smartplus Documentation

Release 0.9.0

Yves Chemisky

Feb 01, 2018

Contents

1 About **3**

2 Smartplus Libraries **5**
- 2.1 Continuum Mechanics . . . 5
  - 2.1.1 The Continuum Mechanics Library . . . 5
  - 2.1.2 The Constitutive Library . . . 8
  - 2.1.3 The Damage Library . . . 12
  - 2.1.4 The Recovery Props Library . . . 13
- 2.2 Geometry . . . 14
- 2.3 Homogenization . . . 14
  - 2.3.1 The Eshelby Library . . . 14
- 2.4 Identification . . . 14
- 2.5 Material . . . 14
- 2.6 Maths . . . 14
  - 2.6.1 The Rotation Library . . . 14
  - 2.6.2 The Statistics Library . . . 14
- 2.7 Phase . . . 14
- 2.8 Solver . . . 14
  - 2.8.1 Solver . . . 14
  - 2.8.2 Step . . . 15

3 Micromechanics **17**

4 Umat **19**

5 Examples **21**
- 5.1 Run a simulation . . . 21
- 5.2 Set up a micro mechanical model . . . 23

Contents:

CHAPTER 1

About

- SMART+ is a scientific library built to facilitate the analysis of mechanics of materials.
- It is built on the top of Armadillo, a high quality C++ linear algebra library.
- It integrates several algorithms for the analysis of heterogeneous materials
- Enjoy!

SMART+ is a C++ library with emphasis on speed and ease-of-use. Its principle focus is to provide tools to facilitate the implementation of up-to-date constitutive model for materials in Finite Element Analysis Packages. This is done by providing a C++ API to generate user material subroutine based on a library of functions. Also, SMART+ provides tools to analyse the behavior of material, considering loading at the material point level.

SMART+ is mainly developed by contributors from the staff and students of Arts et Métiers ParisTech, that are members of the LEM3 laboratory. It is released under the GNU General Public License: GPL, version 3. Several institutions have contributed to the development of SMART+:

- **Arts et Métiers ParisTech** : Top French school of engineers, focused on technology and applications.
- **LEM3**: Laboratory devoted to the study of Microstructures and Mechanics of Materials.
- **CNRS**: National French Center for scientific research. The LEM3 laboratory is labelled from the CNRS.
- **Texas A&M University**: Top University in Texas, USA.

LEM3

LABORATOIRE D'ÉTUDE DES MICROSTRUCTURES
ET DE MÉCANIQUE
DES MATÉRIAUX

CHAPTER 2

Smartplus Libraries

2.1 Continuum Mechanics

2.1.1 The Continuum Mechanics Library

double **tr** (**const** vec &*v*)

Provides the trace of a second order tensor written as a vector v in the SMART+ formalism.

```
vec v = randu(6);
double trace = tr(v);
```

vec **dev** (**const** vec &*v*)

Provides the deviatoric part of a second order tensor written as a vector v in the SMART+ formalism.

```
vec v = randu(6);
vec deviatoric = dev(v);
```

double **Mises_stress** (**const** vec &*v*)

Provides the Von Mises stress σ^{Mises} of a second order stress tensor written as a vector v in the SMART+ formalism.

```
vec v = randu(6);
double Mises_sig = Mises_stress(v);
```

vec **eta_stress** (**const** vec &*v*)

Provides the stress flow $\eta_{stress} = \frac{3/2\sigma_{dev}}{\sigma_{Mises}}$ from a second order stress tensor written as a vector v in the SMART+ formalism (i.e. the shear terms are multiplied by 2, providing shear angles).

```
vec v = randu(6);
vec sigma_f = eta_stress(v);
```

double **Mises_strain** (**const** vec &*v*)

