1 Numerical Examples

In this part some numerical examples using this module is presented. First different kinds of inclusions are listed. Then 2d and 3d examples both in uni field and multi field modelling are given. At the end of this part the usage of the module is given in the IPython environment.

1.1 Various Inclusions

Inclusions in two dimensional or three dimensional cases can be added to the unit cell. User defined mesh and geometry can also be applied.

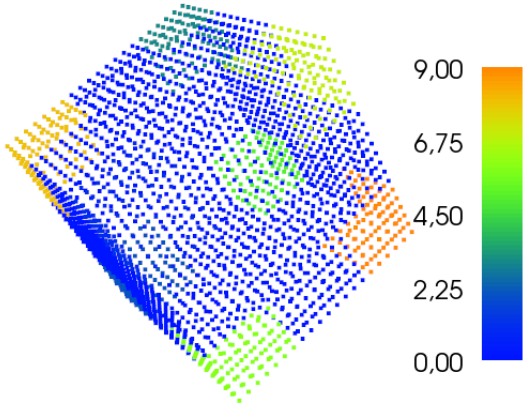

Figure 1.1: Typical free energy profile

[fig: inclusion and geometry 2d, 3d, different inclusions]

1.2 2D Modelling

A unit square is considered in two dimensional calculation. At first a uni field problem is introduced, specifically a Saint Venant-Kirchhoff material in mechanical field. The model is given in the previous chapter. Material parameters and geometry parameters are given in the following table. The inclusion is simply a center located circle.

[table: mat para, geom para]

Boundary condition is periodic boundary condition and the fluctuation at corners is fixed to prevent rigid body movements. And the macro field is set as

$$\bar{\mathbf{F}} = \begin{bmatrix} 0.9 & 0 \\ 0 & 1 \end{bmatrix},$$

which is an uni axial compression test. Here the macro deformation is set not very large. Because of the non-linearity nature of the problem, the load cannot set to be too high. For large macro deformation input a quasi static loading process needs to be imposed. This will achieve more obvious loading. The fluctuation plot, strain plot and stress plot is demonstrated here.

[fig: 2d uni fluctuation, strain, stress]

The calculated homogenized tangent moduli is

$$\mathbb{C}_{\text{eff}} = \tag{1.1}$$

As for multi field modelling, the material considered is Neo Hookean type electroactive polymer material in both mechanical and electrical fields,

$$\psi\left(\mathbf{C}, \mathbf{E}\right) = \frac{1}{2}\mu_0\left(\text{tr}[\mathbf{C}] - 3\right) + \frac{\lambda}{4}\left(J^2 - 1\right) - \left(\frac{\lambda}{2} + \mu\right)\ln J - \frac{1}{2}\epsilon_0\left(1 + \frac{\chi}{J}\right) J\left[\mathbf{C}^{-1} : \left(\mathbf{E} \otimes \mathbf{E}\right)\right] \tag{1.2}$$

Macro field input is

$$\overline{\mathbf{F}} = \begin{bmatrix} 0.9 & 0 \\ 0 & 1 \end{bmatrix}, \overline{\mathbf{E}} = \begin{bmatrix} 0.9 \\ 0 \end{bmatrix}$$

[fig: 2d multi]

1.3 3D Modelling

In 3D case we consider a unit cube, where a rectangular block is set as inclusion which cuts through the unit cell. Uni field and multi field problems are both accounted in this circumstance.

[fig: 3d uni]

[fig: 3d multi]

