Computational Materials Science 83 (2014) 488–495

Contents lists available at ScienceDirect

Computational Materials Science

journal homepage: www.elsevier.com/locate/commatsci

How to determine composite material properties using numerical homogenization

Erik Andreassen *, Casper Schousboe Andreasen

Department of Mechanical Engineering, Technical University of Denmark, Nils Koppels Allé, Building 404, Denmark

ARTICLE INFO

Article history:
Received 25 May 2013
Received in revised form 22 August 2013
Accepted 3 September 2013
Available online 18 November 2013

Keywords:
Numerical homogenization
Microstructure
MicroFE
Matlab

ABSTRACT

Numerical homogenization is an efficient way to determine effective macroscopic properties, such as the elasticity tensor, of a periodic composite material. In this paper an educational description of the method is provided based on a short, self-contained Matlab implementation. It is shown how the basic code, which computes the effective elasticity tensor of a two material composite, where one material could be void, is easily extended to include more materials. Furthermore, extensions to homogenization of conductivity, thermal expansion, and fluid permeability are described in detail. The unit cell of the periodic material can take the shape of a square, rectangle, or parallelogram, allowing for all kinds of 2D periodicities.

1. Introduction

The microstructure of composite materials, where two or more materials are combined to achieve a material with attractive properties, can often be described by a unit cell, which is periodically repeated in one or more directions, as illustrated in Fig. 1.

Such periodic, or almost periodic, microstructures can be found in materials such as fiber composites and bone. Information about the microstructure can be obtained by e.g. a CT-scan. The technique described in the following can thereafter be applied to find the effective properties of the material. For human bone this has been done by e.g. Hollister [1], who has also been among the first to apply the technique for the design of metal and polymer implants [2,3].

Assuming length scales where the theory of elasticity can be applied and perfect bonding between the different materials in the unit cell, homogenization can be used to compute the macroscopic composite material properties. Homogenization relies on an asymptotic expansion of the governing equations, which allows for a separation of scales. This is valid when there is a clear separation between the macro- and microscopic length scales. The theory behind homogenization is covered in detail in several works, some of the first being [4,5]. Another good theoretical introduction to the subject can be found in [6].

According to the theory of homogenization, the macroscopic elasticity tensor E^H_{ijkl} of a periodic composite material can be computed as:

$$E^H_{ijkl} = \frac{1}{|V|}\int_V E_{pqrs}\left(\varepsilon^{0(ij)}_{pq} - \varepsilon^{(ij)}_{pq}\right)\left(\varepsilon^{0(kl)}_{rs} - \varepsilon^{(kl)}_{rs}\right)\mathrm{d}V \tag{1}$$

where $|V|$ denotes the volume of the unit cell, E_{pqrs} is the locally varying stiffness tensor, $\varepsilon^{0(ij)}_{pq}$ are prescribed macroscopic strain fields (in 2D there are three; e.g. unit strain in the horizontal direction (11), unit strain in the vertical direction (22), and unit shear strain (12 or 21)), while the locally varying strain fields $\varepsilon^{(ij)}_{pq}$ are defined as:

$$\varepsilon^{(ij)}_{pq} = \varepsilon_{pq}(\chi^{ij}) = \frac{1}{2}\left(\chi^{ij}_{p,q} + \chi^{ij}_{q,p}\right) \tag{2}$$

based on the displacement fields χ^{kl} found by solving the elasticity equations with a prescribed macroscopic strain

$$\int_V E_{ijpq}\varepsilon_{ij}(v)\varepsilon_{pq}(\chi^{kl})\mathrm{d}V = \int_V E_{ijpq}\varepsilon_{ij}(v)\varepsilon^{0(kl)}_{pq}\mathrm{d}V \quad \forall v \in V \tag{3}$$

where v is a virtual displacement field. For most practical problems the homogenization is performed numerically by discretizing and solving Eq. (3) using e.g. the finite element method. This is often referred to as numerical homogenization. The numerical homogenization procedure is also well described in the literature. One of the first detailed descriptions of the procedure can be found in [7], while [8–10] provide a three paper review of both numerical homogenization and how it is used in conjunction with topology optimization to design periodic materials.

However, the implementation can still seem daunting, and with the small and self-contained Matlab example provided in Appendix A, we try to lower the barrier for using numerical homogenization. The code computes the homogenized elasticity tensor for a two

* Corresponding author.
E-mail address: erand@mek.dtu.dk (E. Andreassen).

0927-0256/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.commatsci.2013.09.006

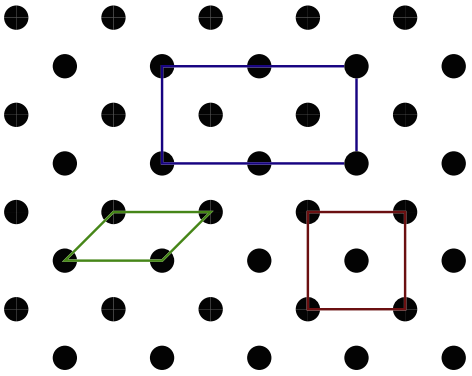

Fig. 1. A section of a 2D periodic microstructure consisting of two materials (white and black). The red line encloses a square unit cell, the blue a rectangular unit cell, and the green a parallelogram unit cell. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

material composite. A detailed description of the implementation is provided in Section 2, while examples of extensions to three material composites, homogenized thermal expansion, and thermal conductivity are given in Section 3. Section 4 is devoted to a slightly more involved extension of homogenized permeability.

2. Matlab implementation

In the basic Matlab implementation we treat the case of a composite consisting of two materials. The unit cell is discretized using bilinear finite elements (plane strain elements are used, but plane stress can be specified by providing modified material data), and an indicator matrix `x` specifies whether a finite element contains material 1 ($x_e = 1$) or material 2 ($x_e = 2$).

