ME543: Sound and Vibration Project Report: Computational Modelling of Acoustic Logic Gates

Ahmet Burak Yıldırım[1]

[1] *Department of Mechanical Engineering, Bilkent University, Ankara 06800, Turkey*

(Dated: March 23, 2023)

This report aims to demonstrate a computational model for the passive logic gates, in particular, OR and XOR gates, which were described in [1]. As the main difference, this project aims to model the logic gates in MATLAB, whereas [1] realizes the passive logic gates both in experiments and COMSOL simulations. The computational model relies on discretizing the wave equation in 2D and embeds the associated rigid boundary conditions throughout the gate geometry. The simulations show that the computational model for the OR gate shows the expected behavior; however model for the XOR gate is incapable of demonstrating the XOR gate characteristics.

INTRODUCTION TO ACOUSTIC LOGIC GATES

Development of logic gates may be considered as one of the most fundamental advancements in modern electronics. Due to the importance of logic gates, novel ways of designing and controlling logic gates have been a hot topic for research for a long time. Recent developments in optics showed that optical logic gates, which are to be used in photonic circuits, can be achieved both theoretically and experimentally, simply relying on the constructive and destructive interference of light [1]. These developments in optics then drew some attention in the field of acoustics and yielded research in acoustic logic gates, which are now being realized thanks to the self-collimation effect [2], curved metamaterial surfaces [3], or passive structures consisting of Helmholtz resonator arrays that were proposed to be modeled in this project [1]. The acoustic logic gates are furthermore mentioned to have potential applications in computing, information processing, and integrated acoustics [1, 3]. With this, one can see the reasoning behind the efforts in the computational modeling of logic gates and its further implications.

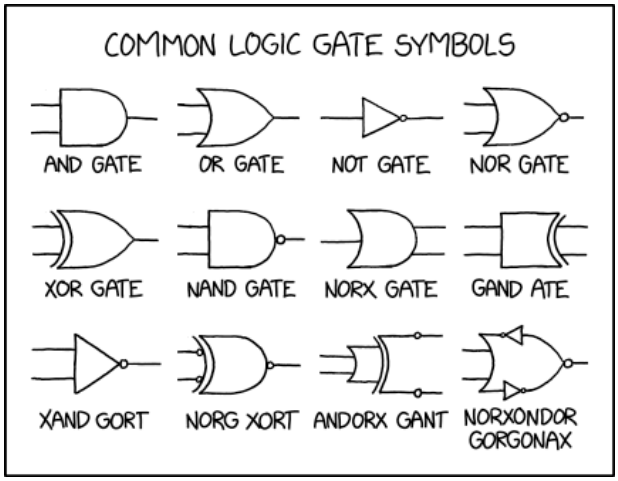

FIG. 1. Comic on logic gates, from xkcd [4]

MODELLING

The modeling procedure consists of the choice of the logic gates of interest, the reasoning behind the choices, the discretization scheme for the wave equation and simulation details, and lastly, the design of the gate geometries. In summary, the choice of the logic gates was determined to **keep the computational demands within the affordable limit**; the discretization is done using **finite difference method for the spatial solution** and **explicit Euler method for temporal solution** of the wave equation; and the geometries were **extracted from the main article** by assigning rigid boundary conditions.

Logic gate selection

The researchers claimed that the passive structures, including several Helmholtz resonators with different width ratios, may be turned into OR, XOR, AND, and NOT logic gates [1].
The parameters which are used to define the geometry are the width and length of a single channel (w, l), the width, gap length, and total length of a single Helmholtz resonator (w_2, l_1, l_2), the inner-channel width (w_1), the wall thickness (w_1), and the width of output channels and length of intersection region (d, d_1). Despite the high number of parameters in the geometric description, these parameters are necessary but insufficient to model the structure. The regions for the incident wave at the inlet and the transmitted wave at the outlet are ill-defined, and their lengths are not prescribed. For simplicity, these regions are assumed to be squares by using the width information. Moreover, the prescribed wall thickness is observed not to satisfy the total length (l). This means that the wall thickness in the lateral direction (w_3) may differ from the wall thickness in the longitudinal direction.

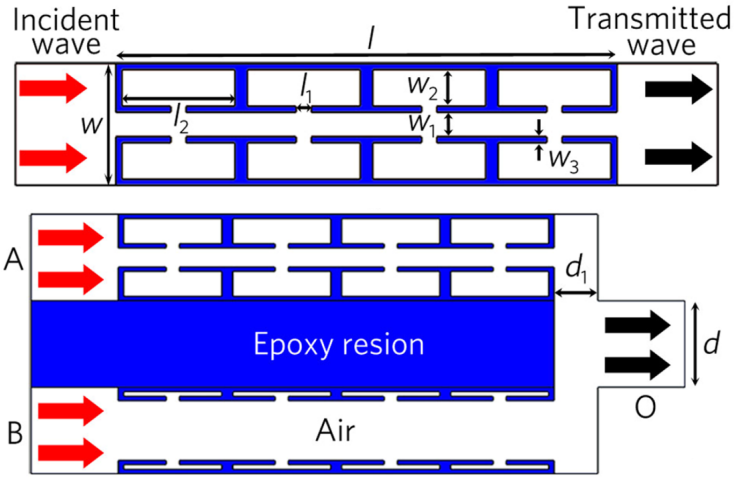

FIG. 2. Schematic of the logic gate geometries. Upper and lower channels may have different width ratios (w_1/w) according to the logic gate of interest. Taken from [1].

In this project, due to the complexity of AND gate geometry, which would have required finer meshing and more computational power, and the triviality of NOT gate geometry which has the identical geometrical structure as XOR gate, modeling of the logic gates is limited to OR and XOR gates [1]. The geometries for OR and XOR logic gates which are included in this project can be found in Figure 3, whereas AND and NOT logic gates which are excluded from this project, can be found in Figures 4 and 5.

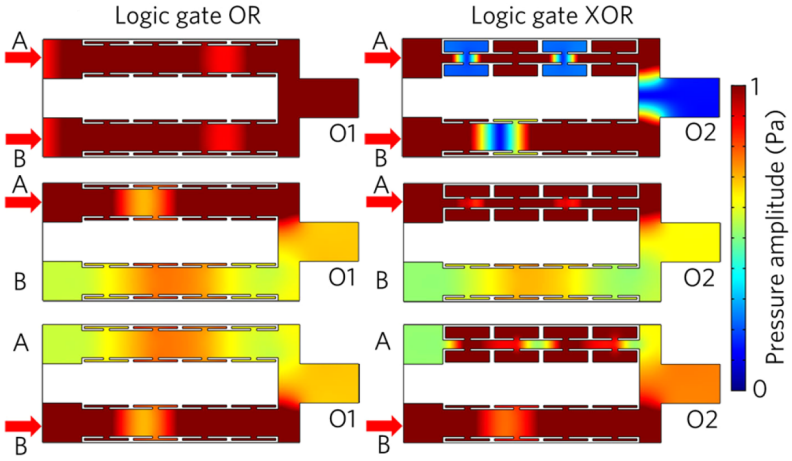

FIG. 3. Exemplary OR and XOR logic gates including the spatial pressure amplitude distributions for different input signals. Taken from [1].

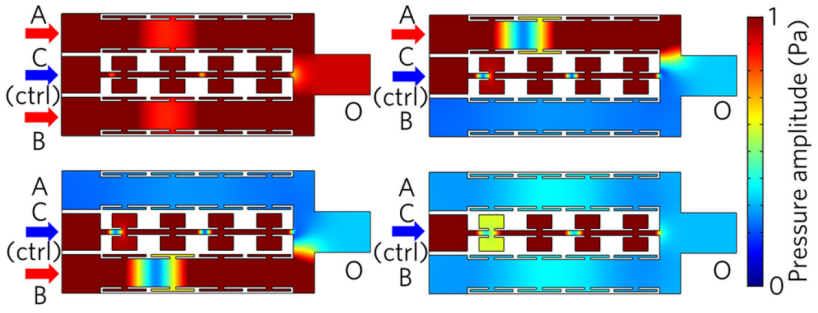

FIG. 4. Exemplary AND logic gate including the spatial pressure amplitude distribution for different input signals. The structure includes a third channel (C) as the control signal. Taken from [1].

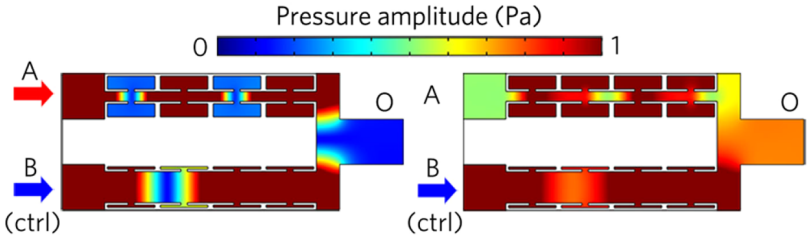

FIG. 5. Exemplary NOT logic gate including the spatial pressure amplitude distribution for different input signals. As will be seen by the following figures, the NOT gate structure is identical to the XOR gate, and the researchers have considered two cases already present in their XOR gate study. Taken from [1].

The proposed logic gate geometries for XOR and OR gates can be found in Figure 5. One can easily model the logic gate for the NOT gate by introducing the width ratios associated with the model into the script. For the AND gate, on the other hand, since the structure consists of not 2 but 3 channels, the current script does not support its modeling of it.

Wave equation

To model the propagation of waves within the logic gate geometry in time, one needs to numerically solve the wave equation in 2D (x, y):

$$\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} \tag{1}$$

where p is the pressure and c is the speed of sound.

For the numerical solution, there are multiple schemes present: finite difference method, finite element method, and finite volume method. The finite difference method is selected and used in this project to ease the implementation. Time advancement is obtained using the implicit method (backward Euler method) that is known to be unconditionally stable. With these methods, the error propagation within the solution is 1^{st} order in time and 2^{nd} order in space.

The implicit method has the following equation for time advancement:

$$\frac{p_{i,j}^{t+1} - p_{i,j}^{t}}{\Delta t} + \frac{c}{2\Delta x}\left(p_{i+1,j+1}^{t+1} - p_{i-1,j-1}^{t+1}\right) = 0 \tag{2}$$

where Δt is the timestep and Δx is the grid size.

This equation can be used to determine the stability condition, often called CFL (Courant–Friedrichs–Lewy) condition, that is $CFL = c\frac{\Delta t}{\Delta x} < 1$. For higher stability, the CFL value is often selected as ≤ 0.7, which will be set to 0.7 in this

project and used to determine the timestep Δt for the used grid size $\Delta x = 1$.

The finite difference discretization results in the following set of equations:

$$p_{i,j,t+\Delta t} = 2p_{i,j,t} - p_{i,j,t-\Delta t} + CFL^2 \left(p_{i+1,j,t} + p_{i,j+1,t} + p_{i-1,j,t} + p_{i,j-1,t} - 4p_{i,j,t}\right) \tag{3}$$

$$p_{0,j,t+\Delta t} = p_{1,j,t} - \frac{CFL-1}{CFL+1}\left(p_{1,j,t+\Delta t} - p_{0,j,t}\right) \tag{4}$$

$$p_{L,j,t+\Delta t} = p_{L-1,j,t} - \frac{CFL-1}{CFL+1}\left(p_{L-1,j,t+\Delta t} - p_{L,j,t}\right) \tag{5}$$

$$p_{i,0,t+\Delta t} = p_{i,1,t} - \frac{CFL-1}{CFL+1}\left(p_{i,1,t+\Delta t} - p_{i,0,t}\right) \tag{6}$$

$$p_{i,L,t+\Delta t} = p_{i,L-1,t} - \frac{CFL-1}{CFL+1}\left(p_{i,L-1,t+\Delta t} - p_{i,L,t}\right) \tag{7}$$

where eqn. (3) is to be used for the grids not near the boundaries, eqn. (4) and eqn. (5) are to be used for the grids neighboring the boundaries for $x = 0$ and $x = L$, and eqn. (6) and eqn. (7) to be used for the grids neighboring the boundaries for $y = 0$ and $y = L$ given that 0 and L are some coordinates corresponding to the nearby of boundaries. One can note that with eqn.s (4-7), the boundaries are assumed to satisfy **rigid boundary conditions**, as prescribed by the researchers [1].

