Homogenization for Multi Field Modelling

Part II: Implementation and Numerical Examples

Yi Hu

April 17, 2016

Contents

In the first part of this report the underlying theory and the FEM framework are introduced, which constitute the basis of our simulation. The current part focus on the implementation and numerical results. A unit cell module is established in FEniCS framework using Python language. A unit cell is a Representative Volume Element (RVE) with its edge length as one. We use this term in order to be consistent with implementation. In the current implementation the unit cell is a unit square in two dimensional case and a unit cube for three dimensional. The most functionalities of homogenization in composite material modelling are realized in this module.

1 Introduction

The approach in this unit cell module is motivated by the work in the group. The micro-macro coupling part of this module originates from [8], where the homogenization of large strain deformation is developed. As for the implementation of computational homogenization in FE^2 scheme, matrix representation and algorithms for small strain problems are presented in detail in [6]. Various kinds of boundary condition are also taken into account in [6]. The reformulation of the homogenization problem into a optimization context is introduced in [4][5] [7], which extends the application of computational homogenization in material modelling. It is a great unification of the material modelling, since various kinds of materials could fit in this scheme, e.g. plasticity material where Karush-Kuhn-Tucker condition in plasticity modelling is also a optimization procedure.

Homogenization of composite material in multiple fields can be achieved. Previous work has been done by a lot of researchers. The method used in the current work is inspired by [2] and [13]. In the case of multiple fields modelling the formulation becomes more intricate. An energy approach will lead to a simpler representation of the problem and unravel the basic characteristics of material properties. The formulation in [9] is a good example of this kind. The Legendre transformation performed in the literature provides a good feature of the energy based approach. For more recent work about algorithms in the magneto-electro-mechanically coupled problem we refer to [11] and [10]. The latter also discusses the stability analysis of coupled problem in fully detail, which is an important issue in multi field modelling and homogenization [1].

As for the implementation part the FEniCS book [14] gives extensive instructions about the functionalities of FEniCS. A unit cell module is implemented on this basis. To test the module several numerical examples are simulated. These examples start from a basic hyperelastic composite material with circle inclusions to three dimensional coupled field problems.

This part of the report is arranged in the following order. First we developed the computational homogenization especially for coupled field modelling. This is followed by the ideas behind implementation and introduction about module and its functionalities. Then a section about numerical examples is provided. The report closes with a summary concluding all the work that has been done in this “Forschungsmodul” and the ongoing improvement in the future.

2 Derivation and Implementation

In this chapter composite materials modelling in coupled fields is investigated. First we extend computational homogenization of composite material to coupled field problems, which involves the calculation of fluctuation of coupled fields, averaged quantities and effective tangent moduli. Theses tasks are done by the current unit cell module. In the module energy based formulation is exploited. In this context the material model is easy to generate and the equilibrium is also easy to establish. The considered materials are Neo-Hookean type materials (may contain magnetic or electrical energy) and the composites containing periodic micro structures.

Details of the problem setting and derivation are presented in the following two sections. Then some issues in implementation are shortly discussed. The source code and user documentation are clarified in the end of this chapter.

2.1 Problem Setting

The discussion focuses on unit cells consisting of multiple materials. A unit cell can be two dimensional or three dimensional, usually a square or a cube with edge length one. Each material component occupies a range of domain in the cell, which are inclusions. All the inclusions possess no intersections in the domain. Multiple fields can be taken into account, such as mechanical field, temperature field and electrical field. Material types are hyperelastic material, thermomechanical Simo-Pister material (introduced in the course Elements of Non-linear Continuum Thermodynamics) and Neo-Hookean electroactive material. The energy function of electroactive material is extracted from the literature [2]. All the materials in the current material library share the common feature, i.e. no dissipative. Plasticity and viscosity are beyond the scope of the current work.

The energy functions used are summarized below.

Saint Venant-Kirchhoff Material [3]

$$\psi\left(\mathbf{E}\right) = \frac{\lambda}{2}\left[\mathrm{tr}(\mathbf{E})\right]^2 + \mu \mathrm{tr}\left(\mathbf{E}^2\right), \tag{2.1}$$

where $\mathbf{E}$ is Green Lagrange Tensor.

Simo Pister Material [15]

$$\psi\left(\theta, \mathbf{C}\right) = \frac{1}{2}\mu_0\left(I_C - 3\right) + \frac{1}{2}\lambda_0\left[\ln\left(\det\mathbf{C}\right)^{\frac{1}{2}}\right]^2 + \left(m_0\Delta\theta\mu_0\right)\ln(\det\mathbf{C})^{\frac{1}{2}} - \rho_0 c_V\left(\theta\ln\frac{\theta}{\theta_0} - \Delta\theta\right), \tag{2.2}$$

where $\mathbf{C}$ is right Cauchy-Green tensor, θ is temperature.

Neo-Hookean Type Electroactive Material [2]

$$\psi\left(\mathbf{C}, \mathbf{E}\right) = \frac{1}{2}\mu_0\left(\mathrm{tr}[\mathbf{C}] - 3\right) + \frac{\lambda}{4}\left(J^2 - 1\right) - \left(\frac{\lambda}{2} + \mu\right)\ln J - \frac{1}{2}\epsilon_0\left(1 + \frac{\chi}{J}\right)J\left[\mathbf{C}^{-1} : \left(\mathbf{E}\otimes\mathbf{E}\right)\right], \tag{2.3}$$

where $\mathbf{C}$ is right Cauchy-Green tensor, $\mathbf{E}$ is electric field intensity.

2.2 Derivation

In this section the derivation concerning the unit cell problem as well as a scheme for the calculation of the effective tangent moduli are presented. We first formulate the total energy of composite materials, then derive equilibrium from the obtained total energy. Solving the equilibrium will give fluctuation in the micro scale. At last calculation of effective quantities is performed, which will be required for the macro scale problem.

