materials

MDPI

Article

Numerical Homogenization of Orthotropic Functionally Graded Periodic Cellular Materials: Method Development and Implementation

Behnam Shahbazian, Victor Bautista Katsalukha and Mirmilad Mirsayar *

Department of Aerospace, Physics, and Space Sciences, Florida Institute of Technology, Melbourne, FL 32901, USA; bshahbazian2021@my.fit.edu (B.S.); vbautista2016@my.fit.edu (V.B.)
* Correspondence: mmirsayar@fit.edu

Abstract: This study advances the state of the art by computing the macroscopic elastic properties of 2D periodic functionally graded microcellular materials, incorporating both isotropic and orthotropic solid phases, as seen in additively manufactured components. This is achieved through numerical homogenization and several novel MATLAB implementations (known in this study as *Cellular_Solid*, *Homogenize_test*, *homogenize_ortho*, and *Homogenize_test_ortho_principal*). The developed codes in the current work treat each cell as a material point, compute the corresponding cell elasticity tensor using numerical homogenization, and assign it to that specific point. This is conducted based on the principle of scale separation, which is a fundamental concept in homogenization theory. Then, by deriving a fit function that maps the entire material domain, the homogenized material properties are predicted at any desired point. It is shown that this method is very capable of capturing the effects of orthotropy during the solid phase of the material and that it effectively accounts for the influence of void geometry on the macroscopic anisotropies, since the obtained elasticity tensor has different E_1 and E_2 values. Also, it is revealed that the complexity of the void patterns and the intensity of the void size changes from one cell to another can significantly affect the overall error in terms of the predicted material properties. As the stochasticity in the void sizes increases, the error also tends to increase, since it becomes more challenging to interpolate the data accurately. Therefore, utilizing advanced computational techniques, such as more sophisticated fitting methods like the Fourier series, and implementing machine learning algorithms can significantly improve the overall accuracy of the results. Furthermore, the developed codes can easily be extended to accommodate the homogenization of composite materials incorporating multiple orthotropic phases. This implementation is limited to periodic void distributions and currently supports circular, rectangular, square, and hexagonal void shapes.

Keywords: periodic functionally graded cellular materials; 2D numerical homogenization; MATLAB code; elasticity tensor; orthotropic materials; isotropic materials

Citation: Shahbazian, B.; Bautista Katsalukha, V.; Mirsayar, M. Numerical Homogenization of Orthotropic Functionally Graded Periodic Cellular Materials: Method Development and Implementation. *Materials* **2024**, *17*, x. https://doi.org/10.3390/xxxxx

Academic Editor: Gaetano Giunta

Received: 6 November 2024
Revised: 3 December 2024
Accepted: 9 December 2024
Published: date

1. Introduction

The term "cellular structures" is quite descriptive, denoting a medium consisting of a void and solid material (i.e., the matrix), where each void is encased by a solid framework called a cell [1]. Cellular structures have a wide range of applications, including in biomedicine [2–4], aerospace [5–7], civil [8–10], and automotive industries [11–13], to name but a few [14–23]. While traditional cellular structures consist of uniform patterns (i.e., uniform cell densities), they can be constructed with spatially varying shapes, sizes, and cell orientations to achieve optimized performance in regard to various different types of applications and to achieve the desired combination of properties, such as high strength [24], lightweight [25], effective heat dissipation [26], etc. [27–34]. The constitutive

response and optimal design of cellular structures with uniform and spatially varying patterns have been investigated by numerous researchers in the past [35–44]. Spatially graded cellular structures can be found in nature with different scales, ranging from nanometers to meters. Examples are diatoms [45], butterfly wings [46], grass stems [47], dentin [48], sea urchin spines [49], and bone [50]. Such spatially graded natural patterns have inspired many researchers during the design and building of optimal artificial components, by studying the structural response at different levels of hierarchy [51–59]. For example, an artificial bone can be constructed by mimicking the hierarchical porous structure of a bone at different scale lengths and can be further optimized in regard to different loading conditions, using various optimization techniques. Recent advances in additive manufacturing (AM) techniques have enabled researchers to precisely build cellular structures with different dimensions using different shapes, orientations, and cell sizes, ranging from microns to meters [60–69], allowing the micromechanical constitutive behavior of micro-sized cells to be tailored in regard to the macroscopic structural response. However, it is well known that the manufacturing process (e.g., the printing direction) can significantly affect the mechanical response of the component [70–76]. As a result, additively manufactured spatially graded microcellular structures may exhibit highly anisotropic behavior, as a result of both the geometrical configurations (cell pattern) and the material properties (printing direction).

The analysis of cellular structures is indeed a significant challenge, due to the inherent intricacy of such materials. One way to avoid this complex task is to utilize homogenization methods, according to which the original non-continuous structure is equalized with a homogeneous analogous medium, where both structures exhibit the same macroscopic material properties. In regard to this technique, the heterogeneous structure is divided into small parts, known as representative volume elements (RVEs), and analyzed to determine their material properties. Then, the obtained properties are integrated and then averaged to form a continuous medium, with the same overall material behavior as the original non-homogenous one. In other words, homogenization is a bridge to cover the gap between the microscale behavior of cellular materials and the macroscale requirements of engineering applications [77].

Several methods have been proposed and utilized for numerical homogenization. These methods include, but are not limited to, Bloch's theorem and the Cauchy–Born hypothesis [78], where the former theorem is used to describe the wave behavior in periodic structures and the latter associates the deformation of a crystal lattice with the macroscopic strain in the material, and, together, they determine the relationship between the microscale and macroscale performance of the medium. Some popular approaches include micropolar theory [79–81], which extends classical continuum mechanics to account for microstructural effects by considering the microscopic rotation of the particles within the material; the strain energy equivalence method [82,83], which equates the strain energy in the microstructure of the material with that of an equivalent homogeneous medium; the beam theory approach [84–86], which utilizes the principles of beam mechanics by simplifying the structure into a sequence of beams and, finally, computes the overall properties based on the properties and arrangement of these beams; the multi-scale homogenization method [87], which integrates the material behavior information obtained from across the microscale to the macroscale in order to make a more accurate approximation of the overall properties of the medium; the machine learning approach [88], which uses algorithms to foresee the homogenized properties of a material by using microstructural data; and the asymptotic homogenization (AH) approach [89], which is a mathematical technique that uses asymptotic expansions to estimate the macroscale behavior of the material. If the homogenization equation is discretized and solved using finite element analysis (FEA) or other numerical methods, it is commonly called numerical homogenization. The advantages of this approach are abundant. For example, this procedure can be conducted on a wide range of materials and various microstructures with anisotropy or different complexities (whether they are periodic or non-periodic) and it can

be easily customized or integrated with other methods. Andreassen and Andreasen [90] used the theory of homogenization and presented a MATLAB R2023a code to calculate the macroscopic elasticity tensor of two- or multi-material systems made of *isotropic* materials (where one of the materials could be void) with *uniform* patterns of voids. Also, they described and extended their code for the homogenization of fluid permeability, thermal expansion, and conductivity. Later, Dong et al. [91] used this code and expanded it to achieve a homogenized constitutive matrix of 3D cellular materials or multi-material composites.

In this work, several MATLAB codes are developed to obtain the homogenized material properties of microcellular materials with *functionally graded void patterns*, made from *both isotropic and orthotropic materials*. To this end, first, a separate code is developed to build the desired periodic functionally varying cellular structure with different void shapes (circular, hexagonal, and rectangular/square). Then, numerical homogenization is adopted to compute homogenized elasticity tensors assigned to the centroid of each unit cell. If the material in the solid phase is isotropic, the approach presented in [90] is taken (herein known as the "reference elasticity tensor"). However, the MATLAB implementation in [90] cannot consider material anisotropy. To address this deficiency, a new homogenization code was developed herein, which includes a parameter called the *printing angle*. The overall microcellular structure is achieved by stacking unit cells, where each unit cell is treated as a point. A fit function is then assigned to create a continuous surface from the discrete material points (centroids of unit cells). This approach is advantageous because it allows for the prediction of the elasticity tensor at other points in the homogenized material domain. By comparing the reference elasticity tensor and the predicted one, the accuracy of the current computations is evaluated. If the material in the solid phase is isotropic, the outputs are the reference and the predicted elasticity tensors (at any desired points), the fit function that maps the entire medium and its coefficients for each elasticity tensor component, the overall average element-wise percentage error, and the plots of the microcellular structure and the corresponding relative density. If the material in the solid phase is orthotropic, the outputs are the same but for both global coordinates and axes of orthotropy and the plots of material properties of the entire domain. Note that the developed codes are well capable of considering macroscopic anisotropy, whether it comes from the orthotropic behavior of the solid phase or the geometry of the unit cell. Also, this methodology is only suitable for periodic void patterns since non-periodic patterns lack a repeating unit cell that can serve as a representative volume element (RVE). Without a well-defined RVE, it becomes challenging to homogenize the material properties accurately, as the microstructural variations cannot be captured by a single representative sample. If non-periodic void patterns are in mind, alternative modeling approaches such as direct numerical simulations or stochastic homogenization methods can be utilized, which are more complex and often increase the computational cost. Furthermore, it is worth mentioning that the presented MATLAB implementation can be used in the homogenization of composite materials with more than one orthotropic phase. At the end, some examples are solved, and the accuracy of the numerical implementation and its capabilities are addressed. Future work could focus on incorporating advanced materials (such as nanoparticles and multifunctional composites [92,93]) or materials that exhibit non-linear behavior, into homogenization frameworks.

2. Theoretical Framework and MATLAB Implementation

By assuming that the size of the unit cell is significantly smaller than the entire cellular structure (i.e., microcellular materials) and the bonding between different length scales/materials is perfect, the theory of elasticity describes the macroscopic stiffness tensor C_{ijkl}^{H} using the following equation:

$$C_{ijkl}^{H} = \frac{1}{|V|}\int_{V} C_{pqrs}\left(\varepsilon_{pq}^{0(ij)} - \varepsilon_{pq}^{(ij)}\right)\left(\varepsilon_{rs}^{0(kl)} - \varepsilon_{rs}^{(kl)}\right)dV, \quad (1)$$

where $|V|$, C_{pqrs} and $\varepsilon_{pq}^{0(ij)}$ are the volume of the unit cell, the locally varying stiffness tensor, and the prescribed macroscopic strain fields, respectively [90,94]. Moreover, $\varepsilon_{pq}^{(ij)}$ is the locally varying strain fields, which are defined as follows:

$$\varepsilon_{pq}^{(ij)} = \varepsilon_{pq}(\chi^{ij}) = \frac{1}{2}(\chi_{p,q}^{ij} + \chi_{q,p}^{ij}). \quad (2)$$

The displacement fields (χ^{kl}) can be found by solving the following equation:

$$\int_V C_{ijpq}\varepsilon_{ij}(v)\varepsilon_{pq}(\chi^{kl})dV = \int_V C_{ijpq}\varepsilon_{ij}(v)\varepsilon_{pq}^{0(kl)}dV \qquad \forall v \epsilon V \quad , \quad (3)$$

where v is the virtual displacement field. Generally, Equation (3) is solved numerically by discretizing the cell domain. To this end, both the left-hand side (i.e., the stiffness matrix) and the right-hand side (i.e., the mechanical force vectors due to the macroscopic unit strains ε^0) of this equation need to be discretized. To calculate displacement fields in Equation (3) by using FE methods, we proceed as follows:

$$\boldsymbol{K\chi} = \boldsymbol{F}, \quad (4)$$

where $\boldsymbol{K}$ is the stiffness matrix and $\boldsymbol{F}$ is the mechanical force vector due to the corresponding macroscopic unit strains, where, herein, the strains are chosen to be:

$$\varepsilon_1^0 = (1,0,0)^T, \varepsilon_2^0 = (0,1,0)^T, \varepsilon_3^0 = (0,0,1)^T. \quad (5)$$

Equation (5) physically means that in the first case, the unit strain is applied along the x-axis; in the second case, the unit strain is applied along the y-axis; and the last case corresponds to a pure shear strain. Consequently, the force vectors can be calculated by:

$$\boldsymbol{F} = \sum_{e=1}^{N} \int_{V_e} \boldsymbol{B}_e^T \boldsymbol{C}_e \boldsymbol{\varepsilon}^0 dV_e, \quad (6)$$

where N, $\boldsymbol{B}_e$, $\boldsymbol{C}_e$, and V_e are the total number of elements in a unit cell, the element strain displacement matrix, the element stiffness matrix, and the volume of the element, respectively.

Regarding the left-hand side of Equation (4), for the stiffness matrix, we have:

$$\boldsymbol{K} = \sum_{e=1}^{N} \int_{V_e} \boldsymbol{B}_e^T \boldsymbol{C}_e \boldsymbol{B}_e dV_e \, , \quad (7)$$

where the element stiffness matrix ($\boldsymbol{C}_e$) for an isotropic material depends on Lame's first and second parameters and they can be, respectively, found using Equations (8) and (9) as follows:

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \quad (8)$$

$$\mu = \frac{E}{2(1+\nu)}, \quad (9)$$

where E is the Young's modulus and ν is the Poisson's ratio. If the plane stress conditions are in mind, Lame's first parameter can be modified and used as:

$$\hat{\lambda} = \frac{2\mu\lambda}{\lambda+2\mu}. \quad (10)$$

The mentioned procedure is often referred to as numerical homogenization and has been widely utilized by various researchers in the past [90–94]. Figure 1 shows two examples of different 2D periodic cellular patterns, where each cell can be described by the parallelogram-shaped unit cells. Note that the shape of the unit cell is greatly influenced by the overall structure of the cellular domain. The unit cell should be designed to best capture the periodic repetition of the voids (see Figure 1)

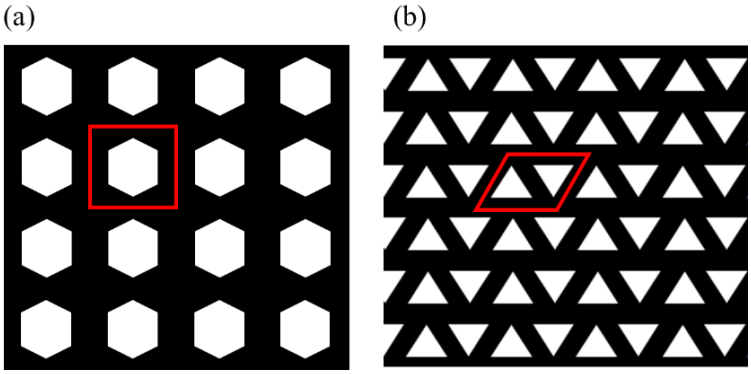

Figure 1. Examples of 2D periodic cellular patterns with parallelogram-shaped unit cells containing (**a**) hexagonal and (**b**) triangular voids.

In this approach, the unit cell itself will be discretized and then solved by finite element (FE) methods. Figure 2 illustrates the structure of the FE mesh and its corresponding geometrical constraints, together with the actual meshed unit cell consisting of two material phases. In this case, an indicator matrix X denotes whether the element contains material one ($X_e = 1$) or material two ($X_e = 2$). As can be seen in this figure, the unit cell can conveniently be characterized by three geometrical parameters of width (l_x), height (l_y), and the angle (φ) between the x-axis and the left wall of the unit cell. To avoid overly distorted elements, a range of 45°≤ φ ≤ 135° is usually recommended [90].