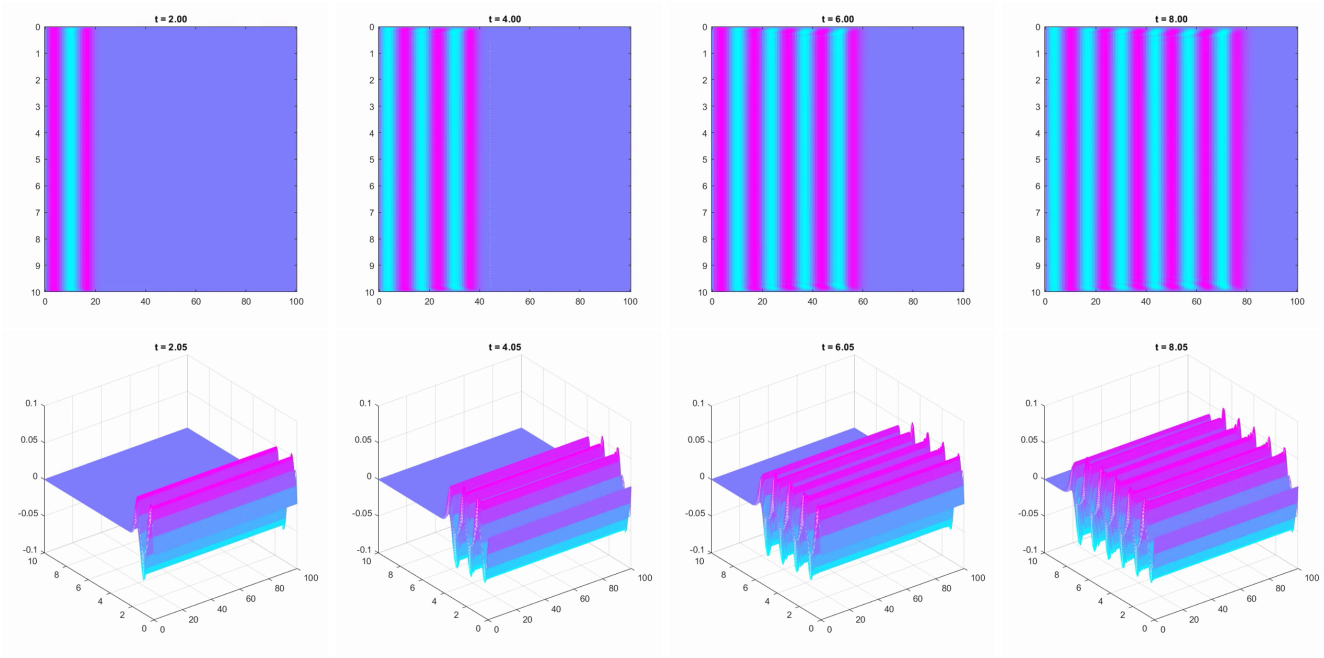

FIG. 6. Propagation of a sinusoidal wave in an environment without boundaries using implicit and finite difference methodology for the numerical solution. Upper images show the propagation in 2D with pressure magnitude, and lower images show the corresponding sine waves in 3D.

Boundaries

To construct the logic gate geometries shown for OR and XOR gates in Figure 3, one needs to define the geometric parameters to be used as an input and will determine the coordinates of the rigid boundaries.

As mentioned before, the geometric information published by the researchers is insufficient to generate the geometries [1]. Thus, while filling the missing parts with the most reasonable values by visually investigating the proposed

geometries, see Figure 3, one must note that this discrepancy may lead to symmetry and wave cancellation errors. The geometric parameters which are used to construct the simulations are tabulated as follows. One can note that all the parameters are nondimensional with respect to the input signal wavelength, which requires determining the input sine wave frequencies so that frequency in air satisfies the dimensional relation, $f_{air} = 3430$ Hz.

TABLE I. Geometric Inputs Used for Simulation Construction

Geometric Property	**Value**	**Geometric Property**	**Value**
Input signal wavelength (λ)	*to be set*	Vertical wall thickness (w_3)	$0.05\ w$
Length of upper/lower channel (l)	$0.5\ \lambda$	Horizontal wall thickness (w_4)	$\frac{1}{5}\ (l - 4\ l_2)$
Width of upper/lower channel (w)	$0.5\ \lambda$	Width of upper channel Helmholtz resonator ($w_{2,upper}$)	$\frac{1}{2}\ (w - w_{1,upper} - 4\ w_3)$
Width of output channel (d)	$0.1\ \lambda$	Width of lower channel Helmholtz resonator ($w_{2,lower}$)	$\frac{1}{2}\ (w - w_{1,lower} - 4\ w_3)$
Width of intersection region (d_1)	$0.055\ \lambda$	Total length of logic gate (L_x)	$w + l + d_1 + d$
Gap length of Helmholtz resonator (l_1)	$0.03\ l$	Total width of logic gate (L_y)	$2\ w + d$
Total length of Helmholtz resonator (l_2)	$0.23\ l$	Grid size in x-direction (ΔX)	1
Width of upper channel Helmholtz resonator ($w_{1,upper}$)	OR Gate: $0.69\ \lambda$ XOR Gate: $0.69\ \lambda$	Grid size in y-direction (ΔY)	1
Width of lower channel Helmholtz resonator ($w_{1,lower}$)	OR Gate: $0.69\ \lambda$ XOR Gate: $0.22\ \lambda$	Speed of sound ($c = \lambda\ f$)	1

With the parameters given in Table 1, the following OR and XOR geometries were constructed in a MATLAB environment, which then used rigid boundary coordinates for wave propagation simulations.

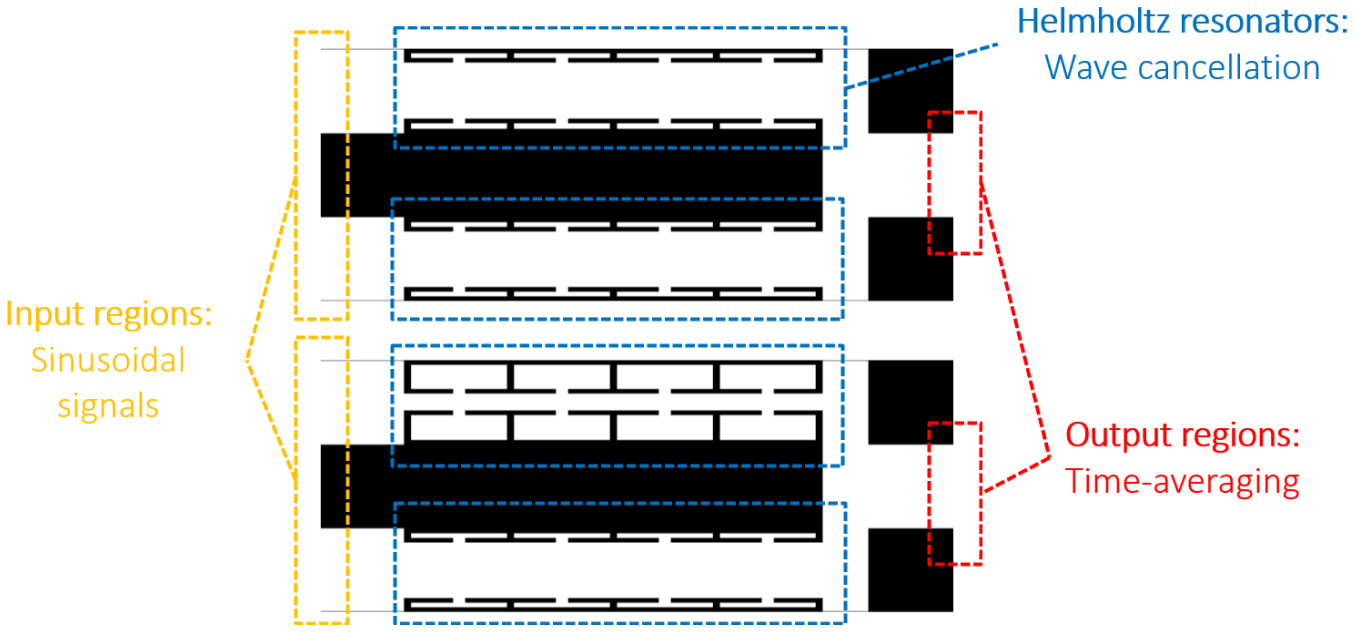

FIG. 7. OR and XOR logic gates geometries constructed in MATLAB. The upper geometry belongs to the OR gate, and the lower gate belongs to the XOR gate. Helmholtz resonators and input and output regions are depicted in the figure.

RESULTS AND COMPARISON

Recalling that the geometric parameters were nondimensional with respect to the input wavelength λ, one must determine the input sine wave frequency in terms of dimensional numbers to replicate the input conditions provided by the researchers [1]. The sinusoidal input wave has the following form:

$$p(x = 0, \text{inlet}_{y=0 \text{ to } y=\text{end}}, t) = A\ sin(2\pi\ t\ f); \tag{8}$$

The amplitude term, A, in eqn. (8) may be replaced with either 1 or 0 according to the input signal conditions, which may have the following combinations: {Upper channel, Lower channel} = $\{1,1\}$, $\{1,0\}$, $\{0,1\}$. Note that the $\{0,0\}$ condition is disregarded due to its triviality. During the simulations, the pressure amplitudes have been considered for visualization and averaging due to the importance of the amplitude but not the magnitude in the acoustical sense. At this point, one can further define a nondimensional frequency by using the frequency of air (f_{air}), as researchers used air to fill the channels, the speed of sound c and the wavelength of the input signal, which determines the geometric parameters: $f' = f \times \lambda/c$. This approach becomes convenient when dealing with the working band, where the researchers showed that OR and XOR gates may be realized for the range $f \approx 3000 - 3800$ Hz. This range approximately corresponds to $f' = 0.9 - 1.1$ regarding the nondimensional frequency. After the definition of nondimensional frequency, the wave propagation simulations were conducted by sweeping over 1o frequency values ($f' = 0.1, 0.3, ..., 1.7, 1.9$). The average pressure values at the outlet region were measured at the end of each simulation for a particular gate (OR or XOR) and a particular input state ($\{1,1\}$, $\{1,0\}$ or $\{0,1\}$). Each simulation lasted for 20000 timesteps, and a total of 60 simulations were conducted with this runtime for each (10 frequencies, 2 gates, 3 input states). Due to the visualization efforts and better readability of the script, the script was not optimized, and each of the 60 simulations required 20+ minutes of runtime with the available computational resources. Consequently, it became inconvenient to create a graphical user interface and control every geometric parameter (Table 1) and input parameter (eqn. 8), although there were efforts.

By sweeping over the frequencies, one determines whether the logic gates can be realized with the used numerical scheme without worrying about the dimensional numbers or possible problems due to geometric and meshing. Simulation snapshots from 5 different frequency values ($f' = 0.1, 0.5, 0.9, 1.3, 1.7$) are presented in the following figures, Figures 8-13.