Provides the Von Mises strain ε^{Mises} of a second order stress tensor written as a vector v in the SMART+ formalism.

```
vec v = randu(6);
double Mises_eps = Mises_strain(v);
```

vec **eta_strain** (**const** vec &*v*)
Provides the strain flow $\eta_{strain} = \frac{2/3\varepsilon_{dev}}{\varepsilon_{Mises}}$ from a second order strain tensor written as a vector v in the SMART+ formalism (i.e. the shear terms are multiplied by 2, providing shear angles).

```
vec v = randu(6);
vec eps_f = eta_strain(v);
```

mat **v2t_strain** (**const** vec &*v*)
Converts a second order strain tensor written as a vector v in the SMART+ formalism into a second order strain tensor written as a matrix m.

```
vec v = randu(6);
mat m = v2t_strain(v);
```

vec **t2v_strain** (**const** mat &*strain*)
Converts a second order strain tensor written as a matrix m in the SMART+ formalism into a second order strain tensor written as a vector v.

```
mat m = randu(6,6);
vec v = t2v_strain(m);
```

mat **v2t_stress** (**const** vec &*v*)
Converts a second order stress tensor written as a vector v in the SMART+ formalism into a second order stress tensor written as a matrix m.

```
vec v = randu(6);
mat m = v2t_stress(v);
```

vec **t2v_stress** (**const** mat &*stress*)
Converts a second order stress tensor written as a matrix m in the SMART+ formalism into a second order stress tensor written as a vector v.

```
mat m = randu(6,6);
vec v = t2v_stress(m);
```

double **J2_stress** (**const** vec &*v*)
Provides the second invariant of a second order stress tensor written as a vector v in the SMART+ formalism.

```
vec v = randu(6);
double J2 = J2_stress(v);
```

double **J2_strain** (**const** vec &*v*)
Provides the second invariant of a second order strain tensor written as a vector v in the SMART+ formalism.

```
vec v = randu(6);
double J2 = J2_strain(v);
```

double **J3_stress** (**const** vec &*v*)
Provides the third invariant of a second order stress tensor written as a vector v in the SMART+ formalism.

```
vec v = randu(6);
double J3 = J3_stress(v);
```

double **J3_strain**(**const** vec *&v*)
Provides the third invariant of a second order strain tensor written as a vector v in the SMART+ formalism.

```
vec v = randu(6);
double J3 = J3_strain(v);
```

double **Macaulay_p**(**const** double *&d*)
This function returns the value if it's positive, zero if it's negative (Macaulay brackets <>+)

double **Macaulay_n**(**const** double *&d*)
This function returns the value if it's negative, zero if it's positive (Macaulay brackets <>-)

double **sign**(**const** double *&d*)
This function returns the value if it's negative, zero if it's positive (Macaulay brackets <>-)

vec **normal_ellipsoid**(**const** double *&u*, **const** double *&v*, **const** double *&a1*, **const** double *&a2*, **const** double *&a3*)
Provides the normalized vector to an ellipsoid with semi-principal axes of length a1, a2, a3. The direction of the normalized vector is set by angles u and v. These 2 angles correspond to the rotations in the plan defined by the center of the ellipsoid, a1 and a2 directions for u, a1 and a3 ones for v. u = 0 corresponds to a1 direction and v = 0 correspond to a3 one. So the normal vector is set at the parametrized position :

$$x = a_1 cos(u) sin(v) \tag{2.1}$$
$$y = a_2 sin(u) sin(v) \tag{2.2}$$
$$z = a_3 cos(v) \tag{2.3}$$

```
const double Pi = 3.14159265358979323846

double u = (double)rand()/(double)(RAND_MAX) % 2*Pi - 2*Pi;
double v = (double)rand()/(double)(RAND_MAX) % Pi - Pi;
double a1 = (double)rand();
double a2 = (double)rand();
double a3 = (double)rand();
vec v = normal_ellipsoid(u, v, a1, a2, a3);
```

vec **sigma_int**(**const** vec *&sigma_in*, **const** double *&a1*, **const** double *&a2*, **const** double *&a3*, **const** double *&u*, **const** double *&v*)
Provides the normal and tangent components of a stress vector σin in accordance with the normal direction n to an ellipsoid with axes a1, a2, a3. The normal vector is set at the parametrized position :