Motivated by the ideas from [10], a compact representation of field variables is presented. We denote $\mathbf{w}$ as an 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 an extended strain tensor measure here with notation $\mathbf{F}$. The extended stress can be defined in the same manner. It should be noticed that we use a slightly different

notation from the one introduced in the literature [10] (namely without star as superscript), which is for the purpose of consistency with the implementation. We performed an additive split in the extended displacement and the extended deformation. These two parts are respectively the averaged quantity from macro scale, $\overline{(\cdot)}$, and the fluctuation, $\widetilde{(\cdot)}$ from micro scale, which is to be solved. This relation is stated as

$$\mathbf{w} = \overline{\mathbf{w}} + \widetilde{\mathbf{w}}, \ \mathbf{F} = \overline{\mathbf{F}} + \widetilde{\mathbf{F}}. \tag{2.4}$$

The macro and micro deformation decomposition can be viewed in the following figure,

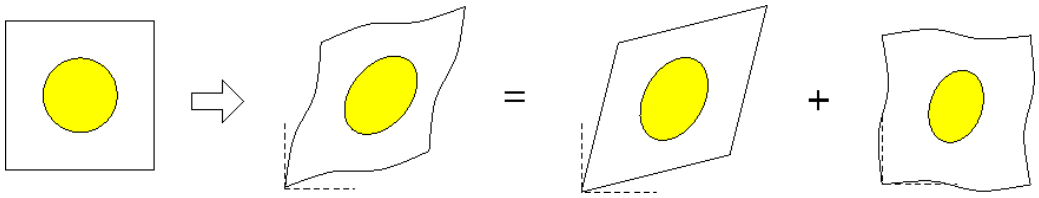


Figure 2.1: macro and micro deformation decomposition

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. 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.

2.2.1 Total Energy

For composites total energy is the sum of energy from every component plus their interface energy (not accounted in the current formulation). There are two different representations of the total energy. One is used to derive 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 the extended strain is given as (here viscosity and plasticity not accounted)

$$\Pi(\mathbf{F}) = \hat{\Pi}(\overline{\mathbf{F}}, \widetilde{\mathbf{F}}) = \sum_{i=1}^{n} \int \psi_i(\mathbf{F}) \, \mathrm{d}\mathbf{x}_i. \tag{2.5}$$

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

$$\Pi(\mathbf{F}(\mathbf{w})) = \Pi(\mathbf{F}(\overline{\mathbf{w}}, \widetilde{\mathbf{w}})) = \sum_{i=1}^{n} \int \psi_i(\mathbf{F}(\mathbf{w})) \, \mathrm{d}\mathbf{x}_i. \tag{2.6}$$

2.2.2 Equilibrium

The equilibrium of the system in a unit cell is built around the stationary point of its total energy with respect to the fluctuation,

$$\frac{\partial \Pi(\mathbf{F}(\mathbf{w}))}{\partial \widetilde{\mathbf{w}}} = \mathbf{0}. \tag{2.7}$$

Above might be a non-linear equation. It is assumed that Newton method is used (other non-linear solvers are also possible). Therefore the Jacobian of this equation is needed. Denoting $\widetilde{\mathbf{v}}$ as the test function for the fluctuation, $\widetilde{\mathbf{w}}_{\mathrm{inc}}$ as the incremental trial function in a Newton step, the weak form of (2.7) and Jacobian can be expressed as

$$L = \frac{\partial \Pi(\mathbf{F}(\mathbf{w}))}{\partial \widetilde{\mathbf{w}}} \cdot \widetilde{\mathbf{v}}, \ a = \frac{\partial L}{\partial \widetilde{\mathbf{w}}} \cdot \widetilde{\mathbf{w}}_{\mathrm{inc}} \tag{2.8}$$

The whole linear equation system can be obtained through assembling. From above L is the linear form and a is the bilinear form. Weak formulation of (2.7) is to search $\widetilde{\mathbf{w}}_{\mathrm{inc}}$ in V, such that the following holds,

$$a(\widetilde{\mathbf{w}}_{\mathrm{inc}}, \widetilde{\mathbf{v}}) = L(\widetilde{\mathbf{v}}), \ \forall \widetilde{\mathbf{v}} \in \hat{V} \tag{2.9}$$

After solving this equation the fluctuation increment is obtained.

2.2.3 Post-processing

When it comes to post-processing, the extended strain is calculated first and the total energy of the material is expressed with (2.5). Noticing (2.4)

$$\mathbf{F} = \overline{\mathbf{F}} + \widetilde{\mathbf{F}}(\widetilde{\mathbf{w}}). \tag{2.10}$$

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 generalized stress for the extended stain is the derivative of local energy, namely

$$\mathbf{P}_i = \frac{\partial \psi_i(\mathbf{F})}{\partial \mathbf{F}}. \tag{2.11}$$

Then the averaged generalized stress is the average of the above formula. When the unit cell has edges with length one, the following integral over the domain is the averaged quantity,

$$\mathbf{P}_{\text{avg}} = \frac{1}{V} \sum_{i=1}^{n} \int \mathbf{P}_i \, \mathrm{d}\mathbf{x}_i. \tag{2.12}$$

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

$$\mathbf{P}_{\text{avg}} = \frac{\partial \frac{1}{V}\Pi(\mathbf{F})}{\partial \mathbf{F}} = \frac{\partial}{\partial \mathbf{F}} \left(\frac{1}{V} \sum_{i=1}^{n} \int \psi_i(\mathbf{F}) \, \mathrm{d}\mathbf{x}_i \right). \tag{2.13}$$