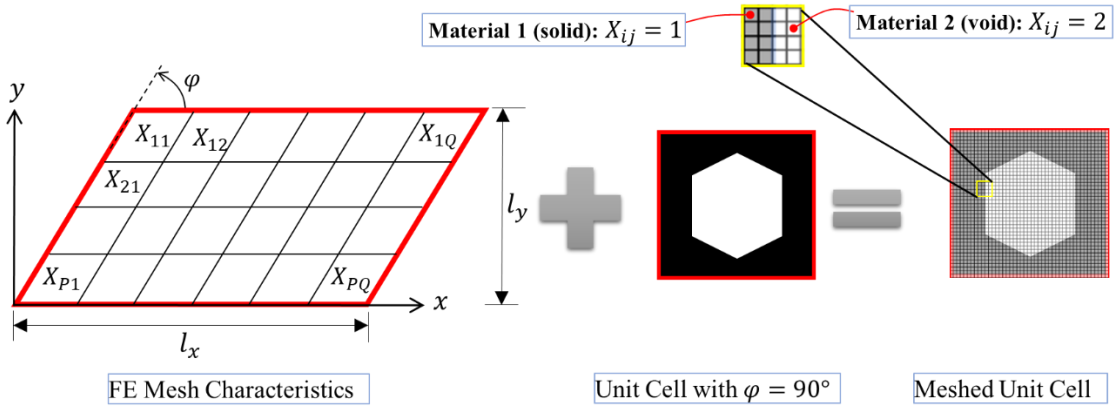

Figure 2. The structure of the FE mesh and its application on a unit cell consisting of two material phases.

As discussed earlier, Andreassen and Andreassen [90] have developed a MATLAB code for the homogenization of composite materials, assuming that the constituent phases are *isotropic*. A comprehensive explanation and possible extensions of the code are thoroughly addressed in [90], so further explanations on this matter are avoided herein for the sake of brevity. However, for the reader's convenience, the code is reported in Appendix A by the name of *homogenize*. The inputs for this code are the dimensions of the unit cell (i.e., l_x and l_y); Lame's first and second parameters for the materials (in this work, the focus is on the homogenization of functionally graded periodic cellular materials so one material is associated with the solid part of the structure and the other one is void); the angle formed by the left wall of the unit cell and the x-axis (i.e., φ); and the indicator matrix, X, which has two purposes. Firstly, it in indicates what material is within the unit cell, and, secondly, the size of this matrix characterizes how fine the discretization is (see Figure 2). Thus, the function can be called (note that "flag" is just a debugging parameter):

```
homogenize(lx, ly, lambda, mu, phi, x, flag).
```

In the current work, two new sets of codes (functions) are developed to model an isotropic cellular structure and to appropriately utilize the *homogenize* code to obtain the corresponding elasticity tensor of the homogenized material domain at any point. The first developed code is called *Cellular_Solid* (see Appendix B), which is responsible for making the unit cell with the desired void and requires three inputs, including the size of the mesh grid (i.e., the size of indicator matrix), the shape of the void, and an "Argument", which specifies the size of the void. In the present work, the void shapes that this code supports are circles, hexagons, squares, and rectangles. The variable (or the argument) that determines the size of the circular or the hexagonal void is the radius and half of the hexagon's diagonal, respectively. Changing these parameters will alter the size of the voids, much like using a magnifying tool. However, for the rectangle, two variables of width and height are required, and if the two are equal, a square is expected. The second code generated herein is called *Homogenize_test* (see Appendix C), and it properly and efficiently uses the other two codes to construct an isotropic periodic cellular structure to collect elasticity tensor data from all discrete points of the homogenized structure to create a continuous function that can estimate the tensor components at any point within the homogenized domain. The first part of this code is designated to initialization and collects four user inputs, including unit cell's matrix size, the unit cell's void shape, and the dimensions of the overall structure. Then, it establishes the grid parameters to define the range for x and y coordinates according to the input dimensions of the structure. Each x and y coordinate pair represents the centroid of a unit cell and can be used as a variable for the argument which identifies the size of the void. This is beneficial, because by assigning a function to the corresponding argument, different void sizes can be achieved by moving from one material point to another. Then, the code saves two main data for each point. The first one is the reference elasticity tensor for the corresponding unit cell obtained from the *homogenize* function. The second one is relative density, obtained by analyzing the material distribution within each unit cell by computing the sum of elements in the matrix X that contains material one (in this case, solid material) divided by the total number of elements. Afterwards, the final structure and the corresponding relative density are plotted. Then, both the relative density and the elasticity tensors are curve fitted by using a predefined function in MATLAB (in this case, "poly55"). Also, the overall average element-wise percentage error for the obtained tensor function is calculated by first obtaining the percentage error for each tensor element by using the following equation:

$$Percentage\ Error = \left|\frac{Reference\ value - Predicted\ value}{Reference\ value + \epsilon}\right| \times 100, \quad (11)$$

where ϵ is a small constant added to avoid division by zero. Then, these individual percentage errors are summed across all tensor elements for all unit cells and, finally, divided by the total number of material points. It is worth noting that in the current work, poly55 is used for its simplicity, computational efficiency, acceptable fitting accuracy, and convenience in polynomial surface fitting. Obviously, any other fitting function can be used herein, including user-developed ones. Note that for complex geometries, it is recommended to use other methods like spline fitting instead of using higher-degree polynomials, since it might result in computational overhead or overfitting. Moreover, if desired, the reference and the fitted elasticity tensors at any specified point can be displayed as an output by hard coding the coordinates of that point.

Recently developed, complex, and ultra-precise additive manufacturing methods have enabled the creation and utilization of unique microcellular structures with functionally graded patterns and tailored mechanical properties. The homogenization of such topologically complex components is a crucial task for analyzing and predicting their behavior under various loading conditions. Due to the nature of this manufacturing process which is performed layer by layer at a specified angle, the final structure often exhibits significant orthotropy in the printing direction [94]. Despite the valuable contributions of Andreassen and Andreassen [90] and although their method is capable of considering

anisotropy induced by cell topology, it fails to consider orthotropic material phases in the homogenization approach. Therefore, their approach may not be suitable for additively manufactured microcellular structures, which are the focus of this work. Figure 3a,b show two examples of anisotropies caused by the topology of the cell and the printing direction, respectively.

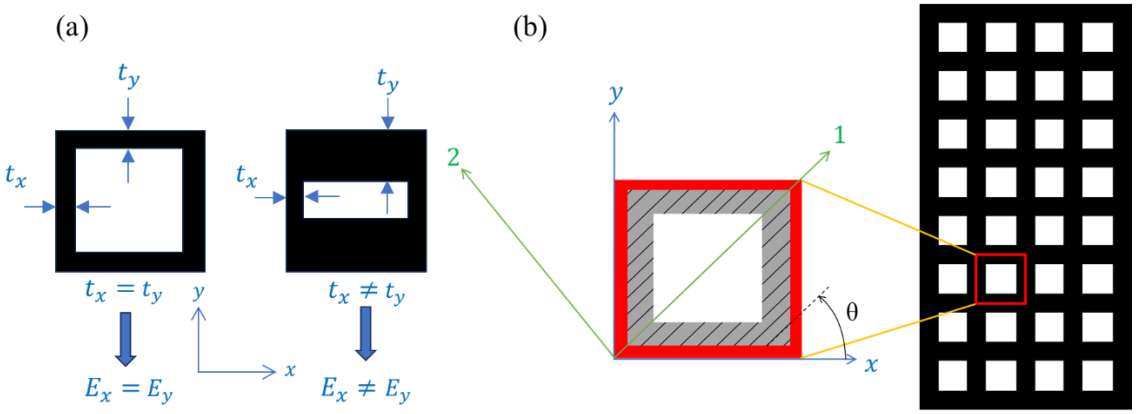

Figure 3. (**a**) An example of geometrically anisotropic cell, (**b**) An orthotropic periodic cellular structure featuring square voids, printing angle θ, with global x-y and principal 1–2 coordinate systems.

Up to this point, the focus of the current method has been on the homogenization of isotropic functionally graded periodic cellular structures, for which two new sets of codes have already been provided. However, the core novelty and innovation of the present work lie in developing a new code capable of considering material orthotropy in the homogenization process as well as spatially varying cellular patterns. The inputs of this new code, which is called *homogenize_ortho* (see Appendix D), are the same as the ones that are used in the *homogenize* function, but since the material orthotropy is considered, Lame's first and second parameters are omitted. Instead, five new inputs of E_1 (Young's modulus in first principal direction), E_2 (Young's modulus in second principal direction), G_{12} (intralaminar shear modulus), ν_{12} (the Poisson's ratio in the 2-direction due to load being applied in the 1-direction), and θ (the printing or the orthotropy angle) are added, so the final form of the line is shown below:

```
homogenize_ortho(lx, ly, E1, E2, G12, nu12, phi, theta, x, flag) .
```

After receiving the required inputs, the code transforms the material properties from principal directions to a global x-y coordinate system by using the well-known transformation equations for orthotropic materials as follows:

$$E_x = \left(\frac{\cos^4(\theta)}{E_1} + \left(\frac{1}{G_{12}} - \frac{2\nu_{12}}{E_1}\right)\sin^2(\theta)\cos^2(\theta) + \frac{\sin^4(\theta)}{E_2}\right)^{-1}, \quad (12)$$

$$E_y = \left(\frac{\sin^4(\theta)}{E_1} + \left(\frac{1}{G_{12}} - \frac{2\nu_{12}}{E_1}\right)\sin^2(\theta)\cos^2(\theta) + \frac{\cos^4(\theta)}{E_2}\right)^{-1}, \quad (13)$$

$$G_{xy} = \left(\frac{1}{G_{12}}\left(\sin^4(\theta) + \cos^4(\theta)\right) + 4\left(\frac{1}{E_1} + \frac{1}{E_2} + \frac{2\nu_{12}}{E_1} - \frac{1}{2G_{12}}\right)\sin^2(\theta)\cos^2(\theta)\right)^{-1}, \quad (14)$$

$$\nu_{xy} = E_x\left(\frac{\nu_{12}}{E_1} - \frac{1}{4}\left(\frac{1}{E_1} + \frac{2\nu_{12}}{E_2}\frac{1}{E_2} - \frac{1}{G_{12}}\right)\sin^2(2\theta)\right). \quad (15)$$

For orthotropic materials, the stiffness matrix C is commonly represented as:

$$C = \begin{bmatrix} E_x & \nu_{xy}E_y & 0 \\ \nu_{yx}E_x & E_y & 0 \\ 0 & 0 & G_{xy} \end{bmatrix}, \quad (16)$$

The transformed material properties are assigned to each element. After that, element-level stiffness matrices (keC) and force vectors (feC) are calculated based on the geometry of the unit cell. Once the boundary conditions and the global stiffness matrix and load vector are defined, the code finds the displacement field (chi) inside of the unit cell under three typical load cases of axial strain in the x-direction (epsilon0_11 = (1, 0, 0)), axial strain in the y-direction (epsilon0_22 = (0, 1, 0)), and shear strain (epsilon0_12 = (0, 0, 1)). Finally, the homogenized elasticity tensor (CH) is calculated by integrating the stress and strain fields over the volume of the unit cell, which provides a macroscopic view of how the material behaves as a continuous medium despite its microscopic heterogeneities.

To effectively use the *homogenize_ortho* function, an innovative code, named *Homogenize_test_ortho_principal,* has been developed (see Appendix E). Like before, this code makes a connection between the *Cellular_Solid* function and creates the unit cells, then stacks these cells to make the overall cellular structure. Each unit cell is represented by a point in the structure, and the corresponding elasticity tensor and relative density are assigned to that point. However, an important issue herein is that the obtained elasticity tensor from the *homogenize_ortho* function is in the global coordinate system and not in the principal directions. To solve this issue, creatively, Equations (12)–(15) are solved backwards to obtain E_1, E_2, G_{12}, and ν_{12} as follows (note that eqn1, eqn2, eqn3, and eqn4 are the same as E_x, E_y, G_{xy}, and ν_{xy}, respectively, as shown in Equations (12)–(15)):

```
eqn1 = 1./(((cosd(theta)).^4)./E1+(1./G12-2.*nu12./E1).*((sind(theta)).^2).*

((cosd(theta)).^2)+((sind(theta)).^4)./E2) == CH(1,1);

eqn2 = 1./(((sind(theta)).^4)./E1+(1./G12-2.*nu12./E1).*((sind(theta)).^2)
.*((cosd(theta)).^2)+((cosd(theta)).^4)./E2) == CH(2,2);

eqn3 = 1./(((sind(theta)).^4+(cosd(theta)).^4)./G12+4.*(1./E1+1./E2+2.*nu12./E1-
1./(2.*G12)).*((sind(theta)).^2).*((cosd(theta)).^2)) == CH(3,3);



eqn4 = CH(1,1).*(nu12./E1-1/4.*(1./E1+2.*nu12./E1+1./E2-1./G12).*
((sind(2*theta)).^2)) == (CH(1,2)/CH(2,2));

            %Solve
Sol = vpasolve([eqn1, eqn2, eqn3, eqn4], [E1,E2,G12,nu12], [CH_principal(1,1)/2;
CH_principal(2,2)/2; CH_principal(3,3)/2; CH_principal(1,2)/2]);
```

Note that the initial guesses need to be hard coded at the beginning of the code in the following lines (fill the blank):

```
CH_principal = zeros(3);
CH_principal(1,1) = ---;
CH_principal(2,2) = ---;
CH_principal(3,3) = ---;
CH_principal(1,2) = ---;
CH_principal(2,1) = CH_principal(1,2);           .
```

Now that the material properties in the principal directions are obtained, they are substituted into Equation (16) to obtain the reference elasticity tensor at each material point. The remainder of the code performs the same operations discussed earlier for the *Homogenize_test* function with the added features of plotting E_1, E_2, G_{12}, ν_{12}. Moreover, the coefficients of the polynomial fitting functions for elasticity tensors and relative density distributions are documented in CSV files. Also, note that, in this case, the tensor functions for both global x-y and principal 1–2 coordinate systems will be displayed. Figure 4 depicts a simple flowchart of the explained procedure.

