Homogenization for Multi Field Modelling

Part II: Implementation and Numerical Examples

Yi Hu

February 24, 2016

Contents

1 Derivation and Implementation

In this chapter modelling of composite materials in multiple fields is investigated. Our target is to achieve multi scale modelling. The current work is a Unit Cell module, which supports the calculation of the micro scale. Next homogenization is carried out to compute the averaged strain, averaged stress and ends with the calculation of effective tangent moduli. In order to obtain an effective and generic formulation energy method is exploited. The considered materials are Neo-Hookean materials and the composites contain periodic micro structures.

Details of the problem setting and derivation are presented in the following two sections. Then some issues in implementation such as alternative way of calculation are shortly discussed. Finally the documentation rules of the code and some points about the usage are clarified in the end of this chapter.

1.1 Problem Setting

The discussion focus on the unit cell consisting of multiple materials. This unit cell can be two dimensional or three dimensional, namely a square or a cube with edge length 1. Each material component occupies a range of domain in the cell, which are inclusions. All the inclusions possess no intersections in the domain. Several fields are taken into account, such as mechanical field, temperature field and electrical field. Material types are hyperelasticity, thermomechanical Simo-Pister material (introduced in the course Elements of Non-linear Continuum Thermodynamics) and Neo-Hookean electroactive rheological material [ref: keip]. All the materials in the current material library share the common feature, i.e. no time dependency. No plasticity and viscosity are considered in the current work.

The material energy potential per unit volume can be summarized here.

Saint Venant-Kirchhoff Material

$$\psi\left(\mathbf{E}\right) = \frac{\lambda}{2}\left[\mathrm{tr}(\mathbf{E})\right]^2 + \mu \mathrm{tr}\left(\mathbf{E}^2\right) \tag{1.1}$$

Simo Pister Material

$$\psi\left(\theta, \mathbf{C}\right) = \frac{1}{2}\mu_0\left(I_C - 3\right) + \left(m_0 \Delta\theta \mu_0\right)\ln(\det \mathbf{C})^{\frac{1}{2}} + \frac{1}{2}\lambda_0\left[\ln\left(\det \mathbf{C}\right)^{\frac{1}{2}}\right]^2 - \rho_0 c_V\left(\theta \ln\frac{\theta}{\theta_0} - \Delta\theta\right) \tag{1.2}$$

Neo Hookean Type Electroactive Material

$$\psi\left(\mathbf{C}, \mathbf{E}\right) = \frac{1}{2}\mu_0\left(\mathrm{tr}[\mathbf{C}] - 3\right) + \frac{\lambda}{4}\left(J^2 - 1\right) - \left(\frac{\lambda}{2} + \mu\right)\ln J - \frac{1}{2}\epsilon_0\left(1 + \frac{\chi}{J}\right) J\left[\mathbf{C}^{-1} : \left(\mathbf{E} \otimes \mathbf{E}\right)\right] \tag{1.3}$$

1.2 Derivation

In this section the derivation concerning about solving the Unit Cell problem as well as calculating the effective tangent moduli are presented. Derivation of the total energy, equilibrium, and effective tangent moduli are included in this section.

We denote $\mathbf{w}$ as extended displacement, which comprises the traditional displacement in mechanical problem as well as extra virtual displacements in other fields, such as temperature T for the temperature field. Accordingly the strain in traditional mechanical problem should be regarded as the extended strain tensor measure here with notation $\mathbf{F}$. The extended stress can be defined in the same manner. Since the problem in micro scale is accounted, there are two parts in each variable, namely the averaged quantity from macro scale, $\overline{(\cdot)}$, and the fluctuation, $\widetilde{(\cdot)}$, to be solved. This relation is stated as

$$\mathbf{w} = \overline{\mathbf{w}} + \widetilde{\mathbf{w}},\ \mathbf{F} = \overline{\mathbf{F}} + \widetilde{\mathbf{F}}. \tag{1.4}$$

One advantage of this notion is that there is then no need to treat the variable separately in the derivation. Besides the merged function feature in FEniCS can be employed and realized to its greatest extent. It is then not necessary to derive the coupled terms in stiffness matrix. All the coupled terms in stiffness matrix is calculated implicitly and automatically.