$$x = a_1 cos(u) sin(v) \tag{2.4}$$
$$y = a_2 sin(u) sin(v) \tag{2.5}$$
$$z = a_3 cos(v) \tag{2.6}$$

```
vec sigma_in = randu(6);
double u = (double)rand()/(double)(RAND_MAX) % Pi - Pi/2;
double v = (double)rand()/(double)(RAND_MAX) % 2*Pi - Pi;
double a1 = (double)rand();
double a2 = (double)rand();
double a3 = (double)rand();
vec sigma_i = sigma_int(sigma_in, a1, a2, a3, u, v));
```

mat **p_ikjl**(**const** vec *&a*)
Provides the Hill interfacial operator according to a normal a (see papers of Siredey and Entemeyer Ph.D. dissertation).

```
vec v = randu(6);
mat H = p_ikjl(v);
```

2.1.2 The Constitutive Library

mat **Ireal** ()

Provides the fourth order identity tensor written in Voigt notation I_{real}, where :

$$I_{real} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{pmatrix}$$

```
mat Ir = Ireal();
```

mat **Ivol** ()

Provides the volumic of the identity tensor I_{vol} written in the SMART+ formalism. So :

$$I_{vol} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

```
mat Iv = Ivol();
```

mat **Idev** ()

Provides the deviatoric of the identity tensor I_{dev} written in the SMART+ formalism. So :

$$I_{dev} = I_{real} - I_{vol} = \begin{pmatrix} 2/3 & -1/3 & -1/3 \\ 0 & 0 & 0 \\ -1/3 & 2/3 & -1/3 \\ 0 & 0 & 0 \\ -1/3 & -1/3 & 2/3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.5 \end{pmatrix}$$

mat **Ireal2**()

Provides the fourth order identity tensor $\widehat{I}$ written in the form. So :

$$\widehat{I} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

For example, this tensor allows to obtain : $L * \widehat{M} = I$ or $\widehat{L} * M = I$, where a matrix $\widehat{A}$ is set by $\widehat{A} = \widehat{I}A\widehat{I}$

```
mat Ir2 = Ireal2();
```

mat **Idev2**()

Provides the deviatoric of the identity tensor $\widehat{I}$ written in the SMART+ formalism. So :

$$I_{dev2} = \begin{pmatrix} 2/3 & -1/3 & -1/3 \\ 0 & 0 & 0 \\ -1/3 & 2/3 & -1/3 \\ 0 & 0 & 0 \\ -1/3 & -1/3 & 2/3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

```
mat Id2 = Idev2();
```

vec **Ith**()

Provide the vector $I_{th} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

```
vec It = Ith();
```

vec **Ir2**()

Provide the vector $I_{r2} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \end{pmatrix}$

```
vec I2 = Ir2();
```

vec **Ir05**()

Provide the vector $I_{r05} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix}$

```
vec I05 = Ir05();
```

mat **L_iso**(**const** double *&C1*, **const** double *&C2*, **const** std::string *&conv*)

Provides the elastic stiffness tensor for an isotropic material. The two first arguments are a couple of elastic properties. The third argument specifies which couple has been provided and the nature and order of coefficients. Exhaustive list of possible third argument : 'Enu','nuE,'Kmu','muK', 'KG', 'GK', 'lambdamu', 'mulambda', 'lambdaG', 'Glambda'.

```
double E = 210000;
double nu = 0.3;
mat Liso = L_iso(E, nu, "Enu");
```

mat **M_iso**(**const** double *&C1*, **const** double *&C2*, **const** string *&conv*)

Provides the elastic compliance tensor for an isotropic material. The two first arguments are a couple of elastic properties. The third argument specify which couple has been provided and the nature and order of coefficients. Exhaustive list of possible third argument : 'Enu','nuE,'Kmu','muK', 'KG', 'GK', 'lambdamu', 'mulambda', 'lambdaG', 'Glambda'.

```
double E = 210000;
double nu = 0.3;
mat Miso = M_iso(E, nu, "Enu");
```

mat **L_cubic**(**const** double *&C1*, **const** double *&C2*, **const** double *&C4*, **const** string *&conv*)

