BME 8730
Fall 2019
Due October 14th 2019 (Extendable)

Samuel Sklar
Implementing Time Domain Diffraction Equations

Implement equations for velocity potential impulse response for both circular and rectangular sources. I suggest you start from the Lockwood paper and refer to the other diffraction papers to the extent that they help you. These papers are in "Resources". The code should also include convolution with source waveform function and differentiation post convolution to obtain pressure (10,20)

Comment your code properly.
Produce a sketch (hand drawn if you prefer) denoting the geometries being used and the variable names in your matlab code. For example, show the arcs and angles for the various permutations of field point locations and shapes.

When dealing with rectangular apertures, your code should, ideally, be able handle all possible orientations – i.e. field points in all possible positions and account for the different order of critical time points depending on BOTH field point location (projected on to transducer plane) and the different permutations of long versus short sides of a rectangular source.
You code, if it doesn't address all geometries, still needs to address those required to answer other questions listed here (see below)

Use your code to replicate the following figures:

Lockwood
Figure 3, 6, 7, 8 and 13 (10,10,10,10,10)

For the CW beamplots, you should consider convolving with a long sinusoidal pulse and detecting the peak to peak amplitude near the center of the pulse (to avoid transient effects near the beginning and end) or use an FFT of the impulse response and extract the correct Fourier component. (If one doesn't work, try the other.) Be sure to use fine time sampling – at least 100 MHz and probably significantly higher. Notice that you need to define a and λ to make your code work. It isn't critical what actual values you use so long as they are consistent and produce the right results.

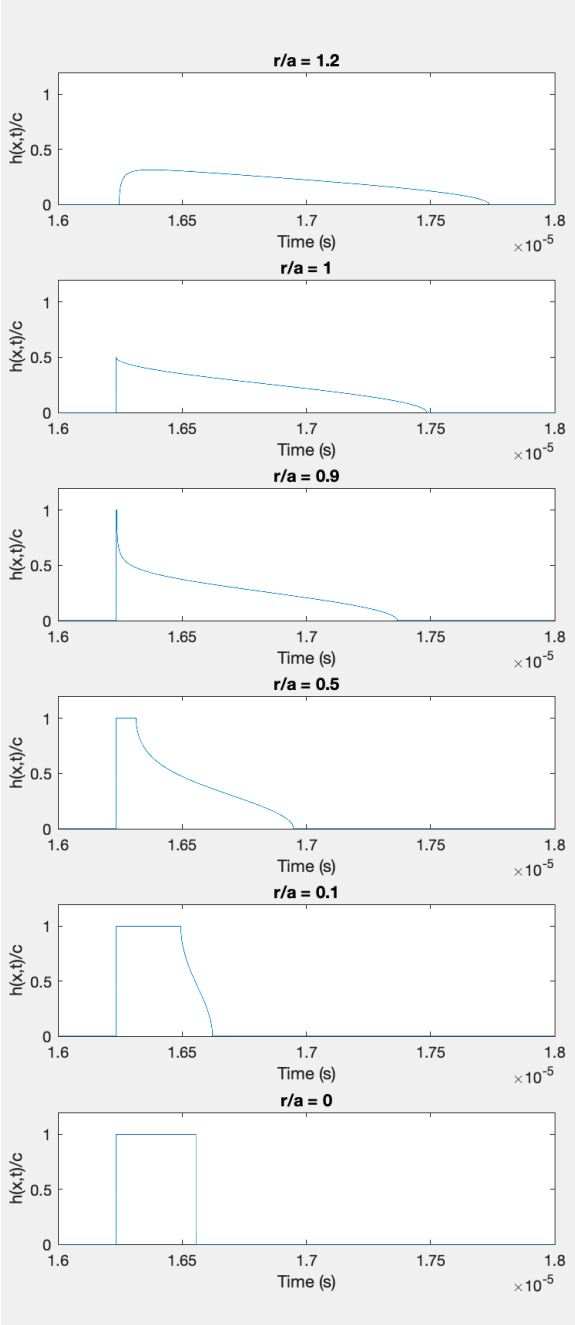
It is fine to discuss progress or questions in class. We have time for that.