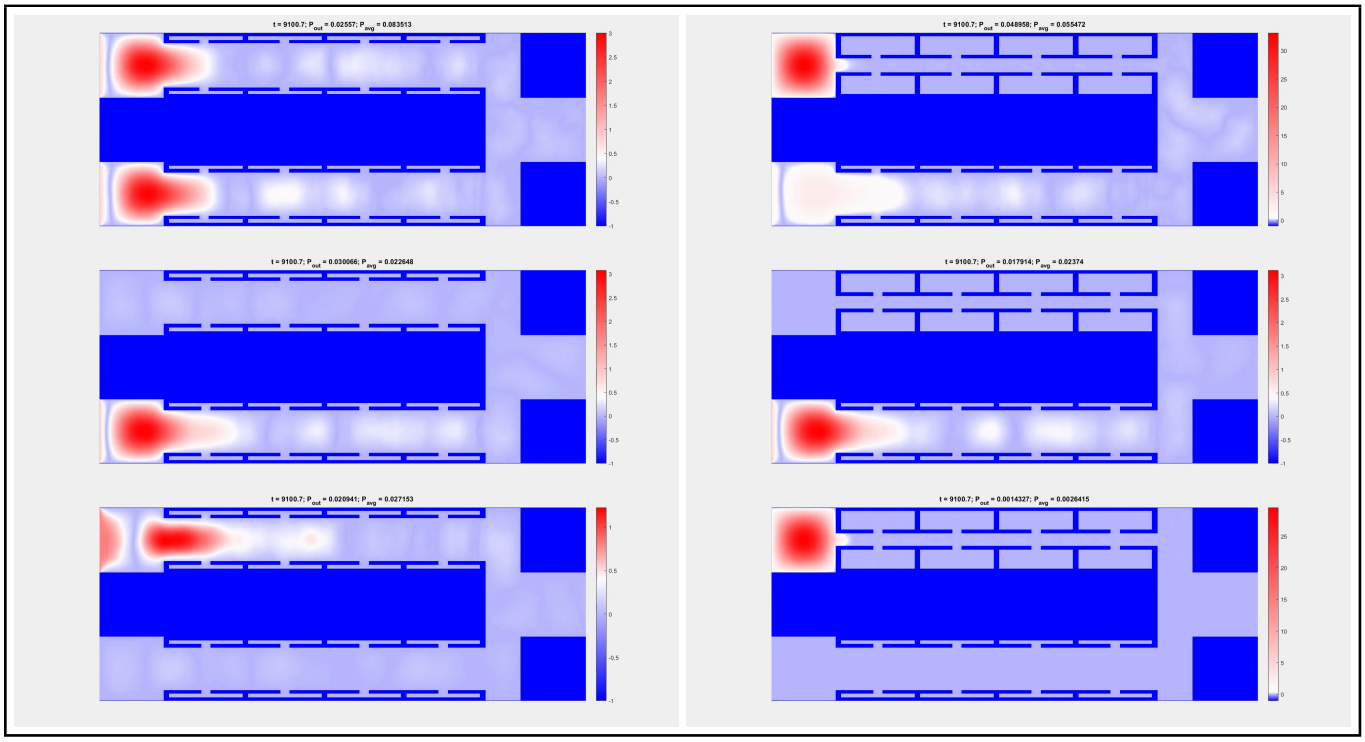

FIG. 8. Snapshots from the simulations with $f' = 0.1$ at a timestep approximately 60% of the total runtime. Left: OR gate, Right: OR gate. Top: Input signal of $\{1,1\}$, Middle: Input signal of $\{1,0\}$, Bottom: Input signal of $\{0,1\}$.

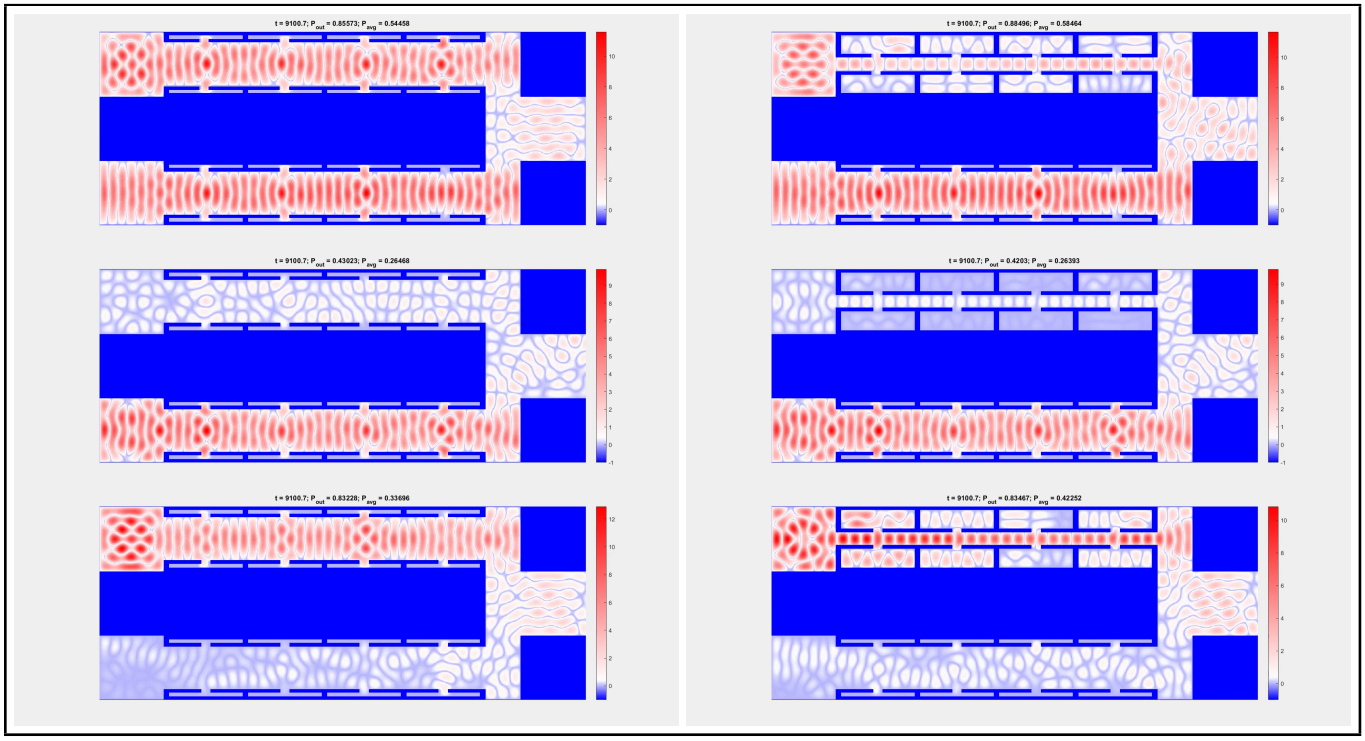

FIG. 9. Snapshots from the simulations with $f' = 0.5$ at a timestep approximately 60% of the total runtime. Left: OR gate, Right: OR gate. Top: Input signal of $\{1, 1\}$, Middle: Input signal of $\{1, 0\}$, Bottom: Input signal of $\{0, 1\}$.

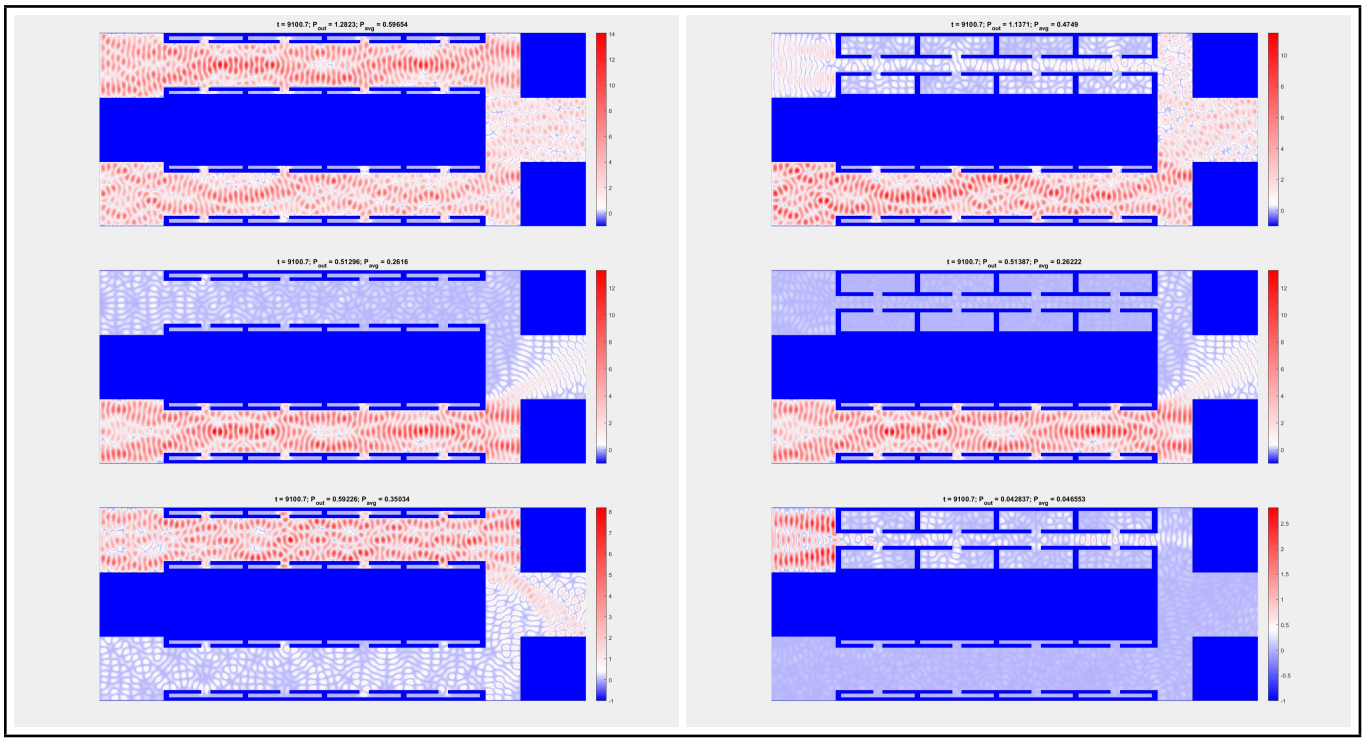

FIG. 10. Snapshots from the simulations with $f' = 0.9$ at a timestep approximately 60% of the total runtime. Left: OR gate, Right: OR gate. Top: Input signal of $\{1, 1\}$, Middle: Input signal of $\{1, 0\}$, Bottom: Input signal of $\{0, 1\}$. Note that this frequency value is within the limits of the working band, and the output pressure time averages should satisfy the truth table accordingly unless there is no restraint due to the numerical scheme or the geometry.

