AE 370 Project 2
Reflected Waves in A Non-Homogeneous Medium

Linyi Hou | github.com/TigerHou2

May 11, 2020

1 Introduction

When a wave passes through the interface between two different media, part of the wave is reflected in the original direction of travel (assuming the wave is traveling perpendicular to the interface). The goal of this project is to examine the behavior of reflected waves in combinations of different media in one dimension.

We will model different media by altering the speed of sound, and develop a finite difference method to approximate the wave equation with initial and boundary conditions. Several real world scenarios will be simulated to qualitatively evaluate the validity of the simulation, in addition to quantitative convergence testing.

2 The Wave Equation

The general form of the wave equation is

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \ , \ 0 \leq t \leq T \ , \ a \leq x \leq b \tag{1}$$

We are interested in the behavior of reflected waves when the medium changes — this will be modeled by a wave speed function $c(x)$:

$$\frac{\partial^2 u}{\partial t^2} = c(x)^2 \frac{\partial^2 u}{\partial x^2} \tag{2}$$

with initial conditions

$$u(x, t = 0) = w(x) \tag{3}$$
$$\dot{u}(x, t = 0) = v(x) \tag{4}$$

and boundary conditions

$$u(x = a, t) = f(t) \tag{5}$$
$$u(x = b, t) = g(t) \tag{6}$$

3 Finite Difference Method

First, we discretize the space variable x into $n+1$ pieces:

$$x_k = a + \frac{(b-a)(k-1)}{n}, \; k = 1, \ldots, n+1 \tag{7}$$

Then, let us use $u_k(t)$ to denote the approximation of the true solution $u(x_k, t)$. Using finite difference to approximate the second derivatives, we have:

$$\frac{d^2 u_k}{dt^2} = \frac{1}{\Delta t^2}\left(u_{k+1}(t) - 2u_k(t) + u_{k-1}(t)\right) + O(\Delta t^2) \tag{8}$$

$$\frac{d^2 u_k}{dx^2} = \frac{1}{\Delta x^2}\left(u_k(t+\Delta t) - 2u_k(t) + u_k(t-\Delta t)\right) + O(\Delta x^2) \tag{9}$$

Substituting Equations (8) and (9) into Equation (2) by ignoring higher order terms and using the variable substitution $\sigma = \frac{c(x)\Delta t}{\Delta x}$:

$$u_k(t+\Delta t) = \sigma^2 u_{k+1}(t) + 2(1-\sigma^2)u_k(t) + \sigma^2 u_{k-1}(t) - u_k(t-\Delta t) \tag{10}$$

now we have obtained the expression for u_k at the next time step. Note that the future state of u_k is dependent not only on its adjacent states at the current time, it also depends on its state at the previous time step.

Equation (10) can be written in matrix form:

$$\begin{aligned}\begin{bmatrix} u_2(t+\Delta t) \\ u_3(t+\Delta t) \\ \vdots \\ u_{n-1}(t+\Delta t) \\ u_n(t+\Delta t) \end{bmatrix} &= \begin{bmatrix} 2(1-\sigma^2) & \sigma^2 & & & \\ \sigma^2 & 2(1-\sigma^2) & \sigma^2 & & \\ & \ddots & \ddots & \ddots & \\ & & \sigma^2 & 2(1-\sigma^2) & \sigma^2 \\ & & & \sigma^2 & 2(1-\sigma^2) \end{bmatrix} \begin{bmatrix} u_2(t) \\ u_3(t) \\ \vdots \\ u_{n-1}(t) \\ u_n(t) \end{bmatrix} \\ &- \begin{bmatrix} u_2(t-\Delta t) \\ u_3(t-\Delta t) \\ \vdots \\ u_{n-1}(t-\Delta t) \\ u_n(t-\Delta t) \end{bmatrix} + \begin{bmatrix} \sigma^2 f(t) \\ 0 \\ \vdots \\ 0 \\ \sigma^2 g(t) \end{bmatrix}\end{aligned} \tag{11}$$

To start up the finite difference method, we know from our observation of Equation (10) that the values of u at two adjacent time steps are required. We also know that the finite difference approximate error should be $\max\left(O(\Delta t^2), O(\Delta x^2)\right)$. The solution to this is provided by [1] and shown below.

From Taylor's theorem:

$$\frac{1}{\Delta t}\left(u_k(t+\Delta t)-u_k(t)\right)=\frac{\partial u_k}{\partial t}(t)+\frac{\Delta t}{2}\frac{\partial^2 u_k}{\partial t^2}(t)+O(\Delta t^2) \tag{12}$$

using the wave equation in Equation (2), Equation (12) becomes

$$\frac{1}{\Delta t}\left(u_k(t+\Delta t)-u_k(t)\right)=\frac{\partial u_k}{\partial t}(t)+O(\Delta t^2)+\frac{\Delta t\, c(x_k)^2}{2}\frac{\partial^2 u_k}{\partial x^2}(t)+O(\Delta t^2) \tag{13}$$

rearranging, ignoring higher order terms, and substituting $t=0$:

$$u_k(\Delta t)=u_k(0)+\Delta t\frac{\partial u_k}{\partial t}(0)+\frac{\Delta t^2 c(x_k)^2}{2}\frac{\partial^2 u_k}{\partial x^2}(0) \tag{14}$$

