Acoustic beamforming and imaging

UC 400L with photoacoustics

Written by Laura A. Cobus, July 2022

Last updated: March 3, 2025

Aims

In this set of experiments, we will use a Matlab simulation to explore how to control, focus, and detect acoustic waves with an ultrasonic transducer array. We will use the (freeware) k-Wave package to simulate the propagation of acoustic waves through the human body. We will then use this knowledge to create an image from real-world data acquired *in vivo* by a medical ultrasound machine, and to explore more sophisticated approaches to wave imaging and characterization.

You are expected to produce a written report on your findings from the numerical simulations. Questions are included in this manual to guide your report writing. Illustrate the report with plots (such as those that you are asked to produce in the instructions) and images/photos of the kwave simulations.

Equipment required

A working installation of Matlab, with the following additional packages: Computational Toolbox.

Contents

1 Introduction and background

Acoustic imaging is a widespread and flexible medical imaging technology. In this technique, acoustic waves are sent into the human body, and 'echos' of the waves bouncing off of tissues and bones are detected. These reflected signals are then used to form an image of the area under investigation.

Most acoustic medical imaging applications use *ultrasonic waves* – acoustic waves with frequencies higher than the range of human hearing. While ultrasound ranges from 20 kHz to several GHz, most medical imaging devices use frequencies in the MHz range. Ultrasound transducers are used to emit and detect ultrasound (See Ref. [1] for a basic introduction to acoustic reflectivity, impedance, and ultrasound imaging.)

For most medical ultrasound imaging, acoustic wave propagation through biological tissues is described as bulk wave propagation through a fluid, in which only longitudinal waves can propagate. The goal is to create a spatial map of the *acoustic reflectivity* R of the medium. R can be defined as the difference in incoming and reflected acoustic pressure at an interface between two fluids:

$$R \equiv \frac{P_i}{P_r}. \tag{1}$$

The value of R is directly related to the acoustic *impedance* Z of each fluid [1],

$$Z = \rho c, \tag{2}$$

where is ρ is the medium density and c the acoustic wave speed. Note that human tissues (muscle, fat, blood, bone, etc.) can generally be considered to be *non-dispersive*, i.e. we can assume that all frequencies travel at the same speed c. Here, we will also neglect any acoustic attenuation (loss due to thermal/viscous effects).

The macroscopic quantity Z can be related to the velocity (ν) of the microscopic particles in the medium and to the pressure by the acoustic Ohm's law:

$$Z = P/\nu. \tag{3}$$

For incidence at zero angle, $\theta_i = 0$, the reflectivity of an interface between two materials is

$$R \approx \frac{Z_2 - Z_1}{2Z_0}, \tag{4}$$

where Z_1 and Z_2 are the impedances of the first and second materials, respectively, and Z_0 is the average medium impedance.

Question 1: In your own words, describe the structure and operation of an acoustic transducer, including an explanation of the piezoelectric effect.

Ultrasonic transducer arrays

In medical ultrasound applications, an *ultrasonic transducer array* is usually preferred; this is a collection of tiny piezoelectric transducers, called *array elements*. Each element is connected to a controlling computer, as sketched in Fig. 1a. The array is placed on the surface of the medium to be imaged; for medical imaging, this is the human skin. The geometry of the system is usually defined as shown in Fig. 1a; depth is axis z, and lateral distance is axis x.

Each element can emit and receive independently. The simplest acquisition sequence with an array is sketched in Figs. 1(b,c): one element emits a short acoustic pulse, which propagates into the medium (Fig. 1b). Immediately afterwards, another array element starts to record (Fig. 1c). The signal that is recorded includes waves which have reflected off of various objects in the medium and arrived at the recording element.