In Fig. 2 the structure of the mesh used to discretize the unit cell and the indicator matrix `x` are illustrated. Fig. 2b also shows how the geometry of the unit cell is specified in the Matlab code. The homogenization function needs six user specified inputs. The two first arguments (`lx` and `ly`) are the width, l_x and height, l_y, of the unit cell. The third argument (`lambda`) is a vector containing Lame's first parameter for material 1 and for material 2. Similarly, the fourth argument (`mu`) is a vector with Lame's second parameter for the two materials. The fifth argument (`phi`) is the angle, ϕ, between the horizontal axis and the left wall in the unit cell. Finally, the sixth argument is the indicator matrix `x`. The discretization is determined from the size of `x`; number of rows equals number of elements in the vertical direction, and number of columns equals number of elements in the horizontal direction.

Remark the angle ϕ should be given in degrees and to avoid overly distorted elements it should not be smaller than 45° nor larger than 135°. As discussed in [11] a parallelogram unit cell allows for the analysis of general periodic materials, including polygonal cells.

Calling the function in Appendix A as:

```
x = randi([1 2],200)
homogenize(1,1,[.01 2],[0.02 4],90,x)
```

will compute the effective properties of a random microstructure consisting of two materials, where the stiff material has an elasticity modulus of about 100 times the soft material. The homogenization is done by discretizing the unit cell with 200 times 200 bilinear elements, since that is the size of `x`. The different parts of the homogenization procedure implementation are explained in detail in the following.

If only a single material is used the stiffness will be constant throughout the unit cell resulting in zero displacements i.e. $\varepsilon_{ij} = 0$ and the original stiffness is obtained when applying Eq. (1).

2.1. The element stiffness matrix and load vectors (lines 17 and 86–125)

The elasticity equation from (3) can be discretized using the finite element method. The left hand side, i.e. the stiffness matrix, yields:

$$\mathbf{K} = \sum_{e=1}^{N} \int_{V_e} \mathbf{B}_e^T \mathbf{C}_e \mathbf{B}_e \mathrm{d}V_e \tag{4}$$

where the summation denotes the assembly of N finite elements. The matrix $\mathbf{B}_e$ is the element strain–displacement matrix, V_e is the volume of element e, and $\mathbf{C}_e$ is the constitutive matrix for the element, which for an isotropic material (we assume the materials used to build the composite are isotropic) is:

$$\mathbf{C}_e = \lambda_e \cdot \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \mu_e \cdot \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where λ_e and μ_e are Lamé's first and second parameter for the material in element e, respectively. Lamé's parameters can be computed from Young's modulus E and the Poisson's ratio v using the relations:

$$\lambda = \frac{vE}{(1+v)(1-2v)}, \quad \mu = \frac{E}{2(1+v)} \tag{6}$$

And to get plane stress properties Lamé's first parameter must further be modified as follows:

$$\hat{\lambda} = \frac{2\mu\lambda}{\lambda + 2\mu} \tag{7}$$

In the initialization the element stiffness matrix is split into two corresponding parts, such that the stiffness matrix is a function of the material properties in the elements:

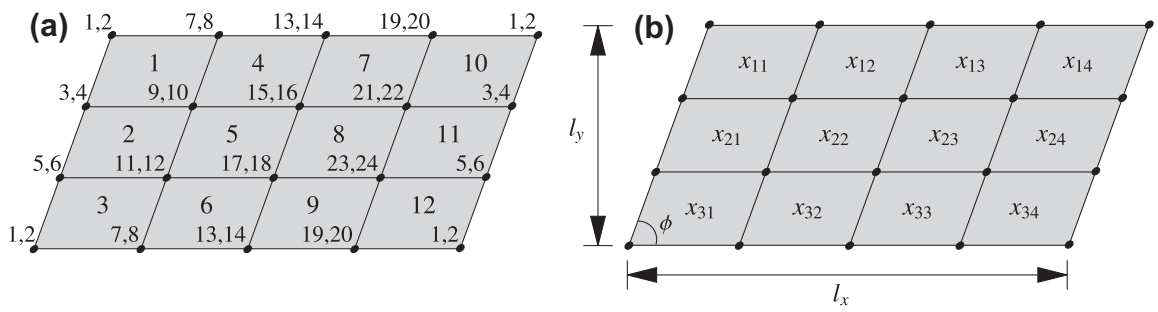

Fig. 2. (a) Illustration of finite element mesh used to discretize unit cell (element numbers are big, and degrees of freedom are small) and (b) corresponding structure of indicator matrix `x`.

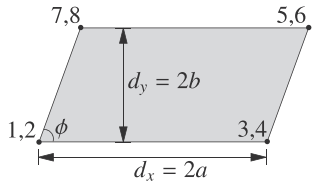

Fig. 3. Illustration of how local degrees of freedom in an element are numbered.

$$\mathbf{K} = \sum_{e=1}^{N} \mathbf{k_e} = \sum_{e=1}^{N} \left(\lambda_e \mathbf{k}_\lambda + \mu_e \mathbf{k}_\mu\right) \tag{8}$$

where the split simplifies the extension to more materials.

The discretization of the right hand side of (3) is the loads $\mathbf{f}^i$ which correspond to macroscopic volumetric straining

$$\mathbf{f}^i = \sum_e \int_{V_e} \mathbf{B}_e^T \mathbf{C}_e \boldsymbol{\varepsilon}^i \mathrm{d}V_e \tag{9}$$

where the strains are chosen to be:

$$\boldsymbol{\varepsilon}^1 = (1,0,0)^T, \quad \boldsymbol{\varepsilon}^2 = (0,1,0)^T, \quad \boldsymbol{\varepsilon}^3 = (0,0,1)^T \tag{10}$$

However, any three independent strains can be used, but for simplicity unit strains in the coordinate directions have been choosen. The load vectors are assembled using a split as for the stiffness matrix:

$$\mathbf{f}^i = \sum_e \left(\lambda_e \mathbf{f}_\lambda^i + \mu_e \mathbf{f}_\mu^i\right) \tag{11}$$

Finally, the displacement fields are computed solving the finite element problem with three loadcases (six in 3D):

$$\mathbf{K}\boldsymbol{\chi}^i = \mathbf{f}^i, \quad i = 1, \ldots, 3 \tag{12}$$

where the displacement vectors $\boldsymbol{\chi}^i$ are assumed to be V-periodic.

