The new `TFEL/Material/Homogenization` library, application to 'mean-field' homogenization schemes on particulate composites

MFront User Day 2025

Antoine MARTIN[1], Thomas HELFER[1]

[1]CEA, DES, IRESNE, DEC, Cadarache F-13018 Saint-Paul-lez-Durance, France

20 November 2025

Introduction

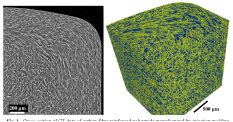
Introduction

- National research project - **A**dvanced **NO**n-linear **HO**mogenization for Structural A**NA**lysis

Introduction

- National research project - **A**dvanced **NO**n-linear **HO**mogenization for Structural A**NA**lysis

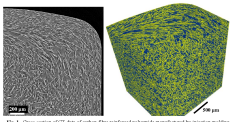
- Mean-field homogenization schemes for computation at the structure scale



Introduction

- National research project - **A**dvanced **NO**n-linear **HO**mogenization for Structural A**NA**lysis

- Mean-field homogenization schemes for computation at the structure scale

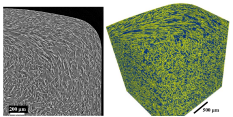


- Missing an open-source software for using these schemes in a large community

Introduction

- National research project - **A**dvanced **NO**n-linear **HO**mogenization for Structural A**NA**lysis

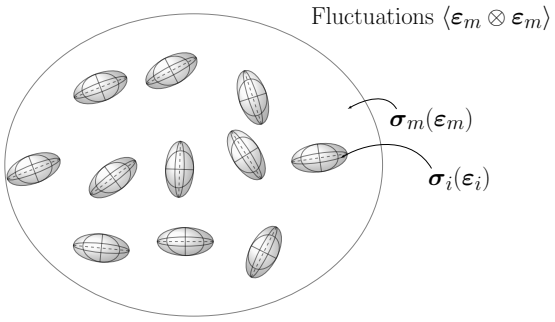
- Mean-field homogenization schemes for computation at the structure scale



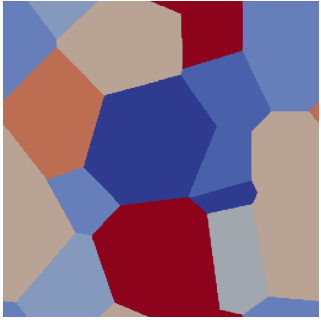
- Missing an open-source software for using these schemes in a large community
- Implementation of theses schemes via `MFront`

What does ‘mean-field homogenization’ mean ?

Particulate microstructures



Polycrystals



The homogenization problem

Spheres in isotropic matrix		Complex microstructure
Linear $\underline{\sigma} = \underline{\underline{C}} : \underline{\varepsilon}$	$\Rightarrow$	Non-linear $\underline{\sigma}(\underline{\varepsilon})$
RVE		Structure scale $\times n_G$

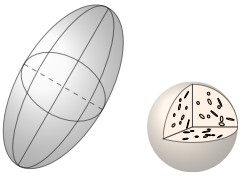
1. **The TFEL/Material/Homogenization library**
   The `tfel.material.homogenization` module
   Some computation at the structure scale

2. **Non-linear homogenization**
   Linear visco-elasticity
   The `@BehaviourVariable` keyword
   Automatic differentiation and variational schemes

The TFEL/Material/Homogenization library

TFEL/Material

tfel::material::homogenization::elasticity

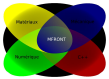


- Hill tensors, Strain localisation tensors
- Classical homogenization schemes (Dilute, Mori-Tanaka, Ponte-Castañeda & Willis, Self-consistent...)
- Ellipsoidal inclusions
- Isotropic distributions, transverse isotropic, aligned inclusions

- github.com/thelfer/tfel/tree/master/include/TFEL/Material (source code)
- thelfer.github.io/tfel/web/tfel-material.html#homogenization (documentation)

The TFEL/Material/Homogenization library

MFront



```
using namespace
tfel::material::homogenization;
```

`mfront --obuild` ↓ `material.mfront`

Structural analysis

Cast3M

SIMULIA ABAQUS

Python

```
import
tfel.material.homogenization
```

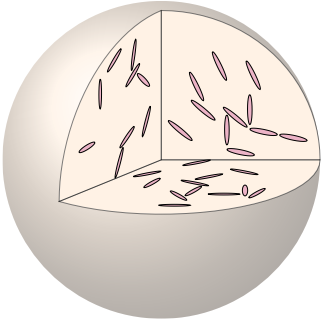
↓

Structural analysis

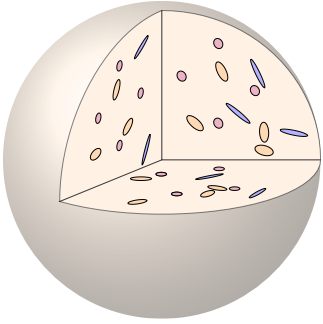
sfepy

The `tfel.material.homogenization` module : 2 types of implementation

Biphasic homogenization



`ParticulateMicrostructure` objects



The `tfel.material.homogenization` module, biphasic

```python
import tfel.material as tmat
import tfel.material.homogenization as hm
import tfel.math as tm
import numpy as np

print("IsotropicModuli")
kg=tmat.KGModuli(3e6,2e6)
print("K: ",kg.kappa,"G: ",kg.mu)

Enu=tmat.YoungNuModuli(1e6,0.2)
print("E: ",Enu.young,"ν: ", Enu.nu)

lg=tmat.LambdaMuModuli(0.5e6,1e6)
print("λ: ",lg.lamb,"μ: ",lg.mu)

Enu2=kg.ToYoungNu()
print("E: ",Enu2.young,"ν: ",Enu2.nu)
```

[2]

```
IsotropicModuli
K:  3000000.0 G:  2000000.0
E:  1000000.0 ν:  0.2
λ:  500000.0 μ:  1000000.0
E:  4909090.909090909 ν:  0.22727272727272727
```

