Homogenization for Multi Field Modelling

Part I: Theories and FEniCS

Yi Hu

February 24, 2016

Contents

In this student project, Homogenization Method is utilized to investigate behaviour of composites under multiple fields. Many applications can be found in the field of Material Modelling, specifically for coupled problems of composites, e.g. Elastroactive Polymers. A novel Finite Element framework, FEniCS, is employed to achieve this goal. FEniCS is a collection of libraries and modules and uses Python (or C++) as interface language. Regarded as a dynamical language, Python stands out as a fast language for prototyping. Another important feature about FEniCS is its specialization in code generation with respect to bilinear forms, linear forms and function spaces in the formulation of computational problems, which translates directly the mathematical language into codes and accelerates the trial process of new models and new computation methods. With this powers, a Unit Cell module is realized, where the calculation (include homogenized properties) of composites in micro scale could be performed.

1 Introduction

Composites play an important role in engineering, as they combine the advantages of each components in materials. In order to model composite materials, different materials and different material configurations need to be considered. There are several methods to achieve this modelling. Intuitively a full simulation can be realized, where micro structures and different materials would be represented explicitly in the simulated object. This results in a tremendous many degrees of freedom consuming not only large amount of memory but also the computation resources. Hence a full simulation accounting for all the micro structures and inclusions is not the most efficient way.

A multi scale model is conducive in modelling [search], where the deformation in small scale will be reflected in the corresponding macro scale. Moreover scale separation would benefit from various computing techniques such as parrellelization and model reduction[ref]. These techniques would possibly boost the efficiency of computation.

There are several approaches tackling the multi scale problem [search and ref].
In this work Homogenization Method is used.

2 Homogenization Method

The method used in the current work is Homogenization Method. It was proposed in 1970s by Babuska and collaborators, [ref: abdulle]. The main purpose of this method is to make use of the scale separation, in order to obtain a reduced PDE for the macroscopic problem. The macroscopic problem always contains the information from the according micro scale, and it is often represented as "effective parameters". These effective parameters are often calculated in the sense of "averaging procedure" or "homogenization". As for the micro scale problem, various types of problem arise. They could be solved with either finite difference or finite element method, other kinds of numerical methods could also be employed. An extensive review is from the book [ref: multiscale thomas y hou]. A more general framework, the Heterogeneous Multiscale Method (HMM), was proposed by Bjorn Engquist. This method extend the idea of homogenization and introduce generic methodology between macro scale and micro scale. An introductory review could be found in [ref: weinan e].

In this part the basic idea of Homogenization Method is presented with a one dimensional example. Then the application to the 3d elliptic PDE is briefly discussed. Hill Mandel requirements should be fulfilled in the energy conserving problem. Hence they are stated in the end of this part. We confine our discussion here mainly on the material possessing periodic structures.

2.1 Periodic Structures

Periodicity appears frequently in composites, for instance material with fiber or particle reinforcement. The periodicity in these materials takes the form of periodic geometry and periodic materials. Concerning about material parameters they could be expressed with periodic functions of coordinates. For example, Young's Modulus can be written in the following form,

$$\mathbb{C}(\mathbf{x} + \mathbf{Y}) = \mathbb{C}(\mathbf{x}). \tag{2.1}$$

2.2 Scale Separation

When two scale problem is addressed, field variable could be expanded as follows,

$$\mathbf{\Phi}^\epsilon(\mathbf{x}) = \mathbf{\Phi}^0(\mathbf{x}, \mathbf{y}) + \epsilon\mathbf{\Phi}^1(\mathbf{x}, \mathbf{y}) + \epsilon\mathbf{\Phi}^2(\mathbf{x}, \mathbf{y}) + \cdots, \tag{2.2}$$

In the above formula, $\mathbf{x}$ is the position vector of a point, which is deemed as the *macroscopic* coordinate. $\mathbf{y} = \mathbf{x}/\epsilon$ is a *microscopic* coordinate, which stands for *rapid* oscillation. The physical nature of the right hand side is the decomposition of macro scale dependency and micro scale dependency with respect to reference cell. The purpose of setting $\mathbf{y} = \mathbf{x}/\epsilon$ is achieving a closed form expressed with the original coordinates. The ratio ϵ means that the quantity will vary $1/\epsilon$ faster in microscopic level. When ϵ goes to 0, functions $\mathbf{\Phi}^0(\mathbf{x}, \mathbf{y}), \mathbf{\Phi}^1(\mathbf{x}, \mathbf{y}), \cdots$ are smooth in $\mathbf{x}$ and $\mathbf{Y}$-periodic in $\mathbf{y}$.

