Acoustic Wave Equation with Interior Boundary Conditions

Arnold Peter Ruymgaart*

University of Washington
AMATH581 Final Project Github
apruymgaart@gmail.com

December 10, 2021

Abstract

In this project we wish to simulate the acoustics of a Helmholtz Resonator (HR) as may be found in a vented loudspeaker enclosure design. The vent functions as a Helmholtz resonator and has a low frequency attenuation effect. The enclosure is compartmentalized due to the vent design. Therefore, in order to carry out the simulation we must implement reflecting boundary conditions in the interior of the simulation region. In addition, a mixture of absorbing and reflecting boundary conditions are needed at the edges of the simulation region. Absorbing boundary conditions for the wave equation require additional attention and are implemented as described by [Alam & Mohiuddin, 2021].

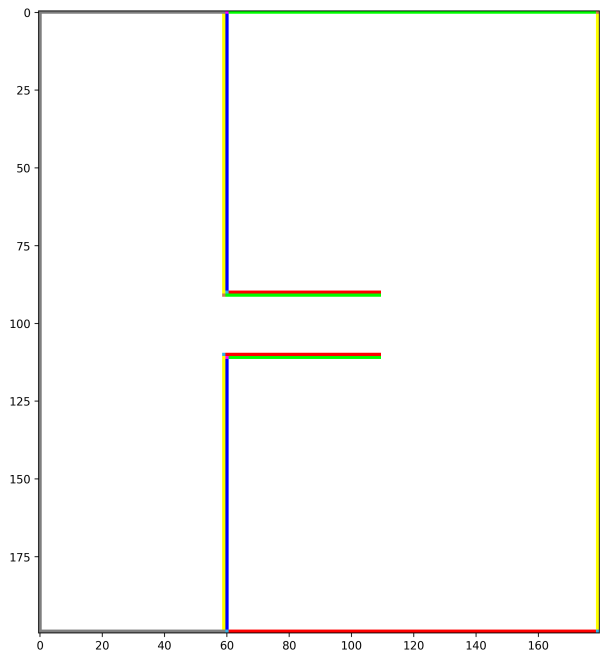
I. Introduction

Ultimately, we would like to simulate an audio speaker in an enclosure with a bass tube mounted into the front panel. this is the same panel in which the audio source speaker is also mounted. The bass tube acts as a Helmholtz Resonator (HR) and amplifies a certain range of low frequencies above a certain cutoff frequency called the resonance frequency. Below the resonance frequency, the sound waves emitted from the tube port cancel those emitted by the speaker.

The loudspeaker enclosure must contain several interior compartment boundaries that form the Helmholtz resonator. A Helmholtz resonator can be modeled by a simple one dimensional equation which is included here for reference is Section II.i. Although this equation can produce an accurate estimate of resonance frequency (eigenvalues), it cannot capture distortion effects that arise from its 3D shape.

In this project we'll carry out 2D simulations for proof of concept validating the boundary method implementation and attempt to simulate Helmholtz resonance in a 2D slice of a rectangular HR design. The system could be extended to 3D relatively easily.

Figure 1: *Simulation Area of Helmholtz Resonator Enclosure with Interior Boundaries*



Boundary conditions are colored: grey=absorbing, yellow=reflect right, blue=reflect left, green=reflect top, red=reflect bottom. Note that interior boundaries have two adjacent layers.

*Contact for further information

II. Theoretical Background

i. Helmholtz Resonator Spring-Mass Model

For reference, we first introduce a simplistic 1D model of a Helmholtz Resonator that is analogous to a spring-mass (harmonic oscillator) system. The air in the main volume of the resonator acts like the "spring" to the mass of air in the tube that connects to the exterior. If we start with a uniform above ambient acoustic pressure in the HR main volume, it will escape through the resonator tube in form of pressure (sound) waves. As the air mass moves out at the velocity of sound, inertia of the mass causes more air to leave the resonator than the initial pressure difference. This causes the pressure in the main volume of the HR to drop below ambient. We now get a sound wave moving into the HR. This is followed by another, lesser amplitude wave out and the process cycles at dampening amplitude. With an appropriate source, the dampening is removed and the system is described by:

$$m\ddot{x} + kx = 0 \tag{1}$$

The mass of the air is proportional to the volume: $m = \rho V$. Air density ρ equals about $1.225 kg/m^3$. The approximate **resonance frequency** ω can be obtained from solving eqn. 1. With second derivative operator D_{tt} we have the eigenfunction equation $D_{tt}x = -\frac{k}{m}x$ with solutions $x(t) = A\cos(\omega t + b)$ and eigenvalue $\omega^2 = k/m$.

$$\omega = \frac{1}{2\pi}\sqrt{\frac{k}{m}} = \frac{1}{2\pi}\sqrt{\frac{\rho c^2 S^2/V}{\rho SL}} \tag{2}$$

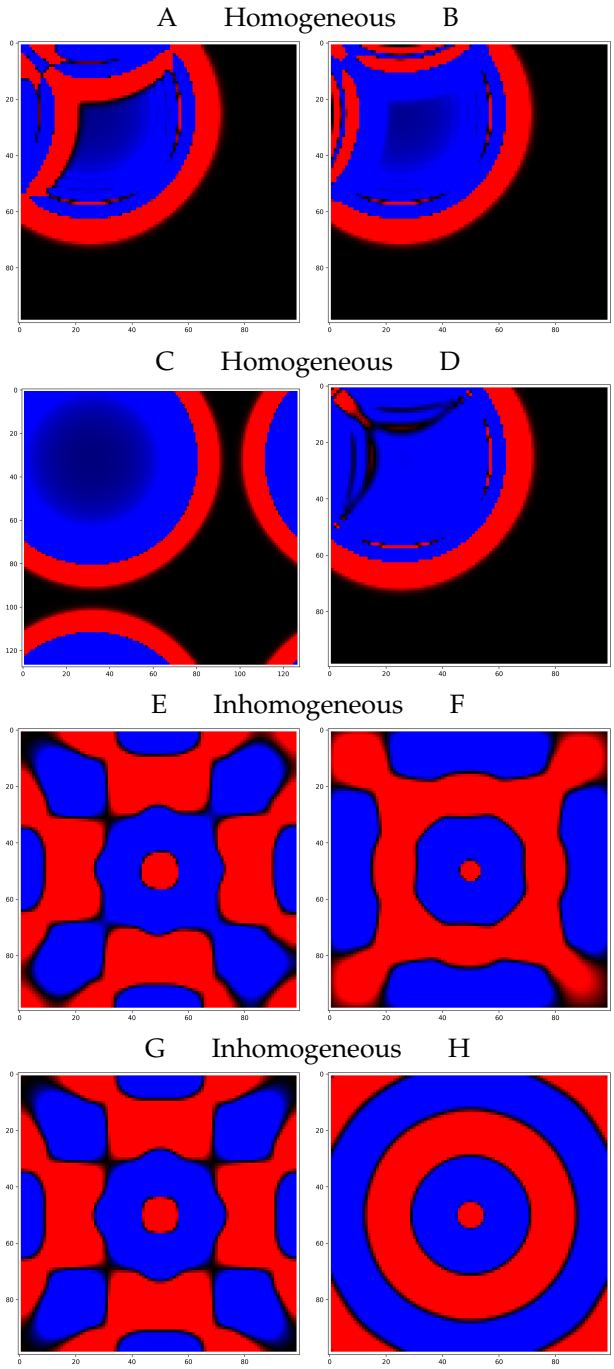
$$= \frac{c}{2\pi}\sqrt{\frac{A}{LV}} = \frac{343m}{2\pi}\sqrt{\frac{A}{LV}} \tag{3}$$

where c is the speed of sound (343 m/s), A is the tube cross-section area, V is the resonator (box) volume and L is the tube length.