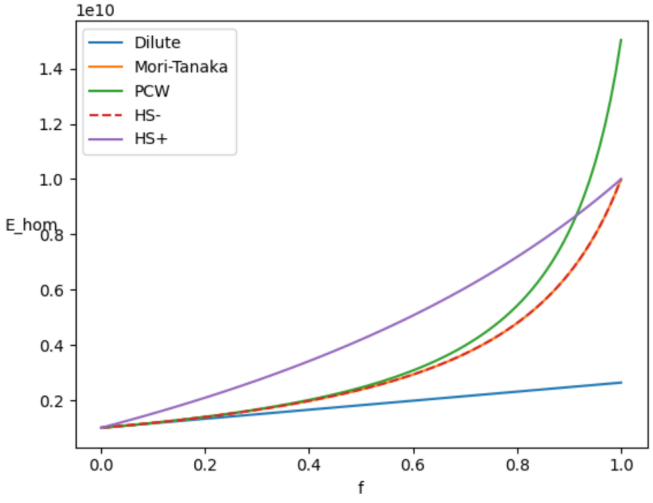
The `tfel.material.homogenization` module, biphasic

```python
# Spherical inclusions
for f in f_i:
    KG_DS=hm.computeSphereDiluteScheme(IM,f,IMi)
    EnuDS=KG_DS.ToYoungNu()
    KG_MT=hm.computeSphereMoriTanakaScheme(IM,f,IMi)
    EnuMT=KG_MT.ToYoungNu()
    E_i_DS.append(EnuDS.young)
    E_i_MT.append(EnuMT.young)

#PCW scheme
D=hm.Distribution(tm.TVector3D([1.,0.,0.]),0.9,tm.TVector3D([0.,1.,0.]),1,1)
E_i_PCW=[]
for f in f_i:
    C_PCW=hm.computeIsotropicPCWScheme(IM,f,IMi,1,1,1,D)
    G=(C_PCW.__getitem__(0,0)-C_PCW.__getitem__(0,1))/2
    K=C_PCW.__getitem__(0,0)-4*G/3
    kg=tmat.KGModuli(K,G)
    Enu=kg.ToYoungNu()
    E_i_PCW.append(Enu.young)
```

The `tfel.material.homogenization` module, biphasic

...



The `tfel.material.homogenization` module, biphasic

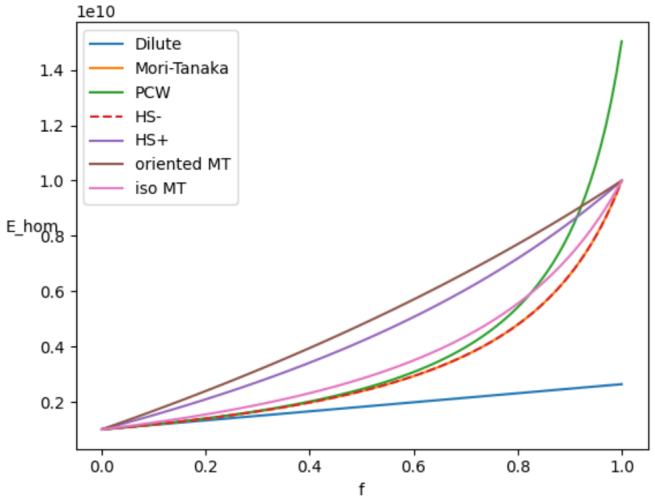
```python
# Ellipsoidal inclusions
a=10.
b=1.
c=1.

## Isotropic distribution of orientations
E_iso=[]
for f in f_i:
    KG_I_MT=hm.computeIsotropicMoriTanakaScheme(IM,f,IMi,a,b,c)
    Enu_iso=KG_I_MT.ToYoungNu()
    E_iso.append(Enu_iso.young)

## Oriented ellipsoids
n_a=tm.TVector3D([1.,0.,0.])
n_b=tm.TVector3D([0.,1.,0.])
E_O=[]
for f in f_i:
    C_O=hm.computeOrientedMoriTanakaScheme(IM,f,IMi,n_a,a,n_b,b,c)
    S_O=np.linalg.inv(C_O)
    E_O.append(1/S_O[0,0])
```

The `tfel.material.homogenization` module, biphasic

...



The `tfel.material.homogenization` module, biphasic

```python
young=1e9
nu=0.2
P0=hm.computeSphereHillTensor(young,nu)

IM0=tmat.YoungNuModuli(young,nu)
P0 = hm.computeSphereHillTensor(IM0)

e=10.
n_a=tm.TVector3D([0.,0.,1.])
P1=hm.computeAxisymmetricalHillTensor(young,nu,n_a,e)
P0_axi = hm.computeAxisymmetricalHillTensor(IM0,n_a,e)

n_b=tm.TVector3D([0.,1.,0.])
a=10.
b=1.
c=3.
P2=hm.computeHillTensor(young,nu,n_a,a,n_b,b,c)
P0_ellipsoid = hm.computeHillTensor(IM0,n_a,a,n_b,b,c)
```

Python

The `tfel.material.homogenization` module, biphasic

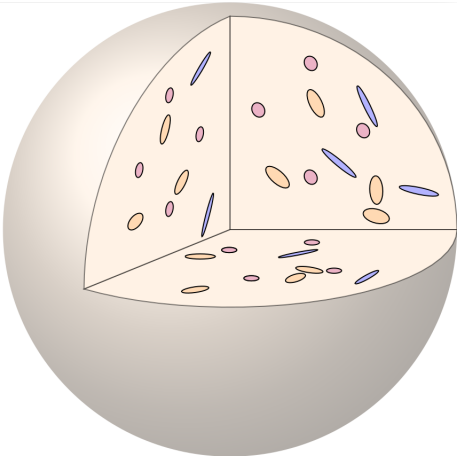
```
# Spherical inclusion
A_S=hm.computeSphereLocalisationTensor(young,nu,young_i,nu_i)

# Axisymmetric ellipsoidal inclusion (or spheroid)
e=20.
n=tm.TVector3D([0.,0.,1.])
A_AE=hm.computeAxisymmetricalLocalisationTensor(young,nu,young_i,nu_i,n,e)

# General ellipsoidal inclusion
a=10.
b=1.
c=3.
n_a=tm.TVector3D([0.,0.,1.])
n_b=tm.TVector3D([0.,1.,0.])

A_GE=hm.computeLocalisationTensor(young,nu,young_i,nu_i,n_a,a,n_b,b,c)
```

