MFRON03 – Test de l'interface *Code_Aster-MFront* pour des lois avec orthotropie

Résumé :

Ce test valide des comportements orthotrope définis à l'aide de *MFront* par comparaison avec des comportement similaires de *Code_Aster* .

Modélisation A : cette modélisation permet de valider le modèle élastique orthotrope par comparaison au test SSNV225C sur un point matériel.

Modélisation B : cette modélisation permet de valider le modèle élasto-visco-plastique monocristallin avec intégration implicite, par comparaison au modèle MONOCRISTAL du test zmat007a sur un agrégat à 10 grains.

Modélisation C : cette modélisation permet de valider le modèle élasto-visco-plastique monocristallin avec intégration implicite, et définition complète de la famille de systèmes de glissement et de la matrice d'interaction, par comparaison au modèle MONOCRISTAL du test zmat007a sur un agrégat à 10 grains.

1 Problème de référence

1.1 Géométrie

La géométrie de la modélisation A est identique à celle du test SSLV113A.

La géométrie de la modélisation B est identique à celle du test ZMAT007A (agrégat à 10 grains).

1.2 Propriétés des matériaux

Les coefficients du comportement Mfront sont, pour la modélisation A :

C1	11000	E_L
C2	5000	E_T
C5	8000	E_N
C6	0,396	NU_LT
C7	0,11	NU_TN
C8	0,15	NU_LN
C9	10500	G_LT
C10	13000	G_TN
C11	7000	G_LN

Le fichier Mfront définissant le comportement élastique orthotrope est :

```
@Parser    Implicit;

@Behaviour elasorth;

@Algorithm NewtonRaphson_NumericalJacobian;


@OrthotropicBehaviour;
@RequireStiffnessTensor;

@Theta 1.;
@AuxiliaryStateVariable real seq;

@TangentOperator{
  Dt = D;
}

@ComputeStress{
  sig = D*eel;
}
@Integrator{
  feel += -deto;
}

@UpdateAuxiliaryStateVars{
  seq = sqrt(sig|sig);
}
```

Le fichier Mfront définissant le comportement monocristallin de type CFC avec élasticité isotrope (modélisation B) est :

```
@Parser    Implicit;
@Behaviour monocristal;
@Algorithm NewtonRaphson_NumericalJacobian;
@OrthotropicBehaviour;
@IsotropicElasticBehaviour;
@RequireStiffnessTensor;
@Theta 1.;
@Epsilon 1.e-8 ;
@IterMax 100 ;
@MaterialProperty stress young;
@MaterialProperty real nu;
@MaterialProperty real m;
@MaterialProperty real K;
@MaterialProperty real C;
@MaterialProperty real R0;
@MaterialProperty real Q;
@MaterialProperty real b;
@MaterialProperty real d1;
@Includes{
#include"TFEL/Material/Lame.hxx"
#include"TFEL/Material/MetallicCFCSlidingSystems.hxx"
#include"TFEL/Material/MetallicCFCGenericSlidingSystemsInteractionMatrix.hxx"
}
@Link{"-lTFELMaterialSingleCristals"};
@StateVariable strain    g[12];
@AuxiliaryStateVariable real p[12];
@AuxiliaryStateVariable real a[12];
@LocalVariable real    lambda;
@LocalVariable real    mu;
@Parameter h1,h2,h3,h4,h5,h6;
h1.setDefaultValue(1.);
h2.setDefaultValue(0.);
h3.setDefaultValue(0.);
h4.setDefaultValue(0.);
h5.setDefaultValue(0.);
h6.setDefaultValue(0.);
/* Initialize Lame coefficients */
@InitLocalVars{
  using namespace tfel::material::lame;
  lambda = computeLambda(young,nu);
  mu = computeMu(young,nu);
}
@TangentOperator{
  using namespace tfel::material::lame;
  StiffnessTensor Hooke;
  Stensor4 Je;
  computeElasticStiffness<N,Type>::exe(Hooke,lambda,mu);
  getPartialJacobianInvert(Je);
  Dt = Hooke*Je;
}
@ComputeStress{
    sig = (lambda*trace(eel)*Stensor::Id()+2*mu*eel);
}
@Integrator{
  typedef MetallicCFCSlidingSystems<N> CFCSlidingSystems;
  typedef MetallicCFCGenericSlidingSystemsInteractionMatrix InteractionMatrix;
  const CFCSlidingSystems& ss = CFCSlidingSystems::getMetallicCFCSlidingSystems();
```

```
const tmatrix<12,12,double>& h  =
 InteractionMatrix::getInteractionMatrix(h1,h2,h3,h4,h5,h6);
  StrainStensor vepsp(real(0));
  real tau[12];
  real vp[12];
  real va[12];
  real ag[12];
  real tma[12];
  real tmR[12];
  real Rp[12] ;
  real pe[12] ;
  for(unsigned short i=0;i!=12;++i){
     ag[i]  = abs(dg[i]);
     pe[i] = Q*(1.-exp(-b*(p[i]+ag[i]))) ;
  }
  for(unsigned short i=0;i!=12;++i){
     const stensor<N>& mus = ss.mus[i];
     Rp[i] = R0 ;
     for(unsigned short j=0;j!=12;++j){
        Rp[i] +=h(i,j)*pe[i] ;
     }
     tau[i] = mus | sig ;
     va[i]  = (dg[i]-d1*a[i]*ag[i])/(1.+d1*ag[i]);
     tma[i] = tau[i]-C*(va[i]+a[i]) ; ;
     tmR[i] = abs(tma[i])-Rp[i] ;
     if (tmR[i]>0.){
         real sgn=tma[i]/abs(tma[i]);
         vp[i]  = dt*sgn*pow((tmR[i]/K),m);
         }
     else{
         vp[i]=0.;
     }
     vepsp+=vp[i]*mus ;
  }
  feel += vepsp-deto;
  for(unsigned short i=0;i!=12;++i){
    fg[i]  -= vp[i];
  }
}
@UpdateAuxiliaryStateVars{
  for(unsigned short i=0;i!=12;++i){
     p[i]+=abs(dg[i]);
     a[i]+=(dg[i]-d1*a[i]*p[i])/(1.+d1*p[i]);
  }
}
```