Comparing (2.13) and (2.12) the following must hold,

$$\frac{\partial}{\partial \mathbf{F}} \left(\sum_{i=1}^{n} \int \psi_i(\mathbf{F}) \, \mathrm{d}\mathbf{x}_i \right) = \sum_{i=1}^{n} \int \frac{\partial \psi_i(\mathbf{F})}{\partial \mathbf{F}} \, \mathrm{d}\mathbf{x}_i. \tag{2.14}$$

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,

$$\frac{\partial}{\partial \mathbf{F}} \left(\int \psi_i(\mathbf{F}) \, \mathrm{d}\mathbf{x}_i \right) = \int \frac{\partial \psi_i(\mathbf{F})}{\partial \mathbf{F}} \, \mathrm{d}\mathbf{x}_i. \tag{2.15}$$

Numerical experiments show that this equality holds. Moreover in this formulation the integral is calculated in the reference configuration, while the extended strain is in current configuration. This implies the integral and differentiation with respect to the extended strain is independent hence interchangeable. 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.

$$\mathbb{C}_{\text{avg}} = \frac{1}{V} \sum_{i=1}^{n} \int \mathbb{C}_i \, \mathrm{d}\mathbf{x}_i = \frac{1}{V} \sum_{i=1}^{n} \int \frac{\partial^2 \psi_i(\mathbf{F})}{\partial \mathbf{F}^2} \, \mathrm{d}\mathbf{x}_i. \tag{2.16}$$

$$\mathbb{C}_{\text{avg}} = \frac{\partial^2 \frac{1}{V}\Pi(\mathbf{F})}{\partial \mathbf{F}^2} = \frac{\partial^2}{\partial \mathbf{F}^2} \left(\frac{1}{V} \sum_{i=1}^{n} \int \psi_i(\mathbf{F}) \, \mathrm{d}\mathbf{x}_i \right) \tag{2.17}$$

The 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

$$\mathbf{P}_{\text{avg}} = \mathbf{P}_{\text{avg}} \left(\overline{\mathbf{F}}, \widetilde{\mathbf{F}} \right). \tag{2.18}$$

Hence the effective moduli is as follows,

$$\mathbb{C}_{\text{eff}} = \frac{\partial \mathbf{P}_{\text{avg}} \left(\overline{\mathbf{F}}, \widetilde{\mathbf{F}} \right)}{\partial \overline{\mathbf{F}}} \tag{2.19}$$

Recalling (2.13), the following formula holds, (using $\mathbf{F} = \overline{\mathbf{F}} + \widetilde{\mathbf{F}}$ and chain rule)

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

$$\xrightarrow[V=1]{\text{unit cell}} \mathbb{C}_{\text{eff}} = \mathbb{C}_{\text{avg}} + \int_{\text{cell}} \mathbb{C} : \frac{\partial \widetilde{\mathbf{F}}}{\partial \overline{\mathbf{F}}} \, \mathrm{d}\mathbf{x} \tag{2.20}$$

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

2.3 Details 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 an easy manner. `cell_geom.py` specifies the inclusions of the material as well as the periodic mapping. Meshes can be imported or generated within FEniCS. 3D and 2D unit cells are available. An inclusion could be circle in 2D or sphere in 3D. Rectangular inclusion and brick inclusion are also realized in the current state. Boundaries and corners of unit cells can be marked for imposing Neumann boundary condition. The main part of this module is `cell_computation.py`, pre-processing, formulation and solving, and post-processing are contained in this file.

2.3.1 cell_material.py

In this file class and functions for material definition are involved. The main part of this file is the `class Material`. The definition of material starts with its energy function. This energy function receives invariants as the function arguments. Then one can instantiate a new material with an energy function and its parameters. Next one defines 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 we 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])
```

2.3.2 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 condition of the problem. 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)

# 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. For the two dimensional case we refer to the example in the FEniCS forum, while the three dimensional case needs more investigation of its internal mapping definition. For 2D case, the edges on coordinate axis are marked as reference edges, and other edges are then mapped onto these two edges. For 3D case, the main facets are marked, which are formed 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. Detailed realization of 3D case can be viewed in the module. 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)
```

2.3.3 cell_computation.py

This is the main part of the unit cell module. Pre-processing contains merging functions and splitting functions, generate the 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 concentrates the calculation of extended strain, generalized stress, averaged extended strain, averaged generalized stress, averaged moduli, and effective tangent moduli. Plotting the result is also realized in this file.

In the pre-processing step, using `field_merge()`, `field_split()` or `set_field()` makes the multi field modelling easy to handle. 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. We refer the work in [12]. 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

$$\int_{\text{cell}} \delta\widetilde{\mathbf{F}} : \mathbb{C} : \left(\Delta\overline{\mathbf{F}} + \Delta\widetilde{\mathbf{F}}\right) \, \mathrm{d}\mathbf{x} = 0 \tag{2.21}$$

Then we substitute all the quantities with discretized one and rewrite the equation in matrix. Noting that $\mathbf{w}_h$ as the discretized fluctuation, $\mathbf{L}$ as the matrix operator that transform the extended fluctuation into its extended

stress, and $\mathbf{K}$ as the stiffness matrix, then the second term in (2.20) is expressed as

$$\mathbf{L}^T \cdot \frac{\Delta \mathbf{w}_h}{\Delta \overline{\mathbf{F}}} \tag{2.22}$$

The vector fraction corresponds to the derivative. This fraction 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 (2.21) can be transformed with the matrix notation as

$$\mathbf{K} \cdot \Delta \mathbf{w}_h + \mathbf{L} \cdot \Delta \overline{\mathbf{F}} = \mathbf{0} \tag{2.23}$$

The overall expression of effective tangent moduli is then

$$\mathbf{C}_{\text{eff}} = \mathbf{C}_{\text{avg}} - \mathbf{L}^T \cdot \mathbf{K}^{-1} \cdot \mathbf{L} \tag{2.24}$$