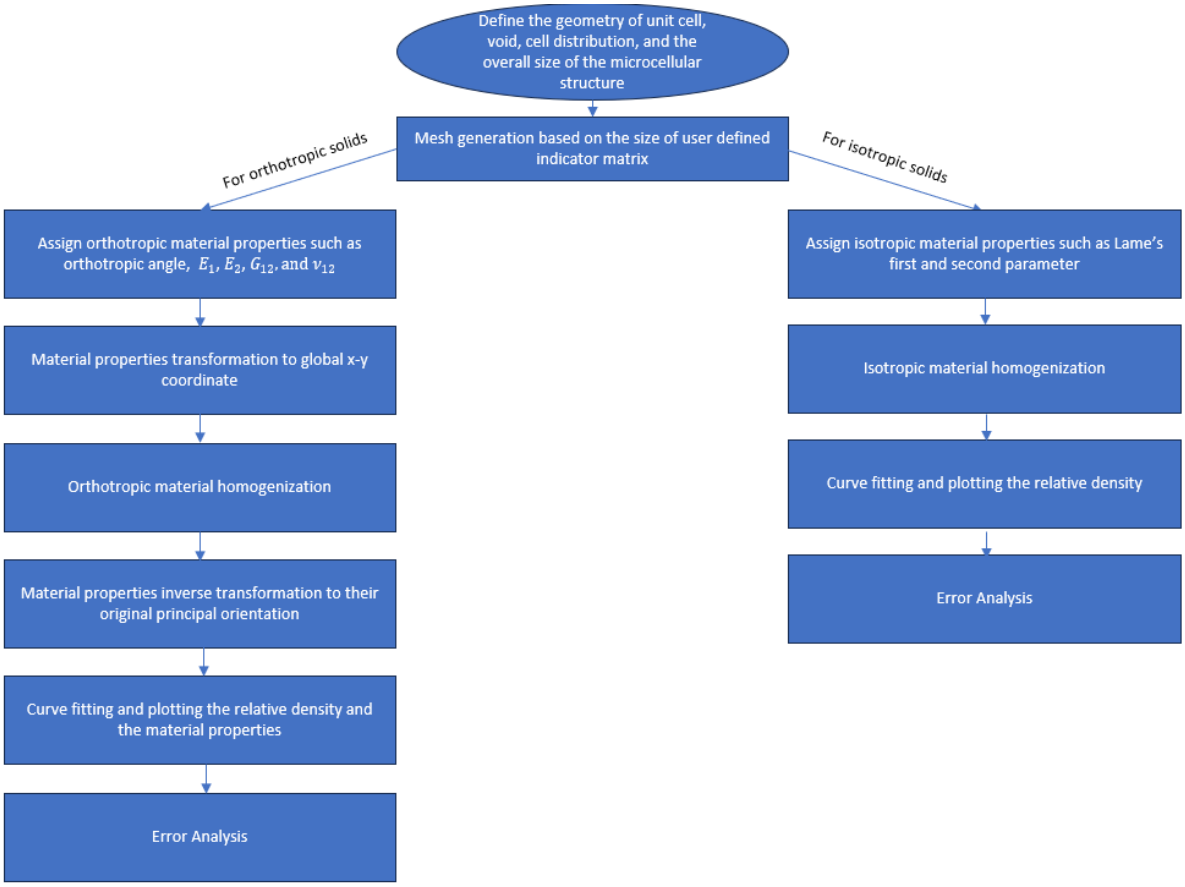

Figure 4. The flowchart of the homogenization process for orthotropic and isotropic solid phase in periodic functionally graded microcellular materials.

3. Results and Discussion

In this section, several examples of homogenizing periodic functionally graded cellular structures, considering both orthotropic and isotropic cases, are presented. For the isotropic ones, six different cases are considered. Note that before running the code, the inputs for the *homogenize* function need to be hard coded by making changes in the following line:

```
CH = homogenize(1,1,[115.4 1],[76.9 0.769],90,X,flag).
```

As mentioned, the first two inputs are l_x and l_y (the dimensions of the unit cell in x and y directions), which, in this example, are both equal to unity. The third input is a one-by-two matrix, where the first entry is Lame's first parameter for the first material (in this case, the solid phase) and the second entry is Lame's first parameter for the second material (which is void in this work). Note that, according to [90], when dealing with the void, it is recommended to use one-hundredth of the value used for the solid material. The next input is Lame's second parameter, and it follows the same rule as Lame's first parameter. Note that, herein, a hypothetical material with a Young's modulus of 200 GPa and a

Poisson's ratio of 0.3 is considered for the solid phase. Note that by substituting these values in Equations (8) and (9), Lame's first and second parameters for the solid material are computed. The fifth input is the angle between the horizontal and the inclined wall of the unit cell, φ, in degrees, which, in this example, is equal to 90°. This means that by considering the given l_x and l_y, the unit cell is a one-by-one square. Finally, X is the size of the indicator matrix used for the discretization of the unit cell, and its size will be established once the code is run, so no changes are required here. To summarize it, the hard coded inputs are l_x = 1, l_y = 1, Lame's first parameter for the solid is 115.4, Lame's first parameter for the void is 1, Lame's second parameter for the solid is 76.9, Lame's second parameter for the void is 0.769, φ = 90 (meaning the unit cell is square given that its width and height are equal to unity), and, finally, the size of the indicator matrix, which does not need to be hard coded since it will be checked once the code is run.

Once these inputs are hard coded, the *Homogenize_test* can be run. When the code is run, it asks for four inputs of UC (unit cell) matrix size, UC's void shape, the structure's width, and the structure's height. Here, a unit cell matrix size of 50 is utilized. This means that the cell will be discretized with a 50-by-50 mesh. Regarding the void shape, three cases of circle, rectangle, and hexagon have been defined, and to choose any of the aforementioned geometries, the user can simply type the name of the shape. Note that the desired function needs to be hard coded in the corresponding void size argument. The input for both the width and height of the structure is chosen to be 20. This means that the entire cellular material domain consists of 400 unit cells (and, therefore, material points) stacked together.

Figure 5 shows two functionally graded cellular structures (Figure 5a,c) with their corresponding relative density plot (Figure 5b,d). The first structure is a cellular medium with varying rectangular-shaped voids in the y-direction, while the voids in the other one have the shape of a square, and their sizes change in the diagonal direction. Both structures have simple patterns, resulting in smooth changes in material properties from one point to another. Consequently, applying a fitting function capable of giving a good approximation for the actual data (i.e., the reference elasticity tensor obtained from the *homogenize* function) will not be difficult. This is confirmed by the relatively small amounts obtained as overall average element-wise percentage error, which are 1.09% and 1.91% for the cases illustrated in Figure 5a,c, respectively.

There is an intriguing phenomenon hidden in the structure shown in Figure 5a. Even though the material of the solid phase is isotropic, the structure exhibits anisotropy at the unit cell level due to the geometry of the voids. Note that, unlike Figure 5c, the unit cells are not symmetric in both the x and y directions (cf. Figure 3a). Therefore, the unit cell returns different values of elastic modulus along the x and y directions, computed herein as E_x =161.53 GPa and E_y =128.92 GPa. Note that elongated voids aligned in the x direction make the material stiffer along that direction. A high or random discrepancy in the void sizes across different unit cells leads to significant variations in material properties, making it mathematically challenging to develop a function that can accurately predict these properties. For instance, Figure 6a depicts a structure with circular voids and a highly diverse void pattern. As can be seen in this example, the size of the voids and their position change dramatically, both pattern-wise and size-wise, so assigning a fit function will be challenging. For this reason, it is not surprising to see that the overall average element-wise percentage error in this case is 13.95%. Meanwhile, Figure 6c shows a structure with hexagonal void shapes, where slower changes in void sizes occur as we move from the center to the edges of the medium. In addition to that, the stochasticity in the pattern is much less than that of 5a, resulting in an overall average element-wise percentage error of 1.51%. The relative density for both structures is depicted in Figure 6b,d.

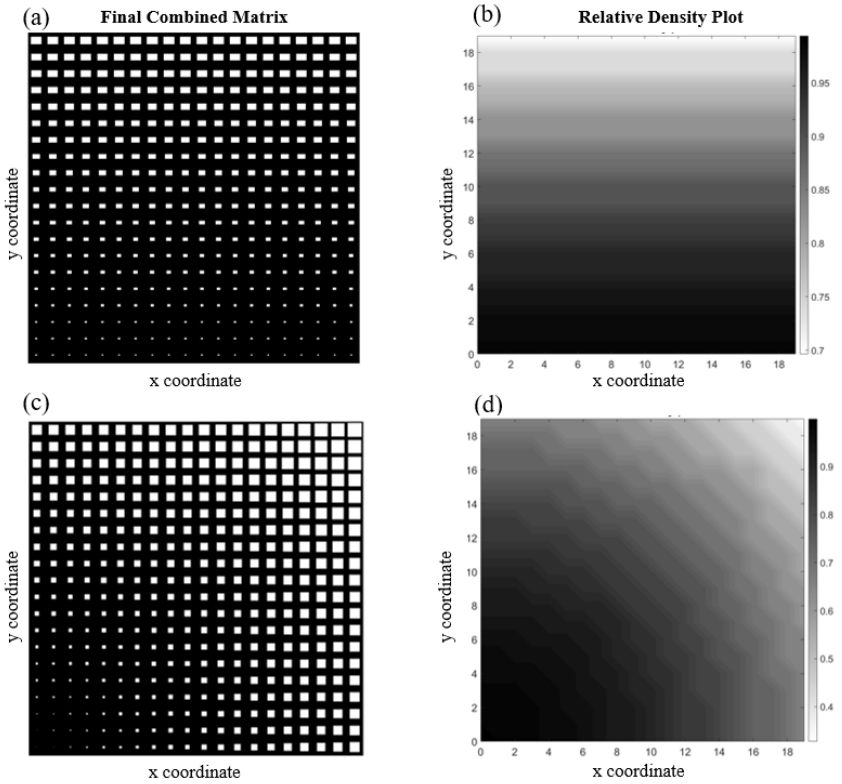

Figure 5. Two functionally graded cellular structures with (**a**) rectangular voids that increase in size in the y-direction and (**b**) its relative density plot together with (**c**) a structure with diagonally increasing square voids and (**d**) the corresponding relative density plot.

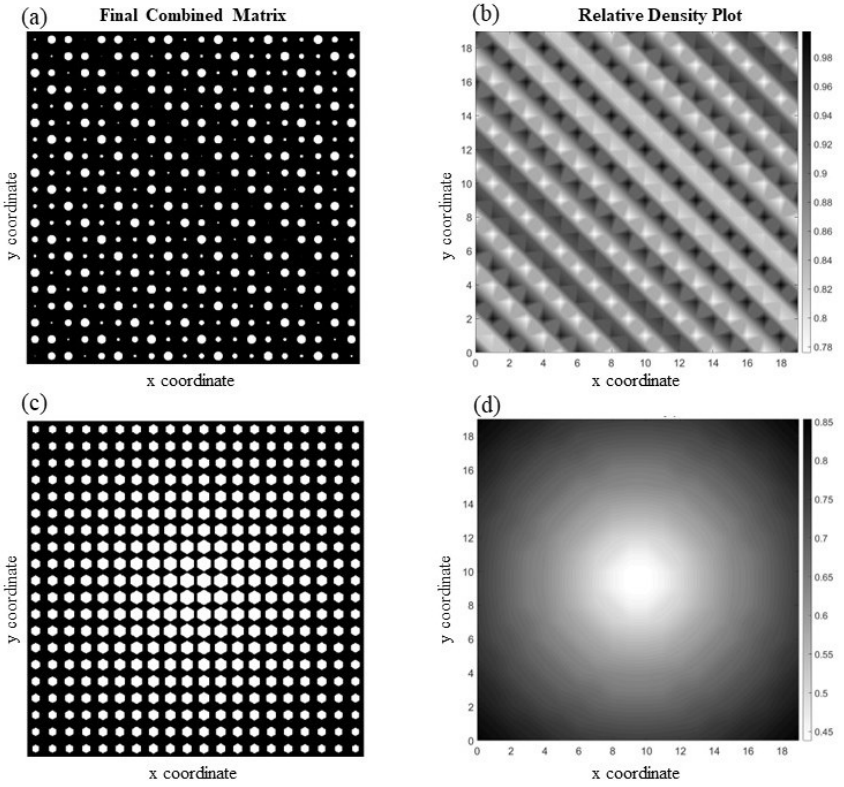

Figure 6. Two examples of functionally graded cellular structures with isotropic material phases: (**a**,**b**) a diverse circular void pattern and its corresponding relative density plot; (**c**,**d**) hexagonal voids and the corresponding relative density plot.

Two highly diverse structures in terms of void size and pattern are illustrated in Figure 7a,c. Figure 7a depicts a cellular structure with a pattern of circular voids that change diagonally, resembling the rippling effect of water waves as one droplet of water merges into another, and the structure in Figure 7c has random size voids in each unit cell. The complexity in both structures is noticeable in the density plots illustrated in Figure 7b,d. Due to the complex characteristics of the voids in these structures, it is not surprising to see that the overall average element-wise percentage error for the first structure is 7.44% and for the second one is 13.73%, which is high.

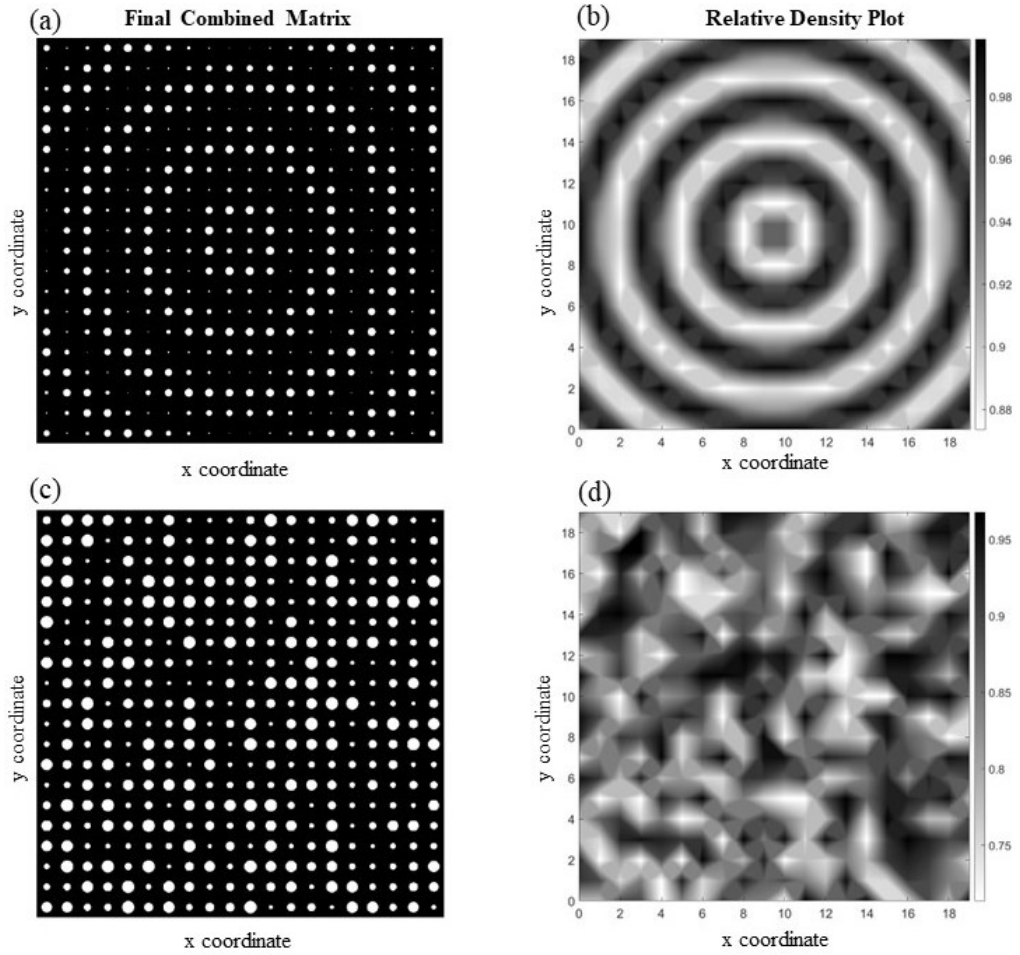

Figure 7. Examples of complex periodic microcellular structures: (**a**,**b**) illustrate a medium with circular voids that resembles a waive alongside its corresponding relative density plot; (**c**,**d**) present a structure containing random circular voids and its associated relative density plot.