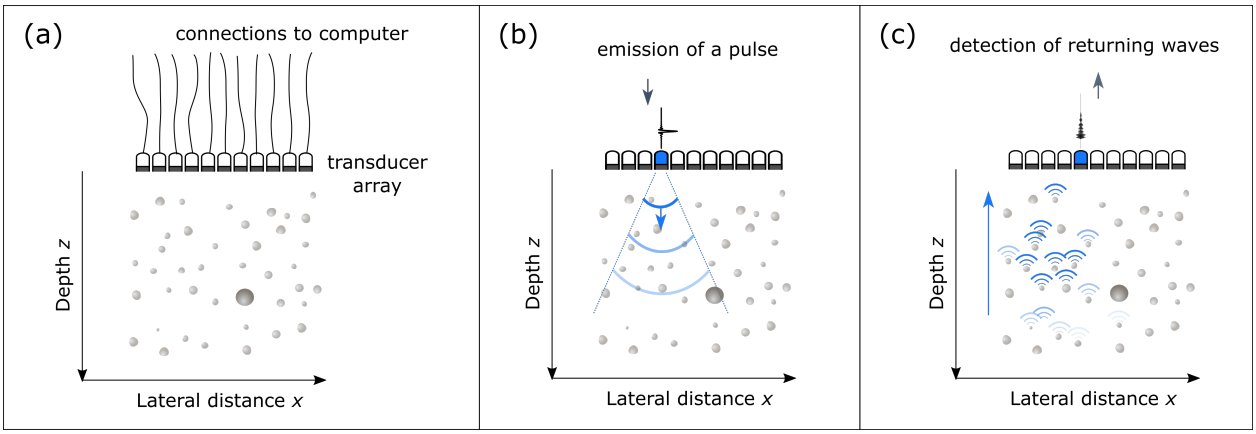

Figure 1: (a) Sketch of a transducer array and typical geometry of an area to be imaged. (b) Emission from one element (c) Detection with one element.

Beamforming

Beamforming is the process of shaping a beam into a desired geometry and direction. While most students will be more familiar with this concept from optics, beamforming is equally possible in acoustics, and shares many of the same basic concepts.

In optics, a beam of light can be shaped as it goes through a lens. The phase of the light is changed by different amounts as different parts of the beam travel different paths through the glass of the lens (see Ref. [3] for more details). This is how we can create, for example, a plane-wave (collimated beam) from a diverging source, or an image from a diverging source. In acoustics, we can do the same type of beamforming by changing the phase of the acoustic waves; in this experiment, we will accomplish this by applying different time delays to different parts of the beam. This analogy is sketched in Fig. 2, and we will see how this works in more detail in Part 2.3 of this experiment.

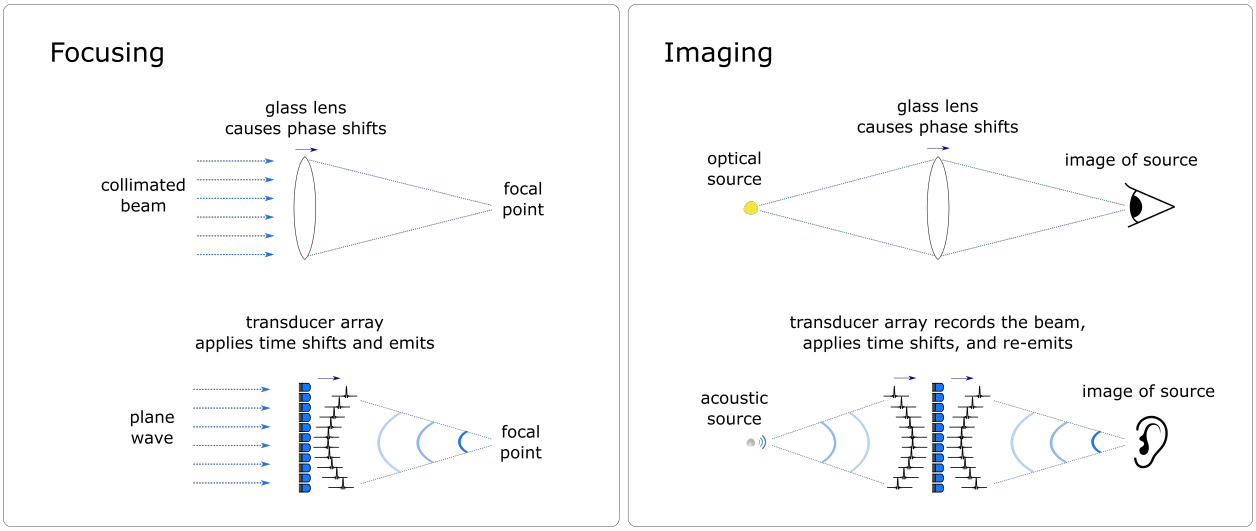

Figure 2: Sketches of the analogies between (a) optical and acoustic focusing, and (b) optical and acoustic imaging.

2 Numerical simulations

Setup

1. Ensure that you have Matlab installed and working on your computer.
2. Download and install the open-source k-Wave package. This can be done either in the Add-On Explorer in Matlab (under the tab "Apps/Get More Apps"), or manually from the matlab website: http://www.k-wave.org/ (follow the directions on the 'install' page of the website).
3. Find the zip folder which contains Matlab scripts for use with this experiment: *AcousticBeamformingLab_scripts.zip*. Unzip this folder - this will be your working directory. Navigate to this folder in matlab. Do not change the scripts in the subfolder 'subfunctions'.

Emission and detection with array elements

For this section, you will be adding to and changing the code in the script *simu_reflection_blank.m*. It is a good idea to save a copy of that file (e.g. *simu_reflection.m*) and make your changes to this new script, so that you can always refer back to the original if you need to.