The characteristic of field variable can be viewed in the diagram [figure: scale decomposition]

2.3 One Dimensional Problem

Many books and review papers list one dimensional problem, such as [ref: ciorsanescu]. Here we briefly go through one dimensional elasticity problem. More detailed derivation could be referred to [ref: B.hassani].

The governing equations such as the equilibrium and Hooke's law are,

$$\begin{cases} \dfrac{\partial \sigma^\epsilon}{\partial x} + \gamma^\epsilon = 0 \\ \sigma^\epsilon = E^\epsilon \dfrac{\partial u^\epsilon}{\partial x}, \end{cases} \tag{2.3}$$

Noting that ϵ in superscript represents its periodic property. γ^ϵ is the body weight of material. If E^ϵ and γ^ϵ are uniform in macro coordinate and only differ inside each cell, then the following relation holds,

$$E^\epsilon(x, x/\epsilon) = E^\epsilon(x/\epsilon) = E(y), \tag{2.4}$$

The relation with respect to body weight is likewise. Utilizing the double scale expansion referring to (2.2) it follows that,

$$\begin{cases} u^\epsilon(x) = u^0(x,y) + \epsilon u^1(x,y) + \epsilon^2 u^2(x,y) + \cdots \\ \sigma^\epsilon(x) = \sigma^0(x,y) + \epsilon\sigma^1(x,y) + \epsilon^2\sigma^2(x,y) + \cdots, \end{cases} \tag{2.5}$$

After substitution and equating the according terms, we have

$$\begin{cases} 0 = E(y)\left(\dfrac{\partial u^0}{\partial y}\right) \\ \sigma^0 = E(y)\left(\dfrac{\partial u^0}{\partial x} + \dfrac{\partial u^1}{\partial y}\right) \\ \sigma^1 = E(y)\left(\dfrac{\partial u^1}{\partial x} + \dfrac{\partial u^2}{\partial y}\right), \end{cases} \tag{2.6}$$

and

$$\begin{cases} \dfrac{\partial \sigma^0}{\partial y} = 0 \\ \dfrac{\partial \sigma^0}{\partial x} + \dfrac{\partial \sigma^1}{\partial y} + \gamma(y) = 0, \end{cases} \tag{2.7}$$

Simplification of (2.6) and (2.7) yields

$$\sigma^0(x) = \left(Y / \int_Y \frac{\mathrm{d}y}{E(y)}\right) \frac{\mathrm{d}u^0(x)}{\mathrm{d}x}. \tag{2.8}$$

Define the *homogenized modulus of elasticity* as follows,

$$E^H = 1 / \left(\frac{1}{Y}\int_0^Y \frac{\mathrm{d}\eta}{E(\eta)}\right). \tag{2.9}$$

Then the original problem is transformed to

$$\begin{cases} \sigma^0(x) = E^H \dfrac{\mathrm{d}u^0(x)}{\mathrm{d}x} \\ \dfrac{\mathrm{d}\sigma^0}{\mathrm{d}x} + \bar{\gamma} = 0, \end{cases} \tag{2.10}$$

where $\bar{\gamma} = 1/Y \int_Y \gamma(y)$ is the average of γ inside the cell. From (2.10) the differential equation for displacement holds as

$$\frac{\mathrm{d}^2 u^0(x)}{\mathrm{d}x^2} = -\frac{\bar{\gamma}}{E^H} \tag{2.11}$$