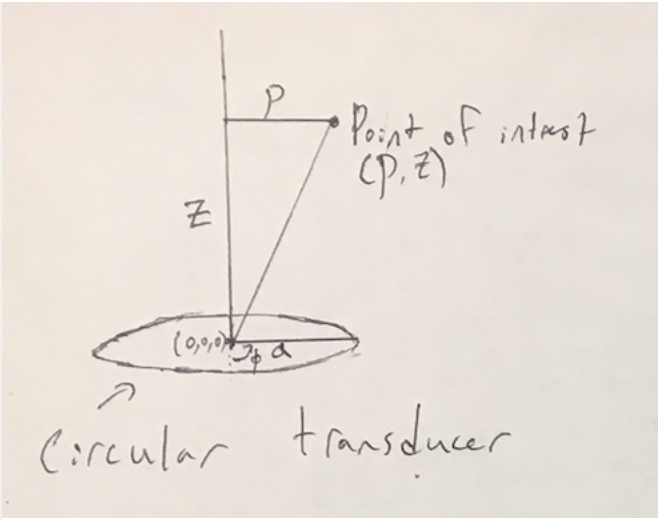
Please try to do as much of the coding yourself but work together to debug (and acknowledge). If coding is proving to be a real challenge, pair up. However, if you do this please try to learn the operation of the code equally.

Submit, paper or PDF/word doc, figures and code – at least the core code doing the trigonometry.

Work for Circular Aperture:

Recreation of figure 3 with geometry Code on next page.

