

Modeling the Bandstructure of Macroscopic Phononic Crystals using Finite Element Analysis

Submitted by: Karthik Gururangan

Dept. of Materials Science and Engineering

Northwestern University

Date Submitted: 3/21/2018

Table of Contents

Introduction	Page 3
Lattice Translational Symmetry and Bloch's Theorem	Page 3
Finite Element Formulation	Page 4
Periodic Boundary Conditions	Page 6
Patch Test Verification	Page 7
Phononic Crystal Bandstructure	Page 8
Conclusion	Page 10
References	Page 10
Appendix: Matlab Code.....	Page 11

List of Figures

Fig 1 - Diagram of high-symmetry path along square crystal IBZ.....	Page 4
Fig 2 - Partitioning FEA domain for application of periodic boundary conditions.....	Page 7
Fig 3 - Schematic of 2D elastic domain model used for patch test	Page 7
Fig 4 - Displacement field results from patch test	Page 8
Fig 5 - Schematic of FEA model used to simulate phononic crystal	Page 8
Fig 6 - Calculated band structures of phononic crystal	Page 9

I. Introduction

Since the advent of advanced fabrication techniques with sub-micron level precision, there has been a huge interest in designing macroscopic materials that display quantum mechanical properties. In particular, condensed matter theory explores phenomena unique to solids, such as bandstructures and bandgaps, that result from the spatial lattice symmetry upon which crystal structures are based on. Analogous features can be engineered in macroscopic systems that obey lattice-translational symmetry, resulting in the extensively-researched photonic and phononic crystals. These are materials that present a crystalline environment to propagating waves in the system — electromagnetic waves in photonic crystals and mechanical waves in phononic crystals. The crystalline environment profoundly changes the wave dispersion relation and can open up bandgaps, regions of energy for which wave propagation in certain directions is forbidden. In the phononic crystal, these bandgaps correspond to regions for which acoustic longitudinal waves are forbidden, essentially providing regions of complete reflection of sound. This makes the modeling of bandstructures and bandgaps a pivotal role in the design of banded metamaterials. Due to the macroscopic nature of phononic crystals, this modeling is amenable to finite-element analysis. This work will explore the calculation of phonon bandstructures using elasticity theory in two dimensions.

II. Lattice-Translational Symmetry and Bloch's Theorem

A symmetry group consists of N operators $\{\hat{P}_i\}$ for $i = 1 \dots N$. Each operator acts on functions of space to perform some symmetry operation (e.g. rotation or translation), defined by a linear transformation matrix $\hat{P}_i f(\mathbf{r}) = f(\mathbf{R}_i^{-1} \mathbf{r})$. For rotations, \mathbf{R}_i would be a rotation matrix; for translations, \mathbf{R}_i is a diagonal matrix $f(\mathbf{R}^{-1} \mathbf{r}) = f(\mathbf{r} + \mathbf{T}_{uvw})$ where

$$\{\mathbf{T}_{uvw}\} = \{u\mathbf{a}_1 + v\mathbf{a}_2 + w\mathbf{a}_3 \mid u, v, w = 0 \dots L\}$$

are the lattice translation vectors defined on an $L \times L \times L$ lattice of repeating units (the operator labels i and uvw can be used interchangeably with $i = u + v + w$). In all groups, the N operators can be sorted into K categories called classes ($\{C^{(j)}\}$) such that $\{\hat{P}_i\} = \{C^{(0)}, \{C^{(1)}\} \dots \{C^{(K)}\}\}$. All groups require $C_0 = E$ — the identity operator, defined by $Ef(\mathbf{r}) = f(\mathbf{r})$. The operators belonging to the same class transform functions in the same way - a central result of group theory is that there exist K projection operators $\hat{\rho}^{(j)}$ for $j = 1 \dots K$, such that

$$\tilde{f}^{(j)} = \hat{\rho}^{(j)} f = \sum_{uvw} [\chi^{(j)}(\hat{P}_{uvw})]^* \hat{P}_{uvw} f$$

where $\chi^{(j)}(\hat{P}_{uvw})$ is a scalar that describes the transformation of a function according to the j^{th} irreducible representation of the symmetry group for the uvw^{th} symmetry operator. The key feature of the lattice-translation group is that it is cyclic (and hence also Abelian), meaning that

- (1) The application of the operators \hat{P}_{uvw} is commutative, meaning that

$$\hat{P}_{uvw}\hat{P}_{u'v'w'}f(\mathbf{r}) = \hat{P}_{u'v'w'}\hat{P}_{uvw}f(\mathbf{r})$$
- (2) Sequential application of the same few symmetry operators generates all other operators the group via $\hat{P}_{uvw}f(\mathbf{r}) = (\hat{P}_{100})^u(\hat{P}_{010})^v(\hat{P}_{001})^wf(\mathbf{r}) = f(\mathbf{r} + \mathbf{T}_{uvw})$
- (3) As a by-product of (1), each of the L^3 symmetry operators forms a class by itself so the irreducible representation label j and symmetry operator labels uvw are interchangeable.

The values of the symmetry operators can be found by noting that if a function is translated L units in the x -, y -, or z -direction, it returns to the same spot, so $(\hat{P}_{100})^L = (\hat{P}_{010})^L = (\hat{P}_{001})^L = 1 \Rightarrow \hat{P}_{uvw} = e^{\frac{2\pi i}{L}(u+v+w)}$ — the symmetry operators are simply scalar values (the L^{th} roots of unity). It is also true that for cyclic groups, the symmetry operators *are* the transformation scalars and that there exists only one unique transformed result from application of all $j = 1 \dots K$ projection operators. The result is that in systems for which the governing equations follow lattice-translational symmetry, any function of space $f(\mathbf{r})$ has a symmetry-adapted expansion $\tilde{f}^{(uvw)}$ (with $\tilde{f}^{(uvw)}(\mathbf{r} + \mathbf{T}_{uvw}) = \tilde{f}^{(uvw)}(\mathbf{r})$):

$$\tilde{f}^{(uvw)}(\mathbf{r}) = \sum_{uvw} [\chi^{(uvw)}(\hat{P}_{uvw})]^* \hat{P}_{uvw} f(\mathbf{r}) = \sum_{uvw} e^{-\frac{2\pi i}{L}(u+v+w)} \hat{P}_{uvw} f(\mathbf{r}) \quad u, v, w = 0 \dots L - 1$$

Since the L^{th} roots of units form an orthonormal basis, this expression can be inverted (by multiplying an exponential on both sides) to obtain Bloch's Theorem

$$\hat{P}_{uvw}f(\mathbf{r}) = e^{\frac{2\pi i}{L}(u'+v'+w')} \tilde{f}^{(uvw)}(\mathbf{r}) = f(\mathbf{r} + \mathbf{T}_{uvw})$$

$$f(\mathbf{r} + \mathbf{T}) = e^{i\mathbf{k} \cdot \mathbf{T}} \tilde{f}^{(\mathbf{k})}(\mathbf{r})$$

where is customary in solid-state physics to define the crystal momentum in reciprocal lattice space \mathbf{k} such that $\mathbf{k} \equiv \mathbf{k}_{uvw} = u\mathbf{b}_1 + v\mathbf{b}_2 + w\mathbf{b}_3$ so that $\mathbf{k}_{uvw} \cdot \mathbf{T}_{uvw} = \frac{2\pi i}{L}(u + v + w)$ where \mathbf{b}_i are the reciprocal lattice vectors. This gives \mathbf{k} its intuitive meaning as a propagation direction in reciprocal space (and hence a corresponding direction in real space), but its introduction into the problem is purely a result of symmetry.

