Modelling acoustic wave propagation in an axially symmetric system with a finite-difference time-domain method

Erlend Magnus Viggen

November 12, 2008

Abstract

A finite-difference time-domain method for modelling acoustic wave propagation in an axially symmetric system is described, and the results of an implementation of this method are shown. The model is shown to be reasonably accurate for high frequencies with resolution settings that give a running time of roughly a minute, and increasingly accurate with increasing resolution. The numerical errors of the model are looked at, as well as what algorithms take up its running time. The model is used to dismiss possible changes in the nature of acoustic propagation in air due to the shape of the system.

I Introduction

This text shows the details behind a model of acoustic wave propagation in an axially symmetric system, using a finite-difference time-domain method. This model has applications in studying acoustic propagation in systems such as horns and circular ducts. The model can also answer several general questions about propagation in axially symmetric systems. For one, if a horn has a very sharp curve, do the wavefronts still always propagate in a direction parallel to the wall of the horn? And do systems such as this cause dispersion in an otherwise non-dispersive medium such as air?

It is also useful to look at the precision of the model compared to how long a simulation using it takes. Analyzing which parts in it take the most computational time might aid in optimizing the model.

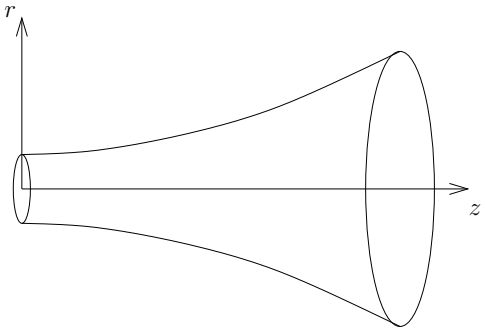

Figure 1: A system that is axially symmetric about the z axis, here represented by a horn. Wave propagation is studied inside the horn.

II Finite difference method

When using finite difference methods, the space of the system is divided into a grid of nodes that are equally spaced along each dimension. Each node holds the state of the system at that point. In this problem, there are two physical dimensions, r and z, as seen in figure 1, as well as one dimension of time. With this grid, it is possible to estimate the value of a differential at one point using the values of this point and the neighboring points. For instance, the first and second derivatives of the parameter $\phi(r)$ at node i in the grid are [2]

$$\frac{d\phi_i}{dr} \approx \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta r}, \tag{1}$$

$$\frac{d^2\phi_i}{dr^2} \approx \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{(\Delta r)^2}. \tag{2}$$

Here Δr is the spacing between nodes on the r axis.

Acoustic wave propagation is governed by the wave equation [1],

$$\nabla^2 p = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}, \tag{3}$$

where $p = p(r, z, t)$ is the axially symmetric sound pressure and ∇^2 is the Laplace operator, which in cylindrical coordinates is [1]

$$\nabla^2 = \frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2} + \frac{\partial^2}{\partial z^2}. \tag{4}$$

When utilized on a axially symmetric function, $\partial^2/\partial\theta^2$ gives zero, and the Laplace operator simplifies to

$$\nabla^2 = \frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{\partial^2}{\partial z^2}. \tag{5}$$

When equation 5 is put into equation 3, it becomes

$$\left(\frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{\partial^2}{\partial z^2}\right) p = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}. \tag{6}$$

Each derivative in this equation can be approximated using a finite difference approach. With the notation used, i is an r coordinate, j a z coordinate and k a t "coordinate", so that for instance $r = i\Delta r$. This system of spatial coordinate notation is also shown in figure 2. With this notation, the terms then become:

$$\frac{\partial^2}{\partial r^2} p_i \approx \frac{p_{i+1} - 2p_i + p_{i-1}}{(\Delta r)^2} = I_{r^2}, \tag{7}$$

$$\frac{1}{r}\frac{\partial}{\partial r} p_i \approx \frac{1}{r}\frac{p_{i+1} - p_{i-1}}{2\Delta r} = I_r, \tag{8}$$

$$\frac{\partial^2}{\partial z^2} p_j \approx \frac{p_{j+1} - 2p_i + p_{j-1}}{(\Delta z)^2} = I_{r^2}, \tag{9}$$

$$\frac{1}{c^2}\frac{\partial^2}{\partial t^2} p_k \approx \frac{p_{k+1} - 2p_k + p_{k-1}}{(\Delta t)^2} = I_{t^2}, \tag{10}$$