We calculate the resonance frequency of the resonator shown in Figure 1. The simulation area width is $0.2m$ and height is $0.22m$. The resonator volume starts at $n = 60$ and has a width of 0.132. We assume the depth is the same as the width at $0.132m$. The resonator volume then is $V = 0.132m \cdot 0.132m \cdot 0.22m = 0.00383m^3$. The tube has length $L = 0.056m$ and radius $r = 0.0105m$ so tube cross area is $A = \pi r^2 = 0.000346m^2$. Then

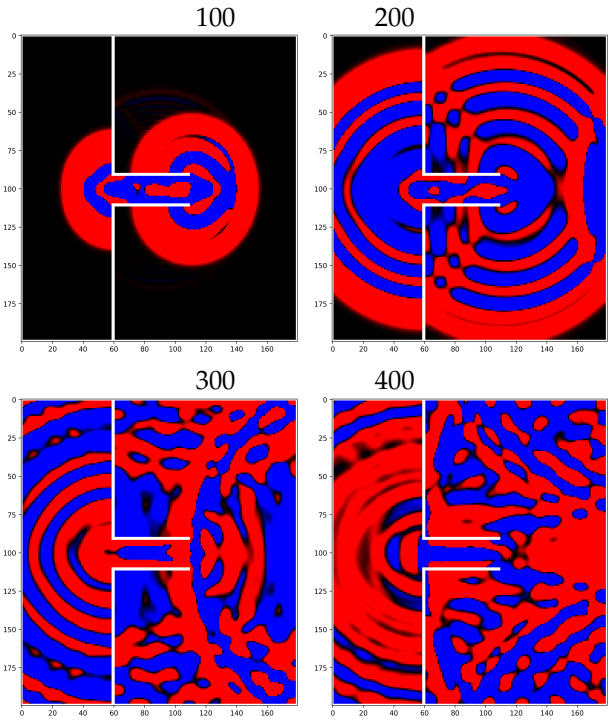
$$\omega = 54.59\sqrt{\frac{0.000346m^2}{0.056m \cdot 0.00383m^3}} \simeq 69.34s^{-1} \tag{4}$$

Figure 2: *Wave Equation With Reflecting (In & Out of Phase), Periodic & Absorbing Boundary Conditions*



In each case, a square simulation area is used and the homogeneous 2D wave equation is solved by finite difference Eqn. 20. The initial condition (IC) in A-D is a Gaussian centered in the top left quarter. In E-H there is zero IC but there is a wave generating Gaussian source at the center of the simulation. A snapshot is taken at step 800 (at the same time for each). Red color $\implies u_{m,n} > 0$ and blue color $\implies u_{m,n} < 0$. **(A & E)** Neumann boundaries. The front of the deflected waves in the left image are of opposite phase. **(B & F)** Dirichlet boundaries cause out of phase deflection. **(C & G)** Periodic boundaries. **(D & H)** Absorbing boundaries.

Figure 3: *Testing Interior Boundaries*



In this case we solve the homogeneous equation with an initial gaussian pressure in the tube. The boundaries are as shown in Figure 1. This tests interior reflective BC and absorbing BC on the left edge and top-left, bottom left of the simulation area. We can see reflections inside the HR and no reflections outside.

ii. Acoustic Wave Equation

We first define **acoustic pressure** $u(t,x,y)$ as local deviation from ambient pressure. Acoustic pressure is a scalar field and may be positive or negative. Negative acoustic pressure means the local pressure at position x,y is below ambient pressure.

The homogeneous 2^{nd} order **wave equation** (2D)

$$u_{tt} = c^2\nabla^2 u \tag{5}$$

$$u_{tt} = c^2(u_{xx} + u_{yy}) \tag{6}$$

This is a hyperbolic differential equation. If we have a sound source $s(t,x,y)$ then the system is described by the inhomogeneous wave equation:

$$u_{tt} = c^2\nabla^2 u + s(t,x,y) \tag{7}$$

ii.1 Reflecting Boundaries

If waves bounce back at the boundaries we have **reflective boundary** conditions. It may be tempting to compare wave behavior to diffusion but waves behave differently. In diffusion, a zero flux $u_x = 0$ Neumann BC will reflect while Dirichlet BC can absorb (and be a sink or source). In case of the wave equation, both Dirichlet and Neumann boundary conditions will cause reflections. In case of Dirichlet BC, the reflection will be out of phase and in case Neumann the reflection will be in-phase. This behavior is experimentally verified and illustrated in Figure 2. For sound reflecting surfaces we wish in-phase deflection so we can apply Neumann BC.

In the HR simulation (see Figure 1), we have reflecting edges on the right side of the simulation volume and in the interior of the resonator as well. In front of the enclosure we need to simulate a small portion of the exterior. The left simulation box edge should therefore be absorbing in order to simulate an open room. This is confirmed in the HR simulations done in this project, visualized in the bottom left panel of Figure 4 and discussed in Section IV.

ii.2 Absorbing Boundaries

In 2D or higher, there is no simple solution for absorbing boundary conditions. And all solutions in 2 or more dimensions are approximate.

In order to get an absorbing boundary, at a left or right boundary point we solve the first order wave equation,

$$u_x = cu_t \;\; or \;\; u_x = -cu_t \tag{8}$$

At a top or bottom absorbing boundary point,

$$u_y = cu_t \;\; or \;\; u_y = -cu_t \tag{9}$$

We could obtain the boundary point with same 2nd order accuracy as the interior with the forward space (and still central time) 1st difference:

$$u_x = cu_t \tag{10}$$

$$\frac{4u^i_{m,n+1} - 3u^i_{m,n} - u^i_{m,n+2}}{2h_x} = c\frac{u^{i+1}_{m,n} - u^{i-1}_{m,n}}{2h_t} \tag{11}$$

$$(4u^i_{m,n+1} - 3u^i_{m,n} - u^i_{m,n+2})\frac{h_t}{ch_x} = u^{i+1}_{m,n} - u^{i-1}_{m,n} \tag{12}$$

$$u^{i+1}_{m,n} = (4u^i_{m,n+1} - 3u^i_{m,n} - u^i_{m,n+2})\frac{h_t}{ch_x} + u^{i-1}_{m,n} \tag{13}$$

Instead, the authors of [Alam & Mohiuddin, 2021] discretize this as follows (left BC shown):

$$u^{i+1}_{m,0} = u^i_{m,1} + \frac{CFL-1}{CFL+1}\left(u^{i+1}_{m,1} - u^i_{m,0}\right) \tag{14}$$

The CFL number $\frac{ch_t}{h_{space}}$ is further described in the stability discussion, Section ii.6.

The discretized BC that we have adopted in this project, Equation 14 looks like an implicit equation because the RHS contains $u_{1,n}^{i+1}$ at time step $i+2$ but this is not the case. Here, the interior points $u_{m,n}^{i+1}$ are all calculated first. Then the absorbing boundary is done as above using the interior point $u_{m,1}^{i+1}$ (noting the boundary point is $u_{m,0}^{i+1}$). Note: this is Python/C indexing starting at zero. Matlab/FORTRAN index starts at 1 (add 1 to n above).

ii.3 Inhomogeneous: Sound Source

To model a sound source (like a speaker) we use a modulated Gaussian $G(x,y)$. The amplitude is modulated by a sinusoid: $S(x,y,t) = G(x,y)\cdot\cos(bt)$. This is the source S used in the inhomogeneous equation:

$$u_{tt}(x,y,t) = c^2(u_{xx}(x,y,t) + u_{yy})(x,y,t) + S(x,y,t) \tag{15}$$

ii.4 IBVP

In case of ODE's (solution y is a function of just t), an IVP becomes a BVP if we have a boundary condition (which can only be at the end t). At the end of the simulation time we must have a specific value of y and/or y_t significantly limiting the possible solution functions y. We have used a linear system (direct) or a shooting approach to find a solution. In case of PDE based dynamical systems (solutions y are equations that have at least one other variable besides t), we can have boundary conditions in the spatial domain without imposing a end time BC. This is the type of problem solved here, an IBVP. Although we have boundary conditions, we don't need a shooting approach because we don't have a "final condition".