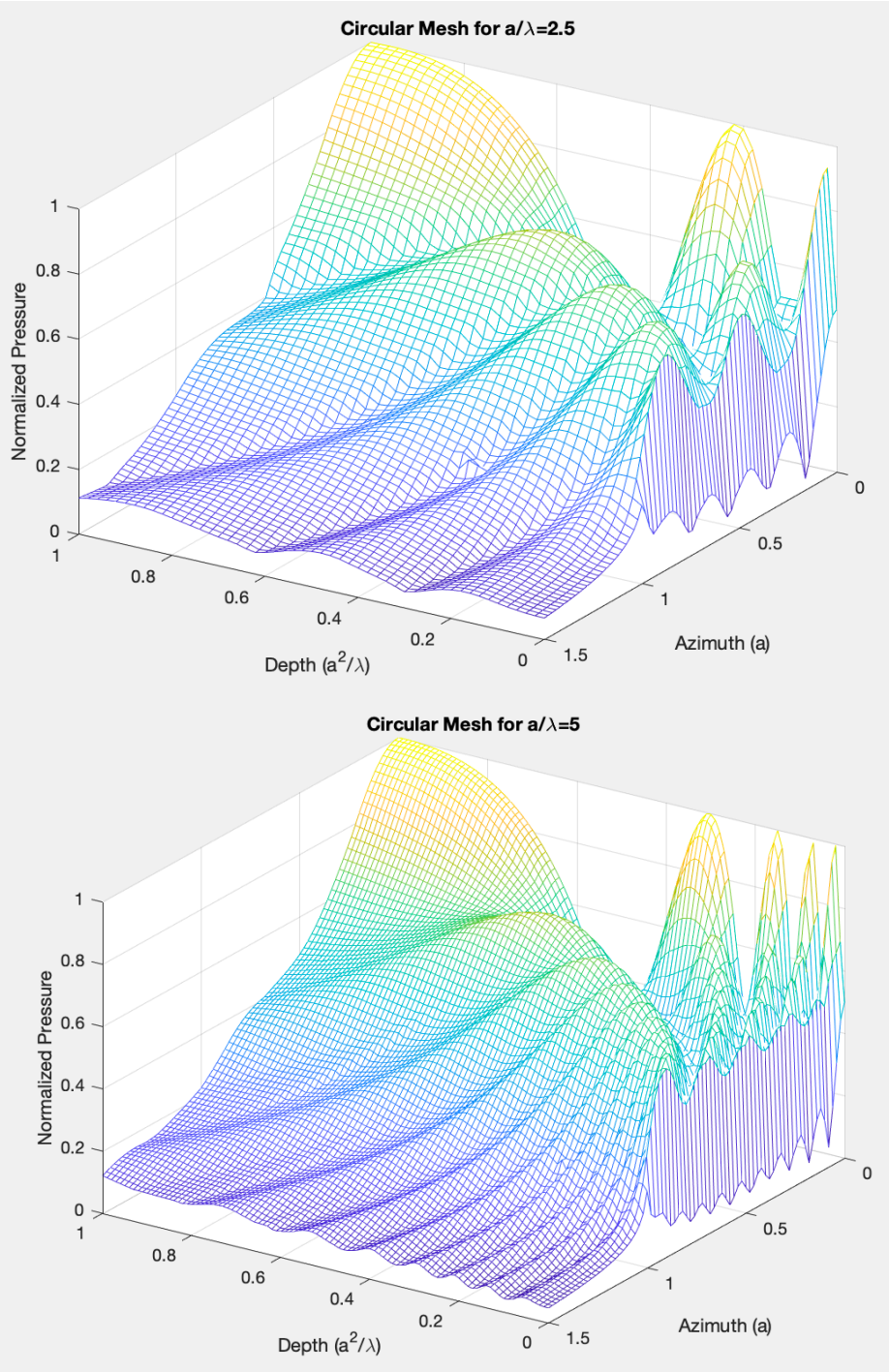




```
%% Circular piston paramaters
c=1540;                 % Speed of sound in body (m/s)
z=25e-3;                % Range (m)
a=5e-3;                 % Piston Radius (m)
frq=1e11;               % Freqency Hz
r=[6,5,4.5,2.5,0.5,0]*1e-3; % Radial distance of point from axis
ts=cell(length(r),1);   % Bin for times within the detection time frame
tcon=ts;                % Bin for times within the h=1 time frame
hs=ts;                  % Bin for transfer fn values
tpre=ts;                % Bin for times before first detection
tpost=ts;               % Bin for times after detection frame
tmax=1.6*(sqrt((z^2)+((a+max(r)).^2))./c);  % End of transfer fn time window
t=zeros(length(r),floor(tmax*frq));         % Finnal time array
h=t;                                        % Finnal transfer functino array
figure
for i=1:length(r)       % Loop through positions
    tff=0;
    %% h calculation
    if a>r(i)           % Define time of first detection if point within piston radius
         tff=z./c;
         tf=sqrt((z.^2)+(r(i)-a).^2)./c;
    else                % Define time of first detection if point out of piston radius
        tf=sqrt((z.^2)+(r(i)-a).^2)./c;
    end
    te=sqrt((z^2)+((a+r(i)).^2))./c; % Define time of last detection
    ts{i}=[tf:(1/frq):te];           % Array of times durring detection
    ts{i}=ts{i}(2:end-1);              % Remove overlap between time cells
    % h calculating line
      hs{i}=(1/pi).*acos(((((c.^2).*(ts{i}.^2))-(z.^2)+(r(i).^2)-(a.^2))...
        ./(2.*r(i).*((c.^2.*ts{i}.^2)-z.^2).^(.5))));
    %% array construction
    if a>r(i)