The computations of $\mathbf{k}_\lambda$, $\mathbf{k}_\mu$, $\mathbf{f}_\lambda^i$, and $\mathbf{f}_\mu^i$ are all done in the call to `elementMatVec` in line 17. Here we utilize the identical shape of the elements (not necessary as such for the homogenization, but makes the implementation simpler) by only executing this function once.

The element integration is done by mapping to an isoparametric element and computing the integral numerically using four Gauss points as described in e.g. [12]. The local numbering of degrees of freedom in the element is illustrated in Fig. 3, wherefrom the coordinate matrix in the computation of the Jacobian in lines 104 and 105 can be deduced. The use of tan (ϕ) in line 104 does not cause an issue when $\phi = 90°$, because Matlab just returns a very large number, implying 1/tan (90°) = 0.

In lines 119–123 the element matrix and element loads are computed, and as mentioned earlier these are kept in two separate parts; one should be multiplied with λ_e and one with μ_e.

2.2. Degrees of freedom and periodic boundary conditions (lines 19–36)

In order to assemble the global stiffness matrix and load vectors, we utilize the concept of an index matrix denoted `edofMat`, as also done in [13]. Consider the 12 element mesh in Fig. 2a. If the degrees of freedom were not periodic `edofMat` would have the structure in Fig. 4a. Each row i (shown in bold), contains the global degrees of freedom associated with element i. The non-periodic `edofMat`, which is created in line 21, is used to index into a periodic version of the mesh to create a periodic `edofMat` in line 35. The structure of the periodic `edofMat` is shown in Fig. 4b.

The periodic boundary conditions are imposed using elimination. This corresponds to using the same nodes on two opposite faces. This is implemented by using the matrix `edofMat` for a full, regular grid to index into a periodic version of the grid.

2.3. Assembly of the stiffness matrix (lines 37–46)

The assembly of the sparse stiffness matrix is based on triplets. First vectors with the row and column indices of the non-zero entries are created from `edofMat` in lines 39 and 40. Then matrices with the element material properties λ_e and μ_e are assigned in lines 42 and 43 based on the indicator matrix `x`. If the user supplies an indicator matrix with values other than 1 or 2, the element material properties will be wrongly assigned.

In line 45 the element matrices `keLambda` and `keMu` are multiplied with the corresponding element properties and added together. The resulting vector `sK` contains 64 (8 · 8) entries for each element. Finally, in line 46 the global stiffness matrix is assembled using the triplets.

2.4. Load vectors and solution (lines 47–54)

In lines 49–53 the load vectors are assembled in a similar fashion as the stiffness matrix. Thereafter the system in Eq. (12) is solved for $\boldsymbol{\chi}^i$. Due to small numerical differences Matlab might not recognize K as a symmetric matrix and therefore use a more general, but slower, linear solver. If speed is an issue, this could be remedied by adding `K = 0.5*(K + K.');`.

2.5. Homogenization (lines 55–84)

Before the homogenized elasticity tensor is computed the displacements of an element corresponding to the unit strain cases are found. This is simply done by solving for the element's nodal displacement corresponding to the uniform strains in Eq. (10), while constraining enough degrees of freedom to make the element stiffness matrix non-singular. This is done in line 62 as:

```
chi0_e([3 5:end],:)=ke([3 5:end],[3 5:end])\fe([3 5:end],:);
```

where the first, second, and fourth degree of freedom are constrained. The resulting element displacements are the same for all elements, since all elements are equivalent in the mesh used.

When the displacements have been obtained, the entries in the homogenized constitutive matrix $\mathbf{C}^H$ can be found as:

$$C_{ij}^H = \frac{1}{|V|} \sum_{e=1}^{N} \int_{V_e} \left(\boldsymbol{\chi}_e^{0(i)} - \boldsymbol{\chi}_e^{(i)}\right)^T \mathbf{k}_e \left(\boldsymbol{\chi}_e^{0(j)} - \boldsymbol{\chi}_e^{(j)}\right) \mathrm{d}V_e \tag{13}$$

where $\boldsymbol{\chi}_e^0$ contains the three displacement fields corresponding to the unit strains in Eq. (10), and $\boldsymbol{\chi}_e$ contains three columns corresponding to the three displacement fields resulting from

(a)

1	3	4	11	12	9	10	1	2
2	5	6	13	14	11	12	3	4
3	7	8	15	16	13	14	5	6
4	11	12	19	20	17	18	9	10
⋮	⋮	⋮	⋮	⋮				
12	31	32	39	40	37	38	29	30

(b)

1	3	4	9	10	7	8	1	2
2	5	6	11	12	9	10	3	4
3	1	2	7	8	11	12	5	6
4	9	10	15	16	13	14	7	8
⋮	⋮	⋮	⋮	⋮				
12	19	20	1	2	5	6	23	24

Fig. 4. Structure of the `edofMat`-matrix for 12 element mesh in (a) a non-periodic version, and (b) periodic version. The bold numbers denote the row numbers and indicate which element the degrees of freedom in the row belong to.

globally enforcing the unit strains in Eq. (10). The indices in parentheses refer to the column number. $|V|$ is the unit cell volume.

The sum in Eq. (13) is computed in lines 71–82, and it can be seen that again the summation is split in a λ and μ part, which is added together after being multiplied with the corresponding element material properties. After the homogenization is done, the homogenized elasticity tensor is displayed.

For meshes where the elements differ in shape, and thus the numerical integration must be performed for each element, the homogenization can be written as:

$$\mathbf{C}^H = \frac{1}{|V|}\sum_{e=1}^{N}\int_{V_e}(\mathbf{I}-\mathbf{B}_e\boldsymbol{\chi}_e)^T\mathbf{C}_e(\mathbf{I}-\mathbf{B}_e\boldsymbol{\chi}_e)\mathrm{d}V_e \tag{14}$$

where $\mathbf{I}$ is a three times three identity matrix (six in 3D). The term $\mathbf{B}_e\boldsymbol{\chi}_e$ can be interpreted as the strains caused by the non-homogeneous material distribution.

3. Extensions

In the following some extensions to allow for more materials and the homogenization of other properties are presented. The mentioned lines will refer to line numbers of the original unextended version of the code attached in Appendix A.