1.2.1 Total Energy

For composites total energy is the sum of energy from every component. There are two different representations of the total energy. One is for the use of equilibrium. Hence the dependency in energy is the corresponding extended displacement. The other is for the calculation of extended stress and tangent moduli. In this case the total energy should be expressed with the extended strain. Using the notation given above, these two types of total energy are stated as follows (assume there are n different materials in this composite). First the original presentation of energy using extended strain is given as (here viscosity and plasticity not accounted)

$$\Pi(\overline{\mathbf{F}}, \widetilde{\mathbf{F}}) = \Pi(\mathbf{F}) = \sum_{i=1}^{n} \int \psi_i(\mathbf{F}) \, \mathrm{d}\mathbf{x}_i. \tag{1.5}$$

As extended strain is actually a function of extended displacement, meaning $\mathbf{F} = \mathbf{F}(\mathbf{w}) = \mathbf{F}(\overline{\mathbf{w}}, \widetilde{\mathbf{w}})$, the total energy can be represented as

$$\Pi(\overline{\mathbf{w}}, \widetilde{\mathbf{w}}) = \Pi(\mathbf{w}) = \sum_{i=1}^{n} \int \psi_i(\mathbf{w}) \, \mathrm{d}\mathbf{x}_i. \tag{1.6}$$

1.2.2 Equilibrium

The equilibrium of the system is built around the stationary point of total energy with respect to fluctuation. It is stated as below,

$$\frac{\partial \Pi(\mathbf{w})}{\partial \widetilde{\mathbf{w}}} = \mathbf{0} \tag{1.7}$$

Above might be a non-linear equation. The Jacobian of this equation is therefore always needed. Denoting $\mathbf{v}$ as the test function for fluctuation, $\mathbf{w}_{\text{inc}}$ as the trial function, the weak form of (1.7) and Jacobian can be expressed as

$$L = \frac{\partial \Pi(\mathbf{w})}{\partial \widetilde{\mathbf{w}}} \cdot \mathbf{v}, \;\; a = \frac{\partial L}{\partial \widetilde{\mathbf{w}}} \cdot \mathbf{w}_{\text{inc}} \tag{1.8}$$

The whole linear equation system and stiffness matrix can be obtained through assembling. From above L is the linear form and a is the bilinear form. Weak formulation of (1.7) is to search $\mathbf{w}_{\text{inc}}$ in V, such that the following holds,

$$a(\mathbf{w}_{\text{inc}}, \mathbf{v}) = L(\mathbf{v}), \;\; \forall \mathbf{v} \in \hat{V} \tag{1.9}$$

After solving this equation fluctuation is obtained.

1.2.3 Post-processing

When it comes to post-processing, the extended strain is calculated first and the total energy of the material is expressed with (1.5). Noticing (1.4)

$$\mathbf{F} = \overline{\mathbf{F}} + \widetilde{\mathbf{F}}(\widetilde{\mathbf{w}}). \tag{1.10}$$

This can be seen as a "mapping" of fluctuation into the strain space. The total energy is formulated with $\Pi(\mathbf{F})$ or $\Pi(\overline{\mathbf{F}}, \widetilde{\mathbf{F}})$.

The actual local extended stress is the derivative of local energy, namely

$$\mathbf{P}_i = \frac{\partial \psi_i(\mathbf{F})}{\partial \mathbf{F}}. \tag{1.11}$$

Then the averaged extended stress is the average of the above formula. When the unit cell has edge with 1, integral of $\mathbf{P}$ over the domain is the averaged quantity,

$$\mathbf{P}_{\text{avg}} = \sum_{i=1}^{n} \int \mathbf{P}_i \, \mathrm{d}\mathbf{x}_i. \tag{1.12}$$

This method can be implemented in FEniCS. However a more simple way of deriving is the formula,

$$\mathbf{P}_{\text{avg}} = \frac{\partial \Pi(\mathbf{F})}{\partial \mathbf{F}} = \frac{\partial}{\partial \mathbf{F}} \left(\sum_{i=1}^{n} \int \psi_i(\mathbf{F}) \, \mathrm{d}\mathbf{x}_i \right). \tag{1.13}$$

Comparing (1.11) and (1.12) the following must hold,

$$\frac{\partial}{\partial \mathbf{F}} \left(\sum_{i=1}^{n} \int \psi_i(\mathbf{F}) \, \mathrm{d}\mathbf{x}_i \right) = \sum_{i=1}^{n} \int \frac{\partial \psi_i(\mathbf{F})}{\partial \mathbf{F}} \, \mathrm{d}\mathbf{x}_i. \tag{1.14}$$