Regarding the boundary conditions on both ends deliver the result

$$u(x) = -\frac{\bar{\gamma}}{E^H}\frac{x^2}{2} + \frac{\bar{\gamma}}{E^H} Lx$$

2.4 General Elliptical PDE

If a general PDE for three dimensional problem is taken into account, it would be more intricate, as the solution is often sought in the sense of weak form. In this circumstance a homogenized weak form is considered instead of the homogenized differential operator. Then the limit of homogenized weak form should converge to the weak form without homogenization, which is called *G-convergence*, [ref: a comparison, u michi]. As for the case of elasticity tensor in the sense of differential operator G-convergence is the following,

$$\lim_{\epsilon\to 0} \frac{\partial}{\partial x_i}\left[C^\epsilon_{ijkl}\frac{\partial u^\epsilon_k}{\partial x_l}\right] \to \frac{\partial}{\partial x_i}\left[\bar{C}_{ijkl}\frac{\partial u_k}{\partial x_l}\right] \tag{2.12}$$

A quick overview of the general problem is given in the review [ref: hassani]. Several key points of the general problem are listed here. With the notation of general elliptical operator using

$$\mathcal{A}^\epsilon = \frac{\partial}{\partial x_i}\left(a_{ij}(\mathbf{y})\frac{\partial}{\partial x_j}\right). \tag{2.13}$$

general problem could then be described as,

$$\begin{cases} \mathcal{A}^\epsilon \mathbf{u}^\epsilon = \mathbf{f} & \text{in } \Omega \\ \mathbf{u}^\epsilon = \mathbf{0} & \text{on } \partial\Omega \end{cases} \tag{2.14}$$

Employing the double scale expansion for both the field variable $\mathbf{u}^\epsilon$ and the differential operator $\mathcal{A}^\epsilon$, namely (notice that chain rule is applied when differentiating)

$$\begin{cases} \mathbf{u}^\epsilon(\mathbf{x}) = \mathbf{u}^0(\mathbf{x},\mathbf{y}) + \epsilon\mathbf{u}^1(\mathbf{x},\mathbf{y}) + \epsilon^2\mathbf{u}^2(\mathbf{x},\mathbf{y}) + \cdots \\ \mathcal{A}^\epsilon = \dfrac{1}{\epsilon^2}\mathcal{A}^1 + \dfrac{1}{\epsilon}\mathcal{A}^2 + \mathcal{A}^3 \end{cases} \tag{2.15}$$

Here $\mathcal{A}^1, \mathcal{A}^2, \mathcal{A}^3$ is defined as follows

$$\mathcal{A}^1 = \frac{\partial}{\partial y_i}\left(a_{ij}(\mathbf{y})\frac{\partial}{\partial y_j}\right);\ \mathcal{A}^2 = \frac{\partial}{\partial y_i}\left(a_{ij}(\mathbf{y})\frac{\partial}{\partial x_j}\right) + \frac{\partial}{\partial y_i}\left(a_{ij}(\mathbf{y})\frac{\partial}{\partial x_j}\right);\ \mathcal{A}^3 = \frac{\partial}{\partial x_i}\left(a_{ij}(\mathbf{y})\frac{\partial}{\partial x_j}\right).$$

Substitution with the above differential operators and comparing with the according terms it follows

$$\begin{cases} \mathcal{A}^1\mathbf{u}^0 = \mathbf{0} \\ \mathcal{A}^1\mathbf{u}^1 + \mathcal{A}^2\mathbf{u}^0 = \mathbf{0} \\ \mathcal{A}^1\mathbf{u}^2 + \mathcal{A}^2\mathbf{u}^1 + \mathcal{A}^3\mathbf{u}^0 = \mathbf{f}. \end{cases} \tag{2.16}$$

Referring [ref: cioranescu] it is known that if a $\mathbf{Y}$-periodic function u has a unique solution in terms of $\mathcal{A}^1$ operator, i.e.

$$\mathcal{A}^1\mathbf{u} = \mathbf{F} \quad \text{in the reference cell.} \tag{2.17}$$