Any function that obeys Bloch's Theorem can be written as

$$f(\mathbf{r}) = e^{i\mathbf{k} \cdot \mathbf{r}} \tilde{f}^{(\mathbf{k})}(\mathbf{r})$$

where $\tilde{f}^{(k)}(r)$ is periodic in the lattice. This is easily verified by substituting this form into Bloch's Theorem. If we consider $r \rightarrow r + T$ in the above expression

$$\begin{aligned} f(r + T) &= e^{ik \cdot (r+T)} \tilde{f}^{(k)}(r + T) \\ &= e^{ik \cdot T} \left(e^{ik \cdot r} \tilde{f}^{(k)}(r + T) \right) \\ &= e^{ik \cdot T} \left(e^{ik \cdot r} \tilde{f}^{(k)}(r) \right) \\ &= e^{ik \cdot T} f(r) \end{aligned}$$

which recovers the original form of Bloch's Theorem.

In the limit that $L \rightarrow \infty$, the spacings between reciprocal lattice points becomes infinitesimally small and the k vector becomes a continuous variable in reciprocal space defined over the range $k_i \in [-\pi/a_i, \pi/a_i]$ where a_i is the lattice spacing in the i^{th} real space direction. A crystal bandstructure is a plot of the wave frequency as a function of propagation direction in reciprocal space, k . It is entirely analogous to calculating the dispersion relation $\omega(k)$ of a traveling plane wave described by $\exp(i(k \cdot r - \omega t))$. As a consequence of discrete-translational symmetry, the crystal momentum is conserved modulo $2\pi/a_i$ in any direction; hence, the bandstructure is only considered over a symmetric repeat unit in reciprocal space called the Brillouin zone (BZ). The shape of the BZ depends on the real-space lattice. For the simple square lattice investigated in this work, $a_1 = a_0 E_x$ and $a_2 = a_0 E_y \Rightarrow b_1 = 2\pi/a_0 E_x$ and $b_2 = 2\pi/a_0 E_y$ so the BZ is also a square with side-lengths reciprocally related to the real-space repeat unit. It turns out that the BZ is often symmetric itself and there exists an even smaller unit of k -space that uniquely defines the frequency response - this is called the irreducible Brillouin zone (IBZ). High-symmetry points in the IBZ are given conventional letters and bandstructures are plotted along a high-symmetry path through these points for multidimensional problems, as shown in Fig. 1.

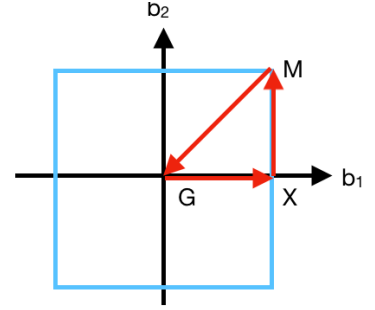


Figure 1: Diagram of BZ of 2D square lattice. The IBZ is defined by area enclosed in the red arrows. The customary path to travel is from $G \rightarrow X \rightarrow M \rightarrow G$

III. Finite Element Formulation

This work considers conservation of linear momentum on a square domain Ω subject to periodic boundary conditions in the x - and y -directions.

$$\nabla \cdot \sigma + \vec{b} = \rho \vec{u}_{,tt}$$

The weak form for this equation is found by making use of (1) the elastic constitutive law $\vec{\sigma} = C \nabla \vec{u}$ where C is the 4th-rank tensor of elastic constants and (2) Green's theorem to rewrite the integration by parts into a surface integral boundary term.

$$\int_{\Omega} \nabla \cdot (f \nabla g) d\Omega = \int_{\Omega} f(\nabla \cdot \nabla g) d\Omega + \int_{\Omega} \nabla g \cdot \nabla f d\Omega = \int_{\partial\Omega} f(\nabla g \cdot \hat{n}) dS + \int_{\Omega} \nabla g \cdot \nabla f d\Omega$$

Following these manipulations, we arrive at the familiar form form amenable to FEA shape function substitution:

$$\int_{\Omega} w \rho \vec{u}_{,tt} d\Omega + \int_{\Omega} \nabla w C \nabla \vec{u} d\Omega = \int_{\Omega} w \vec{b} d\Omega$$

Note that the boundary term from integration by parts vanishes since the displacement has its essential boundary conditions enforced along the perimeter of the simulation domain. Applying Bloch's theorem to the displacement, weight function, and body force, we can expand out the terms in the weak form to obtain (note that all Bloch-symmetric quantities are designated with a tilde):

$$e^{2i(k_x x + k_y y)} \int_{\Omega} \tilde{w} \rho \vec{u}_{,tt} d\Omega + e^{2i(k_x x + k_y y)} \left(\int_{\Omega} \nabla \tilde{w} C \nabla \vec{u} d\Omega + i \vec{k} \cdot \int_{\Omega} \tilde{w} C \vec{u} d\Omega \right) = e^{2i(k_x x + k_y y)} \int_{\Omega} \tilde{w} \vec{b} d\Omega$$

Following the Bubnov-Galerkin method, the displacement and weight function are expanded in the same shape function basis consisting of bilinear quadrilateral functions: $\vec{u} = N\Delta$. The derived discrete form is given by:

$$\left(\int_{\Omega} N^T \rho N d\Omega \right) \Delta_{,tt} + \left(\int_{\Omega} B^T C B d\Omega + i \vec{k} \cdot \int_{\Omega} N^T C N d\Omega \right) \Delta = \int_{\Omega} N^T b d\Omega$$

This can be condensed into a more familiar form by defining an augmented B-matrix (per node) such that

$$\tilde{B}_I = \begin{bmatrix} \frac{\partial N_I}{\partial x} & 0 \\ 0 & \frac{\partial N_I}{\partial y} \\ \frac{\partial N_I}{\partial y} & \frac{\partial N_I}{\partial x} \end{bmatrix} + \begin{bmatrix} i k_x N_I & 0 \\ 0 & i k_y N_I \\ i k_y N_I & i k_x N_I \end{bmatrix}$$

and as usual, the element B-matrix is formed by concatenating the 4 nodal B-matrices (or as many matrices as there are elemental nodes): $\tilde{B} = [\tilde{B}_1, \tilde{B}_2, \tilde{B}_3, \tilde{B}_4]$. An element with such a B-matrix is known as a Bloch element. With this form, the finite-element equations are recast into the standard form from linear analysis.

$$\left(\int_{\Omega} N^T \rho N d\Omega \right) \Delta_{,tt} + \left(\int_{\Omega} \tilde{B}^T C \tilde{B} d\Omega \right) \Delta = \int_{\Omega} N^T \tilde{b} d\Omega$$

$$M \Delta_{,tt} + \tilde{K}(k) \Delta = \tilde{F}$$

The only difference here is that by virtue of lattice translational symmetry, there are additional terms that introduce a dependence on \mathbf{k} in the stiffness matrix. To solve for the dispersion relation of the propagating wave ($\omega(k)$), we can apply a spectral decomposition of the displacement vector:

$$(\tilde{K}(k) - \omega^2 M) \Delta = \sum_j \tilde{f}_j e^{i\omega_j t}$$

The natural band structure of a lattice material is found by taking the free oscillation case (no force applied), resulting in the following system of equations for the dispersion relation.