3.1. One material phase and void

Single-phase architecture cellular materials can be simulated by assigning a very soft second material. I. e. $\lambda_2 = 10^{-9}\lambda_1$, and $\mu_2 = 10^{-9}\mu_1$. Alternatively, the void elements can be ignored when solving the system by substituting line 54 with the lines:

```
activedofs = edofMat(x==1,:);
activedofs = sort(unique(activedofs(:)));
chi = zeros(ndof,3);
chi(activedofs(3:end),:) = ...
  K(activedofs(3:end),activedofs(3:end))...
\F(activedofs(3:end),:);
```

The last approach will save some computational time, but it does of course require material 1 to be connected. Furthermore, to get the correct homogenized constitutive matrix the material properties of material 2 must be set to 0.

3.2. Three material phases

In this section the extension to three material phases is described. Since the element stiffness matrix and load vectors have been split in a λ and μ part, the extension to more materials is straight-forward. The element material properties are assigned in lines 42 and 43 as:

```
mu = mu(1)*(x==1) + mu(2)*(x==2);
lambda = lambda(1)*(x==1) + lambda(2)*(x==2);
```

The above assumes entries in the indicator matrix contain either a 1 or 2. For three materials line 42 should be substituted with:

```
mu = mu(1)*(x==1) + mu(2)*(x==2) + mu(3)*(x==3);
```

which means the entries in the indicator matrix `x` should take one of the values 1, 2, or 3. Line 43 should of course be modified in the same way as line 42. Furthermore, the input vectors `lambda` and `mu` should have three entries each.

Assuming we have a material structure as the one shown in Fig. 1 where every other row of circular inclusions, each with a radius of 1/8 of the spacing between their centers, is alternating between material 2 and 3 in a matrix of material 1. If we assign the material properties $\lambda_1 = 1$, $\lambda_2 = 50$, $\lambda_3 = 100$ and $\mu_1 = 2$, $\mu_2 = 40$, $\lambda_3 = 80$ to the three phases, the resulting homogenized constitutive matrix is:

$$\mathbf{C}^H = \begin{bmatrix} 5.8 & 1.2 & 0.0 \\ 1.2 & 5.8 & 0.0 \\ 0.0 & 0.0 & 2.3 \end{bmatrix} \tag{15}$$

In Fig. 5 the corresponding square unit cell is shown together with force distributions. The force distributions illustrates clearly how the force vectors in Eq. (12) only have non-zero entries at the interfaces between the materials. If there are no material interfaces, there would be no local deformations in the unit cell. That is the solutions to Eq. (12) would be vectors of zeros.

3.3. Thermal conductivity

The homogenization equations for the thermal conductivity are analogous to those of the elastic problem though it is enough to solve for a scalar field – the temperature. Remark also that the problem is identical for electrical conductivity. The equations can be written as:

$$\int_V v_{,i}\mu_{ij}T^k_{,j}\mathrm{d}V = \int_V v_{,i}\mu_{ij}T^{0(k)}_{,j}\mathrm{d}V \quad \forall v \in V \tag{16}$$

$$\mu^H_{ij} = \frac{1}{|V|}\int_V (T^{0(i)}_{,l} - T^{(i)}_{,l})\mu_{lm}(T^{0(j)}_{,m} - T^{(j)}_{,m})\mathrm{d}V \tag{17}$$

where v is a virtual temperature field, μ_{ij} is the conductivity tensor and T is the temperature field. The finite element problem and the homogenization is analogous to the elasticity tensor homogenization. However, only one material parameter, μ, is necessary to describe the conductivity of an isotropic material, which should be an input vector to the homogenization function. Therefore, the changes described below assume the user gives the conductivity

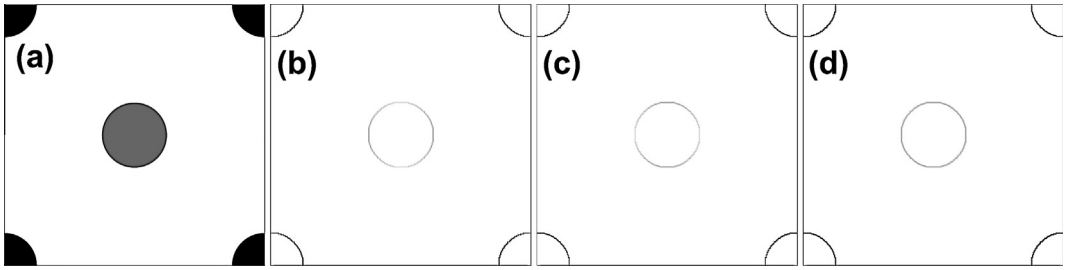

Fig. 5. (a) Square unit cell for a three material structure. Force magnitudes for $\mathbf{f}^1$, $\mathbf{f}^2$, and $\mathbf{f}^3$ are shown in (b), (c), and (d), respectively. Dark indicates large magnitude, while white indicates a zero magnitude.

of the materials in the input argument `mu` and that the input argument `lambda` is a vector of zeros. Then only minor changes to the code are necessary to compute the conductivity tensor of a composite.

In the computations of the element matrices only one line needs to be changed; line 88 should be updated to read:

```
CMu = diag([1 1 0]); CLambda = zeros(3);
```

This assures that the odd rows and columns in `keMu` contain the contributions associated with variations in potential in the x-direction (horizontal), and the even rows and columns contain the corresponding contributions for variations in the y-direction.

As mentioned, only a scalar field is necessary, this means that the element matrix is really a four times four matrix, found by summing the contributions from the odd and even rows/columns. To keep the changes in the code to a minimum, this four times four matrix is put into the odd rows and columns of `keMu` by adding the below line after the call to `elementMatVec` in line 17:

```
keMu(1:2:end,1:2:end) = keMu(1:2:end,1:2:end) + ...
    keMu(2:2:end, 2:2:end);
```

Now the remaining parts of the code should be modified to work with the odd rows and columns. The solution of the finite element problem in line 54 should be changed to:

```
chi = zeros(ndof,2);
chi(3:2:ndof,:) = K(3:2:end,3:2:end)\...
    [F(3:2:end,1) F(4:2:end,2)];
```

Similarly, the solution of the element strain cases in line 62 should be changed to:

```
chi0_e(3:2:end,1:2) = keMu(3:2:end,3:2:end)\...
    [feMu(3:2:end,1) feMu(4:2:end,2)];
```

Finally, the two loops in lines 71 and 72 should only run through indices 1 and 2.