For component number n is independent of the extended strain $\mathbf{F}$, the sum over n components could be extracted out. Then the interchange of integral and derivative needs to be shown,

$$\frac{\partial}{\partial \mathbf{F}} \left(\int \psi_i(\mathbf{F}) \, \mathrm{d}\mathbf{x}_i \right) = \int \frac{\partial \psi_i(\mathbf{F})}{\partial \mathbf{F}} \, \mathrm{d}\mathbf{x}_i. \tag{1.15}$$

Numerical experiments show that this equality holds. Conceptually if energy depends only on the extended strain, the integral and derivative are interchangeable. More consideration should be devoted, if a valid proof is sought after. These two formulation also end with two different implementations in FEniCS.

Analogously the derivation of the averaged tangent moduli can be carried out in two different manners.

$$\mathbb{C}_{\text{avg}} = \sum_{i=1}^{n} \int \mathbb{C}_i \, \mathrm{d}\mathbf{x}_i = \sum_{i=1}^{n} \int \frac{\partial^2 \psi_i(\mathbf{F})}{\partial \mathbf{F}^2} \, \mathrm{d}\mathbf{x}_i. \tag{1.16}$$

$$\mathbb{C}_{\text{avg}} = \frac{\partial^2 \Pi(\mathbf{F})}{\partial \mathbf{F}^2} = \frac{\partial^2}{\partial \mathbf{F}^2} \left(\sum_{i=1}^{n} \int \psi_i(\mathbf{F}) \, \mathrm{d}\mathbf{x}_i \right) \tag{1.17}$$

Next step is to derive the homogenized parameters required for the macro scale, i.e. effective tangent moduli. This tangent moduli is the derivative of averaged extended stress $\mathbf{P}_{\text{avg}}$ with respect to macro strain $\overline{\mathbf{F}}$. $\mathbf{P}_{\text{avg}}$ is expressed with

$$\mathbf{P}_{\text{avg}} = \mathbf{P}_{\text{avg}} \left(\overline{\mathbf{F}}, \widetilde{\mathbf{F}} \right). \tag{1.18}$$

Hence the effective moduli is as follows,

$$\mathbb{C}_{\text{eff}} = \frac{\partial \mathbf{P}_{\text{avg}} \left(\overline{\mathbf{F}}, \widetilde{\mathbf{F}} \right)}{\partial \overline{\mathbf{F}}} \tag{1.19}$$

Recalling (1.13), the following formula holds, (using $\mathbf{F} = \overline{\mathbf{F}} + \widetilde{\mathbf{F}}$ and chain rule)

$$\mathbb{C}_{\text{eff}} = \frac{\partial}{\partial \overline{\mathbf{F}}} \left(\frac{1}{V} \int_{\text{cell}} \mathbf{P} \, \mathrm{d}\mathbf{x} \right) = \frac{\partial}{\partial \mathbf{F}} \left(\frac{1}{V} \int_{\text{cell}} \mathbf{P} \, \mathrm{d}\mathbf{x} \right) : \frac{\partial \overline{\mathbf{F}} + \widetilde{\mathbf{F}}}{\partial \overline{\mathbf{F}}} = \left(\frac{1}{V} \int_{\text{cell}} \mathbb{C} \, \mathrm{d}\mathbf{x} \right) : \left(\mathbb{I} + \frac{\partial \widetilde{\mathbf{F}}}{\partial \overline{\mathbf{F}}} \right)$$

$$\xrightarrow[V=1]{\text{unit cell}} \mathbb{C}_{\text{eff}} = \mathbb{C}_{\text{avg}} + \int_{\text{cell}} \mathbb{C} : \frac{\partial \widetilde{\mathbf{F}}}{\partial \overline{\mathbf{F}}} \, \mathrm{d}\mathbf{x} \tag{1.20}$$

Expressing the second term numerically is the main task in implementation, which will be discussed in the next section.