Les fichiers Mfront définissant le comportement monocristallin avec élasticité orthotrope (modélisation C) et définition de la famille de systèmes de glissement et de la matrice d'interaction sont :

```
@Parser    Implicit;
@Behaviour monocristal;
@Algorithm NewtonRaphson_NumericalJacobian;
@OrthotropicBehaviour;
@RequireStiffnessTensor;
@Theta 1.;
@Epsilon 1.e-8 ;
@IterMax 100 ;
@MaterialProperty real ind;
@MaterialProperty real m;
@MaterialProperty real K;
@MaterialProperty real C;
@MaterialProperty real R0;
```

```
@MaterialProperty real Q;
@MaterialProperty real b;
@MaterialProperty real H;
@MaterialProperty real d1;
@StateVariable strain    g[12];
@AuxiliaryStateVariable real p[12];
@AuxiliaryStateVariable real a[12];
@Includes{
#include"TFEL/FSAlgorithm/copy.hxx"
}
@TangentOperator{
  Stensor4 Je;
  getPartialJacobianInvert(Je);
  Dt = D*Je;
}
@Import "CFCGlidingSystems.mfront";
@Import "CFCInteractionMatrix.mfront";
@ComputeStress{
  sig = D*eel;
}
@Integrator{
  StrainStensor vepsp(real(0));
  real tau[12];
  real vp[12];
  real va[12];
  real ag[12];
  real tma[12];
  real tmR[12];
  real Rp[12] ;
  real pe[12] ;
  // systemes de glissement
  for(unsigned short i=0;i!=12;++i){
    ag[i]  = abs(dg[i]);
    pe[i] = Q*(1.-exp(-b*(p[i]+ag[i]))) ;
  }
  for(unsigned short i=0;i!=12;++i){
    Rp[i] = R0 ;
     for(unsigned short j=0;j!=12;++j){
        Rp[i] +=mh(i,j)*pe[i] ;
     }
    tau[i] = mus[i] | sig ;
    va[i]  = (dg[i]-d1*a[i]*ag[i])/(1.+d1*ag[i]);
    tma[i] = tau[i]-C*(va[i]+a[i]) ;
    tmR[i] = abs(tma[i])-Rp[i] ;
    if (tmR[i]>0.){
      real sgn=tma[i]/abs(tma[i]);
      vp[i]  = dt*sgn*pow((tmR[i]/K),m);
    }
    else{
      vp[i]=0.;
    }
    vepsp+=vp[i]*mus[i] ;
  }
  feel += vepsp-deto;
  for(unsigned short i=0;i!=12;++i){
    fg[i]  -= vp[i];
  }
}
@UpdateAuxiliaryStateVars{
  for(unsigned short i=0;i!=12;++i){
```

```
    p[i]+=abs(dg[i]);
    a[i]+=(dg[i]-d1*a[i]*p[i])/(1.+d1*p[i]);
  }
}
```