The `tfel.material.homogenization` module, `ParticulateMicrostructure`



The `tfel.material.homogenization` module, `ParticulateMicrostructure`

```
print("ParticulateMicrostructure")
IM0=tmat.KGModuli(1e7,1e7)
micro_1=hm.ParticulateMicrostructure(IM0)
print("micro_1 is isotropic ? ",micro_1.is_isotropic_matrix())

C0=tm.ST2toST23D(1e7*np.eye(6))
micro_2=hm.ParticulateMicrostructure(C0)
print("micro_2 is isotropic ? ",micro_2.is_isotropic_matrix())

sphere_1=hm.Sphere()
sph=hm.Sphere()
spheroid=hm.Spheroid(10,2)
ellipso=hm.Ellipsoid(10,2,3)
print("spheroid lengths: ",spheroid.axis_length(),spheroid.transverse_length())
print("ellipsoid lengths: ",ellipso.semi_lengths)
```

[3]

```
ParticulateMicrostructure
micro_1 is isotropic ?  True
micro_2 is isotropic ?  False

 Create Inclusion objects
spheroid lengths:  10.0 2.0
ellipsoid lengths:  [10.0, 2.0, 3.0]
```

The `tfel.material.homogenization` module, `ParticulateMicrostructure`

```
IMi=tmat.KGModuli(1e9,1e9)
sph_dist=hm.SphereDistribution(sph,0.2,IMi)
ellipsoid_dist_iso=hm.IsotropicDistribution(ellipso,0.1,IMi)

print("fraction: ",ellipsoid_dist_iso.fraction)
print("spheres are locally isotropic ? ",sph_dist.is_isotropic())
```

[4] Pytho

```
Create InclusionDistribution objects
fraction:  0.1
spheres are locally isotropic ?  True
```

```
print("Construct your microstructure")
micro_1.addInclusionPhase(sph_dist)
print("number of phases: ",micro_1.get_number_of_phases())
print("matrix fraction: ",micro_1.get_matrix_fraction())
```

[5] Pytho

```
Construct your microstructure
number of phases:  2
matrix fraction:  0.8
```

The `tfel.material.homogenization` module, `ParticulateMicrostructure`

```
[6]
micro_1.addInclusionPhase(ellipsoid_dist_iso)
print("number of phases: ",micro_1.get_number_of_phases())
print("matrix fraction: ",micro_1.get_matrix_fraction())
```

Pytho

```
number of phases:  3
matrix fraction:  0.7000000000000001
```

```
[7]
micro_1.removeInclusionPhase(0)
print("number of phases: ",micro_1.get_number_of_phases())
print("matrix fraction: ",micro_1.get_matrix_fraction())
```

Pytho

```
number of phases:  2
matrix fraction:  0.9000000000000001
```

The `tfel.material.homogenization` module, `ParticulateMicrostructure`

```
[8]
ell_dist=micro_1.get_inclusionPhase(0)
print("fraction: ",ell_dist.fraction)
print("stiffness: ",ell_dist.stiffness)
```
Python

```
fraction:  0.1
stiffness:  [[2.33333e+09,3.33333e+08,3.33333e+08,0,0,0]
 [3.33333e+08,2.33333e+09,3.33333e+08,0,0,0]
 [3.33333e+08,3.33333e+08,2.33333e+09,0,0,0]
 [0,0,0,2e+09,0,0]
 [0,0,0,0,2e+09,0]
 [0,0,0,0,0,2e+09]]
```

```
[9]
print("Compute localisators")
Ai=ell_dist.computeMeanLocalisator(IM0)
print("ellipsoid distribution: ",Ai)
```
Python

```
Compute localisators
ellipsoid distribution:  [[0.0285593,0.00286032,0.00286032,0,0,0]
 [0.00286032,0.0285593,0.00286032,0,0,0]
 [0.00286032,0.00286032,0.0285593,0,0,0]
```

The `tfel.material.homogenization` module, `ParticulateMicrostructure`

```
[9]
print("Compute localisators")
Ai=ell_dist.computeMeanLocalisator(IM0)
print("ellipsoid distribution: ",Ai)
```

Python

```
Compute localisators
ellipsoid distribution:  [[0.0285593,0.00286032,0.00286032,0,0,0]
 [0.00286032,0.0285593,0.00286032,0,0,0]
 [0.00286032,0.00286032,0.0285593,0,0,0]
 [0,0,0,0.025699,0,0]
 [0,0,0,0,0.025699,0]
 [0,0,0,0,0,0.025699]]
```

```
[10]
A=sph_dist.computeMeanLocalisator(C0,12)
print("sphere distribution: ",A)
```

Python

```
sphere distribution:  [[0.00953403,0.000199871,0.000199871,3.36099e-20,-7.65471e-22,-9.46358e-22]
 [0.000199871,0.00953403,0.000199871,-7.37432e-20,1.75951e-21,1.56774e-21]
 [0.000199871,0.000199871,0.00953404,-4.14778e-21,-1.29524e-21,-3.08083e-21]
 [3.77191e-20,-8.20047e-20,-1.24954e-20,0.00933416,-1.26026e-21,-9.99444e-22]
 [1.50441e-22,-7.72663e-22,-2.08978e-21,-4.44265e-23,0.00933417,-6.7904e-20]
 [2.95311e-21,8.3215e-21,-4.95442e-21,-8.15546e-22,-6.19521e-20,0.00933417]]
```

The `tfel.material.homogenization` module, `ParticulateMicrostructure`

```
print("micro_1 is isotropic ? ",micro_1.is_isotropic_matrix())
print("number of phases: ",micro_1.get_number_of_phases(),"\n")
print("matrix stiffness ",micro_1.get_matrix_elasticity(),"\n")

hmDS=hm.computeDiluteScheme(micro_1)
hmMT=hm.computeMoriTanakaScheme(micro_1)
print("DS: ",hmDS.homogenized_stiffness)
print("MT: ",hmMT.homogenized_stiffness)
```