ii.5 Finite Difference Solver

If we make a uniform discretization in both time and space, using **superscript** index i for time position and **subscript** indices m,n for space mesh position then the homogeneous 2nd order wave equation can be approximated by the following explicit central differences method:

$$u_{tt} = c^2(u_{xx} + u_{yy}) \tag{16}$$

$$\frac{u_{m,n}^{i+1} - 2u_{m,n}^{i} + u_{m,n}^{i-1}}{h_t^2} = c^2\left(\frac{u_{m-1,n}^{i} - 2u_{m,n}^{i} + u_{m+1,n}^{i}}{h_x^2} + \frac{u_{m,n-1}^{i} - 2u_{m,n}^{i} + u_{m,n+1}^{i}}{h_y^2}\right) \tag{17}$$

$$u_{n,m}^{i+1} - 2u_{n,m}^{i} + u_{n,m}^{i-1} = \frac{c^2h_t^2}{h_x^2}(u_{m-1,n}^{i} - 2u_{m,n}^{i} + u_{m+1,n}^{i}) + \frac{c^2h_t^2}{h_y^2}(u_{m,n-1}^{i} - 2u_{m,n}^{i} + u_{m,n+1}^{i}) \tag{18}$$

$$u_{n,m}^{i+1} = \frac{c^2h_t^2}{h_x^2}(u_{m-1,n}^{i} - 2u_{m,n}^{i} + u_{m+1,n}^{i}) + \frac{c^2h_t^2}{h_y^2}(u_{m,n-1}^{i} - 2u_{m,n}^{i} + u_{m,n+1}^{i}) + 2u_{n,m}^{i} - u_{n,m}^{i-1} \tag{19}$$

For a uniform mesh in space ($h_x = h_y = h_{space}$) with source $s_{m,n}(t)$ we can write

$$u_{n,m}^{i+1} = \left(\frac{ch_t}{h_{space}}\right)^2(u_{m,n-1}^{i} + u_{m-1,n}^{i} - 4u_{m,n}^{i} + u_{m+1,n}^{i} + u_{m,n+1}^{i}) + 2u_{n,m}^{i} - u_{n,m}^{i-1} + h_t^2 s_{m,n}^{i} \tag{20}$$

The last term $s_{m,n}(t)$ in the RHS makes this the inhomogeneous difference equation ($s_{m,n}^{i} = 0 \implies$ homogeneous).

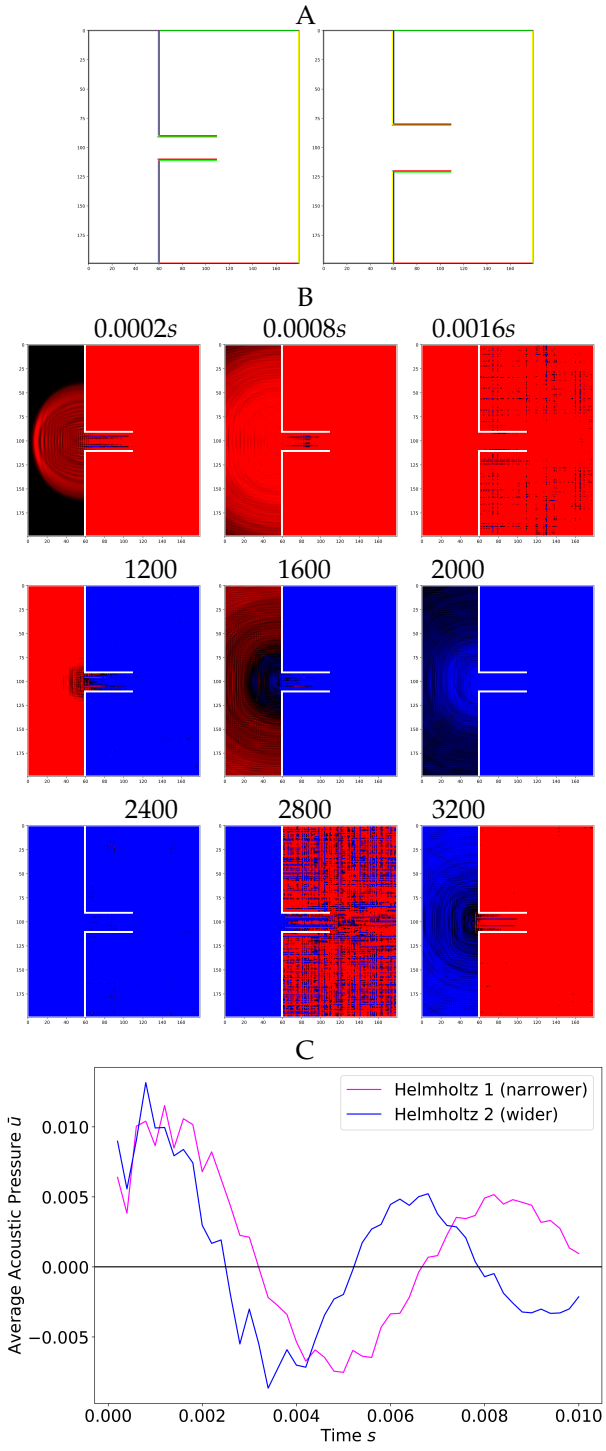
ii.6 Stability

The above-mentioned explicit scheme (Eqn. 20) is conditionally stable. The Courant–Friedrichs–Lewy (CFL) number, with uniform grid $h_s = h_x = h_y$

$$CFL = \frac{ch_t}{h_s} \tag{21}$$

The condition for stability is $CFL \leq \frac{1}{\sqrt{N_{dim}}}$ where N_{dim} is the number of spatial dimensions. For 2D $CFL \leq \frac{1}{\sqrt{2}}$. The Von Neumann stability analysis with complete derivation of the CFL condition is included in appendix A

Figure 4: *Resonator With Uniform Positive Pressure IC*



This experiment tests the Helmholtz Resonance. The homogeneous wave equation is solved with a uniform positive pressure IC inside the HR. There is no source yet waves are formed and the average pressure inside the HR alternates sign. The experiment is carried out twice, one for each BC set shown in panel A noting the set on the right has a wider tube. Panel C plots the average pressure of grid cells just in front of the HR tube exterior over time. The narrower tube is expected to have a lower resonance frequency which is confirmed here with a longer wavelength: 0.0075s vs 0.0055s. See Section IV C.

iii. Spectral Solutions

Depending on boundary conditions, at least the spatial derivatives (the RHS of Eqn. 5) could be carried out in Fourier space. In Fourier space the 2^{nd} derivatives are simply multiplications by $(ik_{m,n})^2 = -k^2_{m,n}$. So on a grid, we could do a Hadamard (element-wise) product with a matrix of these expressions. Starting with Eqn. 5

$$u_{tt} = c^2(u_{xx} + u_{yy}) \tag{22}$$

$$\frac{u^{i+1}_{n,m} - 2u^i_{n,m} + u^{i-1}_{n,m}}{h_t^2} = c^2\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)u^i_{m,n} \tag{23}$$

$$u^{i+1}_{n,m} - 2u^i_{n,m} + u^{i-1}_{n,m} = c^2h_t^2\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)u^i_{m,n} \tag{24}$$

Re-arrange to obtain an update equation for $u_{n,m}$:

$$u^{i+1}_{n,m} = c^2h_t^2\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)u^i_{m,n} + 2u^i_{n,m} - u^{i-1}_{n,m} \tag{25}$$

In this case, unlike Eqn. 17 we did not write numerical approximations of the spatial derivatives. Only the LHS time derivative is discretized. We then take the 2D-FT both sides in x and y dimensions in order to get rid of the spatial derivatives:

$$\tilde{u}^{i+1}_{n,m} = c^2h_t^2\left(-k^2_{x_{m,n}} - k^2_{y_{m,n}}\right)\tilde{u}^i_{m,n} + 2\tilde{u}^i_{n,m} - \tilde{u}^{i-1}_{n,m} \tag{26}$$

We observe the entire scheme exists in Fourier space and the Laplacian has become a product of scalar wavenumber coordinate values $\cdot(-k^2_{x_{m,n}} - k^2_{y_{m,n}})$. Discretized cartesian coordinate positions $(x, y)_{m,n}$ at position index m, n correspond to wavenumber coordinate positions $(k_x, k_y)_{m,n}$ in the Fourier domain and transformed $\tilde{u}(k_x, k_y, t)$ is a function of k_x, k_y rather than x, y. The Laplacian term is calculated with a simple element-wise product of the grid $\tilde{u}_{m,n} \cdot (-k^2_{x_{m,n}} - k^2_{y_{m,n}})$. The entire RHS of equation 26 can be carried out with element-wise grid operations (which may remain in vector or grid form). Because these derivatives can be done in Fourier space *without involving adjacent grid points*, there are no needed stencils (nor stencil alternative matrix multiplication). Note in Appendix C implementation, the function *waveEquationSpectralUNext* does not call the stencil function. The authors of [Ramezani, 2008] provide a 1D version of Eqn. 26.

Although significantly more computationally efficient, *the spectral approach is limited to periodic boundary conditions at simulation edges* and does not allow for interior BC. The FFT method is therefore ruled out for use in the HR simulation. However the 2DFFT option

is included in the implementation and may be used to reproduce the simulation shown in Figure 2 C & G (this was done).

