MFRON02 – Test de l'interface Code_Aster-MFront avec des lois d'endommagement

Résumé :

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

Modélisation A : cette modélisation permet de valider le modèle élasto-visco-plastique avec endommagement de Hayhurst et intégration implicite, par comparaison au modèle HAYHURST du test SSNV225C sur un point matériel.

Modélisation B : cette modélisation permet de valider le modèle élasto-visco-plastique avec endommagement de Hayhurst et intégration explicite, par comparaison au modèle HAYHURST du test SSNV225A sur un point matériel.

Modélisation C : cette modélisation permet de valider le modèle avec endommagement de Mazars, par comparaison au modèle MAZARS du test SSNP113A sur un élément 3D.

Modélisation D : cette modélisation permet de valider le modèle avec endommagement de Mazars, par comparaison au modèle MAZARS du test SSLA103C sur un élément axisymétrique, avec prise en compte du séchage et de l'hydratation.

Modélisation E : cette modélisation permet de valider le modèle avec endommagement de Mazars, par comparaison au modèle MAZARS du test SSNP161A sur un élément en contraintes planes.

1 Problème de référence

1.1 Géométrie

La géométrie des modélisations A et B est un point matériel. Celle de la modélisation C est identique à celle du test SSNP113A.

1.2 Propriétés des matériaux

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

C1	145000	Young
C2	0.3	Poisson
C5	9,691	K
C6	5,82516E-11	eps0
C7	27,9317	sig0
C8	3,E4	h1
C9	-280	h2
C10	0,33	H1*
C11	1	H2*
C12	9.707593E-08	A0
C13	0,5	AlphaD
C14	1.	DELTA1
C14	0	DELTA2