$$\det(\tilde{K} - \omega^2 M) = 0$$

When programming FEA in more than 1 dimension, care must be taken to ensure a consistent ordering of the nodal displacement vector Δ for any elemental or global calculation. In particular, Voigt notation is used for elements (required via B-matrix construction) and “x then y” ordering is used for global matrices. Examples of the two different orderings are shown below:

$$\Delta_{element} = \begin{bmatrix} \Delta_{1x} \\ \Delta_{1y} \\ \Delta_{2x} \\ \Delta_{2y} \\ \vdots \end{bmatrix} \quad \Delta_{global} = \begin{bmatrix} \Delta_{1x} \\ \Delta_{2x} \\ \vdots \\ \Delta_{1y} \\ \Delta_{2y} \\ \vdots \end{bmatrix}$$

We also recall that the elastic constants tensor takes on the simple form of a 3 x 3 matrix in 2D Voigt notation:

$$C = \begin{bmatrix} \lambda + 2\mu & \lambda & 0 \\ \lambda & \lambda + 2\mu & 0 \\ 0 & 0 & \mu \end{bmatrix}$$

where λ and μ are the Lamé parameters defined by

$$\lambda = \frac{\nu E}{(1 + \nu)(1 - 2\nu)} \text{ and } \mu = \frac{E}{2(1 + \nu)}$$

where E is the Young's modulus and ν is the Poisson ratio. In acoustic applications, it is typically preferable to describe displacement in terms of longitudinal (out-of-plane) or transverse (in-plane) wave propagation.

$$u = u^L + u^T \Rightarrow u_{,tt}^L = c_L^2 \nabla^2 u^L \quad u_{,tt}^T = c_T^2 \nabla^2 u^T$$

$$\nabla \times u^L = 0 \quad \nabla \cdot u^T = 0$$

$$c_L^2 = \frac{\lambda + 2\mu}{\rho} \quad c_T^2 = \frac{\mu}{\rho}$$

Solving for the longitudinal and transverse modes independently using the same Bloch element FE formulation amounts to solving the original elastodynamic equations in an uncoupled orthogonal system of coordinates. As we will see, this separation of wave polarization is necessary to observe the bandgaps that can appear in these systems.

IV. Periodic Boundary Conditions

It is necessary to impose periodic boundary conditions on nodes in the model. Intuitively, periodic boundary conditions wrap the square domain back into itself, effectively allowing nodes outside the simulation domain to be accessed as values of nodes in the simulation domain. This is implemented by dividing the domain into 9 different regions with corresponding nodal vectors, as shown in Fig 2. The displacements are arranged into 9 nodal supervectors and a block-transformation matrix can be defined that maps \mathbf{v}_6 to \mathbf{v}_2 , \mathbf{v}_5 to \mathbf{v}_1 , and $\mathbf{v}_7, \mathbf{v}_8, \mathbf{v}_9$ to \mathbf{v}_3 .

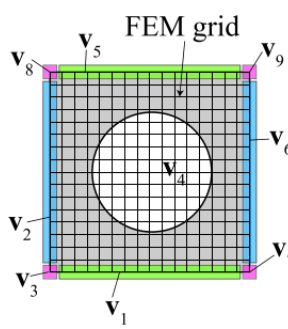
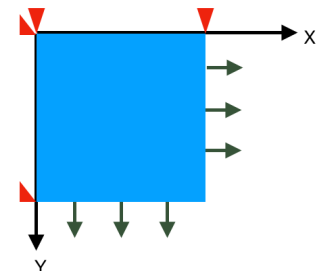
$$T = \begin{bmatrix} I_{n_B \times n_B} & 0_{n_B \times n_L} & 0_{n_B \times 2} & 0_{n_B \times n_i} \\ 0_{n_L \times n_B} & I_{n_L \times n_L} & 0_{n_L \times 2} & 0_{n_L \times n_i} \\ 0_{2 \times n_B} & 0_{2 \times n_L} & I_{2 \times 2} & 0_{2 \times n_i} \\ 0_{n_i \times n_B} & 0_{n_i \times n_L} & 0_{n_i \times 2} & I_{n_i \times n_i} \\ I_{n_B \times n_B} & 0_{n_B \times n_L} & 0_{n_B \times 2} & 0_{n_B \times n_i} \\ 0_{n_L \times n_B} & I_{n_L \times n_L} & 0_{n_L \times 2} & 0_{n_L \times n_i} \\ 0_{2 \times n_B} & 0_{2 \times n_L} & I_{2 \times 2} & 0_{2 \times n_i} \\ 0_{2 \times n_B} & 0_{2 \times n_L} & I_{2 \times 2} & 0_{2 \times n_i} \\ 0_{2 \times n_B} & 0_{2 \times n_L} & I_{2 \times 2} & 0_{2 \times n_i} \end{bmatrix}$$


Figure 2: Partition of domain into 9 supervectors in order to apply periodic boundary conditions

The boundary conditions are imposed on global matrices via linear transformation: $K^{per} = T^T K T$ and $M^{per} = T^T M T$ and the result is diagonalized using a generalized eigenvalue solver to extract the frequencies.

V. Patch Test Verification

In order to verify whether the FEA program was working, a simple



test case was developed. Since the periodic Bloch element does not have a well-known patch test, a regular 2D elasticity problem was tested instead. This is still reasonable validation for a working program because the key features like proper domain meshing, element matrix calculation, and global matrix assembly are independent of whether the regular or augmented B-matrix is used in the calculation. The following elastic plate is considered under plane stress conditions. For this problem $E = 1$ and $\nu = 0$. It is constrained in the x directions at points $(0, 0)$ and $(0, L)$ and it is constrained in the y direction at points $(0, 0)$ and $(L, 0)$. It is being pulled uniformly across both free faces by a force per unit length $P = 10$. From elasticity theory, we know that the strains should be constant throughout the body of the plate equal to $\epsilon_{xx} = \epsilon_{yy} = 2\epsilon_{xy} = P/E = 10$ with deviation from ideality near the constrained points. Along the plate diagonal, polar displacements can be measured with known strains of $\epsilon_{rr} = E^{-1}P\sqrt{2}$ and $\epsilon_{r\theta} = 0$. These solutions will be used as benchmarks when evaluating displacements through the body and computing the slope across different axes. The following is the displacement distribution at each node in the plate for a mesh of 100×100 elements.