        tcon{i}=[tff:(1/frq):tf];       % Array of times where detection only in apature
        tpre{i}=[0:(1/frq):tff];        % Array of times before detection
        tpre{i}=tpre{i}(1:end-1);
    else
        tpre{i}=[0:(1/frq):tf];          % Array of times before detection
        tcon{i}=[];
    end

    tpost{i}=[te:(1/frq):tmax];      % Array of times after detection

    % Checks to see if the right number of times exist if not adds another
    % point on the end or removes one
    % (I could get it to work wihtout this but it is more robust this way)

    while length(tpre{i})+length(tcon{i})+length(ts{i})+length(tpost{i}) < length(t(i,:))
        tpost{i}=[tpost{i},tpost{i}(end)+(1/frq)];
    end
    while length(tpre{i})+length(tcon{i})+length(ts{i})+length(tpost{i}) > length(t(i,:))
        tpost{i}=tpost{i}(1:end-1);
    end

    t(i,:)=[tpre{i},tcon{i},ts{i},tpost{i}]; % Combine time cells into finnal array
%% output of transfer function
    h(i,length(tpre{i})+length(tcon{i})+1:length(tpre{i})+length(tcon{i})+length(ts{i}))=hs{i};
% Defines points where h=1
    h(i,length(tpre{i})+1:length(tpre{i})+length(tcon{i}))=1;
    %% plot
    subplot(length(r),1,i)
    plot(t(i,:),h(i,:))
    xlabel('Time (s)')
    ylabel('h(x,t)/c')
    axis([1.6e-5 1.8e-5 0 1.2])
    title(['r/a = ',num2str(r(i)/a)])
end
```

Recreation of figures 7 and 8:

Code is basically the same as above but altered by putting the functional portion into a function and adding a call script to put variables into the function and plot to deal with the greater quantity of data more easily. Code on next two pages.