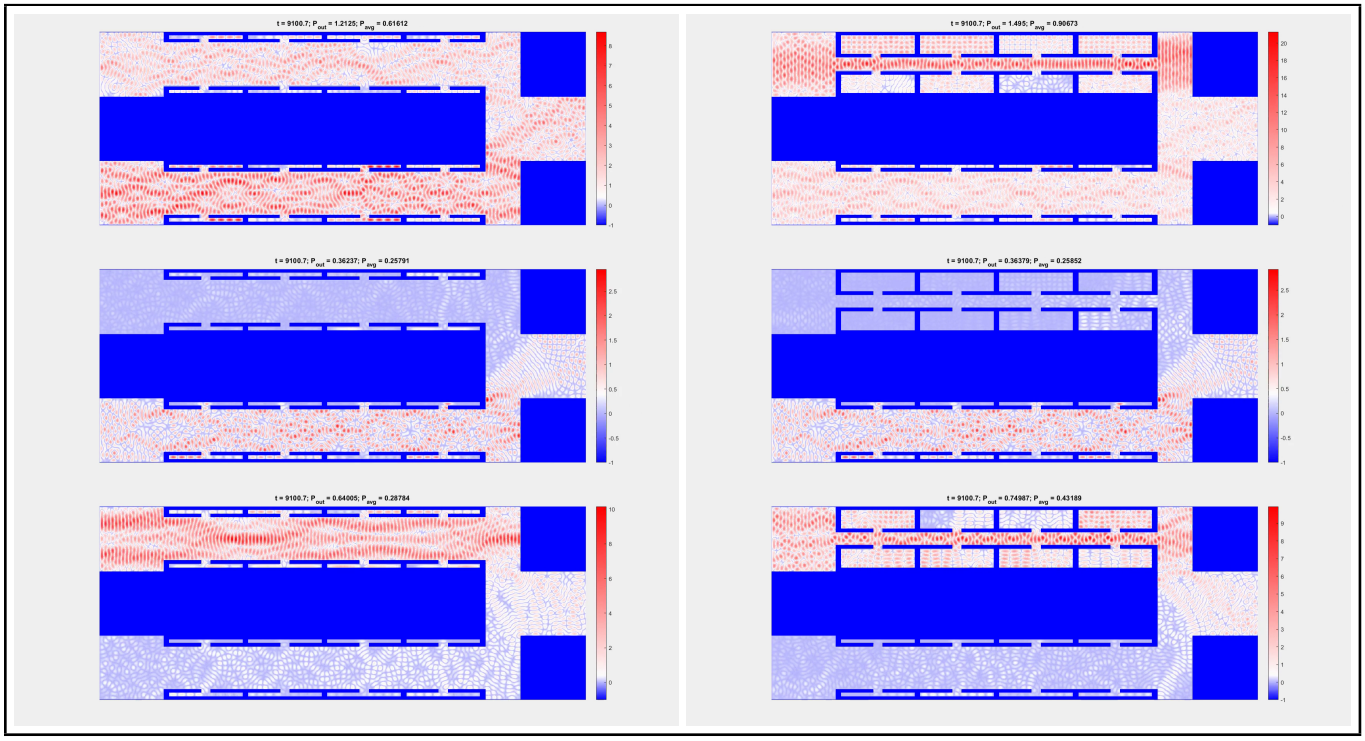

FIG. 11. Snapshots from the simulations with $f' = 1.3$ at a timestep approximately 60% of the total runtime. Left: OR gate, Right: OR gate. Top: Input signal of $\{1, 1\}$, Middle: Input signal of $\{1, 0\}$, Bottom: Input signal of $\{0, 1\}$.

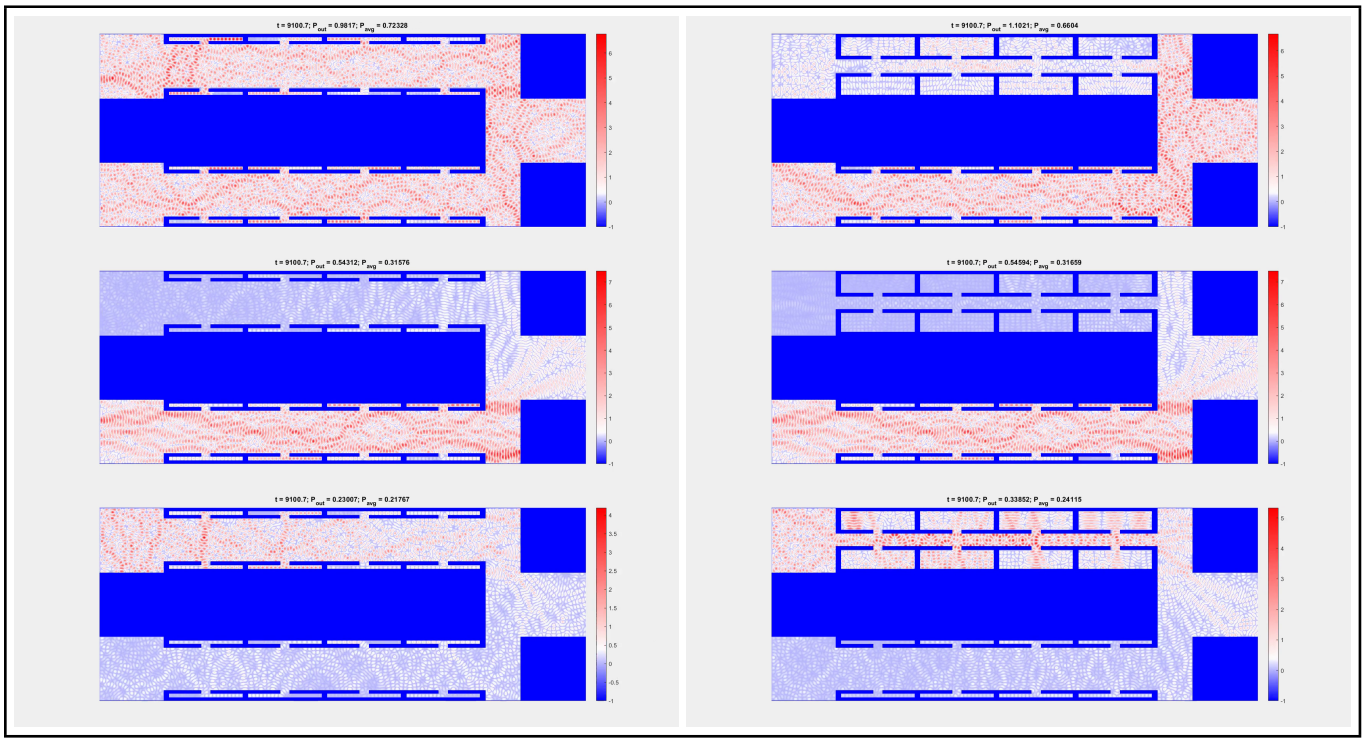

FIG. 12. Snapshots from the simulations with $f' = 1.7$ at a timestep approximately 60% of the total runtime. Left: OR gate, Right: OR gate. Top: Input signal of $\{1, 1\}$, Middle: Input signal of $\{1, 0\}$, Bottom: Input signal of $\{0, 1\}$. Note that as the frequency increases, the Helmholtz resonators become useless to create a phase difference and cancel waves, but rather become regions that waves can travel without significantly interrupting the other waves.

The time-averaged pressure amplitudes at the outlet region, where neither the averaging region nor the averaging methodology was defined by the researchers, may be obtained for the simulations done over a range of frequencies.

The results presented in the following figure show that computational modeling of the OR gate was **successful** as the OR gate results in the following truth table:

$$\text{Input signal: } \{1,1\} \rightarrow \text{Output signal: } \{1\}$$
$$\text{Input signal: } \{1,0\} \rightarrow \text{Output signal: } \{1\}$$
$$\text{Input signal: } \{0,1\} \rightarrow \text{Output signal: } \{1\}$$

where the threshold pressure amplitude value of 0.4 Pa was used in accordance with the researchers [1].

On the other hand, the following figure shows that computational modeling of the XOR gate was **unsuccessful** as the XOR gate did not result in the ideal truth table:

$$\text{Input signal: } \{1,1\} \rightarrow \text{Output signal: } \{0\}$$
$$\text{Input signal: } \{1,0\} \rightarrow \text{Output signal: } \{1\}$$
$$\text{Input signal: } \{0,1\} \rightarrow \text{Output signal: } \{1\}$$

It rather resulted in the following truth table:

$$\text{Input signal: } \{1,1\} \rightarrow \text{Output signal: } \{1\}$$
$$\text{Input signal: } \{1,0\} \rightarrow \text{Output signal: } \{0\}$$
$$\text{Input signal: } \{0,1\} \rightarrow \text{Output signal: } \{1\}$$

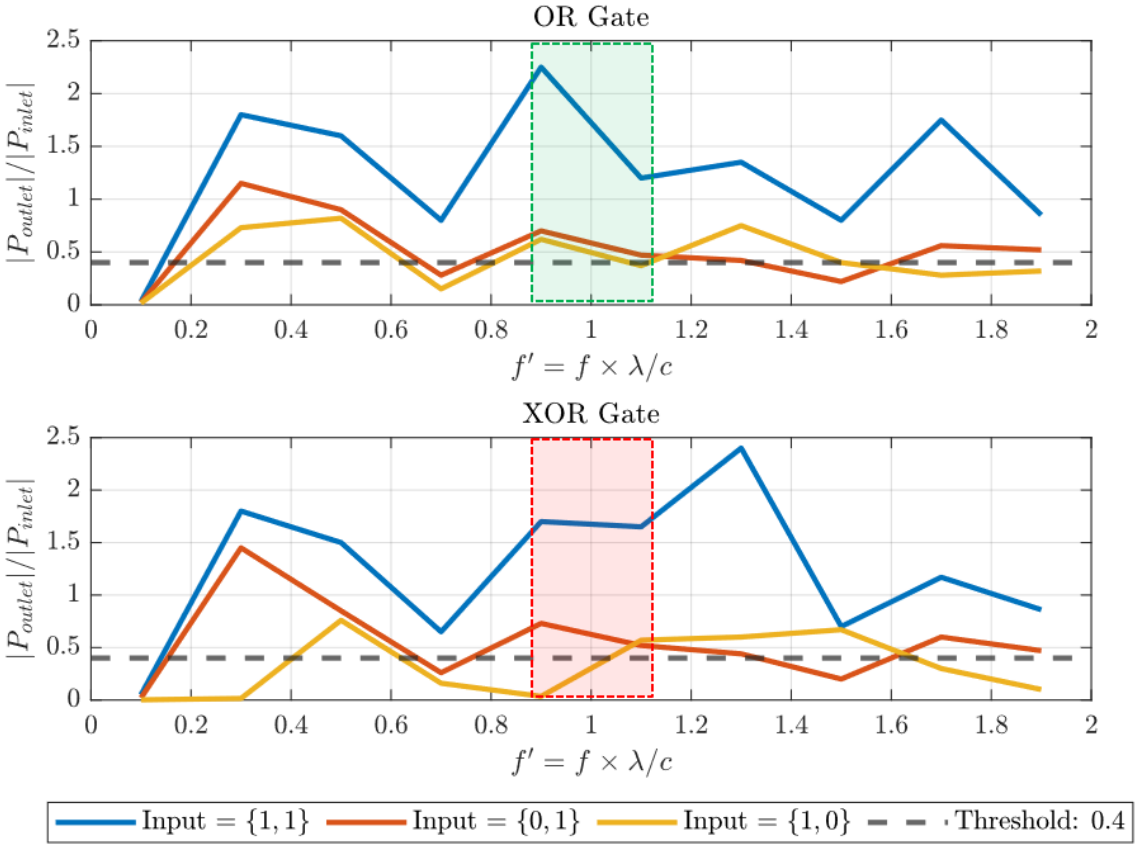

FIG. 13. Time-averaged output pressures over the range of nondimensional frequencies. The working ranges that are proposed by the researchers are depicted on the plots, where the computational model of OR gate was able to realize the expected results, whereas XOR gate was not able to realize not only in the working band but over non of the studied frequency values.

CONCLUSION

To conclude this study, we may consider some observations done throughout the project, starting from the construction of the simulations to the obtained results:

As can be seen in Figure 14, the wave canceling and waves which are emerging in the same space periodically in time are observed. This may be argued due to the Helmholtz resonators placed in the channels. However, as the time-averaged pressure amplitudes within the whole logic gate geometry are not studied, but the focus was only on the outlet region due to memory limitations, there is no quantitative analysis of the pressure distribution over the logic gates. One may note from Figures 3-5 that the COMSOL analysis done by the researchers showed that the pressure distribution was not uniform but had low amplitude regions, such as some parts of the regions or inside of the Helmholtz resonators.