4. In the first section of the code, add the path to where kwave is installed on your computer:

   ```
   addpath(genpath('folder.kwave'));
   ```

   where `folder.kwave` is the name of the path. (If you installed k-wave via the Add-On Explorer, you can find the installation folder by going back through the Add-On Explorer and clicking the button 'Open Folder'.)

In this script, we define a medium with a density and acoustic wave speed similar to that of soft human tissue. We then place a transducer array just inside the upper surface of the medium, and use it to emit and record ultrasonic signals. Go through the code section by section, reading the comments and making sure you understand each step. In many sections, some code is purposely left out – **you will need to complete the code to perform the required tasks**. The following sections give more detail on working with the code and running the simulation.

Defining the transducer array

The function *define_transducer_array_2D* defines the transducer array. You can type

```
help define_transducer_array_2D
```

into the Command Window to see a basic description of the function, or open it in the editor to see the code. Do not change this function.

5. Examine the resulting `array` structure : you can either type `array` in the command line, or double click on the structure called "array" in the Workspace.

   All information about this array is held in this structure. Note that most units of distance are in metres.

6. The distance between neighbouring array elements is called the *pitch* (Fig. 3a). In the array structure, this is `array.element.pitch`, in [m]. The width of each element is `array.element.width`, in [m].

Question 2: Calculate the total length of the transducer array in millimetres (show your work). Does this agree with the value held in `array.width`?

It is possible to define more than one array – however, in this lab we will just use one array for both emission and reception. This mimics the most common imaging situation in which only one side of the medium is accessible for probe placement (e.g. for medical ultrasound).

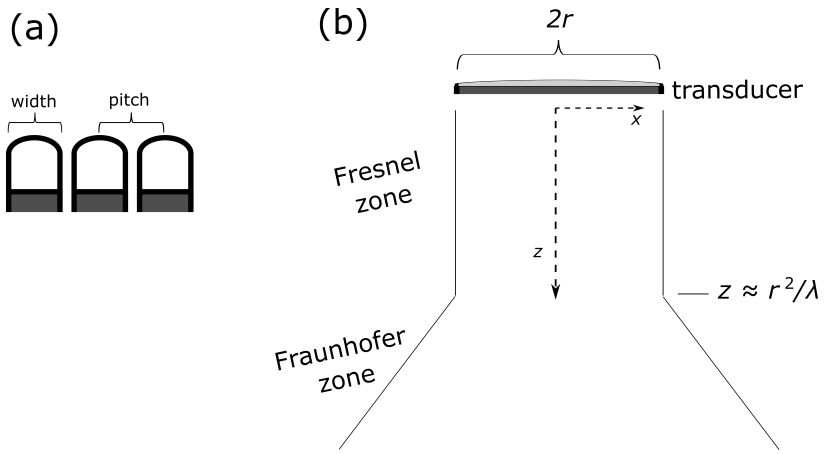

Figure 3: Geometry of (a) the elements of a transducer array, and (b) the acoustic beam resulting from a transducer of diameter $2r$.

Defining the input pulse

We now define the acoustic pulse that each array element can emit. This is done with the function `define_input_pulse.m`. This function does not take any inputs. It returns a structure called `pulse` which holds all of the information on the input pulse. Examine this structure.

Running the simulation

7. In the "DEFINE EMISSION" section, choose a single element to emit.
8. Now that the medium, input pulse and transducer array have been defined, run the simulation:

   ```
   [K, array, ~] = run_simulation(array);
   ```

   Observe how the wave spreads through the sample.
9. Modify your code to emit with more than one element at the same time. What do you observe? ?

Spatial sampling

An important technical consideration is that the grid spatial sampling (the distance in millimetres between subsequent points on the computational grid) should obey the Shannon-Nyquist theorem. That means that, for a particular frequency, you need *at least* two grid points per wavelength for the wave to be able to propagate through the grid. In practice, it's good to allow for at least 5 points per wavelength (this also allows for the existence of signals with higher frequencies than the emission pulse).

10. Add a check like this to the appropriate place in your code, using the code below. There are blanks left for you to fill out, which look like this: ⟨blank⟩.

```
%%% Check that our geometry and emission pulse are ok for simulation
mult = 5;
if(pulse.wavelength_ref<=mult*dy) % check for the lateral direction (y)
    disp(['Error! Wavelength should be >' num2str(mult) ' pts (lateral)'])
        ;
    return
end
if ⟨condition⟩ % check for the lateral direction (x)
    disp(['Error! Wavelength should be >' num2str(mult) ' pts (depth)']);
```

```
    return
end
```

2.1 Simulating acoustic propagation through human tissue

Warning : Be careful about copy-pasting code from this pdf directly into the Matlab editor. Some characters, like the caret ^, may not come out properly, leading to errors when running the code.

