Stochastic Homogenization — Analysis and Numerics

武思蒙

Division of Mechanics, SUSTech

October, 2022

Preface

Periodic homogenization doesn't suffice to describe a general class of materials in nature, let alone to predict the effective property of them. The assumption for microstructures to be periodically arranged is too strong and actually, unnecessary. It is only for the simplification when performing analysis. In this piece of work, we liberate problems from periodic or almost periodic, and capture the core nature of randomness.

We first review some basic knowledge from probability theory and ergodic theory, then goes the analysis of stochastic homogenization, followed by some necessary computations, some of which are based on finite element or wavelet, while the others are based on a fairly new technique — deep learning.

2022 October, in Shenzhen, China
Wu Simeng (武思蒙)

Dedicated to my parents W. Huaijun
& L. Li
and my piggy ZRL qwq

Contents

Part I

Stochastic Homogenization

Chapter 1

Digest of Probability Theory and Stochastic Process

1.1 Probability space and properties

1.2 Random variables

Chapter 2

Random Homogenization

2.1 Assumptions

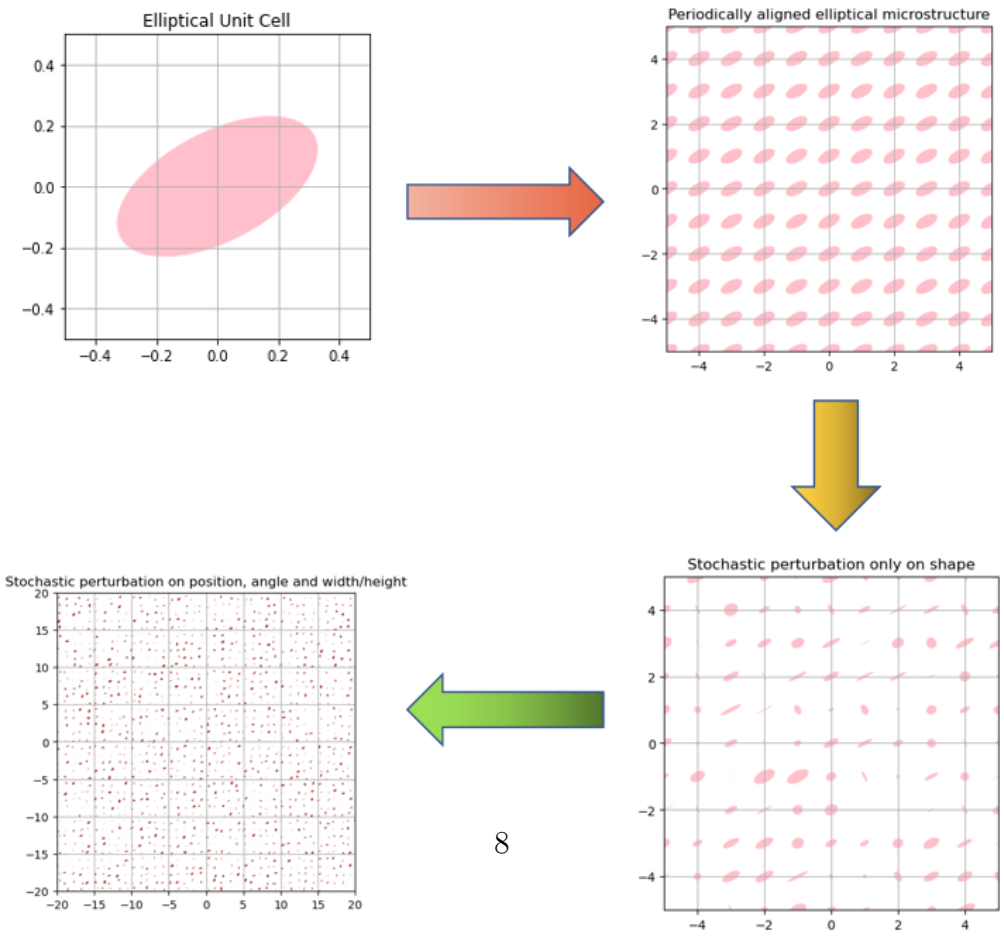

Figure 2.1

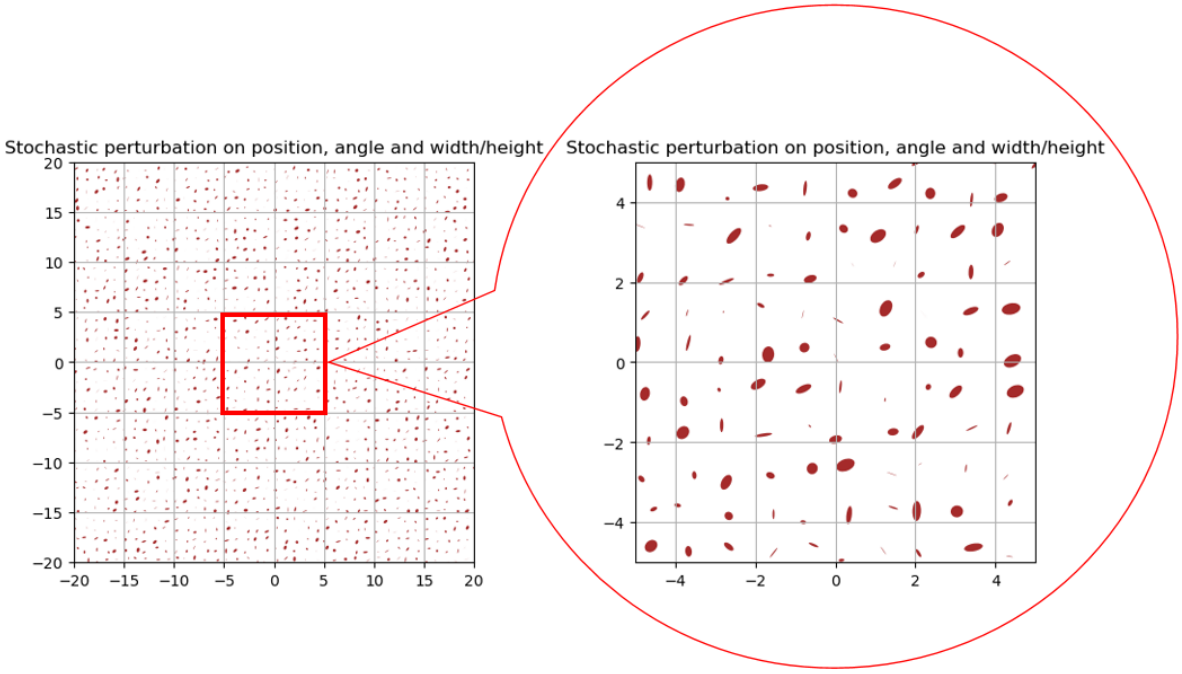
Stochastic perturbation on position, angle and width/height
Stochastic perturbation on position, angle and width/height