The homogenized conductivity for the composite in Fig. 1, with high conducting circular disks/cylinders ($\mu_1 = 10$) with a radius $r = 0.125 l_x$ in a matrix with a lower conductivity ($\mu_2 = 1$), is $\mu^H = 1.175$. This fits perfectly with the analytical result derived in [14], which for a low disk volume fraction $f = 2\pi r^2$ can accurately be expressed by the low order approximation:

$$\mu^H = \mu_2 \frac{1 + 2\beta f}{1 - \beta f - 0.075422\beta^2 f^6} \tag{18}$$

with $\beta = (\mu_2 - \mu_1)/(\mu_2 + \mu_1)$.

3.4. Thermal expansion

In [15] numerical homogenization was used together with topology optimization to design a three material (two materials and void) isotropic composite with a negative thermal expansion. The composite's unit cell is pictured in Fig. 6.

The homogenized thermal stress tensor can be computed as:

$$\beta_{ij}^H = \frac{1}{|V|} \int_V E_{pqrs} \left(\alpha_{pq} - \varepsilon_{pq}^{\alpha} \right) \left(\varepsilon_{rs}^{0(ij)} - \varepsilon_{rs}^{(ij)} \right) \mathrm{d}V \tag{19}$$

where α_{pq} is the thermal expansion tensor, corresponding to a unit strain for a unit thermal load. The strain field $\varepsilon_{pq}^{\alpha}$ is computed according to Eq. (2) based on the displacement field Γ, which is found for a unit thermal load:

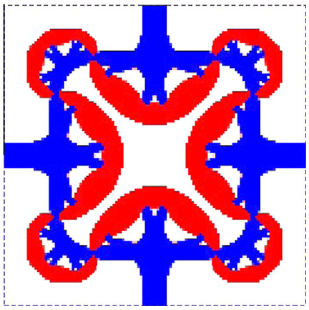

Fig. 6. A unit cell of a three material composite with macroscopic negative thermal expansion. The white part is void, modeled by setting the material stiffness to a millionth of the solid parts. The thermal expansion coefficient of the red and blue material are α_2 and α_3, respectively. The red and blue material have identical elastic properties, but $\alpha_2/\alpha_3 = 10$. Figure courtesy of Ole Sigmund, Technical University of Denmark. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

$$\int_V \varepsilon_{ij}(v) E_{ijpq} \varepsilon_{pq}(\Gamma) \mathrm{d}V = \int_V \varepsilon_{ij}(v) E_{ijpq} \alpha_{pq} \mathrm{d}V \quad \forall v \in V \tag{20}$$

Continuing from the three material extension, the homogenization function needs to be extended with an input vector (`alpha`) specifying the thermal expansion coefficient of the three materials. Furthermore, after line 43 a line assigning the corresponding element thermal expansion coefficients must be added:

```
alpha = alpha(1)*(x==1) + alpha(2)*(x==2) + alpha(3)*
   (x==3);
```

In order to compute the macroscopic thermal expansion using homogenization it is necessary to find the displacement vector $\boldsymbol{\Gamma}$ corresponding to a unit thermal strain load case:

$$\mathbf{K}\boldsymbol{\Gamma} = \mathbf{f}_{\alpha} \tag{21}$$

with the unit thermal load defined as:

$$\mathbf{f}_{\alpha} = \sum_e \int_{V_e} \mathbf{B}_e^T \mathbf{C}_e \boldsymbol{\alpha}_e \mathrm{d}V_e \tag{22}$$

where $\boldsymbol{\alpha}_e$ is the unit thermal strain corresponding to a unit increase in the temperature field. Therefore, `elementMatVec` must be extended with the computation of this element load vector. This is done by initializing an additional load vector below line 94:

```
feAlpha = zeros(8,1);
```

and adding the computation of it below line 123:

```
feAlpha = feAlpha + weight*(B.'*CMu*[1;1;0]);
```

And remember to return the element load vector from the function. It is clear that the load vector's two parts, corresponding to μ_e and λ_e, are equal and that is why only the first part is computed.

The load assembly must also be modified. Lines 51–53 are substituted with:

```
sF = [sF;feAlpha(:)*(alpha(:).*(lambda(:)+mu(:))).'];
iF = repmat(edofMat',4,1);
jF = [ones(8,nel); 2*ones(8,nel);
    3*ones(8,nel); 4*ones(8,nel)];
F = sparse(iF(:), jF(:), sF(:), ndof, 4);
```

This means that when the system is solved, the fourth displacement vector is $\boldsymbol{\Gamma}$. For sake of clarity assign this vector to its own variable by adding after line 54:

```
gamma = chi(:,4);
```

The element displacements $\boldsymbol{\Gamma}_e^0$ corresponding to the unit thermal strains must also be computed. This is done by changing the number of columns in line 59 from three to four, and adding:

```
fe = [fe 2*feAlpha];
```

below line 61, where one should remember that `feAlpha` only contains one part of the thermal strain to understand the multiplication with two. Now, $\boldsymbol{\Gamma}_e^0$ can be computed by adding after line 62:

```
gamma0 = alpha(:)*chi0_e(:,4)';
```

The entries in the homogenized thermal stress vector $\boldsymbol{\beta}$ can be found as:

$$\beta_i^H = \frac{1}{|V|}\sum_e \int_{V_e} \left(\boldsymbol{\Gamma}_e^0 - \boldsymbol{\Gamma}_e\right)^T \mathbf{k}_e \left(\boldsymbol{\chi}_e^{0(i)} - \boldsymbol{\chi}_e^i\right) \mathrm{d}V_e \tag{23}$$

This is done in the code snippet below, which can be added in the outer for-loop starting on line 71.

```
sumLambda = ((gamma0(:,:)-gamma(edofMat))*...
    keLambda).*(chi0(:,:,i)-chi(edofMat+(i-1)*ndof));
sumMu = ((gamma0(:,:)-gamma(edofMat))*keMu).*...
    (chi0(:,:,i)-chi(edofMat+(i-1)*ndof));
sumLambda = reshape(sum(sumLambda,2), nely, nelx);
sumMu = reshape(sum(sumMu,2), nely, nelx);
beta(i) = 1/cellVolume*sum(sum(lambda.*sumLambda+...
   mu. *sumMu));
```

Finally, the homogenized thermal strain vector is given as the solution to:

$$\mathbf{C}^H \boldsymbol{\alpha}^H = \boldsymbol{\beta}^H \tag{24}$$

which is solved as C^H/β in the Matlab code.