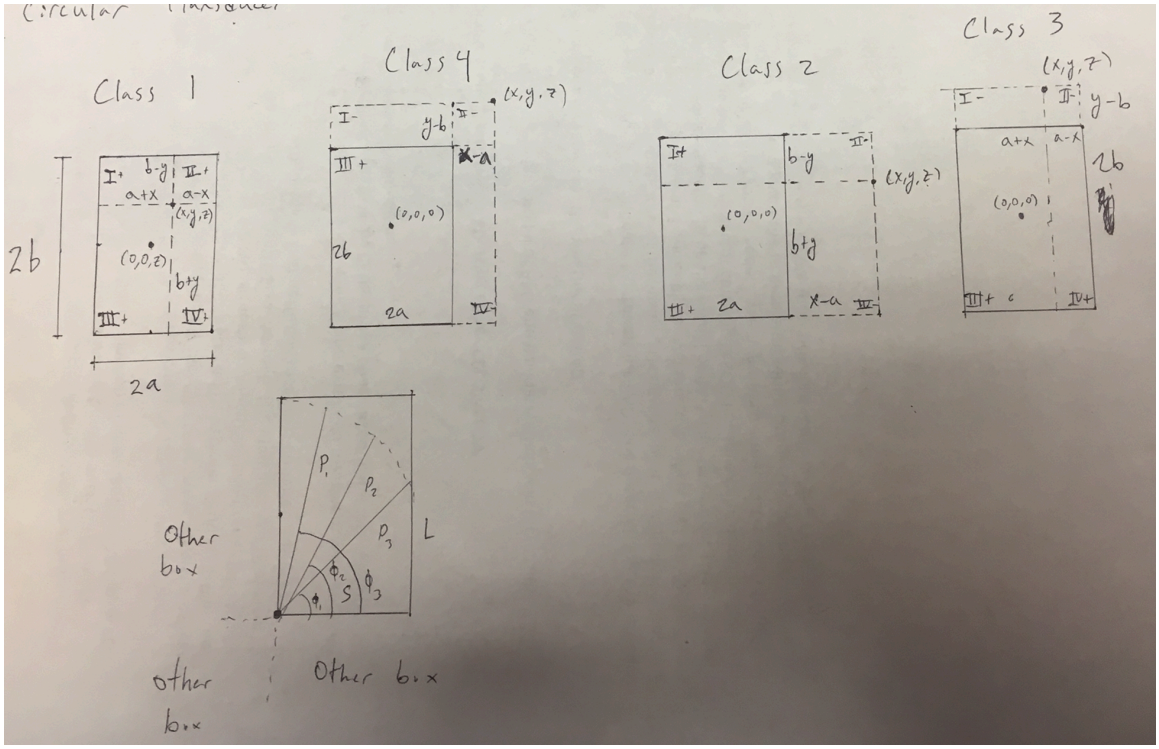


```
function [p]= pressfun(z,r,lam)
%% Circular piston paramaters
r=abs(r);
c=1540;                 % Speed of sound in body (m/s)
a=5e-3;                 % Piston Radius (m)
frq=1e10;               % Freqency Hz
ts=cell(length(r),1);   % Bin for times within the detection time frame
tcon=ts;                % Bin for times within the h=1 time frame
hs=ts;                  % Bin for transfer fn values
tpre=ts;                % Bin for times before first detection
tpost=ts;               % Bin for times after detection frame
tmax=1.6*(sqrt((z^2)+((a+max(r)).^2))./c);  % End of transfer fn time window
t=zeros(length(r),floor(tmax*frq));         % Finnal time array
h=t;                                        % Finnal transfer functino array
for i=1:length(r)       % Loop through positions
    tff=0;
    %% h calculation
    if a>r(i)           % Define time of first detection if point within piston radius
         tff=z./c;
         tf=sqrt((z.^2)+(r(i)-a).^2)./c;
    else                % Define time of first detection if point out of piston radius
        tf=sqrt((z.^2)+(r(i)-a).^2)./c;
    end
    te=sqrt((z^2)+((a+r(i)).^2))./c; % Define time of last detection
    ts{i}=[tf:(1/frq):te];           % Array of times durring detection
    ts{i}=ts{i}(2:end-1);              % Remove overlap between time cells
    % h calculating line
      hs{i}=(1/pi).*acos(((((c.^2).*(ts{i}.^2))-(z.^2)+(r(i).^2)-(a.^2))...
        ./(2.*r(i).*((c.^2.*ts{i}.^2)-z.^2).^(.5))));
    %% array construction
    if a>r(i)
        tcon{i}=[tff:(1/frq):tf];       % Array of times where detection only in apature
        tpre{i}=[0:(1/frq):tff];        % Array of times before detection
        tpre{i}=tpre{i}(1:end-1);
    else
        tpre{i}=[0:(1/frq):tf];          % Array of times before detection
        tcon{i}=[];
    end

    tpost{i}=[te:(1/frq):tmax];      % Array of times after detection

    % Checks to see if the right number of times exist if not adds another
    % point on the end or removes one
    % (I could get it to work wihtout this but it is more robust this way)

    while length(tpre{i})+length(tcon{i})+length(ts{i})+length(tpost{i}) < length(t(i,:))
        tpost{i}=[tpost{i},tpost{i}(end)+(1/frq)];
    end
    while length(tpre{i})+length(tcon{i})+length(ts{i})+length(tpost{i}) > length(t(i,:))
        tpost{i}=tpost{i}(1:end-1);
    end

    t(i,:)=[tpre{i},tcon{i},ts{i},tpost{i}]; % Combine time cells into finnal array
%% output of transfer function
    h(i,length(tpre{i})+length(tcon{i})+1:length(tpre{i})+length(tcon{i})+length(ts{i}))=hs{i};
% Defines points where h=1
    h(i,length(tpre{i})+1:length(tpre{i})+length(tcon{i}))=1;
%% pressure calculation
dh=diff(h(i,:));                % differentiation of transfer function
wav=sin(2*pi*t(i,:)*c./lam);    % creation of sin wave
hwc=conv(dh,wav);               % convolution
prS=floor((length(hwc)./2)-(length(hwc)./5));   % Lower bounding for amplitude check
prE=ceil((length(hwc)./2)+(length(hwc)./5));    % Higher bounding for amplitude check
p=max(hwc(prS:prE))-min(hwc(prS:prE));          % Calculate pressure
end
end
```

```
ilam=1e-3;                          % wavelength
xs=[-7.5:.1:0]*1e-3;                % x field
zs=[0:.25:(25/(ilam*1e3))]*1e-3; % z field
ps=zeros(length(xs),length(zs)); % pressure matrix initialization
for n=1:length(zs)
    n
    for i=1:length(xs)
        ps(i,n)= pressfun(zs(n),xs(i),ilam); % call pressure calculation fn
    end
end
ps=ps./max(max(ps));    % normalize
%% plot
figure;
mesh((zs.*1e3)./(25./(ilam.*1e3)),abs((xs.*1e3)./5),ps)
xlabel('Depth (a^2/\lambda)')
ylabel('Azimuth (a)')
zlabel('Normalized Pressure')
title('Circular Mesh for a/\lambda=5')
```