What about using a Chebyshev and/or Legendre polynomial basis? Chebyshev spectral solutions for the wave equation (at least the spatial component) do exist. *Chebyshev, Legendre, Gauss-Hermite and other polynomials extend spectral methods (FFT based) to different boundary conditions*. Non-periodic function f can be extended (repeated) in a periodic way. However, boundaries in the interior of the simulation region cause problems for these semi-spectral methods as well. This is the reason for use of finite difference in the HR simulation of this project. There are Finite Element (FE) alternatives and combination FE-spectral methods beyond the scope of this project that could be used and will be briefly discussed in Section V.

III. Algorithm Implementation & Development

The algorithm implementation consists of

A. Stencil method providing periodic, Dirichlet & Neumann boundary conditions as well as any possible combinations

B. Finite difference wave equation solver 20 & Spectral wave equation solver 26

C. Implementation of wave equation absorbing BC in main scheme

(A). The stencil method implemented here in the function **stencil()** in script $pde\_findiff\_functions.py$ provided in Appendix B. is similar to a convolution approach: a 3×3 grid "window" is iterated over all grid points. The stencil returns a 3×3 mini grid of the points adjacent to the center point. In case of a periodic boundary, the value on the other side of the grid is returned. In case of a Dirichlet boundary, the specified constant value is returned. In case of a Neumann boundary $u^i_{m,a+1} = u^i_{m,a-1}$ so the value on the other side of the center point (in the interior) is returned. The method is similar to the use of "ghost" cells $u^i_{m,a-1}$ but no additional cells are kept in memory because in each case the value can be calculated. Any cell in the grid may be a boundary cell and the boundary status of each grid cell is kept in auxiliary matrix B. This matrix is built in the function **squareSim** or **processBcIni**. Both are implemented in the script *draw_boundaries.py* provided in Appendix B.

The stencil avoids matrix multiplication so it does not suffer from the memory limitations of (dense) matrix multiplication but it is inefficient: $\mathcal{O}(N^2)$. Dense matrix multiplication is worse $\mathcal{O}(N^3)$ but the sparse band-diagonal matrices that arise from derivatives are $\mathcal{O}(N^2)$. The stencil method is therefore similar in time complexity to sparse matrix multiplication. It is possible to construct a mostly sparse matrix and replace the stencil with sparse matrix multiplication. However, the added complexity of the internal boundary conditions would make construction of the Laplacian operator matrix a difficult task.

(B). With the stencil method at hand, implementation of the leapfrog scheme Eqn. 20 is simple. The implementation is the function **waveEquationUnext()** in the script *acoustic_simulation.py* provided in Appendix C. The function loops over all interior points and uses the stencil provided adjacent cell values. The spectral version, Eqn. 26 is implemented in function **waveEquationSpectralUNext**, also in the script *acoustic_simulation.py*. An input flag allows the use of one or the other.

(C). Implementation of the absorbing boundary conditions was not done in the stencil method as it is a wave function specific algorithm while the stencil method is general. Absorbing BC cells are initiated to the value 0 in the stencil and calculated in the main simulation loop, in the script *acoustic_simulation.py*. The boundary cell values are calculated with Eqn.14 as described by [Alam & Mohiuddin, 2021].

IV. Computational Results

A set of experimental systems were solved with Eqn. 20 or Eqn. 26

Each system is invoked by a shell command as listed in *run.sh* provided in Appendix: C.i. The scripts had the following main goals:

A. Test various (reflecting and absorbing) simulation edge boundary conditions

B. Test internal boundary conditions

C. Investigate Helmholtz Resonator

(A) The results of simulation BC testing are shown in Figure 2. In panels $A - D$ we solve the homogeneous equation (no source) by finite difference. In each case, the initial condition is a Gaussian pressure in the center of the top-left quadrant of the simulation area. The images shown are taken at step $i = 800$. We can see in-phase (panel A) and out of phase (panel B) reflecting waves

corresponding to Neumann and Dirichlet boundary conditions resp. In panel C we have periodic boundaries. In panel D we test the absorbing boundaries (**). We note that there is some minimal reflection as the absorbing BC are an approximate solution.

In panels E-H of Figure 2, we have the same boundary conditions as in panel A-D but instead of a nonzero initial condition, we place a time dependent source $\cos(b * t) * Gaussian$ and solve the inhomogeneous equation. We note in this case, the absorbing BC working well (as in [Alam & Mohiuddin, 2021]). The minor reflections as seen in panel D are not sufficient to cause disruption when a source is present.

(B) The results of interior BC testing are shown in Figure 3. The initial condition is a positive pressure inside the tube. The panels are time progression of the simulation. We can see that all the boundaries are working as intended and none are "leaking". We see reflections in the interior of the HR but not outside since the left edge of the simulation area is absorbing.

(C) By the spring-mass analogy, a uniform positive pressure IC in the resonator (and no source) should result in the emission of sound waves at the resonance frequency. As the initial positive pressure results in a positive pressure wave, because inertia, the sign of the average pressure in the HR should flip. The HR test is visualized in Figure 4. Two HR enclosures were made with only one difference: the cross-section (tube diameter) is of a different size while all else is the same. A narrower tube produces a lower resonance frequency by Eqn. 2. The simulation solves the homogeneous equation (no source) with the initial condition a uniform above ambient acoustic pressure throughout the HR volume (and zero outside). This is exactly the scenario described in Section II.i. Essentially the air "spring" is compressed. We see the system resonate (several pressure cycles) as expected. It is worth keeping in mind the entire sequence of + and - pressure waves comes from the single uniform + initial condition. This means the system oscillates without any source. And, as expected, the oscillation is at higher frequency with the wider tube. We take these results to confirm we have witnessed Helmholtz resonance in these simulations. The resonance frequency of the narrower tube was calculated $69.34s^{-1}$ in Eqn. 4. The wider tube has a cross-section area $0.0014m^2$ and frequency of 139.47. The wavelengths obtained from the simulations are about $0.0075s$ and $0.0055s$ with frequencies $133s^{-1}$ and $181s^{-1}$. They are not an exact match but we are simulating a 2D "slice". In reality the system is 3D. We are using a third dimension of assumed depth same as width but the slice simulation would be the same if the assumed depth was different. This means the 2D simulation cannot produce results consistent with 3D and this experiment should be extended to 3D.

V. 581 Method Contribution & Extensions

i. 581 Methods used in this project

Many methods that were covered in the AMATH581 course have contributed to this project. These methods have been discussed throughout the previous sections. Specifically,

- IVP, BVP and IBVP are discussed in Section ii.4
- The Finite Difference(FD) method of solving an IBVP arising from a PDE ("time stepping a PDE") in particular, the leapfrog method was discussed in Section II.ii.5.
- Discretization of boundary conditions is discussed in Sections II.ii.1 and II.ii.2.
- The spectral method of solving an IBVP arising from a PDE is discussed in Section II.iii along with advantages and limitations.
- Von Neumann stability analysis is discussed in Section ii.6 and included in Appendix A
- Computational complexity is addressed in numerous sections

ii. Extensions

The conditionally stable scheme implemented by FD is an explicit scheme. Implicit schemes may further improve the region of stability. Finite element (FE) methods allow solution on more complicated domains and should be able to accommodate the interior boundary conditions encountered in this project. The **spectral element method** combines FE with spectral methods. The authors of [Zhu, 2011] combine the spectral element method with an implicit Newmark time integral method to solve the acoustic wave equation.

