Biomedical Ultrasound: Fundamentals of Imaging and Micromachined Transducers

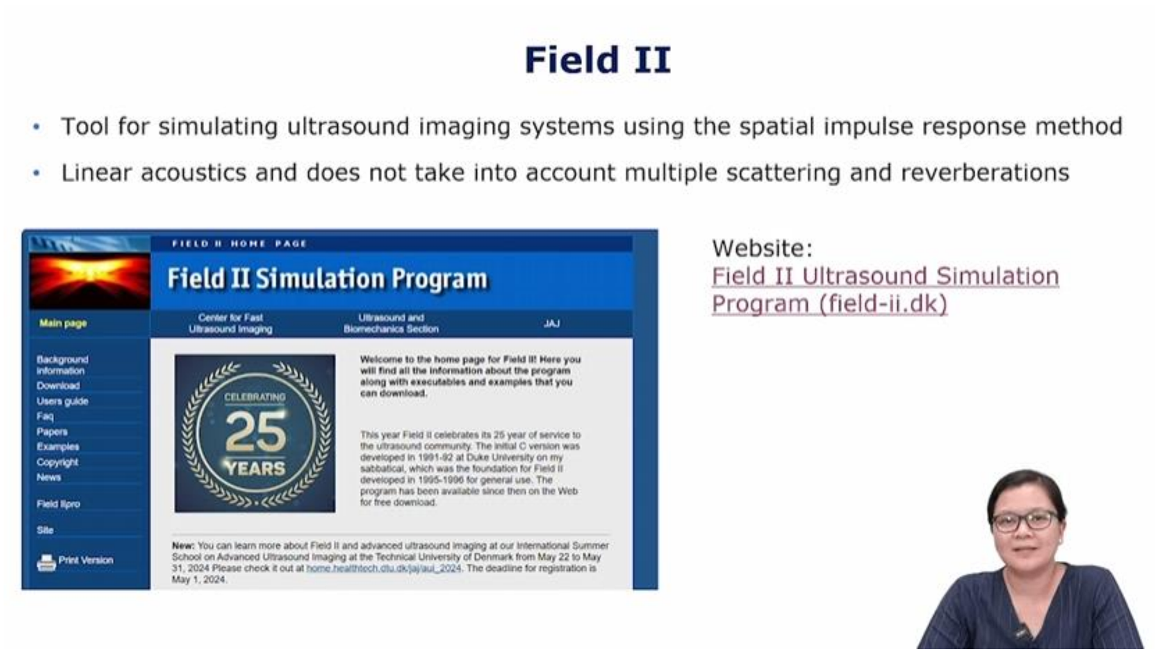
Prof. Karla P. Mercado-Shekhar, Prof. Himanshu Shekhar, Prof. Hardik Jeetendra Pandya

IIT Gandhinagar, IISc Bangalore

Lecture: 34

Intro to Field II simulation - impulse response, beam pattern

Hello, welcome to today's lecture. Today I will be introducing a simulation program called Field II. Field II is a tool that is used for simulating ultrasound imaging systems using the spatial impulse response method. It is governed by linear acoustics theory and therefore nonlinear wave propagation is not incorporated in this program. It also does not take into account multiple scattering and as well as reverberations. So here is a website that includes all the information you can find with the Field II simulation program.



Field II

- Tool for simulating ultrasound imaging systems using the spatial impulse response method
- Linear acoustics and does not take into account multiple scattering and reverberations

Welcome to the home page for Field II! Here you will find all the information about the program along with executables and examples that you can download.