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

$$\mathbb{C}_{\text{avg}} = \int_{\text{cell}} \delta \mathbf{F}_{\text{test\_const}} : \mathbb{C} : \Delta \mathbf{F}_{\text{trial\_const}} \, \mathrm{d}\mathbf{x}. \tag{2.25}$$

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 $\overline{\mathbf{F}}$. 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)
```

2.4 Documentation

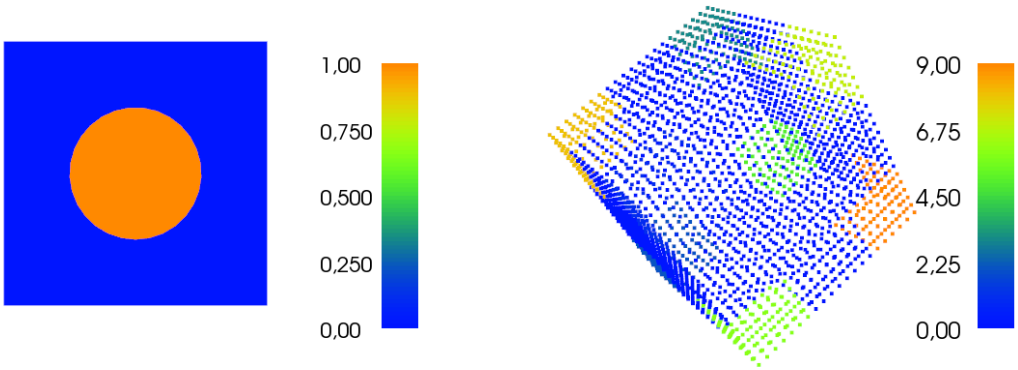
This unit cell module contains a Python documentation in the original code. Docstrings are used in the implementation. The format of docstrings is reStructuredText. A Sphinx documentation manual is generated for the module. Besides, the usage of this module is clarified with examples and the unittest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

3 Numerical Examples

In this chapter 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.

3.1 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.



(a) 2D Circle Inclusion

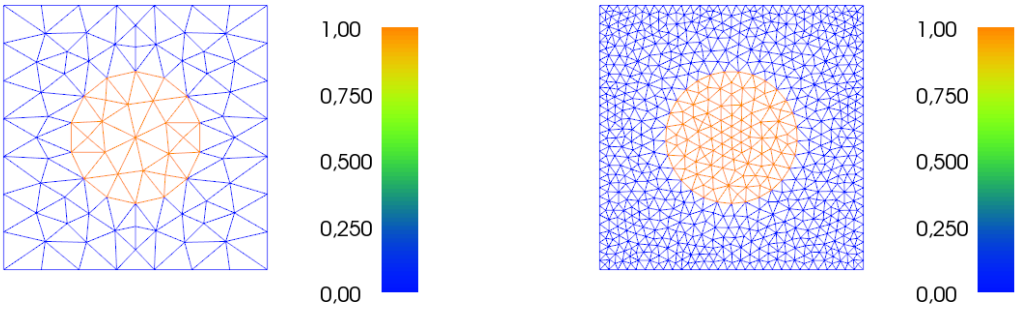
(b) 3D Unit Cube with Inclusion at Corners

Figure 3.1: Various Inclusion Types

3.2 2D Modelling

3.2.1 Uni-Field 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 circle located at the center of this square with radius 0.25.



(a) Coarse Mesh

(b) Fine Mesh

Figure 3.2: Geometry and Mesh

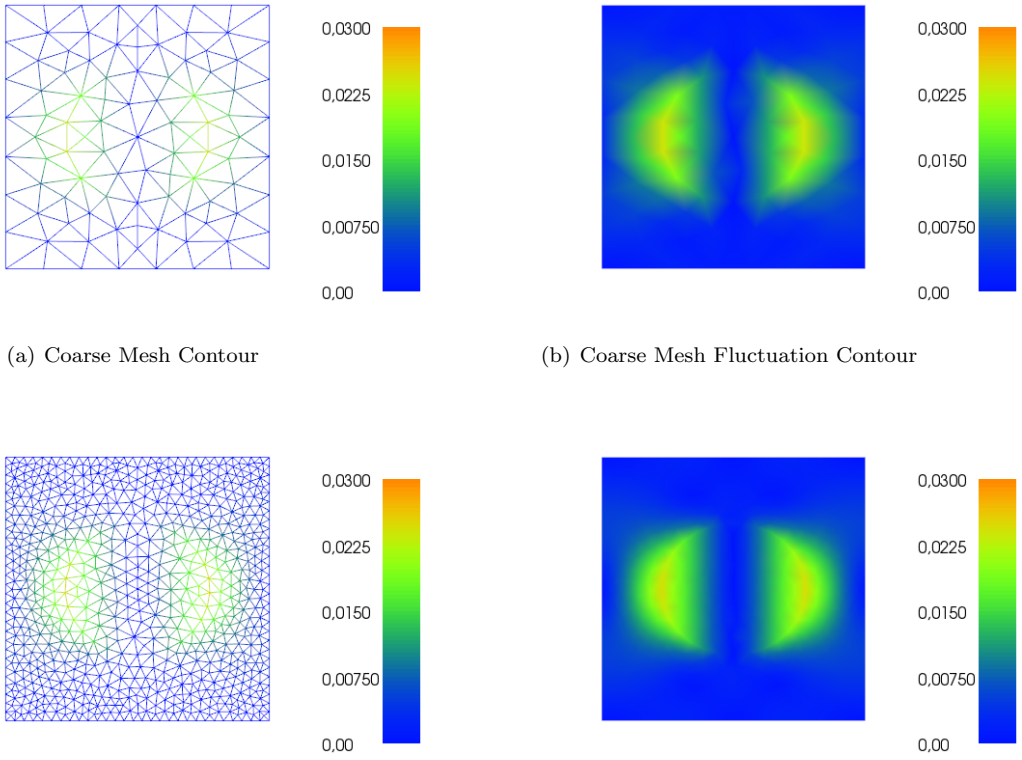
Table 3.1: Material Parameters

	Young's Modulus	Poisson Ratio
Matrix Material	10.0	0.3
Inclusion Material	1000.0	0.3

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-linear nature of the problem, the load cannot be set too high. For large macro deformation input a quasi static loading process needs to be imposed. The fluctuation plot is demonstrated here.



(a) Coarse Mesh Contour

(b) Coarse Mesh Fluctuation Contour

(c) Fine Mesh Contour

(d) Fine Mesh Fluctuation Contour

Figure 3.3: Uniaxial Compression Test: Fluctuation Result

If shear macro strain is defined, namely

$$\bar{\mathbf{F}} = \begin{bmatrix} 1. & 0.5 \\ 0 & 1 \end{bmatrix},$$

the result will look like

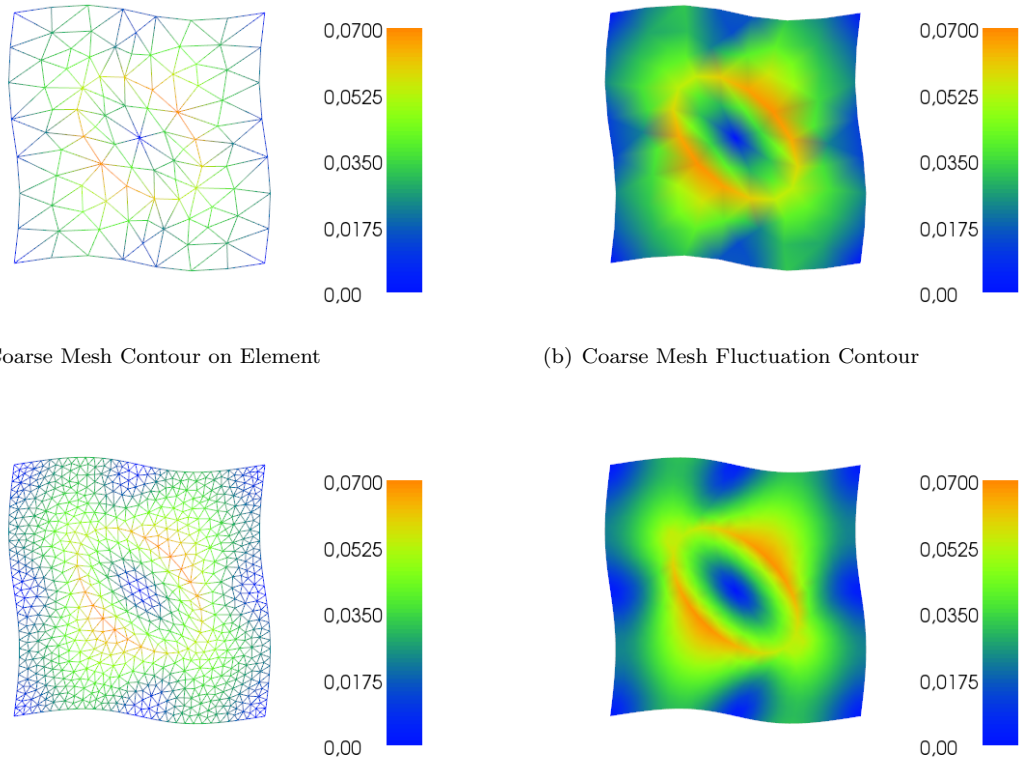
The calculated homogenized tangent moduli (from fine mesh) for uniaxial compression load case is