Le fichier Mfront définissant le comportement élastoplastique de Hayhurst en implicite est :

```
@Parser Implicit;
@Behaviour Hayhurst;
@Theta 0.5 ;
@MaterialProperty stress young;
@MaterialProperty real nu;
@MaterialProperty real rho;
@MaterialProperty real alpha;
@MaterialProperty real K;
@MaterialProperty real epsi0;
@MaterialProperty real sigma0;
@MaterialProperty real h1;
@MaterialProperty real h2;
@MaterialProperty real H1star;
@MaterialProperty real H2star;
@MaterialProperty real A0;
@MaterialProperty real alphaD;
@MaterialProperty real delta1;
@MaterialProperty real delta2;
@Includes{
#include"TFEL/Material/Lame.hxx"
}
@StateVariable real p;
@StateVariable real H1;
@StateVariable real H2;
```

```
@StateVariable real    d;
@LocalVariable real    lambda;
@LocalVariable real    mu;
/* Initialize Lamé 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;
  Stensor Jd;
  computeElasticStiffness<N, Type>::exe(Hooke, lambda, mu);
  getPartialJacobianInvert(Je, Jd);
  Dt = (1-d-dd)*Hooke*Je+-((Jd)^(Hooke*(eel+deel)));
}
@ComputeStress{
  if(d > 1.-1.e-8){
    sig= Stensor(0.);
  } else {
    sig = (1.-d)*(lambda*trace(eel)*Stensor::Id()+2*mu*eel);
  }
}
@Integrator{
  Stensor n = Stensor(0.);
  const real seq = sigmaeq(sig);
  Stensor sig0 = (lambda*trace(eel)*Stensor::Id()+2*mu*eel);
  const real seq0 = sigmaeq(sig0);
  if(seq > 1.e-8*young){
    const real H1_ = H1+theta*dH1;
    const real H2_ = H2+theta*dH2;
    const real d_ = d+theta*dd;
    const real H_ = H1_+H2_;
    const real shp = sinh(seq*(1-H_)/K/(1-(d_)));
    const real chp = sqrt(1.+shp*shp);
    const real trsig=max(trace(sig),0.);
    const real trsig0=max(trace(sig0),0.);
    const real shd = sinh((alphaD*trsig + (1-alphaD)*seq)/sigma0);
    const real chd = sqrt(1.+shd*shd);
    const real inv_seq = 1/seq;
    const real dtrside=(3.*lambda+2.*mu)*theta*(1.-d_)*trsig/trace(sig);
    n = 1.5*deviator(sig)*inv_seq;
    Stensor dsvmde=2.*mu*theta*(1.-d_)*n;
    Stensor dsvm0de=2.*mu*theta*n;
    // systeme a resoudre
    feel += dp*n-deto;
    fp = dp-epsi0*dt*shp;
    fH1 = dH1-h1*dp*(H1star-delta1*H1_)*inv_seq;
    fH2 = dH2-h2*dp*(H2star-delta2*H2_)*inv_seq;
    fd = dd - A0 * shd*dt;
    // jacobienne
    dfeel_ddeem += 2.*mu*theta*dp*inv_seq*(1-d_)*(Stensor4::M()-(n^n));
    dfeel_ddp = n;
    dfp_ddeem = - (dt*epsi0*chp*(1-H_)/K)*dsvm0de;
    dfp_ddH1 = dt*epsi0*chp*theta*seq0/K;
    dfp_ddH2 = dfp_ddH1;
    dfH1_ddeem = h1*dp*(H1star - delta1*H1_)*inv_seq*inv_seq*dsvmde;
    dfH2_ddeem = h2*dp*(H2star - delta2*H2_)*inv_seq*inv_seq*dsvmde;
```

```
dfH1_ddp = - h1*inv_seq*(H1star - delta1*H1_) ;
dfH2_ddp = - h2*inv_seq*(H2star - delta2*H2_) ;
dfH1_ddH1+= h1*delta1*dp*theta*inv_seq ;
dfH2_ddH2+= h2*delta2*dp*theta*inv_seq ;
dfH1_ddd= h1*dp*(H1star - delta1*H1_)*theta*seq0*inv_seq*inv_seq;
dfH2_ddd= h2*dp*(H2star - delta2*H2_)*theta*seq0*inv_seq*inv_seq;
dfd_ddd += A0*dt*chd/sigma0 *(1-alphaD)*theta*seq0 ;
if (trsig > 1.e-8*young) {
dfd_ddeel=-A0*dt*chd/sigma0*(alphaD*dtrsde*Stensor::Id()+(1-alphaD)*dsvmde);
dfd_ddd += A0*dt*chd/sigma0 *alphaD*theta*trsig0 ;
} else {
dfd_ddeel = -A0*dt*chd/sigma0 *(1-alphaD)*dsvmde;
}
} else {
feel -= deto;      }
}
```

Le fichier Mfront définissant le comportement élastoplastique de Hayhurst en explicite ne diffère du précédent que par l'intégration :

```
@Parser RungeKutta;
@Behaviour Hayhurst;
@Algorithm rk54;
...
@Derivative{
seq = sigmaeq(sig);
Stensor sig0 = (lambda*trace(eel)*Stensor::Id()+2*mu*eel);
const real seq0 = sigmaeq(sig0);
if(seq > 0.01*young){
return false;
}
if(endo > 1.-1.e-8){
sig= Stensor(0.);
dp=0.;
dendo=0.;
dH1=0.;
dH2=0.;
}
real inv_seq(0);
n = Stensor(0.);
if(seq > 1.e-8*young){
H=H1+H2 ;
dp = epsi0*sinh(seq0*(1-H)/K);
dH1=h1/seq * (H1star - delta1*H1) *dp;
dH2=h2/seq * (H2star - delta2*H2) *dp;
trsigplus=0;
if (sequid==0){
real v1;
real v2;
real v3;
sig.computeEigenValues(v1,v2,v3);
trsigplus=max(0.,v1);
}
if (sequid==1){
trsig=trace(sig);
trsigplus=0.;
if(trsig>0){
trsigplus=trsig;
}
}
dendo=A0 * sinh( (alphaD*trsigplus + (1-alphaD)*seq )/sigma0 ) ;
```

```
    inv_seq = 1/seq;  
    n      = 1.5*deviator(sig)*inv_seq;  
  }  
  devp = dp*n ;  
  deel = deto - devp; }
```