Using the initial conditions from Equations (3) and (4),

$$u_k(0)=w(x_k) \tag{15}$$

$$\frac{\partial u_k}{\partial t}(0)=v(x_k) \tag{16}$$

and from another Taylor expansion with Equation (3) we also have

$$\begin{aligned}\frac{\partial^2 u_k}{\partial x^2}(0)&=\frac{1}{\Delta x^2}\left(u_{k+1}(0)-2u_k(0)+u_{k-1}(0)\right)\\&=\frac{1}{\Delta x^2}\left(w(x_{k+1})-2w(x_k)+w(x_{k-1})\right)\end{aligned} \tag{17}$$

which allows us to arrive at the equation

$$u_k(\Delta t)=\frac{1}{2}\sigma^2 w(x_{k+1})+(1-\sigma^2)w(x_k)+\frac{1}{2}\sigma^2 w(x_{k-1})+\Delta t\, v(x_k)\ ,\quad k=1,\ldots,n+1 \tag{18}$$

To simplify notation, let us define the following:

$$\boldsymbol{u}^0=[u_2(0),u_3(0),\cdots,u_{n-1}(0),u_n(0)] \tag{19}$$

$$\boldsymbol{u}^1=[u_2(\Delta t),u_3(\Delta t),\cdots,u_{n-1}(\Delta t),u_n(\Delta t)] \tag{20}$$

$$\boldsymbol{u}^i=[u_2((i-1)\Delta t),u_3((i-1)\Delta t),\cdots,u_{n-1}((i-1)\Delta t),u_n((i-1)\Delta t)] \tag{21}$$

and also rewrite Equation (11) in vector form:

$$\boldsymbol{u}^{i+1}=\boldsymbol{A}\boldsymbol{u}^i-\boldsymbol{u}^{i-1}+\boldsymbol{h}(t) \tag{22}$$

Summarizing, we have the two time steps required to start the numerical method, with error on the order of $\max\left(O(\Delta t^2),O(\Delta x^2)\right)$:

$$\boldsymbol{u}^0=w(\boldsymbol{x}) \tag{23}$$

$$\boldsymbol{u}^1=\frac{1}{2}\boldsymbol{A}\boldsymbol{u}^0+\Delta t\, v(\boldsymbol{x})+\frac{1}{2}\boldsymbol{h}(0) \tag{24}$$

It should also be noted that due to our error being max $\left(O(\Delta t^2), O(\Delta x^2)\right)$, we should choose $\Delta t \approx \Delta x$. Otherwise, the larger delta would dominate the error term, rendering the effect of refining the other delta useless. For the remainder of the paper, we will choose

$$\Delta t = 0.9 * \frac{c'}{\Delta x} \quad , \quad c' = \max_x c(x), \; a \le x \le b. \tag{25}$$

4 Simulation Setup and Validation

4.1 Initial and Boundary Conditions

For the remainder of the paper, we will be considering $t \in [0, 8]$, $x \in [0, 8]$, with

Initial Conditions:

$$u(x, t = 0) = 0 \tag{26}$$

$$\dot{u}(x, t = 0) = 0 \tag{27}$$

Boundary Conditions:

$$u(x = a, t) = \frac{1}{\pi}\left(1 - \cos(2\pi t)\right)\left(\frac{\pi}{2} - \tan^{-1}\left(\frac{t-1}{10^{-5}}\right)\right) \tag{28}$$

$$u(x = b, t) = 0 \tag{29}$$

The boundary condition shown in Equation (28) simply generates a one-pulse wave from the left-hand side of the graph and then returns to 0, as shown in Figure 1.