Since $I_{t^2} = I_{r^2} + I_r + I_{z^2}$,

$$\frac{p_{k+1} - 2p_k + p_{k-1}}{c^2 (\Delta t)^2} = I_{r^2} + I_r + I_{z^2}. \tag{11}$$

After some calculation, this results in the **unbounded update function**:

$$\boxed{\begin{aligned} p_{k+1} = p(&-2\alpha + 2\gamma + 2) \\ &+ p_{i+1}(\alpha + \beta) \\ &+ p_{i-1}(\alpha - \beta) \\ &+ p_{j+1}(\gamma) \\ &+ p_{j-1}(\gamma) \\ &+ p_{k-1}. \end{aligned}} \tag{12}$$

For clarity and to save space, only the coordinates that are different from i, j and k have been explicitly written. Where no other coordinates are specified, i, j and k are implicit, so that $p_{i+1} = p_{i+1,j,k}$. Also, three simplifying quantities have been introduced:

$$\alpha = \frac{c^2 (\Delta t)^2}{(\Delta r)^2}, \tag{13}$$

$$\beta = \frac{1}{r}\frac{c^2 (\Delta t)^2}{2\Delta r}, \tag{14}$$

$$\gamma = \frac{c^2 (\Delta t)^2}{(\Delta z)^2}. \tag{15}$$

This update function is used to update most of the nodes in the system from their state at time k to their state at time $k+1$. For the nodes lying at boundaries (i.e. by a wall or on the z axis), special update functions are needed.

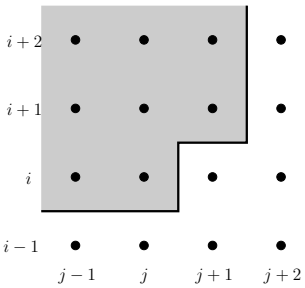

Figure 2: A boundary exists between nodes in the wall (the gray area) and nodes in air (the white area).

III Boundary conditions

For a node that is on the axis, i.e. $r = 0$ and $i = 0$, L'Hôpital's rule must be used on the I_r term, reducing it to I_{r^2}. Also, since the node p_0 is on the z axis and the system is axially symmetric, $p_{-1} = p_1$. This gives the **down-bounded update function**,

$$\boxed{\begin{aligned} p_{k+1} = p(&-4\alpha + 2\gamma + 2) \\ &+ p_{i+1}(4\alpha) \\ &+ p_{j+1}(\gamma) \\ &+ p_{j-1}(\gamma) \\ &+ p_{k-1}. \end{aligned}} \tag{16}$$

Some nodes are inside a wall, as shown in figure 2. These are not to be updated, and are only present because it is simpler to handle a square grid of nodes. When a node inside the system itself is beside a wall, the update functions in equations 12 and 16 must be altered.

The derivative $\partial p/\partial x$ equals zero at a rigid boundary perpendicular to the x direction [1]. Therefore, if p_i is a node beside a wall, and its neighbor p_{i+1} is inside the wall,

$$\frac{p_{i+1} - p_i}{\Delta r} = 0, \tag{17}$$

meaning that

$$p_{i+1} = p_i. \tag{18}$$

With this known, the update functions for all other boundings can be calculated. These can be found in appendix A.

IV Implementation

An efficient way to time-step through the system's evolution is by sparse matrix multiplication

in MATLAB. The pressure of every node in the system for the current and previous time steps are stored in a sparse vector, and a sparse matrix is used to perform every node's update function for each step. The matrix also copies the pressures for the current time step in $[p]_k$ to the pressures for the previous time step in $[p]_{k+1}$. If $0 \leq i \leq I$ and $0 \leq j \leq J$, the pressures $p_{i,j,k}$ in the vector $[p]_k$ are organized like shown below:

$$[p]_k = \begin{bmatrix} p_{0,0,k} \\ p_{1,0,k} \\ \vdots \\ p_{I,0,k} \\ p_{0,1,k} \\ \vdots \\ p_{I,J,k} \\ p_{0,0,k-1} \\ \vdots \\ p_{I,J,k-1} \end{bmatrix}_k \tag{19}$$

A matrix $[A]$ is created that performs time-stepping in the system, so that

$$[p]_{k+1} = [A]\,[p]_k\,. \tag{20}$$

The matrix is constructed according to the update functions given in sections II, III and appendix A. For further details on how $[A]$ is constructed, see appendix B.