Figure 3: Schematic of 2D elastic plate loading and boundary conditions used for patch test

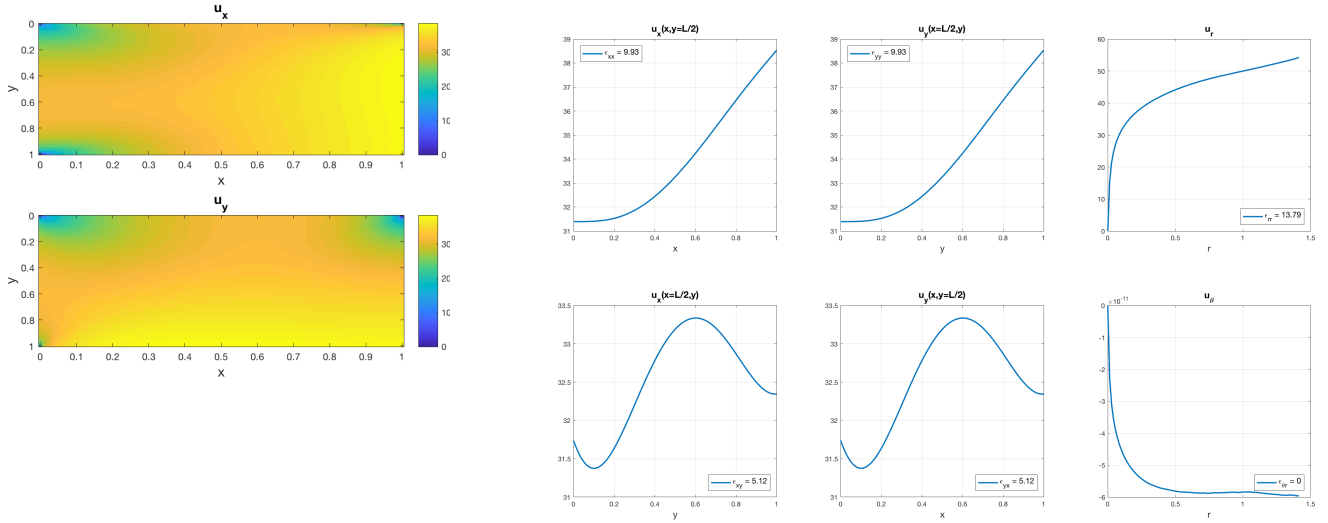
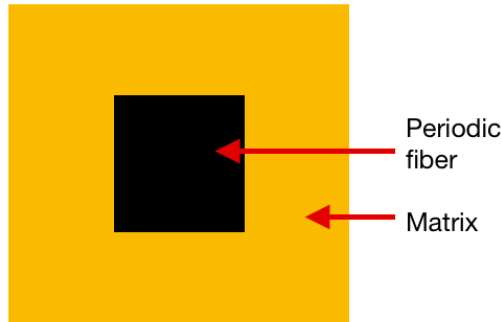


Figure 4: Left: displacement field over 2D elastic plate for 100×100 mesh. Right: Displacement profiles along various directions with measured slopes to verify in center of plate constant strains

The results of the patch test were converged with respect to mesh density and 100×100 produced consistent measures of displacement slope (strain). The displacement profiles of u_x and u_y show a linear region of constant strain along both the x- and y-directions across the bulk of the plate. The slopes of these linear regions were calculated and they match almost exactly with the analytical prediction. The same is true of the polar coordinate strains evaluated along the body diagonal. With this verification that the 2D elasticity FEA code is working properly, we will move on to the modeling of a phononic crystal bandstructure using modal analysis following the formulation outlined in part III.

VI. Phononic Crystal Bandstructure

The phononic crystal was created using a $20 \text{ cm} \times 20 \text{ cm}$ square domain meshed with a 9×9 element grid as a representative volume element (RVE). Elements in the center were designated with



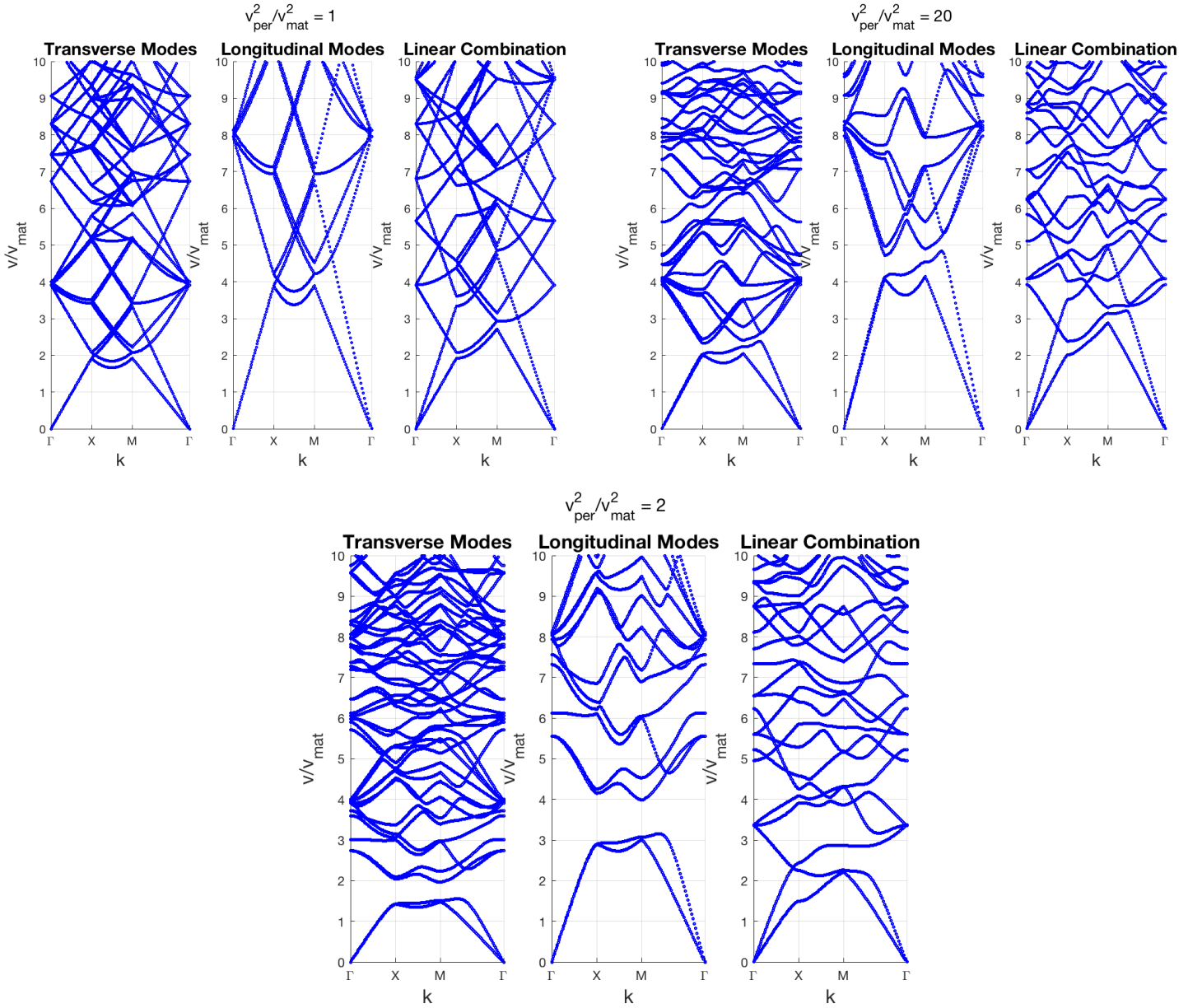
material properties of the periodic fiber material while the rest were given the properties of the matrix. The material properties for the matrix and periodic fiber (E and ρ) are defined such that they are normalized to the matrix values; therefore, we are only concerned with analyzing the bandstructure as a function of v_{per}/v_{mat} , the wavespeed contrast between fiber and matrix