With the above extensions the thermal expansion coefficient of the composite in Fig. 6 is computed to be $-4.3\alpha_2$, which fits well with the reported $-4.2\alpha_2$ in [15]. The small discrepancy can be explained by the fact that [15] used a continuous interpolation between the three materials, while we have thresholded to get an indicator matrix with discrete entries. Furthermore, the discretization used here is based on a compressed image, and not the mesh used in [15].

4. Fluid permeability

The base program can also be extended such that the fluid permeability of a porous structure, having one solid and one fluid phase, can be computed. The permeability can be used in macroscopic porous flow simulations of slowly moving incompressible fluids using Darcy's law. Examples of materials where the permeability influences the design are discussed in e.g. [16,17].

Stokes flow is modeled within the fluid domain while the solid, no flow domain, is enforced using Brinkman penalization. The penalization parameter, ζ, is zero in the fluid domain while it takes a large value e.g. $\zeta = 10^6$ in the solid domain. Elaborating on the existing implementation a pressure degree of freedom is added to every node. In order to avoid pressure oscillations due to the even ordered elements a pressure stabilization method is applied [18]. The permeability can be computed as the volume average of the fluid velocity for Stokes flow with a prescribed unit pressure gradient. The weak form including Brinkman penalization and stabilization reads:

$$\int_V v_{i,j} u_{i,j}^k dV + \int_V v_{i,i} p dV + \int_V \zeta v_i u_i^k dV = \int_V v_i \delta_{ik} dV \quad \forall v \in V_0 \tag{25}$$

$$\int_V q u_{i,i}^k dV - \int_V \tau q_{,i} p_{,i} dV = \quad 0 \quad \forall q \in Q \tag{26}$$

where $\tau = \frac{h^2}{12}$ is the pressure stabilization coefficient with h being the longest diagonal of the element, u is the velocity field, p is the pressure and v and q are virtual velocity and pressure fields, respectively. From the velocity field the permeability can be computed as:

$$\kappa_{ik} = \frac{l^2}{|V|}\int_V u_i^k \mathrm{d}V \tag{27}$$

where l is the characteristic length of the unit cell as the permeability, opposed to the effective stiffness, is size dependent.

The equation system is slightly more complicated for flow problems and more element matrices are needed. Therefore line 17 needs to be changed to:

```
[ke, ke_brink,be,pe,le] = ...
    elementMatVec(dx/2, dy/2,phi);
```

in order to obtain the additional Brinkman (`ke_brink`), coupling (`be`), stabilization (`pe`), and load (`le`) terms. The extra element matrices also need to be implemented yielding the following changes in the `elementMatVec` function, where first the function header needs to be adjusted accordingly:

```
function [ke, ke_brink, be, pe, le] = elementMatVec (a,b,phi)
```

Extra element matrices must be initialized along with the stabilization parameter. Therefore, lines 93 and 94 should be substituted with:

```
ke = zeros(8); ke_brink = zeros(8);
be = zeros(8,4); pe = zeros(4);
le = zeros(4,1);
h2 = 4*(a^2+b^2+2*a*b*abs(cos(phi)/sin(phi)));
stab = h2/12;
```

As the pressure coupling utilize the shape functions, these are included by adding the shape function matrix `N` after line 99:

```
N = 1/4*[(1-y)*(1-x) 0 (1-y)*(1+x) 0 ...
    (1+y)*(1+x) 0 (1-x)*(1+y) 0;
    0 (1-y)*(1-x) 0 (1-y)*(1+x) ...
    0 (1+y)*(1+x) 0 (1-x)*(1+y)] ;
```

The pressure gradient matrix is computed by inserting:

```
Bp = invJ*[dNx;dNy;];
```

after line 107, while the additional element matrices and vectors can be obtained by substituting the lines 119–123 with:

```
ke = ke + weight*(B'*CMu*B);
ke_brink = ke_brink + weight*(N'*N);
be = be + weight*(B'*[1 1 0]'*N(1:4:end));
pe = pe + weight*(Bp'*Bp*stab);
le = le + weight*(N(1,1:2:end)');
```

In the main program an extra degree of freedom, assigned to the pressure, is added in every node by after line 21 to introduce the lines:

```
edofVecp = 2*(nelx+1)*(nely+1) + ...
    reshape(1*nodenrs(1:end-1,1:end-1)+1,nel,1);
edofMatp = repmat(edofVecp,1,4) + ...
    repmat([0 nely+[1 0] -1],nel,1);
```

In line 36 the total number of dofs needs to be updated (`ndof = 3*nnP;`). The dof-vector also needs an extension in order to accommodate the pressure by adding these two lines after line 35:

```
dofVector(2*nn+1:3*nn) = 2*nnP+nnPArray(:);
edofMatp= dofVector(edofMatp);
```

The design dependence and the assembly of the system matrix is altered. Due to the pressure coupling it is now a saddle-point system assembled as:

```
%% ASSEMBLE STIFFNESS MATRIX
zeta = zeta(1)*(x==1) + zeta(2)*(x==2);
K = repmat(ke(:),1,nel)+ke_brink(:)*zeta(:).';
B = repmat(be(:),nel,1);
P = repmat(pe(:),nel,1);
iK = kron(edofMat,ones(8,1))';
iB = iK(:,1:2:end);
iP = kron(edofMatp,ones(4,1))';
jK = kron(edofMat,ones(1,8))';
jB = kron(edofMatp,ones(1,8))';
jP = jB(1:2:end,:);
sA = [K(:); B; B; -P];
iA = [iK(:); iB(:); jB(:); iP(:)];
jA = [jK(:); jB(:); iB(:); jP(:)];
A = sparse(iA,jA,sA);
```

while the load vectors can be assembled by changing lines 49–52 to:

```
iF = [reshape(edofMat(:,1:2:end)',4*nel,1)' ...
    reshape(edofMat(:,2:2:end)',4*nel,1)']';
jF = [ones(4*nel,1); 2*ones(4*nel,1)];
sF = repmat(le(:),2*nel,1);
F = sparse(iF,jF,sF,3*nnP,2);
```