The nodes in the system at $z = 0$ are the "prime movers" of the system. They are not subject to the update functions in A, but are instead locked to some time-varying function, for instance a sine function at a certain frequency, or a Mexican hat wavelet.

The Mexican hat wavelet bears special mention as it is not very widely used. It is given by [3]

$$\psi(x) \propto \left(1 - x^2\right) \mathrm{e}^{-x^2/2}, \tag{21}$$

where $x \propto t$. The shape of the function can be seen in figure 3.

The Mexican hat wavelet does not have a frequency and period like a sine function, but a "period" $T = 1/f$ can be defined as the time between the two minima at each side. From $\psi'(x) = 0$ it can be found that these minima are at $x = -\sqrt{3}$ and $x = \sqrt{3}$. When x is defined as $x = Cft$ where f is frequency and C a constant,

$$\Delta x = CfT = C = 2\sqrt{3}, \tag{22}$$

meaning that

$$x = 2\sqrt{3}ft. \tag{23}$$

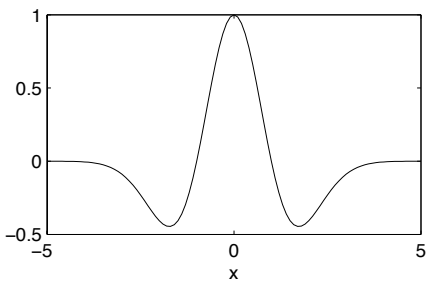

Figure 3: An unscaled Mexican hat wavelet.

This is how time and frequency affect the wavelet signal in the code given in section B. When starting the wavelet, x starts at $-3\sqrt{3}$ to get a reasonably smooth start for the system.

V Results

The images of the results will be as shown in figure 4. Every square in the image represents a node. A light color represents positive pressure while a dark color represents negative pressure. The large black area in the upper left of most images represents the nodes that are in the wall. These have been coloured black to make the boundary easily visible. It also bears mention that the images are plotted in a way so that the lightness for a point is not proportional to its p value, but instead $\mathrm{sgn}(p)\sqrt{p}$. This is done to exaggerate the pressures so that waves far away from the source also are clearly visible.

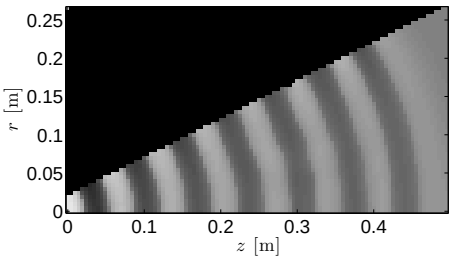

Figure 4: Sine wave propagation in a horn with linearly increasing radius. The curve is proportional to z.

All simulations have been performed with a system length $L = 0.5$ m, a signal frequency $f = 5000$ Hz and a sound propagation velocity $c = 343$ m/s. All images are of the system's state after the time it takes for sound to propagate from one end of the system to the other. This is about 1.5 ms.

The reason for this is that the end of the system is also a hard wall, as open edges are difficult to simulate.

Unless anything else is specified, the simulation results are given for the resolutions $\Delta r = 0.005$ m, $\Delta z = 0.005$ m and $\Delta t = 10^{-7}$ s.

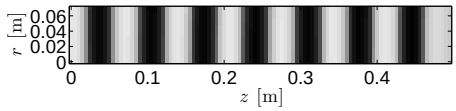

Figure 5: Sine wave propagation in a system with a constant circular cross-section.

Figure 5 shows sine wave propagation in a circular duct of constant radius. It shows a wave that propagates exactly according to the k_{00} waveguide mode as given in *Fundamentals of Acoustics* [1].

From figures 4 and 5, it is also clear that the wavelength λ is slightly in excess of 6.7 cm, which corresponds well with the analytical value for a frequency of $f = 5000$ Hz, $\lambda = 6.86$ cm.