Figure 6: Calculated band structures from FEA model at various wavespeed contrasts between matrix and fiber. Note that a bandgap opens for finite contrast and reduces as phases become increasingly dissimilar.

material. Recall that the nominal wavespeed in medium is defined by $v = \sqrt{E/\rho}$. The RVE has an areal ratio of 1:9 of fiber to matrix. The band structure was calculated for RVEs of different wavespeed contrasts. The results show that a gap indeed opens at a contrast of 2. The bandgap then decreases as the fiber and matrix become less acoustically similar. Such an effect could be expected as two dissimilar materials

would not allow for coherent wave transmission at the interface; by the time the fiber is 20 times stiffer than the matrix, the fiber centers are effectively glorified scattering centers and the bandgap disappears. Note that even in this case, there are still distinct non-overlapping bands but the bandgap is gone, meaning that there is no interval of frequency for which no state (k-vector) exists. We also point out that the bandgap cannot be seen when looking at the bandstructure corresponding to mixed transverse and longitudinal modes. These curves result from solving the full elastodynamic equations without separating into orthogonal polarization coordinates. The

Figure 5: Schematic of FEA model used to simulate phononic crystal



linear combination diagrams clearly combine features of both transverse and longitudinal modes, but they describe propagation of waves of mixed polarization. This analysis tells us that only waves of entirely transverse or longitudinal propagation will encounter a bandgap in the phononic crystal.

VII. Conclusion

Devices that implement phononic crystals include noise-cancelling materials and acoustic waveguides. Designing these materials typically requires placing the frequency stop-band at a desired range. In the current model, the absolute position of the stop band was not able to be shifted, however its size can be modulated by changing the wavespeed contrast. In general, this model has other parameters that can affect the bandstructure beyond wavespeed contrast, including E and ρ of each material, the domain size, the areal fraction of fiber, and lattice

topology of the RVE. The model considered in this work represents a simple cubic 2D crystal but one could investigate the effects of BCC, FCC, or even more complex arbitrary symmetric tilings. Such considerations, while extensive, can be performed within the 2D FEA model developed in this work.

VIII. References

- [1] M Hussein. Dynamics of Banded Materials and Structures: Analysis, Design, and Computation in Multiple Scales. PhD diss, University of Michigan, 2004.
- [2] M Hussein. Theory of damped Bloch waves in elastic media. Phys Rev B 80. 2009.
- [3] M Hussein et al. Dynamics of Phononic Materials and Structures: Historical Origins, Recent Progress, and Future Outlook. Appl. Mech. Rev 66. 2014.
- [4] M Hussein. Reduced Bloch mode expansion for periodic media band structure calculations. Proc R Soc A. 2009.
- [5] W Axmann et al. An Efficient Finite Element Method for Computing Spectra of Photonic and Acoustic Band-Gap Materials. J Comp Physics 150. 1999.
- [6] I Veres et al. Complex band structures of two dimensional phononic crystals: Analysis by the finite element method. J Appl Phys 114. 2013

Appendix: Matlab Code

myPhononBandStructure_v7.m

```
clear all
clc
close all
% Define side length of simulation box
L = 0.02;
```

```

% Periodic lattice spacing (distance between periodic structures)
a = [L,L];
% elastic properties of matrix and lattice material, respectively
youngs = [1.8,1];
poisson = [0.34, 0.34];
rho = [1,1];
% set mesh density
Nx = 9;
Ny = 9;
% generate mesh
elements = my2DMesh_small(L,Nx,Ny,youngs,poisson,rho);
% mesh{I,1} = 4x2 matrix of element I's nodal coordinates in set ordering
% down_left -> down_right -> up_right -> up_left (1->2->3->4)
% mesh{I,2} = element connectivity for local nodes in element I

% kspace mesh density
nkpt = 50;

% number of eigenfrequencies we will obtain
numnodes = (Nx+1)*(Ny+1);
ndof = 2*numnodes;

% resolution of k-space plotting grid
s = linspace(0,1,nkpt);

% high symmetry points in space
kG = [0,0]';
kX = [pi/a(1), 0]';
kM = [pi/a(1),pi/a(2)]';

% Define path vector in k-space
kGtoX = (kG + s.*(kX - kG));
k1 = norm(kGtoX(:,end)- kGtoX(:,1));
kXtoM = (kX + s.*(kM - kX));
k2 = norm(kXtoM(:,end) - kXtoM(:,1));
kMtoG = (kM + s.*(kG - kM));
k3 = norm(kMtoG(:,end)- kMtoG(:,1));

% plotting k-vectors, normalized to length in k-space
k1path = linspace(0,k1,nkpt);
k2path = linspace(k1,k1+k2,nkpt);
k3path = linspace(k1+k2,k1+k2+k3,nkpt);
ktot = k1+k2+k3;
dk = ktot/(3*nkpt);

kcoord = [k1path, k2path, k3path];
kvec = [kGtoX, kXtoM, kMtoG];

nl = Ny - 1;    nb = Nx - 1;    ni = numnodes - 2*nl - 2*nb - 4;
nl = 2*nl;    nb = 2*nb;    ni = 2*ni;
nfree = nl + nb + ni + 2;

%Kassemble = zeros(nfree,nfree,length(kcoord));
freqL = zeros(nfree,length(kcoord));
freqT = freqL;
freqtot = freqL;
orderVec = myNineSetOrder(numnodes,Nx,Ny);

tic
for i = 1:length(kcoord)

    [Ktot,Mtot] = myKMblochelement(kvec(:,i),elements,Nx,Ny);
    [KL,KT,M] = myKMbloch_long_trans(kvec(:,i),elements,Nx,Ny);

    [KLp, Mp] = myPeriodicBC(numnodes,Nx,Ny,KL(orderVec,orderVec),M(orderVec,orderVec));
    [KTp, Mp] = myPeriodicBC(numnodes,Nx,Ny,KT(orderVec,orderVec),M(orderVec,orderVec));
    [Kptot, Mptot] =
myPeriodicBC(numnodes,Nx,Ny,Ktot(orderVec,orderVec),Mtot(orderVec,orderVec));

    [VL,DL] = eigs(KLp,Mp,nfree);
    [VT,DT] = eigs(KTp,Mp,nfree);
    [Vtot,Dtot] = eigs(Kptot,Mptot,nfree);

```

```

    freqL(:,i) = diag(DL);
    freqT(:,i) = diag(DT);
    freqtot(:,i) = diag(Dtot);

    %Kassemble(:, :, i) = Kp*Mp^-1;

end
toc

%[Vseq,freq] = eigenshuffle(Kassemble);

%% Plotting
close all

nmodes = nfree;

vm = sqrt(youngs(2)/rho(2));

omegaL = sqrt(abs(freqL));
omegaL = omegaL*L./vm;

omegaT = sqrt(abs(freqT));
omegaT = omegaT*L./vm;

omegaTot = sqrt(abs(freqtot));
omegaTot = omegaTot*L./vm;

figure(1)
subplot(131)
hold on
for j = 1:length(kcoord)

    plot(kcoord(j), omegaT(end-nmodes+1:end, j), 'bs', 'MarkerSize', 2)

end
hold off
xlabel('k', 'FontSize', 15)
ylabel('v/v_{mat}', 'FontSize', 15)
xticks([0 k1, k1+k2, k1+k2+k3])
xticklabels({'\Gamma', 'X', 'M', '\Gamma'})
title('Transverse Modes', 'FontSize', 15)
axis([0 k1+k2+k3 0 10])
grid on

subplot(132)
hold on
for j = 1:length(kcoord)

    plot(kcoord(j), omegaL(end-nmodes+1:end, j), 'bs', 'MarkerSize', 2)

end
hold off
xlabel('k', 'FontSize', 15)
ylabel('v/v_{mat}', 'FontSize', 15)
xticks([0 k1, k1+k2, k1+k2+k3])
xticklabels({'\Gamma', 'X', 'M', '\Gamma'})
title('Longitudinal Modes', 'FontSize', 15)
axis([0 k1+k2+k3 0 10])
grid on

subplot(133)
hold on
for j = 1:length(kcoord)

    plot(kcoord(j), omegaTot(end-nmodes+1:end, j), 'bs', 'MarkerSize', 2)