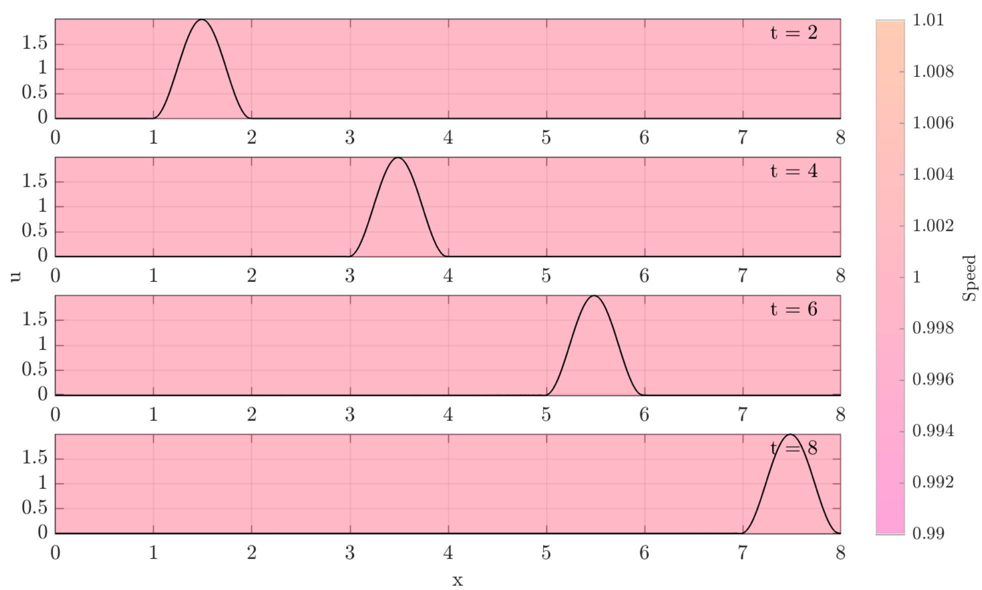
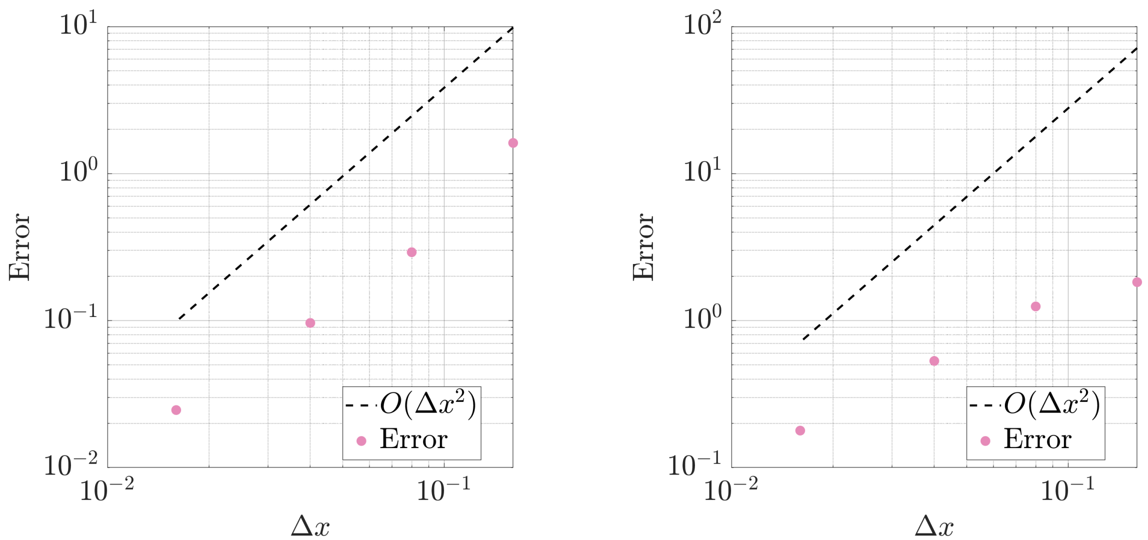


Figure 1: One-pulse waveform (top to bottom: wave development over time)

4.2 Validation - Convergence

We will examine the convergence of the method derived in Section 3 by computing the normalized difference in $\boldsymbol{u}(T)$ across different step sizes.

Using a homogeneous medium, we obtain a waveform similar to the one shown in Figure 1, and a convergence rate of $O(\Delta x^2)$ as expected, as shown in Figure 2a. However, using a non-homogeneous medium results in the error convergence rate being slower than $O(\Delta x^2)$, as shown in Figure 2b. The corresponding waveform is shown in Figure 3.



(a) Homogeneous medium

(b) Non-homogeneous medium

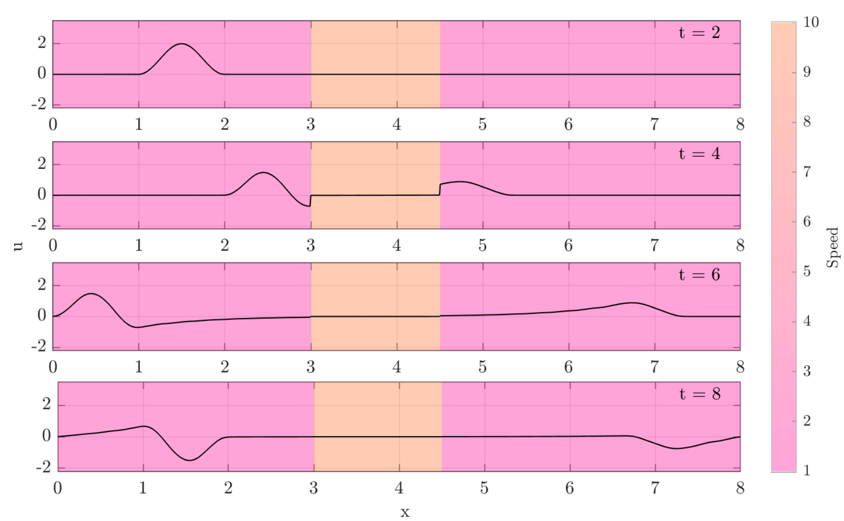


Figure 3: Non-homogeneous medium waveform over time

It is suspected that the non-constant speed of sound $c(x)$ slightly destabilizes the numerical method due to the CFL condition, which requires