[11] Python

```
Compute homogenization schemes
micro_1 is isotropic ?  True
number of phases:  2

matrix stiffness  [[2.33333e+07,3.33333e+06,3.33333e+06,0,0,0]
 [3.33333e+06,2.33333e+07,3.33333e+06,0,0,0]
 [3.33333e+06,3.33333e+06,2.33333e+07,0,0,0]
 [0,0,0,2e+07,0,0]
 [0,0,0,0,2e+07,0]
 [0,0,0,0,0,2e+07]]

DS:  [[3.01193e+07,5.03092e+06,5.03092e+06,0,0,0]
 [5.03092e+06,3.01193e+07,5.03092e+06,0,0,0]
```

The `tfel.material.homogenization` module, `ParticulateMicrostructure`

```
DS:  [[3.01193e+07,5.03092e+06,5.03092e+06,0,0,0]
 [5.03092e+06,3.01193e+07,5.03092e+06,0,0,0]
 [5.03092e+06,5.03092e+06,3.01193e+07,0,0,0]
 [0,0,0,2.50884e+07,0,0]
 [0,0,0,0,2.50884e+07,0]
 [0,0,0,0,0,2.50884e+07]]
MT:  [[3.08483e+07,5.21059e+06,5.21059e+06,0,0,0]
 [5.21059e+06,3.08483e+07,5.21059e+06,0,0,0]
 [5.21059e+06,5.21059e+06,3.08483e+07,0,0,0]
 [0,0,0,2.56377e+07,0,0]
 [0,0,0,0,2.56377e+07,0]
 [0,0,0,0,0,2.56377e+07]]
```

[12]

```
hmSC=hm.computeSelfConsistentScheme(micro_1,14,True)
print("SC: ",hmSC.homogenized_stiffness)
```

Python

```
SC:  [[3.38025e+07,6.023e+06,6.023e+06,0,0,0]
 [6.023e+06,3.38025e+07,6.023e+06,0,0,0]
 [6.023e+06,6.023e+06,3.38025e+07,0,0,0]
 [0,0,0,2.77795e+07,0,0]
 [0,0,0,0,2.77795e+07,0]
```

The `tfel.material.homogenization` module, `ParticulateMicrostructure`

```
hmSC_iso=hm.computeSelfConsistentScheme(micro_2,14,True)
hmSC_aniso=hm.computeSelfConsistentScheme(micro_2,12,False,10)
print("SC iso: ",hmSC_iso.homogenized_stiffness,"\n")
print("SC aniso: ",hmSC_aniso.homogenized_stiffness)
```

[13] Python

```
micro_2 is isotropic ?  False
number of phases:  2

matrix stiffness  [[1e+07,0,0,0,0,0]
 [0,1e+07,0,0,0,0]
 [0,0,1e+07,0,0,0]
 [0,0,0,1e+07,0,0]
 [0,0,0,0,1e+07,0]
 [0,0,0,0,0,1e+07]]

SC iso:  [[2.2714e+07,695958,810825,0,0,0]
 [697941,1.24744e+07,625334,0,0,0]
 [812605,625132,1.34205e+07,0,0,0]
 [0,0,0,1.26525e+07,0,0]
 [0,0,0,0,1.3885e+07,0]
 [0,0,0,0,0,1.21922e+07]]

SC aniso:  [[2.55792e+07,605641,763194,-8.98619e-11,-7.89174e-12,1.05627e-14]
```

The `tfel.material.homogenization` module, `ParticulateMicrostructure`

```
[14]
A_i_DS=hmDS.mean_strain_localisation_tensors
print("A_0_DS: ",A_i_DS[0],"A_1_DS: ",A_i_DS[1])

micro_1.replaceMatrixPhase(IM0)
pola=[tm.Stensor3D(6*[0.]),tm.Stensor3D(6*[1.e8])]
hmDS_pola=hm.computeDiluteScheme(micro_1,polarisations=pola)
P_eff_DS=hmDS_pola.effective_polarisation
print("P_eff_DS: ",P_eff_DS)
```

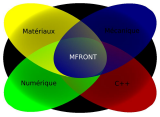
```
A_0_DS:  [[1,0,0,0,0,0]
 [0,1,0,0,0,0]
 [0,0,1,0,0,0]
 [0,0,0,1,0,0]
 [0,0,0,0,1,0]
 [0,0,0,0,0,1]] A_1_DS:  [[0.0285593,0.00286032,0.00286032,0,0,0]
 [0.00286032,0.0285593,0.00286032,0,0,0]
 [0.00286032,0.00286032,0.0285593,0,0,0]
 [0,0,0,0.025699,0,0]
 [0,0,0,0,0.025699,0]
 [0,0,0,0,0,0.025699]]
P_eff_DS:  [ 342800 342800 342800 256990 256990 256990 ]
```

Some computation at the structure scale

mfront

Homogenized behaviour

`MoriTanaka.mfront`

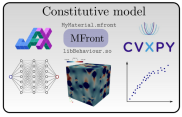


⇒

`dolfinx_materials`

Interface

```
import dolfinx_materials
```



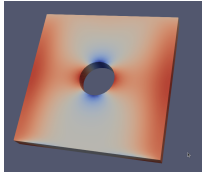
- github.com/bleyerj/dolfinx_materials

⇒

FEniCS-x

Structural analysis

`hole_plate.py`



Some computation at the structure scale

```
material = MFrontMaterial(
    os.path.join(current_path, "mfront_laws/src/libBehaviour.so"),
   "MoriTanaka",
    material_properties={'E0': E0,'nu0': nu0,'Ei': Ei,'nui': nui,'a': a,'b': b,'c': c}
)

qmap = QuadratureMap(domain, 2, material)

def fraction(x):
    return 0.1*(1+0*x[0])

def normal_a(x):
    theta=(2*abs(x[1]/L-0.5))*np.pi/2
    return np.sin(theta),0.*np.sin(theta),np.cos(theta)

frac = fem.Function(W)
frac.interpolate(fraction)
na = fem.Function(V)
na.interpolate(normal_a)

qmap.register_external_state_variable("frac", frac)
qmap.register_external_state_variable("na", na)
```