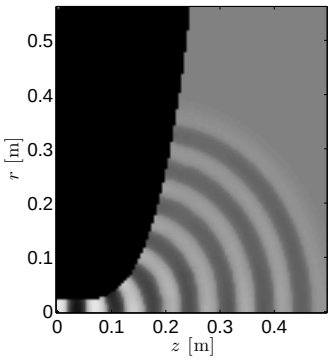

Figure 6: Sine wave propagation in a horn with a sharply exponential curve. The curve is proportional to z^4.

Figure 6 shows sine wave propagation in a system where the radius is sharply increasing after a point. The circular wavefronts in the figure are not entirely concentric, as can be clearly seen by the inner circles having a centre further to the left than the outer circles. Each circle's propagation direction by the wall is still parallel to the wall, even in this unconventional geometry. This can be seen from the wavefronts being perpendicular to the wall.

With the wavelet signal type, the limitations of the finite difference model for relatively large Δr and Δz is starting to show. Figure 7 shows a lot

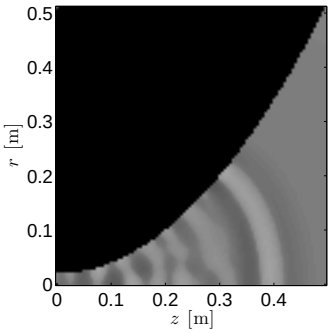

Figure 7: Wavelet propagation in a horn with a smooth exponential curve. The curve is proportional to z^2.

of disturbance in the wake of the wavelet. This disturbance occurs when the peak of the wavelet passes around 0.05 m, where the radius widens by Δr, which is a quite significant increase when the spatial resolution is that low.

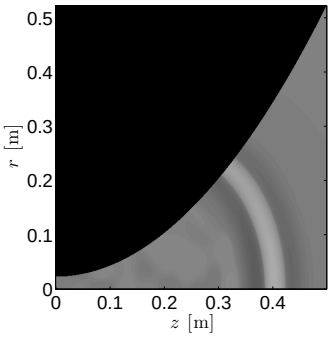

Figure 8: The same simulation as in figure 7, but with $\Delta r = \Delta z = 0.001$ m instead of 0.005 m.

The model is significantly improved by improving the spatial resolutions Δr and Δz to 0.001 m, as seen in figure 8. The "numerical turbulence" is strongly decreased. The downside is that the simulation takes far longer, since the size of $[p]$ and $[A]$ has increased greatly. A middle road can be taken by setting $\Delta r = 0.001$ m and $\Delta z = 0.005$ m, but the problem isn't lessened to the degree shown in figure 8.

That it is possible to decrease these effects by improving the spatial resolution indicates that the problem is a purely numerical one, and that there should be very little actual turbulence in the wake of the wavelet. The reason why this effect is not apparent for sine signal type is likely that the am-

plitude of the reflection is insignificant compared to the amplitude of the next part of the sine. As mentioned at the start of this section, the pressures are drawn in an exaggerated manner, so that the disturbances in figure 7 might not really be all that significant compared to the amplitude of the peak near the start of the system.

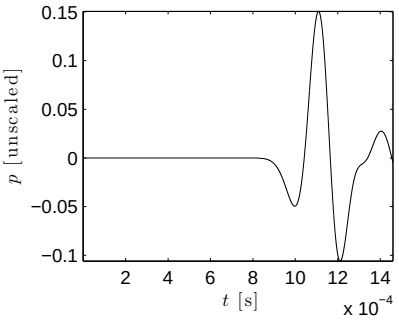

Figure 9: The pressure at the point $z = 0.25$ m, $r = 0.1$ m in the system in figure 7 as a function of time, with a wavelet signal.

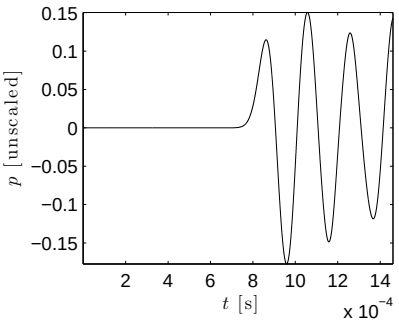

Figure 10: The pressure at the point $z = 0.25$ m, $r = 0.1$ m in the system in figure 7 as a function of time, with a sine signal.