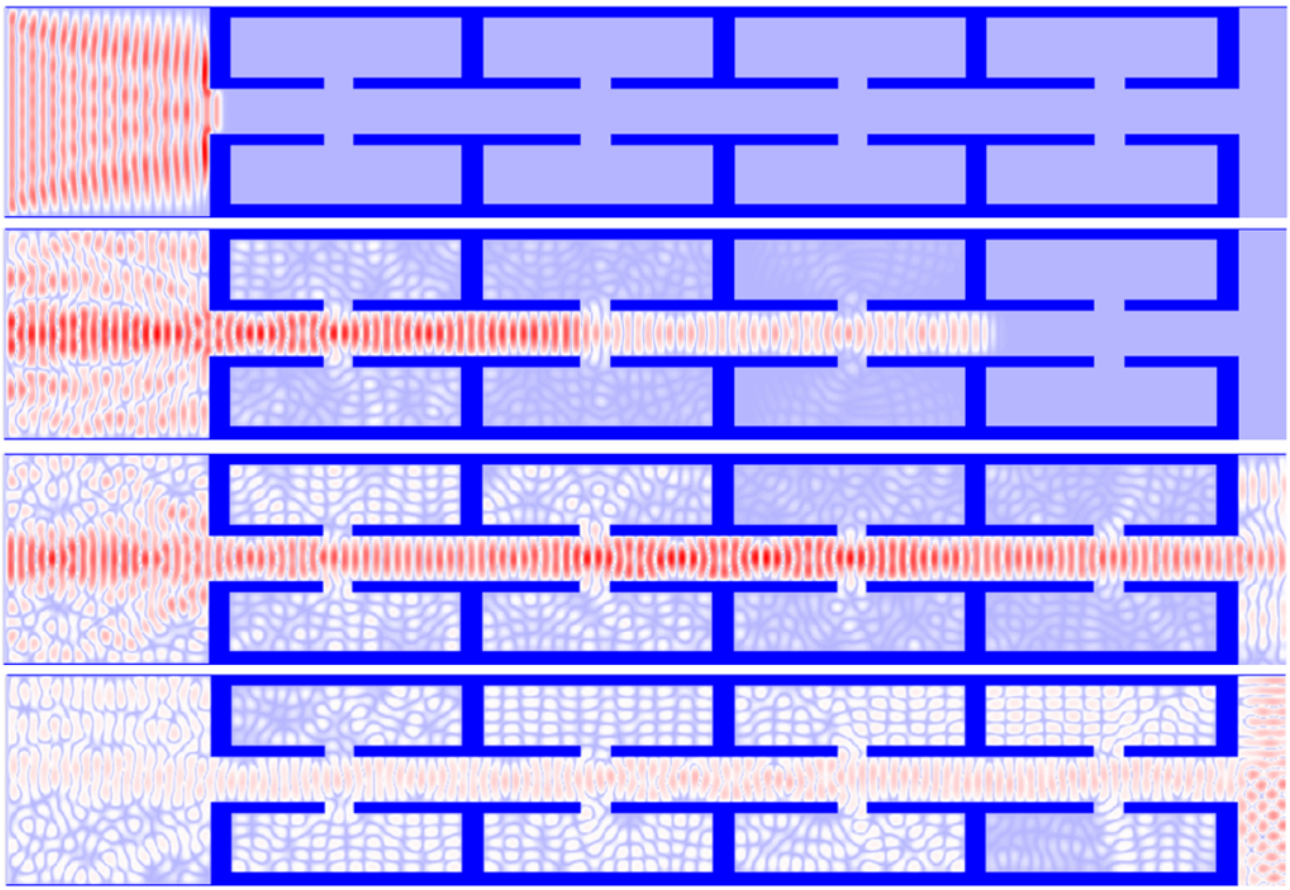

FIG. 14. f=1.9.

As the main goal of this project, one of the logic gates was successfully simulated in MATLAB, which was achieved regardless of the uncertainties in geometry, working band, and averaging methodology. On the other hand, despite the OR gate model showing the expected behavior, the XOR gate model was not able to reproduce the results for none of the studied frequency values. This observation can be interpreted as there was no shift in the working band for any reason, so the expected behavior might have been observed for a different working band. The insufficiency is due to the numerical method and logic gate geometry, such as meshing.

The last observation may be further supported by the fact that the gates did not preserve the symmetry of pressure amplitudes, either due to geometric asymmetries that are present but neglected in the script or due to the accumulation of errors in the numerical solution of the wave equation. A naive approach to eliminate the current inefficiencies may be using the finite element method for the solution of the wave equation and decreasing the grid sizes, which should result in fewer errors within the numeric and better meshing.

REFERENCES

[1] Wang, Y., Xia, Jp., Sun, Hx. et al., "Binary-phase acoustic passive logic gates," *Nature Sci Rep*, 2019. [Online]. Available: https://doi.org/10.1038/s41598-019-44769-0. [Accessed: 19-May-2022].

[2] Ting Zhang, Ying Cheng, Jian-zhong Guo, et al., "Acoustic logic gates and Boolean operation based on self-collimating acoustic beams," *Appl. Phys. Lett.*, 2015. [Online]. Available: https://doi.org/10.1063/1.4915338. [Accessed: 19-May-2022].

[3] Yu-Jing Lu et al., "Acoustic logic gates by a curved waveguide with ultrathin metasurfaces," *J. Phys. D: Appl. Phys.*, 2020. [Online]. Available: https://doi.org/10.1088/1361-6463/ab483e. [Accessed: 19-May-2022].

[4] "Logic Gates," *xkcd*. [Online]. Available: https://xkcd.com/2497/. [Accessed: 16-May-2022].

[5] Komatsuzaki T., Iwata Y., Morishita S., "Modelling of Incident Sound Wave Propagation around Sound Barriers Using Cellular Automata," *Springer*, 2012. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-33350-7_40. [Accessed: 19-May-2022].

[6] Chapra, S. C., & Canale, R. P., "Numerical Methods for Engineers," *McGraw-Hill Higher Education*, 2015.