Some computation at the structure scale

```
15  @MaterialProperty real E0;
16  @MaterialProperty real nu0;
17  @MaterialProperty real Ei;
18  @MaterialProperty real nui;
19  @MaterialProperty real a;
20  @MaterialProperty real b;
21  @MaterialProperty real c;
22
23  @ExternalStateVariable real frac;
24  @ExternalStateVariable tvector<3,real> na;
25
26  @LocalVariable Stensor4 Ce;
27
28  @Integrator{
29  const tvector<3,real> ez={0.,0.,1.};
30  const tvector<3,real> ex={1.,0.,0.};
31  auto pro=cross_product<real>(na,ez);
32  auto no=norm(pro);
33  tvector<3,real> nb = (no==0.) ? ex : 1./no*pro ;
34  using namespace tfel::material::homogenization::elasticity;
35  Ce=computeOrientedMoriTanakaScheme<real>(E0,nu0,frac,Ei,nui,na,a,nb,b,c);
36  sig = Ce*(eto+deto);
37  }
```

Some computation at the structure scale

```
0.2222222222222222
temps total :  1.8116800785064697
0 [[ 0.05160761 -0.01178059 -0.0084022   0.01985264  0.00194916 -0.00024863]]
0 [ 5.14697354e+06 -2.73162729e+03 -1.00381303e+03 -3.78895769e+04
 -1.17305114e+04 -7.97593828e+01]
0.4444444444444444
temps total :  1.6045703887939453
0 [[ 0.10321522 -0.02356118 -0.0168044   0.03970527  0.00389832 -0.00049726]]
0 [ 1.02939471e+07 -5.46325459e+03 -2.00762607e+03 -7.57791537e+04
 -2.34610228e+04 -1.59518766e+02]
0.6666666666666666
temps total :  1.585526943206787
0 [[ 0.15482283 -0.03534176 -0.0252066   0.05955791  0.00584748 -0.00074588]]
0 [ 1.54409206e+07 -8.19488188e+03 -3.01143910e+03 -1.13668731e+05
 -3.51915342e+04 -2.39278148e+02]
0.8888888888888888
```

The linear visco-elastic problem

Spheres in isotropic matrix		Complex microstructure
Linear $\underline{\sigma} = \underline{\underline{C}} : \underline{\varepsilon}$	$\Rightarrow$	Non-linear $\underline{\sigma}(\underline{\varepsilon})$
RVE		Structure scale $\times n_G$

The linear visco-elastic problem

Spheres in isotropic matrix		Complex microstructure
Linear $\underline{\sigma} = \underline{\underline{C}} : \underline{\varepsilon}$	$\Rightarrow$	Non-linear $\underline{\sigma}(\underline{\varepsilon})$
RVE		Structure scale $\times n_G$
Complex microstructure	$\Rightarrow$	Many internal variables $\times n_G$

The linear visco-elastic problem

Spheres in isotropic matrix		~~Complex microstructure~~
Linear $\underline{\sigma} = \underline{\underline{C}} : \underline{\varepsilon}$	$\Rightarrow$	Non-linear $\underline{\sigma}(\underline{\varepsilon})$
RVE		Structure scale $\times n_G$
Complex microstructure	$\Rightarrow$	Many internal variables $\times n_G$

Molinari and Idiart schemes

Additive interaction law (Molinari) :

$$\dot{\underline{\varepsilon}}_i - \dot{\underline{\varepsilon}}_m = \left[\left(\underline{\underline{\mathbf{M}}}^v_m\right)^{-1} - \left(\underline{\underline{\mathbf{P}}}^v_{i,m}\right)^{-1}\right]^{-1} : (\underline{\sigma}_i - \underline{\sigma}_m) + \left[\underline{\underline{\mathbf{M}}}^{-1}_e - \left(\underline{\underline{\mathbf{P}}}^e_{i,m}\right)^{-1}\right]^{-1} : (\dot{\underline{\sigma}}_i - \dot{\underline{\sigma}}_m)$$

- *A self consistent approach of the large deformation polycrystal viscoplasticity* (Molinari, 1987)

Variational approach (Idiart) :

$$\min_{\underline{\alpha}} \left\{ \min_{\underline{\varepsilon} \in \mathcal{K}(\bar{\underline{\varepsilon}})} \langle w(\underline{\varepsilon}, \underline{\alpha}) \rangle + \Delta t \langle \varphi(\dot{\underline{\alpha}}) \rangle \right\}$$

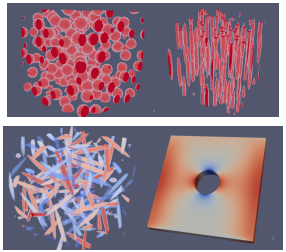
$$w(\underline{\varepsilon}, \underline{\alpha}) = \frac{1}{2}(\underline{\varepsilon} - \underline{\alpha}) : \underline{\underline{\mathbf{C}}} : (\underline{\varepsilon} - \underline{\alpha})$$

$$\varphi(\dot{\underline{\alpha}}) = \frac{1}{2}\dot{\underline{\alpha}} : \underline{\underline{\mathbf{M}}} : \dot{\underline{\alpha}}$$

$$\min_{\bar{\underline{\alpha}}, \underline{\alpha}_s} \left\{ \min_{\underline{e} \in \mathcal{K}(\bar{\underline{\varepsilon}} - \bar{\underline{\alpha}})} w(\underline{e} - \underline{\alpha}_s) + \Delta t \min_{\dot{\underline{\alpha}}_c \in \mathcal{K}(\dot{\bar{\underline{\alpha}}})} \varphi(\dot{\underline{\alpha}}_c + \dot{\underline{\alpha}}_s) \right\}$$