For the orthotropic functionally graded periodic cellular materials, two examples are shown in Figures 8 and 9. For this part, the appropriate homogenization code, *Homogenize_test_ortho_principal*, needs to be used, and note again that, before running the code, the inputs must be hard coded as:

```
homogenize_ortho(1,1,[150 1],[9 0.01],[8 0.08],[0.3 0.3],90,theta,X,flag);.
```

Like the isotropic example, the first two inputs are the unit cell dimensions (both are set to one). The third input is a matrix where the first entry is E_1 for material one (in this case, 150 GPa) and the second entry is E_1 for material two. The fourth through sixth entries are matrices associated with E_2, G_{12}, and ν_{12}. Note that, once more, the mentioned rule is applied here, in which for the void, one-hundredth of the value used for the solid material is used. The next entries are $\varphi = 90°$, the printing angle (θ), and finally the size of

the indicator matrix. The last two values, the printing angle and matrix size, will be prompted for input from the user once the code is run, so no prior changes are required.

Figure 8 shows an orthotropic periodic microcellular structure with square voids and a printing angle of 30° (Figure 8a) along with its corresponding relative density, E_1, E_2, G_{12}, and ν_{12} plots (Figure 8b–f, respectively). Even though the patten seems to be complicated, the variation in the void size is not significant, which makes it relatively easy to fit a function capable of predicting the datapoints with acceptable precision. This is verified by knowing that the overall average element-wise percentage error for this case is 3.44%.

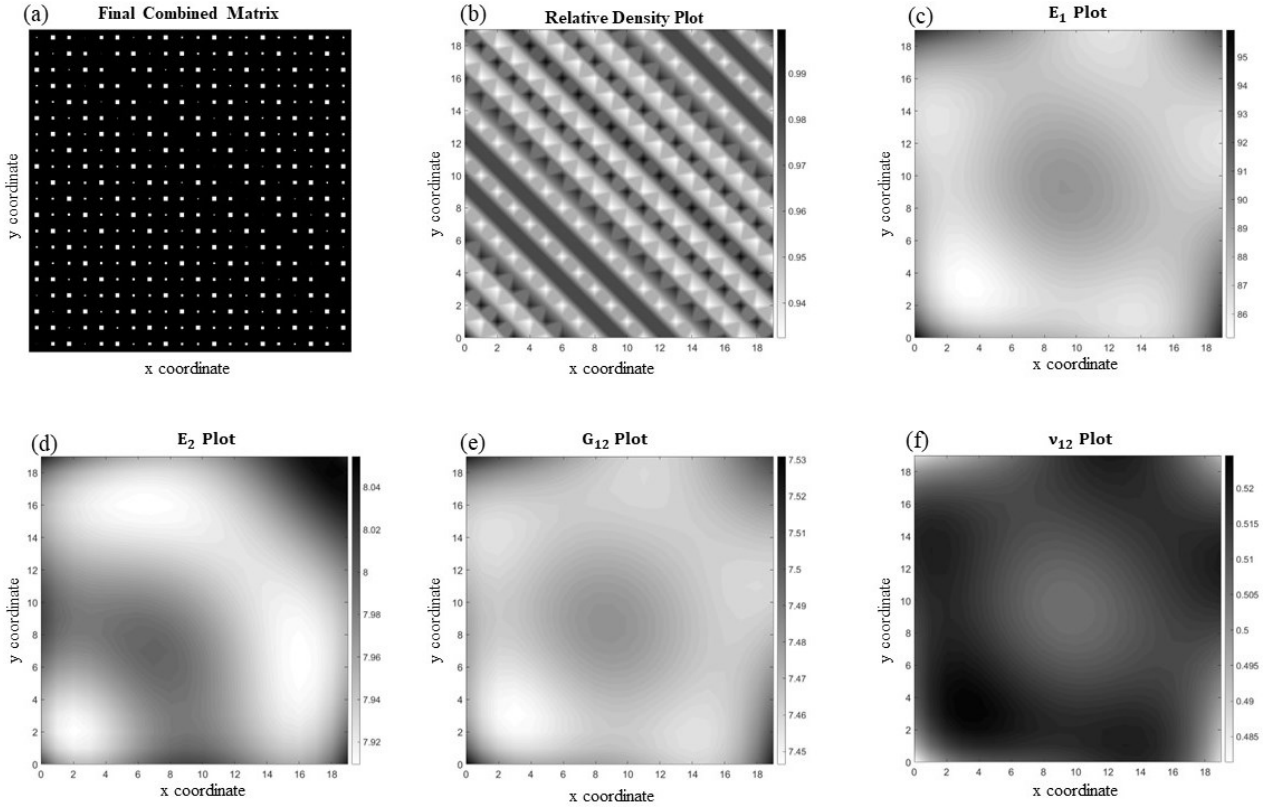

Figure 8. (**a**) A periodic functionally graded microcellular structure with orthotropy angle of 30° and its associated homogenized material properties including (**b**) relative density, (**c**) E_1, (**d**) E_2, (**e**) G_{12}, and (**f**) ν_{12}.

Figure 9 illustrates an orthotropic periodic functionally varying microcellular structure but with a more complex pattern (Figure 9a), which resembles a ripple emanating from the lower left of the medium. The orthotropy angle (i.e., printing orientation angle) is 60°, and the unit cells contain circular voids, with sizes that change more dramatically compared to the previous example. This dissimilarity leads to considerable changes in the material properties at each unit cell and consequently makes it difficult to fit a function capable of accurately capturing these variations. This results in an overall average element-wise percentage error of 6.80% for this case. The corresponding homogenized material properties are depicted in Figure 9b–f.

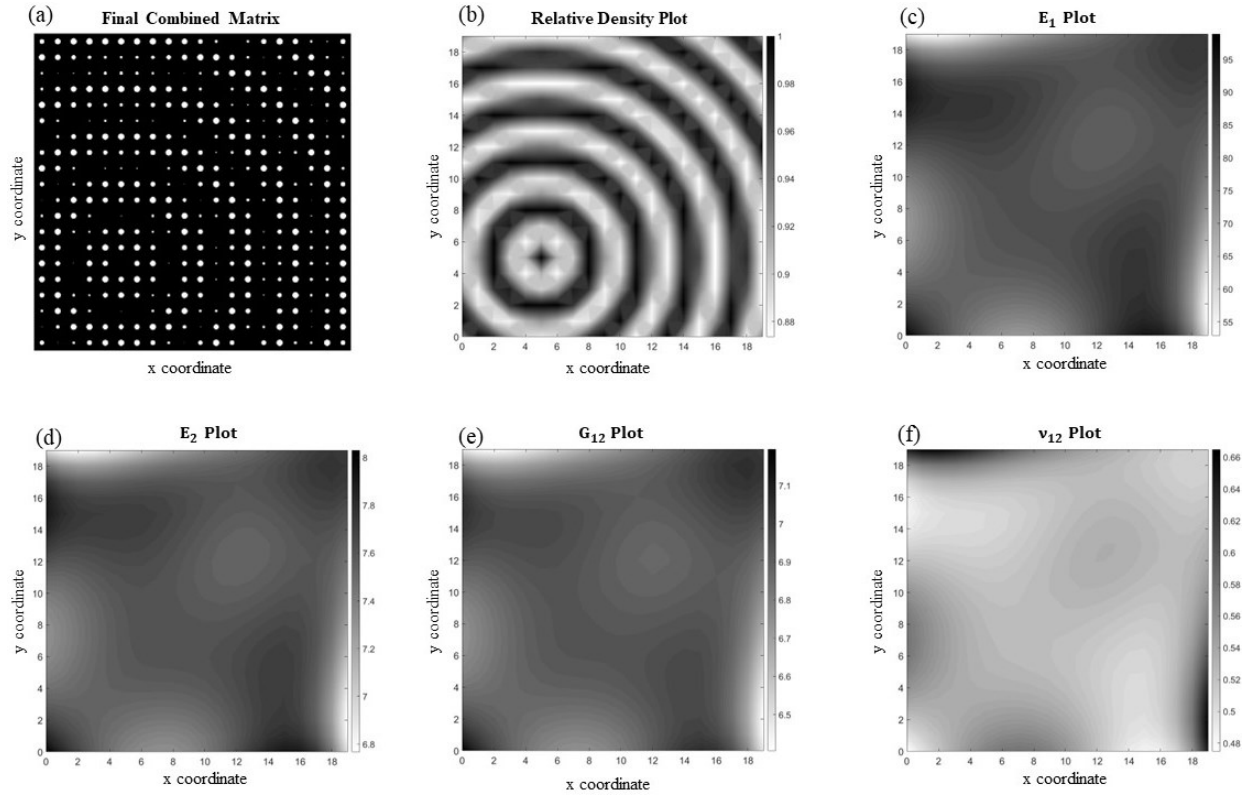

Figure 9. (**a**) A complex periodic microcellular structure with orthotropy angle of 60° and its corresponding homogenized material properties, including (**b**) relative density, (**c**) E_1, (**d**) E_2, (**e**) G_{12}, and (**f**) ν_{12}.

4. Conclusions

Recent developments in additive manufacturing have not only broadened the horizon for creating structures with intricate features (like microcellular solids) but have also accentuated the necessity of understanding and analyzing them under various loading conditions. Homogenization is a powerful computational tool for the simplification of materials with complex discrete properties into an equivalent continuum medium. This technique is vital in various fields, particularly in structural and fracture analysis, where simulating crack nucleation and propagation in complex (i.e., heterogeneous) materials is not only complicated but also computationally expensive [95–101]. In the current method, for the first time, homogenization is utilized to obtain the macroscopic elasticity tensor of functionally graded periodic cellular materials, considering both isotropic and orthotropic constituents, from the equivalent homogeneous counterparts. To this end, several MATLAB implementations are developed and presented for future users. Regarding structures with isotropic solids, two new sets of codes are developed. The first one, named *Cellular_Solid*, defines both the geometry of the unit cell and the void inside of it, while the other one, named *Homogenize_test*, has several duties. Firstly, it retrieves the unit cell from *Cellular_Solid* and then effectively utilizes a homogenization code already reported in the literature (herein by the name of *homogenize*) to obtain the homogenized unit cell's elasticity tensor. Each unit cell is then stacked up, and the final structure is generated and plotted along with its corresponding relative density. This code treats the unit cells as material points and assigns the homogenized elasticity tensor to their centroids. Finally, using a predefined MATLAB fit function, namely "poly55" in the current work, this code maps the medium and predicts the elasticity tensor at any given material point.

The primary innovation of this work lies in analyzing spatially varying cellular materials while considering the effects of material orthotropy in the solid phase, to account for the orthotropic nature of these materials fabricated via additive manufacturing. For

this case, another two sets of codes are developed. The first code is a novel homogenization code, which is capable of considering material orthotropy. Finally, the last code, which is called *Homogenize_test_ortho_principal*, makes a connection between other codes and not only plots the structure and its relative density but also illustrates E_1, E_2, G_{12}, and ν_{12} for the equivalent homogeneous medium. In both the isotropic and orthotropic cases, the overall average element-wise percentage error is reported too. This work showed that by using the MATLAB implementation provided herein, the homogenized elasticity tensors in both isotropic and orthotropic (whether it is due to the material or the asymmetrical geometry of the void) functionally graded periodic cellular materials can be obtained accurately, conveniently, and with low computational costs. One of the advantages of the current method is that each entry of the final elasticity tensor is an equation that maps the entire domain. Having an output like that is beneficial because it can be easily used in advanced fracture models such as XFEM or peridynamics. Also, it was revealed that the total accuracy of the predicted macroscopic elasticity tensor highly depends on the complexity of the void patterns and how severe the void sizes change from cell to cell. It was shown that the more complex they become, the harder it is to fit a function that can accurately represent each material point. Thus, the fitting function plays an important role in the overall accuracy; as the stochasticity in the structure increases, a more sophisticated fitting function is required. For instance, utilizing a fit function based on Furrier series might be a prudent choice since it is especially useful for modeling periodic data, due to its ability to represent any periodic function by decomposing it into a sum of sine and cosine terms. Moreover, it was revealed that as the relative density decreases (i.e., porosity increases), the material's stiffness and strength normally decrease, and the geometry-induced anisotropy becomes more pronounced due to the large void size.

Author Contributions: B.S.: Writing—review and editing, writing—original draft, visualization, methodology, investigation, formal analysis, conceptualization, coding. V.B.K.: Writing, investigation, formal analysis, methodology, conceptualization, coding. M.M.: Funding acquisition, writing—review, editing, supervision, resources, project administration, methodology, conceptualization. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Science Foundation of the United States (NSF), grant number 2317406.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data and MATLAB codes supporting the reported results are provided in the Appendix of this article. Additional data or materials can be requested from the corresponding author.