Les coefficients matériau pour la modélisation C sont (cf. SSNP113A)

C1	3,2E10	YOUN
C2	0,2	POISS
C5	1,15	AC
C6	0,8	AT
C7	1391,3	BC
C8	10000,	BT
C9	0,7	k
C10	9,37E-5	ed0

Le fichier Mfront définissant le comportement de Mazars par une résolution directe (non itérative) (modélisation C) est :

```
@Parser DefaultParser;  
@Behaviour mazars;  
@ProvidesTangentOperator;  
  
@MaterialProperty stress young;  
@MaterialProperty real nu;  
@MaterialProperty real rho;  
@MaterialProperty real alpha;  
@MaterialProperty real Ac;  
@MaterialProperty real At;  
@MaterialProperty real Bc;  
@MaterialProperty real Bt;  
@MaterialProperty real k;  
@MaterialProperty real ed0;  
  
@Includes{  
#include"TFEL/Material/Lame.hxx"  
}  
  
@AuxiliaryStateVariable real d;  
@AuxiliaryStateVariable real Y;  
@AuxiliaryStateVariable real Tmax;  
@AuxiliaryStateVariable real eeqcor;  
  
@LocalVariable real    lambda;  
@LocalVariable real    mu;  
@LocalVariable real    A;  
@LocalVariable real    B;  
@LocalVariable real    eeqc;  
  
/* Initialize Lamé coefficients */  
@InitLocalVars{  
  using namespace tfel::material::lame;  
  lambda = computeLambda(young,nu);
```

```
mu = computeMu(young, nu);
}

@Integrator{
const Stensor etop=eto+deto;
const real dmax=0.99999;
const real normeps= etop | etop ;
if(normeps<1.e-30){
sig=Stensor (real(0));
}
else{
real e1;
real e2;
real e3;
real ppe1;
real ppe2;
real ppe3;
real pns1;
real pns2;
real pns3;
real pps1;
real pps2;
real pps3;
etop.computeEigenValues(e1,e2,e3);
Stensor sig0 = (lambda*trace(etop)*Stensor::Id()+2*mu*etop);
real s1;
real s2;
real s3;
sig0.computeEigenValues(s1,s2,s3);
real gam=1. ;
if (min(s1,min(s2,s3))<0) {
pns1=min(0.,s1);
pns2=min(0.,s2);
pns3=min(0.,s3);
gam=-sqrt(pns1*pns1+pns2*pns2+pns3*pns3)/(pns1+pns2+pns3);
}
ppe1=max(0.,e1);
ppe2=max(0.,e2);
ppe3=max(0.,e3);
pps1=max(0.,s1);
pps2=max(0.,s2);
pps3=max(0.,s3);
eeqc= gam*sqrt(ppe1*ppe1+ppe2*ppe2+ppe3*ppe3);
eeqcor=max(eeqc,eeqcor);
real r=(pps1+pps2+pps3)/(abs(s1)+abs(s2)+abs(s3));
A=At*(2*r*r*(1.-2*k)-r*(1-4*k))+Ac*(2*r*r-3*r+1);
B=r*r*Bt+(1-r*r)*Bc;
real Y1=max(ed0,eeqcor);
Y=max(Y1,Y);
d=1-(1-A)*ed0/Y-A*exp(-B*(Y-ed0));
d=min(dmax,d);
sig = (1.-d)*(lambda*trace(etop)*Stensor::Id()+2*mu*etop);
}
if(computeTangentOperator_){
using namespace tfel::material::lame;
StiffnessTensor Hooke;
computeElasticStiffness<N,Type>::exe(Hooke,lambda,mu);
Dt = (1.-d)*Hooke;
}
}
```

```
@UpdateAuxiliaryStateVars{
    Tmax=max (Tmax,T) ;
}
```