Provides the elastic stiffness tensor for a cubic material. Arguments are the stiffness coefficients C11, C12 and C44.

```
double C11 = (double)rand();
double C12 = (double)rand();
doubel C44 = (double)rand();
mat Liso = L_cubic(C11,C12,C44);
```

mat **M_cubic**(**const** double *&C1*, **const** double *&C2*, **const** double *&C4*, **const** string *&conv*)

Provides the elastic compliance tensor for a cubic material. Arguments are the stiffness coefficients C11, C12 and C44.

```
double C11 = (double)rand();
double C12 = (double)rand();
double C44 = (double)rand();
mat Miso = M_cubic(C11,C12,C44);
```

mat **L_ortho**(**const** double *&C11*, **const** double *&C12*, **const** double *&C13*, **const** double *&C22*, **const** double *&C23*, **const** double *&C33*, **const** double *&C44*, **const** double *&C55*, **const** double *&C66*, **const** string *&conv*)

Provides the elastic stiffness tensor for an orthotropic material. Arguments could be all the stiffness coefficients or the material parameter. For an orthotropic material the material parameters should be : Ex,Ey,Ez,nuxy,nuyz,nxz,Gxy,Gyz,Gxz.

The last argument must be set to "Cii" if the inputs are the stiffness coefficients or to "EnuG" if the inputs are the material parameters.

```
double C11 = (double)rand();
double C12 = (double)rand();
double C13 = (double)rand();
double C22 = (double)rand();
double C23 = (double)rand();
double C33 = (double)rand();
double C44 = (double)rand();
double C55 = (double)rand();
double C66 = (double)rand();
mat Lortho = L_ortho(C11, C12, C13, C22, C23, C33, C44, C55, C66,"Cii");
```

mat **M_ortho**(**const** double *&C11*, **const** double *&C12*, **const** double *&C13*, **const** double *&C22*, **const** double *&C23*, **const** double *&C33*, **const** double *&C44*, **const** double *&C55*, **const** double *&C66*, **const** string *&conv*)

Provides the elastic compliance tensor for an orthotropic material. Arguments could be all the stiffness coefficients or the material parameter. For an orthotropic material the material parameters should be : Ex,Ey,Ez,nuxy,nuyz,nxz,Gxy,Gyz,Gxz.

The last argument must be set to "Cii" if the inputs are the stiffness coefficients or to "EnuG" if the inputs are the material parameters.

```
double C11 = (double)rand();
double C12 = (double)rand();
double C13 = (double)rand();
double C22 = (double)rand();
double C23 = (double)rand();
double C33 = (double)rand();
double C44 = (double)rand();
double C55 = (double)rand();
double C66 = (double)rand();
mat Mortho = M_ortho(C11, C12, C13, C22, C23, C33, C44, C55, C66,"Cii");
```

mat **L_isotrans**(**const** double *&EL*, **const** double *&ET*, **const** double *&nuTL*, **const** double *&nuTT*, **const** double *&GLT*, **const** int *&axis*)

Provides the elastic stiffness tensor for an isotropic transverse material. Arguments are longitudinal Young modulus EL, transverse young modulus, Poisson's ratio for loading along the longitudinal axis nuTL, Poisson's ratio for loading along the transverse axis nuTT, shear modulus GLT and the axis of symmetry.

```
double EL = (double)rand();
double ET = (double)rand();
double nuTL = (double)rand();
double nuTT = (double)rand();
double GLT = (double)rand();
double axis = 1;
mat Lisotrans = L_isotrans(EL, ET, nuTL, nuTT, GLT, axis);
```

mat **M_isotrans**(**const** double *&EL*, **const** double *&ET*, **const** double *&nuTL*, **const** double *&nuTT*, **const** double *&GLT*, **const** int *&axis*)

Provides the elastic compliance tensor for an isotropic transverse material. Arguments are longitudinal Young modulus EL, transverse young modulus, Poisson's ratio for loading along the longitudinal axis nuTL, Poisson's ratio for loading along the transverse axis nuTT, shear modulus GLT and the axis of symmetry.

```
double EL = (double)rand();
double ET = (double)rand();
double nuTL = (double)rand();
double nuTT = (double)rand();
double GLT = (double)rand();
double axis = 1;
mat Misotrans = M_isotrans(EL, ET, nuTL, nuTT, GLT, axis);
```

mat **H_iso**(**const** double *&etaB*, **const** double *&etaS*)