Figure 2.2

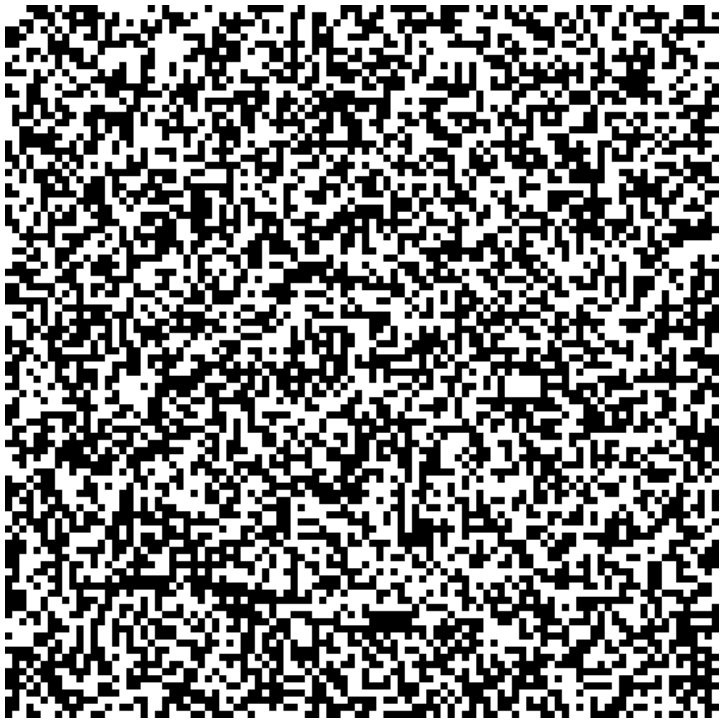

Figure 2.3: Random generated checkerboard at each slot

Part II

Computational Homogenization

Chapter 3

Finite Element Method

Verification of torsional rigidity of homogeneous square:

Timoshenko and Goodier's analytic result: from p. 278 of [7]

$$D = \frac{M}{\theta} = 0.1406G(2a)^4 = 0.1406 \times 4^4 = 35.9936$$

My mesh integration result: using the program 4

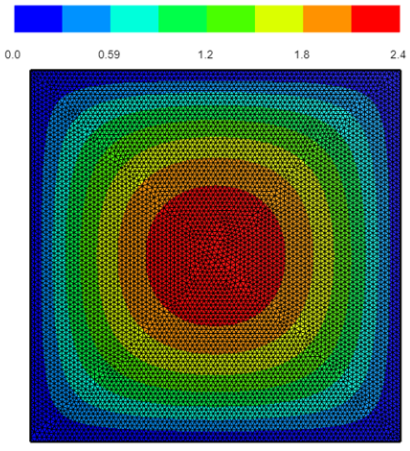

Figure 3.1: cloud picture

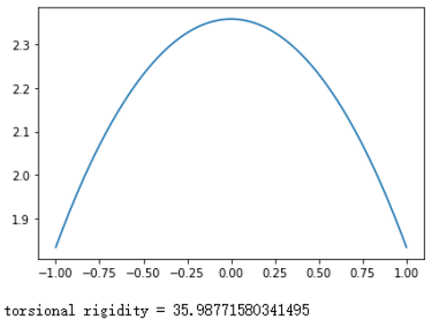

Figure 3.2: middle curve and rigidity

$$D = 35.98771580341495$$

When considering program 5 The computation result is

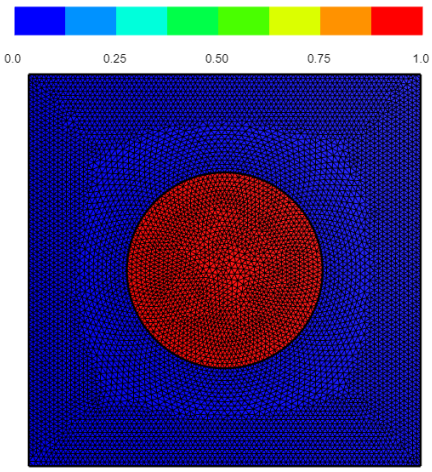

Figure 3.3: inner material

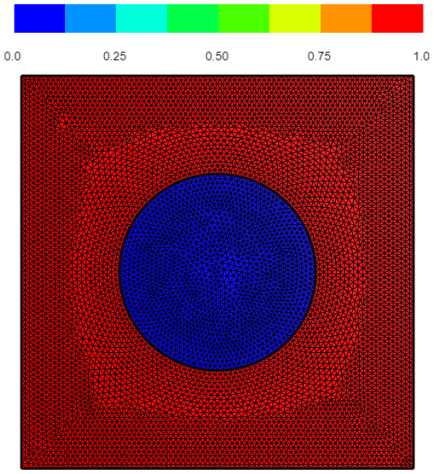

Figure 3.4: outer material

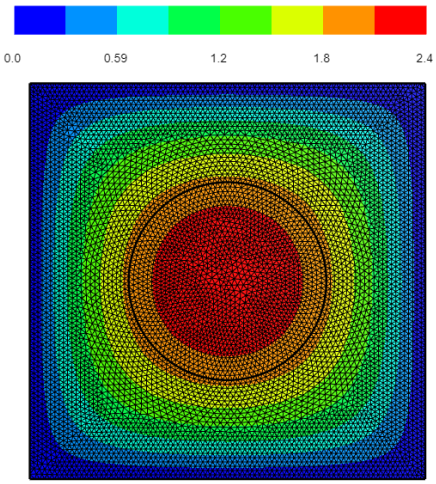

Figure 3.5: cloud picture

$$D = 35.98771580336777$$

When there're small perturbation of the inner material, e.g., if $\mu_1 = \frac{1}{1.01} = 0.9900990099009901$, the resultant rigidity reads