Figure 9 shows that the general shape of the wavelet is intact further out in the system, even though there are some disturbances at the end of the signal due to the numerical turbulence. Figure 10 shows a similar disturbance tendency, but as the general sine shape is preserved, the effect looks weaker than it actually is in the system images. These results have been found with the standard resolutions of $\Delta r = \Delta z = 0.005$ m.

Figure 11 shows that an improved spatial resolution improves the shape of the recorded wavelet so that it more resembles the excitation signal shown in figure 3. The disturbance at the end is greatly reduced, and the two minima are closer in amplitude. The amplitudes are also larger than in figure

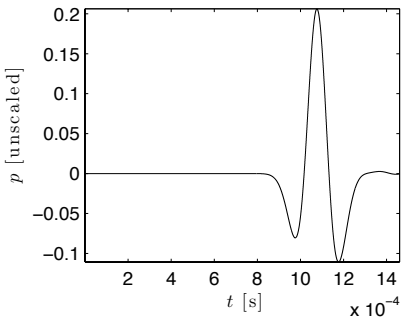

Figure 11: The same pressure as in figure 9 with an improved spatial resolution of $\Delta r = \Delta z = 0.001$ m.

9. This is likely due to the diminishing of the reflections discussed earlier in this section, allowing more of the signal energy to propagate directly out into the horn.

It is puzzling that the extremal points of the wavelet are affected in such a way that the first minimum is smaller than it should be compared to the first maximum and second minimum. This trend can also be seen in figure 10, where the first extremal point is small compared to the second and third. As this trend is also present for the monofrequency sine signal type, we can rule out that it stems from dispersion. That the amplitude difference is strongly diminished by improving the spatial resolution indicates that it stems instead from numerical errors.

The fact that the general shape of the wavelet is the same in figure 11 as in figure 3 and that the same errors are found for a monofrequency signal indicates that no dispersion is going on in the system, even close to the wall.

A final note must be made about the speed of the model. It takes 80 seconds to run a simulation as shown in figure 7 on a 2.2 GHz Intel Core 2 Duo processor (only one core is actually used) running MATLAB 7.4.0 in Mac OS X 10.5.5, with spatial resolution $\Delta r = \Delta z = 0.005$ m and time resolution $\Delta t = 10^{-7}$ s. The MATLAB profiler indicates that with a frameskip of 100 (one image of system state drawn per 100 steps), 63% of the CPU time was used on sparse matrix multiplication, and 26% on drawing.

With an improved spatial resolution of $\Delta r = \Delta z = 0.001$ m, running the same simulation takes as much as 34 minutes. The running time has increased roughly 25 times, which is the same increase as the resolution. From this, we see that running time has a roughly linear relationship to the resolution.

It might be possible to use a parallel algorithm for the matrix multiplication so that the full power of multi-core processors could be utilized. Unfortunately, parallellization is not well supported in MATLAB at this time.

VI Conclusion

It is quite possible to use finite-difference time-domain methods to model a 2D problem such as this, but the result is not as fast as other methods for modelling acoustic propagation, such as the TLM method. As it is, this implementation is too slow to get good accuracy on larger systems for high frequencies, but depending on the application, it might be good enough to use with low frequencies. Lower frequencies means longer wavelengths, which would likely allow a significantly lower resolution. If high-frequency simulations are required, a raytracing approach might be more appropriate instead. In addition, as the curved horn surface creates reflection problems with low spatial resolutions, the removal of curved surfaces would also improve results for low resolutions.

The large fraction of time spent on sparse matrix multiplication indicates that it is hard to tweak the code to run a lot faster, as the built-in algorithms for this in MATLAB are quite good apart from the fact that they only use one processor core.

The model been able to answer the two questions posed in the introduction. It has shown that wavefronts by the wall propagate parallel to the wall even in an unconventional horn geometry. In addition, no dispersion is introduced in the system by the horn shape, as can be seen from the wavelet shape in figure 11.

References

[1] Lawrence E. Kinsler, Austin R. Frey, Alan B. Coppens, and James V. Sanders, *Fundamentals of Acoustics*, 4th ed., John Wiley & Sons, 2000.