11. Add a circular reflective target to the medium. This target will be our (very simplified) model of an object in the human body such as a blood vessel, bone, or cancerous tumour. As these objects have different impedances than the surrounding soft tissue, ultrasound is reflected from their outer surfaces in a way that we can detect and image.

    In your script, add the following lines after the definition of the `medium` structure.

```
%% ADD A CIRCULAR REFLECTIVE TARGET

rx = 11; % target position, depth [mm]
ry = 12; % target position, lateral direction [mm]

r = 0.45*pulse.wavelength_ref*1e3;
% target radius in multiples of wavelength [mm]

refl = 3;  % reflectivity

% converting target position in mm to pts on the grid
rpts = r/dx*1e-3; %[points]
xc = rx/dx*1e-3;  % center of circle x
yc = ry/dy*1e-3;  % center of circle y

% placing the target into the grid
for i = 1:Nx
    for j=1:Ny
        dist = sqrt((i-xc).^2+(j-yc).^2);
        if any(dist < rpts)
            medium.sound_speed(i,j) = ...
                medium.sound_speed(i,j)*refl(dist<rpts);
        end
    end
end
```

12. Run the simulation, emitting with only one element. What do you observe? (Note: if the simulation takes an unreasonably long time to run, you can decrease the number of points in the grid to `Ny = 225; Nx = 189;`).

Visualizing and interpreting results

When the simulation starts, all emission elements emit, and at the same time, all elements start recording. When the simulation has completed, it outputs a matrix `K` which holds all of the time-dependent recorded signals. This matrix has dimensions:

`K(number of detection elements, time).`

13. Clean up `K` with the following:

```
K = K - mean(K,2);  % get rid of any constant offset
```

Plot one of the detected signals (choose any one) as a function of time in μs:

```
figure; clf
plot(⟨time array⟩,⟨detected signal⟩);
```

The signal recorded by the nth detection element, can be extracted from `K` via: 'squeeze(K(n,:))', and the time in μs from

```
t = (kgrid.t_array + pulse.length_s/2)*1e6;  % time [us]
```

Because the emitted pulse has a finite duration (it is not a delta function), we define $t = 0$ as half the input pulse length; `pulse.length_s/2`.

14. You should see two pulses in the detected signal: (1) a pulse corresponding to the emitted pulse that has propagated straight along the array between source and detector, and (2) the reflected signal from the target (you will have to zoom in to see this one). We aren't interested in the first one, so you can get rid of it by completing the following:

```
for i= 1:array.element.num % loop over all detected signals
    t_direct = ⟨⟩; % travel time straight from emitter to detector
    npts_direct = t_direct/pulse.Fs; % convert to # of points
    K(i,1:npts_direct) = 0;
end
```

Note: for real-world transducer arrays, the first arriving pulse has often travelled along the plastic array surface, meaning that it would arrive much sooner than it does in the simulation. Here, you can assume that it travels at speed `medium.sound_speed_ref`. In both Fresnel and Fraunhofer regimes, we can use ray theory to describe the time required for a pulse to travel a certain distance. You can check your calculation by estimating the arrival time of the pulse in your plot above.

15. Now plot the emitted pulse and one of the detected signals in `K` (choose any one) on the same graph, as a function of time in μs. Don't forget to label all of your axes, with units. Include a legend.

    As the pulse spreads in the medium, the energy per unit area decreases and some energy is absorbed and scattered; thus, the detected pulse will be smaller than the input one. For this plot, you can multiply the detected pulse by an arbitrary factor to visualize it on the same vertical scale as the input pulse.

16. Plot a few more detected pulses for various detection elements, and observe that they have different time delays.

Question 3: You can change the reflectivity of the target by varying the parameter `refl`. What effect does this have on the reflected signals that you detect?

Question 4: Write a mathematical expression for how t, the time delay of a detected pulse, depends on z, the depth of the target, and $x - x_0$, the lateral distance between input and output elements. Choose one of the signals on your plot. From the time delay of the reflected pulse, estimate the depth of the target.

Plotting intensity

While `K` contains the amplitude of the detected acoustic fields, ultrasound images display reflected acoustic *intensity*. This intensity is usually a relative intensity, such as I/I_0, where I and I_0 are the intensities of the detected and source acoustic fields, respectively. Another option, **which we will use here**, is to plot I/I_{max}, where I_{max} is the maximum detected intensity.

Ultrasound images typically display acoustic **intensity** on a **decibel (dB) scale**, which enables a larger range of detected intensities to be displayed [1].