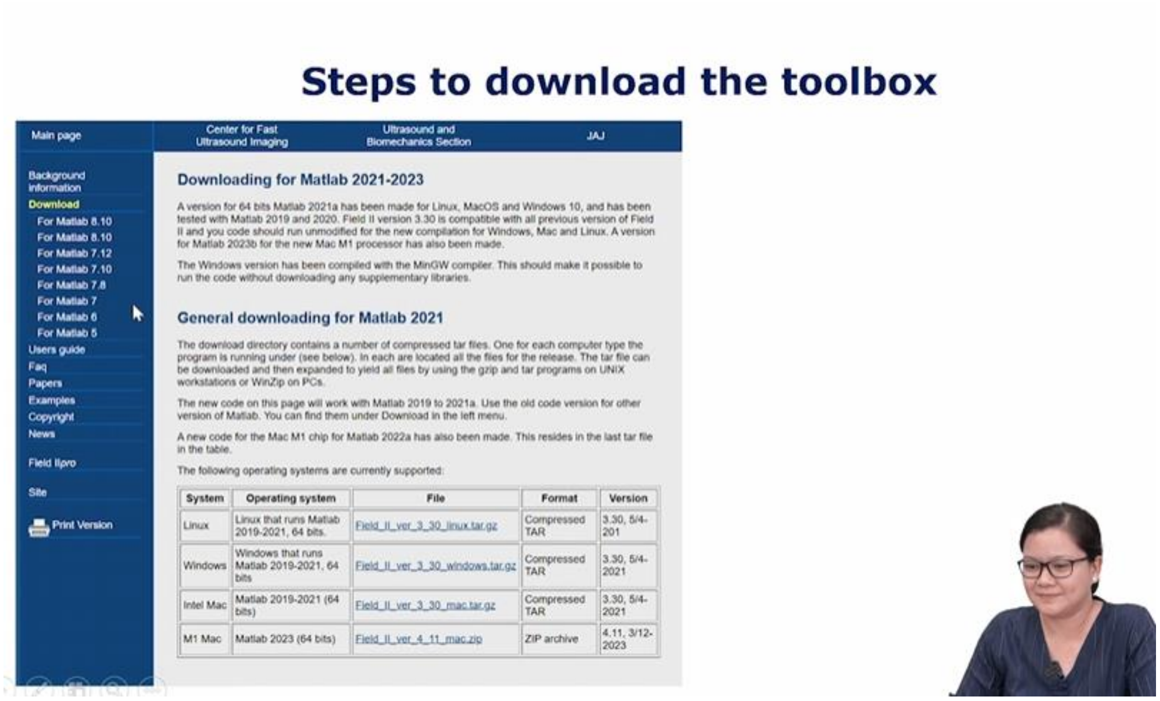
This year Field II celebrates its 25 year of service to the ultrasound community. The initial C version was developed in 1991-92 at Duke University on my sabbatical, which was the foundation for Field II developed in 1995-1996 for general use. The program has been available since then on the Web for free download.

New: You can learn more about Field II and advanced ultrasound imaging at our International Summer School on Advanced Ultrasound Imaging at the Technical University of Denmark from May 22 to May 31, 2024 Please check it out at home.healthtech.dtu.dk/jaj/aui_2024. The deadline for registration is May 1, 2024.

Website:
Field II Ultrasound Simulation Program (field-ii.dk)

You can find it in field-ii.dk. And this website contains a lot of information incorporating the background, a user guide, as well as several examples that can be implemented using this program.

Now here are some steps on how to get started.



Steps to download the toolbox

Downloading for Matlab 2021-2023

A version for 64 bits Matlab 2021a has been made for Linux, MacOS and Windows 10, and has been tested with Matlab 2019 and 2020. Field II version 3.30 is compatible with all previous version of Field II and you code should run unmodified for the new compilation for Windows, Mac and Linux. A version for Matlab 2023b for the new Mac M1 processor has also been made.

The Windows version has been compiled with the MinGW compiler. This should make it possible to run the code without downloading any supplementary libraries.

General downloading for Matlab 2021

The download directory contains a number of compressed tar files. One for each computer type the program is running under (see below). In each are located all the files for the release. The tar file can be downloaded and then expanded to yield all files by using the gzip and tar programs on UNIX workstations or WinZip on PCs.