$$\sigma = \frac{c(x)\Delta t}{\Delta x} \leq 1. \tag{30}$$

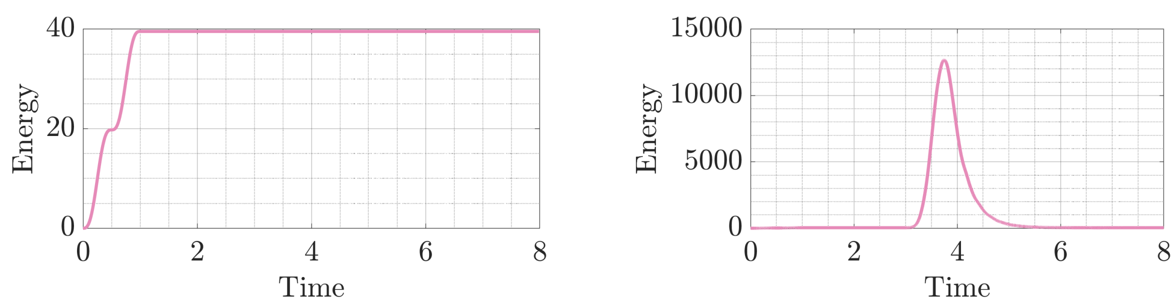
However, as of writing this paper, no clear approach was determined to eliminate this instability. Since the error still converges (only to a lesser order), we choose to proceed with the method while checking convergence for each simulation to ensure validity.

4.3 Validation - Conservation of Energy

To further validate the numerical method, we will also apply the conservation of energy:

$$const. = E(t) = \int_a^b \left(\frac{\partial u}{\partial t}\right)^2 dx + \int_a^b c(x)^2 \left(\frac{\partial u}{\partial x}\right)^2 dx \tag{31}$$

Again, we compare the results between a homogeneous and a non-homogeneous medium:



(a) Homogeneous medium (b) Non-homogeneous medium

We find that for a homogeneous medium energy is conserved throughout the simulation. Energy is initially added to the system via the pulse boundary condition shown in Equation (28), and then holds constant.

However, in a non-homogeneous medium energy does not seem to be conserved while the wave is passing through the altered medium. We know this is impossible due to the conservation of energy. This effect, however, vanishes upon the wave leaving the changed medium and back into the original medium.

Future work should be done to examine how to properly model the conservation of energy in a non-homogeneous medium. It is suspected that this inaccuracy is also the cause of the mismatch in convergence rate discussed in Section 4.2.

5 Results

We can now study the behavior of reflected waves using our numerical method. We will examine three types of non-homogeneous media: water against a wall, air around a plate, and a planetary atmosphere. Animations and code used for this paper can be found at

https://github.com/TigerHou2/ae370/tree/master/Project02/gifs

https://github.com/TigerHou2/ae370/tree/master/Project02

5.1 Discrete Non-Homogeneous Media (A-B)

We will first examine the simplest case of non-homogeneous media: two distinct media A and B are joined together to form structure A-B. For example, waves in a swimming pool impacting the sides of the pool can be modeled using this approach.

We model the water with wave speed $c_{water} = 1.0$, and the wall with wave speed $c_{wall} = 3.0$. As shown in Figure 5, the majority of the wave is reflected back to the left hand side of the horizontal axis, while a weaker wave is seem propagating to the right, through the interface. This also matches the expected behavior of waves in a pool from common sense.

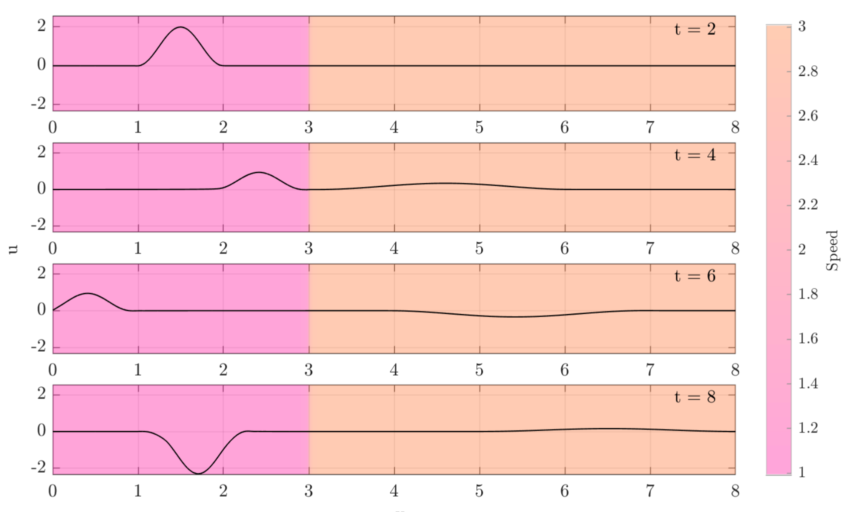


Figure 5: Discrete non-homogeneous medium (type A-B)

5.2 Discrete Non-Homogeneous Media (A-B-A)

We will now model a medium B inserted in the middle of medium A, creating an A-B-A structure. As an example, consider waves propagating in air blocked by a thin steel wall.