Work for Rectangular Aperture:

Here is the geometries of the different classes of calculation for transfer function calculation depending on the point of interest's relationship to the aperture.

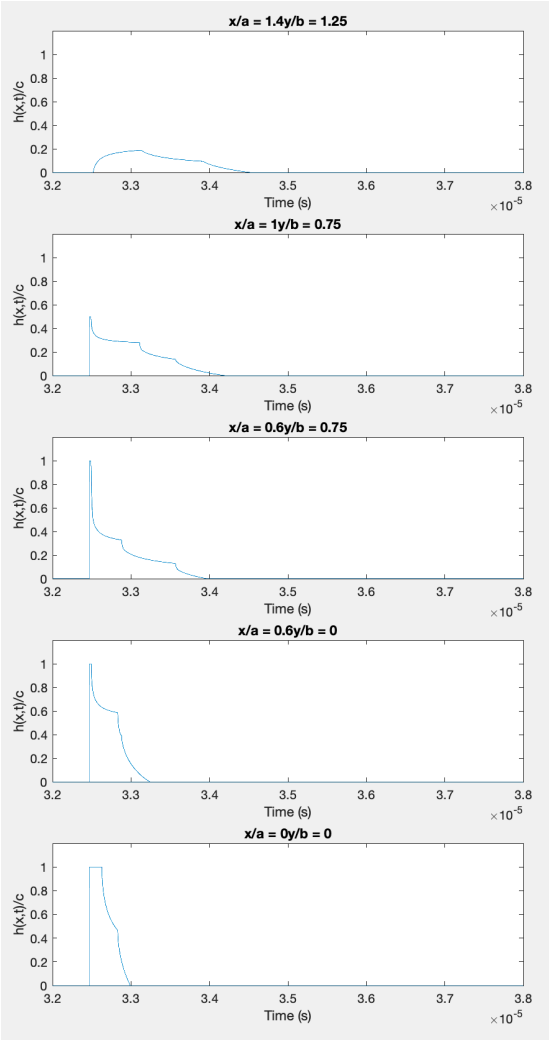


Unlike with the circular aperture the calculations for both the figure 6 and 13 recreations are run through the same function but using different scripts, the function code is on the next two pages.

```
function [h,t,p]=rectapp(x,y,z,lam,tmax)
%% rectangular piston paramaters
c=1540;                 % Speed of sound in body (m/s)
a=5e-3;                 % short axis (m)
b=7.5e-3;               % long axis (m)
xset= zeros(4,1);       % Set of box x axis lengths
yset= zeros(4,1);       % set of box y axis lengths
sset= zeros(4,1);       % Set of box short axis lengths
lset= zeros(4,1);       % set of box long axis lengths
frq=1e10;               % Freqency Hz
critTs=zeros(4);        % Critical time bin
tpre=cell(4,1);   % Bin for times before the detection time frame
toneset=tpre;     % Bin for when h=1
tshort=tpre;      % Bin for short detection before long
tlong=tpre;       % Bin between long and end
tpost=tpre;       % Bin for times after detection time frame
hs=cell(4,5);                   % Bin for transfer fn values
t=zeros(1,floor(tmax*frq));    % Finnal time array
h=t;                            % Finnal transfer functino array
hsq=zeros(4,floor(tmax*frq));  % H components from each rectangle
sign=ones(4,1);
%% Class of position
if abs(x)<=a
    if abs(y)<=b
        class=1;        % Inside apature
    else
        class=2;        % outside in y dimension
    end
else
    if abs(y)<=b
        class=3;        % outside in x dimension
    else
        class=4;        % outside in both dimensions
    end
end
%% Loop through rectangles
for i=1:4
    %% Define regtangle dimensions and sign of opperation
    if class==1
        xset(i)=a-abs(x).*(-1)^i;
        yset(i)=b-abs(y).*(-1)^ceil(i/2);
    end
    if class==2
        xset(i)=a-abs(x).*(-1)^i;
        if i<=2
            yset(i)=abs(y)-b;
            sign(i)=-1;
        else
            yset(i)=2*b;
        end
    end
    if class==3
        yset(i)=b-abs(y).*(-1)^i;
        if i<=2
            xset(i)=abs(x)-a;
            sign(i)=-1;
        else
            xset(i)=2*a;
        end
    end
    if class==4
        if i==1
            yset(i)=2*b;
            xset(i)=2*a;
        end
        if i==2
            yset(i)=abs(y)-b;
            xset(i)=2*a;
            sign(i)=-1;
        end
        if i==3
            xset(i)=abs(x)-a;
            yset(i)=2*b;
            sign(i)=-1;
        end
        if i==4
            yset(i)=abs(y)-b;
            xset(i)=abs(x)-a;
        end
    end
```