VI. Summary and Conclusions

i. Interior BC Challenges

The simulation area is a rectangular grid. The boundaries at the simulation area edges needed to be a mixture of reflecting and absorbing conditions. Absorbing conditions for the 2- or 3D wave equation are approximate. Further complicating the project was the need for reflecting boundaries in the interior of the simulation area needed

for the walls of the Helmholtz resonator. A finite difference scheme was chosen as the simplest solution to overcome these challenges. Extensions are discussed in Section V.ii.

ii. Observed Helmholtz Resonance

Interestingly, acoustic behavior consistent with Helmholtz Resonance was observed in the 2D HR simulation as discussed in Section IV C. Also, a Helmholtz resonator with wider cross-section tube area also was demonstrated to produce a higher resonance frequency consistent with predictions by the simple spring-mass model included in Section II.i. However, the 2D model cannot fully account for the HR behavior and frequencies are not an exact match. The model should be extended to 3D. This could be done with the same methods introduced here but these are not optimal when it comes to computational efficiency ($\mathcal{O}(N^2)$)

iii. Computational Efficiency

The spectral method implemented is significantly faster than the FD scheme but as noted in Section II.iii it cannot be directly used with BC other than periodic and interior BC. Chebyshev polynomials extend the spectral method to other BC but we are left with the challenge of interior BC. Finite element methods should solve both problems (of efficiency and BC). A combination implicit spectral element method is proposed in Section VI.ii

iv. Project

All source code for this project is provided on GitHub:
https://github.com/aruymgaart/AMATH/tree/master/acoustic_simulation

References

[Alam & Mohiuddin, 2021] Alam, Loskor & Mohiuddin (2021). Time Dependent Wave Propagation Modeling Using Finite Difference Scheme of 2D Wave Equation Based on Absorbing and Reflecting Boundaries. *Journal of Applied Mathematics and Physics*, 2021, 9, p. 2334-2344

[Ramezani, 2008] Mehdi Ramezani, Mehdi Dehghan, Mohsen Razzaghi (2008). Combined Finite Difference and Spectral Methods for the Numerical Solution of Hyperbolic Equation with an Integral Condition. *Numerical Methods for Partial Differential Equations*, Volume 24, Issue 1, p. 1-8

[Zhu, 2011] Changyun Zhu, Guoliang Qin, Jiazhong Zhang (2011). Implicit Chebyshev spectral element method for acoustics wave equations. *Finite Elements in Analysis and Design*, Volume 47, 2011, p. 184-194

A. Von Neumann Stability Analysis of the 2D Wave Equation

An ansatz solution to the wave equation is a 2D (arbitrary 2D direction) plane wave described by the function $u(x,y,t) = e^{i(k_x x + k_y y - \omega t)}$. The discrete PDE (as in Eqn 20 but using j for time index to avoid confusion with the imaginary unit):

$$\frac{u^{j+1}_{m,n} - 2u^{j}_{m,n} + u^{j-1}_{m,n}}{h_t^2} = c^2 \left(\frac{u^{j}_{m-1,n} - 2u^{j}_{m,n} + u^{j}_{m+1,n}}{h_x^2} + \frac{u^{j}_{m,n-1} - 2u^{j}_{m,n} + u^{j}_{m,n+1}}{h_y^2} \right) \tag{27}$$

with $x = nh_x$, $y = mh_y$ and time $t = jh_t$

$$u^{j}_{m+1,n} = e^{i(k_x n h_x + k_y (m+1) h_y - \omega j h_t)} = e^{ik_y h_y} e^{i(k_x n h_x + k_y m h_y - \omega j h_t)} = e^{ik_y h_y} u^{j}_{m,n} \tag{28}$$

$$u^{j}_{m-1,n} = e^{i(k_x n h_x + k_y (m-1) h_y - \omega j h_t)} = e^{-ik_y h_y} e^{i(k_x n h_x + k_y m h_y - \omega j h_t)} = e^{-ik_y h_y} u^{j}_{m,n} \tag{29}$$

Similarly:

$$\begin{aligned} u^{j}_{m,n+1} &= e^{ik_x h_x} u^{j}_{m,n} \qquad & u^{j}_{m,n-1} &= e^{-ik_x h_x} u^{j}_{m,n} \\ u^{j+1}_{m,n} &= e^{-i\omega h_t} u^{j}_{m,n} \qquad & u^{j-1}_{m,n} &= e^{i\omega h_t} u^{j}_{m,n} \end{aligned} \tag{30}$$

Substitute these into the PDE (equation 27):

$$\frac{e^{-i\omega h_t} - 2 + e^{i\omega h_t}}{h_t^2} u^{j}_{m,n} = c^2 \left(\frac{e^{-ik_y h_y} - 2 + e^{ik_y h_y}}{h_x^2} + \frac{e^{-ik_x h_x} - 2 + e^{ik_x h_x}}{h_y^2} \right) u^{j}_{m,n} \tag{31}$$

Assuming $h_x = h_y = h_s$

$$e^{-i\omega h_t} - 2 + e^{i\omega h_t} = c^2 \frac{h_t^2}{h_s^2} (e^{-ik_y h_s} + e^{ik_y h_s} + e^{-ik_x h_s} + e^{ik_x h_s} - 4)$$

Using the identity $e^{iz} + e^{-iz} = 2\cos z$

$$2\cos(\omega h_t) - 2 = c^2 \frac{h_t^2}{h_s^2} (2\cos(k_y h_s) + 2\cos(k_x h_s) - 4)$$

$$\cos(\omega h_t) - 1 = c^2 \frac{h_t^2}{h_s^2} (\cos(k_y h_s) + \cos(k_x h_s) - 2)$$

Now using the identity $2\sin^2(x/2) = 1 - \cos(x) \implies -2\sin^2(x/2) = \cos(x) - 1$

$$-2\sin^2\left(\omega \frac{h_t}{2}\right) = c^2 \frac{h_t^2}{h_s^2} (\cos(k_y h_s) + \cos(k_x h_s) - 2)$$

$$-2\sin^2\left(\omega \frac{h_t}{2}\right) = c^2 \frac{h_t^2}{h_s^2} \left(-2\sin^2\left(k_y \frac{h_s}{2}\right) - 2\sin^2\left(k_x \frac{h_s}{2}\right) \right)$$

$$\sin^2\left(\omega \frac{h_t}{2}\right) = c^2 \frac{h_t^2}{h_s^2} \left(\sin^2\left(k_y \frac{h_s}{2}\right) + \sin^2\left(k_x \frac{h_s}{2}\right) \right)$$

$$\sin\left(\omega \frac{h_t}{2}\right) = c \frac{h_t}{h_s} \sqrt{\sin^2\left(k_y \frac{h_s}{2}\right) + \sin^2\left(k_x \frac{h_s}{2}\right)}$$

If wavenumber $k_y = k_x = k$ then

$$\sin\left(\omega \frac{h_t}{2}\right) = c \frac{h_t}{h_s} \sqrt{2\sin^2\left(k \frac{h_s}{2}\right)}$$

Note: for the 1D wave equation we get: $\sin(\omega\frac{h_t}{2}) = c\frac{h_t}{h_s}\sqrt{\sin^2\left(k\frac{h_x}{2}\right)}$ (the difference is the 2 in the square root)

$$\sin\left(\omega\frac{h_t}{2}\right) = c\frac{h_t}{h_s}\sin\left(k\frac{h_s}{2}\right)\sqrt{2}$$

$$\frac{1}{\sqrt{2}}\cdot\frac{\sin\left(\omega\frac{h_t}{2}\right)}{\sin\left(k\frac{h_s}{2}\right)} = c\frac{h_t}{h_s}$$

In order to obtain real solutions, it must be that

$$\frac{1}{\sqrt{2}} \geq c\frac{h_t}{h_s} \tag{32}$$

$$CFL \leq \frac{1}{\sqrt{2}} \tag{33}$$

Note: this derivation differs a bit from the analysis method discussed in the 581 course it should be synonymous. The method discussed in the course starts with assumption of separation of variables of the solution function $u(x,y,t) = g(t)f(x,y)$ and decompose $f = \sum_j c_j e^{ik_j x}$ with a Fourier series. We note that error ϵ propagates with the same scheme as u. Since the difference equation is linear, each term of the series behaves the same and the same as the series. The latter implies we only need to consider growth of one term in the series and $\epsilon_j(x,t) = g(t)e^{ik_j x}$. For the plane wave Ansatz, separation of variables applies:

$$u(x,y,t) = e^{i(k_x x + k_y y - \omega t)} = e^{-i\omega t}e^{i(k_x x + k_y y)}$$

with $g(t) = e^{-i\omega t}$

B. Functions used

The functions defined in the following scripts are utilized in the main simulation code. The **stencil** function returns a 9-element stencil in a 2D simulation grid and applies the boundary conditions. This stencil can be used by a solver of second order accuracy.