We model the air with wave speed $c_{air} = 1.0$ and the steel wall with wave speed $c_{wall} = 10.0$. We see from Figure 6 that the majority of the wave passes through the medium while only a small portion is reflected. This once again matches with common sense — a loud noise will propagate through a wall and only ring back at a much lower volume.

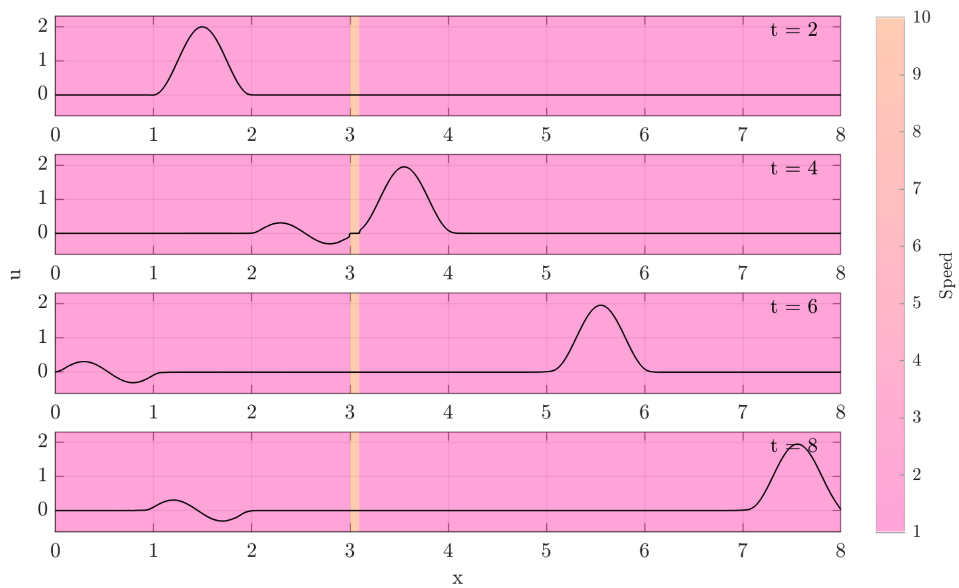


Figure 6: Discrete non-homogeneous medium (type A-B-A)

The wave energy stored in the left hand side of the wall is tracked through time in Figure 7. We see that the energy injected into the system has mostly dissipated through the wall.

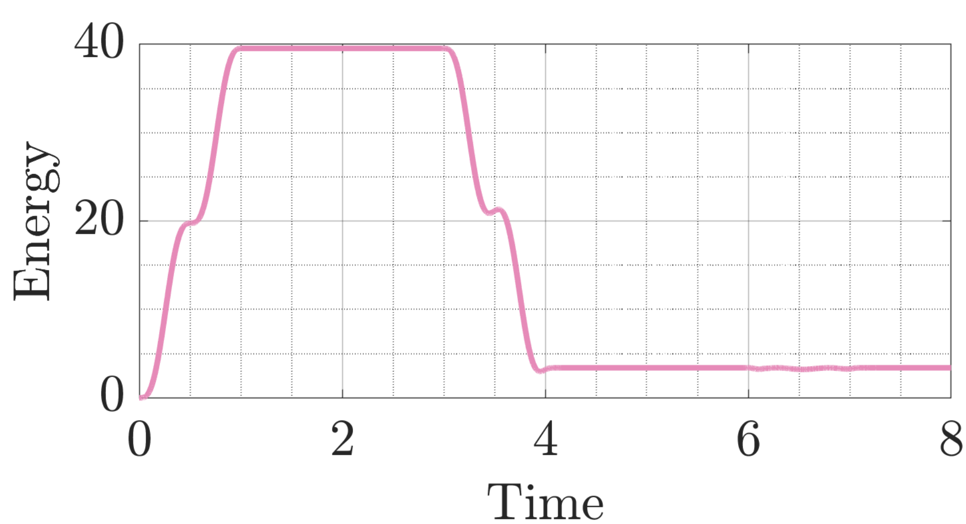


Figure 7: Wave energy on the interval $0 \leq x \leq 3$

5.3 Continuous Non-Homogeneous Media

Finally, we will model wave propagation through a continuously varying medium. An example of this is Earth's atmosphere, where the density varies by altitude. The shock wave from an asteroid entering the Earth's atmosphere can be modeled using this method.

We model the atmosphere from top ($x = 0$) to bottom ($x = 8$) with a varying local speed of sound from 3.0 to 1.0. The shock can be seen to rapidly propagate through the upper atmosphere (left side) with a small amplitude, and rapidly increase in amplitude as it slows down in the lower atmosphere (right side).