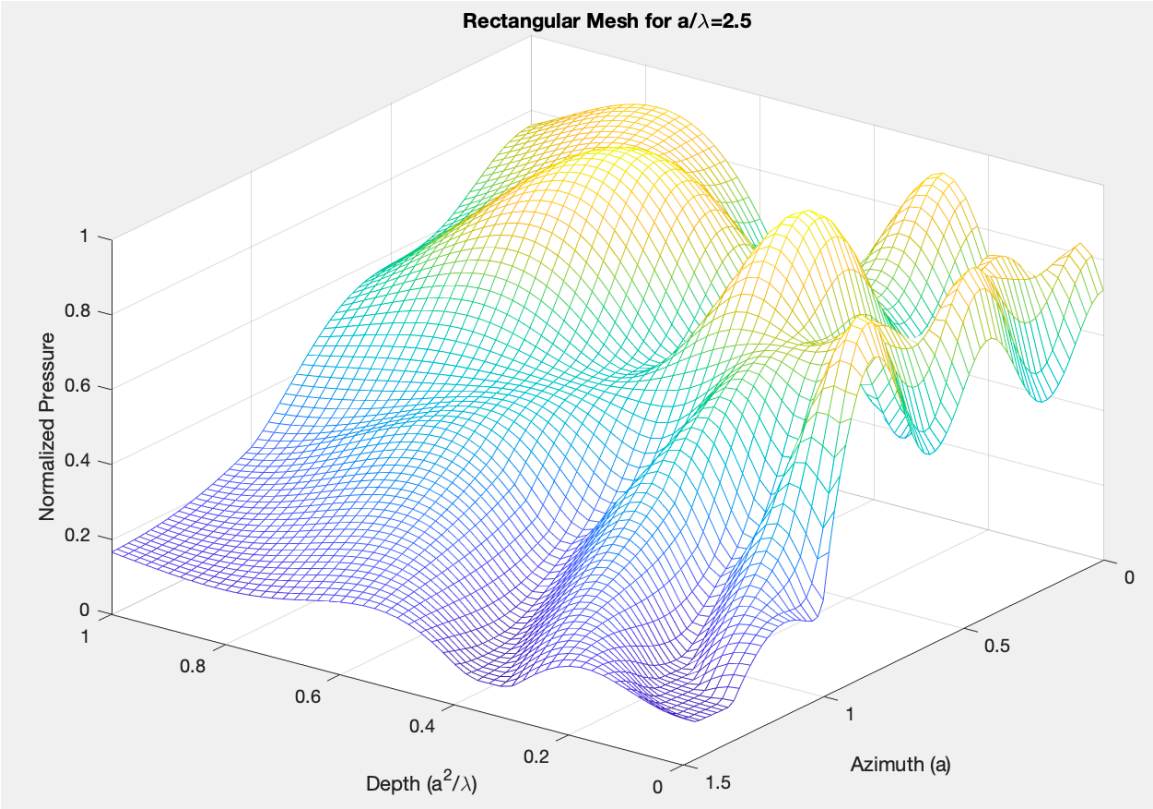
```
    %% Order sets by size
    if xset(i)>yset(i)
        sset(i)=yset(i);
        lset(i)=xset(i);
    else
        sset(i)=xset(i);
        lset(i)=yset(i);
    end
    %% Time set up
    critTs(i,1)=z/c; % Define first critical time
    critTs(i,2)=sqrt((z^2)+(sset(i)^2))/c; % Define second critical time
    critTs(i,3)=sqrt((z^2)+(lset(i)^2))/c; % Define third critical time
    critTs(i,4)=sqrt((z^2)+(sset(i)^2)+(lset(i)^2))/c; % Define fourth critical time
    tpre{i}=[0:(1/frq):critTs(i,1)]; % Times before detection
    toneset{i}=[critTs(i,1):(1/frq):critTs(i,2)]; % Times where responce is one
    toneset{i}=toneset{i}(2:end); % Shorten front to avoid overlap
    tshort{i}=[critTs(i,2):(1/frq):critTs(i,3)]; % Times after short edge is reached
    tshort{i}=tshort{i}(2:end); % Shorten front to avoid overlap
    tlong{i}=[critTs(i,3):(1/frq):critTs(i,4)];% Times after long edge is reached
    tlong{i}=tlong{i}(2:end-1); % Shorten front and end to avoid overlap
    tpost{i}=[critTs(i,4):(1/frq):tmax]; % Times after detection
    %% make sure there are no size artifacts
    while length(tpre{i})+length(toneset{i})+length(tshort{i})...
        +length(tlong{i})+length(tpost{i})>length(t)
        tpost{i}=tpost{i}(1:end-1);
    end
    while length(tpre{i})+length(toneset{i})+length(tshort{i})...
        +length(tlong{i})+length(tpost{i})<length(t)
        tpost{i}=[tpost{i},0];
    end
    %% set of h calculations for each time frame
    hs{i,1}=tpre{i}*0;
    hs{i,2}=ones(1,length(toneset{i})).*(pi/2);
    hs{i,3}=(pi/2)-acos(sset(i)./sqrt((c^2).*(tshort{i}.^2)-(z^2)));
    hs{i,4}=(pi/2)-acos(sset(i)./sqrt((c^2).*(tlong{i}.^2)-(z^2)))...
        -acos(lset(i)./sqrt((c^2).*(tlong{i}.^2)-(z^2)));
    hs{i,5}=tpost{i}*0;

    hsq(i,:)=[hs{i,1},hs{i,2},hs{i,3},hs{i,4},hs{i,5}];

    h=h+(hsq(i,:).*sign(i));
end
t=[tpre{i},toneset{i},tshort{i},tlong{i},tpost{i}];
h=h./(2.*pi);
%% pressure calc
dh=diff(h);               % differentiation of transfer function
wav=sin(2*pi*t*c./lam);   % creation of sin wave
hwc=conv(dh,wav);              % convolution
prS=floor((length(hwc)./2)-(length(hwc)./5));   % Lower bounding for amplitude check
prE=ceil((length(hwc)./2)+(length(hwc)./5));    % Higher bounding for amplitude check
p=max(hwc(prS:prE))-min(hwc(prS:prE));          % Calculate pressure
end
```