```

```

end
hold off
xlabel('k','FontSize',15)
ylabel('v/v_{mat}','FontSize',15)
xticks([0 k1, k1+k2, k1+k2+k3])
xticklabels({'\Gamma','X','M','\Gamma'})
title('Linear Combination','FontSize',15)
axis([0 k1+k2+k3 0 10])
grid on

suptitle('v^2_{per}/v^2_{mat} = 1.8')

```

myKMbloch_long_trans.m

```

function [KL,KT,M] = myKMbloch_long_trans(kvec,elements,Nx,Ny)

k1 = kvec(1);    k2 = kvec(2);

numnodes = (Nx+1)*(Ny+1);
numel = Nx*Ny;
nen = 4;

% shape functions and derivatives on isoparametric domain
n1 = @(xi) 1/4*(1-xi(1)).*(1-xi(2));    dn11 = @(xi) -1/4*(1-xi(2));    dn12 = @(xi) -1/4*(1-
xi(1));
n2 = @(xi) 1/4*(1+xi(1)).*(1-xi(2));    dn21 = @(xi) 1/4*(1-xi(2));    dn22 = @(xi)
-1/4*(1+xi(1));
n3 = @(xi) 1/4*(1+xi(1)).*(1+xi(2));    dn31 = @(xi) 1/4*(1+xi(2));    dn32 = @(xi)
1/4*(1+xi(1));
n4 = @(xi) 1/4*(1-xi(1)).*(1+xi(2));    dn41 = @(xi) -1/4*(1+xi(2));    dn42 = @(xi) 1/4*(1-
xi(1));

% gaussian quadrature points
xg = [-1/sqrt(3), -1/sqrt(3);
       1/sqrt(3), -1/sqrt(3);
       1/sqrt(3), 1/sqrt(3);
       -1/sqrt(3), 1/sqrt(3)];
wt = [1, 1, 1, 1];
ng = length(wt);

KT = zeros(2*numnodes,2*numnodes);
KL = zeros(2*numnodes,2*numnodes);
M = zeros(2*numnodes,2*numnodes);

%K = sparse(2*numnodes,2*numnodes);
%M = sparse(2*numnodes,2*numnodes);

% loop over the elements and assemble global matrices
for jj = 1:numel

    % preallocate element matrices
    me = zeros(2*nen,2*nen);
    keL = zeros(2*nen,2*nen);
    keT = zeros(2*nen,2*nen);

    % element coordinates in (X,Y) reference configuration (4x2 matrix)
    X = elements{jj,1};

    % element mechanical properties
    E = elements{jj,3};    % young's modulus
    v = elements{jj,4};    % poisson ratio
    rho = elements{jj,5};    % density
    % lame parameters
    lambda = (v*E)/( (1+v)*(1-2*v) );
    mu = E/(2*(1+v));

    % construct jacobian matrix to get from isoparametric to reference

```

```

% confi
jacobian = @(xig) [X(1,1)*dn11(xig) + X(2,1)*dn21(xig) + X(3,1)*dn31(xig) + X(4,1)*dn41(xig),
X(1,1)*dn12(xig) + X(2,1)*dn22(xig) + X(3,1)*dn32(xig) + X(4,1)*dn42(xig);
X(1,2)*dn11(xig) + X(2,2)*dn21(xig) + X(3,2)*dn31(xig) + X(4,2)*dn41(xig),
X(1,2)*dn12(xig) + X(2,2)*dn22(xig) + X(3,2)*dn32(xig) + X(4,2)*dn42(xig)];

% local to global coordinate mapping (1x4 vector)
sctr = [elements{jj,2};
elements{jj,2}+numnodes];
% Implement mapping from element matrix ordering [u1x, u1y, u2x, u2y,...] to
% global matrix ordering [u1x, u2x, u3x, ... u1y, u2y, u3y,...]
% sctrVec = [sctr(1), sctr(1) + numnodes, sctr(2), sctr(2) + numnodes, sctr(3), sctr(3) +
numnodes, sctr(4), sctr(4) + numnodes];
% sctrVec gets elements in K, M, F that line up with Voigt notation
% ordering in element vectors
sctrVec = sctr(:)';

% Loop over integration points
for g = 1:ng

    % gauss integration point in isoparametric domain
    xig = xg(g,:);

    jac = jacobian(xig);
    jac = jac';
    jacin = jac^-1;
    JX = det(jac);

    % use jacobian to express shape function derivatives over
    % reference domain in terms of isoparametric derivatives
    dn1x = dn11(xig)*jacin(1,1) + dn12(xig)*jacin(2,1);
    dn1y = dn11(xig)*jacin(1,2) + dn12(xig)*jacin(2,2);

    dn2x = dn21(xig)*jacin(1,1) + dn22(xig)*jacin(2,1);
    dn2y = dn21(xig)*jacin(1,2) + dn22(xig)*jacin(2,2);

    dn3x = dn31(xig)*jacin(1,1) + dn32(xig)*jacin(2,1);
    dn3y = dn31(xig)*jacin(1,2) + dn32(xig)*jacin(2,2);

    dn4x = dn41(xig)*jacin(1,1) + dn42(xig)*jacin(2,1);
    dn4y = dn41(xig)*jacin(1,2) + dn42(xig)*jacin(2,2);

    %dn1x = 2/(L/N)*(dn11(xig) + dn12(xig));
    %dn1y = 2/(L/N)*(dn11(xig) + dn12(xig));
    %dn2x = 2/(L/N)*(dn21(xig) + dn22(xig));
    %dn2y = 2/(L/N)*(dn21(xig) + dn22(xig));
    %dn3x = 2/(L/N)*(dn31(xig) + dn32(xig));
    %dn3y = 2/(L/N)*(dn31(xig) + dn32(xig));
    %dn4x = 2/(L/N)*(dn41(xig) + dn42(xig));
    %dn4y = 2/(L/N)*(dn41(xig) + dn42(xig));

    % Construct the B-matrix in Voigt notation [dn1/dx, 0; 0, dn1/dy, dn1/dy, dn1/dx]
    % evaluate on isoparametric domain, use jacobian to change
    % integration coordinate
    Bel = [dn1x, 0;
0, dn1y;
dn1y, dn1x];

    Bek1 = [1i*k1*n1(xig), 0;
0, 1i*k2*n1(xig);
1i*k2*n1(xig), 1i*k1*n1(xig)];

    Be2 = [dn2x, 0;
0, dn2y;
dn2y, dn2x];

    Bek2 = [1i*k1*n2(xig), 0;
0, 1i*k2*n2(xig);
1i*k2*n2(xig), 1i*k1*n2(xig)];

    Be3 = [dn3x, 0;