Provides the viscoelastic tensor H, providing Bulk viscosity etaB and shear viscosity etaS. It actually returns :

$$H_iso = \begin{pmatrix} \eta_B & \eta_B & \eta_B \\ 0 & 0 & 0 \\ \eta_B & \eta_B & \eta_B \\ 0 & 0 & 0 \\ \eta_B & \eta_B & \eta_B \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

```
double etaB = (double)rand();
double etaS = (double)rand();
mat Hiso = H_iso(etaB, etaS);
```

void el_pred

Provides the stress tensor from an elastic prediction There are two possible ways:

1. From the elastic stiffness tensor and the trial elastic strain: parameters : L : Stiffness matrix; Eel ; elastic strain vector, ndi (optional, default = 3): number of dimensions

```
mat L = L_iso(70000, 0.3,"Enu");
vec Eel;
Eel.randu(6);
int ndi = 3;
vec sigma =  el_pred(L, Eel, ndi);
```

2. From the previous stress increment, providing the elastic stiffness tensor and the trial elastic strain increment: parameters : sigma_start: The previous stress, L : Stiffness matrix; Eel : elastic strain vector, ndi (optional, default = 3): number of dimensions

```
vec sigma_start = zeros(6);
sigma_start.randu(6);
mat L = L_iso(70000, 0.3,"Enu");
vec Eel;
Eel.randu(6);
int ndi = 3;
vec sigma =  el_pred(sigma_start,L, Eel, ndi);
```

2.1.3 The Damage Library

double **damage_weibull**(**const** vec *&stress*, **const** double *&damage*, **const** double *&alpha*, **const** double *&beta*, **const** double *&DTime*, **const** string *&criterion*)

Provides the damage evolution δD considering a Weibull damage law. It is given by : $\delta D = (1 - D_{old}) *$

$\Big(1 - exp\big(-1(\frac{crit}{\beta})^{\alpha}\big)\Big)$ Parameters of this function are: the stress vector σ, the old damage D_{old}, the shape parameter α, the scale parameter β, the time increment ΔT and the criterion (which is a string).

The criterion possibilities are : "vonmises" : $crit = \sigma_{Mises}$ "hydro" : $crit = tr(\sigma)$ "J3" : $crit = J3(\sigma)$ Default value of the criterion is "vonmises".

```
double varD = damage_weibull(stress, damage, alpha, beta, DTime, criterion);
```

double **damage_kachanov** (**const** vec *&stress*, **const** vec *&strain*, **const** double *&damage*, **const** double *&A0*, **const** double *&r*, **const** string *&criterion*)

Provides the damage evolution δD considering a Kachanov's creep damage law. It is given by : $\delta D = \Big(\frac{crit}{A_0(1-D_{old})}\Big)^r$ Parameters of this function are: the stress vector σ, the strain vector ϵ, the old damage D_{old}, the material properties characteristic of creep damage (A_0, r) and the criterion (which is a string).

The criterion possibilities are : "vonmises" : $crit = (\sigma * (1+\varepsilon))_{Mises}$ "hydro" : $crit = tr(\sigma * (1+\varepsilon))$ "J3" : $crit = J3(\sigma * (1+\varepsilon))$ Here, the criterion has no default value.

```
double varD = damage_kachanov(stress, strain, damage, A0, r, criterion);
```

double **damage_miner** (**const** double *&S_max*, **const** double *&S_mean*, **const** double *&S_ult*, **const** double *&b*, **const** double *&B0*, **const** double *&beta*, **const** double *&Sl_0*)

Provides the constant damage evolution ΔD considering a Woehler- Miner's damage law. It is given by : $\Delta D = \big(\frac{S_{Max}-S_{Mean}+Sl_0*(1-b*S_{Mean})}{S_{ult}-S_{Max}}\big) * \big(\frac{S_{Max}-S_{Mean}}{B_0*(1-b*S_{Mean})}\big)^{\beta}$ Parameters of this function are: the max stress value σ_{Max}, the mean stress value σ_{Mean}, the "ult" stress value σ_{ult}, the b, the B_0, the β and the Sl_0.