Recreation of figure 6:




```
x=[1.4,1,0.6,0.6,0].*5e-3;  % Set of x values
y=[1.25,0.75,0.75,0,0].*7.5e-3; % Set of y values
z=50e-3; % Z value
lam=1e-3; % a lam just so the code will run
frq=1e10; % frequency of sampling
a=5e-3;   % x dimension half width
b=7.5e-3; % y dimension half width
c=1540;   % speed of sound in body
% maxium t we want to calculate values to
tmax=1.2*(sqrt((z^2)+((2*a+max(abs(x))).^2)+((2*b+max(abs(y))).^2))./c);
t=zeros(length(x),floor(tmax*frq)); % bin for times
h=t; % bin for transfer function values
p=zeros(length(x),1); % bin for pressures that we dont use here
%% plot
figure;
hold on
for i=1:length(x)
    [h(i,:),t(i,:),p(i,:)]=rectapp(x(i),y(i),z,lam,tmax); % call function
    subplot(length(x),1,i)
    plot(t(i,:),h(i,:))
    xlabel('Time (s)')
    ylabel('h(x,t)/c')
    axis([3.2e-5 3.8e-5 0 1.2])
    title(['x/a = ',num2str(x(i)/5e-3),'y/b = ',num2str(y(i)/7.5e-3)])
end
```

Recreation of figure 13:




```
ilam=2e-3;                          % wavelength
xs=[-7.5:.1:0]*1e-3;                % x field
zs=[0:.25:(25/(ilam*1e3))]*1e-3; % z field
y=0;                                % y value
frq=1e10;   % frequency of sampling
a=5e-3;     % x axis apature half width
b=7.5e-3;   % y axis apature half width
c=1540;     % speed of sound
% maxium time we want to calculate to
tmax=1.2*(sqrt((z^2)+((2*a+max(abs(x))).^2)+((2*b+max(abs(y))).^2))./c);
ps=zeros(length(xs),length(zs)); % pressure matrix initialization
%% function call
for n=1:length(zs)
    n
    for i=1:length(xs)
        [h,t,ps(i,n)]=rectapp(xs(i),y,zs(n),ilam,tmax);
    end
end
ps=ps./max(max(ps));    % normalize
%% plot
figure;
mesh((zs.*1e3)./(25./(ilam.*1e3)),abs((xs.*1e3)./5),ps)
xlabel('Depth (a^2/\lambda)')
ylabel('Azimuth (a)')
zlabel('Normalized Pressure')
title('Rectangular Mesh for a/\lambda=2.5')
```