$$\mathbb{C}_{\text{eff}} = \begin{bmatrix} 1.05215604 \times 10^{+01} & 6.81100019 \times 10^{-04} & 8.10922941 \times 10^{-04} & 6.0931974 \times 10^{+00} \\ 6.81100019 \times 10^{-04} & 3.03957695 \times 10^{+00} & 4.12022593 \times 10^{+00} & -2.43801203 \times 10^{-04} \\ 8.10922941 \times 10^{-04} & 4.12022593 \times 10^{+00} & 2.98311033 \times 10^{+00} & -3.19622254 \times 10^{-04} \\ 6.09319740 \times 10^{+00} & -2.43801203 \times 10^{-04} & -3.19622254 \times 10^{-04} & 1.74423266 \times 10^{+01} \end{bmatrix}, \quad (3.1)$$

while for simple shear load case is

$$\mathbb{C}_{\text{eff}} = \begin{bmatrix} 22.27828078 & 11.45747779 & 4.84645322 & 7.52308193 \\ 11.45747779 & 15.71048687 & 6.80879894 & 11.93754869 \\ 4.84645322 & 6.80879894 & 7.35347903 & 4.53857874 \\ 7.52308193 & 11.93754869 & 4.53857874 & 23.42982073 \end{bmatrix}, \quad (3.2)$$

Similar fluctuation contour shows that the calculation is consistent between fine and coarse mesh. It is also shown that the interface between inclusion and matrix material is a critical part.



(a) Coarse Mesh Contour on Element

(b) Coarse Mesh Fluctuation Contour

(c) Fine Mesh Contour on Element

(d) Fine Mesh Fluctuation Contour

Figure 3.4: Simple Shear Test: Fluctuation Result

3.2.2 Multi-Field Modelling

As for multi-field modelling, the material in consideration is Neo-Hookean type electroactive polymer material,

$$\psi\left(\mathbf{C},\mathbf{E}\right)=\frac{1}{2}\mu_0\left(\mathrm{tr}[\mathbf{C}]-3\right)+\frac{\lambda}{4}\left(J^2-1\right)-\left(\frac{\lambda}{2}+\mu\right)\ln J-\frac{1}{2}\epsilon_0\left(1+\frac{\chi}{J}\right)J\left[\mathbf{C}^{-1}:\left(\mathbf{E}\otimes\mathbf{E}\right)\right]. \tag{3.3}$$

Macro field input is

$$\overline{\mathbf{F}}=\begin{bmatrix}0.9 & 0\\ 0 & 1\end{bmatrix},\overline{\mathbf{E}}=\begin{bmatrix}0.\\ 0.2\end{bmatrix}$$

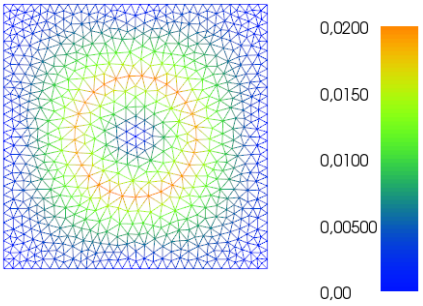
Material input is in the following table,

Table 3.2: Material Parameters

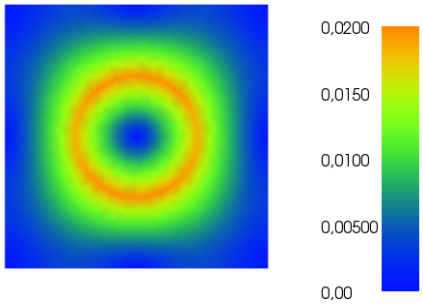
	Young's Modulus	Poisson Ratio	Electric Susceptibility
Matrix Material	2×10^{5}	0.4	7
Inclusion Material	2×10^{8}	0.3	7000

The fluctuation result is illustrated below (only the fine mesh is given). And the effective tangent moduli is

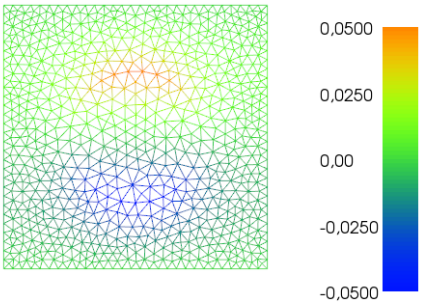