17. Convert `K` to decibels of relative intensity (see Ref [1]):

```
K_dB = 20*log10(abs(K).^2/max(abs(K(:))).^2);
```

18. Make a 2D visualization of *all* of the recorded signals using the Matlab function **imagesc**:

```
x = array.element.lateral*1000; % lateral distance [mm]
figure;
imagesc(x,t,abs(squeeze(K_dB)).');
```

Add a colorbar, and label all axes appropriately.

Question 5: Are any of the features of this 2D plot described by your mathematical expression for t?

Tissue microstructure

In reality, soft human tissue varies in composition at a small scale – that is, a scale smaller than our ultrasonic wavelength. To better simulate the propagation of ultrasound in soft human tissue, we can add some random variations in sound speed to our medium.

19. In your script, add the following lines just before the target is added:

```
%% ADD NOISE

noise = wgn(Nx,Ny,1);
noise = (noise/max(noise(:)))*medium.sound_speed_ref*0.15;
medium.sound_speed = medium.sound_speed+noise;
```

Observe what happens when you run the simulation. What effect do these tiny scatterers have on the propagating waves? (Hint: it might help to consider your previous observations using the circular target.)

Frequency content

For wave control and imaging, it is important to have an understanding of the Fourier series and Fourier transform. Reference [5] provides a good overview of the Fourier transform (and there are many other good references on these subjects). In this lab, we will use the Fourier transform to examine the frequency content of the signals that we produce.

20. Compute the Fourier transform of your input pulse using the built-in matlab function "fft", and plot it as a function of frequency in MHz:

```
% input pulse
signal = squeeze(pulse.signal(array.element.emission,:));
Nt     =  pulse.signal_length;  % signal length [pts]
Dt     =  Nt/pulse.Fs*1e6;  % signal length [us]
freq = (0:(Nt-1))/Dt; % [MHz]

pulse.signal_fft = abs(fft⟨⟩,[],2);

figure;
plot(⟨frequency⟩,⟨frequency-domain signal⟩,'ko-');
```

Question 6: Why are there two peaks in your plotted spectra?

21. Zoom into the lower-frequency of the two peaks:

```
xlim([0 10]);
```

22. Check that the central frequency of the peak corresponds to that which you defined for your input pulse, earlier in the script. What parameter of your input pulse can you change so that this peak in the frequency spectrum is narrower or wider, while still centered around the same central frequency?

23. Now do the same analysis of one of your signals detected by the array:

```
% detected signal
signal_detected = squeeze(K(array.element.emission(1),:));
Dt = max(t); % [us]
freq = (0:(kgrid.Nt-1))/Dt; % [MHz]

signal_fft = fft(signal_detected);
```

Plot the resulting spectrum on the same graph as that of the input pulse. Make sure to label all of your axes, with units, and include a legend.

Question 7: Why are there more points in the frequency spectrum of the detected signal than that of the emitted signal?

Frequency filtering

We can use the `fft` function to filter out frequencies that we don't want in our signals. Here, we will introduce a new structure 'Img' which holds the parameters for data analysis and image reconstruction.

24. Filter one of the detected signals using a Hanning window [7]:

```
Img.fc = ⟨⟩;  % [Hz] central frequency of input pulse
Img.freq_lim = [0.1 7];  % [MHz]

idxFreq=find(freq>=Img.freq_lim(1) & freq<=Img.freq_lim(2));
Nfkeep=length(idxFreq);

% Hanning Filter on selected frequency
HANN=zeros(kgrid.Nt,1);
HANN(idxFreq)=hann(Nfkeep);

signal_fft_filt = signal_fft.*HANN;
```

Question 8: Why is the result complex?

Plot the filtered signal on the same graph as the the input pulse and unfiltered detected signal. Also plot the filter function (HANN). Try to get them all to aproximately the same amplitude so you can nicely see the effect of the filter.

25. Apply the frequency filter to the entire `K` matrix, and then take the inverse Fourier transform:

```
% Apply Hanning Filter to entire dataset
Kfft = fft(K,[],2);
Kfft=Kfft.*HANN.';

% Go back into the time domain
Kfilt=ifft(Kfft,[],2);
```

26. Plot the resulting filtered signal in the time domain with the input pulse and unfiltered detected signal. Update the legend.

Time-domain filtering

27. Now we will apply a time-domain filter, just to clean up any possible stray signals at the beginning or end of the time period. This filter is:

```
TimeFilt = tukeywin(kgrid.Nt,0.025);
```

28. Apply the filter to the entire `Kfilt` matrix and plot the result with the other time-domain signals. Include the filter function on this plot as well. Update the legend.

2.2 A-mode imaging

For some applications, ultrasound imaging machines produce 'A-mode' images. These are plots of depth-dependent reflected intensity $I_{x_i}(z)$ for a particular lateral position x_i, created by emitting and receiving with the same element (or a single ultrasonic transducer).

We will use the standard convention that depth is the vertical distance downwards from the array (so that the array is at depth $z = 0$).