1.3 Issues in Implementation

The unit cell module is made up of three files, `cell_geom.py`, `cell_material.py`, `cell_computation.py`. `cell_material.py` defines the material in composites. Three material models are included in the material library, while user defined material can be also implemented in a easy manner. `cell_geom.py` specifies the inclusions of the material as well as the periodic mapping in geometry. Meshes can be imported or generated within FEniCS. 3d and 2d unit cells are available. The inclusion could be circle in 2d and sphere in 3d. Rectangular inclusion and brick inclusion are also realized in the current state. Marking boundaries and corners of unit cells can work within this framework, which are often needed in the boundary conditions. The main part of this module is `cell_computation.py`, pre-processing, formulation and solving, and post-processing are involved in this file. Some essential considerations of the implementation are discussed in the following parts.

1.3.1 cell_material.py

In this file class and functions for material definition are involved. The main part of this file is `class Material`. The definition of material starts with writing the energy function. This energy function receives invariants as the function arguments. Then one can initiate a new material with energy function and its parameters. Next define invariants and pass them into the already defined material. By calling the name of material (`__call__` is overrided) with the dependent functions, the instantiation of a material is complete.

Here list the definition of Saint Venant-Kirchhoff Material

```
def st_venant_kirchhoff(E_m, nu_m):
  # Material parameters
    mu = E_m / (2 * (1 + nu_m))
    lmbda = E_m * nu_m / ((1 + nu_m) * (1 - 2 * nu_m))
```

```
  # Energy function
    def psi(inv, lmbda, mu):
        return 0.5 * lmbda * (inv[0]) ** 2 + mu * inv[1]

  # Instantiation with energy function and material parameters
    svk = Material(psi, [lmbda, mu])

    def invariant1(F):
        dim = F.geometric_dimension()
        I = Identity(dim)
        C = F.T * F
        E = 0.5 * (C - I)
        return tr(E)

    def invariant2(F):
        dim = F.geometric_dimension()
        I = Identity(dim)
        C = F.T * F
        E = 0.5 * (C - I)
        return tr(E.T * E)

  # Feed the invariant generators
    svk.invariant_generator_append((0,), [invariant1, invariant2])

    return svk
```

And the usage of this code is illustrated here.

```
from dolfin import *
from cell_material import st_venant_kirchhoff

mesh = UnitSquareMesh(2, 2)
VFS = VectorFunctionSpace(mesh, 'CG', 1)
w = Function(VFS)
F = grad(w)

E_m, nu_m = 10.0, 0.3
svk = st_venant_kirchhoff(E_m, nu_m)

svk([F])
```

1.3.2 cell_geom.py

The main class of this file is `class UnitCell`. Its instance is instantiated with a FEniCS `mesh`, and inclusions can be passed at the initiation stage or by using the member method `set_append_inclusion()`. Another important method is `add_mark_boundary()`, which will mark the boundary facets, edges or corners for later formulation of boundary conditions of problems. Inclusions are added by first instantiating an object of an inclusion class, e.g. `class InclusionCircle`. Then pass it to the instance method `set_append_inclusion()`. The usage is shown below,

```
from dolfin import *
from cell_geom import UnitCell

mesh = UnitSquareMesh(40, 40, 'crossed')
inc1 = InclusionCircle(2, (0.1, 0.1), 0.5)
inc_group = {'circle_inc1': inc1}

# Direct initiation with inclusion
cell = UnitCell(mesh, inc_group)
s
# Set and append inclusion
cell = UnitCell(mesh)
cell.set_append_inclusion(inc_group)
```

Another focus in this file is to define periodic mapping for unit cell. It is not trivial with FEniCS, as it has its own rule of defining mapping. Two dimensional case can be referred with the example in the FEniCS forum, while three dimensional case needs more investigation of its internal mapping definition. For 2d case, mark the edges on coordinate axis as reference edges, and map other edges into these two edges. For 3d case, mark the main facets, which form between the main axis, edges and corners not on the main axis are filtered out. The mapping then is grouped into two different categories, edge mapping and facet mapping. Edges in the same direction should be mapped to the edge on the main axis, while facet mapping is simply between two opposite facets. Usage of this periodic mapping is as follows,

```
from dolfin import *
from cell_geom import PeriodicBoundary_no_corner

a, b, c = 3, 6, 9
mesh_3d = UnitCubeMesh(a, b, c)

# 3 is for 3d
FS_3d = FunctionSpace(mesh_3d, 'CG', 1, constrained_domain=PeriodicBoundary_no_corner(3))
f = Function(FS_3d)
```

1.3.3 cell_computation.py

This is the main part of the unit cell module. Pre-processing contains merging functions and splitting functions, generate macro extended strain. Then it enters the formulation of Finite Element problem, which includes calculating total energy, imposing boundary conditions, bilinear and linear form formulation. With `comp_fluctuation()` the fluctuation $\widetilde{\mathbf{w}}$ is obtained. Post-processing concerns about the calculation of strain, stress, averaged strain, averaged stress, averaged moduli, and effective moduli. Plotting the result with FEniCS is also wrapped for the use of material modelling.