Then the right hand side of the above equation, $\mathbf{F}$ should satisfy

$$\overline{\mathbf{F}} = \frac{1}{|Y|}\int_Y \mathbf{F}\ \mathrm{d}\mathbf{y} = \mathbf{0}. \tag{2.18}$$

Applying this proposition to (2.16) several times the field variable could be expressed with the following form,

$$\mathbf{u}^1(\mathbf{x},\mathbf{y}) = \chi^i(\mathbf{y})\frac{\partial \mathbf{u}(\mathbf{x})}{\partial x_j} + \xi(\mathbf{x}) \tag{2.19}$$

Function $\chi^i(\mathbf{y})$ is local solution of the problem, which has $\mathbf{Y}$-periodic property. The local problem is

$$\mathcal{A}^1\chi^j(\mathbf{y}) = \frac{\partial a_{ij}(\mathbf{y})}{\partial y_i} \quad \text{in the reference cell.} \tag{2.20}$$

Hence the macro scale problem (homogenized problem) can be written as

$$\mathcal{A}^H\mathbf{u} = \mathbf{f}, \tag{2.21}$$

with

$$\mathcal{A}^H = a_{ij}^H\frac{\partial^2}{\partial x_i \partial x_j}. \tag{2.22}$$

And the effective coefficients are related with the solution of micro scale problem,

$$a_{ij}^H = \frac{1}{|Y|}\int_Y \left(a_{ij}(\mathbf{y}) + a_{ik}(\mathbf{y})\frac{\partial \chi^j}{\partial y_k}\right)\ \mathrm{d}\mathbf{y} \tag{2.23}$$

2.5 Hill-Mandel Condition

After introducing the general mathematical concepts about homogenization methods, we move to its application in material modelling. In this case a Representative Volume Element (RVE) is always investigated. Homogenization of the coefficients is then obtained through calculation on RVE. As RVE represents a material in the micro scale, the behaviour of RVE should resemble the material in this scale. Therefore the model for micro scale should be able to capture such features, for instance the continuum mechanical equilibrium of composites in the micro scale. Besides the boundary condition of micro scale model should also be compatible with macro scale. The Hill-Mandel condition needs to be fulfilled [ref: r.hill 1952 from gluege]

The Hill-Mandel condition states that the total stress power on the micro scale should be equal to the stress power at relevant point on the macro scale. For small strain, the following equation holds,

$$\langle \boldsymbol{\sigma} \cdot \dot{\boldsymbol{\varepsilon}} \rangle = \langle \boldsymbol{\sigma} \rangle \cdot \langle \dot{\boldsymbol{\varepsilon}} \rangle\,, \tag{2.24}$$

where $\langle \cdot \rangle$ means averaging of the considered variable.

3 Finite Element Framework FEniCS

FEniCS project was started at University of Chicago and Charmers University of Technology in 2003. The name "FEniCS" consists "FE", "Finite Element" and "CS", "Computational Software". "ni" makes the name sound like "phenix" and end with a meaningful name. It is a collection of packages that make the best of each package or concepts to realize Computational Mathematical and Modelling. The main interface language of FEniCS is Python, which is a fast language for prototyping, while the most of codes and libraries are implemented in C++, whose efficiency in computing stands out. Various linear algebra backends and parallelization allow FEniCS boost in speed. As for modelling, UFL (Unified Form Language) and FFC (FEniCS Form Compiler) make the direct translation of mathematical formulation such as linear form and bilinear form into symbolic codes possible. It can be regarded as the language of modelling, which simplify the modelling process to a large extent. [ref: literature]

This part concentrates on the introduction of FEniCS. Firstly the work flow and each component of FEniCS will be presented. Then the most simple example is given in order to clarify its efficiency in modelling. At last some key features of FEniCS is listed and discussed, which provide the flexibility in research and real world problems.

