Progress report

R Adhithya

General information

MOOSE (Multiphysics Object Oriented Simulation Environment) is an opensource finite element framework for creating massively parallel simulation tools. [1] It is maintained by Idaho National Laboratory (INL)

Working with MOOSE

To run simulations in MOOSE, an input file is used. The user is allowed to build applications using the provided source code incorporating different physics into the simulation as needed. The steps needed are explained below:

- *./moose/scripts/stork.sh <Name of project>* builds the applications
- Inside the application there is a makefile which can be edited to include different physics needed.
- *make -j4* command compiles the application and creates an executable. By default an optimized application is created. If the user wants to debug the code, *export METHOD=dbg* command has to be used before compilation.
- *./<executable name> -i <input file name>.i* is used to run the executable.

Input file format in MOOSE

The standard input file format is shown below:

- $[Mesh]$ - The information needed to generate mesh for the given problem. MOOSE is capable of generating 2-D mesh. The boundaries are given default names like left, right, top, bottom. This can be checked by using the command *./<app name> -i <input file>.i –mesh-only* while running the app. The mesh information can be saved separately in a file by using *./<app name> -i <input file>.i –mesh-only <meshfile name>.e*. Mesh generated using ABAQUS or GMSH can also be used.
- $[Variables]$ - Variables in the simulation are defined
- $[Kernels]$ - MOOSE uses the kernel system to run simulations. In the derivation of a standard FEM problem, after the test functions are substituted the biliniear and linear terms, $a(u, v)$ and $b(v)$ are inner products which are called kernels. Different kernels are available for different problems.
- $[BCs]$ - Boundary conditions needed is mentioned.
- $[Executioner]$ - The solver needed to solve the problem is specified.
- $[Outputs]$ - Moose has a built in post-processor tool but paraview can be used for any additional post processing

Debugging MOOSE applications

GNU GDB debugger can be used to debug MOOSE applications via command line. The follwing commands have to be used to debug MOOSE applications

- *gdb ./<executable name>* starts the debugger
- *break <source file>.C:<line number>* adds a breakpoint at the <line number>
- *run -i inputfile.i* to run the file within the gdb debugger.
- *continue* can be used to run the code until next breakpoint, *next* can be used to execute the code line by line, *step* can be used to step into any functions in the code. *info breakpoints* can be used to check the number of breakpoints *delete <breakpoint number>* deletes the corresponding breakpoint

Adding custom files to MOOSE

Every MOOSE application created had a source folder and an include folder. To write a custom file, a .C and a .h file have to be created and stored inside the src and include folders respectively before compiling using *make -j4*.

Problem Definition

To develop an extended spectral finite element method for band structure calculations in phononic crystals.[2] The

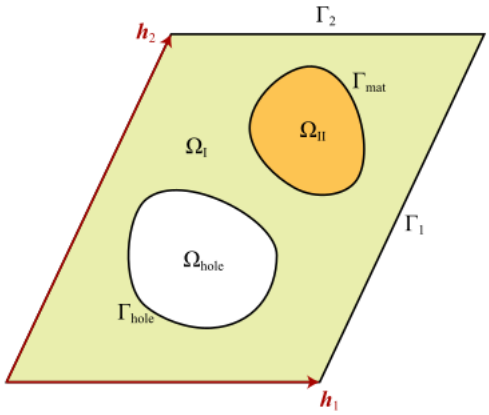

Fig. 1 Bloch-periodic boundary value problem with a hole and two distint materials

strong form for the above problem is obtained from the elastodynamic boundary-value problem.

$$\begin{aligned} \nabla.(\mathbf{C}(\mathbf{x}) : \nabla_s \mathbf{u}(\mathbf{x})) + \lambda\rho(\mathbf{x})\mathbf{u}(\mathbf{x}) &= 0 \\ \mathbf{u}(\mathbf{x}+\mathbf{h_i}) &= \mathbf{u}(\mathbf{x})\exp(i\mathbf{k}.\mathbf{h_i}) \\ \sigma(\mathbf{x}+\mathbf{h_i}).\mathbf{n}(\mathbf{x}+\mathbf{h_i}) &= \sigma(\mathbf{x}).\mathbf{n}(\mathbf{x})\exp(i\mathbf{k}.\mathbf{h_i}) \end{aligned} \tag{1}$$

where $\lambda = \omega^2$.

$$\begin{aligned} \sigma(\mathbf{x}).\mathbf{n}(\mathbf{x}) &= 0 \text{ on } \Gamma_{hole} \\ [[\mathbf{u}(\mathbf{x})]] &= 0 \text{ on } \Gamma_{mat} \\ [[\sigma(\mathbf{x}).\mathbf{n}(\mathbf{x})]] &= 0 \text{ on } \Gamma_{mat} \end{aligned} \tag{2}$$

The weak form is obtained using stardard use of test function and integrating eq. (1) by parts.

$$a(\mathbf{u},\mathbf{v}) + \frac{\gamma\kappa}{h^2} j(\mathbf{u},\mathbf{v}) = \lambda[b(\mathbf{u},\mathbf{v}) + \gamma_M j(\mathbf{u},\mathbf{v})] \quad \forall \in S \tag{3}$$

where,

$$\begin{aligned} a(\mathbf{u},\mathbf{v}) &:= \int_\Omega \nabla_\mathbf{s}\mathbf{v}^*(\mathbf{x}) : \mathbf{C}(\mathbf{x}) : \nabla_\mathbf{s}\mathbf{u}(\mathbf{x}) d\mathbf{x} \\ b(\mathbf{u},\mathbf{v}) &:= \int_\Omega \rho(\mathbf{x})\mathbf{v}^*(\mathbf{x}).\mathbf{u}(\mathbf{x}) dx \end{aligned} \tag{4}$$

$$\begin{aligned} S &:= \mathbf{u} \in [H^1(\Omega_I \cup \Omega_{II}]^2, [[\mathbf{u}]] \quad = 0 \; on \; \Gamma_{mat}, \\ \mathbf{u}(\mathbf{x}+\mathbf{h_i}) &= \mathbf{u}(\mathbf{x})\exp(i\mathbf{k}.\mathbf{h_i}) \\ K \times \mathbf{u}'(\mathbf{x}+\mathbf{h_i}) &= \mathbf{u}'(\mathbf{x_i})\exp(i\mathbf{k}.\mathbf{h_i}) \end{aligned} \tag{5}$$

where, $j(u,v)$ is the ghost penalty stabilization term used to improve matrix-conditioning. The above eigenvalue problem needs to be solved.