In the pre-processing step, using `field_merge()`, `field_split()` or `set_field()` makes the multi field modelling easy to handle with. Macro extended strain should also be merged and split in the case of multi field modelling. All these steps are wrapped in `input()` method.

As for the formulation and solving step, the code is rather straightforward. The boundary conditions here are Dirichlet boundary condition for fluctuation at every corner of the unit cell.

There are much more techniques in the post-processing step. The work in [ref: keip] is referred. The essential part is to derive the formula for the term

$$\frac{\partial \widetilde{\mathbf{F}}}{\partial \overline{\mathbf{F}}}$$

The equilibrium of the disturbed system is built with

$$\operatorname{div}\left[\mathbb{C} : \left(\Delta\overline{\mathbf{F}} + \Delta\widetilde{\mathbf{F}}\right)\right] = \mathbf{0}$$

The weak form is given as

$$\int_{\text{cell}} \delta\widetilde{\mathbf{F}} : \mathbb{C} : \left(\Delta\overline{\mathbf{F}} + \Delta\widetilde{\mathbf{F}}\right)\, \mathrm{d}\mathbf{x} = 0 \tag{1.21}$$

Then we substitute all the quantities with discretized one and represent the equation in matrix formulation. Noting that $\mathbf{w}_h$ as the discretized fluctuation $\mathbf{L}$ as the matrix operator that transform the extended fluctuation into its extended strain and $\mathbf{K}$ as the stiffness matrix, then the second term in (1.20) is expressed as

$$\sum_{\text{cell}} \mathbf{C} \cdot \left(\mathbf{L} \cdot \frac{\mathbf{w}_h}{\Delta\overline{\mathbf{F}}}\right) \tag{1.22}$$

The vector fraction corresponds to the derivative. It is arranged in the same way as Jacobian matrix, where the $\mathbf{L}$ operator is applicable and let the dimension of the equation match with the dimension of tangent moduli matrix. The above (1.21) can be transformed with the matrix notation as

$$\mathbf{K} \cdot \mathbf{w}_h + \mathbf{L}^T \cdot \Delta\overline{\mathbf{F}} = \mathbf{0} \tag{1.23}$$

The overall expression of effective tangent moduli is [need to be considered again!!! there is an extra big C]

$$\mathbf{C}_{\text{eff}} = \mathbf{C}_{\text{avg}} - \mathbf{L}^T \cdot \mathbf{K}^{-1} \cdot \mathbf{L} \tag{1.24}$$

As for the implementation, the trick to obtain the corresponding matrices is to left or right multiply constant function. For averaged merged moduli, the following form is used

$$\mathbb{C}_{\text{avg}} = \int_{\text{cell}} \mathbf{F}_{\text{test\_const}} : \mathbb{C} : \mathbf{F}_{\text{trial\_const}}\, \mathrm{d}\mathbf{x}. \tag{1.25}$$

The corresponding code for this expression is

```
# Trial and test function to multiply with
F_const_trial = TrialFunction(self.strain_const_FS)
F_const_test = TestFunction(self.strain_const_FS)

# Derivate in the test and trial function direction, the same with left and right multiply
dPi_dF = derivative(self.Pi, self.F_merge, F_const_test)
ddPi_dF = derivative(dPi_dF, self.F_merge, F_const_trial)
# Integral over the unit cell
C_avg = assemble(ddPi_dF)
```

The same trick applies to the second term of the effective tangent moduli, where constant trial function is chosen for $\bar{\mathbf{F}}$. The implementation is as follows.

```
# Trial function
F_bar_trial = TrialFunction(self.strain_const_FS)
# F_w is the linear form generated from the fluctuation solving step
L2 = derivative(self.F_w, self.F_bar_merge, F_bar_trial)
B2 = assemble(L2)

# sensitivity method is just for efficient implementation of calculating K\L
LTKL2 = self.sensitivity(B2)
```

1.4 Documentation

This unit cell module contains a Python documentation in the original code. Docstrings are used in the implementation. The format of docstring is reStructuredText. A Sphinx documentation manual is generated for the module. Besides the usage of this module is clarified with examples and the unittest cases are also included for testing the new functionalities.