Le fichier Mfront définissant le comportement de Mazars en implicite (modélisations D et E) est :

```
@Parser Implicit;
@Behaviour mazars;
@Algorithm NewtonRaphson_NumericalJacobian;

@Theta 1. ;
@Epsilon 1.e-12 ;
@IterMax 100 ;

@MaterialProperty stress young;
@MaterialProperty real nu;
@MaterialProperty real rho;
@MaterialProperty real alpha;
@MaterialProperty real Ac;
@MaterialProperty real At;
@MaterialProperty real Bc;
@MaterialProperty real Bt;
@MaterialProperty real k;
@MaterialProperty real ed0;

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

@StateVariable real d;
@AuxiliaryStateVariable real Y;
@AuxiliaryStateVariable real Tmax;
@AuxiliaryStateVariable real eeqcor;

@LocalVariable real lambda;
@LocalVariable real mu;
@LocalVariable real A;
@LocalVariable real B;
@LocalVariable real eeqc;

/* Initialize Lamé 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;
Stensor Jd;
computeElasticStiffness<N, Type>::exe (Hooke, lambda, mu);
getPartialJacobianInvert (Je, Jd);
Dt = (1-d-dd) * Hooke * Je +- ((Jd) ^ (Hooke * (eel+deel)));
// cout << "Dt : " << Dt << endl;
}

@ComputeStress{
sig = (1.-d) * (lambda*trace (eel) *Stensor::Id() +2*mu*eel);
```

```
}
@Integrator{
  const Stensor etop=eto+deto;
  const real dmax=0.99999;
  real e1;
  real e2;
  real e3;
  real ppe1;
  real ppe2;
  real ppe3;
  real pns1;
  real pns2;
  real pns3;
  real pps1;
  real pps2;
  real pps3;
  etop.computeEigenValues(e1,e2,e3);
  Stensor sig0 = (lambda*trace(etop)*Stensor::Id()+2*mu*etop);
  real s1;
  real s2;
  real s3;
  sig0.computeEigenValues(s1,s2,s3);
  real gam=1. ;
  if (min(s1,min(s2,s3))<0) {
    pns1=min(0.,s1);
    pns2=min(0.,s2);
    pns3=min(0.,s3);
    gam=-sqrt(pns1*pns1+pns2*pns2+pns3*pns3)/(pns1+pns2+pns3);
  }
  ppe1=max(0.,e1);
  ppe2=max(0.,e2);
  ppe3=max(0.,e3);
  pps1=max(0.,s1);
  pps2=max(0.,s2);
  pps3=max(0.,s3);
  eeqc= gam*sqrt(ppe1*ppe1+ppe2*ppe2+ppe3*ppe3);
  eeqcor=max(eeqc,eeqcor);
  real r=(pps1+pps2+pps3)/(abs(s1)+abs(s2)+abs(s3));
  A=At*(2*r*r*(1.-2*k)-r*(1-4*k))+Ac*(2*r*r-3*r+1);
  B=r*r*Bt+(1-r*r)*Bc;
  real Y1=max(ed0,eeqcor);
  Y=max(Y1,Y);
  feel -= deto;
  real dnew=1-(1-A)*ed0/Y-A*exp(-B*(Y-ed0));
  dnew=min(dmax,dnew);
  fd = dd-dnew+d;
}
@UpdateAuxiliaryStateVars{
  Tmax=max(Tmax,T);
}
}
```

1.3 Conditions aux limites et chargements

Pour chaque modélisation, les chargements et conditions aux limites sont identiques à ceux des tests de référence.

2 Solution de référence

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

3 Modélisation A

3.1 Caractéristiques de la modélisation

Modélisation point matériel en grandes déformations avec intégration implicite, identique à SSNV225C.

3.2 Grandeurs testées et résultats

Comparaison avec SSNV225C (même solution de référence : SSNV225A)

Identification	Instants (h)	Référence	Tolérance
<i>EPYY</i>	2000	0,020968	1%
<i>EPYY</i>	4000	0,05093	4 %
<i>VII(endo)</i>	2000	0,0323	1 %
<i>VII(endo)</i>	4000	0,06808	0,5 %
<i>dEPYY/dt</i>	1520	6,6539E-006	1,5 %

4 Modélisation B

4.1 Caractéristiques de la modélisation

Modélisation point matériel avec intégration explicite, comparable à SSNV225A, mais en petites déformations.