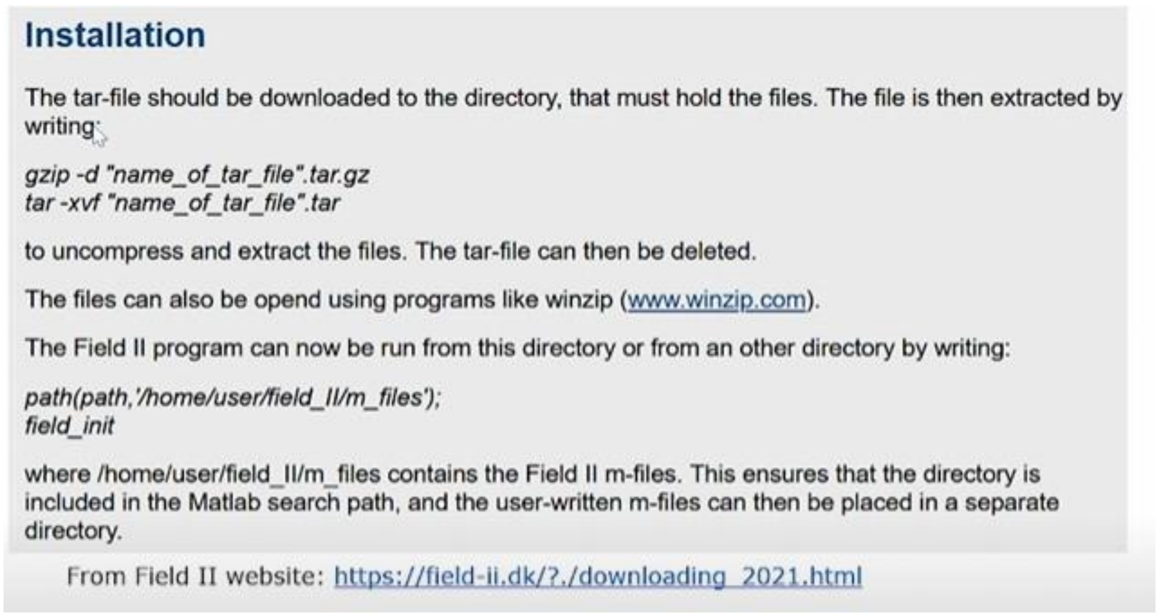
The new code on this page will work with Matlab 2019 to 2021a. Use the old code version for other version of Matlab. You can find them under Download in the left menu.

A new code for the Mac M1 chip for Matlab 2022a has also been made. This resides in the last tar file in the table.

The following operating systems are currently supported:

System	Operating system	File	Format	Version
Linux	Linux that runs Matlab 2019-2021, 64 bits.	Field_II_ver_3_30_linux.tar.gz	Compressed TAR	3.30, 5/4-201
Windows	Windows that runs Matlab 2019-2021, 64 bits	Field_II_ver_3_30_windows.tar.gz	Compressed TAR	3.30, 5/4-2021
Intel Mac	Matlab 2019-2021 (64 bits)	Field_II_ver_3_30_mac.tar.gz	Compressed TAR	3.30, 5/4-2021
M1 Mac	Matlab 2023 (64 bits)	Field_II_ver_4_11_mac.zip	ZIP archive	4.11, 3/12-2023

First, you have to download the toolbox from the Field II simulation website. And here on the left menu bar, there is a download link here you can download different versions of the program depending on your MATLAB version. The Field II website also has instructions on how to install the program.



Installation

The tar-file should be downloaded to the directory, that must hold the files. The file is then extracted by writing:

```
gzip -d "name_of_tar_file".tar.gz
tar -xvf "name_of_tar_file".tar
```

to uncompress and extract the files. The tar-file can then be deleted.

The files can also be opend using programs like winzip (www.winzip.com).

The Field II program can now be run from this directory or from an other directory by writing:

```
path(path,'/home/user/field_II/m_files');
field_init
```

where /home/user/field_II/m_files contains the Field II m-files. This ensures that the directory is included in the Matlab search path, and the user-written m-files can then be placed in a separate directory.