29. Convert the time array to depth z. while we will define depth in terms of distance from the array . To convert time to depth, use the following:

```
z = t/1e6*medium.sound_speed_ref/2*1000; % [mm] depth
```

30. Plot an A-mode image of your target, in dB of intensity. Verify that your target appears at the correct depth.

    Keep in mind that, while we have defined depth in terms of distance from the array, kWave defines its computational grid from negative to positive values. You have set your target at a distance `rx` from the top of the computational grid. You can find the position of the top of the computational grid from `kgrid.x_vec(1)`, and the position of the transducer array on the computational grid in `array.element.depth(1)`.

2.3 Beamforming

This is a good point to save a new version of your script to continue on with (we will now change the script substantially).

Beamforming in emission

Focused beamforming

Now, we will use the transducer array to emit focused beams. By calculating the appropriate time delays to apply to the acoustic waves, you can precisely control the spatial location of the focus.

31. Go back to emitting with only three elements. Modify your script to add a time delay to some of the elements, using the following lines:

```
delays = [3 0 2]*1e-6; % [s]

% set delays to zero for non-emitting elements
array.delays=zeros(1,array.element.num);
array.delays(array.element.emission) = delays;

[K, array, ~] = run_simulation(array, [], true);
```

32. Using your expression for t (Question 5), adjust the time delays such that the wavefronts arrive at the target at the same time.

33. Now, emit with *all* emission elements, and adjust all time delays such that all waves are focused onto the target. Make a 2D plot of `K_dB` as before.

Question 9: What do you see in your 2D plot after this focusing at the target? How does this relate to what you observed in the previous simulations using two transducer arrays?

Plane wave beamforming

You can also use the transducer array to emit plane waves!

34. Emit with all of the elements, with zero delay. This approximates emission from a single transducer whose width is the width of the transducer array. If the array had infinite width and almost zero pitch, the emitted beam would be an ideal plane wave; our finite-sized transducer array only emits approximate, quasi-plane waves. For the purposes of this experiment, however, we will refer to these beams as plane waves.

35. Now emit a plane wave which is angled 20 degrees from the axis of the transducer array. Consult Ref. [4] for help on how this can be done.

2.4 B-mode Imaging

You have already seen that the acoustic analogy of an optical focusing lens is the application of time delays which 'curve' the outgoing beam into a focused beam. Now, we want to do the same thing with the waves that are returning from the target: if we apply time delays and add up the delayed signals, we form an 'image' of the medium just at the target (Fig. 2) position, $(x_{\text{target}}, z_{\text{target}})$. We want to create a full 2D picture of the entire medium – called a *Bmode image*. We thus need to perform this delay-and-sum process for each point (x, z) in the medium. In other words, we need to add up the waves which have travelled from the emission element to a particular point (x, z) in the medium and back to a detection element at position (x_0, z_0). And we need to do this *for each point* (x, z), and for all j detection elements:

$$\text{Image}(x, z) = \sum_j K\left[i, j, \tau(x_i, z_i; x, z; x_j, z_j)\right], \tag{5}$$

where $\tau(x_i, z_i; x, z; x_j, z_j)$ holds the travel time required to go from a the source array element at (x_i, z_i) to point (x, z) in the medium and back to a detection array element at (x_j, z_j).

36. Go back to emitting with just one element (and recording with all elements)

37. Write a script to create an image of the simulated medium using Eq. 5. This can be done with the following structure (fill in the blanks and add lines as you need):

```
% initialize the array I(x,z) : define an array of zeros with size [x,z]
img_data = zeros(⟨⟩);

for xx = 1:array.nb_sources  % loop over detection element lateral
    position
     for zz = 1:⟨⟩  % loop over image pixels in depth
         R = squeeze(K(⟨⟩,⟨⟩ ));

         transmit_distance = ⟨⟩ ; % [m]
         receive_distance =  ⟨⟩ ; % [m]
```

```
        time_delay = ⟨⟩ ; %  [s] total time delay

        % apply the time delays to the data
        tmp = interp1(t,R,time_delay);

        tmp(isnan(tmp))=0;  % Avoid NaNs

        img_data(zz,:) = img_data(zz,:) + tmp;
    end

    % you can watch how the image forms with each subsequent detection:
    imagesc(⟨⟩, ⟨⟩, abs(img_data))
    ylabel('depth (mm)')
    xlabel('lateral position (mm)');
    title(['Detection ' aa])
    drawnow
end
```

38. Plot the image in dB of intensity.

If your calculations were correct, the resulting image (viewed with `imagesc`) should display a spot in the exact same place as the target. However, the spot may be distorted compared with the circular target; this is normal, and due to the limited information that can be obtained by emitting with just one array element. The image should also display a grainy, noisy quality which is a result of the variations in the tissue microstructure. This effect is called ***speckle*** [8], and is extremely common in ultrasound imaging (and many other types of wave imaging).