Acknowledgments: The authors would like to acknowledge the National Science Foundation of the United States (NSF), CMMI program, Mechanics of Materials and Structures (award # 2317406) for the financial support of this research.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. The *homogenize* Code Originally Reported in [90]

```
function CH = homogenize(lx, ly, lambda, mu, phi, x, flag)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 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);
if flag==1
    disp('keLambda size');
    size(keLambda)
    disp('feLambda size');
    size(feLambda)
end
% 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);
if flag==1
    disp('Lambda size');
    size(lambda)
end
% The corresponding stiffness matrix entries
sK = keLambda(:)*lambda(:).' + keMu(:)*mu(:).';
K  = sparse(iK(:), jK(:), sK(:), ndof, ndof);
if flag==1
    disp('sK');
    sK(:,1)
end
if flag==1
    disp('iK size');
    size(jK)
    disp('jK size');
    size(jK)
    disp('sK size');
    size(sK)
    disp('K size');
```

```
    size(K)
end
%% 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);
if flag==1
    disp('iF size');
    size(jF)
    disp('jF size');
    size(jF)
    disp('sF size');
    size(sF)
    disp('F size');
    size(F)
end
% 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));
if flag==1
    disp('chi size');
    size(chi)
    disp('chi0 size');
    size(chi0)
    disp('chi0_e size');
    size(chi0_e)
    disp('edofMat');
    size(edofMat)
    disp('ndof');
    ndof
end
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));
    if flag==1 && i==2 && j==2
        disp('sumLambda1 size')
        size(sumLambda)
    end
    if flag==1 && i==2 && j==2
        disp('edofMat+(i-1)*ndof size')
        size(edofMat+(i-1)*ndof)
```

```
    end
    sumMu = ((chi0(:,:,i) - chi(edofMat+(i-1)*ndof))*keMu).*...
      (chi0(:,:,j) - chi(edofMat+(j-1)*ndof));
    sumLambda = reshape(sum(sumLambda,2), nely, nelx);
    if flag==1 && i==2 && j==2
        disp('sumLambda2 size')
        size(sumLambda)
    end
    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
```

Appendix B. The *Cellular_Solid* Code

```
function largeMatrix = draw_shape(matrixSize, shape_type, varargin)
    % Initialize the large matrix with all elements as 1
    largeMatrix = ones(matrixSize);

    switch shape_type
        case 'circle'
            % varargin{1} is the radius of the circle
            circleRadius = varargin{1};

             % Initialize the current matrix with all elements as 1
                matrix = ones(matrixSize);

                % Calculate the coordinates of the center
                centerX = (matrixSize + 1) / 2;
                centerY = (matrixSize + 1) / 2;

                % Create a meshgrid of coordinates
                [X, Y] = meshgrid(1:matrixSize, 1:matrixSize);

                % Create a circular mask with 2s inside the circle
                circularMask = (X - centerX).^2 + (Y - centerY).^2 <= circleRadius.^2;

                % Update the current matrix with the circular mask
                matrix(circularMask) = 2;

                % Combine the current matrix with the large matrix
                largeMatrix = max(largeMatrix, matrix);

        case 'rectangle'
            % varargin{1} is the width of the rectangle
            % varargin{2} is the length of the rectangle
            rectangleWidth = varargin{1};
            rectangleLength = varargin{2};

            % Calculate the coordinates of the center
                centerX = floor((matrixSize + 1) / 2);
                centerY = floor((matrixSize + 1) / 2);

                % Calculate the starting and ending coordinates of the rectangle
                startX = centerX - floor(rectangleWidth / 2);
                endX = centerX + floor(rectangleWidth / 2);
```

```
        startY = centerY - floor(rectangleLength / 2);
        endY = centerY + floor(rectangleLength / 2);

        % Ensure the coordinates are within the matrix boundaries
        startX = max(1, startX);
        endX = min(matrixSize, endX);
        startY = max(1, startY);
        endY = min(matrixSize, endY);

        % Create the rectangle in the matrix
        largeMatrix(startY:endY, startX:endX) = 2;

case 'hexagon'
    % varargin{1} is half of the hexagon diagonal
    w = varargin{1};

    % % 1- Below is the hexagonal with two vertices next to each other
    % % use only this part or part 2
    % a11 = (matrixSize + 1) / 2 - w;
    % a12 = (matrixSize + 1) / 2;
    %
    % a21 = (matrixSize + 1) / 2 - w*cosd(60);
    % a22 = (matrixSize + 1) / 2 - w*sind(60);
    %
    % a31 = (matrixSize + 1) / 2 + w*cosd(60);
    % a32 = (matrixSize + 1) / 2 - w*sind(60);
    %
    % a41 = (matrixSize + 1) / 2 + w;
    % a42 =(matrixSize + 1) / 2;
    %
    % a51 = (matrixSize + 1) / 2 + w*cosd(60);
    % a52 = (matrixSize + 1) / 2 + w*sind(60);
    %
    % a61 = (matrixSize + 1) / 2 - w*cosd(60);
    % a62 = (matrixSize + 1) / 2 + w*sind(60);



    % 2- Below is the hexagonal with one vertices on top
    % use only this part or part 1
    a11 = (matrixSize + 1) / 2;
```

```
            a12 = (matrixSize + 1) / 2 - w;

            a21 = (matrixSize + 1) / 2 + w*sind(60);
            a22 = (matrixSize + 1) / 2 - w*cosd(60);

            a31 = (matrixSize + 1) / 2 + w*sind(60);
            a32 = (matrixSize + 1) / 2 + w*cosd(60);

            a41 = (matrixSize + 1) / 2 ;
            a42 =(matrixSize + 1) / 2 + w;

            a51 = (matrixSize + 1) / 2 - w*sind(60);
            a52 = (matrixSize + 1) / 2 + w*cosd(60);

            a61 = (matrixSize + 1) / 2 - w*sind(60);
            a62 = (matrixSize + 1) / 2 - w*cosd(60);



            % Specify the vertices of the hexagon (user-defined)
            hexagonVertices = [
                a11, a12;  % Vertex 1
                a21, a22;  % Vertex 2
                a31, a32;  % Vertex 3
                a41, a42; % Vertex 4
                a51, a52;  % Vertex 5
                a61, a62 ];  % Vertex 6


            % Create a binary mask for the hexagon
            hexagonMask = poly2mask(hexagonVertices(:, 1), hexagonVertices(:, 2), ma-
trixSize, matrixSize);

            % Set the corresponding elements in the matrix to 2
            largeMatrix(hexagonMask) = 2;


        case 'coordinates'
            % varargin{1} is a matrix containing the coordinates of the vertices
            vertices = varargin{1};

            % Create a binary mask for the triangle
```

```
            shapeMask = poly2mask(vertices(:, 1), vertices(:, 2), matrixSize, matrixSize);

            % Set the corresponding elements in the matrix to 2
            matrix(shapeMask) = 2;

        otherwise
            error('Unknown shape type');
    end


end
```

Appendix C. The *Homogenize_test* Code

```
clc
clear all
close all

%User inputs

matrixSize = input('Enter UC matrix size: ');
shape = input('Enter UCs void shape (circle, rectangle, hexagon): ','s');
x_length = input('Enter the structure width: ');
y_length = input('Enter the structure height: ');



% Defining grid parameters
x_range = 0:1:x_length-1;
y_range = 0:1:y_length-1;
x_center = (x_length-1)/2;
y_center = (y_length-1)/2;
RD = zeros(x_length, y_length);
% Define a structure to hold the coordinates and elasticity tensors
plate = struct('x', [], 'y', [], 'tensor', []);
RD_struct = struct('x', [], 'y', [], 'RD', []);

%Dummy flag
y_now = pi();
%Full structure visualization
FullStructure = [];
% Initialize a row holder for each row of unit cells
```

```
rowHolder = [];
flag=0;
% Loop through the grid and compute the elasticity tensor for each point
for y = fliplr(y_range)
    % Clear rowHolder for the new row
    rowHolder = [];
    for x = x_range
            switch shape
                case 'circle'


                    %study 5
                    % argument = sin(sqrt((x - x_center)^2 + (y - y_center)^2))*10;



                    %study 3
                    % argument = abs(sin(x+y))*matrixSize/4+1;
                    % if argument >= floor(matrixSize/2)
                    %     argument = floor(matrixSize/2)-1;
                    % end

                    %case 6
                    argument = randi([5, 15]);


                    X = Cellular_Solid(matrixSize, shape, argument);
                case 'rectangle'

                   %study 1
                   % argument1 = (y+3)*1.5;
                   % argument2 = y+3;

                   %study 2
                     argument1 = (sqrt(y^2+x^2))*1.5;
                     argument2 = (sqrt(y^2+x^2))*1.5;

                    X = Cellular_Solid(matrixSize, shape, argument1, argument2);
                case 'hexagon'

                    %study 4
                    argument = 24-(sqrt((y-y_center)^2+(x-x_center)^2))*0.9;
```

```
                    X = Cellular_Solid(matrixSize, shape, argument);



            end
             CH = homogenize(1,1,[115.4 1],[76.9 0.769],90,X,flag);

            flag=0;

        %Save "relative density"
        RD_struct(end+1)=struct('x', x, 'y', y, 'RD', sum(sum(X == 1))/(matrixSize^2));
        RD(x+1,y+1) = sum(sum(X == 1))/(matrixSize^2);

        % Add the data to the structure
        plate(end+1) = struct('x', x, 'y', y, 'tensor', CH);
        % Concatenate this unit cell to the row holder
        rowHolder = [rowHolder, X];
    end
    % Once a full row of unit cells is formed, concatenate it to the FullStructure
    FullStructure = [FullStructure; rowHolder];
end
plate = plate(2:end); %Get rid of first empty entry
RD_struct = RD_struct(2:end);
%% Plotting
% Calculate the range of densities
densityMin = min(RD(:));
densityMax = max(RD(:));

% Define contour levels
numLevels = 50; % You can change this value
contourLevels = linspace(densityMin, densityMax, numLevels);
figure(1)
contourf(x_range,y_range,RD',contourLevels,'LineColor', 'none')
colormap(flipud(gray));
colorbar
caxis([densityMin, densityMax]); % Set color axis scaling
% Adjusting font sizes individually
title('Relative density plot', 'FontSize', 20)  % Set font size for title
xlabel('x coordinate', 'FontSize', 20)          % Set font size for x-axis label
ylabel('y coordinate', 'FontSize', 20)          % Set font size for y-axis label
```

```
set(gca, 'FontSize', 19)                          % Set font size for axis ticks

axis equal
%Full cellular/porous structure
figure(2)
imshow(FullStructure, [1, 2]);
% Adjusting font sizes individually
title('Final Combined Matrix', 'FontSize', 20)  % Set font size for title
xlabel('x coordinate', 'FontSize', 20)          % Set font size for x-axis label
ylabel('y coordinate', 'FontSize', 20)          % Set font size for y-axis label

axis equal
%% Obtain the "reference" Elasticity tensor at a given set of coordinates
% x_coordinate = floor(x_length/2); %This gives the CH at the center of the plate
% y_coordinate = floor(y_length/2);
%Manual input
x_coordinate = 15; %This gives the CH at the center of the plate
y_coordinate = 15;
tensor = getTensorAtCoordinate(plate, x_coordinate, y_coordinate); %Uses the funnction at
the end

%% Curve fitting the Elasticity Tensor Map
% Initialize tensor_function as a cell array
tensor_function = cell(3);

for i=1:3
   for j=1:3
        x_data = [plate.x];
        y_data = [plate.y];
        tensor_data = cellfun(@(t) t(i,j), {plate.tensor});

        % Define the fit type, e.g., a polynomial
        fitType = fittype('poly55'); % second-degree polynomial

        % Perform the fit

        tensor_function{i,j} = fit([x_data', y_data'], tensor_data', fitType);
    end
end
tensor_function

%% Curve fitting the Relative Density Map
```

```
x_data = [plate.x];
y_data = [plate.y];
RD_data = [RD_struct.RD];
% Define the fit type, e.g., a polynomial
fitType = fittype('poly55'); % second-degree polynomial

% Perform the fit
%RDL = reshape(RD, [], 1);
RD_function = fit([x_data', y_data'], RD_data', fitType);
% For instance, for the 88th entry in the structure, fitted and actual RD
% are:
L = 88;
% To know its coordinates use:
% x_coordinate = plate(88).x
% y_coordinate = plate(88).y
RD_Fitted = RD_function(plate(L).x, plate(L).y)
RD_Actual = RD_struct(L).RD

%% Evaluating the curve-fitted Elasticity tensor from coordinate
%Accessing  specific equation (entry of the tensor)
specific_function = tensor_function{1,2};
%Evaluating the function at a given x and y value
x_value = 15; % Replace with the desired x value
y_value = 15; % Replace with the desired y value
result = specific_function(x_value, y_value);
%Evaluating full Elasticity tensor at a given y value
elasticity_tensor = zeros(3);
for i = 1:3
    for j = 1:3
        elasticity_tensor(i,j) = tensor_function{i,j}(x_value, y_value);
    end
end
elasticity_tensor

%% Evaluating error (fit analysis)

% Initialize variable for accumulating total element-wise percentage error
total_percentage_error = 0;
total_elements = 0;

% Loop through all coordinates in 'plate' to accumulate the percentage errors
```

```
for i = 1:length(plate)
    x = plate(i).x;
    y = plate(i).y;

    % Extract the reference tensor from 'plate'
    tensor_exact = plate(i).tensor;

    % Evaluate the approximate tensor at the same coordinates
    for m=1:3
        for n=1:3
            tensor_approx(m,n) = tensor_function{m,n}(x, y);
        end
    end

    % Compute the element-wise percentage error
    % Avoid division by zero by adding a small constant (e.g., 1e-9)
    percentage_error_matrix = abs((tensor_exact - tensor_approx)./ (tensor_exact + 1e-9)) *
100;

    % Sum up the percentage errors and count the number of elements
    total_percentage_error = total_percentage_error + sum(sum(percentage_error_matrix));
    total_elements = total_elements + numel(percentage_error_matrix);
end

% Compute the overall average element-wise percentage error
overall_avg_percentage_error = total_percentage_error / total_elements;

fprintf('The overall average element-wise percentage error is %.2f%%\n', overall_avg_per-
centage_error);


%% getTensorAtCoordinate "reference" function
%Get a specific tensor from coordinate (function)
function tensor = getTensorAtCoordinate(plate, x, y)
    for i = 1:length(plate)
        if isequal(plate(i).x, x) && isequal(plate(i).y, y)
            tensor = plate(i).tensor
            return;
        end
    end
    error('Coordinate not found in the structure.');
end
```