```



```

        0, dn3y;
        dn3y, dn3x];

Bek3 = [li*k1*n3(xig), 0;
        0, li*k2*n3(xig);
        li*k2*n3(xig), li*k1*n3(xig)];

Be4 = [dn4x, 0;
        0, dn4y;
        dn4y, dn4x];

Bek4 = [li*k1*n4(xig), 0;
        0, li*k2*n4(xig);
        li*k2*n4(xig), li*k1*n4(xig)];

% element B-matrix
Be = [Be1 + Bek1, Be2 + Bek2, Be3 + Bek3, Be4 + Bek4];

% element nodal matrix: orders in {ulx, uly, u2x, u2y, ... }
Ne = [ n1(xig), 0, n2(xig), 0, n3(xig), 0, n4(xig), 0;
       0, n1(xig), 0, n2(xig), 0, n3(xig), 0, n4(xig)];

% compute element mass matrix
me = me + wt(g)*(Ne')*rho*Ne*JX;

% compute element stiffness matrix

%C = [lambda + 2*mu, lambda, 0;
      lambda, lambda+2*mu, 0;
      0, 0, mu];
% plane strain
%C = lambda/v * [1-v, v, 0;
                v, 1-v, 0;
                0, 0, (1-2*v)/2];
% plane stress
%C = E/(1-v^2)*[ 1 v 0;
                v 1 0;
                0 0 (1-v)/2];

% longitudinal & transverse wave speeds
cL = sqrt( (lambda + 2*mu)/rho );
cT = sqrt( mu/rho );

keL = keL + wt(g)*Be'*cL^2*Be*JX;
keT = keT + wt(g)*Be'*cT^2*Be*JX;

end

% scatter into global matrices
KT(sctrVec,sctrVec) = KT(sctrVec,sctrVec) + keT;
KL(sctrVec,sctrVec) = KL(sctrVec,sctrVec) + keL;
M(sctrVec,sctrVec) = M(sctrVec,sctrVec) + me;

end

end

myPeriodicBC.m

function [Kp,Mp] = myPeriodicBC(numnodes,Nx,Ny,K,M)
% K and M must be in nine subvector set order!

% apply periodic b.c. (not adding bloch factors because u_k(r) is periodic
% in lattice (rve space)
% just mapping R -> L, T -> B, and 3 other corners -> Lb; interior nodes
% don't change
% number of dof associated with the left, bottom, and interior boundary

```

```

% nodes
nl = Ny - 1;    nb = Nx - 1;    ni = numnodes - 2*nl - 2*nb - 4;

nl = 2*nl;    nb = 2*nb;    ni = 2*ni;

% bottom    % left    % LB    % int
T_pbc = [ eye(nb,nb), zeros(nb,nl), zeros(nb,2), zeros(nb,ni);
          zeros(nl,nb), eye(nl,nl), zeros(nl,2), zeros(nl,ni);
          zeros(2,nb), zeros(2,nl), eye(2,2), zeros(2,ni);
          zeros(ni,nb), zeros(ni,nl), zeros(ni,2), eye(ni,ni);
          % condensed out dof: ( T, R, RB, LT, RT)
          eye(nb,nb), zeros(nb,nl), zeros(nb,2), zeros(nb,ni);
          zeros(nl,nb), eye(nl,nl), zeros(nl,2), zeros(nl,ni);
          zeros(2,nb), zeros(2,nl), eye(2,2), zeros(2,ni);
          zeros(2,nb), zeros(2,nl), eye(2,2), zeros(2,ni);
          zeros(2,nb), zeros(2,nl), eye(2,2), zeros(2,ni)];

Kp = T_pbc'*K*T_pbc;
Mp = T_pbc'*M*T_pbc;

```

```
end
```

myNineSetOrder.m

```

function [orderVec] = myNineSetOrder(numnodes,Nx,Ny)

% global node numbers for boundary and interior points
nodes_B = [2:Nx;
           2+numnodes:Nx+numnodes];
nodes_L = [Nx+2:Nx+1:numnodes-2*Nx-1;
           Nx+2+numnodes:Nx+1:numnodes-2*Nx-1+numnodes];
nodes_T = [numnodes-Nx+1:numnodes-1;
           numnodes-Nx+1+numnodes:numnodes-1+numnodes];
nodes_R = [2*(Nx+1):Nx+1:numnodes-Nx-1;
           2*(Nx+1)+numnodes:Nx+1:numnodes-Nx-1+numnodes];
nodes_LB = [1;
            1+numnodes];
nodes_LT = [numnodes-Nx;
            numnodes-Nx+numnodes];
nodes_RB = [Nx+1;
            Nx+1+numnodes];
nodes_RT = [numnodes;
            numnodes+numnodes];
NN = 1:numnodes;
NN( [nodes_B(1,:), nodes_L(1,:), nodes_T(1,:), nodes_R(1,:), nodes_LB(1,:), nodes_LT(1,:),
      nodes_RB(1,:), nodes_RT(1,:)] ) = [];
nodes_int = [NN;
             NN + numnodes];
% order is chosen based on the ordering of the transformation matrix
%orderVec = [nodes_L(:)', nodes_R(:)', nodes_B(:)', nodes_T(:)', nodes_LB(:)', nodes_RB(:)',
%            nodes_LT(:)', nodes_RT(:)', nodes_int(:)'];

orderVec =
[nodes_B(:)', nodes_L(:)', nodes_LB(:)', nodes_int(:)', nodes_T(:)', nodes_R(:)', nodes_RB(:)', nodes_LT
(:)', nodes_RT(:)'];

end

```

my2DElasticPlate.m

```

clear all
close all

```

```

% Define side length of simulation box
L = 1;
% Periodic lattice spacing (distance between periodic structures)
a = [L,L];
% elastic properties of matrix and lattice material, respectively
youngs = [1,1];
poisson = [0,0];
% Need high density contrast (>100) to get "band structure" looking things
rho = [1,1];
% set mesh density
N = 100;
Nx = N;
Ny = N;
nen = 4;
numel = Nx*Ny;
numnodes = (Nx+1)*(Ny+1);
% generate mesh
elements = my2DMesh(L,Nx,Ny,youngs,poisson,rho);
hx = L/Nx;  hy = L/Ny;

Papp = 10;
%Pload = Papp*[1/Nx, 1/Ny];
Pload = Papp*[1/Nx, 1/Ny];

[K,M] = myKijMij_v2(elements,Nx,Ny);

% create the patch test force vector
F = zeros(2*numnodes,1);
idq = [];
tol = 1e-10;

for J = 1:numel

isboundary = elements{J,6};
X = elements{J,1};

% local to global coordinate mapping (1x4 vector)
sctr = [elements{J,2};
        elements{J,2}+numnodes];
% Implement mapping from element matrix ordering [u1x, u1y, u2x, u2y,...] to
% global matrix ordering [u1x, u2x, u3x, ... u1y, u2y, u3y,...]
%sctrVec = [sctr(1), sctr(1) + numnodes, sctr(2), sctr(2) + numnodes, sctr(3), sctr(3) +
numnodes, sctr(4), sctr(4) + numnodes];
sctrVec = sctr(:)';

if isboundary

fe = zeros(2*nen,1);

% right hand side
if abs(X(2,1) - L) < tol
    fe(3) = Pload(1);  fe(5) = Pload(1);
    F(sctrVec) = fe;
    %idq = [idq, sctrVec(3), sctrVec(5)];
end

% top side
if abs(X(3,2) - L) < tol
    fe(6) = Pload(2);  fe(8) = Pload(2);
    F(sctrVec) = fe;
    %idq = [idq, sctrVec(6), sctrVec(8)];
end

% left side
%if X(1,1) == 0
%    fe(1) = -Pload(1);  fe(7) = -Pload(1);
%    F(sctrVec) = fe;
%    idq = [idq, sctrVec(1), sctrVec(7)];