$$D = 35.956661972872176$$

Chapter 4

Neural Networks

Torch 安装过程易 bug 事项总结：

1. 修改 conda 默认启动环境：在 Scripts 文件夹中找到 activate.bat，用文本打开，末尾添加自己想要默认启动的虚拟环境。
2. 新建虚拟环境后 jupyter notebook 中显示找不到 torch 模块，这是因为少了一些依赖关系，输入”conda install jupyter”, ”conda install nb_conda” 后即可正常使用。
3. 需要用 GPU 辅助计算时，应当先安装 v11.3（从英伟达控制面板的左下角系统信息中查看合适的版本号）的 cuda，再从 pytorch 官网查找同时安装 torch+cuda 的命令，最后在虚拟环境下的 prompt 中输入指令安装。

In FEM method, we've discussed when the coefficient function is simple function. However, when the coefficient field exhibits multi-scale property,

for example,

$$A(x_1, x_2) = \prod_{k=1}^{7}(1 + 0.2\cos(2^k\pi(x_1 + x_2)))(1 + 0.2\sin(2^k\pi(x_2 - 3x_1)))$$

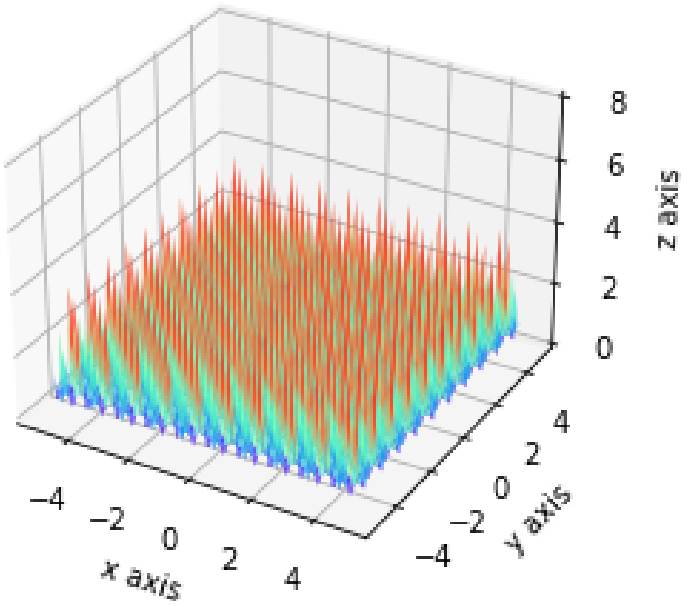

Figure 4.1: plot of $A(x_1, x_2)$

Part III

晶体结构

Chapter 5

晶格

我们将考虑理想晶格，也就是那些既没有缺陷，又在空间上无限延伸的周期性排布方式。世界上没有真正的理想晶格，即便半导体的硅片可以做到没有晶体缺陷，但他们也是有限尺寸的。

晶格的格点表示为**初基晶轴**（primitive lattice vector）的整线性组合

$$\begin{aligned} R^{[l]} &= l_i \hat{A}_i, \quad l_i \in \mathbb{Z} \\ &= \hat{H} l \\ &= \begin{bmatrix} \hat{A}_1^{(1)} & \hat{A}_2^{(1)} & \hat{A}_3^{(1)} \\ \hat{A}_1^{(2)} & \hat{A}_2^{(2)} & \hat{A}_3^{(2)} \\ \hat{A}_1^{(3)} & \hat{A}_2^{(3)} & \hat{A}_3^{(3)} \end{bmatrix} \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} \end{aligned} \tag{5.1}$$