Appendix D. The *homogenize_ortho* Code

```
function CH = homogenize_ortho(lx, ly, E1, E2, G12, nu12, phi, theta, x, flag)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% lx        = Unit cell length in x-direction.
% ly        = Unit cell length in y-direction.
% E1        = Young's Modulus in first principal direction for both materials. Two entries.
% E2        = Young's Modulus in second principal direction for both materials. Two en-
tries.
% G12       = Shear Modulus in direction 1-2.
% nu12      = Poisson's ratio in direction 1-2.
% phi       = Angle between horizontal and vertical cell wall. Degrees
% theta     = Ply angle
% x         = Material indicator matrix. Size used to determine nelx/nely
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% INITIALIZE
% Deduce discretization
[nely, nelx] = size(x);
% Stiffness matrix for orthotropic materials
dx = lx/nelx; dy = ly/nely;
nel = nelx*nely;
% Material properties coordinate transformation (Local 12 to Global xy)
Ex = 1./(((cosd(theta)).^4)./E1+(1./G12-
2.*nu12./E1).*((sind(theta)).^2).*((cosd(theta)).^2)+((sind(theta)).^4)./E2);
Ey = 1./(((sind(theta)).^4)./E1+(1./G12-
2.*nu12./E1).*((sind(theta)).^2).*((cosd(theta)).^2)+((cosd(theta)).^4)./E2);
Gxy = 1./(((sind(theta)).^4+(cosd(theta)).^4)./G12+4.*(1./E1+1./E2+2.*nu12./E1-
1./(2.*G12)).*((sind(theta)).^2).*((cosd(theta)).^2));
%nu_xy = Ex.*(nu12./E1.*((sind(theta)).^4+(cosd(theta)).^4)-(1./E1+1./E2-
1./G12).*((sind(theta)).^2).*((cosd(theta)).^2));
nu_xy = Ex.*(nu12./E1-1/4.*(1./E1+2.*nu12./E1+1./E2-1./G12).*((sind(2*theta)).^2));

% Material properties in the different elements
Ex = Ex(1)*(x==1) + Ex(2)*(x==2);
Ey = Ey(1)*(x==1) + Ey(2)*(x==2);
Gxy = Gxy(1)*(x==1) + Gxy(2)*(x==2);
nu_xy = nu_xy(1)*(x==1) + nu_xy(2)*(x==2);
%Linearize
Ex=Ex(:); Ey=Ey(:); Gxy=Gxy(:); nu_xy=nu_xy(:);
```

```
% Construct the orthotropic stiffness matrix C for each element
C = zeros(nel, 3, 3);
C(:,1,1) = Ex;
C(:,1,2) = nu_xy .* Ey;
C(:,2,1) = (Ey./Ex) .* nu_xy .* Ex;
C(:,2,2) = Ey;
C(:,3,3) = Gxy;

%Elements Stiffness Matrix and Load Vector
[keC, feC] = elementMatVecOrtho(dx/2, dy/2, phi, C, nel, flag);

% 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))';

% The corresponding stiffness matrix entries for orthotropic materials
sK = reshape(keC, [64, nel]);  % keC should already account for E1, E2, G12, and nu12
K  = sparse(iK(:), jK(:), sK(:), ndof, ndof);

%% LOAD VECTORS AND SOLUTION
% Assembly three load cases corresponding to the three strain cases
sF = reshape(feC, [24, nel]);  % feC should already account for E1, E2, G12, and nu12
```

```
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

ke0=zeros(8,8);
fe0=zeros(8,3);
for k = 1:nel
    ke0(:,:)=keC(:,:,k);
    fe0(:,:)=feC(:,:,k);

    % epsilon0_11 = (1, 0, 0)
    chi0(k,[3 5:end],1) = ke0([3 5:end],[3 5:end])\fe0([3 5:end],1);
    % epsilon0_22 = (0, 1, 0)
    chi0(k,[3 5:end],2) = ke0([3 5:end],[3 5:end])\fe0([3 5:end],2);
    % epsilon0_12 = (0, 0, 1)
    chi0(k,[3 5:end],3) = ke0([3 5:end],[3 5:end])\fe0([3 5:end],3);
end

CH = zeros(3);
cellVolume = lx*ly;
sumC = zeros(nel,8);
for i = 1:3
  for j = 1:3
      chii = chi(edofMat+(i-1)*ndof);
      chij = chi(edofMat+(j-1)*ndof);

      for k = 1:nel
        ke_h = squeeze(keC(:,:,k));
        sumC_h = ((chi0(k,:,i) - chii(k,:))*ke_h).*...
               (chi0(k,:,j) - chij(k,:));
        for q = 1:8
            sumC(k,q) = sumC_h(q);
        end
      end
        sumC = sum(sumC,2);
```

```
        % Homogenized elasticity tensor
        CH(i,j) = 1/cellVolume*sum(sumC);
  end
end

%% COMPUTE ELEMENT STIFFNESS MATRIX AND FORCE VECTOR (NUMERICALLY)
function [keC, feC] = elementMatVecOrtho(a, b, phi, C, nel, flag)
% Constitutive matrix contributions
% Initialize
    keC = zeros(8, 8, nel);
    feC = zeros(8, 3, nel);
    L = zeros(3, 4); L(1, 1) = 1; L(2, 4) = 1; L(3, 2:3) = 1;
    xx = [-1/sqrt(3), 1/sqrt(3)]; yy = xx;
    ww = [1, 1];
for el = 1:nel
    C_local = squeeze(C(el,:,:));
    ke_local = zeros(8, 8);  % Initialize for this element
    fe_local = zeros(8, 3);  % Initialize for this element
    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;
        % Update the local ke and fe for this element
        ke_local = ke_local + weight * (B' * C_local * B);
```

```
        fe_local = fe_local + weight * (B' * C_local * diag([1, 1, 1]));
      end
    end
  % Store the local ke and fe in the global 3D arrays
    keC(:,:, el) = ke_local;
    feC(:,:, el) = fe_local;
end
```

Appendix E. The *Homogenize_test_ortho_principal* Code

```
clc
clear all
close all

%User inputs (let's ignore the 'coordinates' option for now)
matrixSize = input('Enter UC matrix size: ');
shape = input('Enter UCs void shape (circle, rectangle, hexagon): ','s');
x_length = input('Enter the structure width: ');
y_length = input('Enter the structure height: ');
theta = input('Enter the printing angle [degrees]: ');
theta_dummy = theta-1:0.1:theta+1;
% Define grid parameters
x_range = 0:1:x_length-1;
y_range = 0:1:y_length-1;
x_center = (x_length-1)/2+2;
y_center = (y_length-1)/2-3;
RD = zeros(x_length, y_length);
% Define a structure to hold the coordinates and elasticity tensors
plate = struct('x', [], 'y', [], 'tensor', [], 'principal_tensor', []);
RD_struct = struct('x', [], 'y', [], 'RD', []);

%Dummy flag
y_now = pi();
%Full structure visualization
FullStructure = [];
% Initialize a row holder for each row of unit cells
rowHolder = [];
flag=0;
CH_principal = zeros(3);
CH_principal(1,1) = 150;
CH_principal(2,2) = 9;
CH_principal(3,3) = 8;
```

```
CH_principal(1,2) = 0.3;
CH_principal(2,1) = CH_principal(1,2);
% Loop through the grid and compute the elasticity tensor for each point
for y = fliplr(y_range)
    % Clear rowHolder for the new row
    rowHolder = [];
    for x = x_range

            switch shape
                case 'circle'
% write your function for circular voids here
       argument = sin(sqrt((x - 5)^2 + (y - 5)^2))*10;

                    X = NSF_function(matrixSize, shape, argument);
                case 'rectangle'


% write your function for rectangular/square voids here


                    argument1 = abs(sin(x+y))*matrixSize/4+1;
                    if argument1 >= floor(matrixSize/2)
                        argument1 = floor(matrixSize/2)-1;
                    end

                    argument2 = abs(sin(x+y))*matrixSize/4+1;
                    if argument2 >= floor(matrixSize/2)
                        argument2 = floor(matrixSize/2)-1;
                    end




                    X = NSF_function(matrixSize, shape, argument1, argument2);
                case 'hexagon'
% write your function for hexagonal voids here

                    argument = floor(sqrt(y^2+x^2))+1;
                    if argument >= floor(matrixSize/2)
                        argument = floor(matrixSize/2)-1;
                    end
                    X = NSF_function(matrixSize, shape, argument);
```

```
            end
            CH = homogenize_ortho(1,1,[150 1],[9 0.01],[8 0.08],[0.3 0.3],90,theta,X,flag);

            syms E1 E2 G12 nu12 real positive

            %system of eqns
            eqn1 = 1./(((cosd(theta)).^4)./E1+(1./G12-
2.*nu12./E1).*((sind(theta)).^2).*((cosd(theta)).^2)+((sind(theta)).^4)./E2) == CH(1,1);
            eqn2 = 1./(((sind(theta)).^4)./E1+(1./G12-
2.*nu12./E1).*((sind(theta)).^2).*((cosd(theta)).^2)+((cosd(theta)).^4)./E2) == CH(2,2);
            eqn3 =
1./(((sind(theta)).^4+(cosd(theta)).^4)./G12+4.*(1./E1+1./E2+2.*nu12./E1-
1./(2.*G12)).*((sind(theta)).^2).*((cosd(theta)).^2)) == CH(3,3);
            eqn4 = CH(1,1).*(nu12./E1-1/4.*(1./E1+2.*nu12./E1+1./E2-
1./G12).*((sind(2*theta)).^2)) == (CH(1,2)/CH(2,2));

            %Solve
            Sol = vpasolve([eqn1, eqn2, eqn3, eqn4], [E1,E2,G12,nu12], [CH_princi-
pal(1,1)/2; CH_principal(2,2)/2; CH_principal(3,3)/2; CH_principal(1,2)/2]);

            % Check if solution is empty
            if isempty(Sol.E1) || isempty(Sol.E2) || isempty(Sol.G12) || isempty(Sol.nu12)
                warning('No solution found for the given values of CH and theta.');
                   x
                   y
                   for i = theta_dummy
                       syms E1 E2 G12 nu12 real positive

                        %system of eqns
                        eqn1 = 1./(((cosd(theta_dummy)).^4)./E1+(1./G12-
2.*nu12./E1).*((sind(theta_dummy)).^2).*((cosd(theta_dummy)).^2)+((sind(theta_dummy)).^4)./
E2) == CH(1,1);
                        eqn2 = 1./(((sind(theta_dummy)).^4)./E1+(1./G12-
2.*nu12./E1).*((sind(theta_dummy)).^2).*((cosd(theta_dummy)).^2)+((cosd(theta_dummy)).^4)./
E2) == CH(2,2);
                        eqn3 =
1./(((sind(theta_dummy)).^4+(cosd(theta_dummy)).^4)./G12+4.*(1./E1+1./E2+2.*nu12./E1-
1./(2.*G12)).*((sind(theta_dummy)).^2).*((cosd(theta_dummy)).^2)) == CH(3,3);
                        eqn4 = CH(1,1).*(nu12./E1-1/4.*(1./E1+2.*nu12./E1+1./E2-
1./G12).*((sind(2*theta_dummy)).^2)) == (CH(1,2)/CH(2,2));

                        %Solve
```

```
                        Sol = vpasolve([eqn1, eqn2, eqn3, eqn4], [E1,E2,G12,nu12],
[CH_principal(1,1)/2; CH_principal(2,2)/2; CH_principal(3,3)/2; CH_principal(1,2)/2]);
                        if isempty(Sol.E1) || isempty(Sol.E2) || isempty(Sol.G12) ||
isempty(Sol.nu12)
                            warning('No solution found for the given values of CH and
theta.');
                        else
                            break;
                        end
                    end
            else
                CH_principal = zeros(3);
                % If a solution is found, assign it to CH_principal
                CH_principal(1,1) = Sol.E1;
                CH_principal(2,2) = Sol.E2;
                CH_principal(3,3) = Sol.G12;
                CH_principal(1,2) = Sol.E2*Sol.nu12;
                CH_principal(2,1) = CH_principal(1,2);
            end



                flag=0;
            %end
            %Save "relative density"
            RD_struct(end+1)=struct('x', x, 'y', y, 'RD', sum(sum(X == 1))/(matrixSize^2));
            RD(x+1,y+1) = sum(sum(X == 1))/(matrixSize^2);
            %y_now=y;
            % Add the data to the structure
            plate(end+1) = struct('x', x, 'y', y, 'tensor', CH, 'principal_tensor', CH_princi-
pal);
            % Concatenate this unit cell to the row holder
            rowHolder = [rowHolder, X];
        end
        % Once a full row of unit cells is formed, concatenate it to the FullStructure
        FullStructure = [FullStructure; rowHolder];
end
plate = plate(2:end); %Get rid of first empty entry
RD_struct = RD_struct(2:end);
%% Plotting
% Calculate the range of densities
densityMin = min(RD(:));
```

```
densityMax = max(RD(:));

% Define contour levels
numLevels = 50; % You can change this value
contourLevels = linspace(densityMin, densityMax, numLevels);
figure(1)
contourf(x_range,y_range,RD',contourLevels,'LineColor', 'none')
colormap(flipud(gray));
colorbar
caxis([densityMin, densityMax]); % Set color axis scaling
title('Relative density plot')
xlabel('x coordinate')
ylabel('y coordinate')
set(gca, 'FontSize', 19)
axis equal
%Full cellular/porous structure
figure(2)
imshow(FullStructure, [1, 2]);
title('Final Combined Matrix');
xlabel('x coordinate')
ylabel('y coordinate')

axis equal
%% Obtain the "reference" Elasticity tensor at a given set of coordinates
% x_coordinate = floor(x_length/2); %This gives the CH at the center of the plate
% y_coordinate = floor(y_length/2);
%Manual input
x_coordinate = 9; %This gives the CH at the center of the plate
y_coordinate = 2;
tensor = getTensorAtCoordinate(plate, x_coordinate, y_coordinate); %Uses the funnction at
the end

%% Curve fitting the Elasticity Tensor Map
% Initialize tensor_function as a cell array
tensor_function = cell(3);

for i=1:3
   for j=1:3
        x_data = [plate.x];
        y_data = [plate.y];
        tensor_data = cellfun(@(t) t(i,j), {plate.tensor});
```

```
        % Define the fit type, e.g., a polynomial
        fitType = fittype('poly55'); % second-degree polynomial

        % Perform the fit
        tensor_function{i,j} = fit([x_data', y_data'], tensor_data', fitType);
    end
end
tensor_function

%% Curve fitting the Principal Elasticity Tensor Map
% Initialize tensor_function as a cell array
principal_tensor_function = cell(3);

for i=1:3
   for j=1:3
        x_data = [plate.x];
        y_data = [plate.y];
        principal_tensor_data = cellfun(@(t) t(i,j), {plate.principal_tensor});

        % Define the fit type, e.g., a polynomial
        fitType = fittype('poly55'); % second-degree polynomial

        % Perform the fit
        principal_tensor_function{i,j} = fit([x_data', y_data'], principal_tensor_data',
fitType);
    end
end
principal_tensor_function

%% Extracting stiffness tensor coefficients into CSV file

all_coeffs = []; % Initialize empty matrix to collect all coefficients

% Variable names for the coefficients in the order they are returned by coeffvalues
variable_names = {'p00', 'p10', 'p01', 'p20', 'p11', 'p02', 'p30', 'p21', 'p12', 'p03',
'p40', 'p31', 'p22', 'p13', 'p04', 'p50', 'p41', 'p32', 'p23', 'p14', 'p05'};

% List of desired indices as (row, column) pairs
desired_indices = [1 1; 1 2; 2 2; 3 3];

% Loop over the desired indices
for index = 1:size(desired_indices, 1)
```

```
    % obtain the row and column from the current index pair
    row = desired_indices(index, 1);
    col = desired_indices(index, 2);

    % Collect current coefficients from the (row, col) entry
    current_coeffs = coeffvalues(tensor_function{row,col});

    % Append to the all_coeffs matrix
    all_coeffs = [all_coeffs; current_coeffs];
end

% Convert the full matrix to a table before writing to CSV
all_coeffs_table = array2table(all_coeffs, 'VariableNames', variable_names);

% Write the full table of coefficients to a CSV file
writetable(all_coeffs_table, 'selected_tensor_coefficients1.csv');

%% Extracting principal stiffness tensor coefficients into CSV file

principal_all_coeffs = []; % Initialize empty matrix to collect all coefficients

% Variable names for the coefficients in the order they are returned by coeffvalues
variable_names = {'p00', 'p10', 'p01', 'p20', 'p11', 'p02', 'p30', 'p21', 'p12', 'p03',
'p40', 'p31', 'p22', 'p13', 'p04', 'p50', 'p41', 'p32', 'p23', 'p14', 'p05'};

% List of desired indices as (row, column) pairs
desired_indices = [1 1; 1 2; 2 2; 3 3];

% Loop over the desired indices
for index = 1:size(desired_indices, 1)
    % obtain the row and column from the current index pair
    row = desired_indices(index, 1);
    col = desired_indices(index, 2);

    % Collect current coefficients from the (row, col) entry
    current_coeffs = coeffvalues(principal_tensor_function{row,col});

    % Append to the all_coeffs matrix
    principal_all_coeffs = [principal_all_coeffs; current_coeffs];
end

% Convert the full matrix to a table before writing to CSV
```

```
principal_all_coeffs_table = array2table(principal_all_coeffs, 'VariableNames', varia-
ble_names);

% Write the full table of coefficients to a CSV file
writetable(principal_all_coeffs_table, 'principal_tensor_coefficients.csv');
%% Curve fitting the Relative Density Map

x_data = [plate.x];
y_data = [plate.y];
RD_data = [RD_struct.RD];
% Define the fit type, e.g., a polynomial
fitType = fittype('poly33'); % second-degree polynomial

% Perform the fit
%RDL = reshape(RD, [], 1);
RD_function = fit([x_data', y_data'], RD_data', fitType);


% "L" is a specific index in the data structure and displays RD for that
% specific index
L = 88;
% obtain the coordinates for L = 88 by typing the following lines
% x_coordinate = plate(88).x
% y_coordinate = plate(88).y

RD_Fitted = RD_function(plate(L).x, plate(L).y)
RD_Actual = RD_struct(L).RD

%% Extracting density coefficients to CSV file

% Collect the coefficients for the fit
RD_coeffs = coeffvalues(RD_function);

% Create a variable names array corresponding to the coefficients of the poly33 model
variable_names = {'p00', 'p10', 'p01', 'p20', 'p11', 'p02', 'p30', 'p21', 'p12', 'p03'};

% Convert the coefficients to a table with appropriate variable names
RD_coeffs_table = array2table(RD_coeffs, 'VariableNames', variable_names);

% Write the table to a CSV file
writetable(RD_coeffs_table, 'RD_coefficients.csv');
```

```
%% Evaluating the curve-fitted Elasticity tensor from coordinate
%Accessing  specific equation (entry of the tensor)
specific_function = tensor_function{1,2};
%Evaluating the function at a given x and y value
x_value = 9; % Replace with the desired x value
y_value = 2; % Replace with the desired y value
result = specific_function(x_value, y_value);
%Evaluating full Elasticity tensor at a given y value
elasticity_tensor = zeros(3);
for i = 1:3
    for j = 1:3
        elasticity_tensor(i,j) = tensor_function{i,j}(x_value, y_value);
    end
end
elasticity_tensor

%% Evaluating error (fit analysis)

% Initialize variable for accumulating total element-wise percentage error
total_percentage_error = 0;
total_elements = 0;

% Loop through all coordinates in 'plate' to accumulate the percentage errors
for i = 1:length(plate)
    x = plate(i).x;
    y = plate(i).y;

    % Extract the reference tensor from 'plate'
    tensor_exact = plate(i).tensor;

    % Evaluate the approximate tensor at the same coordinates
    for m=1:3
        for n=1:3
            tensor_approx(m,n) = tensor_function{m,n}(x, y);
        end
    end

    % Compute the element-wise percentage error
    % Avoid division by zero by adding a small constant (e.g., 1e-9)
    percentage_error_matrix = abs((tensor_exact - tensor_approx)./ (tensor_exact + 1e-9)) *
100;
```

```
    % Sum up the percentage errors and count the number of elements
    total_percentage_error = total_percentage_error + sum(sum(percentage_error_matrix));
    total_elements = total_elements + numel(percentage_error_matrix);
end

% Compute the overall average element-wise percentage error
overall_avg_percentage_error = total_percentage_error / total_elements;

fprintf('The overall average element-wise percentage error is %.2f%%\n', overall_avg_per-
centage_error);

%% Plotting E1


x=0:x_length-1;
y=0:y_length-1;
E1=zeros(x_length,y_length);
for i=x+1
    for j=y+1
        E1(i,j) = principal_tensor_function{1,1}(i-1,j-1);
    end
end
Min=min(min(E1));
Max=max(max(E1));
numLevels = 50;
contourLevels = linspace(Min, Max, numLevels);
figure(3)
contourf(x_range,y_range,E1',contourLevels,'LineColor', 'none')
colormap(flipud(gray));
colorbar
caxis([Min, Max]); % Set color axis scaling
title('E_{1} plot')
xlabel('x coordinate')
ylabel('y coordinate')
set(gca, 'FontSize', 19)
axis equal

%% Plotting E2

x=0:x_length-1;
y=0:y_length-1;
E2=zeros(x_length,y_length);
```

```
for i=x+1
    for j=y+1
        E2(i,j) = principal_tensor_function{2,2}(i-1,j-1);
    end
end
Min=min(min(E2));
Max=max(max(E2));
numLevels = 50;
contourLevels = linspace(Min, Max, numLevels);
figure(4)
contourf(x_range,y_range,E2',contourLevels,'LineColor', 'none')
colormap(flipud(gray));
colorbar
caxis([Min, Max]); % Set color axis scaling
title('E_2 plot')
xlabel('x coordinate')
ylabel('y coordinate')
set(gca, 'FontSize', 19)
axis equal
%% Plotting G12


x=0:x_length-1;
y=0:y_length-1;
G12=zeros(x_length,y_length);
for i=x+1
    for j=y+1
        G12(i,j) = principal_tensor_function{3,3}(i-1,j-1);
    end
end
Min=min(min(G12));
Max=max(max(G12));
numLevels = 50;
contourLevels = linspace(Min, Max, numLevels);
figure(5)
contourf(x_range,y_range,G12',contourLevels,'LineColor', 'none')
colormap(flipud(gray));
colorbar
caxis([Min, Max]); % Set color axis scaling
title('G_{12} plot')
xlabel('x coordinate')
ylabel('y coordinate')
```

```
set(gca, 'FontSize', 19)
axis equal

%% Plotting nu12


x=0:x_length-1;
y=0:y_length-1;
nu12=zeros(x_length,y_length);
for i=x+1
    for j=y+1
        nu12(i,j) = (principal_tensor_function{1,2}(i-1,j-1))/(principal_tensor_func-
tion{2,2}(i-1,j-1));
    end
end
Min=min(min(nu12));
Max=max(max(nu12));
numLevels = 50;
contourLevels = linspace(Min, Max, numLevels);
figure(6)
contourf(x_range,y_range,nu12',contourLevels,'LineColor', 'none')
colormap(flipud(gray));
colorbar
caxis([Min, Max]); % Set color axis scaling
title('\nu_{12} plot')
xlabel('x coordinate')
ylabel('y coordinate')

set(gca, 'FontSize', 19)
axis equal




%% getTensorAtCoordinate "reference" function
%Get a specific tensor from coordinate (function)
function tensor = getTensorAtCoordinate(plate, x, y)
    for i = 1:length(plate)
        if isequal(plate(i).x, x) && isequal(plate(i).y, y)
            tensor = plate(i).tensor
            return;
```

```
        end
    end
    error('Coordinate not found in the structure.');
end
```