1.4 Simulation Template

In this section a simulation template is provided. The presentation of this template introduces all the common interface of this module, which allows users to gain an overview of all the usages of this module. This code sample starts with importing all the files of unit cell module. Then geometry is defined within `cell_geom.py` followed with material definition within `cell_material.py`. Completing the setting of the problem calculation is initiated with defining an object of `MicroComputation` from `cell_computation.py`. After inputting macro field and field variables which need to be solved, micro fluctuation can be calculated using the member method `comp_fluctuation()` and the user defined solver parameters. It is seen that an instance of `MicroComputation` can be called every time step and reused among several different cells if the setting of geometry and materials does not change. The only thing to notice is that a new FEniCS `Function` object should be provided in the interim. This template can be used in IPython interactively or embedded in other computation which requires a unit cell investigation.

```
from dolfin import *

import numpy as np

# Module files
import cell_geom as geom
import cell_material as mat
import cell_computation as comp

# Set linear algebra backend, supported are Eigen and PETSc
parameters['linear_algebra_backend'] = 'Eigen'

# Geometry definition
mesh = Mesh(r'./m_fine.xml')
cell = geom.UnitCell(mesh)
inc = geom.InclusionCircle(2, (0.5, 0.5), 0.25)
inc_di = {'circle_inc': inc}
cell.set_append_inclusion(inc_di)

# Material definition
E_m, nu_m, E_i, nu_i = 10.0, 0.3, 1000.0, 0.3
mat_m = mat.st_venant_kirchhoff(E_m, nu_m)
mat_i = mat.st_venant_kirchhoff(E_i, nu_i)
mat_li = [mat_m, mat_i]

# Fields definition
VFS = VectorFunctionSpace(cell.mesh, "CG", 1,
                          constrained_domain=geom.PeriodicBoundary_no_corner(2))

def deform_grad_with_macro(F_bar, w_component):
    return F_bar + grad(w_component)

w = Function(VFS)
strain_space = TensorFunctionSpace(mesh, 'DG', 0)

# Micro Computation instantiation
compute = comp.MicroComputation(cell, mat_li,
                                [deform_grad_with_macro],
                                [strain_space])

# Input macro field and field variable
F_bar = [0.9, 0., 0., 1.]
```

```
compute.input([F_bar], [w])

# Set linear solver and parameters and solve the problem
comp.set_solver_parameters('non_lin_newton', lin_method='direct',
                       linear_solver='cholesky')
compute.comp_fluctuation(print_progress=True, print_solver_info=False)

compute.view_fluctuation()

# Load step
delta = 0.01

# Calculate with different load steps
for i in range(10):
    F_bar[0] -= delta
    print F_bar
    compute.input([F_bar], [w])
    compute.comp_fluctuation(print_progress=True, print_solver_info=False)
```

1.5 Convergence Comparison with Finite Difference Calculation

The convergence of the homogenized effective tangent moduli is investigated with Finite Difference calculation. Here Finite Difference calculation stands for the difference of the averaged extended stress with respect to small change of the macro extended strain input. This method leads to the following expression,

$$\mathbb{C}_{\text{eff}} = \frac{\partial \mathbf{P}_{\text{avg}}\left(\overline{\mathbf{F}}, \widetilde{\mathbf{F}}\right)}{\partial \overline{\mathbf{F}}} \;\Rightarrow\; \mathbb{C}_{\text{eff}} = \frac{\Delta \mathbf{P}_{\text{avg}}}{\Delta \overline{\mathbf{F}}}. \tag{1.3}$$

The fraction of vectors or matrices is arranged in the same manner with the effective tangent moduli. In the calculation varying only a component of $\overline{\mathbf{F}}$ is realized and the change of $\mathbf{P}_{\text{avg}}$ is calculated. The division ends with corresponding terms in $\mathbb{C}_{\text{eff}}$.

