 8 par rapport à `Cast3M` :
    - en fonction des cas ;
    - sur les cas testés (!) ;
- nouvelles lois de comportement :
  - éléments joints (zones cohésives) ;
  - lois multi-physiques (`UMATDIFF`) ;
  - `etc..`

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

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

```
@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             */
```

```
@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 ;
}
```