References

1. Shahbazian, B.; Mirsayar, M. Fracture mechanics of cellular structures: Past, present, and future directions. *Eng. Solid Mech.* **2023**, *11*, 231–242.
2. Parthasarathy, J.; Starly, B.; Raman, S. A design for the additive manufacture of functionally graded porous structures with tailored mechanical properties for biomedical applications. *J. Manuf. Process* **2011**, *13*, 160–170.
3. Zhao, S.; Li, S.J.; Hou, W.T.; Hao, Y.L.; Yang, R.; Murr, L.E. Microstructure and mechanical properties of open cellular Ti–6Al–4V prototypes fabricated by electron beam melting for biomedical applications. *Mater. Technol.* **2016**, *31*, 98–107.
4. Abate, K.M.; Nazir, A.; Yeh, Y.P.; Chen, J.E.; Jeng, J.Y. Design, optimization, and validation of mechanical properties of different cellular structures for biomedical application. *Int. J. Adv. Manuf. Technol.* **2020**, *106*, 1253–1265.
5. Heo, H.; Ju, J.; Kim, D.M. Compliant cellular structures: Application to a passive morphing airfoil. *Compos. Struct.* **2013**, *106*, 560–569.
6. Hohe, J.; Hardenacke, V.; Fascio, V.; Girard, Y.; Baumeister, J.; Stöbener, K.; Weise, J.; Lehmhus, D.; Pattofatto, S.; Zeng, H.; et al. Numerical and experimental design of graded cellular sandwich cores for multi-functional aerospace applications. *Mater. Des.* **2012**, *39*, 20–32.
7. Lv, J.; Ren, X.; Song, C.; Zhang, H. Two-Scale Topology Optimization of the 3D Plant-Inspired Adaptive Cellular Structures for Morphing Applications. *J. Aerosp. Eng.* **2020**, *33*, 04020032.
8. Davalos, J.F.; Qiao, P.; Xu, X.F.; Robinson, J.; Barth, K.E. Modeling and characterization of fiber reinforced plastic honeycomb sandwich panels for highway bridge applications. *Compos. Struct.* **2001**, *52*, 441–452.
9. Nazir, A.; Jeng, J.Y. Buckling behavior of additively manufactured cellular columns: Experimental and simulation validation. *Mater. Des.* **2020**, *186*, 108349.
10. Horn, T.J.; Harrysson, O.L.; Marcellin-Little, D.J.; West, H.A.; Lascelles, B.D.X.; Aman, R. Flexural properties of Ti6Al4V rhombic dodecahedron open cellular structures fabricated with electron beam melting. *Addit. Manuf.* **2014**, *1–4*, 2–11.
11. Borsellino, C.; Di Bella, G. Reinforced biomimetic cellular structures for automotive applications. *Mater. Des.* **2009**, *30*, 4054–4059.
12. Saenz-Dominguez, I.; Tena, I.; Esnaola, A.; Sarrionandia, M.; Torre, J.; Aurrekoetxea, J. Design and characterization of cellular composite structures for automotive crash-boxes manufactured by out of die ultraviolet cured pultrusion. *Compos. Part. B Eng.* **2019**, *160*, 217–224.
13. Mkrtchyan, L.; Maier, M.; Huber, U. Structural polyurethane foam: Testing and modelling for automotive applications. *Int. J. Crashworthiness* **2008**, *13*, 523–532.
14. Hang, X.; He, S.; Dong, Z.; Minnick, G.; Rosenbohm, J.; Chen, Z.; Yang, R.; Chang, L. Nanosensors for single cell mechanical interrogation. *Biosens. Bioelectron.* **2021**, *179*, 113086.
15. Banhart, J. Manufacture, characterisation and application of cellular metals and metal foams. *Prog. Mater. Sci.* **2001**, *46*, 559–632.
16. Srivastava, V.C.; Sahoo, K.L. Processing, stabilization and applications of metallic foams. *Mater. Sci.* **2007**, *25*, 733–753.
17. Salimon, A.; Brechet, Y.; Ashby, M.F.; Greer, A.L. Potential applications for steel and titanium metal foams. *J. Mater. Sci.* **2005**, *40*, 5793–5799.
18. Benedetti, M.; Du Plessis, A.; Ritchie, R.O.; Dallago, M.; Razavi, S.M.; Berto, F. Architected cellular materials: A review on their mechanical properties towards fatigue-tolerant design and fabrication. *Mater. Sci. Eng. R. Rep.* **2021**, *144*, 100606.
19. Fuganti, A.; Lorenzi, L.; Grønsund, A.; Langseth, M. Aluminum foam for automotive applications. *Adv. Eng. Mater.* **2000**, *2*, 200–204.
20. Costanza, G.; Solaiyappan, D.; Tata, ME. Properties, applications and recent developments of cellular solid materials: a review. *Maters*. **2023**, 16, 7076.
21. Najmon, J.C.; Jacob, D.J.; Wood, Z.M.; Tovar, A. Cellular helmet liner design through bio-inspired structures and topology optimization of compliant mechanism lattices. *SAE Int. J. Transp. Saf.* **2018**, *6*, 217–236.
22. Murr, L.E.; Gaytan, S.M.; Medina, F.; Lopez, H.; Martinez, E.; Machado, B.I.; Hernandez, D.H.; Martinez, L.; Lopez, M.I.; Wicker, R.B.; et al. Next-generation biomedical implants using additive manufacturing of complex, cellular and functional mesh arrays. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* **2010**, *368*, 1999–2032.
23. Freyman, T.M.; Yannas, I.V.; Gibson, L.J. Cellular materials as porous scaffolds for tissue engineering. *Prog. Mater. Sci.* **2001**, *46*, 273–282.
24. Daynes, S.; Feih, S.; Lu, W.F.; Wei, J. Optimization of functionally graded lattice structures using isostatic lines. *Mater. Des.* **2017**, *127*, 215–223.
25. Zhu, L.; Li, N.; Childs, P.R.N. Light-weighting in aerospace component and system design. *Propuls. Power Res.* **2018**, *7*, 103–119.