The result shows that in uni field and multi fields the implementation of the previous derivation gives a good approximate to the Finite Difference calculation of effective tangent moduli. The script for multiple fields modelling is given here.

```
import sys
sys.path.append(r'../')

from dolfin import *
import numpy as np

import cell_computation as com
import cell_geom as ce
import cell_material as ma
from copy import deepcopy

import logging
logging.getLogger('FFC').setLevel(logging.WARNING)

# Geometry definition
mesh = Mesh(r"../m.xml")
cell = ce.UnitCell(mesh)
inc = ce.InclusionCircle(2, (0.5, 0.5), 0.25)
inc_di = {'circle_inc': inc}
cell.set_append_inclusion(inc_di)

# Electroactive polymer material model
E_m, nu_m, Kappa_m = 2e5, 0.4, 7.
n = 1000
E_i, nu_i, Kappa_i = 1000 * E_m, 0.3, n * Kappa_m

mat_m = ma.neo_hook_eap(E_m, nu_m, Kappa_m)
mat_i = ma.neo_hook_eap(E_i, nu_i, Kappa_i)
mat_li = [mat_m, mat_i]

# Field variables
VFS = VectorFunctionSpace(cell.mesh, "CG", 1,
                          constrained_domain=ce.PeriodicBoundary_no_corner(2))
FS = FunctionSpace(cell.mesh, "CG", 1,
                   constrained_domain=ce.PeriodicBoundary_no_corner(2))
w = Function(VFS)
```

```
el_pot_phi = Function(FS)
strain_space_w = TensorFunctionSpace(mesh, 'DG', 0)
strain_space_E = VectorFunctionSpace(mesh, 'DG', 0)

def deform_grad_with_macro(F_bar, w_component):
    return F_bar + grad(w_component)
def e_field_with_macro(E_bar, phi):
    return E_bar - grad(phi)

# Computation initiation
comp = com.MicroComputation(cell, mat_li,
                        [deform_grad_with_macro, e_field_with_macro],
                        [strain_space_w, strain_space_E])

# Wrap for calculating averaged stress
def avg_mer_stress(F_bar, E_bar):
    comp.input([F_bar, E_bar], [w, el_pot_phi])
    comp.comp_fluctuation()
    return comp.avg_merge_stress()

# Calculate the corresponding components and stack them into a matrix
def conv_check_component(label, compo, delta):
    C_eff_component_FD = np.zeros(shape=(len(delta),6), dtype=float)
    if label is 'F':
        for i, d in enumerate(delta):
            F_minus = deepcopy(F_bar)
            F_minus[compo] = F_bar[compo] - d/2
            F_plus = deepcopy(F_bar)
            F_plus[compo] = F_bar[compo] + d/2

            P_minus = avg_mer_stress(F_minus, E_bar)
            P_plus  = avg_mer_stress(F_plus, E_bar)

            C_eff_component_FD[i,:] = (P_plus - P_minus)/d
    elif label is 'E':
        for i, d in enumerate(delta):
            E_minus = deepcopy(E_bar)
            E_minus[compo] = E_bar[compo] - d/2
            E_plus = deepcopy(E_bar)
            E_plus[compo] = E_bar[compo] + d/2

            P_minus = avg_mer_stress(F_bar, E_minus)
            P_plus  = avg_mer_stress(F_bar, E_plus)

            C_eff_component_FD[i,:] = (P_plus - P_minus)/d
    else:
        raise Exception('no such field label')

    return C_eff_component_FD

# Macro field input
F_bar = [1.1, 0., 0.1, 1.]
E_bar = [0., 0.2]

# Finite difference step
delta = [0.01, 0.01/2, 0.01/4, 0.01/8]

# Finite difference calculation (the fourth component of C_eff)
C_eff_component_FD = conv_check_component('F', 3, delta)

# Homogenization calculation
comp = com.MicroComputation(cell, mat_li,
                        [deform_grad_with_macro, e_field_with_macro],
                        [strain_space_w, strain_space_E])
comp.input([F_bar, E_bar], [w, el_pot_phi])
comp.comp_fluctuation()
C_eff = comp.effective_moduli_2()

# Observe the difference
component = C_eff[:,3]
tmp = np.outer(np.ones((len(delta),1)),np.transpose(component))
error = np.linalg.norm(tmp - C_eff_component_FD, axis=1)/np.linalg.norm(component)
```

The output of the error (or calculation difference between two methods) is in the following code block.

```
In [61]: error
```

```
Out[61]:
array([  9.12302965e-06,   2.28075104e-06,   5.70186024e-07,
         1.42533577e-07])
```

The result shows that the homogenized solution of effective tangent moduli is in good agreement with the one calculated from Finite Difference method in both uni field case and multi field case.