3.1 The Structure of FEniCS

The work flow of FEniCS can be described with the following diagram [pic: fenics blocks]. It is seen that the whole process is decomposed into three major parts. Each part represents one or several functionalities. For form compiler UFL language is involved. With the help of form compiler (FFC), C++ code for the whole model is obtained. Till this point the mathematical derivation, which is often the weak form of a PDE, is converted into pure symbolic efficient C++ codes. The interface between symbolic codes and numerical codes is called UFC (Unified Form-assembly Code). It works as a fixed interface that achieve efficient assembling of finite elements. [ref: website]

Numerical algorithms enter after the UFC interface. The main package that carries the numerical algorithm is DOLFIN. DOLFIN's task is to wrap functionalities of each components and coordinate them. The user interface of DOLFIN is Python, where efficient extraction of properties and manipulation are allowed in a easy fashion.

Other building blocks such as mshr for geometrical modelling and mesh generation, and FIAT (FInite element Automatic Tabulator) for automatic finite element generation for arbitrary orders. Linear algebra backends work as add-ons for FEniCS, which provides the possibility of extension. The visualization package in use is VTK, a substitute of the old one, Viper. The output files could be in various formats. Therefore software like ParaView could apply in terms of visualization. The full structure of FEniCS can be viewed in the following diagram.

[pic: full fenics]

3.2 Simple Poisson Problem

An implementation of simple Poisson problem is presented in the official tutorial of FEniCS. This example is reproduced here to give a quick overview of the usage and convince its simplicity in modelling. [ref: website]

```
from dolfin import *

# Create mesh and define function space
mesh = UnitSquareMesh(6, 4)
V = FunctionSpace(mesh, 'Lagrange', 1)

# Define boundary conditions
# Value function for boundary condition x[0]->x, x[1]->y
u0 = Expression('1 + x[0]*x[0] + 2*x[1]*x[1]')
# Mark boundary
def u0_boundary(x, on_boundary):
    return on_boundary
# Add Dirichlet boundary condition
bc = DirichletBC(V, u0, u0_boundary)

# Define variational problem
```

```
u = TrialFunction(V)
v = TestFunction(V)
f = Constant(-6.0)
# Bilinear Form
a = inner(nabla_grad(u), nabla_grad(v))*dx
# Linear Form
L = f*v*dx

# Compute solution
u = Function(V)
solve(a == L, u, bc)

# Plot solution
plot(u, interactive = True)

# Dump solution to file in VTK format
file = File('poisson.pvd')
file << u
```

As can be immediately seen in the code, the encapsulation of almost every functionality is well preserved in the package. Moreover, the name of functions and classes are in good consistent with the mathematical language. The modelling always begins with defining its geometry and function space. This setting is similar with solving a PDE in weak form that domain and function space is usually given in the condition. Geometry and mesh can also be necessarily imported using external files. Various formats are supported, such as `.msh` generated by `gmsh`. One thing to notice is that element type should be defined when defining th function space. As shown here `Lagrange`. The order of element is simply 1.

It follows with defining the boundary conditions such as Dirichlet boundary conditions. An `Expression` is a class holding the codes of creating a mathematical function in the domain. It is extensible with C++. Here we use it to define the boundary values. As for imposing this boundary condition, one need to mark the boundary first. The structure of marking boundary points, lines, and facets are all derived from this structure. `on_boundary` is only a predefined marker that will return `True` when on the boundary.

The mathematical model of this problem is

$$\begin{cases} -\Delta u(\mathbf{x}) = 0 & \text{in } \Omega \\ u(\mathbf{x}) = u_0 & \text{on } \partial\Omega. \end{cases} \tag{3.1}$$

The according weak form can be written as

$$a(u, v) = L(v), \ \forall v \in \hat{V} \tag{3.2}$$

with the bilinear form and linear form defined as

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\mathbf{x}, \ L(v) = \int_\Omega f v \, \mathrm{d}\mathbf{x}.$$

The sought after function is u in function space $V = \left\{v \in H^1(\Omega) : v = u_0 \text{ on } \partial\Omega\right\}$ and the according $\hat{V}$ is defined as $\hat{V} = \left\{v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega\right\}$. In the code the above mathematical model is conveniently expressed with `a = inner(nabla_grad(u), nabla_grad(v))*dx` and `L = f*v*dx`.

Next is the solving step, where one should redefine a new `Function` to hold the solution. Here overriding the above defined `u` is implemented. Post processing such as visualization of the result follows with `plot`. Output of the result is also standard with the listed commands.