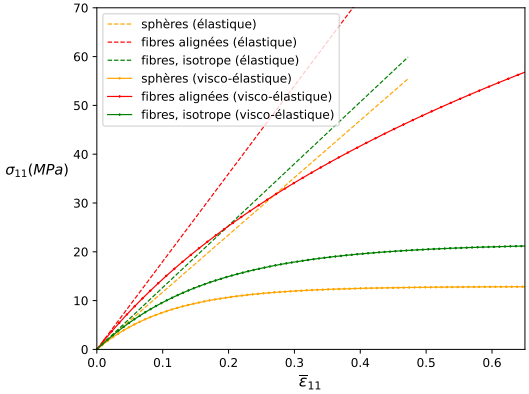
- *Thermodynamic potentials for viscoelastic composites* (Idiart, 2025)

Reinforced viscoelastic material



Number of elements	Behaviour	Total
20k	0.21s	1.16s
40k	0.41s	3.31s

Average time/proc./it. (14 proc., 13th Gen Intel® Core™ i7)



Axial stress near the hole (Idiart's scheme, different orientations)

Reinforced viscoelastic material

```
# Loading the behaviour
material = MFrontMaterial(
    os.path.join(current_path, "mfront_laws/src/libBehaviour.so"),
    "Idiart_elastic_inclusions",
    material_properties={'ka0': 0.5555555e8,'mu0': 0.41666666e8,'kav0': 0.4e8,'muv0': 0.2e8,'kar': 0.384615385
    #"Idiart_random_elastic_inclusions",
    #material_properties={'ka0': 0.5555555e8,'mu0': 0.41666666e8,'kav0': 0.4e8,'muv0': 0.2e8,'kar': 0.38461538
   #"Molinari_viscoelastic",
   #material_properties={'el':0,'ka0': 0.5555555e8,'mu0': 0.41666666e8,'kav0': 0.4e8,'muv0': 0.2e8,'kar': 0.38
   #"Molinari_Explicit",
   #material_properties={'E_0': 1.e8,'nu_0': 0.2,'sigy0': 10.e6,'kk0': 0.,'fi': 0.1,'E_i': 1.e9,'nu_i': 0.3},
   #"Molinari_MT",
   #material_properties={'E_0': 1.e8,'nu_0': 0.2,'sigy0': 10.e6,'kk0': 0.,'fi': 0.1,'E_i': 1.e9,'nu_i': 0.3},
)

if rank == 0:
    print(material.internal_state_variable_names)
    print(material.gradient_names, material.gradient_sizes)
    print(material.flux_names, material.flux_sizes)

qmap = QuadratureMap(domain, 2, material)
```

Reinforced viscoelastic material

```
69  P0e=computeAxisymmetricalHillPolarisationTensor(kg0,n,e);
70  P0v=computeAxisymmetricalHillPolarisationTensor(kgv0,n,e);
71
72  C_=C0+fr*invert(invdC+P0e-fr*P0e);
73  M_=M0+fr*invert(P0v-fr*P0v);
74  Cs=fr/f0*invert(invert(C0)-P0e);
75  D0=M0-tau0*C0;
76  Ms=tau0*Cs+fr/f0*(D0-D0*P0v*D0);}
77
78  @Integrator {
79  //jacobian
80  tfel::math::map_derivative<0,0,Stensor,Stensor>(mat) =(M_/dt+C_);
81  tfel::math::map_derivative<6,0,Stensor,Stensor>(mat)=((M0+D0*P0v*(M_-M0))/dt+C0);
82  tfel::math::map_derivative<0,6,Stensor,Stensor>(mat)=((M_-M0)*(I-P0v*D0)/dt+C_-C0);
83  tfel::math::map_derivative<6,6,Stensor,Stensor>(mat)=-((Ms-D0*P0v*(M_-M0)*(I-P0v*D0))/dt+Cs);
84
85  tfel::math::map_derivative<0,0,Stensor,real>(f_)=C_*(eto+deto)-C_*(alpha_)-(C_-C0)*(alpha_s_r);
86  tfel::math::map_derivative<6,0,Stensor,real>(f_)=-(C0*alpha_-Cs*alpha_s_r-C0*(eto+deto));
87
88  TinyMatrixInvert<12,real>::exe(mat);
89  tfel::math::tmatrix<12,1,real> dv=mat*f_;
90
91  dalpha_=tfel::math::map_derivative<0,0,Stensor,real>(dv);
92  dalpha_s_r=tfel::math::map_derivative<6,0,Stensor,real>(dv);
93
94  sig+=C_*(deto-dalpha_-dalpha_s_r)+C0*dalpha_s_r;
```

The non-linear problem

Spheres in isotropic matrix		Complex microstructure
Linear $\underline{\sigma} = \underline{\underline{C}} : \underline{\varepsilon}$	$\Rightarrow$	Non-linear $\underline{\sigma}(\underline{\varepsilon})$
RVE		Structure scale $\times n_G$

The non-linear problem

Spheres in isotropic matrix		~~Complex microstructure~~
Linear $\underline{\sigma} = \underline{\underline{C}} : \underline{\varepsilon}$	$\Rightarrow$	Non-linear $\underline{\sigma}(\underline{\varepsilon})$
RVE		Structure scale $\times n_G$
Complex microstructure	$\Rightarrow$	Many internal variables $\times n_G$

The non-linear problem

Spheres in isotropic matrix		~~Complex microstructure~~
Linear $\underline{\sigma} = \underline{\underline{C}} : \underline{\varepsilon}$	$\Rightarrow$	Non-linear $\underline{\sigma}(\underline{\varepsilon})$
RVE		Structure scale $\times n_G$
Complex microstructure	$\Rightarrow$	Many internal variables $\times n_G$
Non-linear on each phase	$\Rightarrow$	Need to integrate the local behaviour Need to linearize the behaviour

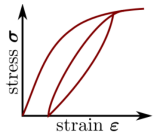
Generic local behaviours and tangent operators

- Macroscopic tangent operator can sometimes be computed with local tangent operators

Local behaviour

`matrix.mfront`

`inclusion.mfront`



Internal variables

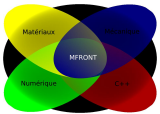
`@BehaviourVariable`

$\Rightarrow$

$$\underline{\sigma}_i(\underline{\varepsilon}_i), \frac{\partial \Delta \underline{\sigma}_i}{\partial \Delta \underline{\varepsilon}_i}$$