APPENDIX: MATLAB SCRIPT

```
clc; clear

%%% Wave equation: (d^2/dx^2 +cd d^2/dy^2) p = 1/c^2 (d^2/dt^2) p

%% OR Gate + XOR Gate, Total runtime: 20000
% Domain definition
c = 1;
lambda = 2000; % 10 cm ... 10 cm to 2000 units lambda
prefactors = (0.1:0.2:1.9); % as a prefactor for frequency sweep
OUTPUT_VALUES = []; % save output pressure values

for i = 1:size(prefactors,2)

    prefactor = prefactors(i); disp("prefactor is " + num2str(prefactor) + ".");
    freq = prefactor*c/lambda;

    tic
    input = 11;
    filename = "OR_input" + num2str(input) + "prefactor_" + num2str(prefactor) + ".gif";
    filename2 = "OR_input" + num2str(input) + "prefactor_" + num2str(prefactor) + ".png";
    out = solver11_OR(lambda, filename, freq, filename2); disp("solver11_OR is completed.");
    OUTPUT_VALUES = [OUTPUT_VALUES; out];

    filename = "XOR_input" + num2str(input) + "prefactor_" + num2str(prefactor) + ".gif";
    filename2 = "XOR_input" + num2str(input) + "prefactor_" + num2str(prefactor) + ".png";
    out = solver11_XOR(lambda, filename, freq, filename2); disp("solver11_XOR is completed.");
    OUTPUT_VALUES = [OUTPUT_VALUES; out];

   input = 10;
    filename = "OR_input" + num2str(input) + "prefactor_" + num2str(prefactor) + ".gif";
    filename2 = "OR_input" + num2str(input) + "prefactor_" + num2str(prefactor) + ".png";
    out = solver10_OR(lambda, filename, freq, filename2); disp("solver10_OR is completed.");
    OUTPUT_VALUES = [OUTPUT_VALUES; out];

    filename = "XOR_input" + num2str(input) + "prefactor_" + num2str(prefactor) + ".gif";
    filename2 = "XOR_input" + num2str(input) + "prefactor_" + num2str(prefactor) + ".png";
    out = solver10_XOR(lambda, filename, freq, filename2); disp("solver10_XOR is completed.");
    OUTPUT_VALUES = [OUTPUT_VALUES; out];

     input = 01;
     filename = "OR_input01" + "prefactor_" + num2str(prefactor) + ".gif";
     filename2 = "OR_input01" + "prefactor_" + num2str(prefactor) + ".png";
     out = solver01_OR(lambda, filename, freq, filename2); disp("solver01_OR is completed.");
     OUTPUT_VALUES = [OUTPUT_VALUES; out];

    filename = "XOR_input01" + "prefactor_" + num2str(prefactor) + ".gif";
    filename2 = "XOR_input01" + "prefactor_" + num2str(prefactor) + ".png";
    out = solver01_XOR(lambda, filename, freq, filename2); disp("solver01_XOR is completed.");
    OUTPUT_VALUES = [OUTPUT_VALUES; out];
    toc

end

filename = 'savefile.mat';
save(filename)

function out = solver11_OR(lambda, filename, freq, filename2)

    % Geometric constraints
    l = lambda / 2;
    w = lambda / 10;
    d = 0.1 * lambda;
    d1 = 0.055 * lambda;

    l1 = (0.03 * l);
    l2 = (0.23 * l);
    w1_upper = (0.69 * w); % changing
    w1_lower = (0.69 * w); % changing
    w3 = (0.05 * w);
    w2_upper = ((w - w1_upper - 4 * w3) / 2);
    w2_lower = ((w - w1_lower - 4 * w3) / 2);
    w4 = ((l-4*12)/5);
```

```
ld2 = ((l-4*l1)/8);
ld3 = ((l2-l1)/2);

Lx = w + l + d1 + d;
Ly = 2*w + d;
dx = 1; dy = 1;
nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;

x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);

% Field definition

p = 0*ones(nx,ny);
p_prev = p; p_next = p;

CFL = 0.7; % c * dt/dx
c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
dt = CFL*dx/c;
output_pres = [];
time = [];

% Initial conditions
T = 20000*dt; % Runtime

t = 0;
f = figure(1);
set(gcf,'Position',[100 100 1400 500])

while t<T
    % Absorbing Boundaries
    p_next(1,:) = p(2,:) + ((CFL-1)/(CFL+1))*(p_next(2,:)-p(1,:));
    p_next(end,:) = p(end-1,:) + ((CFL-1)/(CFL+1))*(p_next(end-1,:)-p(end,:));

    % Rigid Boundaries
    p(1:end, [1 end]) = 0; % Upper and lower boundaries
    %p([1 end], 1:end) = 0; % Left and right boundaries
    p(1:w+l, w+dx:end-w-2*dx) = 0; % Middle region
    p(w+l+d1:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
    p(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3

    p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
        w-w3-l*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
    p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
        end-w+w3+0*dx:end-w+w3+w2_lower+l*dx]) = 0; % 2/3
    p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
    p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3

    t = t+dt;
    p_prev = p; p = p_next;

    %p_next(1:end,1) = p(1:end,2) + ((CFL-1)/(CFL+1))*(p_next(1:end,2)-p(1:end,1));
    p_next(1:end,end) = p(1:end,end-1) + ((CFL-1)/(CFL+1))*(p_next(1:end,end-1)-p(1:end,end));

    % Source 1
    p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);

    % Source 2
    p(2, 2:lambda/10-1) = 1*sin(2*pi*t*freq);

    % Rigid Boundaries
    p(1:end, [1 end]) = 0; % Upper and lower boundaries
    %p([1 end], 1:end) = 0; % Left and right boundaries
    p(1:w+l, w+dx:end-w-2*dx) = 0; % Middle region
    p(w+l+d1:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
    p(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3

    p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
        w-w3-l*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
    p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
        end-w+w3+0*dx:end-w+w3+w2_lower+l*dx]) = 0; % 2/3
    p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
    p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3

    % Finite difference solution
    for i = 2:nx-1
        for j = 2:ny-1
            p_next(i,j) = 2*p(i,j) - p_prev(i,j) + CFL^2 * ( p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
        end
    end

    k = abs(p);
    % Rigid Boundaries
    k(1:end, [1 end]) = -1; % Upper and lower boundaries
    %p([1 end], 1:end) = 0; % Left and right boundaries
    k(1:w+l, w+dx:end-w-2*dx) = -1; % Middle region
    k(w+l+d1:end,[1:w w+d+dx:end]) = -1; % Right bottom and top regions
    k(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = -1; k(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = -1; % 1/3

    k([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
        w-w3-l*dx-w2_upper:w-w3-2*dx]) = -1; % 2/3
    k([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
        end-w+w3+0*dx:end-w+w3+w2_lower+l*dx]) = -1; % 2/3
    k([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = -1; % 3/3
    k([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = -1; % 3/3

    if round(mod(t/dt,25)) == 1
        output = mean(mean(k(end-d/2:end,w+dx:w+d))); %max(max(k(w+l+d1:end,w+dx:w+d)));
        output_pres = [output_pres output];
        time = [time t];
    end
    if round(mod(t/dt,25)) == 1
        % Visualize
        % COLORBAR
        indexValue = 0.4;     % value for which to set a particular color
        topColor = [1 0 0];         % color for maximum data value (red = [1 0 0])
```

```
            indexColor = [1 1 1];       % color for indexed data value (white = [1 1 1])
            bottomcolor = [0 0 1];      % color for minimum data value (blue = [0 0 1])
            largest = max(max(k));
            smallest = min(min(k));
            L = size(p,1);
            index = L*abs(indexValue-smallest)/(largest-smallest);
            customCMap1 = [linspace(bottomcolor(1),indexColor(1),100*index)',...
                        linspace(bottomcolor(2),indexColor(2),100*index)',...
                        linspace(bottomcolor(3),indexColor(3),100*index)'];
            customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index))',...
                        linspace(indexColor(2),topColor(2),100*(L-index))',...
                        linspace(indexColor(3),topColor(3),100*(L-index))'];
            customCMap = [customCMap1;customCMap2];  % Combine colormaps
            colormap(customCMap)

            imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
            colorbar; axis equal;
            grid off; box off; axis off
            ax = gca;
            ax.YDir = 'normal';
            tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output_pres)));
            txt = "t = " + tstring + "; P_{out} = " + pstring + "; P_{avg} = " + astring;
            title(txt); %title(sprintf('t = %.6f',t));
            drawnow limitrate
            frame = getframe(f);
            im = frame2im(frame);
            [imind,cm] = rgb2ind(im,256);
            if t == dt
                imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
            else
                imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
            end
        end
    end
    figure(2); plot(time,output_pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
    out = output_pres;
    close all
end

function out = solver10_OR(lambda, filename, freq, filename2)

    % Geometric constraints
    l = lambda / 2;
    w = lambda / 10;
    d = 0.1 * lambda;
    d1 = 0.055 * lambda;

    l1 = (0.03 * l);
    l2 = (0.23 * l);
    w1_upper = (0.69 * w); % changing
    w1_lower = (0.69 * w); % changing
    w3 = (0.05 * w);
    w2_upper = ((w - w1_upper - 4 * w3) / 2);
    w2_lower = ((w - w1_lower - 4 * w3) / 2);
    w4 = ((l-4*l2)/5);
    ld2 = ((l-4*l1)/8);
    ld3 = ((l2-l1)/2);

    Lx = w + l + d1 + d;
    Ly = 2*w + d;
    dx = 1; dy = 1;
    nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;

    x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);

    % Field definition

    p = 0*ones(nx,ny);
    p_prev = p; p_next = p;

    CFL = 0.7; % c * dt/dx
    c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
    dt = CFL*dx/c;
    output_pres = [];
    time = [];

    % Initial conditions
    T = 20000*dt; % Runtime

    t = 0;
    f = figure(1);
    set(gcf,'Position',[100 100 1400 500])

    while t<T
        % Absorbing Boundaries
        p_next(1,:) = p(2,:) + ((CFL-1)/(CFL+1))*(p_next(2,:)-p(1,:));
        p_next(end,:) = p(end-1,:) + ((CFL-1)/(CFL+1))*(p_next(end-1,:)-p(end,:));

        % Rigid Boundaries
        p(1:end, [1 end]) = 0; % Upper and lower boundaries
        %p([1 end], 1:end) = 0; % Left and right boundaries
        p(1:w+l, w+dx:end-w-2*dx) = 0; % Middle region
        p(w+l+d1:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
        p(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3

        p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
              w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
        p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
              end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
        p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
        p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3

        t = t+dt;
        p_prev = p; p = p_next;

        %p_next(1:end,1) = p(1:end,2) + ((CFL-1)/(CFL+1))*(p_next(1:end,2)-p(1:end,1));
        p_next(1:end,end) = p(1:end,end-1) + ((CFL-1)/(CFL+1))*(p_next(1:end,end-1)-p(1:end,end));

        % Source 1
```

```
        p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);

        % Source 2
        %p(2, 2:lambda/10-1) = 1*sin(2*pi*t*freq);

        % Rigid Boundaries
        p(1:end, [1 end]) = 0; % Upper and lower boundaries
        %p([1 end], 1:end) = 0; % Left and right boundaries
        p(1:w+1, w+dx:end-w-2*dx) = 0; % Middle region
        p(w+1+dl:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
        p(w:w+1, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3

        p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
              w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
        p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
              end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
        p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
        p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3

        % Finite difference solution
        for i = 2:nx-1
            for j = 2:ny-1
                p_next(i,j) = 2*p(i,j) - p_prev(i,j) + CFL^2 * ( p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
            end
        end

        k = abs(p);
        % Rigid Boundaries
        k(1:end, [1 end]) = -1; % Upper and lower boundaries
        %p([1 end], 1:end) = 0; % Left and right boundaries
        k(1:w+1, w+dx:end-w-2*dx) = -1; % Middle region
        k(w+1+dl:end,[1:w w+d+dx:end]) = -1; % Right bottom and top regions
        k(w:w+1, [1+dx:w3+dx w-w3-dx:w]) = -1; k(w:w+1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = -1; % 1/3

        k([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
              w-w3-1*dx-w2_upper:w-w3-2*dx]) = -1; % 2/3
        k([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
              end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = -1; % 2/3
        k([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = -1; % 3/3
        k([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = -1; % 3/3

        if round(mod(t/dt,25)) == 1
            output = mean(mean(k(end-d/2:end,w+dx:w+d))); %max(max(k(w+1+dl:end,w+dx:w+d)));
            output_pres = [output_pres output];
            time = [time t];
        end
        if round(mod(t/dt,25)) == 1
            % Visualize
            % COLORBAR
            indexValue = 0.4;     % value for which to set a particular color
            topColor = [1 0 0];         % color for maximum data value (red = [1 0 0])
            indexColor = [1 1 1];       % color for indexed data value (white = [1 1 1])
            bottomcolor = [0 0 1];      % color for minimum data value (blue = [0 0 1])
            largest = max(max(k));
            smallest = min(min(k));
            L = size(p,1);
            index = L*abs(indexValue-smallest)/(largest-smallest);
            customCMap1 = [linspace(bottomcolor(1),indexColor(1),100*index)',...
                        linspace(bottomcolor(2),indexColor(2),100*index)',...
                        linspace(bottomcolor(3),indexColor(3),100*index)'];
            customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index))',...
                        linspace(indexColor(2),topColor(2),100*(L-index))',...
                        linspace(indexColor(3),topColor(3),100*(L-index))'];
            customCMap = [customCMap1;customCMap2];  % Combine colormaps
            colormap(customCMap)

            imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
            colorbar; axis equal;
            grid off; box off; axis off
            ax = gca;
            ax.YDir = 'normal';
            tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output_pres)));
            txt = "t = " + tstring + "; P_{out} = " + pstring + "; P_{avg} = " + astring;
            title(txt); %title(sprintf('t = %.6f',t));
            drawnow limitrate
            frame = getframe(f);
            im = frame2im(frame);
            [imind,cm] = rgb2ind(im,256);
            if t == dt
                imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
            else
                imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
            end
        end
    end
    figure(2); plot(time,output_pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
    out = output_pres;
    close all
end

function out = solver01_OR(lambda, filename, freq, filename2)

    % Geometric constraints
    l = lambda / 2;
    w = lambda / 10;
    d = 0.1 * lambda;
    dl = 0.055 * lambda;

    l1 = (0.03 * l);
    l2 = (0.23 * l);
    w1_upper = (0.69 * w); % changing
    w1_lower = (0.69 * w); % changing
    w3 = (0.05 * w);
    w2_upper = ((w - w1_upper - 4 * w3) / 2);
    w2_lower = ((w - w1_lower - 4 * w3) / 2);
    w4 = ((l-4*l2)/5);
    ld2 = ((l-4*l1)/8);
```

```
ld3 = ((l2-l1)/2);

Lx = w + l + dl + d;
Ly = 2*w + d;
dx = 1; dy = 1;
nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;

x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);

% Field definition

p = 0*ones(nx,ny);
p_prev = p; p_next = p;

CFL = 0.7; % c * dt/dx
c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
dt = CFL*dx/c;
output_pres = [];
time = [];

% Initial conditions
T = 20000*dt; % Runtime

t = 0;
f = figure(1);
set(gcf,'Position',[100 100 1400 500])

while t<T
    % Absorbing Boundaries
    p_next(1,:) = p(2,:) + ((CFL-1)/(CFL+1))*(p_next(2,:)-p(1,:));
    p_next(end,:) = p(end-1,:) + ((CFL-1)/(CFL+1))*(p_next(end-1,:)-p(end,:));

    % Rigid Boundaries
    p(1:end, [1 end]) = 0; % Upper and lower boundaries
    %p([1 end], 1:end) = 0; % Left and right boundaries
    p(1:w+l, w+dx:end-w-2*dx) = 0; % Middle region
    p(w+l+dl:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
    p(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3

    p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
        w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
    p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
        end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
    p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
    p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3

    t = t+dt;
    p_prev = p; p = p_next;

    %p_next(1:end,1) = p(1:end,2) + ((CFL-1)/(CFL+1))*(p_next(1:end,2)-p(1:end,1));
    p_next(1:end,end) = p(1:end,end-1) + ((CFL-1)/(CFL+1))*(p_next(1:end,end-1)-p(1:end,end));

    % Source 1
    %p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);

    % Source 2
    p(2, 2:lambda/10-1) = 1*sin(2*pi*t*freq);

    % Rigid Boundaries
    p(1:end, [1 end]) = 0; % Upper and lower boundaries
    %p([1 end], 1:end) = 0; % Left and right boundaries
    p(1:w+l, w+dx:end-w-2*dx) = 0; % Middle region
    p(w+l+dl:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
    p(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3

    p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
        w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
    p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
        end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
    p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
    p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3

    % Finite difference solution
    for i = 2:nx-1
        for j = 2:ny-1
            p_next(i,j) = 2*p(i,j) - p_prev(i,j) + CFL^2 * ( p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
        end
    end

    k = abs(p);
    % Rigid Boundaries
    k(1:end, [1 end]) = -1; % Upper and lower boundaries
    %p([1 end], 1:end) = 0; % Left and right boundaries
    k(1:w+l, w+dx:end-w-2*dx) = -1; % Middle region
    k(w+l+dl:end,[1:w w+d+dx:end]) = -1; % Right bottom and top regions
    k(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = -1; k(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = -1; % 1/3

    k([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
        w-w3-1*dx-w2_upper:w-w3-2*dx]) = -1; % 2/3
    k([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
        end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = -1; % 2/3
    k([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = -1; % 3/3
    k([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
        [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = -1; % 3/3

    if round(mod(t/dt,25)) == 1
        output = mean(mean(k(end-d/2:end,w+dx:w+d))); %max(max(k(w+l+dl:end,w+dx:w+d)));
        output_pres = [output_pres output];
        time = [time t];
    end
    if round(mod(t/dt,25)) == 1
        % Visualize
        % COLORBAR
        indexValue = 0.4;     % value for which to set a particular color
        topColor = [1 0 0];         % color for maximum data value (red = [1 0 0])
        indexColor = [1 1 1];       % color for indexed data value (white = [1 1 1])
```

```
            bottomcolor = [0 0 1];      % color for minimum data value (blue = [0 0 1])