26. Das, S.; Sutradhar, A. Multi-physics topology optimization of functionally graded controllable porous structures: Application to heat dissipating problems. *Mater. Des.* **2020**, *193*, 108775.
27. Exerowa, D.; Kruglyakov, P.M. *Foam and Foam Films: Theory, Experiment, Application*; Elsevier: Amsterdam, The Netherlands, 1997.
28. Férey, G. Hybrid porous solids: Past, present, future. *Chem. Soc. Rev.* **2008**, *37*, 191–214.
29. Burschka, J.; Pellet, N.; Moon, S.J.; Humphry-Baker, R.; Gao, P.; Nazeeruddin, M.K.; Grätzel, M. Sequential deposition as a route to high-performance perovskite-sensitized solar cells. *Nature* **2013**, *499*, 316–319.
30. Huang, Z.; Chen, H.; Huang, Y.; Ge, Z.; Zhou, Y.; Yang, Y.; Xiao, P.; Liang, J.; Zhang, T.; Shi, Q.; et al. Ultra-broadband wide-angle terahertz absorption properties of 3D graphene foam. *Adv. Funct. Mater.* **2018**, *28*, 1704363.
31. Huang, P.; Wu, F.; Shen, B.; Zheng, H.; Ren, Q.; Luo, H.; Zheng, W. Biomimetic porous polypropylene foams with special wettability properties. *Compos. Part B Eng.* **2020**, *190*, 107927.
32. Liu, Q.; Gao, S.; Zhao, Y.; Tao, W.; Yu, X.; Zhi, M. Review of layer-by-layer self-assembly technology for fire protection of flexible polyurethane foam. *J. Mater. Sci.* **2021**, *56*, 9605–9643.
33. Schaedler, T.A.; Carter, W.B. Architected cellular materials. *Annu. Rev. Mater. Res.* **2016**, *46*, 187–210.
34. Prabhu, S.; Raja, V.K.; Nikhil, R. Applications of cellular materials—An overview. *Appl. Mech. Mater.* **2015**, *766*, 511–517.
35. Gibson, I.J.; Ashby, M.F. The mechanics of three-dimensional cellular materials. *Proc. R. Soc. A Math. Phys. Sci.* **1982**, *382*, 43–59.
36. Gibson, L.J. Modelling the mechanical behavior of cellular materials. *Mater. Sci. Eng. A* **1989**, *110*, 1–36.
37. Gibson, L.J.; Ashby, M.F.; Schajer, G.S.; Robertson, C.I. The mechanics of two-dimensional cellular materials. *Proc. R. Soc. A Math. Phys. Sci.* **1982**, *382*, 25–42.
38. Maiti, S.K.; Gibson, L.J.; Ashby, M.F. Deformation and energy absorption diagrams for cellular solids. *Acta Metall.* **1984**, *32*, 1963–1975.
39. Zhang, Y.; Gao, L.; Xiao, M. Maximizing natural frequencies of inhomogeneous cellular structures by Kriging-assisted multiscale topology optimization. *Comput. Struct.* **2020**, *230*, 106197.
40. Zhang, Y.; Xiao, M.; Li, H.; Gao, L.; Chu, S. Multiscale concurrent topology optimization for cellular structures with multiple microstructures based on ordered SIMP interpolation. *Comput. Mater. Sci.* **2018**, *155*, 74–91.
41. Li, D.; Dai, N.; Tang, Y.; Dong, G.; Zhao, Y.F. Design and optimization of graded cellular structures with triply periodic level surface-based topological shapes. *J. Mech. Des.* **2019**, *141*, 071402.
42. Li, D.; Liao, W.; Dai, N.; Dong, G.; Tang, Y.; Xie, Y.M. Optimal design and modeling of gyroid based functionally graded cellular structures for additive manufacturing. *Comput. Aided Des.* **2018**, *104*, 87–99.
43. Zhang, P.; Toman, J.; Yu, Y.; Biyikli, E.; Kirca, M.; Chmielus, M.; To, A.C. Efficient design optimization of variable-density hexagonal cellular structure by additive manufacturing: Theory and validation. *J. Manuf. Sci. Eng.* **2015**, *137*, 021004.
44. Tamijani, A.Y.; Velasco, S.P.; Alacoque, L. Topological and morphological design of additively manufacturable spatially varying periodic cellular solids. *Mater. Des.* **2020**, *196*, 109155.
45. Losic, D.; Rosengarten, G.; Mitchell, J.G.; Voelcker, N.H. Pore architecture of diatom frustules: Potential nanostructured membranes for molecular and particle separations. *J. Nanosci. Nanotechnol.* **2006**, *6*, 982–989.
46. Zhang, W.; Gu, J.; Liu, Q.; Su, H.; Fan, T.; Zhang, D. Butterfly effects: Novel functional materials inspired from the wings scales. *Phys. Chem. Chem. Phys.* **2014**, *16*, 19767–19780.
47. Karam, G.N.; Gibson, L.J. Elastic buckling of cylindrical shells with elastic cores—I. Analysis. *Int. J. Solids Struct.* **1995**, *32*, 1259–1283.
48. Montoya, C.; Arola, D.; Ossa, E.A. Importance of tubule density to the fracture toughness of dentin. *Arch. Oral Biol.* **2016**, *67*, 9–14.
49. Chen, H.; Yang, T.; Wu, Z.; Deng, Z.; Zhu, Y.; Li, L. Quantitative 3D structural analysis of the cellular microstructure of sea urchin spines (II): Large-volume structural analysis. *Acta Biomater.* **2020**, *107*, 218–231.
50. Ostertag, A.; Peyrin, F.; Fernandez, S.; Laredo, J.D.; de Vernejoul, M.C.; Chappard, C. Cortical measurements of the tibia from high resolution peripheral quantitative computed tomography images: A comparison with synchrotron radiation micro-computed tomography. *Bone* **2014**, *63*, 7–14.
51. Kladovasilakis, N.; Tsongas, K.; Tzetzis, D. Finite element analysis of orthopedic hip implant with functionally graded bioinspired lattice structures. *Biomimetics* **2020**, *5*, 44.
52. Nian, Y.; Wan, S.; Li, X.; Su, Q.; Li, M. How does bio-inspired graded honeycomb filler affect energy absorption characteristics? *Thin-Walled Struct.* **2019**, *144*, 106269.
53. Li, D.; Liao, W.; Dai, N.; Xie, Y.M. Comparison of mechanical properties and energy absorption of sheet-based and strut-based gyroid cellular structures with graded densities. *Materials* **2019**, *12*, 2183.
54. Jin, Y.; Kong, H.; Zhou, X.; Li, G.; Du, J. Design and Characterization of Sheet-Based Gyroid Porous Structures with Bioinspired Functional Gradients. *Materials* **2020**, *13*, 3844.
55. Xiang, X.; Zou, S.; San Ha, N.; Lu, G. Energy absorption of bio-inspired multi-layered graded foam-filled structures under axial crushing. *Compos. Part. B Eng.* **2020**, *198*, 108216.
56. Mirzaali, M.J.; de la Nava, A.H.; Gunashekar, D.; Nouri-Goushki, M.; Veeger, R.P.E.; Grossman, Q.; Angeloni, L.; Ghatkesar, M.K.; Fratila-Apachitei, L.E.; Ruffoni, D.; et al. Mechanics of bioinspired functionally graded soft-hard composites made by multi-material 3D printing. *Compos. Struct.* **2020**, *237*, 111867.
57. Peng, C.; Tran, P. Bioinspired functionally graded gyroid sandwich panel subjected to impulsive loadings. *Compos. Part. B Eng.* **2020**, *188*, 107773.

58. Tan, C.; Zou, J.; Li, S.; Jamshidi, P.; Abena, A.; Forsey, A.; Moat, R.J.; Essa, K.; Wang, M.; Zhou, K.; et al. Additive manufacturing of bio-inspired multi-scale hierarchically strengthened lattice structures. *Int. J. Mach. Tools Manuf.* **2021**, *167*, 103764.
59. Yin, H.; Wang, X.; Wen, G.; Zhang, C.; Zhang, W. Crashworthiness optimization of bio-inspired hierarchical honeycomb under axial loading. *Int. J. Crashworthiness* **2021**, *26*, 26–37.
60. Opgenoord, M.M.; Willcox, K.E. Design for additive manufacturing: Cellular structures in early stage aerospace design. *Struct. Multidiscip. Optim.* **2019**, *60*, 411–428.
61. Nazir, A.; Abate, K.M.; Kumar, A.; Jeng, J.Y. A state-of-the-art review on types, design, optimization, and additive manufacturing of cellular structures. *Int. J. Adv. Manuf. Technol.* **2019**, *104*, 3489–3510.
62. Yang, L.; Harrysson, O.; Cormier, D.; West, H.; Gong, H.; Stucker, B. Additive manufacturing of metal cellular structures: Design and fabrication. *JOM* **2015**, *67*, 608–615.
63. Robbins, J.; Owen, S.J.; Clark, B.W.; Voth, T.E. An efficient and scalable approach for generating topologically optimized cellular structures for additive manufacturing. *Addit. Manuf.* **2016**, *12*, 296–304.
64. Cheng, L.; Zhang, P.; Biyikli, E.; Bai, J.; Robbins, J.; To, A. Efficient design optimization of variable-density cellular structures for additive manufacturing: Theory and experimental validation. *Rapid Prototyp. J.* **2017**, *23*, 660–677.
65. Liu, T.; Guessasma, S.; Zhu, J.; Zhang, W. Designing Cellular Structures for Additive Manufacturing Using Voronoi–Monte Carlo Approach. *Polymers* **2019**, *11*, 1158.
66. Chu, J.; Engelbrecht, S.; Graf, G.; Rosen, D.W. A comparison of synthesis methods for cellular structures with application to additive manufacturing. *Rapid Prototyp. J.* **2010**, *16*, 275–283.
67. Hassanin, H.; Alkendi, Y.; Elsayed, M.; Essa, K.; Zweiri, Y. Controlling the properties of additively manufactured cellular structures using machine learning approaches. *Adv. Eng. Mater.* **2020**, *22*, 1901338.
68. Limmahakhun, S.; Oloyede, A.; Sitthiseripratip, K.; Xiao, Y.; Yan, C. 3D-printed cellular structures for bone biomimetic implants. *Addit. Manuf.* **2017**, *15*, 93–101.
69. Felzmann, R.; Gruber, S.; Mitteramskogler, G.; Tesavibul, P.; Boccaccini, A.R.; Liska, R.; Stampfl, J. Lithography-based additive manufacturing of cellular ceramic structures. *Adv. Eng. Mater.* **2012**, *14*, 1052–1058.
70. Xu, Y.; Zhang, H.; Šavija, B.; Figueiredo, S.C.; Schlangen, E. Deformation and fracture of 3D printed disordered lattice materials: Experiments and modeling. *Mater. Des.* **2019**, *162*, 143–153.
71. Hmeidat, N.S.; Pack, R.C.; Talley, S.J.; Moore, R.B.; Compton, B.G. Mechanical anisotropy in polymer composites produced by material extrusion additive manufacturing. *Addit. Manuf.* **2020**, *34*, 101385.
72. Kok, Y.; Tan, X.P.; Wang, P.; Nai, M.L.S.; Loh, N.H.; Liu, E.; Tor, S.B. Anisotropy and heterogeneity of microstructure and mechanical properties in metal additive manufacturing: A critical review. *Mater. Des.* **2018**, *139*, 565–586.
73. Somireddy, M.; Czekanski, A. Anisotropic material behavior of 3D printed composite structures– Material extrusion additive manufacturing. *Mater. Des.* **2020**, *195*, 108953.
74. Carroll, B.E.; Palmer, T.A.; Beese, A.M. Anisotropic tensile behavior of Ti–6Al–4V components fabricated with directed energy deposition additive manufacturing. *Acta Mater.* **2015**, *87*, 309–320.
75. Zhu, Y.; Tian, X.; Li, J.; Wang, H. The anisotropy of laser melting deposition additive manufacturing Ti–6.5 Al–3.5 Mo–1.5 Zr–0.3 Si titanium alloy. *Mater. Des.* **2015**, *67*, 538–542.
76. Spoerk, M.; Savandaiah, C.; Arbeiter, F.; Traxler, G.; Cardon, L.; Holzer, C.; Sapkota, J. Anisotropic properties of oriented short carbon fibre filled polypropylene parts fabricated by extrusion based additive manufacturing. *Compos. Part A Appl. Sci. Manuf.* **2018**, *113*, 95–104.
77. Evans, A.G.; Hutchinson, J.W.; Ashby, M.F. Multifunctionality of cellular metal systems. *Prog. Mater. Sci.* **1998**, *43*, 171–221.
78. Phani, A.S.; Woodhouse, J.; Fleck, N.A. Wave propagation in two-dimensional periodic lattices. *J. Acoust. Soc. Am.* **2006**, *119*, 1995–2005.
79. Askar, A.; Cakmak, A.S. A structural model of a micropolar continuum. *Int. J. Eng. Sci.* **1968**, *6*, 583–589.
80. Chen, Y.; Liu, X.N.; Hu, G.K.; Sun, Q.P.; Zheng, Q.S. Micropolar continuum modelling of bi-dimensional tetrachiral lattices. *Proc. R. Soc. A Math. Phys. Eng. Sci.* **2014**, *470*, 20130734.
81. Wang, X.L.; Stronge, W.J. Micropolar theory for two–dimensional stresses in elastic honeycomb. *Proc. R. Soc. Ser. A Math. Phys. Eng. Sci.* **1999**, *455*, 2091–2116.
82. Hohe, J.; Becker, W. Determination of the elasticity tensor of non-orthotropic cellular sandwich cores. *Tech. Mech. Eur. J. Eng. Mech.* **1999**, *19*, 259–268.
83. Staszak, N.; Garbowski, T.; Szymczak-Graczyk, A. Solid Truss to Shell Numerical Homogenization of Prefabricated Composite Slabs. *Materials* **2021**, *14*, 4120.
84. Gibson, L.J.; Ashby, M.F.; Schajer, G.S.; Robertson, C.I. The mechanics of two-dimensional cellular materials. *Proc. R. Soc. A Math. Phys. Sci.* **1982**, *382*, 25–42.
85. Masters, I.G.; Evans, K.E. Models for the elastic deformation of honeycombs. *Compos. Struct.* **1996**, *35*, 403–422.
86. Christensen, R.M. Mechanics of cellular and other low-density materials. *Int. J. Solids Struct.* **2000**, *37*, 93–104.
87. Vigliotti, A.; Deshpande, V.S.; Pasini, D. Non-linear constitutive models for lattice materials. *J. Mech. Phys. Solids* **2014**, *64*, 44–60.
88. Arbabi, H.; Bunder, J.E.; Samaey, G.; Roberts, A.J.; Kevrekidis, I.G. Linking machine learning with multiscale numerics: Data-driven discovery of homogenized equations. *JOM* **2020**, *72*, 4444–4457.
89. Hassani, B.; Hinton, E. A review of homogenization and topology optimization I—Homogenization theory for media with periodic structure. *Comput. Struct.* **1998**, *69*, 707–717.

90. Andreassen, E.; Andreasen, C.S. How to determine composite material properties using numerical homogenization. *Comput. Mater. Sci.* **2014**, *83*, 488–495.
91. Dong, G.; Tang, Y.; Zhao, Y.F. A 149 line homogenization code for three-dimensional cellular materials written in MATLAB. *J. Eng. Mater. Technol.* **2019**, *141*, 011005.
92. Lalegani, Z.; Ebrahimi, S.S.; Hamawandi, B.; La Spada, L.; Batili, H.; Toprak, M.S. Targeted dielectric coating of silver nanoparticles with silica to manipulate optical properties for metasurface applications. *Mater. Chem. Phys.* **2022**, *287*, 126250.
93. Lincoln, R.L.; Scarpa, F.; Ting, V.P.; Trask, R.S. Multifunctional composites: A metamaterial perspective. *MFM* **2019**, *2*, 043001.
94. Mirsayar, M. On brittle fracture of two-dimensional lattices with material anisotropies. *Fatigue Fract. Eng. Mater. Struct.* **2022**, *45*, 1929–1941.
95. Yuan, J.; He, S.; Chen, C.; Wang, L. Phase-field fracture analysis of heterogeneous materials based on homogenization method. *Acta Mech.* **2024**, *235*, 1083–1107.
96. Massabò, R.; Darban, H. Mode II dominant fracture of layered composite beams and wide-plates: A homogenized structural approach. *Eng. Fract. Mech.* **2019**, *213*, 280–301.
97. Mirsayar, M.; Shahbazian, B. A novel three-dimensional notch fracture criterion via effective critical distances. *Int. J. Mech. Sci.* **2024**, *271*, 109149.
98. Greco, F. Homogenized mechanical behavior of composite micro-structures including micro-cracking and contact evolution. *Eng. Fract. Mech.* **2009**, *76*, 182–208.
99. Mirsayar, M.M. A combined stress/energy-based criterion for mixed-mode fracture of laminated composites considering fiber bridging micromechanics. *Int. J. Mech. Sci.* **2021**, *197*, 106319.
100. Yuan, J.; Mao, Y.; Chen, C. Multiple-phase-field modeling for fracture of composite materials. *Mech. Adv. Mater. Struct.* **2022**, *29*, 7476–7490.
101. Mirsayar, M.M. A generalized criterion for fatigue crack growth in additively manufactured materials–Build orientation and geometry effects. *Int. J. Fatigue* **2021**, *145*, 106099.