[2] Ulf R. Kristiansen, *Endelige Differansers Metode i Akustikk*, 2003, published as a course supplement.

[3] The MathWorks, *MATLAB Wavelet Toolbox documentation*, mexh function.

A Bounded update functions

For simplicity, a system of directions is used with the axes as shown in figure 1 and every result image in section V. The unbounded update function is found in equation 12, and the down-bounded update function is found in equation 16.

The **left-bounded update function**:

$$\begin{aligned} p_{k+1} = [&p(-2\alpha + 2\gamma + 2) \\ &+ p_{i+1}(\alpha + \beta) \\ &+ p_{i-1}(\alpha - \beta) \\ &+ p_{j+1}(\gamma) \\ &+ p_{k-1}]/(1 - \gamma) \end{aligned} \tag{24}$$

The **up-left-bounded update function**:

$$\begin{aligned} p_{k+1} = [&p(-2\alpha + 2\gamma + 2) \\ &+ p_{i-1}(\alpha - \beta) \\ &+ p_{j+1}(\gamma) \\ &+ p_{k-1}]/(1 - \alpha - \beta - \gamma) \end{aligned} \tag{25}$$

The **up-bounded update function**:

$$\begin{aligned} p_{k+1} = [&p(-2\alpha + 2\gamma + 2) \\ &+ p_{i-1}(\alpha - \beta) \\ &+ p_{j+1}(\gamma) \\ &+ p_{j-1}(\gamma) \\ &+ p_{k-1}]/(1 - \alpha - \beta) \end{aligned} \tag{26}$$

The **up-right-bounded update function**:

$$\begin{aligned} p_{k+1} = [&p(-2\alpha + 2\gamma + 2) \\ &+ p_{i-1}(\alpha - \beta) \\ &+ p_{j-1}(\gamma) \\ &+ p_{k-1}]/(1 - \alpha - \beta - \gamma) \end{aligned} \tag{27}$$

The **right-bounded update function**:

$$\begin{aligned} p_{k+1} = [&p(-2\alpha + 2\gamma + 2) \\ &+ p_{i+1}(\alpha + \beta) \\ &+ p_{i-1}(\alpha - \beta) \\ &+ p_{j-1}(\gamma) \\ &+ p_{k-1}]/(1 - \gamma) \end{aligned} \tag{28}$$

The **down-right-bounded update function**:

$$\begin{aligned} p_{k+1} = [&p(-4\alpha + 2\gamma + 2) \\ &+ p_{i+1}(4\alpha) \\ &+ p_{j-1}(\gamma) \\ &+ p_{k-1}]/(1 - \gamma) \end{aligned} \tag{29}$$

The **down-left-bounded update function**:

$$\begin{aligned} p_{k+1} = [&p(-4\alpha + 2\gamma + 2) \\ &+ p_{i+1}(4\alpha) \\ &+ p_{j+1}(\gamma) \\ &+ p_{k-1}]/(1 - \gamma) \end{aligned} \tag{30}$$