Homogenized behaviour

`homogenization_scheme.mfront`



Internal variables per phase

$$\underline{\Sigma}(\underline{E}), \frac{\partial \Delta \underline{\Sigma}}{\partial \Delta \underline{E}}$$

Molinari additive law/ Phase transformation in metals

- Local behaviours can be arbitrary, implementation of homogenized behaviour will not change

  - (Mercier and al. 2019)

$$\dot{\underline{\varepsilon}}_\alpha - \dot{\underline{\varepsilon}}_m = \left[\left(\underline{\underline{\mathbb{M}}}_m^v\right)^{-1} - \left(\underline{\underline{\mathbb{P}}}_\alpha^v\right)^{-1}\right]^{-1} : (\underline{\sigma}_\alpha - \underline{\sigma}_m) + \left[\left(\underline{\underline{\mathbb{M}}}_m^e\right)^{-1} - \left(\underline{\underline{\mathbb{P}}}_\alpha^e\right)^{-1}\right]^{-1} : (\dot{\underline{\sigma}}_\alpha - \dot{\underline{\sigma}}_m)$$

$$\dot{\underline{\varepsilon}}^{\mathrm{vp}}(\underline{\sigma}) = \frac{\partial \dot{\underline{\varepsilon}}^{\mathrm{vp}}}{\partial \underline{\sigma}}(\underline{\sigma}_m) : \underline{\sigma} + \dot{\underline{\varepsilon}}^{\mathrm{vp}}_{\mathrm{ref}}(\underline{\sigma}_m)$$

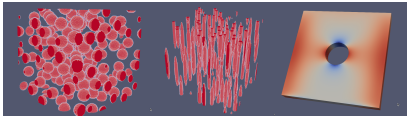
$$\underline{\underline{\mathbb{M}}}_m^v = \frac{\partial \dot{\underline{\varepsilon}}^{\mathrm{vp}}}{\partial \underline{\sigma}}(\underline{\sigma}_m) = \frac{1}{\Delta t}\left[\left(\frac{\partial \underline{\sigma}_m}{\partial \underline{\varepsilon}_m}\right)^{-1} - \underline{\underline{\mathbb{M}}}_m^e\right]$$

  - (Coret 2001) + "Taylor/Voigt" scheme

$$\underline{\mathbb{E}} = \sum_\alpha f_\alpha(T)\underline{\varepsilon}_\alpha + \underline{\mathbb{E}}^{\mathrm{tp}}(T, \underline{\Sigma}) + \underline{\mathbb{E}}^{\mathrm{thm}}(T)$$

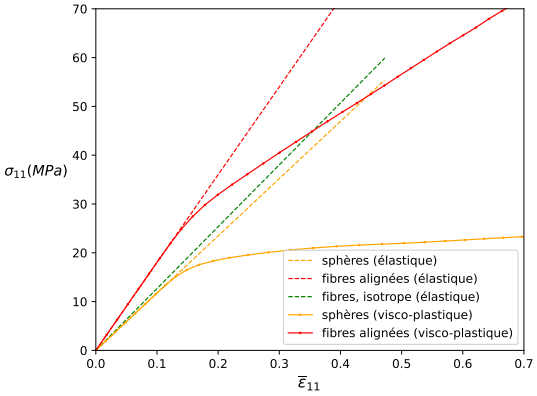
$$\frac{\partial \underline{\Sigma}}{\partial E} = \sum_\alpha f_\alpha \frac{\partial \underline{\sigma}_\alpha}{\partial \underline{\varepsilon}_\alpha}\left(\underline{\underline{\mathbb{I}}} - \frac{\partial \underline{\mathbb{E}}^{\mathrm{tp}}}{\partial \underline{\Sigma}} : \frac{\partial \underline{\Sigma}}{\partial \underline{\mathbb{E}}}\right)$$

Visco-plastic matrix and elastic inclusions



Number of elements	Behaviour	Total
20k	0.68s	1.69s
40k	1.22s	4.29s

Average time/proc./it. (14 proc.)



Axial stress near the hole (Molinari law, visco-plastic matrix)

Visco-plastic matrix and elastic inclusions

```
52  @BehaviourVariable mat {
53  file: "Perzyna.mfront",
54  variables_suffix: "0",
55  store_gradients: false,
56  store_thermodynamic_forces: false,
57  external_names_prefix: "MatrixPhase",
58  shared_external_state_variables: {".+"}
59  };
60
```

Visco-plastic matrix and elastic inclusions

```
// computation of tangent modulus and evp_dot for following time step;
  initialize(mat);
  mat.eto=eto0;
  mat.deto=deto0;
  mat.sig=sig0;
  constexpr auto mat_smflag = TangentOperatorTraits<MechanicalBehaviourBase::STANDARDSTRAINBASEDBEHAVIO
  const auto r1 = mat.integrate(mat_smflag,CONSISTENTTANGENTOPERATOR);
  Dt0 = mat.getTangentOperator();

  using namespace tfel::material::homogenization::elasticity;
  Stensor4 invPtgi;
  constexpr real epsi = std::numeric_limits<real>::min();
  auto invAtg=mat.invAtg;
  auto deter =tfel::math::det(invAtg);
  if (abs(deter)<abs(epsi)){
    Atg=Stensor4::zero();
    Ltg=Stensor4::zero();
  } else {
    Atg=invert(invAtg);
    auto pair = tfel::material::computeKappaMu(Atg);
    auto mutg=std::get<1>(pair);
    invPtgi=5*mutg*K;
    Atg=2*mutg*K;
    Ltg=-1./(3*mutg)*K;
  }
  evp_dot_0=mat.evp_dot;
```

The non-linear problem

Spheres in isotropic matrix		Complex microstructure
Linear $\underline{\sigma} = \underline{\underline{C}} : \underline{\varepsilon}$	$\Rightarrow$	Non-linear $\underline{\sigma}(\underline{\varepsilon})$
RVE		Structure scale $\times n_G$

The non-linear problem

Spheres in isotropic matrix		Complex microstructure
Linear $\underline{\sigma} = \underline{\underline{C}} : \underline{\varepsilon}$	$\Rightarrow$	Non-linear $\underline{\sigma}(\underline{\varepsilon})$
RVE		Structure scale $\times n_G$
Non-linear	$\Rightarrow$	Need to better take into account the fluctuations