Solution Methodology

A 1-D problem shown in [2] is considered as the first step towards solving the problem. To obtain the dispersion curve eq. (4) needs to be solved for all the pairs of wave vector k. There are two ways of approaching the problem.[3]

- Direct approach : Real ω is imposed and the eigenvalue problem is solved to obtain the wave vectors. It is seen that the implementation is cumbersome as there are two unknowns, k_x, k_y after imposing ω
- Inverse approach : Real wave numbers are imposed and the eigenvalue problem is solved to find frequencies. This approach is said to easier to obtain bandgaps.

The second task is to apply Bloch boundary condition. In theory, the Bloch boundary condition is applied using a transfer matrix applied on the global stiffness and mass matrix.[4]

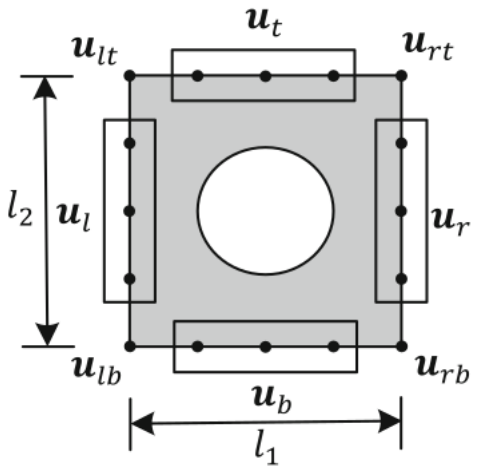

Fig. 2 Discretized unit cell boundary and degrees of freedom

$$\mathbf{u}_0 = \mathbf{R}_k \hat{\mathbf{u}}_0 \tag{6}$$

where,

$$\mathbf{u_0} = \begin{bmatrix} \mathbf{u_i} \\ \mathbf{u_b} \\ \mathbf{u_t} \\ \mathbf{u_l} \\ \mathbf{u_r} \\ \mathbf{u_{lb}} \\ \mathbf{u_{rb}} \\ \mathbf{u_{rt}} \\ \mathbf{u_{lt}} \end{bmatrix} \quad \hat{\mathbf{u}}_\mathbf{0} = \begin{bmatrix} \mathbf{u_i} \\ \mathbf{u_b} \\ \mathbf{u_l} \\ \mathbf{u_{lb}} \end{bmatrix} \tag{7}$$

The constraint matrix for the unit cell in fig. 2 can be written as:

$$\mathbf{R}_k = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & Iexp(2\pi i k_2 l_2) & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & Iexp(2\pi i k_1 l_1) & 0 \\ 0 & 0 & 0 & I \\ 0 & 0 & 0 & Iexp(2\pi i k_1 l_1) \\ 0 & 0 & 0 & Iexp[2\pi i(k_1 l_1 + k_2 l_2)] \\ 0 & 0 & 0 & Iexp(2\pi i k_2 l_2) \end{bmatrix} \tag{8}$$

where I and 0 represent the identity and zero matrices of appropriate sizes. The constrain is multiplied to the global stiffness matrix to apply the Bloch boundary condition.

$$\begin{aligned} \hat{\mathbf{K}} &= \mathbf{R}_k^* \mathbf{K} \mathbf{R}_k \\ \hat{\mathbf{M}} &= \mathbf{R}_k^* \mathbf{M} \mathbf{R}_k \end{aligned} \tag{9}$$

where $\mathbf{R}_\mathbf{k}^*$ is the hermitian of $\mathbf{R}_\mathbf{k}$. The periodic boundary condition option in the MOOSE was explored for solving the problem. The PeriodicBC is only capable of handling plain periodic boundary condition i.e. $u(x + h_i) = u(x)$. The option of using a transformation function is given to transform when there is periodicity between non-parallel edges or faces. Using the transformation function for the bloch periodic case was also explored but the codes in the source file for the periodic boundary conditions has reference to libMesh functions. It was not clear how this can be used.

References

[1] Slaughter, A. E., Permann, C. J., Miller, J. M., Alger, B. K., and Novascone, S. R., "Continuous integration, in-code documentation, and automation for nuclear quality assurance conformance," *Nuclear Technology*, Vol. 207, No. 7, 2021, pp. 923–930.

[2] Chin, E. B., Mokhtari, A. A., Srivastava, A., and Sukumar, N., "Spectral extended finite element method for band structure calculations in phononic crystals," *Journal of Computational Physics*, Vol. 427, 2021, p. 110066.

[3] Cool, V., Deckers, E., Van Belle, L., and Claeys, C., "A guide to numerical dispersion curve calculations: explanation, interpretation and basic Matlab code," *arXiv preprint arXiv:2311.09843*, 2023.

[4] Alberdi, R., Zhang, G., and Khandelwal, K., "An isogeometric approach for analysis of phononic crystals and elastic metamaterials with complex geometries," *Computational Mechanics*, Vol. 62, 2018, pp. 285–307.