Listing 1: *PDE Solver Functions*

```
# AP Ruymgaart
# Finite difference PDE solver
import copy, sys, numpy as np

PERIODIC                                                    =   1               #
REFLECT_TOP, REFLECT_BOTTOM, REFLECT_LEFT, REFLECT_RIGHT    =   2,  4,   8,  16 # green, red, blue, yellow
DIRICHLET_TOP, DIRICHLET_BOTTOM, DIRICHLET_LEFT, DIRICHLET_RIGHT = 32, 64, 128, 256 #
ABSORBING                                                   = 512               # black
bcNames = {  1:'PERIODIC',          2: 'REFLECT_TOP',     4:'REFLECT_BOTTOM',
             8:'REFLECT_LEFT',     16:'REFLECT_RIGHT',   32:'DIRICHLET_TOP',
            64:'DIRICHLET_BOTTOM',128:'DIRICHLET_LEFT',256:'DIRICHLET_RIGHT',
           512:'ABSORBING'}
'''
STENCIL FUNCTION
returns 3x3 grid while applying any combination of boundary conditions as specified in boundary matrix B
  u_pm, u_pc, u_pp
  u_cm, u_cc, u_cp
  u_mm, u_mc, u_pp
'''
def stencil(j,i,G,B,d=0, verbose=False):
  Ly, Lx = B.shape[0], B.shape[1]
  bc = B[j,i]
  ym, yc, yp, xm, xc, xp = j-1, j, j+1, i-1, i, i+1
  if verbose and bc !=0:
        szName = "\n"
        for bit in [1,2,4,8,16,32,64,128,256,512]:
            if bc & bit: szName += "%s " % ( bcNames[bit] )
        print(szName, ym, yc, yp, xm, xc, xp)

  #-- ABSORBING --
  if bc == ABSORBING: return 0,0,0,0,0,0,0,0,0 # defer (put zeros for now)

  #-- INTERIOR (NO BC) --
  if bc == 0:
    u_pm, u_pc, u_pp = G[yp,xm], G[yp,xc], G[yp,xp]
    u_cm, u_cc, u_cp = G[yc,xm], G[yc,xc], G[yc,xp]
    u_mm, u_mc, u_pp = G[ym,xm], G[ym,xc], G[ym,xp]
    return u_pm, u_pc, u_pp, u_cm, u_cc, u_cp, u_mm, u_mc, u_pp

  #-- PERIODIC (at rectangular box edges only) --
  if bc == PERIODIC:
    if ym < 0    : ym += Ly
    if yp >= Ly : yp -= Ly
    if xm < 0    : xm += Lx
    if xp >= Lx : xp -= Lx
    u_pm, u_pc, u_pp = G[yp,xm], G[yp,xc], G[yp,xp]
    u_cm, u_cc, u_cp = G[yc,xm], G[yc,xc], G[yc,xp]
    u_mm, u_mc, u_pp = G[ym,xm], G[ym,xc], G[ym,xp]
    return u_pm, u_pc, u_pp, u_cm, u_cc, u_cp, u_mm, u_mc, u_pp

  bNeumann = bc & (REFLECT_TOP + REFLECT_BOTTOM + REFLECT_LEFT + REFLECT_RIGHT) !=0

  #-- DIRICHLET (as written, at edges only) --
  u_pm, u_pc, u_pp, u_cm, u_cc, u_cp, u_mm, u_mc, u_pp = d,d,d,d,d,d,d,d,d
  if not bNeumann:
    if bc & DIRICHLET_BOTTOM:
      if bc & DIRICHLET_LEFT:
        u_cm, u_cc, u_cp = d, G[yc,xc], G[yc,xp]
        u_mm, u_mc, u_pp = d, G[ym,xc], G[ym,xp]
      elif bc & DIRICHLET_RIGHT:
        u_cm, u_cc, u_cp = G[yc,xm], G[yc,xc], d
        u_mm, u_mc, u_pp = G[ym,xm], G[ym,xc], d
      else:
        u_cm, u_cc, u_cp = G[yc,xm], G[yc,xc], G[yc,xp]
        u_mm, u_mc, u_pp = G[ym,xm], G[ym,xc], G[ym,xp]
    elif bc & DIRICHLET_TOP:
      if bc & DIRICHLET_LEFT:
        u_pm, u_pc, u_pp = d, G[yp,xc], G[yp,xp]
```

```
        u_cm, u_cc, u_cp = d, G[yc,xc], G[yc,xp]
      elif bc & DIRICHLET_RIGHT:
        u_pm, u_pc, u_pp = G[yp,xm], G[yp,xc], d
        u_cm, u_cc, u_cp = G[yc,xm], G[yc,xc], d
      else:
        u_pm, u_pc, u_pp = G[yp,xm], G[yp,xc], G[yp,xp]
        u_cm, u_cc, u_cp = G[yc,xm], G[yc,xc], G[yc,xp]
    else:
      if bc & DIRICHLET_LEFT:
        u_pm, u_pc, u_pp = d, G[yp,xc], G[yp,xp]
        u_cm, u_cc, u_cp = d, G[yc,xc], G[yc,xp]
        u_mm, u_mc, u_pp = d, G[ym,xc], G[ym,xp]
      elif bc & DIRICHLET_RIGHT:
        u_pm, u_pc, u_pp = G[yp,xm], G[yp,xc], d
        u_cm, u_cc, u_cp = G[yc,xm], G[yc,xc], d
        u_mm, u_mc, u_pp = G[ym,xm], G[ym,xc], d
    return u_pm, u_pc, u_pp, u_cm, u_cc, u_cp, u_mm, u_mc, u_pp

  #-- NEUMANN --
  if bc & REFLECT_TOP:
    ym, yc, yp = j+1, j, j+1
  if bc & REFLECT_BOTTOM:
    ym, yc, yp = j-1, j, j-1
  if bc & REFLECT_LEFT:
    xm, xc, xp = i+1, i, i+1
  if bc & REFLECT_RIGHT:
    xm, xc, xp = i-1, i, i-1
  try:
    u_pm, u_pc, u_pp = G[yp,xm], G[yp,xc], G[yp,xp]
    u_cm, u_cc, u_cp = G[yc,xm], G[yc,xc], G[yc,xp]
    u_mm, u_mc, u_pp = G[ym,xm], G[ym,xc], G[ym,xp]
  except:
    szName = ""
    for bit in [1,2,4,8,16,32,128]:
        if bc & bit: szName += "%s " % ( bcNames[bit] )
    print('BC ERROR', szName, 'at', j,i, 'rel pos y',  ym, yc, yp, 'rel pos x', xm, xc, xp)
    exit()
  return u_pm, u_pc, u_pp, u_cm, u_cc, u_cp, u_mm, u_mc, u_pp
```

The following draw_boundaries.py script provides 3 main functions: **boundary2rgb**, **squareSim** & **processBcIni**. The latter two create boundary status matrix B.