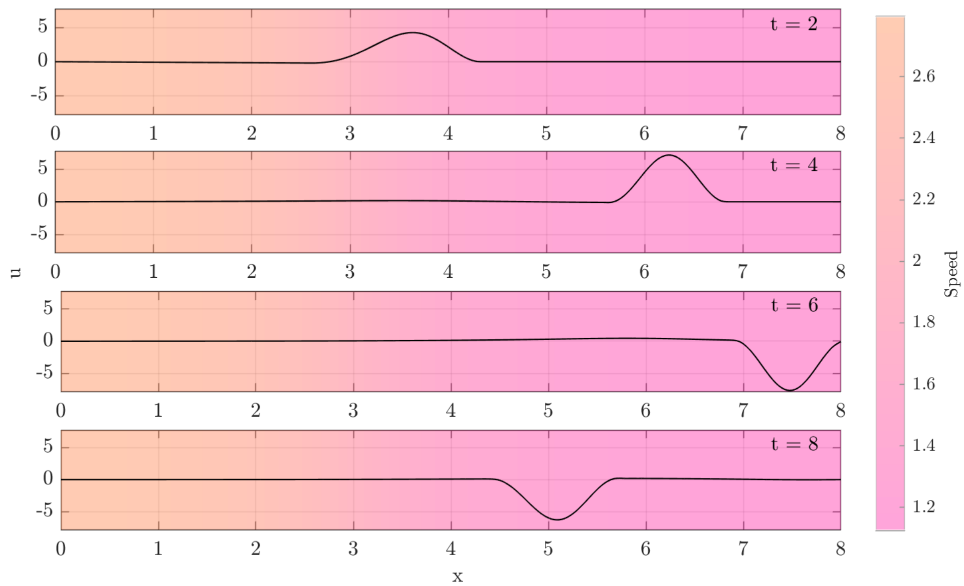


Figure 8: Continuous non-homogeneous medium

This is similar to how a tsunami works: as the wave speed decreases in shallow waters, the amplitude of the tsunami rapidly increases from the conservation of mass. Thus energy is delivered at a higher density for a slower wave.

6 Conclusion

A finite difference numerical method was developed for solving the wave equation. The truncation error was chosen to be second order with respect to both temporal and spatial time steps. Convergence analysis and energy analysis show that the numerical method is suitable for modeling waves with homogeneous speed but has reduced accuracy and limitations for non-homogeneous wave speeds.

Using the finite difference method, several scenarios were explored to better understand how waves behave in non-homogeneous media, drawing inspiration from real life examples including waves in a swimming pool, sound against a wall, and Earth's atmosphere.

Future work on the finite difference method is needed to accurately model energy conservation for non-homogeneous media. Once the method is improved, further studies can be performed. One particular item of interest is a wave speed function dependent on u, i.e. $c(u)$. This would allow modeling of air compression, whereby the speed of sound in air is related to its density by the ideal gas law.

Appendix A: Code

```
%% main.m
%
% Author:
%   Tiger Hou
%
% Description:
%   Finite difference method for reflected waves in changing physical
    media
%
%% simulation
%
% https://www-users.math.umn.edu/~olver/num_/lnp.pdf
%
close all
clear;clc

saveGIF = true;
play = true;

num_pics = 4;
saveFigs = false;

hideWallEnergy = false;

if saveGIF
    fileName = ...
```

```
    input(['Please enter the prefix for your file name(s):' newline],'s');
elseif saveFigs
    fileName = ...
    input(['Please enter the prefix for your file names:' newline],'s');
end

% define left/right boundaries
a = 0;  % left boundary
b = 8;  % right boundary
ln = b-a;

% common starting point for wall and transition point
lc = 3.0;

% define wave speed for wall cases
% tc  = 0.1; % thickness
% spA = 1;  % speed of sound not in wall
% spW = 1;  % speed of sound in wall
% c = @(x) (x< lc) * spA + ...
%          (x>=lc & x < lc+tc) * spW + ...
%          (x>=lc+tc) * spA;

% define wave speed for transition cases
spL = 3.0; % speed of sound on the left
spR = 1.0; % speed of sound on the right
sharpness = 1; % larger value = sharper transition ; use [5, 15, 100]
c = @(x) ( (spL+spR)/2 - (spL-spR)/2*atan(sharpness*(x-lc))/pi*2 );

% initial conditions
w0 = @(x) zeros(size(x));
v0 = @(x) zeros(size(x));

% boundary conditions
f = @(t) (-cos(t*2*pi)+1)*(pi/2-atan(10000*(t-1)))/pi;
g = @(t) 0;

% # of n points to use
nvect = [50, 500];