2.5 Full matrix capture

39. Modify your script to loop over emission elements, emitting with one array element at a time. If you want to speed up the simulation slightly, you can set the optional input `'PlotSim'` to `false`, i.e. run:

    ```
    [K, array, ~] = run_simulation(array,[],false);
    ```

    Store the results in a 3-dimensional `K` matrix. This type of dataset is often called a *full matrix capture*.

40. Save the entire dataset $K(u_i, u_j, t)$ as a .mat file on your computer (so that, if you need it again or accidentally delete it, you don't have to run the entire set of simulations again). Include the associated structures, e.g.

    ```
    save('Kuu.mat','K','array','pulse','pml','medium','kgrid','folder');
    ```

41. Apply the frequency- and time-domain filtering from the previous section to `K`.

42. Make the image. Now, the equation for the image is:

$$\text{Image}(x,z) = \sum_i \sum_j K\left[i, j, \tau(x_i, z_i; x, z; x_j, z_j)\right], \tag{6}$$

    where $\tau(x_i, z_i; x, z; x_j, z_j)$ holds the travel time required to go from a source array element at (x_i, z_i) to point (x, z) in the medium and back to a detection array element at (x_j, z_j).

    For each emission, plot the resulting image $I(x, z)$ (in dB), *and* the compounded image, summed over all previous emissions.

43. Run the script and observe the results; the compound image should improve with each emission, and converge towards an accurate representation of your simulated medium.

2.6 The matrix approach to imaging

In acoustic imaging, the ideal measurement would be one in which a point source is placed at a spot $\mathbf{r_{in}}$ inside the medium, and a point detector measures the acoustic response at that same spot, or at a point $\mathbf{r_{out}}$ elsewhere in the medium. The entire measured set of these scattering processes can be denoted $\mathbf{\Gamma_{rr}} = [\Gamma(\mathbf{r_{out}}, \mathbf{r_{in}}, t)]$, where subscripts **in** and **out** denote emission and detection of wavefields. For non-invasive imaging, however, sources and detectors are generally confined to the outside of the medium; for seismology, this is the Earth's surface, and for medical imaging, the skin. Then, the experimental dataset typically consists of a matrix of responses which is measured with a sensor array in contact with the system surface. While this matrix can be measured in a transmission geometry, here we concentrate on the *reflection matrix* (source and detector on the same side), $\mathbf{R}$, which is the more practical choice for non-invasive imaging. This dataset is conveniently described using a matrix formalism.

Earlier in this lab, we saw that one can calculate the Bmode image from $\mathbf{R}$ (also called `K` in the numerical simulations sections). Now, we go further. The general idea behind the matrix approach for imaging is to extract, from experimentally-measured data such as $\mathbf{R}$, 'hidden' information about the system under investigation. This information is often uncovered by mathematically decomposing the matrix $\mathbf{R}$ into subspaces, which are then each identified with a particular process or interaction between propagating waves and the medium. With this new information on the acoustic propagation in the medium, a better model of $\mathbf{T}$ can be achieved, and thus a more accurate image. Historically, the matrix formalism was first applied to acoustic imaging in order to extract the set of signals which would focus perfectly at a certain object [8] – also called *focusing functions* in seismology. Over time, other related methods were developed to enable the extraction of different information, enabling more advanced imaging.

The simplest way to acquire $\mathbf{R}$ is in the so-called *canonical basis*: emitting with one transducer at a time, and for each emission recording with all elements the time-dependent backscattered field [8]. Measured in this way, we will denote $\mathbf{R}$ as $\mathbf{R_{uu}}(t) = [R(\mathbf{u_{in}}, \mathbf{u_{out}}, t)]$, where t is time. The positions of the sensor array elements are given by vector $\mathbf{u} = (z_0, x_0)$ for a linear (1D) sensor array.

2.6.1 The focused reflection matrix

So far, we have considered $\mathbf{R}$ in the context of two geometrical bases:

- The *transducer basis* or *canonical basis* $\mathbf{u}$, and
- the *focussed basis* $\mathbf{r} \equiv (x, z)$ that we create when imaging (Eq. 6) [(1)].

The matrix approach essentially uses beamforming to project our matrix $\mathbf{R}$ between these bases to gain additional information.