The non-linear problem

Spheres in isotropic matrix		Complex microstructure
Linear $\underline{\sigma} = \underline{\underline{C}} : \underline{\varepsilon}$	$\Rightarrow$	Non-linear $\underline{\sigma}(\underline{\varepsilon})$
RVE		Structure scale $\times n_G$
Non-linear	$\Rightarrow$	Need to better take into account the fluctuations

$$\langle \underline{\varepsilon} \otimes \underline{\varepsilon} \rangle_r$$

Ponte-Castañeda variational estimates

$$w(\underline{\varepsilon}) = \sum_r \chi^r \, w^r(\underline{\varepsilon}) \qquad \overline{w} = \min_{\underline{\varepsilon} \in \mathcal{K}(\overline{\underline{\varepsilon}})} \langle w(\underline{\varepsilon}) \rangle$$

$$\underline{\underline{\mathbb{L}}}^r = \frac{\partial^2 w^r}{\partial \underline{\varepsilon}^2} \left(\langle \underline{\varepsilon} \otimes \underline{\varepsilon} \rangle_r \right) \underset{\text{LCC}}{\rightarrow} \operatorname{div} \left(\underline{\underline{\mathbb{L}}}^r : \underline{\varepsilon} \right) = 0 \underset{\text{MF scheme}}{\rightarrow} \overline{w}(\overline{\underline{\varepsilon}}) \underset{\text{derivation}}{\rightarrow} \frac{2}{c_r} \frac{\partial \overline{w}}{\partial \underline{\underline{\mathbb{L}}}^r} = \langle \underline{\varepsilon} \otimes \underline{\varepsilon} \rangle_r$$

$$w(\underline{\varepsilon}) = \frac{\sigma_0 \varepsilon_0}{m+1} \left(\frac{\varepsilon_{\text{eq}}}{\varepsilon_0} \right)^{m+1}, \quad \underline{\sigma} = \sigma_0 \left(\frac{\varepsilon_{\text{eq}}}{\varepsilon_0} \right)^m \underline{\mathrm{n}}, \qquad \underline{\mathrm{n}} = \frac{2}{3\varepsilon_{\text{eq}}} \underline{\varepsilon}^d$$

$$\frac{\partial^2 w^r}{\partial \underline{\varepsilon}^2} = \frac{\sigma_0}{\varepsilon_0} \left(\frac{\varepsilon_{\text{eq}}}{\varepsilon_0} \right)^{m-1} \left[m \underline{\mathrm{n}} \otimes \underline{\mathrm{n}} + \frac{2}{3} \left(\underline{\underline{\mathrm{K}}} - \frac{3}{2} \underline{\mathrm{n}} \otimes \underline{\mathrm{n}} \right) \right] = f(\underline{\varepsilon} \otimes \underline{\varepsilon})$$

Ponte-Castañeda variational estimates

$$\underline{\underline{\mathbf{L}}}^r = \frac{\partial^2 w^r}{\partial \underline{\boldsymbol{\varepsilon}}^2}\left(\langle \underline{\boldsymbol{\varepsilon}} \otimes \underline{\boldsymbol{\varepsilon}} \rangle_r\right) \quad \underset{\text{LCC}}{\rightarrow} \quad \operatorname{div}\left(\underline{\underline{\mathbf{L}}}^r : \underline{\boldsymbol{\varepsilon}}\right) = 0 \quad \underset{\text{MFH}}{\rightarrow} \quad \overline{w}(\overline{\underline{\boldsymbol{\varepsilon}}}) \quad \underset{\text{derivation}}{\rightarrow} \quad \frac{2}{c_r}\frac{\partial \overline{w}}{\partial \underline{\underline{\mathbf{L}}}^r} = \langle \underline{\boldsymbol{\varepsilon}} \otimes \underline{\boldsymbol{\varepsilon}} \rangle_r$$

`@BehaviourVariable` → `tfel::material::homogenization` → `tfel::math::enzyme`

Conclusions and perspectives

- Conclusions

Conclusions and perspectives

- Conclusions
  - Construction of a toolbox for linear and non-linear homogenization

Conclusions and perspectives

- Conclusions
  - Construction of a toolbox for linear and non-linear homogenization
  - Calling a local behaviour with its tangent operator

Conclusions and perspectives

- Conclusions
  - Construction of a toolbox for linear and non-linear homogenization
  - Calling a local behaviour with its tangent operator
  - Fast computation at the structure scale

Conclusions and perspectives

- Conclusions
  - Construction of a toolbox for linear and non-linear homogenization
  - Calling a local behaviour with its tangent operator
  - Fast computation at the structure scale
- Perspectives

Conclusions and perspectives

- Conclusions
  - Construction of a toolbox for linear and non-linear homogenization
  - Calling a local behaviour with its tangent operator
  - Fast computation at the structure scale
- Perspectives
  - Addition of `@Brick`'s to reduce the volume of code

Conclusions and perspectives

- Conclusions
  - Construction of a toolbox for linear and non-linear homogenization
  - Calling a local behaviour with its tangent operator
  - Fast computation at the structure scale
- Perspectives
  - Addition of `@Brick`'s to reduce the volume of code
  - Automatic differentiation

Conclusions and perspectives

- Conclusions
  - Construction of a toolbox for linear and non-linear homogenization
  - Calling a local behaviour with its tangent operator
  - Fast computation at the structure scale
- Perspectives
  - Addition of `@Brick`'s to reduce the volume of code
  - Automatic differentiation
  - Reduced-order models for homogenization, machine-learning based homogenization...

Conclusions and perspectives

- Conclusions
  - Construction of a toolbox for linear and non-linear homogenization
  - Calling a local behaviour with its tangent operator
  - Fast computation at the structure scale
- Perspectives
  - Addition of `@Brick`'s to reduce the volume of code
  - Automatic differentiation
  - Reduced-order models for homogenization, machine-learning based homogenization...

antoine.martin@cea.fr, github.com/thelfer/tfel