```

```
% final time
T = 8;
% dt must match dx in order of magnitude
dtvect = 0.9 * (b-a)./nvect/max(c(linspace(a,b,10000)));

% initialize u vect
uvect = cell(size(nvect));

% initialize error vector
final_vect = cell(size(nvect));

% iterate through interior point sizes
for j = 1 : length( nvect )

    % Build n, xj points, A matrix and g vector
        % # of grid points
        n = nvect( j );

        % choose time step matching dx
        dt = dtvect(j);

        % build interp points
        xj = linspace(a,b,n+1)';

        % grid spacing (uniform)
        dx = ( b - a ) / n;

        % sigma function
        s = c(xj) * dt / dx;

        % build A matrix
        diag_ct = diag(2*(1-s(2:end-1).^2));
        diag_dn = diag(s(2:end-2).^2,-1);
        diag_up = diag(s(3:end-1).^2, 1);
        A = diag_ct + diag_dn + diag_up;

        % special sigma functions for f and g
        ssf = c(a) * dt / dx;
        ssg = c(b) * dt / dx;

```

```
        % build h for this set of xj
        h = @(t) [ssf^2*f(t); zeros(size(xj,1)-4,1); ssg^2*g(t)];

    % Iterate through time

        % build time vector
        tvect = 0:dt:T-dt;

        % second order states initialization
        um1 = w0(xj(2:end-1));
        uu  = 0.5 * A * um1 + dt * v0(xj(2:end-1)) + 0.5 * h(0);
        uuvect = repmat(uu,1,length(tvect));
        uuvect(:,1) = um1;
        uuvect(:,2) = uu;

        for i = 3:length(tvect)

            up1 = A * uu - um1 + h(tvect(i));
            um1 = uu;
            uu  = up1;

            uuvect(:,i) = up1;

        end

        % store entire wave solution
        uvect{j} = uuvect;

        % store final state to compare errors later
        final_vect{j} = up1;

end

% --- error comparison
err = nan(length(nvect)-1,1);
xj_ref = xj;
uu_ref = up1;
for j = 1 : length( nvect ) - 1

    n = nvect(j);
```

```
    xj = linspace(a,b,n+1);

    [~,idx] = min( abs( xj_ref - xj(2:end-1) ) );

    uu = final_vect{j};

    err(j) = norm( uu_ref(idx) - uu );

end


%% plot error

figure(1)

cc = err(end)./(dx.^2);
loglog( ln./nvect(1:end-1), cc.*(ln./nvect(1:end-1)).^2, ...
        'k--', 'linewidth', 2 )
hold on

loglog( ln./nvect(1:end-1), err , '.', 'markersize', 32, 'Color', [0.9 0.54 0.72])
legend('$O(\Delta x^2)$', 'Error', 'Location', 'Best');
hold off

xlabel('$\Delta x$')
ylabel('Error')

setgrid(0.3,0.9)
latexify(19,19,28)

if saveFigs
    svnm = ['figures/' fileName '_error'];
    print( '-dpng', svnm, '-r200' )
end


%% plot energy

```

```
figure(2)

% extract the highest resolution data
waveData = uvect{end};
% create x-axis, mapped to highest resolution data
xx = linspace(a,b,size(waveData,1))';

if hideWallEnergy
    % cut both waveData and xx to contain only points before boundary
          xx(ceil(size(waveData,1)*(lc-a)/ln):end,:) = [];
    waveData(ceil(size(waveData,1)*(lc-a)/ln):end,:) = [];
end

dx = ( b - a ) / nvect(end);
dt = dtvect(end);
tvect = 0:dt:T-2*dt;

indices = 1:size(waveData,2)-1;
E = nan(size(indices));
for i = 1:length(indices)
    E(i) = checkEnergy(indices(i),waveData,dx,dt,c,xx);
end
plot(tvect,E,'LineWidth',3,'Color',[0.9 0.54 0.72])

xlabel('Time')
ylabel('Energy')
setgrid(0.3,0.9)
latexify(19,10,28)
expand(0,0.04)

if hideWallEnergy
    fileExt = '_energy_prewall';
else
    fileExt = '_energy';
end
if saveFigs
    svnm = ['figures/' fileName fileExt];
    print( '-dpng', svnm, '-r200' )
end