其中 $\hat{A}_1$ $\hat{A}_2$ $\hat{A}_3$ 称为初基晶轴，他们组成的六面体称为**原胞**（primitive cell），显然初基晶轴的选取是不唯一的，但是原胞的体积 $\hat{\Omega}_0 = |\hat{A}_1 \cdot \hat{A}_2 \times \hat{A}_3|$ 始终保持不变。这个体积在考虑将离散的原胞映射到连续介质场时扮演着重要角色。

由于每个点都由数个原胞共享，因此计数公式为：二维：$4 \times \frac{1}{4}$；三维：$8 \times \frac{1}{8}$. 注意到以下两点，

- 晶格具有平移不变性；
- 原胞不能包含内格点，且原胞含有的格点数为 1。

相对点 x 的点对称操作（Point symmetry operations）定义为一个保持 x 不变的变换。

Chapter 6

Appendix: Source Codes

1. Generating periodic checkerboard structure

```
from PIL import Image
import numpy as np
import math
import random

def create_pattern(width, height):
black_image = PIL.Image.new(mode='RGB', size=(width,
                            height), color=(0,0,0))
image_array = np.array(black_image)
square_width = math.floor(width / 10) #size of a
                            square
white_turn = False

for i in range(0, len(image_array), square_width): #
                            the 3rd argument is step
                            length
white_turn = not white_turn
for j in range(0, len(image_array[0]), square_width):
white_turn = not white_turn
for ii in range(i, i+square_width):
```

```
for jj in range(j, j+square_width):
if white_turn:
image_array[ii,jj] = (255, 255, 255)
checker_pattern = PIL.Image.fromarray(image_array)
checker_pattern.show()

create_pattern(500,500)
```

2. Generating random checkerboard structure

```
import PIL
import numpy as np
import math
import random

def create_pattern(width, height):
black_image = PIL.Image.new(mode='RGB', size=(width,
                                height), color=(0,0,0
                                ))
image_array = np.array(black_image)
square_width = math.floor(width / 100) #size of a
                                square
white_turn = False

for i in range(0, len(image_array[0]), square_width)
                                :
for j in range(0, len(image_array[0]), square_width)
                                :
c = random.random()
if c < 0.5 :
white_turn = True
else :
white_turn = False
for ii in range(i, i+square_width):
```

```
for jj in range(j, j+square_width):
if white_turn:
image_array[ii,jj] = (255, 255, 255)

checker_pattern = PIL.Image.fromarray(image_array)
checker_pattern.show()

create_pattern(500,500)
```

3. Generating perturbed elliptical microstructure

```
# random in position, angle and width/height
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
import random

fig = plt.figure(0)
ax = fig.add_subplot(111, aspect='equal')

for p in range(0, 41, 1):
for k in range(0, 41, 1):
e = Ellipse(xy = ((k-20)+0.5*random.random(),(p-20)+
                          0.5*random.random()),
                          width = 2.5 * 2*0.1 *
                          random.random(), height =
                          1.5 * 2*0.1*random.random
                          (), angle = 90 * random.
                          random())
ax.add_artist(e)
e.set_facecolor("brown")
plt.xlim(-20, 20)
plt.ylim(-20, 20)
ax.grid(True)
plt.title("Stochastic perturbation on position,
                          angle and width/height")
```

```
plt.show()
```