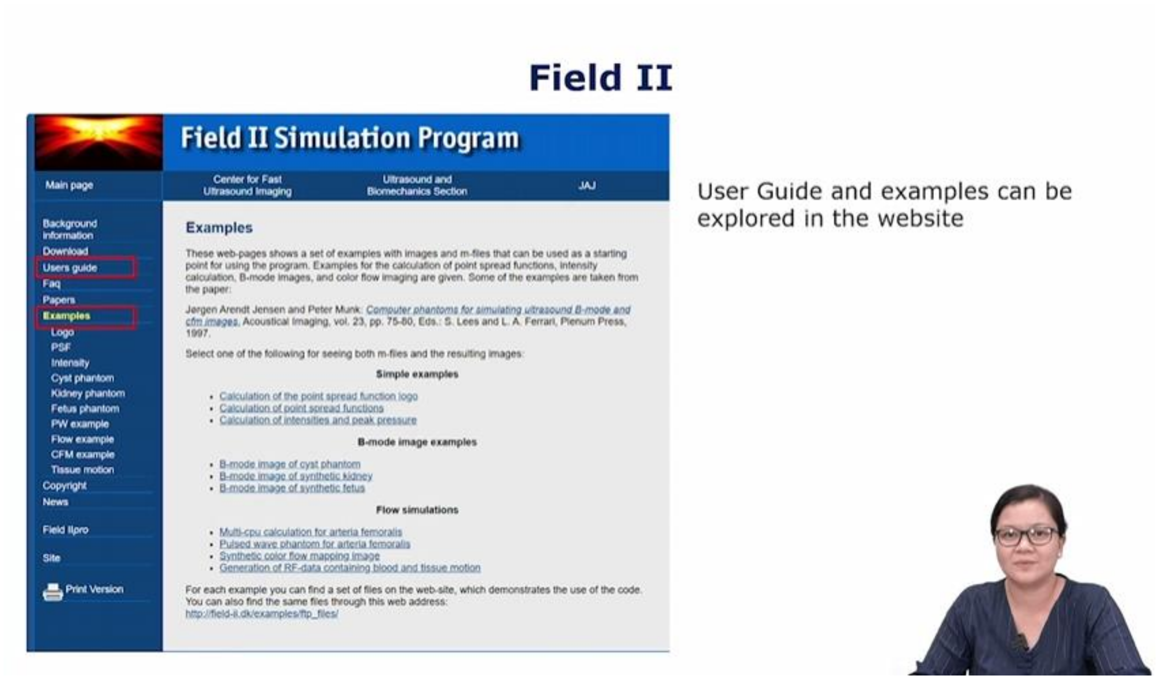
From Field II website: https://field-ii.dk/?./downloading_2021.html

What it includes is a tar file that will be downloaded to your MATLAB directory, which should hold the files. Then the file can be extracted by using this code here, gzip and tar. and that will be used to uncompress and extract the files. The tar file can then be deleted afterwards. And in your MATLAB directory, there will be a folder that contains all the

Field II program .m files. The user guide can also be found in that folder. And when you open MATLAB, you can type the code path using this line here, and field init to initialize the Field II program. So more of this information can be found in the website.

```
path(path,'/home/user/field_II/m_files');
field_init
```

The nice thing about the Field II software is that there's a lot of resources available in the website wherein you have a very comprehensive user guide that looks into the different commands and the different functions that are associated with simulating your ultrasound imaging system. There are also several examples as you can see here. There's an example of point spread functions, an intensity profile, different types of phantoms including cysts, kidney, fetus, other types regarding tissue motion can be found here as well.



Field II

User Guide and examples can be explored in the website

Examples

These web-pages shows a set of examples with images and m-files that can be used as a starting point for using the program. Examples for the calculation of point spread functions, intensity calculation, B-mode images, and color flow imaging are given. Some of the examples are taken from the paper:

Jørgen Arendt Jensen and Peter Munk: *Computer phantoms for simulating ultrasound B-mode and cfm images*, Acoustical Imaging, vol. 23, pp. 75-80, Eds.: S. Lees and L. A. Ferrari, Plenum Press, 1997.