The system is solved constraining a single pressure degree of freedom:

```
chi = zeros(ndof,2);
solfor = 1:ndof-1; % all except last pressure dof
chi(solfor,:) = A(solfor,solfor)\F(solfor,:);
```

The homogenization is, compared to the elastic case, much simpler and reads:

```
CH = zeros(2);
CH(1,1) = sum(chi(dofVector(1:2:2*nn),1));
CH(1,2) = sum(chi(dofVector(2:2:2*nn),1));
CH(2,1) = sum(chi(dofVector(1:2:2*nn),2));
CH(2,2) = sum(chi(dofVector(2:2:2*nn),2));
CH = CH/nel;
```

Finally, substitute the two input arguments `lambda` and `mu` with `zeta`, in which the material permeabilities of the two materials are specified. Solving for the material in Fig. 1 assuming white is fluid ($\zeta = 0$) and black is impermeable material ($\zeta = 10^6$) yields the isotropic permeability $\kappa^H = 0.0207$. This compares well to $\kappa = r^2/(8c)\ (-\ln(c) - 1.476 + 2c - 1.774c^2) = 0.0209$, which is the analytical permeability prediction by [19], where r is inclusion radius and c is solid volume fraction.

5. Other extensions

Other, more involved extensions, could be multiphysics (see e.g. [20]) or optimization of the material distribution using e.g. topology optimization as first described in [21].

Extending the code's applicability to a three-dimensional unit cell is straight-forward, but imply that the computations can become very time-consuming on a personal computer. Therefore, numerical homogenization of a three-dimensional, continuum, unit cell should preferable be done in parallel using an iterative solver, such as a multigrid PCG, to solve the linear system of equations.

6. Conclusion

The Matlab implementation provided here is simple and efficient; it can quickly compute the macroscopic properties of a 2D composite material where the unit cell is finely discretized. However, if the considered unit cell is three-dimensional, a parallel implementation in e.g. Fortran or C++ is better suited. But the same principles still apply, especially the implementation of the periodic boundary conditions should be analogous, since a penalty approach or Lagrange multipliers will increase the solution time considerably. Finally, we want to mention that a 3D unit cell consisting of discrete elements, such as beam elements, can be computationally much cheaper than a continuum unit cell.

Acknowledgement

This work was funded by the Danish Research Agency through the innovation consortium F · MAT.