Default value of Sl_0 is 0.0.

```
double varD = damage_minerl(S_max, S_mean, S_ult, b, B0, beta, Sl_0);
```

double **damage_manson** (**const** double *&S_amp*, **const** double *&C2*, **const** double *&gamma2*)

Provides the constant damage evolution ΔD considering a Coffin-Manson's damage law. It is given by : $\Delta D = \big(\frac{\sigma_{Amp}}{C_2}\big)^{\gamma_2}$ Parameters of this function are: the "amp" stress value σ_{Amp}, the C_2 and the γ_2.

```
double varD = damage_manson(S_amp, C2, gamma2);
```

2.1.4 The Recovery Props Library

The recovery props library provides a set of function to check and evaluate the properties of stiffness and compliance tensors.

void **check_symetries** (mat *L*, string *umat_type*, int *axis*, vec *props*, int *maj_sym*)

Check the symmetries of a stiffness matrix L, and fill the vector of material properties. Depending on the symmetry found, the string umat_type, the axis of symmetry (if applicable) the vector of material properties, and the major symmetry maj_sym (L_ij = L_ji ?). If the major symmetry condition is not fulfilled, the check of symmetries if performed on the symmetric part of L

Table 2.1: Material Symmetries considered

Symmetry	umat_type	axis	size of props
Fully anisotropic	ELANI	0	0
Monoclinic	ELMON	1,2 or 3	0
Orthotropic	ELORT	0	9
Cubic	ELCUB	0	3
Transversely isotropic	ELITR	1,2 or 3	5
Isotropic	ELISO	0	2

```
check_symetries(L, umat_name, axis, props, maj_sym);
```

2.2 Geometry

2.3 Homogenization

2.3.1 The Eshelby Library

2.4 Identification

2.5 Material

2.6 Maths

2.6.1 The Rotation Library

2.6.2 The Statistics Library

2.7 Phase

2.8 Solver

2.8.1 Solver

void **solver**(**const** string *&umat_name*, **const** vec *&props*, **const** double *&nstatev*, **const** double *&psi_rve*, **const** double *&theta_rve*, **const** double *&phi_rve*, **const** double *&rho*, **const** double *&c_p*, **const** std::string *&path_data*, **const** std::string *&path_results*, **const** std::string *&pathfile*, **const** std::string *&outputfile*)

Solves...

Parameters

- **const string &umat_name** –
- **const vec &props** –
- **const double &nstatev** –
- **const double &psi_rve** –
- **const double &theta_rve** –
- **const double &phi_rve** –
- **const double &rho** –
- **const double &c_p** –
- **const string &path_data** –
- **const string &path_results** –

- **const string &pathfile** –
- **const string &outputfile** –

2.8.2 Step

class step

class step_meca : **public** *step*

class step_thermomeca : **public** *step*

CHAPTER 3

Micromechanics

CHAPTER 4

Umat

CHAPTER 5

Examples

5.1 Run a simulation

Here is a simple example to run your first simulation using SMART+. Open the file Path.txt in the folder Control You should find something that look like that

```
#Initial_temperature
293.5
#Number_of_blocks
1

#Block
1
#Loading_type
1
#Repeat
1
#Steps
1

#Mode
1
#Dn_init 1.
#Dn_mini 0.1
#Dn_inc 0.01
#time
30.
#mechanical_state
E 0.3
S 0 S 0
S 0 S 0 S 0
#temperature_state
T 293.5
```

The first part of the file describe the initial temperature conditions, under the tag #Initial_temperature.