Select one of the following for seeing both m-files and the resulting images:

Simple examples

- Calculation of the point spread function logo
- Calculation of point spread functions
- Calculation of intensities and peak pressure

B-mode image examples

- B-mode image of cyst phantom
- B-mode image of synthetic kidney
- B-mode image of synthetic fetus

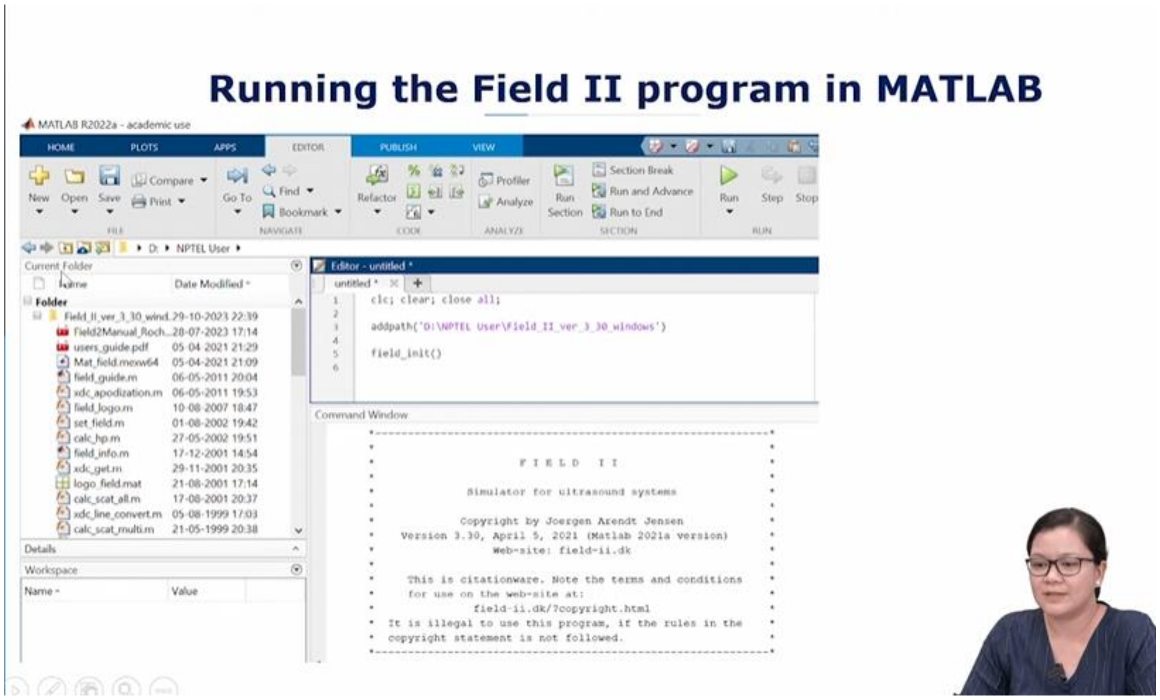
Flow simulations

- Multi-cpu calculation for arteria femoralis
- Pulsed wave phantom for arteria femoralis
- Synthetic color flow mapping image
- Generation of RF-data containing blood and tissue motion

For each example you can find a set of files on the web-site, which demonstrates the use of the code. You can also find the same files through this web address: http://field-ii.dk/examples/ftp_files/

So I encourage you after you to also explore these different types of examples in field II. Here, I show the MATLAB interface where you have the directory set at the folder which contains all your m files as well as a user guide. You can either type the commands in your command window, or you can open up a script where you can type down the code that is necessary to initiate field II and run the different examples. So here I've created a simple script here. As of now, it's untitled in this screenshot.