```

```
%% plot waves

if play || saveGIF

ff = figure(3);

% extract the highest resolution data
waveData = uvect{end};
% create x-axis, mapped to highest resolution data
xx = linspace(a,b,size(waveData,1));

% initialize wave
wave = line(NaN,NaN,'LineWidth',1,'Color','k');
hold on

% make background gradient
bgAlpha = 0.5;
yy = linspace(min(waveData(:)),max(waveData(:)),100);
X = meshgrid(xx,yy);
Z = c(X);
cmap = spring(100);
colormap(cmap(30:60,:))
bg = image(xx,yy,Z,'CDataMapping','scaled');
bg.AlphaData = bgAlpha;
bg_bar = colorbar;
bg_bar.TickLabelInterpreter = 'latex';
bg_bar.Label.String = 'Speed';
bg_bar.Label.Interpreter = 'latex';

% make plot pretty
xlabel('x')
ylabel('u')
latexify(19,15,20)

% mask colorbar to match transparency
annotation('rectangle',...
    bg_bar.Position,...
    'FaceAlpha',bgAlpha,...
    'EdgeColor',[1 1 1],...
```

```
    'FaceColor',[1 1 1]);

% animate and make gif
axis tight manual % this ensures that getframe() returns a consistent size
set(gcf,'Renderer','zbuffer')
filename = ['gifs/' fileName '.gif'];
lim_x = [a;b];
lim_y = [min(waveData(:));max(waveData(:))];
lim_z = [0;1];
update_view(lim_x,lim_y,lim_z);
playtime = T;

% annotate time
posx = lim_x(1) + 0.07 * (lim_x(2) - lim_x(1));
posy = lim_y(1) + 0.90 * (lim_y(2) - lim_y(1));
txt = text(posx,posy,['t = ', '0.00']);

% constrain GIF to a fixed FPS
fps = 60;
frames = playtime * fps;
shutter = ceil(size(waveData,2)/frames);
latexify(19,15,20)

for i = 1:size(waveData,2)

    % update plot
    if mod(i-1,shutter)==0
        set(wave,'XData',xx,'YData',waveData(:,i))
        set(txt, 'String',['t = ', num2str(i*dtvect(end),'%2.2f')])
        pause(1/fps) % pause for proper MATLAB display speed
    end

    % save GIF
    if saveGIF && mod(i-1,shutter)==0

        % capture the plot as an image
        frame = getframe(ff);
        im = frame2im(frame);
        [imind,cm] = rgb2ind(im,256);
```

```
        % write to the GIF file
        if i == 1
          imwrite(imind,cm,filename,...
                    'gif','Loopcount',inf,'DelayTime',1/fps);
        else
          imwrite(imind,cm,filename,...
                    'gif','WriteMode','append','DelayTime',1/fps);
        end

    end

end

hold off

end


%% save screenshots

sc = figure(4);

% take a fixed number of screenshots in equispaced time
shutter = floor(size(waveData,2)/num_pics);
idx = shutter;

% extract the highest resolution data
waveData = uvect{end};
% create x-axis, mapped to highest resolution data
xx = linspace(a,b,size(waveData,1));

% make background gradient
bgAlpha = 0.5;
yy = linspace(min(waveData(:)),max(waveData(:)),100);
[X,Y] = meshgrid(xx,yy);
Z = c(X);

% set axis limits
lim_x = [a;b];
```

```
lim_y = [min(waveData(:));max(waveData(:))];
lim_z = [0;1];

% make figure pretty
latexify(19,12)

for i = 1:num_pics

    subplot(4,1,i)

    % plot function
    pp = plot(xx,waveData(:,idx),'k','LineWidth',0.75);
    idx = idx + shutter;
    hold on
    update_view(lim_x,lim_y,lim_z);

    % expand to window boundary
    expand(0.05,0.15)

    % plot background
    cmap = spring(100);
    colormap(cmap(30:60,:))
    bg = image(xx,yy,Z,'CDataMapping','scaled');
    bg.AlphaData = bgAlpha;

    % define axis limits
    lim_x = [a;b];
    lim_y = [min(waveData(:));max(waveData(:))];
    lim_z = [0;1];
    update_view(lim_x,lim_y,lim_z);

    grid on
    hold off

    % move wave to top of display stack
    uistack(pp,'top')

    % annotate time
    posx = lim_x(1) + 0.91 * (lim_x(2) - lim_x(1));
    posy = lim_y(1) + 0.85 * (lim_y(2) - lim_y(1));
```

```
    text(posx,posy,['t = ', num2str(T/num_pics*i)])

end

hold on
handle = axes(sc,'visible','off');
handle.XLabel.Visible = 'on';
handle.YLabel.Visible = 'on';
xlabel('x')
ylabel('u')
expand(0.01,0.04,0.02,0.06)

latexify(19,12,11)

% plot colorbar
bg_bar = colorbar;
caxis([min(c(xx))-0.01,max(c(xx))+0.01])
bg_bar.TickLabelInterpreter = 'latex';
bg_bar.Label.String = 'Speed';
bg_bar.Label.Interpreter = 'latex';

% mask colorbar to match transparency
annotation('rectangle',...
    bg_bar.Position,...
    'FaceAlpha',bgAlpha,...
    'EdgeColor',[1 1 1],...
    'FaceColor',[1 1 1]);

hold off

if saveFigs
    svnm = ['figures/' fileName '_snaps'];
    print( '-dpng', svnm, '-r200' )
end


%% safety

% if we run part of the code again, don't resave figures
saveFigs = false;
```

```matlab


%% supporting functions

function update_view(lim_x,lim_y,lim_z)

    xlim(lim_x)
    ylim(lim_y)
    zlim(lim_z)

end

function E = checkEnergy(idx,waveData,dx,dt,c,xx)

    dudx = (waveData(2:end,idx) - waveData(1:end-1,idx)) / dx;
    dudt = (waveData(:,idx+1) - waveData(:,idx)) / dt;

    cc = c(xx).^2;

    E = trapz(xx,dudt.^2) ...
      + trapz(xx(1:end-1),cc(1:end-1).*dudx.^2);

end
```

References

[1] P. J. Olver, "Numerical analysis lecture notes." https://www-users.math.umn.edu/~olver/num_/lnp.pdf, 2008. Accessed: 5-12-2019.