$$\begin{bmatrix}
1.26\times10^{+06} & -1.68\times10^{+00} & -2.02\times10^{+00} & 1.05\times10^{+06} & -1.57\times10^{+00} & -1.31\times10^{+04}\\
-1.68\times10^{+00} & 3.23\times10^{+05} & 4.41\times10^{+03} & -1.13\times10^{+00} & -1.70\times10^{+04} & 1.95\times10^{+00}\\
-2.02\times10^{+00} & 4.41\times10^{+03} & 3.27\times10^{+05} & -1.12\times10^{+00} & -1.88\times10^{+04} & 2.17\times10^{+00}\\
1.05\times10^{+06} & -1.13\times10^{+00} & -1.12\times10^{+00} & 1.51\times10^{+06} & 2.25\times10^{+00} & -4.69\times10^{+04}\\
-1.57\times10^{+00} & -1.70\times10^{+04} & -1.88\times10^{+04} & 2.25\times10^{+00} & 1.16\times10^{+05} & 3.48\times10^{+00}\\
-1.31\times10^{+04} & 1.95\times10^{+00} & 2.17\times10^{+00} & -4.69\times10^{+04} & 3.48\times10^{+00} & 9.72\times10^{+04}
\end{bmatrix}.$$



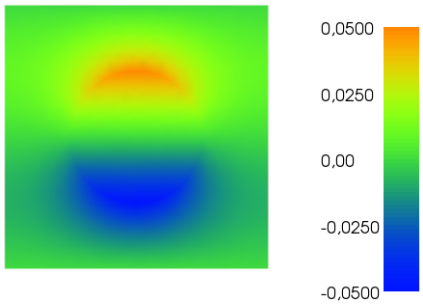
(a) Fluctuation of Displacement on Element Edges



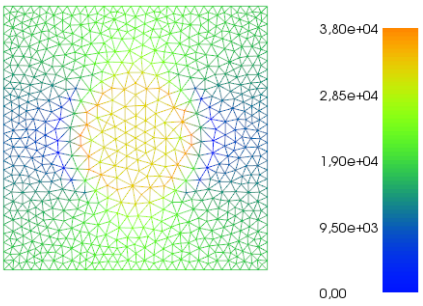
(b) Fluctuation of Displacement



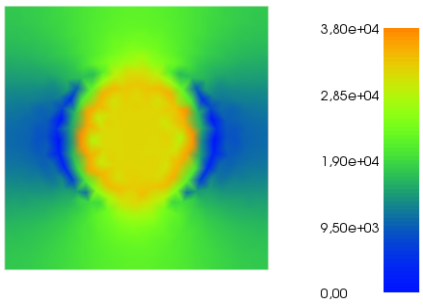
(c) Fluctuation of Electrical Potential on Element Edges



(d) Fluctuation of Electrical Potential



(e) Electrical Field Intensity in Y Direction on Element Edges

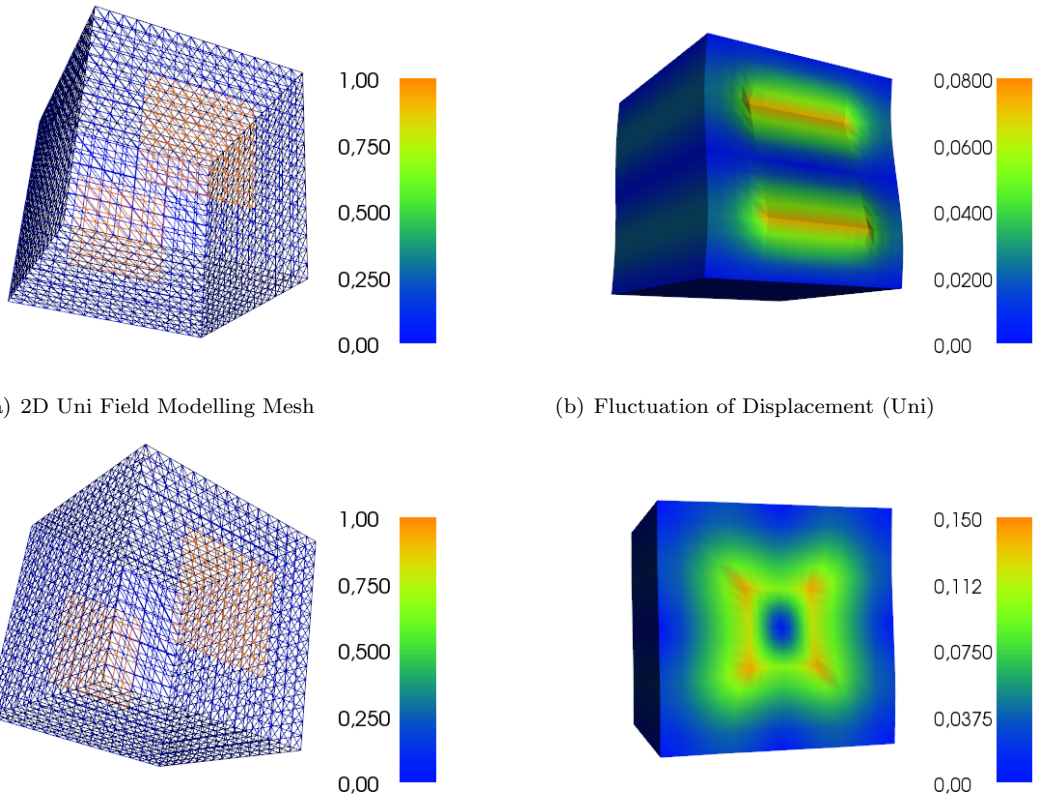


(f) Electrical Field Intensity in Y Direction

Figure 3.5: Multi Field Modelling: Electroactive Polymer in Mechanical Field and Electrical Field

3.3 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. Material parameters are set the same as in 2D cases, while the macro field input for uni-field and multi-field are as follows.



(a) 2D Uni Field Modelling Mesh

(b) Fluctuation of Displacement (Uni)

(c) 3D Multi Field Modelling Mesh

(d) Fluctuation of Displacement (Multi)

Figure 3.6: 3D Modelling in Uni Field and Multi Field Case

In multi-field calculation "SNES" solver in "PETSc" is chosen. The solver parameters are tuned similar as in the 3D examples from FEniCS. In 3D case an iterative solver for the linear equation system is a necessity, otherwise the calculation would take too much time. The potential of 3D modelling in both uni-field and multi-field cases are obvious. The effective tangent moduli for uni field in 3D is given here, which has 81 entries as its fourth rank tensor in 3D. For multi-field problem the homogenized tangent moduli can also be calculated in this scheme. It ends with 12 by 12 huge matrix, where a relative coarser mesh is used.