La famille de systèmes de glissement et de la matrice d'interaction sont définis par

```
@LocalVariable tfel::math::tvector<12,StrainStensor> mus;
@InitLocalVariables{
  const real coefm=1.0/sqrt(2.);
  const real coefn=1.0/sqrt(3.);
  const real nx[12]={ 1.0,1.0,1.0, 1.0, 1.0, 1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0};
  const real ny[12]={ 1.0,1.0,1.0,-1.0,-1.0,-1.0, 1.0, 1.0, 1.0,-1.0,-1.0,-1.0};
  const real nz[12]={ 1.0,1.0,1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
  const real mx[12]={-1.0, 0.0,-1.0,-1.0,0.0,1.0, 0.0,1.0,1.0,-1.0,1.0,0.0};
  const real my[12]={ 0.0,-1.0, 1.0, 0.0,1.0,1.0,-1.0,1.0,0.0, 1.0,0.0,1.0};
  const real mz[12]={ 1.0, 1.0, 0.0, 1.0,1.0,0.0, 1.0,0.0,1.0, 0.0,1.0,1.0};
  for(unsigned short i=0;i!=12;++i){
    tvector<3,real> ns(real(0));
    tvector<3,real> ms(real(0));
    stensor<3,real> mu;
    ns[0]=nx[i]*coefn;
    ns[1]=ny[i]*coefn;
    ns[2]=nz[i]*coefn;
    ms[0]=mx[i]*coefm;
    ms[1]=my[i]*coefm;
    ms[2]=mz[i]*coefm;
    mu[0]=ns[0]*ms[0];
    mu[1]=ns[1]*ms[1];
    mu[2]=ns[2]*ms[2];
    mu[3]=(ns[0]*ms[1]+ns[1]*ms[0])*0.5*sqrt(2);
    mu[4]=(ns[0]*ms[2]+ns[2]*ms[0])*0.5*sqrt(2);
    mu[5]=(ns[1]*ms[2]+ns[2]*ms[1])*0.5*sqrt(2);
    tfel::fsalgo::copy<StensorSize>::exe(mu.begin(),
                 mus[i].begin());
  }
}
//! interaction matrix
@LocalVariable tfel::math::tmatrix<12,12,real> mh;
@InitLocalVariables{
  const real h1 = 1.;
  const real h2 = 0.;
  const real h3 = 0.;
  const real h4 = 0.;
  const real h5 = 0.;
  const real h6 = 0.;
  for(unsigned short i=0;i!=12;++i){
    mh(i,i)=h1;
  }
  mh(1,0)=mh(0,1)=h2;
  mh(2,0)=mh(0,2)=h2;
  mh(2,1)=mh(1,2)=h2;
  mh(3,0)=mh(0,3)=h4;
  mh(3,1)=mh(1,3)=h5;
  mh(3,2)=mh(2,3)=h5;
  mh(4,0)=mh(0,4)=h5;
  mh(4,1)=mh(1,4)=h3;
  mh(4,2)=mh(2,4)=h6;
  mh(4,3)=mh(3,4)=h2;
  mh(5,0)=mh(0,5)=h5;
  mh(5,1)=mh(1,5)=h6;
  mh(5,2)=mh(2,5)=h3;
```

```
  mh(5,3)=mh(3,5)=h2;
  mh(5,4)=mh(4,5)=h2;
  mh(6,0)=mh(0,6)=h5;
  mh(6,1)=mh(1,6)=h4;
  mh(6,2)=mh(2,6)=h5;
  mh(6,3)=mh(3,6)=h6;
  mh(6,4)=mh(4,6)=h3;
  mh(6,5)=mh(5,6)=h5;
  mh(7,0)=mh(0,7)=h6;
  mh(7,1)=mh(1,7)=h5;
  mh(7,2)=mh(2,7)=h3;
  mh(7,3)=mh(3,7)=h5;
  mh(7,4)=mh(4,7)=h5;
  mh(7,5)=mh(5,7)=h4;
  mh(7,6)=mh(6,7)=h2;
  mh(8,0)=mh(0,8)=h3;
  mh(8,1)=mh(1,8)=h5;
  mh(8,2)=mh(2,8)=h6;
  mh(8,3)=mh(3,8)=h3;
  mh(8,4)=mh(4,8)=h6;
  mh(8,5)=mh(5,8)=h5;
  mh(8,6)=mh(6,8)=h2;
  mh(8,7)=mh(7,8)=h2;
  mh(9,0)=mh(0,9)=h5;
  mh(9,1)=mh(1,9)=h5;
  mh(9,2)=mh(2,9)=h4;
  mh(9,3)=mh(3,9)=h6;
  mh(9,4)=mh(4,9)=h5;
  mh(9,5)=mh(5,9)=h3;
  mh(9,6)=mh(6,9)=h6;
  mh(9,7)=mh(7,9)=h3;
  mh(9,8)=mh(8,9)=h5;
  mh(10,0)=mh(0,10)=h3;
  mh(10,1)=mh(1,10)=h6;
  mh(10,2)=mh(2,10)=h5;
  mh(10,3)=mh(3,10)=h3;
  mh(10,4)=mh(4,10)=h5;
  mh(10,5)=mh(5,10)=h6;
  mh(10,6)=mh(6,10)=h5;
  mh(10,7)=mh(7,10)=h5;
  mh(10,8)=mh(8,10)=h4;
  mh(10,9)=mh(9,10)=h2;
  mh(11,0)=mh(0,11)=h6;
  mh(11,1)=mh(1,11)=h3;
  mh(11,2)=mh(2,11)=h5;
  mh(11,3)=mh(3,11)=h5;
  mh(11,4)=mh(4,11)=h4;
  mh(11,5)=mh(5,11)=h5;
  mh(11,6)=mh(6,11)=h3;
  mh(11,7)=mh(7,11)=h6;
  mh(11,8)=mh(8,11)=h5;
  mh(11,9)=mh(9,11)=h2;
  mh(11,10)=mh(10,11)=h2;
}
```