Appendix A. Matlab code

The code can be downloaded from http://dx.doi.org/10.1016/j.commatsci.2013.09.006.

```
function CH = homogenize(lx, ly, lambda, mu, phi, x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% lx = Unit cell length in x-direction.
% ly = Unit cell length in y-direction.
% lambda = Lame's first parameter for both materials. Two entries.
% mu = Lame's second parameter for both materials. Two entries.
% phi = Angle between horizontal and vertical cell wall. Degrees
% x = Material indicator matrix. Size used to determine nelx/nely
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% INITIALIZE
% Deduce discretization
[nely, nelx] = size(x);
% Stiffness matrix consists of two parts, one belonging to lambda and
% one belonging to mu. Same goes for load vector
dx = lx/nelx; dy = ly/nely;
nel = nelx*nely;
[keLambda, keMu, feLambda, feMu] = elementMatVec(dx/2, dy/2, phi);
% Node numbers and element degrees of freedom for full (not periodic) mesh
nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nel,1);
edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],nel,1);
%% IMPOSE PERIODIC BOUNDARY CONDITIONS
% Use original edofMat to index into list with the periodic dofs
```

```
nn = (nelx+1)*(nely+1); % Total number of nodes
nnP = (nelx)*(nely); % Total number of unique nodes
nnPArray = reshape(1:nnP, nely, nelx);
% Extend with a mirror of the top border
nnPArray(end+1,:) = nnPArray(1,:);
% Extend with a mirror of the left border
nnPArray(:,end+1) = nnPArray(:,1);
% Make a vector into which we can index using edofMat:
dofVector = zeros(2*nn, 1);
dofVector(1:2:end) = 2*nnPArray(:)-1;
dofVector(2:2:end) = 2*nnPArray(:);
edofMat = dofVector(edofMat);
ndof = 2*nnP; % Number of dofs
%% ASSEMBLE STIFFNESS MATRIX
% Indexing vectors
iK = kron(edofMat,ones(8,1))';
jK = kron(edofMat,ones(1,8))';
% Material properties in the different elements
lambda = lambda(1)*(x==1) + lambda(2)*(x==2);
mu = mu(1)*(x==1) + mu(2)*(x==2);
% The corresponding stiffness matrix entries
sK = keLambda(:)*lambda(:).' + keMu(:)*mu(:).';
K = sparse(iK(:), jK(:), sK(:), ndof, ndof);
%% LOAD VECTORS AND SOLUTION
% Assembly three load cases corresponding to the three strain cases
sF = feLambda(:)*lambda(:).'+feMu(:)*mu(:).';
iF = repmat(edofMat',3,1);
jF = [ones(8,nel); 2*ones(8,nel); 3*ones(8,nel)];
F = sparse(iF(:), jF(:), sF(:), ndof, 3);
% Solve (remember to constrain one node)
chi(3:ndof,:) = K(3:ndof,3:ndof)\F(3:ndof,:);
%% HOMOGENIZATION
% The displacement vectors corresponding to the unit strain cases
chi0 = zeros(nel, 8, 3);
% The element displacements for the three unit strains
chi0_e = zeros(8, 3);
ke = keMu + keLambda; % Here the exact ratio does not matter, because
fe = feMu + feLambda; % it is reflected in the load vector
chi0_e([3 5:end],:) = ke([3 5:end],[3 5:end])\fe([3 5:end],:);
% epsilon0_11 = (1, 0, 0)
chi0(:,:,1) = kron(chi0_e(:,1)', ones(nel,1));
% epsilon0_22 = (0, 1, 0)
chi0(:,:,2) = kron(chi0_e(:,2)', ones(nel,1));
% epsilon0_12 = (0, 0, 1)
chi0(:,:,3) = kron(chi0_e(:,3)', ones(nel,1));
CH = zeros(3);
cellVolume = lx*ly;
for i = 1:3
  for j = 1:3
    sumLambda = ((chi0(:,:,i) - chi(edofMat+(i-1)*ndof))*keLambda).*...
      (chi0(:,:,j) - chi(edofMat+(j-1)*ndof));
    sumMu = ((chi0(:,:,i) - chi(edofMat+(i-1)*ndof))*keMu).*...
      (chi0(:,:,j) - chi(edofMat+(j-1)*ndof));
    sumLambda = reshape(sum(sumLambda,2), nely, nelx);
    sumMu = reshape(sum(sumMu,2), nely, nelx);
    % Homogenized elasticity tensor
    CH(i,j) = 1/cellVolume*sum(sum(lambda.*sumLambda + mu.*sumMu));
  end
end
disp('--- Homogenized elasticity tensor ---'); disp(CH)

%% COMPUTE ELEMENT STIFFNESS MATRIX AND FORCE VECTOR (NUMERICALLY)
function [keLambda, keMu, feLambda, feMu] = elementMatVec(a, b, phi)
% Constitutive matrix contributions
CMu = diag([2 2 1]); CLambda = zeros(3); CLambda(1:2,1:2) = 1;
% Two Gauss points in both directions
xx=[-1/sqrt(3), 1/sqrt(3)]; yy = xx;
ww=[1,1];
% Initialize
keLambda = zeros(8,8); keMu = zeros(8,8);
feLambda = zeros(8,3); feMu = zeros(8,3);
L = zeros(3,4); L(1,1) = 1; L(2,4) = 1; L(3,2:3) = 1;
for ii=1:length(xx)
  for jj=1:length(yy)
    % Integration point
    x = xx(ii); y = yy(jj);
    % Differentiated shape functions
    dNx = 1/4*[-(1-y) (1-y) (1+y) -(1+y)];
    dNy = 1/4*[-(1-x) -(1+x) (1+x) (1-x)];
    % Jacobian
    J = [dNx; dNy]*[-a a a+2*b/tan(phi*pi/180) 2*b/tan(phi*pi/180)-a; ...
        -b -b b b]';
    detJ = J(1,1)*J(2,2) - J(1,2)*J(2,1);
    invJ = 1/detJ*[J(2,2) -J(1,2); -J(2,1) J(1,1)];
    % Weight factor at this point
    weight = ww(ii)*ww(jj)*detJ;
    % Strain-displacement matrix
    G = [invJ zeros(2); zeros(2) invJ];
    dN = zeros(4,8);
    dN(1,1:2:8) = dNx;
    dN(2,1:2:8) = dNy;
    dN(3,2:2:8) = dNx;
    dN(4,2:2:8) = dNy;
    B = L*G*dN;
    % Element matrices
    keLambda = keLambda + weight*(B' * CLambda * B);
    keMu = keMu + weight*(B' * CMu * B);
    % Element loads
    feLambda = feLambda + weight*(B' * CLambda * diag([1 1 1]));
    feMu = feMu + weight*(B' * CMu * diag([1 1 1]));
  end
end
```

References

[1] S.J. Hollister, Nat. Mater. 4 (2005) 518–524.
[2] J.M. Kemppainen, S.J. Hollister, J. Biomed. Mater. Res. Part A 94 (2010) 9–18.
[3] M. Dias, P. Fernandes, J. Guedes, S. Hollister, J. Biomech. 45 (2012) 938–944.
[4] A. Bensoussan, J.L. Lions, G. Papanicolaou, Asymptotic Analysis for Periodic Structures, North-Holland, 1978.
[5] E. Sanchez-Palencia, Lecture Notes in Physics, vol. 127, Springer-Verlag, 1980.
[6] S. Torquato, Random Heterogeneous Materials/Microstructure and Macroscopic Properties, Springer, New York,NY, 2002.
[7] J. Guedes, N. Kikuchi, Comput. Methods Appl. Mech. Eng. 83 (1990) 143–198.
[8] B. Hassani, E. Hinton, Comput. Struct. 69 (1998) 707–717.
[9] B. Hassani, E. Hinton, Comput. Struct. 69 (1998) 719–738.
[10] B. Hassani, E. Hinton, Comput. Struct. 69 (1998) 739–756.
[11] A.R. Diaz, A. Bernard, Int. J. Numer. Methods Eng. 57 (2003) 301–314.
[12] R.D. Cook, D.S. Malkus, M.E. Plesha, R.J. Witt, Concepts and Applications of Finite Element Analysis, fourth ed., John Wiley and Sons, 2002.
[13] E. Andreassen, A. Clausen, M. Schevenels, B.S. Lazarov, O. Sigmund, Struct. Multidisc. Optim. 43 (2011) 1–16.
[14] W.T. Perrins, D.R. McKenzie, R.C. McPhedran, Proc. Roy. Soc. Lond. A: Math. Phys. Sci. 369 (1979) 207–225.
[15] O. Sigmund, S. Torquato, J. Mech. Phys. Solids 45 (1997) 1037–1067.
[16] J.K. Guest, J.H. Prévost, Int. J. Solids Struct. 43 (2006) 7028–7047.
[17] C.S. Andreasen, O. Sigmund, Struct. Multidisc. Optim. 43 (2011) 693–706.
[18] T.J.R. Hughes, L.P. Franca, Comput. Methods Appl. Mech. Eng. 65 (1987) 85–96.
[19] J. Drummond, M. Tahir, Int. J. Multiphase Flow 10 (1984) 515–540.
[20] Q. Yu, J. Fish, Int. J. Solids Struct. 39 (2002) 6429–6452.
[21] O. Sigmund, Int. J. Solids Struct. 31 (1994) 2313–2329.