Running the Field II program in MATLAB



Name	Date Modified
Folder	
Field_II_ver_3_30_wind...	29-10-2023 22:39
Field2Manual_Roch...	28-07-2023 17:14
users_guide.pdf	05-04-2021 21:29
Mat_field.mexw64	05-04-2021 21:09
field_guide.m	06-05-2011 20:04
xdc_apodization.m	06-05-2011 19:53
field_logo.m	10-08-2007 18:47
set_field.m	01-08-2002 19:42
calc_hp.m	27-05-2002 19:51
field_info.m	17-12-2001 14:54
xdc_get.m	29-11-2001 20:35
logo_field.mat	21-08-2001 17:14
calc_scat_all.m	17-08-2001 20:37
xdc_line_convert.m	05-08-1999 17:03
calc_scat_multi.m	21-05-1999 20:38

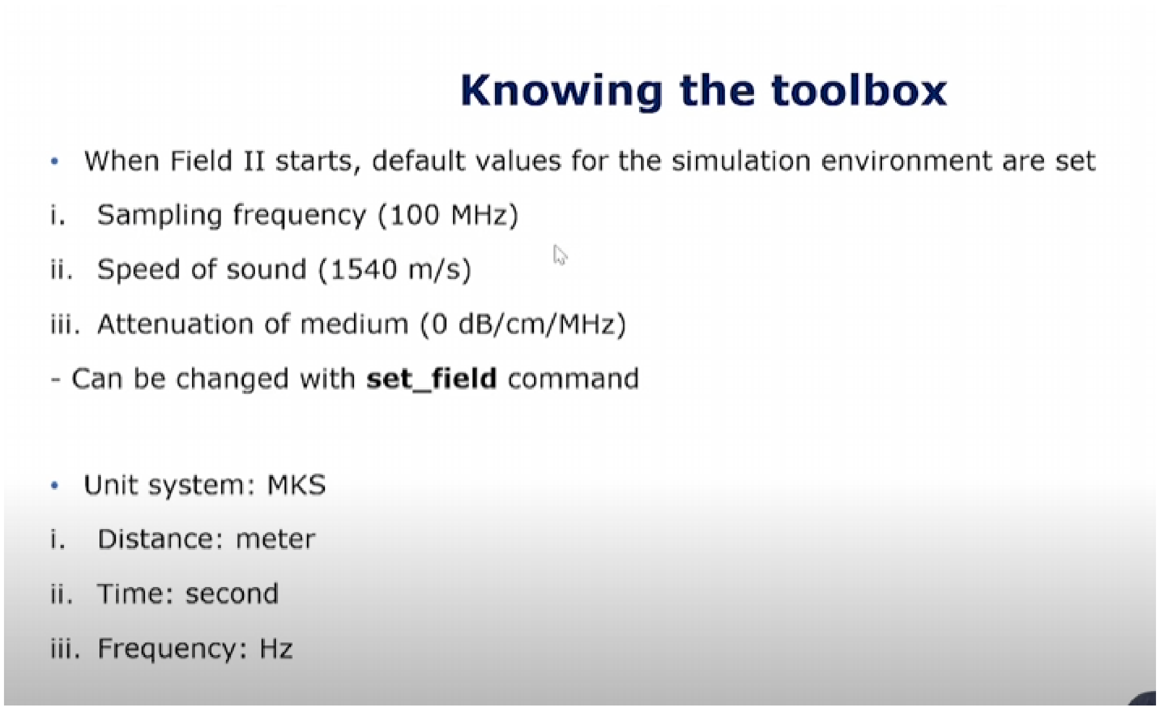
Name	Value

```matlab
clc; clear; close all;

addpath('D:\NPTEL User\Field_II_ver_3_30_windows')

field_init()

```

```
*-------------------------------------------------------------*
*                                                             *
*                        F I E L D   I I                      *
*                                                             *
*              Simulator for ultrasound systems               *
*                                                             *
*             Copyright by Joergen Arendt Jensen              *
*    Version 3.30, April 5, 2021 (Matlab 2021a version)       *
*                   Web-site: field-ii.dk                     *
*                                                             *
*    This is citationware. Note the terms and conditions      *
*    for use on the web-site at:                              *
*              field-ii.dk/?copyright.html                    *
*  It is illegal to use this program, if the rules in the     *
*  copyright statement is not followed.                       *
*-------------------------------------------------------------*
```

But the way to initialize field II is, if you can set up the path towards that Field II directory and type field init. Once this program is run, then you have an introductory text here in the command window. Let's just know the toolbox a bit more.