$$\begin{bmatrix}
1\cdot10^{+02} & 7\cdot10^{+00} & -1\cdot10^{-02} & 3\cdot10^{+00} & 7\cdot10^{+00} & -5\cdot10^{-03} & -1\cdot10^{-02} & -5\cdot10^{-03} & 7\cdot10^{+00} \\
7\cdot10^{+00} & 1\cdot10^{+01} & -5\cdot10^{-02} & 1\cdot10^{+01} & 1\cdot10^{+01} & -3\cdot10^{-02} & -4\cdot10^{-02} & -3\cdot10^{-02} & 3\cdot10^{+00} \\
-1\cdot10^{-02} & -5\cdot10^{-02} & 6\cdot10^{+00} & -3\cdot10^{-02} & -4\cdot10^{-02} & 2\cdot10^{+00} & 6\cdot10^{+00} & 2\cdot10^{+00} & -8\cdot10^{-03} \\
3\cdot10^{+00} & 1\cdot10^{+01} & -3\cdot10^{-02} & -1\cdot10^{+01} & 7\cdot10^{+00} & -3\cdot10^{-02} & -3\cdot10^{-02} & -2\cdot10^{-02} & 1\cdot10^{+00} \\
7\cdot10^{+00} & 1\cdot10^{+01} & -4\cdot10^{-02} & 7\cdot10^{+00} & 2\cdot10^{+01} & -3\cdot10^{-02} & -3\cdot10^{-02} & -2\cdot10^{-02} & 6\cdot10^{+00} \\
-5\cdot10^{-03} & -3\cdot10^{-02} & 2\cdot10^{+00} & -3\cdot10^{-02} & -3\cdot10^{-02} & 5\cdot10^{+00} & 4\cdot10^{-01} & 5\cdot10^{+00} & 1\cdot10^{-02} \\
-1\cdot10^{-02} & -4\cdot10^{-02} & 6\cdot10^{+00} & -3\cdot10^{-02} & -3\cdot10^{-02} & 4\cdot10^{-01} & -1\cdot10^{+01} & 2\cdot10^{+00} & -1\cdot10^{-02} \\
-5\cdot10^{-03} & -3\cdot10^{-02} & 2\cdot10^{+00} & -2\cdot10^{-02} & -2\cdot10^{-02} & 5\cdot10^{+00} & 2\cdot10^{+00} & 6\cdot10^{+00} & 7\cdot10^{-03} \\
7\cdot10^{+00} & 3\cdot10^{+00} & -8\cdot10^{-03} & 1\cdot10^{+00} & 6\cdot10^{+00} & 1\cdot10^{-02} & -1\cdot10^{-02} & 7\cdot10^{-03} & 2\cdot10^{+01}
\end{bmatrix}.$$

3.4 Simulation Template

In this section a simulation template is provided. This template introduces the commonly used part 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`. Once the setting of the problem is completed, micro scale calculation is

initiated with defining an object of `MicroComputation` from `cell_computation.py`. After inputting macro field functions, micro fluctuation can be calculated using the member method `comp_fluctuation()` with 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. Moreover 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)
```