            largest = max(max(k));
            smallest = min(min(k));
            L = size(p,1);
            index = L*abs(indexValue-smallest)/(largest-smallest);
            customCMap1 = [linspace(bottomcolor(1),indexColor(1),100*index)',...
                        linspace(bottomcolor(2),indexColor(2),100*index)',...
                        linspace(bottomcolor(3),indexColor(3),100*index)'];
            customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index))',...
                        linspace(indexColor(2),topColor(2),100*(L-index))',...
                        linspace(indexColor(3),topColor(3),100*(L-index))'];
            customCMap = [customCMap1;customCMap2];  % Combine colormaps
            colormap(customCMap)

            imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
            colorbar; axis equal;
            grid off; box off; axis off
            ax = gca;
            ax.YDir = 'normal';
            tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output_pres)));
            txt = "t = " + tstring + "; P_{out} = " + pstring + "; P_{avg} = " + astring;
            title(txt); %title(sprintf('t = %.6f',t));
            drawnow limitrate
            frame = getframe(f);
            im = frame2im(frame);
            [imind,cm] = rgb2ind(im,256);
            if t == dt
                imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
            else
                imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
            end
        end
    end
    figure(2); plot(time,output_pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
    out = output_pres;
    close all
end

function out = solver11_XOR(lambda, filename, freq, filename2)

    % Geometric constraints
    l = lambda / 2;
    w = lambda / 10;
    d = 0.1 * lambda;
    d1 = 0.055 * lambda;

    l1 = (0.03 * l);
    l2 = (0.23 * l);
    w1_upper = (0.69 * w); % changing
    w1_lower = (0.22 * w); % changing
    w3 = (0.05 * w);
    w2_upper = ((w - w1_upper - 4 * w3) / 2);
    w2_lower = ((w - w1_lower - 4 * w3) / 2);
    w4 = ((l-4*l2)/5);
    ld2 = ((l-4*l1)/8);
    ld3 = ((l2-l1)/2);

    Lx = w + l + d1 + d;
    Ly = 2*w + d;
    dx = 1; dy = 1;
    nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;

    x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);

    % Field definition

    p = 0*ones(nx,ny);
    p_prev = p; p_next = p;

    CFL = 0.7; % c * dt/dx
    c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
    dt = CFL*dx/c;
    output_pres = [];
    time = [];

    % Initial conditions
    T = 20000*dt; % Runtime

    t = 0;
    f = figure(1);
    set(gcf,'Position',[100 100 1400 500])

    while t<T
        % Absorbing Boundaries
        p_next(1,:) = p(2,:) + ((CFL-1)/(CFL+1))*(p_next(2,:)-p(1,:));
        p_next(end,:) = p(end-1,:) + ((CFL-1)/(CFL+1))*(p_next(end-1,:)-p(end,:));

        % Rigid Boundaries
        p(1:end, [1 end]) = 0; % Upper and lower boundaries
        %p([1 end], 1:end) = 0; % Left and right boundaries
        p(1:w+l, w+dx:end-w-2*dx) = 0; % Middle region
        p(w+l+d1:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
        p(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3

        p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
              w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
        p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
              end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
        p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
        p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3

        t = t+dt;
        p_prev = p; p = p_next;

        %p_next(1:end,1) = p(1:end,2) + ((CFL-1)/(CFL+1))*(p_next(1:end,2)-p(1:end,1));
        p_next(1:end,end) = p(1:end,end-1) + ((CFL-1)/(CFL+1))*(p_next(1:end,end-1)-p(1:end,end));

        % Source 1
        p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);
```

```
588
589         % Source 2
590         p(2, 2:lambda/10-1) = 1*sin(2*pi*t*freq);
591
592         % Rigid Boundaries
593         p(1:end, [1 end]) = 0; % Upper and lower boundaries
594         %p([1 end], 1:end) = 0; % Left and right boundaries
595         p(1:w+1, w+dx:end-w-2*dx) = 0; % Middle region
596         p(w+1+d1:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
597         p(w:w+1, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
598
599         p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2_upper ...
                w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
600         p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2_lower:end-w3-dx ...
                end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
601         p([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
                [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
602         p([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
                [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3
603
604         % Finite difference solution
605         for i = 2:nx-1
606             for j = 2:ny-1
607                 p_next(i,j) = 2*p(i,j) - p_prev(i,j) + CFL^2 * ( p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
608             end
609         end
610
611         k = abs(p);
612         % Rigid Boundaries
613         k(1:end, [1 end]) = -1; % Upper and lower boundaries
614         %p([1 end], 1:end) = 0; % Left and right boundaries
615         k(1:w+1, w+dx:end-w-2*dx) = -1; % Middle region
616         k(w+1+d1:end,[1:w w+d+dx:end]) = -1; % Right bottom and top regions
617         k(w:w+1, [1+dx:w3+dx w-w3-dx:w]) = -1; k(w:w+1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = -1; % 1/3
618
619         k([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2_upper ...
                w-w3-1*dx-w2_upper:w-w3-2*dx]) = -1; % 2/3
620         k([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2_lower:end-w3-dx ...
                end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = -1; % 2/3
621         k([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
                [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = -1; % 3/3
622         k([w:w+w4+1d3 w+w4+12-1d3:w+2*w4+12+1d3 w+2*w4+2*12-1d3:w+3*w4+2*12+1d3 w+3*w4+3*12-1d3:w+4*w4+3*12+1d3 w+4*w4+4*12-1d3:w+5*w4+4*12], ...
                [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = -1; % 3/3
623
624         if round(mod(t/dt,25)) == 1
625             output = mean(mean(k(end-d/2:end,w+dx:w+d))); %max(max(k(w+1+d1:end,w+dx:w+d)));
626             output_pres = [output_pres output];
627             time = [time t];
628         end
629         if round(mod(t/dt,25)) == 1
630             % Visualize
631             % COLORBAR
632             indexValue = 0.4;     % value for which to set a particular color
633             topColor = [1 0 0];         % color for maximum data value (red = [1 0 0])
634             indexColor = [1 1 1];       % color for indexed data value (white = [1 1 1])
635             bottomcolor = [0 0 1];      % color for minimum data value (blue = [0 0 1])
636             largest = max(max(k));
637             smallest = min(min(k));
638             L = size(p,1);
639             index = L*abs(indexValue-smallest)/(largest-smallest);
640             customCMap1 = [linspace(bottomcolor(1),indexColor(1),100*index)',...
641                         linspace(bottomcolor(2),indexColor(2),100*index)',...
642                         linspace(bottomcolor(3),indexColor(3),100*index)'];
643             customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index))',...
644                         linspace(indexColor(2),topColor(2),100*(L-index))',...
645                         linspace(indexColor(3),topColor(3),100*(L-index))'];
646             customCMap = [customCMap1;customCMap2];  % Combine colormaps
647             colormap(customCMap)
648
649             imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
650             colorbar; axis equal;
651             grid off; box off; axis off
652             ax = gca;
653             ax.YDir = 'normal';
654             tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output_pres)));
655             txt = "t = " + tstring + "; P_{out} = " + pstring + "; P_{avg} = " + astring;
656             title(txt); %title(sprintf('t = %.6f',t));
657             drawnow limitrate
658             frame = getframe(f);
659             im = frame2im(frame);
660             [imind,cm] = rgb2ind(im,256);
661             if t == dt
662                 imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
663             else
664                 imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
665             end
666         end
667     end
668     figure(2); plot(time,output_pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
669     out = output_pres;
670     close all
671 end
672
673 function out = solver10_XOR(lambda, filename, freq, filename2)
674
675     % Geometric constraints
676     l = lambda / 2;
677     w = lambda / 10;
678     d = 0.1 * lambda;
679     d1 = 0.055 * lambda;
680
681     l1 = (0.03 * l);
682     l2 = (0.23 * l);
683     w1_upper = (0.69 * w); % changing
684     w1_lower = (0.22 * w); % changing
685     w3 = (0.05 * w);
686     w2_upper = ((w - w1_upper - 4 * w3) / 2);
687     w2_lower = ((w - w1_lower - 4 * w3) / 2);
688     w4 = ((l-4*12)/5);
689     ld2 = ((l-4*l1)/8);
690     ld3 = ((l2-l1)/2);
```

```
691
692       Lx = w + l + dl + d;
693       Ly = 2*w + d;
694       dx = 1; dy = 1;
695       nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;
696
697       x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);
698
699       % Field definition
700
701       p = 0*ones(nx,ny);
702       p_prev = p; p_next = p;
703
704       CFL = 0.7; % c * dt/dx
705       c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
706       dt = CFL*dx/c;
707       output_pres = [];
708       time = [];
709
710       % Initial conditions
711       T = 20000*dt; % Runtime
712
713       t = 0;
714       f = figure(1);
715       set(gcf,'Position',[100 100 1400 500])
716
717       while t<T
718           % Absorbing Boundaries
719           p_next(1,:) = p(2,:) + ((CFL-1)/(CFL+1))*(p_next(2,:)-p(1,:));
720           p_next(end,:) = p(end-1,:) + ((CFL-1)/(CFL+1))*(p_next(end-1,:)-p(end,:));
721
722           % Rigid Boundaries
723           p(1:end, [1 end]) = 0; % Upper and lower boundaries
724           %p([1 end], 1:end) = 0; % Left and right boundaries
725           p(1:w+l, w+dx:end-w-2*dx) = 0; % Middle region
726           p(w+l+dl:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
727           p(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
728
729           p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2_upper ...
                  w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
730           p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2_lower:end-w3-dx ...
                 end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
731           p([w:w+w4+ld3 w+w4+12-ld3:w+2*w4+12+ld3 w+2*w4+2*12-ld3:w+3*w4+2*12+ld3 w+3*w4+3*12-ld3:w+4*w4+3*12+ld3 w+4*w4+4*12-ld3:w+5*w4+4*12], ...
                 [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
732           p([w:w+w4+ld3 w+w4+12-ld3:w+2*w4+12+ld3 w+2*w4+2*12-ld3:w+3*w4+2*12+ld3 w+3*w4+3*12-ld3:w+4*w4+3*12+ld3 w+4*w4+4*12-ld3:w+5*w4+4*12], ...
                 [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3
733
734           t = t+dt;
735           p_prev = p; p = p_next;
736
737           %p_next(1:end,1) = p(1:end,2) + ((CFL-1)/(CFL+1))*(p_next(1:end,2)-p(1:end,1));
738           p_next(1:end,end) = p(1:end,end-1) + ((CFL-1)/(CFL+1))*(p_next(1:end,end-1)-p(1:end,end));
739
740           % Source 1
741           p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);
742
743           % Source 2
744           %p(2, 2:lambda/10-1) = 1*sin(2*pi*t*freq);
745
746           % Rigid Boundaries
747           p(1:end, [1 end]) = 0; % Upper and lower boundaries
748           %p([1 end], 1:end) = 0; % Left and right boundaries
749           p(1:w+l, w+dx:end-w-2*dx) = 0; % Middle region
750           p(w+l+dl:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
751           p(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3
752
753           p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2_upper ...
                  w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
754           p([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2_lower:end-w3-dx ...
                 end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
755           p([w:w+w4+ld3 w+w4+12-ld3:w+2*w4+12+ld3 w+2*w4+2*12-ld3:w+3*w4+2*12+ld3 w+3*w4+3*12-ld3:w+4*w4+3*12+ld3 w+4*w4+4*12-ld3:w+5*w4+4*12], ...
                 [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
756           p([w:w+w4+ld3 w+w4+12-ld3:w+2*w4+12+ld3 w+2*w4+2*12-ld3:w+3*w4+2*12+ld3 w+3*w4+3*12-ld3:w+4*w4+3*12+ld3 w+4*w4+4*12-ld3:w+5*w4+4*12], ...
                 [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3
757
758           % Finite difference solution
759           for i = 2:nx-1
760               for j = 2:ny-1
761                   p_next(i,j) = 2*p(i,j) - p_prev(i,j) + CFL^2 * ( p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
762               end
763           end
764
765           k = abs(p);
766           % Rigid Boundaries
767           k(1:end, [1 end]) = -1; % Upper and lower boundaries
768           %p([1 end], 1:end) = 0; % Left and right boundaries
769           k(1:w+l, w+dx:end-w-2*dx) = -1; % Middle region
770           k(w+l+dl:end,[1:w w+d+dx:end]) = -1; % Right bottom and top regions
771           k(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = -1; k(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = -1; % 1/3
772
773           k([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [w3+2*dx:w3+dx+w2_upper ...
                  w-w3-1*dx-w2_upper:w-w3-2*dx]) = -1; % 2/3
774           k([w:w+w4 w+w4+12:w+2*w4+12 w+2*w4+2*12:w+3*w4+2*12 w+3*w4+3*12:w+4*w4+3*12 w+4*w4+4*12:w+5*w4+4*12], [end-w3-w2_lower:end-w3-dx ...
                  end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = -1; % 2/3
775           k([w:w+w4+ld3 w+w4+12-ld3:w+2*w4+12+ld3 w+2*w4+2*12-ld3:w+3*w4+2*12+ld3 w+3*w4+3*12-ld3:w+4*w4+3*12+ld3 w+4*w4+4*12-ld3:w+5*w4+4*12], ...
                  [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = -1; % 3/3
776           k([w:w+w4+ld3 w+w4+12-ld3:w+2*w4+12+ld3 w+2*w4+2*12-ld3:w+3*w4+2*12+ld3 w+3*w4+3*12-ld3:w+4*w4+3*12+ld3 w+4*w4+4*12-ld3:w+5*w4+4*12], ...
                  [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = -1; % 3/3
777
778           if round(mod(t/dt,25)) == 1
779               output = mean(mean(k(end-d/2:end,w+dx:w+d))); %max(max(k(w+l+dl:end,w+dx:w+d)));
780               output_pres = [output_pres output];
781               time = [time t];
782           end
783           if round(mod(t/dt,25)) == 1
784               % Visualize
785               % COLORBAR
786               indexValue = 0.4;     % value for which to set a particular color
787               topColor = [1 0 0];         % color for maximum data value (red = [1 0 0])
788               indexColor = [1 1 1];       % color for indexed data value (white = [1 1 1])
789               bottomcolor = [0 0 1];      % color for minimum data value (blue = [0 0 1])
```

```
            largest = max(max(k));
            smallest = min(min(k));
            L = size(p,1);
            index = L*abs(indexValue-smallest)/(largest-smallest);
            customCMap1 = [linspace(bottomcolor(1),indexColor(1),100*index)',...
                        linspace(bottomcolor(2),indexColor(2),100*index)',...
                        linspace(bottomcolor(3),indexColor(3),100*index)'];
            customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index))',...
                        linspace(indexColor(2),topColor(2),100*(L-index))',...
                        linspace(indexColor(3),topColor(3),100*(L-index))'];
            customCMap = [customCMap1;customCMap2];  % Combine colormaps
            colormap(customCMap)

            imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
            colorbar; axis equal;
            grid off; box off; axis off
            ax = gca;
            ax.YDir = 'normal';
            tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output_pres)));
            txt = "t = " + tstring + "; P_{out} = " + pstring + "; P_{avg} = " + astring;
            title(txt); %title(sprintf('t = %.6f',t));
            drawnow limitrate
            frame = getframe(f);
            im = frame2im(frame);
            [imind,cm] = rgb2ind(im,256);
            if t == dt
                imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
            else
                imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
            end
        end
    end
    figure(2); plot(time,output_pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
    out = output_pres;
    close all
end

function out = solver01_XOR(lambda, filename, freq, filename2)

    % Geometric constraints
    l = lambda / 2;
    w = lambda / 10;
    d = 0.1 * lambda;
    d1 = 0.055 * lambda;

    l1 = (0.03 * l);
    l2 = (0.23 * l);
    w1_upper = (0.69 * w); % changing
    w1_lower = (0.22 * w); % changing
    w3 = (0.05 * w);
    w2_upper = ((w - w1_upper - 4 * w3) / 2);
    w2_lower = ((w - w1_lower - 4 * w3) / 2);
    w4 = ((l-4*l2)/5);
    ld2 = ((l-4*l1)/8);
    ld3 = ((l2-l1)/2);

    Lx = w + l + d1 + d;
    Ly = 2*w + d;
    dx = 1; dy = 1;
    nx = fix(Lx/dx) + 2; ny = fix(Ly/dy) + 2;

    x = linspace(0, Lx, nx); y = linspace(0, Ly, ny);

    % Field definition

    p = 0*ones(nx,ny);
    p_prev = p; p_next = p;

    CFL = 0.7; % c * dt/dx
    c = 1; %c/lambda; freq1 = c/lambda; freq2 = c/lambda;
    dt = CFL*dx/c;
    output_pres = [];
    time = [];

    % Initial conditions
    T = 20000*dt; % Runtime

    t = 0;
    f = figure(1);
    set(gcf,'Position',[100 100 1400 500])

    while t<T
        % Absorbing Boundaries
        p_next(1,:) = p(2,:) + ((CFL-1)/(CFL+1))*(p_next(2,:)-p(1,:));
        p_next(end,:) = p(end-1,:) + ((CFL-1)/(CFL+1))*(p_next(end-1,:)-p(end,:));

        % Rigid Boundaries
        p(1:end, [1 end]) = 0; % Upper and lower boundaries
        %p([1 end], 1:end) = 0; % Left and right boundaries
        p(1:w+1, w+dx:end-w-2*dx) = 0; % Middle region
        p(w+1+d1:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
        p(w:w+1, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+1, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3

        p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
               w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
        p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
                end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
        p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
               [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
        p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
               [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3

        t = t+dt;
        p_prev = p; p = p_next;

        %p_next(1:end,1) = p(1:end,2) + ((CFL-1)/(CFL+1))*(p_next(1:end,2)-p(1:end,1));
        p_next(1:end,end) = p(1:end,end-1) + ((CFL-1)/(CFL+1))*(p_next(1:end,end-1)-p(1:end,end));

        % Source 1
        %p(2, end-lambda/10:end-1) = 1*sin(2*pi*t*freq);