```

```

%end
% bottom side
%if X(1,2) == 0
%   fe(2) = -Pload(2);   fe(4) = -Pload(2);
%   F(sctrVec) = fe;
%   idq = [idq, sctrVec(2), sctrVec(4)];
%end

end

end

% constrain x and y of lower left corner & constrain y of lower right
% corner & constrain x of upper left corner
idd = [1, Ny*(Nx+1)+1, 1+numnodes, Nx+1+numnodes];

idf = 1:2*numnodes;
idf(find(idf==idd(1))) = [];
idf(find(idf==idd(2))) = [];
idf(find(idf==idd(3))) = [];
idf(find(idf==idd(4))) = [];

u = zeros(2*numnodes,1);

u(idf) = K(idf,idf)\( F(idf) - K(idf,idd)*u(idd) );
F(idd) = K(idd,idf)*u(idf) + K(idd,idd)*u(idd);

%% formatting and plotting displacements
close all

UX = reshape(u(1:numnodes),[Ny+1,Nx+1]);   UX = UX';
UY = reshape(u(numnodes+1:2*numnodes),[Ny+1,Nx+1]); UY = UY';

uxy = UX(:,floor(end/2));   uxx = UX(floor(end/2),:);
uyx = UY(floor(end/2),:);   uyy = UY(floor(end/2),:);

uxdiag = diag(UX);   uydiag = diag(UY);

ur = 1/sqrt(2)*(uxdiag + uydiag);
uth = 1/sqrt(2)*(-uxdiag + uydiag);

xe = 0:hx:L;
ye = 0:hy:L;

figure(1)
subplot(211)
imagesc(xe,ye,UX)
xlabel('x','FontSize',14)
ylabel('y','FontSize',14)
title('u_x','FontSize',14)
colorbar
subplot(212)
imagesc(xe,ye,UY)
xlabel('x','FontSize',14)
ylabel('y','FontSize',14)
title('u_y','FontSize',14)
colorbar

figure(2)
subplot(231)
plot(linspace(0,L,Nx+1),uxx,'Linewidth',2)
xlabel('x','FontSize',14)
title('u_x(x,y=L/2)','FontSize',14)
ll = legend('\epsilon_{xx} = 9.93');   set(ll,'FontSize',12,'Location','NorthWest');
grid on
subplot(232)
plot(linspace(0,L,Ny+1),uyy,'Linewidth',2)
xlabel('y','FontSize',14)
title('u_y(x=L/2,y)','FontSize',14)
ll = legend('\epsilon_{yy} = 9.93');   set(ll,'FontSize',12,'Location','NorthWest');
grid on

```

```

subplot(234)
plot(linspace(0,L,Ny+1),uxy,'Linewidth',2)
xlabel('y','FontSize',14)
title('u_x(x=L/2,y)','FontSize',14)
ll = legend('\epsilon_{xy} = 5.12'); set(ll,'FontSize',12,'Location','SouthEast');
grid on
subplot(235)
plot(linspace(0,L,Ny+1),uyx,'Linewidth',2)
xlabel('x','FontSize',14)
title('u_y(x,y=L/2)','FontSize',14)
ll = legend('\epsilon_{yx} = 5.12'); set(ll,'FontSize',12,'Location','SouthEast');
grid on
subplot(233)
plot(linspace(0,L*sqrt(2),Nx+1),ur,'Linewidth',2)
xlabel('r','FontSize',14)
title('u_r','FontSize',14)
ll = legend('\epsilon_{rr} = 13.79'); set(ll,'FontSize',12,'Location','SouthEast');
grid on
subplot(236)
plot(linspace(0,L*sqrt(2),Nx+1),uth,'Linewidth',2)
xlabel('r','FontSize',14)
title('u_{\theta}','FontSize',14)
ll = legend('\epsilon_{\theta r} = 0'); set(ll,'FontSize',12,'Location','SouthEast');
grid on

```

my2DMesh.m

```

function [MESH] = my2DMesh(L,Nx,Ny,youngs,poisson,rho)

% MESH = numelx6 cell array with entries
% col 1: element nodal coordinates in reference config
% col 2: element location matrix for local to global mapping
% col 3: element young's modulus
% col 4: element poisson ratio
% col 5: element density
% col 6: 1 or 0 indicating whether it's a boundary element

youngs_matrix = youngs(2);      youngs_per = youngs(1);
poisson_matrix = poisson(2);    poisson_per = poisson(1);
rho_matrix = rho(2);            rho_per = rho(1);

numel = Nx*Ny;
hx = L/Nx;  hy = L/Ny;

MESH = cell(numel,6);
% col 1: nodal coords 4x2
% col 2: connectivity matrix

% local node numbering is counter clockwise 1, 2, 3, 4
loccoords = [0, 0;
             hx, 0;
             hx, hy;
             0, hy];
locglob = [1, 2, 2+(Nx+1), 1+(Nx+1)];

%numnodes = (Nx+1)*(Ny+1);

c = 1;
for i = 1:Nx
    for j = 1:Ny

        %Le = zeros(4,numnodes);
        MESH{c,1} = [ loccoords(:,1) + hx*mod(i-1,Nx), loccoords(:,2) + hy*mod(j-1,Ny)];
        LM = locglob + (j-1)*(Nx+1);
        %Le(1,LM(1)) = 1;
        %Le(2,LM(2)) = 1;
    end
end

```

```

%Le(3,LM(3)) = 1;
%Le(4,LM(4)) = 1;
MESH{c,2} = LM;

% if element does not lie along the boundary, fill it with periodic
% material
if i ~= 1 && i ~= Nx && j ~= 1 && j ~= Ny
    MESH{c,3} = youngs_per;
    MESH{c,4} = poisson_per;
    MESH{c,5} = rho_per;
    MESH{c,6} = 0;
else
% otherwise fill it with matrix (matrix = boundary)
    MESH{c,3} = youngs_matrix;
    MESH{c,4} = poisson_matrix;
    MESH{c,5} = rho_matrix;
    MESH{c,6} = 1;
end

%if mod(c,2) == 0
%MESH{c,3} = youngs_matrix;
%MESH{c,4} = poisson_matrix;
%MESH{c,5} = rho_matrix;
%else
%MESH{c,3} = youngs_per;
%MESH{c,4} = poisson_per;
%MESH{c,5} = rho_per;
%end
c = c+1;
end
locglob = locglob + ones(1,4);

end

end

```