2 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.

2.1 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.

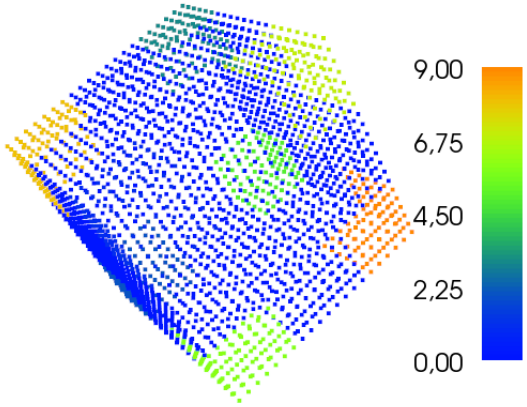

Figure 2.1: Typical free energy profile

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

2.2 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

$$\mathbb{C}_{\text{eff}} = \tag{2.1}$$

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

$$\psi\left(\mathbf{C}, \mathbf{E}\right) = \frac{1}{2}\mu_0\left(\text{tr}[\mathbf{C}] - 3\right) + \frac{\lambda}{4}\left(J^2 - 1\right) - \left(\frac{\lambda}{2} + \mu\right)\ln J - \frac{1}{2}\epsilon_0\left(1 + \frac{\chi}{J}\right) J\left[\mathbf{C}^{-1} : \left(\mathbf{E} \otimes \mathbf{E}\right)\right] \tag{2.2}$$

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]

2.3 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. The parameters and macro fields values are listed in the table below.

[fig: 3d uni]

[fig: 3d multi]

2.4 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)
```

2.5 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,

$$\mathbb{C}_{\text{eff}} = \frac{\partial \mathbf{P}_{\text{avg}}\left(\overline{\mathbf{F}}, \widetilde{\mathbf{F}}\right)}{\partial \overline{\mathbf{F}}} \;\Rightarrow\; \mathbb{C}_{\text{eff}} = \frac{\Delta \mathbf{P}_{\text{avg}}}{\Delta \overline{\mathbf{F}}}. \tag{2.3}$$

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.

3 Summary and Outlook

This "Forschungsmodul" concentrates on homogenization method for multi field modelling, where the novel finite element framework FEniCS is investigated. Some key points of homogenization and basic concepts in FEniCS are presented in this repport. A unified derivation of multi field problem is given in part, derivation and implementation. The strength of FEniCS is revealed for problems of multiple fields. It is also convinced that the modelling process in FEniCS is rather straightforward, as it uses the most mathematical language. Due to Python as interface language, the period of prototyping is shortened.

It is also shown that the current module is easy to extend for other problems, as the most classes are rather generic in the formulation. This will lead to difficulty, if one is not familiar with the method and considerations of the original code. In order to alleviate this drawback, docstrings explaining the codes are well preserved and a example manual will guide user better to understand the usage of the code.

As for improvement of the code, the most relevant one is to expand the functionalities of this unit cell module, which will adapt to other numerical scheme. One of such is FE^2. Much work could be done in this direction in implementation. The upper architecture of the problem is a macro scale problem, which will use the parameters calculated from the micro scale, i.e. homogenized problem. Since the parameters vary from element to element, the assembling might be a problem. If one want to make the best of the efficiency of FEniCS, this parameter data should be transformed into a Function object in FEniCS (or Expression object, it would be more efficient if subclassing of Expression is written in C++). This will add an extra layer of data communication. If this subclassing is achieved and linked to the unit cell module and succeed in handling parallelization nature of FEniCS. It would become a powerful tool to investigate composite behaviour in multiple fields. Another consideration is to design a new assembler that is appropriate for macro scale problem. The new assembler is based on the old one. One must then delve into the detailed implementation written in C++. This method is not generic, as with other type of PDEs or many parameters this method can not be applied. Another improvement of the current work might be plasticity or viscosity material model. For plasticity problem, a optimization problem could be formulated in micro scale and accomplish the calculation. For viscosity the time dependency needs to be included in the implementation.