Listing 2: *Boundary Functions*

```
import copy, sys, numpy as np, matplotlib.pyplot as plt
from pde_findiff_functions import *


def boundary2rgb(B, bInvertBG=True):
  RGB = np.zeros( (B.shape[0], B.shape[1], 3) )
  Rd, Gr, Bl = np.zeros(B.shape), np.zeros(B.shape), np.zeros(B.shape)

  Gr[B == REFLECT_TOP]     = 1
  Rd[B == REFLECT_BOTTOM] = 1
  Bl[B == REFLECT_LEFT]    = 1
  Gr[B == REFLECT_RIGHT], Rd[B == REFLECT_RIGHT] = 1, 1 # yellow

  TR, BR = REFLECT_TOP + REFLECT_RIGHT, REFLECT_BOTTOM + REFLECT_RIGHT
  TL, BL = REFLECT_TOP + REFLECT_LEFT, REFLECT_BOTTOM + REFLECT_LEFT

  Rd[B == TR], Gr[B == TR], Bl[B == TR] = 0.8, 0.5, 0.3
  Rd[B == BR], Gr[B == BR], Bl[B == BR] = 0.2, 0.7, 0.9

  Rd[B == TL], Gr[B == TL], Bl[B == TL] = 1.0, 0.0, 1.0 # magenta
  Rd[B == BL], Gr[B == BL], Bl[B == BL] = 0.2, 0.7, 0.9

  Rd[B == ABSORBING], Gr[B == ABSORBING], Bl[B == ABSORBING] = 0.5, 0.5, 0.5

  RGB[:,:,0], RGB[:,:,1], RGB[:,:,2] = Rd, Gr, Bl

  print(np.max(RGB), np.min(RGB))
  if bInvertBG: RGB[np.where((RGB==[0,0,0]).all(axis=2))] = [255,255,255]
  return RGB


def squareSim(n=100, BC='PERIODIC'):
  B = np.zeros( (n,n) ).astype(int)
```

```
  first, last = 0, n-1

  if BC == 'DIRICHLET': #-- DIRICHLET BOUNDARIES --
    B[:, first], B[ : ,last]  = DIRICHLET_LEFT, DIRICHLET_RIGHT
    B[first, :] += DIRICHLET_TOP
    B[last,  :] += DIRICHLET_BOTTOM
  elif BC == 'NEUMANN': #-- NEUMANN BOUNDARIES --
    B[ : , first], B[ : , last]  = REFLECT_LEFT, REFLECT_RIGHT
    B[first,  :] += REFLECT_TOP
    B[last,   :] += REFLECT_BOTTOM
  elif BC == 'PERIODIC': #-- PERIODIC BOUNDARIES --
    B[: ,first], B[:, last], B[first, :], B[last, :] = PERIODIC, PERIODIC, PERIODIC, PERIODIC
  elif BC == 'ABSORBING':
    B[:, first], B[:, last], B[first, :], B[last, :] = ABSORBING, ABSORBING, ABSORBING, ABSORBING

  print('squareSim', n, BC)
  return B


def processBcIni(lines):
  sz = lines[0].split(',')
  B = np.zeros( (int(sz[0]), int(sz[1])) ).astype(int)
  for line in lines[1:len(lines)]:
     if len(line):
         if line[0] != '#': exec(line)
  return B


def grid2rgb(U):
  B = (U == 9999.99)
  #print(len(B))
  RGB = np.zeros( (U.shape[0], U.shape[1], 3) )
  Rd,Gr,Bl = copy.copy(U), np.zeros(U.shape), copy.copy(U)
  Rd[U < 0] = 0
  Bl[U > 0] = 0
  Gr[B] = 1
  Rd[B] = 0
  Bl[B] = 0

  RGB[:,:,0] = Rd
  RGB[:,:,1] = Gr
  RGB[:,:,2] = np.abs(Bl)

  mx = np.average(RGB)
  return RGB/mx


if __name__ == '__main__':
  cmds = sys.argv[1:len(sys.argv)]
  if len(cmds): fname = cmds[0]
  else:         fname = 'BC.one.ini'
  f = open(fname, 'r')
  lines = f.readlines()
  f.close()
  B = processBcIni(lines)
  RGB = boundary2rgb(B)

  fig, axes = plt.subplots(figsize=(10, 10))
  plt.imshow(RGB)
  plt.savefig('IO/'+fname.replace('.','_')+'.png', dpi=300, bbox_inches='tight')
  plt.clf()
```

C. Main Codes

Listing 3: *Acoustic Wave Equation*

```
# AP Ruymgaart
# Finite difference 2D wave equation
import copy, sys, numpy as np, matplotlib.pyplot as plt
from draw_boundaries import *
from tensorFiles import *
from pde_findiff_functions import *
fft2, fftshift, ifft2 = np.fft.fft2, np.fft.fftshift, np.fft.ifft2

def waveEquationUnext(U,Ulast,S,B,a1,a2):
  Un = np.zeros(U.shape)
  for j in range(U.shape[0]):
    for i in range(U.shape[1]):
      u_pm, u_pc, u_pp, u_cm, u_cc, u_cp, u_mm, u_mc, u_pp = stencil(j,i,U,B)
      ux = u_cm - 2*u_cc + u_cp
      uy = u_pc - 2*u_cc + u_mc
      Un[j,i] = a1*ux + a2*uy + 2*u_cc - Ulast[j,i] + S[j,i]
  return Un

def waveEquationSpectralUNext(Uft, Ulast, dt, c, Lft):
  fact = (c*dt)**2
  UnFt = fact*np.multiply(Uft,Lft) + 2*Uft - Ulast
  return UnFt

def source(t):
  S = np.zeros( (200,180) )
  v = np.cos(t)
  S[121:175,19] = v
  S[121:175,20] = -v
  return S

def guassian(r, x, y, px, py):           return np.exp(-r*(x-px)**2 - r*(y-py)**2  )
def gaussianSource(t, r, x, y, px, py): return np.cos(t)*guassian(r, x, y, px, py)

if __name__ == '__main__':
  IC, source, name = 'none', 'none', 'none'
  simsel, nrSteps, c, dt, W, mod = 1, 1000, 343.0, 0.000001, 0.2, 100
  plot, bSpectral = False, False

  cmds = sys.argv[1:len(sys.argv)]
  if len(cmds) > 0 :
    stype = cmds[0].split(':')
    simsel = 2
    if stype[0] == 'SQUARE':
      n = int(stype[1])
      B   = squareSim(n, stype[2]) # Boundary matrix
      name = 'square_%s' % (stype[2])
    elif stype[0] == 'FILE':
      f = open(stype[1], 'r')
      lines = f.readlines()
      f.close()
      B =  processBcIni(lines)
      name = stype[1].replace('.ini','').replace('BC.','')

  if len(cmds) > 1 : nrSteps = int(cmds[1])
  if len(cmds) > 2 : dt = float(cmds[2])
  if len(cmds) > 3 : plot = cmds[3].lower()[0] == 't'
  if len(cmds) > 4 : mod = int(cmds[4])
  if len(cmds) > 5 : IC = cmds[5].split(':')
  if len(cmds) > 6 : source = cmds[6].split(':')
  if len(cmds) > 7 :
    if cmds[7] == 'SPECTRAL' :
      bSpectral = True
      name += '_SPECTRAL'
  if len(cmds) > 8 : W = float(cmds[8])

  ds  = W/B.shape[1]   # delta_space (x and y)
  H   = ds*B.shape[0]
  CFL =(c*dt)/ds
  a1, a2 = CFL**2, CFL**2
  stable = abs(CFL) < 1/np.sqrt(2)
  bcFact = (CFL - 1)/(CFL + 1)
  print('Time step', dt, 'width=', W,'height=', H, 'nr steps', nrSteps, 'Sim=', simsel, 'IC', IC, 'Source', source)
  print('CFL=', CFL, 'a1', a1, 'STABLE', stable, 'Space ds=', ds)
```

```
74  if not stable: exit()
75  if bSpectral:
76    if not stype[0] == 'SQUARE':
77      print('not supported')
78      exit()
79
80  vU = np.zeros( (nrSteps, B.shape[0], B.shape[1]) )
81  if bSpectral: vU = vU.astype(complex)
82
83  x2 = np.linspace(0, W, B.shape[1]+1)
84  y2 = np.linspace(0, H, B.shape[0]+1)
85  x, y = x2[0:B.shape[1]], y2[0:B.shape[0]]
86  [X,Y] = np.meshgrid(x,y)
87  n = B.shape[0]
88  k = (2*np.pi/(H)) * np.append(np.arange(0,n/2),np.arange(-n/2,0))
89  [Kx,Ky] = np.meshgrid(k,k)
90  Kx2, Ky2 = np.multiply(Kx,Kx), np.multiply(Ky,Ky)
91  Lft = -1.0*(Kx2 + Ky2) #-- Spectral Laplacian --
92
93  if IC[0] == 'GAUSS':
94    G = guassian(40000, X, Y, W*float(IC[1]), H*float(IC[2]))
95    if bSpectral:
96      vU[0,:,:], vU[1,:,:] = fft2(G), fft2(G)
97    else:
98      vU[0,:,:], vU[1,:,:] = G, G
99  elif IC[0] == 'RECT':
100   ix1, ix2, iy1, iy2 = int(IC[1]), int(IC[2]), int(IC[3]), int(IC[4])
101   R = np.zeros(B.shape)
102   R[iy1:iy2,ix1:ix2] = float(IC[5])
103   R[B != 0] = 0.0
104   vU[0,:,:], vU[1,:,:] = R, R
105
106 if source[0] == 'GAUSS':
107   b = float(source[3])*np.pi
108   GS = guassian(40000, X, Y, W*float(source[1]), H*float(source[2]))
109
110 for k in range(2,nrSteps):
111   if source[0] == 'GAUSS':
112     S = GS*np.cos(k*b) # gaussianSource(k*b, 40000, X, Y, W*float(source[1]), H*float(source[2]))
113   elif simsel == 7:
114     b = 0.09*np.pi  # 0.0009 seems to show resonance at dt=0.0006
115     S = source(k*b)
116   else:
117     S = np.zeros(B.shape)
118
119   Ulast = vU[k-2]                                  # time i-1
120   U = vU[k-1]                                      # time i
121   if bSpectral:
122     Un = waveEquationSpectralUNext(U, Ulast, dt, c, Lft)
123   else:
124     Un = waveEquationUnext(U, Ulast, S, B, a1, a2) # this is time i+1
125
126   # special case absorbing boundary available at edges only
127   # dealt with here rather than in stencil
128   jEnd = B.shape[0] - 1
129   iEnd = B.shape[1] - 1
130   for j in range(U.shape[0]):
131     if B[j,0] == ABSORBING:
132       Un[j,0] = U[j,1]  + bcFact*(Un[j,1] - U[j,0] )
133     if B[j,iEnd] == ABSORBING:
134       Un[j,iEnd] = U[j,iEnd-1]  + bcFact*(Un[j,iEnd-1] - U[j,iEnd] )
135
136   for i in range(U.shape[1]):
137     if B[0,i] == ABSORBING:
138       Un[0,i] = U[1,i] + bcFact*(Un[1,i] -  U[0,i])
139     if B[jEnd,i] == ABSORBING:
140       Un[jEnd,i] = U[jEnd-1,i] + bcFact*(Un[jEnd-1,i] -  U[jEnd,i])
141
142   vU[k] = Un
143
144   if k % mod == 0.0:
145     print("step", k)
146     if plot:
147       if bSpectral:
148         img = np.real(ifft2(vU[k]))
149         img = np.average(img) - img  # ?
150       else:
151         img = vU[k]
```

```
        im = grid2rgb(img)
        im[B != 0] = 1
        fig, axes = plt.subplots(figsize=(8, 8))
        plt.imshow(im)
        if False:
          plt.show()
        else:
          pltname = 'IO/sim_%s_%d_%s_%s.png' % (name, k, cmds[5].replace(':','-').replace('0.',''),
                                                cmds[6].replace(':','-').replace('0.','') )
          plt.savefig(pltname, dpi=300, bbox_inches='tight')
          plt.clf()

  if bSpectral:
    realU = np.zeros( (nrSteps, B.shape[0], B.shape[1]) )
    for k in range(nrSteps):
      img = np.real(ifft2(vU[k]))
      realU[k] = np.average(img) - img
    vU = realU

  for k in range(2,nrSteps): vU[k][B != 0] = 9999.99
  numpy2tnsrFile(vU, 'sim_%s_%d_%s_%s.npz' % (name, nrSteps, cmds[5].replace(':','-').replace('0.',''),
                                              cmds[6].replace(':','-').replace('0.','')) )
```