4.2 Grandeurs testées et résultats

Comparaison avec SSNV225A (les résultats diffèrent par le type de déformation, et sont fournis à titre indicatif)

Identification	Instants (h)	Référence	Tolérance
<i>EPYY</i>	2000	0,020968	15%
<i>VII(endo)</i>	2000	0,0323	12%
<i>dEPYY/dt</i>	1520	6,6539E-006	21 %

5 Modélisation C

5.1 Caractéristiques de la modélisation

Modélisation point matériel avec intégration explicite, comparable à SSNP113A

5.2 Grandeurs testées et résultats

Comparaison avec SSNP113A

On compare à 3 pas de temps différents (à la fin de l'étape 1, pendant la phase de croissance de l'endommagement et à la fin du chargement) les déformations, les contraintes ainsi que la valeur de l'endommagement.

Identification	Référence	tolérance %
N°10 ε_{xx}	$9.375 \cdot 10^{-5}$	0,1
σ_{xx}	$3.00 \cdot 10^6$	0,1
ε_{yy}	$-1.875 \cdot 10^{-5}$	0,1
σ_{yy}	0.	10^{-8}
ε_{xy}	0.	10^{-8}
σ_{xy}	0.	10^{-8}
D	0.	10^{-8}

Identification	Référence	tolérance %
N°25 ε_{xx}	$1.64 \cdot 10^{-4}$	0,1
σ_{xx}	$2.04 \cdot 10^6$	0,1
ε_{yy}	$8.67 \cdot 10^{-5}$	0,1
σ_{yy}	$1.35 \cdot 10^6$	0,1
ε_{xy}	$7.03 \cdot 10^{-5}$	0,1
σ_{xy}	$6.34 \cdot 10^5$	0,1
D	0.66211	0,1

Identification	Référence	tolérance %
N°310 ε_{xx}	$1.50 \cdot 10^{-3}$	0,1
σ_{xx}	$3.69 \cdot 10^5$	0,1
ε_{yy}	$2.09 \cdot 10^{-3}$	0,1
σ_{yy}	$4.59 \cdot 10^5$	0,1
ε_{xy}	$1.41 \cdot 10^{-3}$	0,1
σ_{xy}	$2.16 \cdot 10^5$	0,1
D	0.99423	0,1

6 Modélisation D

6.1 Caractéristiques de la modélisation

Modélisation similaire à SSLA103C.

6.2 Grandeurs testées et résultats

Variables	Instant	Type de Référence	Référence	Tolérance
ε_{xx}	3600	ANALYTIQUE	$6.53 \cdot 10^{-4}$	0.50%
ε_{yy}	3600	ANALYTIQUE	$6.53 \cdot 10^{-4}$	0.50%
ε_{xx}^p	3600	ANALYTIQUE	0.	1.00E-006
ε_{yy}^p	3600	ANALYTIQUE	0.	1.00E-006
σ_{xx}	3600	ANALYTIQUE	0.	1.00E-006
σ_{yy}	3600	ANALYTIQUE	0.	1.00E-006
D	3600	ANALYTIQUE	0.	1.00E-006

7 Modélisation E

7.1 Caractéristiques de la modélisation

Modélisation similaire à SSNP161A. Le calcul en contraintes planes est réalisé par la méthode de DEBORST.

7.2 Grandeurs testées et résultats

Déformations et contraintes au nœud *NI* .

Identification	Incrément	Type de référence	Valeur de référence	Tolérance
Contrainte σ_{yy}	7	'ANALYTIQUE'	-10842857 Pa	5%
Déformation ε_{yy}	7	'ANALYTIQUE'	-3,00E-004	5%
Contrainte σ_{yy}	69	'ANALYTIQUE'	-42428571 Pa	5%

8 Synthèse des résultats

Les résultats sont satisfaisants et valident l'interface entre *Code_Aster* et MFRONT en 3D, axisymétrique et contraintes planes, en grandes ou petites déformations, pour des comportements avec endommagement.