Just below, under the tag #Number_of_blocks, you define the number of blocks. Here we will start with a single block, so this value is set to 1. The next part is to define the first block: #Block defines the block number #Loading_type defines the physical problem to solve, which is: 1 – mechanical; 2 – thermomechanical #Repeat is the number of time the block is repeated #Steps is the number of steps of the block

The next part of the file defines the steps of the first block. It always starts with the mode of the step (#mode), which is: 1 – linear; 2 – sinusoidal; 3 – tabular (from a file)

In this example we will consider that the step mode is linear. We therefore need to set up the following

1. The mode of the step (under #mode)
2. The initial size of the first increment (usually 1.), under #Dn_init
3. The minimal size of an increment (usually less than 1.), under #Dn_mini
4. The size of the increment as a fraction of the step $delta n$, under #Dn_inc. (#Dn_inc 0.01 means that 100 increments will be utilized to simulate the step)
5. The time $Delta t$ of the step (under #time). Note that the increment of time for any increment is $delta t = Delta t delta n$

#. The mechanical loading stage at the end of the step (#mechanical_state) The elements are organized such that either stress or strain components are defined in the following order: 11 12 22 13 23 33 The letter 'S' in front of any component means that a stress control is considered in that direction, and the letter 'E' stands for a strain control. Note that those values indicate the state at the end of the step #. The thermal loading stage at the end of the step (#temperature_state) For mechanical loading, the letter 'T' is followed by the temperature at the end of the step. For thermomechanical loading, either the final temperature can be considered (with the letter 'T'), or the thermal flux (with the letter 'Q'). This last quantity is defined as the rate of heat that flows to the material representative volume element considered.

If the #mode is set to 2 - sinusoidal, a sinusoidal evolution is considered automatically between the state of the previous step and the final values indicated in the current step.

If the #mode is set to 3 - tabular, a prescribed evolution is considered, and a file that contains such prescribed evolution must be indicated. In that case, the step block has to be defined like:

```
#Mode
3
#File
tabular_file.txt
#Dn_init 1.
#Dn_mini 0.01
#Consigne
S
0  S
0  0  0
#T_is_set
0
```

In the following example, a biaxial test in the directions 11 and 22 is considered, with a stress control. The temperature is not set, which means that it is constant throughout the step and keep its value from the previous step (or the intial temperature if this is the first step). Note that the time is always indicated in the tabular_file.txt. The struture of the tabular file will be the following:

```
0       0.0     10      10
1       0.01    20      20
2       0.02    30      30
3       0.03    30      30
...
```

The columns define the quantities in the followin order : #ninc, #time, #S11, #S22. The order of the mechanical quantities is always 11,12,22,13,23,33, and if the temperature is set (with the letter 'T' instead of '0'), the following order is always considered: #ninc, #time, #T, #S11, #S22 in the case of the biaxial loading.

```
0       0.0     293.15  10      10
1       0.01    294.15  20      20
2       0.02    295.15  30      30
3       0.03    296.15  30      30
...
```

5.2 Set up a micro mechanical model

The first thing you want to do when setting up a micro mechanical model is to define the microstructure. At a certain scale, you should inform the model about the phases, their volume fraction, geometry and their properties.

First, in the file data/material.dat, you need to enter the material properties corresponding to the micro mechanical model you selected:

For Mori-Tanaka and Self-Consistent: 4 material parameters (and a consequent number of state_variables)

1. props(0) : Number of phases
2. props(1) : File number that stores the microstructure properties
3. props(2) : Number of integration points in the 1 direction
4. props(3) : Number of integration points in the 2 direction

For Periodic layers: 2 material parameters (and a consequent number of state_variables)

1. props(0) : Number of phases
2. props(1) : File number that stores the microstructure properties

The file data/material.dat should look like this for a 2-phase material using a Mor-Tanaka model:

```
Material
Name    MIMTN
Number_of_material_parameters   4
Number_of_internal_variables    10000

#Thermal
density 1.12
c_p   1.64

#Mechancial
nphases 2
file_number 0
nItg1 20
nItg2 20
```

The density and specific heat capacity c_p are utilized only if you want to solve a thermomechanical boundary-value problem.