3.5 Consistency with Finite Difference Calculation

The convergence of the homogenized effective tangent moduli is investigated with Finite Difference calculation. Here Finite Difference calculation stands for calculating the ratio between the change of the averaged extended

stress with respect to small change of the macro extended strain input. This method leads to the following expression,

$$\mathbb{C}_{\text{eff}} = \frac{\partial \mathbf{P}_{\text{avg}}\left(\bar{\mathbf{F}}, \widetilde{\mathbf{F}}\right)}{\partial \bar{\mathbf{F}}} \Rightarrow \mathbb{C}_{\text{eff}} = \frac{\Delta \mathbf{P}_{\text{avg}}}{\Delta \bar{\mathbf{F}}}. \tag{3.4}$$

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

The comparison is given at the end of this demonstration. 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 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.

4 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 report. A unified derivation of multi-field problem is given in the previous chapters. The strength of FEniCS is revealed for problems of multiple fields. It is also demonstrated that the modelling process in FEniCS is rather straightforward, as it embeds mathematical notations in its framework. The time of prototyping is further shortened, which is benefited from the features of Python.

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 methods and notations in the original code. In order to alleviate this drawback, docstrings explaining the code are well preserved and an example manual will guide users to better 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 adapts to other numerical scheme. One of such is FE^2. Much work can 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. the homogenized problem. Since the parameters vary from element to element, the assembling might be a problem. If one wants 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 for other type of PDEs with even more than one homogenized parameters this method can not be applied. Another improvement of the current work might be plasticity or viscosity material model. For plasticity problem, an optimization problem could be formulated in micro scale and accomplish the calculation. For viscosity the time dependency needs to be included in the implementation. Additionally for more detailed simulation the interface between different materials should be accounted.

Bibliography

[1] M. G. D. Geers, V. G. Kouznetsova, and W. A. M. Brekelmans. Multi-scale computational homogenization: Trends and challenges. *Journal of Computational and Applied Mathematics*, 234(7):2175–2182, August 2010.

[2] Marc-André Keip, Paul Steinmann, and Jörg Schröder. Two-scale computational homogenization of electro-elasticity at finite strains. *Computer Methods in Applied Mechanics and Engineering*, 278:62–79, August 2014.

[3] Jerrold E. Marsden and Thomas JR Hughes. *Mathematical foundations of elasticity*. Courier Corporation, 1994. 02755.

[4] C. Miehe. Strain-driven homogenization of inelastic microstructures and composites based on an incremental variational formulation. *International Journal for Numerical Methods in Engineering*, 55(11):1285–1322, December 2002.

[5] C. Miehe. Computational micro-to-macro transitions for discretized micro-structures of heterogeneous materials at finite strains based on the minimization of averaged incremental energy. *Computer Methods in Applied Mechanics and Engineering*, 192(5-6):559–591, 2003.

[6] C. Miehe and A. Koch. Computational micro-to-macro transitions of discretized microstructures undergoing small strains. *Archive of Applied Mechanics*, 72(4-5):300–317, July 2002.

[7] C. Miehe, J. Schotte, and M. Lambrecht. Homogenization of inelastic solid materials at finite strains based on incremental minimization principles. Application to the texture analysis of polycrystals. *Journal of the Mechanics and Physics of Solids*, 50(10):2123–2167, October 2002.

[8] C. Miehe, J. Schotte, and J. Schroder. Computational micro-macro transitions and overall moduli in the analysis of polycrystals at Large strains. *Computational Materials Science*, 16(1-4):372–382, December 1999.

[9] Christian Miehe, Björn Kiefer, and Daniele Rosato. An incremental variational formulation of dissipative magnetostriction at the macroscopic continuum level. *International Journal of Solids and Structures*, 48(13):1846–1866, June 2011.

[10] Christian Miehe, Daniel Vallicotti, and Stephan Teichtmeister. Homogenization and multiscale stability analysis in finite magneto-electro-elasticity. Application to soft matter EE, ME and MEE composites. *Computer Methods in Applied Mechanics and Engineering*, 300:294–346, March 2016.

[11] Joerg Schroeder, Matthias Labusch, and Marc-Andre Keip. Algorithmic two-scale transition for magneto-electro-mechanically coupled problems FE2-scheme: Localization and homogenization. *Computer Methods in Applied Mechanics and Engineering*, 302:253–280, April 2016.

[12] J. Schröder and M.-A. Keip. Two-scale homogenization of electromechanically coupled boundary value problems – consistent linearization and applications. *Computational Mechanics*, 50(2):229–244, 2012.

[13] Jörg Schröder and Marc-André Keip. Two-scale homogenization of electromechanically coupled boundary value problems. *Computational Mechanics*, 50(2):229–244, May 2012.

[14] Garth Wells, Kent-Andre Mardal, and Anders Logg. *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*. Springer, 2012.

[15] P. Wriggers, C. Miehe, M. Kleiber, and J. C. Simo. On the coupled thermomechanical treatment of necking problems via finite element methods. *International Journal for Numerical Methods in Engineering*, 33(4):869–883, February 1992.