B MATLAB code

The MATLAB code that was used to implement the model and generate the figures in section V is given below. The MATLAB language was chosen for its built-in efficient representation of sparse vectors and matrices, its efficient algorithms for sparse matrix multiplication, its great facilities for plotting figures as well as its familiarity to the author.

```
clear all;

% User-set constants
dr = 0.005;                             % r resolution [m]
dz = 0.005;                             % z resolution [m]
dt = 1E-7;                              % t resolution [s]
c = 343;                                % Speed of sound [m/s]
hornlength = 0.5;                       % Length of horn [m]
sourcefreq = 5000;                      % Sound source frequency [Hz]
signaltype = 1;                         % 1: Sine, 2: Wavelet
frameskip = 100;                        % Every "frameskip"'th step is plotted
micz = 0.25;                            % z position of "microphone"
micr = 0.1;                             % r position of "microphone"
geometry = 3;                           % 1: Constant cross-section, 2: Linear curve,
                                        % 3: Slow exponential, 4: Sharp exponential
                                        % 5: Chamber in pipe, 6: Camelback
% Calculated constants
nodelength = round(hornlength / dz);    % Horn length in nodes
iter = round((hornlength/c) / dt);      % Number of iterations
micj = round(micz/dz);
mici = round(micr/dr);
alpha = (c * dt / dr)^2;
betaconst = (c * dt)^2 / (2 * dr);
gamma = (c * dt / dz)^2;

% Determine geometry
switch (geometry)
    case 1
        heights(1:nodelength) = ceil(0.15 * hornlength / dr);
    case 2
        heights = floor(0.025/dr + 0.5 * (0 : nodelength-1)*dz/dr );
    case 3
        heights = floor(0.025/dr + 2 * ((0 : nodelength-1)*dz).^2 / dr);
    case 4
        split = round(nodelength/2);
        heights(1:split) = floor(0.025/dr + 150 * ((0 : split-1)*dz).^4 / dr);
        heights(split+1:nodelength) = heights(end);
    case 5
        split = round(nodelength/4);
        heights(1:nodelength) = ceil(0.1*hornlength / dr);
        heights(split:nodelength-2*split) = ceil(0.25*hornlength / dr);
    case 6
        heights(1:nodelength) = ceil((0.05*hornlength + 0.1 ...
            - 0.1 * cos(4*pi*(1:nodelength)/nodelength))/dr);
    otherwise
        error('Invalid geometry.');
end

nodeheight = max(max(heights));         % Horn height in nodes

% Create empty iteration vector and matrix
p = sparse(nodeheight*nodelength*2, 1, 0);
A = sparse(length(p), length(p), 0);
```

```
% Fill iteration matrix
for j = 1:nodelength
    for i = 1:heights(j)

        if (i > 1)
            beta = betaconst / ((i-1) * dr);
        end

        % Calculate the number of this node and its adjacent nodes
        nodenum = i + (j-1)*nodeheight;
        pinum = nodenum + 1;
        minum = nodenum - 1;
        pjnum = nodenum + nodeheight;
        mjnum = nodenum - nodeheight;
        mknum = nodenum + nodeheight*nodelength;

        % Is this node left-bounded?
        if ( j == 1 || i > heights(j-1) )
            % Is this node also up-bounded or down-bounded?
            if ( i == heights(j) )
                % CASE UL
                coeff = 1 / (1 - alpha - beta - gamma);
                A(nodenum, nodenum) = (- 2*alpha - 2*gamma + 2) * coeff;
                A(nodenum, minum) = (alpha - beta) * coeff;
                A(nodenum, pjnum) = (gamma) * coeff;
                A(nodenum, mknum) = (-1) * coeff;
            elseif (i == 1)
                % CASE DL
                coeff = 1 / (1 - gamma);
                A(nodenum, nodenum) = (- 4*alpha - 2*gamma + 2) * coeff;
                A(nodenum, pinum) = (4*alpha) * coeff;
                A(nodenum, pjnum) = (gamma) * coeff;
                A(nodenum, mknum) = (-1) * coeff;
            else
                % CASE L
                coeff = 1 / (1 - gamma);
                A(nodenum, nodenum) = (- 2*alpha - 2*gamma + 2) * coeff;
                A(nodenum, pinum) = (alpha + beta) * coeff;
                A(nodenum, minum) = (alpha - beta) * coeff;
                A(nodenum, pjnum) = (gamma) * coeff;
                A(nodenum, mknum) = (-1) * coeff;
            end

        % Is this node right-bounded?