The presentation of the simple Poisson problem serves as an introduction to the functionalities of FEniCS. To sum up, the implementation is in accordance with mathematical notation. Pre and post processing of the computation can be realized using handy functions supported by FEniCS. More useful features are discussed in the following section.

3.3 Relevant Key Features

When the problem becomes more complicated, there are more advanced and handy usage to be revealed. Here list only a few that appear in the material modelling and multi field problems.

First we explore the function space in FEniCS. There are several kinds of `FunctionSpace` in FEniCS. If a vector or tensor is used, `VectorFunctionSpace` or `TensorFunctionSpace` should be defined. More powerful usage is the `MixedFunctionSpace`, which receives a list of `FunctionSpace` and compose them together into a merged one. It is especially useful when multiple fields are accounted. When concerning about constructing function spaces, various function space families are available. Thanks to the general representation of function spaces arbitrary order of elements and a lot many of types are possible. For very specific type of function space

requiring periodicity `constrained_type` keyword is set to be a periodic mapping. This mapping is derived from `SubDomain`. Much caution needs to be devoted into the definition of mapping, which can be seen already in the current code for material modelling. Some properties are listed below.

```
# Useful properties and variables for FunctionSpace definition
FS = FunctionSpace(mesh=some_mesh, family='Lagrange', degree=2, constrained_domain=
                                                    some_defined_periodic_mapping)

# Vector and tensor function spaces
VFS = VectorFunctionSpace(mesh, 'Lagrange', 1)
TFS = TensorFunctionSpace(mesh, 'Hermite', 1)

# Merged function Space
MFS = MixedFunctionSpace([FS, VFS, TFS])
```

When it is concerned with multi field modelling, not only merged function but also split functions are required, since in the formulation different terms are calculated with different functions. A mixed function is generated with a `MixedFunctionSpace` and the corresponding component functions are obtained through `split` without losing dependency to the merged function. In the problem formulation it is often the case that inner product, outer product, derivative, and integral etc. are built. This is achieved easily with the help of UFL operators. Some of the operators such as `nabla_grad()` and `inner()` are already mentioned above. One strength of FEniCS is that this formulation is not restricted in the vector and matrix, but applicable by tensors of higher order. The index notation representation is also valid in the formulation, namely `i,j,k,l`, obeying Einstein's summation convention. After the derivation a stiffness matrix is assembled with `assemble`. The usages are as follows.

```
# Merged function space
MFS = MixedFunctionSpace([FS, VFS, TFS])

# Merged function and split functions
merged_f = Function(MFS)
(f, vf, tf) = split(merged_f)

# Integral
L = f*v*dx

# Derivative and its alternative
L_derivative = derivative(L, f, df)
L_diff = diff(L, f)

# Index notation
i, j, k, l = indices(4)
Energy = 0.5*F[i,j]*C_i_[i,j,k,l]*F[k,l]*dx

# Stiffness matrix, note that test function and trial function should be initiated first
f_test = TestFunction(FS)
f_trial = TrialFunction(FS)
a = f_test*C*t_trial*dx
K = assemble(a)
```

The last usage is particularly useful in composite modelling, i.e. defining different domains. The definition is also not complicated. Subdomains are defining through subclassing of `SubDomain`. Overriding its member method `inside()` gives the condition required for classifying the entity into subdomain. In integral the differentials are set accordingly to the subdomain.

```
# Inclusion definition
class Inclusion(SubDomain):
    def inside(self, x, on_boundary):
        d = sqrt((x[0] - 0.5)**2 + (x[1] - 0.5)**2)
        return d<0.25 or near(d,0.25)

# Initiate an instance
circle = Inclusion()

# Set cell domains for integral
domains = CellFunction("size_t", mesh)
domains.set_all(0)
# Mark inclusion
circle.mark(domains,1)

# Integrate accordingly
dx = Measure('dx', domain=mesh, subdomain_data=domains)
Pi = psi_m*dx(0) + psi_i*dx(1)
```