1.3 Conditions aux limites et chargements

Les chargements pour les modélisations A et B sont identiques aux tests SSLV131A et ZMAT007A.

2 Solution de référence

Valeurs des contraintes, déformations et variables internes, par inter-comparaison avec les tests SSLV131A et ZMAT007A.

3 Modélisation A

3.1 Caractéristiques de la modélisation

Modélisation 3D, identique à SSLV131A.

3.2 Grandeurs testées et résultats

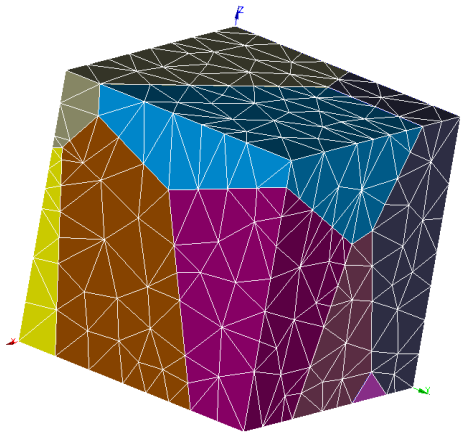
Comparaison avec SSLV131A

Identification	Référence	Tolérance %
champ de déplacement		
$dy(c)$	21	1.E-5
champ EPSI_ELGA		
$EPXY$	3	1.E-5
$EPXZ$	4	1.E-5
$EPYZ$	6	1.E-5
champ SIEF_ELGA		
$SIXX$	601.8754	1.E-5
$SIYY$	80053.665	1.E-5
$SIZZ$	78596.607	1.E-5
$SIXY$	83948.263	1.E-5
$SIXZ$	17339.093	1.E-5
$SIYZ$	126571.71	1.E-5

4 Modélisation B

4.1 Caractéristiques de la modélisation

La géométrie est celle d'un agrégat à 10 grains générée par une procédure python basée sur des cellules de Voronoï. On définit des plans de coupe aux bords pour imposer les conditions aux limites.



4.2 Grandeurs testées et résultats

Comparaison avec ZMAT007A.
Comparaison des résultats obtenus avec *Code_Aster* à 0.3% de déformation globale.

Identification	*référence*	Tolérance %
σ_{xx} de SIEF_ELGA	-16.112	0,01
ε_{xx} de EPSI_ELGA	-1,0250E-03	0,01
ε_{yy} de EPSI_ELGA	-9.3714E-04	0,01
ε_{yy} de EPSP_ELGA	-2 .7745E-04	0,01

5 Modélisation c

5.1 Caractéristiques de la modélisation

Elle est identique à celle de la modélisation B.

5.2 Grandeurs testées et résultats

Comparaison avec ZMAT007A.

Comparaison des résultats obtenus avec *Code_Aster* à 0.3% de déformation globale.

Identification	*référence*	Tolérance %
σ_{xx} de SIEF_ELGA	-16.112	0,01
ε_{xx} de EPSI_ELGA	-1,0250E-03	0,01
ε_{yy} de EPSI_ELGA	-9.3714E-04	0,01
ε_{yy} de EPSP_ELGA	-2 .7745E-04	0,01

6 Synthèse des résultats

Les résultats sont satisfaisants et valident l'interface entre *Code_Aster* et MFRONT en 3D, pour des comportements avec orthotropie.