```

```
        % Source 2
        p(2, 2:lambda/l0-1) = l*sin(2*pi*t*freq);

        % Rigid Boundaries
        p(1:end, [1 end]) = 0; % Upper and lower boundaries
        %p([1 end], 1:end) = 0; % Left and right boundaries
        p(1:w+l, w+dx:end-w-2*dx) = 0; % Middle region
        p(w+l+dl:end,[1:w w+d+dx:end]) = 0; % Right bottom and top regions
        p(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = 0; p(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = 0; % 1/3

        p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
              w-w3-1*dx-w2_upper:w-w3-2*dx]) = 0; % 2/3
        p([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
              end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = 0; % 2/3
        p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = 0; % 3/3
        p([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = 0; % 3/3

        % Finite difference solution
        for i = 2:nx-1
            for j = 2:ny-1
                p_next(i,j) = 2*p(i,j) - p_prev(i,j) + CFL^2 * ( p(i+1,j) + p(i,j+1) - 4*p(i,j) + p(i-1,j) + p(i,j-1));
            end
        end

        k = abs(p);
        % Rigid Boundaries
        k(1:end, [1 end]) = -1; % Upper and lower boundaries
        %p([1 end], 1:end) = 0; % Left and right boundaries
        k(1:w+l, w+dx:end-w-2*dx) = -1; % Middle region
        k(w+l+dl:end,[1:w w+d+dx:end]) = -1; % Right bottom and top regions
        k(w:w+l, [1+dx:w3+dx w-w3-dx:w]) = -1; k(w:w+l, [end-w3:end-dx end-w-dx:end-w+w3-dx]) = -1; % 1/3

        k([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [w3+2*dx:w3+dx+w2_upper ...
              w-w3-1*dx-w2_upper:w-w3-2*dx]) = -1; % 2/3
        k([w:w+w4 w+w4+l2:w+2*w4+l2 w+2*w4+2*l2:w+3*w4+2*l2 w+3*w4+3*l2:w+4*w4+3*l2 w+4*w4+4*l2:w+5*w4+4*l2], [end-w3-w2_lower:end-w3-dx ...
              end-w+w3+0*dx:end-w+w3+w2_lower+1*dx]) = -1; % 2/3
        k([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [2*dx+w2_upper+w3:w3+2*dx+w2_upper+w3 w-w3-w2_upper-w3-2*dx:w-w2_upper-w3-2*dx]) = -1; % 3/3
        k([w:w+w4+ld3 w+w4+l2-ld3:w+2*w4+l2+ld3 w+2*w4+2*l2-ld3:w+3*w4+2*l2+ld3 w+3*w4+3*l2-ld3:w+4*w4+3*l2+ld3 w+4*w4+4*l2-ld3:w+5*w4+4*l2], ...
              [end-w3-w2_lower-w3-1*dx:end-w2_lower-w3-1*dx end-w+2*dx+w2_lower+w3:end-w+w3+2*dx+w2_lower+w3]) = -1; % 3/3

        if round(mod(t/dt,25)) == 1
            output = mean(mean(k(end-d/2:end,w+dx:w+d))); %max(max(k(w+l+dl:end,w+dx:w+d)));
            output_pres = [output_pres output];
            time = [time t];
        end
        if round(mod(t/dt,25)) == 1
            % Visualize
            % COLORBAR
            indexValue = 0.4;     % value for which to set a particular color
            topColor = [1 0 0];         % color for maximum data value (red = [1 0 0])
            indexColor = [1 1 1];       % color for indexed data value (white = [1 1 1])
            bottomcolor = [0 0 1];      % color for minimum data value (blue = [0 0 1])
            largest = max(max(k));
            smallest = min(min(k));
            L = size(p,1);
            index = L*abs(indexValue-smallest)/(largest-smallest);
            customCMap1 = [linspace(bottomcolor(1),indexColor(1),100*index)',...
                        linspace(bottomcolor(2),indexColor(2),100*index)',...
                        linspace(bottomcolor(3),indexColor(3),100*index)'];
            customCMap2 = [linspace(indexColor(1),topColor(1),100*(L-index))',...
                        linspace(indexColor(2),topColor(2),100*(L-index))',...
                        linspace(indexColor(3),topColor(3),100*(L-index))'];
            customCMap = [customCMap1;customCMap2];  % Combine colormaps
            colormap(customCMap)

            imagesc(x,y,k'); colorbar; caxis([-1 max(max(abs(k)))]); % caxis([-1 1]);
            colorbar; axis equal;
            grid off; box off; axis off
            ax = gca;
            ax.YDir = 'normal';
            tstring = num2str(t); pstring = num2str(output); astring = num2str(mean(nonzeros(output_pres)));
            txt = "t = " + tstring + "; P_{out} = " + pstring + "; P_{avg} = " + astring;
            title(txt); %title(sprintf('t = %.6f',t));
            drawnow limitrate
            frame = getframe(f);
            im = frame2im(frame);
            [imind,cm] = rgb2ind(im,256);
            if t == dt
                imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'Loopcount',inf);
            else
                imwrite(imind,cm,filename,'gif','DelayTime',0.01, 'WriteMode','append');
            end
        end
    end
    figure(2); plot(time,output_pres); xlabel("Time"); ylabel("Output Pressure"); saveas(gcf,filename2)
    out = output_pres;
    close all
end
```