        elseif ( j == nodelength )
            % Is this node also up-bounded or down-bounded?
            if ( i == heights(j) )
                % CASE UR
                coeff = 1 / (1 - alpha - beta - gamma);
                A(nodenum, nodenum) = (- 2*alpha - 2*gamma + 2) * coeff;
                A(nodenum, minum) = (alpha - beta) * coeff;
                A(nodenum, mjnum) = (gamma) * coeff;
                A(nodenum, mknum) = (-1) * coeff;
            elseif (i == 1)
                % CASE DR
                coeff = 1 / (1 - gamma);
                A(nodenum, nodenum) = (- 4*alpha - 2*gamma + 2) * coeff;
                A(nodenum, pinum) = (4*alpha) * coeff;
                A(nodenum, mjnum) = (gamma) * coeff;
```

```
            A(nodenum, mknum) = (-1) * coeff;
        else
            % CASE R
            coeff = 1 / (1 - gamma);
            A(nodenum, nodenum) = (- 2*alpha - 2*gamma + 2) * coeff;
            A(nodenum, pinum) = (alpha + beta) * coeff;
            A(nodenum, minum) = (alpha - beta) * coeff;
            A(nodenum, mjnum) = (gamma) * coeff;
            A(nodenum, mknum) = (-1) * coeff;
        end

    % Is this node simply up-bounded?
    elseif ( i == heights(j) )
        % CASE U
        coeff = 1 / (1 - alpha - beta);
        A(nodenum, nodenum) = (- 2*alpha - 2*gamma + 2) * coeff;
        A(nodenum, minum) = (alpha - beta) * coeff;
        A(nodenum, pjnum) = (gamma) * coeff;
        A(nodenum, mjnum) = (gamma) * coeff;
        A(nodenum, mknum) = (-1) * coeff;

    % Is this node simply down-bounded?
    elseif ( i == 1 )
        % CASE D
        coeff = 1;
        A(nodenum, nodenum) = (- 4*alpha - 2*gamma + 2) * coeff;
        A(nodenum, pinum) = (4*alpha) * coeff;
        A(nodenum, pjnum) = (gamma) * coeff;
        A(nodenum, mjnum) = (gamma) * coeff;
        A(nodenum, mknum) = (-1) * coeff;

    % Is this node not bounded at all?
    else
        % CASE N
        coeff = 1;
        A(nodenum, nodenum) = (- 2*alpha - 2*gamma + 2) * coeff;
        A(nodenum, pinum) = (alpha + beta) * coeff;
        A(nodenum, minum) = (alpha - beta) * coeff;
        A(nodenum, pjnum) = (gamma) * coeff;
        A(nodenum, mjnum) = (gamma) * coeff;
        A(nodenum, mknum) = (-1) * coeff;
    end

    % Also copy this node to the last half of the vector (k and k-1)
    A(mknum, nodenum) = 1;
  end
end

% Set plot colour and size
figure
colormap(gray);
set(gca, 'FontSize', 20);
switch geometry
  case 1
    set(gcf, 'OuterPosition', [440, 300, 550, 250]);
  case 2
    set(gcf, 'OuterPosition', [440, 300, 550, 400]);
  case 3
    set(gcf, 'OuterPosition', [440, 300, 400, 500]);
  case 4
    set(gcf, 'OuterPosition', [440, 300, 400, 500]);
```

```
    case 5
        set(gcf, 'OuterPosition', [440, 300, 550, 250]);
    case 6
        set(gcf, 'OuterPosition', [440, 300, 550, 350]);
end

% Perform the iterations
for it = 1:frameskip:iter

    % Set source values and perform a step
    for skipno = 0:frameskip-1
        % Set source values
        t = (it+skipno)*dt;
        switch signaltype
            case 1
                sourceval(it+skipno) = sin(2*pi* sourcefreq *t);
            case 2
                x2 = (2*sqrt(3)*sourcefreq*t - 3*sqrt(3))^2;
                sourceval(it+skipno) = (1 - x2) * exp(-0.5*x2);
        end
        p(1:heights(1)) = sourceval(it+skipno);

        % Store the pressure at a point in the system for later analysis
        p_mic(it+skipno) = p(1 + nodeheight*micj + mici);

        % Perform a step
        p = A*p;
    end

    % Put the pressures into a two-dimensional matrix for display
    pimg = zeros(nodeheight, nodelength);
    for j = 1:nodelength
        colstart = 1 + (j-1) * nodeheight;
        colend = j * nodeheight;
        pimg(:, j) = p(colstart:colend);
        pimg(heights(j)+1 : nodeheight, j) = -1.5;
    end

    % A hack to lock the colour of zero
    pimg(end, 1) = 1.5;

    % Use square root of pressures for plot clarity (can be commented out)
    pimg = abs(pimg.^(1/2)) .* sign(pimg);

    % Plot the state of the system
    imagesc((0:nodelength-1)*dz, (0:nodeheight-1)*dr, pimg);
    axis xy
    axis image
    xlabel('$z$ [m]', 'Interpreter', 'latex');
    ylabel('$r$ [m]', 'Interpreter', 'latex');
    getframe;
end
```