Here, we concentrate on the *focused reflection matrix* $\mathbf{R_{rr}} = [R(\mathbf{r_{in}}, \mathbf{r_{out}})]$, where $\mathbf{r_{in}} \equiv (x_{\text{in}}, z_{\text{in}})$ and $\mathbf{r_{out}} \equiv (x_{\text{out}}, z_{\text{out}})$ are points *inside the medium* [9,10]. $\mathbf{R_{rr}}$ simulates a (theoretical) situation in which we have a grid of transducer elements distributed throughout the medium under examination. This matrix is created by projecting $\mathbf{R_{uu}}$ into the *focussed basis* $\mathbf{r}$ both emission and reception. This projection is essentially Eq. 6, but done separately for $\mathbf{r_{in}}$ and $\mathbf{r_{out}}$. This is expressed mathematically by Eq. 1 in Ref. [12].

44. Open Ref. [12] and find Eq. 1. Read some of the text so that you understand this equation and how it relates to what we are talking about in this section. Do not worry if you don't understand the full article – just use it to better understand this section of the lab.

For simplicity, we will assume in all of our analysis that $z_{\text{in}} = z_{\text{out}}$ (as is also done in Ref. [12]). Thus, we are computing

$$\mathbf{R}_{xx} \equiv R(x_{\text{in}}, x_{\text{out}}, z_{\text{in}} = z_{\text{out}}) = R(x_{\text{in}}, x_{\text{out}}, z). \tag{7}$$

[(1)] Note that although the sets of $\mathbf{u}$ or $\mathbf{r}$ that we use here are not necessarily linearly independent or may not form a complete basis, we will refer to these as 'bases' for simplicity

45. Open the function *Rxx_focusing.m* in the folder 'old subfunctions'. Make sure that you understand what it is doing.
46. This function is an old version that will not work immediately with your code. Modify the function so that it works with your numerical simulation code, and put this working version into your 'subfunctions' folder.
47. Numerically simulate a $\mathbf{R_{uu}}$ (or load a previously-saved one). From that, compute $\mathbf{R}_{xx}$.
48. Examine $\mathbf{R}_{xx}$ by plotting 'slices' of it at various depths z. Compare what you see with the behaviour described in Fig. 2(a,b) of Ref [10].
49. In principle, the conventional Bmode image can be reconstructed from the diagonal elements of $\mathbf{R}_{xx}$. That is, $I(x,z) = R(x_{\text{in}} = x_{\text{out}}, z)$. Reconstruct the Bmode image in this way, and compare with the Bmode image constructed from Eq. 6.

2.7 Photoacoustic imaging

2.7.1 Aberration correction – literature review

1. Do a literature review on aberration correction in both acoustic and photoacoustic imaging. Create a short summary of the basics of both (one page maximum). What are the similarities and differences? Make sure to include references.
2. To your literature review, add another paragraph concentrating on the effect of the source (the thermoacoustic effect) in photoacoustic aberration correction.

2.7.2 Numerical simulations

Coming soon...

A Technical considerations

A.1 Computational grid size

1. Calculations will be much faster if the grid dimensions have small prime number factors. Use the following to find better grid dimensions:

```
checkFactors(min(Ny,Nx)-100,max(Ny,Nx)+100);
close gcf   % the above automatically makes a graph that's not useful
disp(' ');disp(['Current grid size: ' num2str(Ny) ' by ' num2str(Nx)]);
```

A.2 Perfectly matched layer (pml)

Read about this on the kwave documentation page, or other references on numerical simulations. If you see the propagating wave duplicated at opposite edges of the computational grid, or reflecting off of the edge, then you may have to change the pml properties.

References

[1] K. Forinash and W. Christian, https://phys.libretexts.org/ (2020).

Chpt. 6: **Scalar diffraction optics** (diffraction, resolution, Fresnel and Fraunhofer approximations)
Chpt. 7.1.3: **The Decibel scale**
Chpt. 17.1: **Ultrasound and impedance**

[2] J. Goodman, "Introduction to Fourier Optics" 2nd Ed., McGraw-Hill (1996).

[3] E. Hecht, "Optics, Global Edition", 5th Ed. Pearson (2017).

[4] G. Montaldo, M. Tanter, J. Bercoff, N. Benech and M. Fink, "Coherent Plane Wave Compounding for Very High Frame Rate Ultrasonography and Transient Elastography", IEEE T. Ultrason. Ferr. **56**:3 (2009).

[5] J. Tsui and C.-H. Cheng, "Digital techniques for wideband receivers" 3rd Ed., Scitech Publishing (2015). Chapter 3: **Fourier Transform and Convolution**.

[6] V. Perrot, M. Polichetti, F. Varray, and D. Garcia, "So you think you can DAS? A viewpoint on delay-and-sum beamforming", Ultrasonics **111**:106309 (2021).

[7] Wikipedia, "Window function", https://en.wikipedia.org/wiki/Window_function. Accessed 26-oct-2023.

[8] Szabo, T. L., “Diagnostic Ultrasound Imaging, Inside Out”, 4th Ed. Elsevier (2014).