4. FEM computation of torsional rigidity of homogeneous square $(-2,2)^2$

```
from ngsolve import *
import netgen.geom2d as geom2d
from ngsolve.webgui import Draw

ngsglobals.msg_level = 1

geo = geom2d.SplineGeometry()
p1,p2,p3,p4 = [ geo.AppendPoint(x,y) for x,y in [(-2
                                ,-2), (2,-2), (2,2), (
                                -2,2)] ]
geo.Append (["line", p1, p2], leftdomain=1,
                                rightdomain=0)
geo.Append (["line", p2, p3], leftdomain=1,
                                rightdomain=0)
geo.Append (["line", p3, p4], leftdomain=1,
                                rightdomain=0)
geo.Append (["line", p4, p1], leftdomain=1,
                                rightdomain=0)

# generate a triangular mesh of mesh-size maxh
mesh = Mesh(geo.GenerateMesh(maxh=0.05))

# H1-conforming finite element space
fes = H1(mesh, order=3, dirichlet=[1,2,3,4])

# define trial- and test-functions
u = fes.TrialFunction()
v = fes.TestFunction()

# define the coefficient function
```

```
b = 1
c = 2

# the right hand side
f = LinearForm(fes)
f +=  c *v * dx

# the bilinear-form
a = BilinearForm(fes, symmetric=True)
a += b*grad(u)*grad(v)*dx

a.Assemble()
f.Assemble()

# the solution field
gfu = GridFunction(fes)
gfu.vec.data = a.mat.Inverse(fes.FreeDofs(), inverse
                              ="sparsecholesky") * f
                              .vec
# print (u.vec)


# plot the solution (netgen-gui only)
Draw (gfu)
#Draw (-grad(gfu), mesh, "Flux")

# 2D plot
import matplotlib.pyplot as plt
N = 100
ypnts = [i/N for i in range(-N, N+1, 1)]
vals = [gfu(mesh(0,yi)) for yi in ypnts]

plt.plot(ypnts,vals)
plt.show(block=False)
```

```
print ("torsional rigidity =", Integrate ( grad(gfu)
                                *grad(gfu), mesh))
```

5. FEM for multi-materials

```
from ngsolve import *
import netgen.geom2d as geom2d
from ngsolve.webgui import Draw

geo = geom2d.SplineGeometry()
geo.AddCircle(c=(-0,0), r=1, bc="c", leftdomain=2,
                                rightdomain=1)
geo.AddRectangle((-2,-2), (2,2), bcs=["b","r","t","l
                                "], leftdomain=1)

geo.SetMaterial(1, "outer")
geo.SetMaterial(2, "inner")

# generate a triangular mesh of mesh-size maxh
mesh = Mesh(geo.GenerateMesh(maxh=0.05))

fes1 = H1(mesh, definedon="inner")
u1 = GridFunction(fes1, "u1")
u1.Set (1.01)

fes2 = H1(mesh, definedon="outer")
u2 = GridFunction(fes2, "u2")
u2.Set (1)


fes = H1(mesh, order=3,dirichlet="b|l|r|t")
u = fes.TrialFunction()
v = fes.TestFunction()

gfu = GridFunction(fes)
```

```
f = LinearForm(fes)
f += 2*v*dx
f.Assemble()

a = BilinearForm(fes)
a += SymbolicBFI (u1*grad(u)*grad(v), definedon=mesh
                                  .Materials("inner"))
a += SymbolicBFI (u2*grad(u)*grad(v), definedon=mesh
                                  .Materials("outer"))
a.Assemble()

gfu.vec.data = a.mat.Inverse(fes.FreeDofs()) * f.vec

sts = [-grad(gfu)[1], -grad(gfu)[0]]

Draw (u1); Draw (u2)
Draw (gfu)
#Draw (sts, mesh, "Flux")

print ("torsional rigidity =", Integrate ( u1*grad(
                                  gfu)*grad(gfu), mesh)+
                                  Integrate ( u2*grad(
                                  gfu)*grad(gfu), mesh))
```

Bibliography

[1] A. Bensoussan, J.-L. Lions and G. Papanicolaou: Asymptotic Analysis for Periodic Structures. *American Mathematical Society*, 2011.

[2] Doina Cioranescu and Patrizia Donato: An Introduction to Homogenization. *Oxford University Press*, 1999.

[3] Elias M. Stein and Rami Shakarchi: Real Analysis – Measure Theory, Integration, and Hilbert Spaces. *Princeton University Press*, 2005.

[4] Haim Brezis: Functional Analysis, Sobolev Spaces and Partial Differential Equations. *Springer*, 2010.

[5] Lawrence C. Evans: Partial Differential Equations — Second Edition. *American Mathematical Society*, 2010.

[6] Wu Simeng: Variational Method, Finite Element Implementation and Their Applications to Liquid Crystal Elastomers. *Unpublished*, 2022.

[7] S. Timoshenko and J. N. Goodier: Theory of Elasticity — Second Edition. *McGraw-Hill*, 1951.