The file number represents the number of the Nphases[i].dat file, where [i] is replaced by the number value. In this case we should fill the file Nphases0.dat, which looks like this:

```
Number  Coatingof  umat   c    phi_mat  theta_mat  psi_mat  a1  a2  a3  phi_geom ␣
↪theta_geom  psi_geom  nprops  nstatev  props
0       0          ELISO  0.8  0        0          0        1   1   1   0.        0. ␣
↪         0.         3       1        3000     0.4  1.E-5
1       0          ELISO  0.2  0        0          0        1   1   1   0.        0. ␣
↪         0.         3       1        70000    0.4  1.E-5
```

Note that for Mori-Tanaka the first phase in the file should always be the matrix. The characteristics of the phases are described below:

1. Number : The number of the phase
2. Coatingof : If the model is a coating of an other phase. 0 if the phase is not a coating
3. umat : Constitutive model considered
4. c : Volume fraction of the phase
5. phi_mat: First Euler angle corresponding to the material orientation
6. theta_mat: Second Euler angle corresponding to the material orientation
7. psi_mat: Third Euler angle corresponding to the material orientation
8. a1:
9. a2:
10. a3:
11. phi_geom: First Euler angle corresponding to the ellipsoid orientation
12. theta_geom: Second Euler angle corresponding to the ellipsoid orientation
13. psi_geom: Third Euler angle corresponding to the ellipsoid orientation
14. npros: Number of material properties
15. nstatev: Number of scalar internal variables
16. props: The list of material properties

For a wide majority of composites, the orientation of the material coincides with the orientation of the reinforcement (For instance transversely isotropic carbon fibers). However, for metallic polycristals, the two materials systems have to be considered to separate the orientation of the lattice with the orientation of the ellipsoid that represent a grain. This version of SMART+ currently does not support coated inclusions, but the files Nphase[i].dat is prepared so that you can easily add this to a custom micromechancial model.

Note that the Euler system reference utilised (3-1-3 for the most common) is defined in the parameter.hpp file. For instance this system is defined by default in the parameter.hpp:

```
#ifndef axis_psi
#define axis_psi 3
#endif

#ifndef axis_theta
#define axis_theta 1
#endif

#ifndef axis_phi
#define axis_phi 3
#endif
```

In the example here we are defining a 2-phase composite, with spherical reinforcements, considering two phases:

1. An epoxy matrix, 80% volume, with E=3000MPa and nu=0.4, and alpha=1.E-5
2. Aluminium reinforcements: 20% volume, with E=70000MPa and nu=0.3, and alpha=5.E-5

Once these files have been set up, you can run a simulation using the classical solver.

- genindex
- search

Index

C

check_symetries (C++ function), 13

D

damage_kachanov (C++ function), 13
damage_manson (C++ function), 13
damage_miner (C++ function), 13
damage_weibull (C++ function), 12
dev (C++ function), 5

E

eta_strain (C++ function), 6
eta_stress (C++ function), 5

H

H_iso (C++ function), 12

I

Idev (C++ function), 8
Idev2 (C++ function), 9
Ir05 (C++ function), 10
Ir2 (C++ function), 9
Ireal (C++ function), 8
Ireal2 (C++ function), 8
Ith (C++ function), 9
Ivol (C++ function), 8

J

J2_strain (C++ function), 6
J2_stress (C++ function), 6
J3_strain (C++ function), 6
J3_stress (C++ function), 6

L

L_cubic (C++ function), 10
L_iso (C++ function), 10
L_isotrans (C++ function), 11
L_ortho (C++ function), 10

M

M_cubic (C++ function), 10
M_iso (C++ function), 10
M_isotrans (C++ function), 11
M_ortho (C++ function), 11
Macaulay_n (C++ function), 7
Macaulay_p (C++ function), 7
Mises_strain (C++ function), 5
Mises_stress (C++ function), 5

N

normal_ellipsoid (C++ function), 7

P

p_ikjl (C++ function), 7

S

sigma_int (C++ function), 7
sign (C++ function), 7
solver (C++ function), 14
step (C++ class), 15
step_meca (C++ class), 15
step_thermomeca (C++ class), 15

T

t2v_strain (C++ function), 6
t2v_stress (C++ function), 6
tr (C++ function), 5

V

v2t_strain (C++ function), 6
v2t_stress (C++ function), 6