i. Input scripts

Listing 4: *Boundary Input Script - Helmholtz R1*

```
200,180
#-- ABSORBING BOUNDARIES --
B[  0,0:61], B[199,0:61], B[ : ,   0] = ABSORBING, ABSORBING, ABSORBING
#-- REFLECTING - TOP --
B[  0, 61:180] = REFLECT_TOP
B[ 91, 59:110] = REFLECT_TOP # tube
B[111, 60:110] = REFLECT_TOP # tube
#-- REFLECTING - BOTTOM --
B[199, 60:180] = REFLECT_BOTTOM
B[ 90, 60:110] += REFLECT_BOTTOM # tube
B[110, 59:110] = REFLECT_BOTTOM # tube
#-- REFLECTING - RIGHT --
B[   :   ,179] += REFLECT_RIGHT # (includes right corners)
B[  1: 92, 59] += REFLECT_RIGHT
B[110:199, 59] += REFLECT_RIGHT
#-- REFLECTING - LEFT --
B[  0: 91,  60] += REFLECT_LEFT
B[111:200,  60] += REFLECT_LEFT
#-- REFLECTING CORNERS --
B[  0, 60] = REFLECT_LEFT  + REFLECT_TOP
B[199, 60] = REFLECT_LEFT + REFLECT_BOTTOM
```

Listing 5: *Boundary Input Script - Helmholtz R2*

```
200,180
#-- ABSORBING BOUNDARIES --
B[  0,0:61], B[199,0:61], B[ : ,   0] = ABSORBING, ABSORBING, ABSORBING
#-- REFLECTING - TOP --
B[  0, 61:180] = REFLECT_TOP
B[ 81, 59:110] = REFLECT_TOP # tube
B[121, 60:110] = REFLECT_TOP # tube
#-- REFLECTING - BOTTOM --
B[199, 60:180] = REFLECT_BOTTOM
B[ 80, 60:110] += REFLECT_BOTTOM # tube
B[120, 59:110] = REFLECT_BOTTOM # tube
#-- REFLECTING - RIGHT --
B[   :   ,179] += REFLECT_RIGHT # (includes right corners)
B[  1: 82, 59] += REFLECT_RIGHT
B[120:199, 59] += REFLECT_RIGHT
#-- REFLECTING - LEFT --
B[  0: 81,  60] += REFLECT_LEFT
B[121:200,  60] += REFLECT_LEFT
#-- REFLECTING CORNERS --
B[  0, 60] = REFLECT_LEFT  + REFLECT_TOP
B[199, 60] = REFLECT_LEFT + REFLECT_BOTTOM
```

Listing 6: *Shell Run Script*

```
#—— Homogeneous ——                                  nr steps        plot mod     IC              Sorce      spectral y/n
python3 acoustic_simulation.py SQUARE:100:NEUMANN     1001 0.0000003077 T 200 GAUSS:0.25:0.25 NONE:0:0:0
python3 acoustic_simulation.py SQUARE:100:DIRICHLET   1001 0.0000003077 T 200 GAUSS:0.25:0.25 NONE:0:0:0
python3 acoustic_simulation.py SQUARE:100:ABSORBING   1001 0.0000003077 T 200 GAUSS:0.25:0.25 NONE:0:0:0
python3 acoustic_simulation.py SQUARE:100:PERIODIC    1001 0.0000003077 T 200 GAUSS:0.25:0.25 NONE:0:0:0
python3 acoustic_simulation.py SQUARE:128:PERIODIC    1001 0.000001     T 200 GAUSS:0.25:0.25 NONE:0:0:0   SPECTRAL
python3 acoustic_simulation.py SQUARE:128:PERIODIC    1001 0.000001     T 200 GAUSS:0.25:0.25 NONE:0:0:0   FINDIFF
#—— Nonhomogeneous: Gaussian source ——
python3 acoustic_simulation.py SQUARE:100:DIRICHLET   2001 0.0000003077 T 200 NONE:0.0:0.0    GAUSS:0.5:0.5:0.003
python3 acoustic_simulation.py SQUARE:100:NEUMANN     2401 0.0000003077 T 200 NONE:0.0:0.0    GAUSS:0.5:0.5:0.003
python3 acoustic_simulation.py SQUARE:100:PERIODIC    2001 0.0000003077 T 200 NONE:0.0:0.0    GAUSS:0.5:0.5:0.003
python3 acoustic_simulation.py SQUARE:100:ABSORBING   2401 0.0000003077 T 200 NONE:0.0:0.0    GAUSS:0.5:0.5:0.003

python3 acoustic_simulation.py FILE:BC.helmholtz_1.ini 5001 0.000002     T 100 RECT:60:179:1:199:0.05  NONE:0:0:0
python3 acoustic_simulation.py FILE:BC.helmholtz_2.ini 5001 0.000002     T 100 RECT:60:179:1:199:0.05  NONE:0:0:0
python3 acoustic_simulation.py FILE:BC.helmholtz_1.ini 1001 0.0006       T 100 GAUSS:0.7:0.5     NONE:0:0:0
python3 acoustic_simulation.py FILE:BC.helmholtz_1.ini 3001 0.0006       T 100 NONE:0.0:0.0      GAUSS:0.1:0.5:0.001

python3 acoustic_simulation.py FILE:BC.helmholtz_1.ini 1001 0.0006       T 100 GAUSS:0.1:0.1     NONE:0:0:0
python3 acoustic_simulation.py FILE:BC.helmholtz_1.ini 1001 0.0006       T 100 GAUSS:0.5:0.7     NONE:0:0:0
python3 acoustic_simulation.py FILE:BC.helmholtz_1.ini 1001 0.0006       T 100 NONE:0.0:0.0      GAUSS:0.1:0.1:0.003
```