When Field II starts, there are several default values in the simulation environment including the sampling frequency, which is set at 100 MHz. And of course, the sampling frequency is set such that it follows the Nyquist criterion. The speed of sound is set to the average value of soft tissue, which is 1540 m/s. And the attenuation of the medium is set to zero, meaning that the default value is a lossless medium. There is no attenuation as the ultrasound propagates through the depth. Depending on the type of problem you are working on, you can change this parameter using this set field command. Now the set field command is also there, information about that is in the user guide, so you can set it depending on what your parameters are and how you would like your parameters to be.



Knowing the toolbox

- When Field II starts, default values for the simulation environment are set

i. Sampling frequency (100 MHz)

ii. Speed of sound (1540 m/s)

iii. Attenuation of medium (0 dB/cm/MHz)

- Can be changed with **set_field** command

- Unit system: MKS

i. Distance: meter

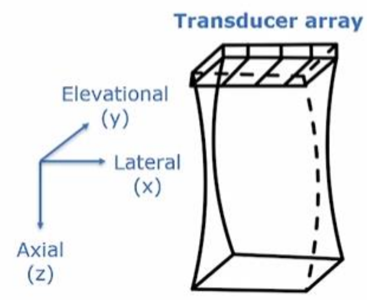
ii. Time: second

iii. Frequency: Hz

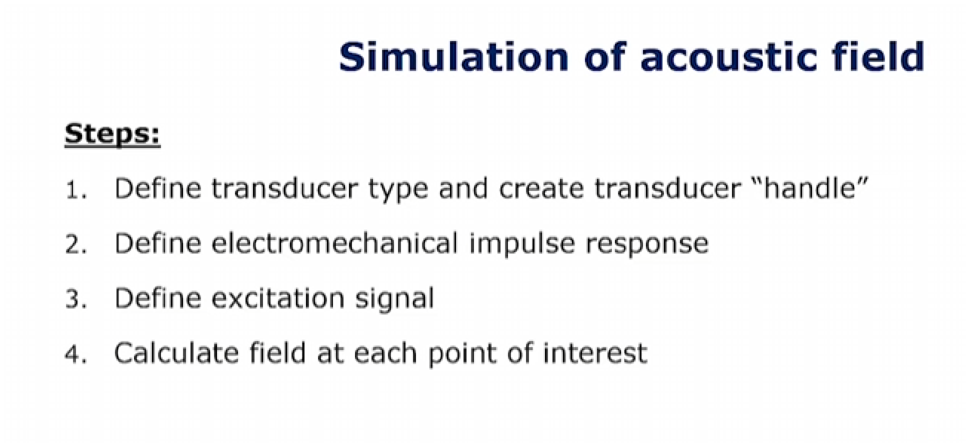
It's also important to know that the Field II toolbox also follows the MKS unit system, which is meters, kilograms, seconds. So the distance would be according to meters, time would be in seconds, and frequency would be according to hertz. We also take note of the coordinate arrangement within the simulation environment. There are three axes. The z-axis is corresponding to the axial depth direction. So if you look at the schematic here, we have our transducer array with multiple elements denoted by the small rectangular boxes. And you can see the beam profile right here, the axial depth is going down away from the transducer using this configuration. So it's from top to bottom. The X axis corresponds to the lateral direction of imaging. So that goes from left to right across an array. And the Y axis corresponds to the elevation direction, which is out of the XZ scan plane of imaging.

Coordinate arrangement

- z-axis: axial depth direction (away from transducer)
- x-axis: lateral direction (left to right across an array)
- y-axis: elevational direction (out of x-z scan plane)



Now the different steps for simulating an acoustic field is involved as follows.

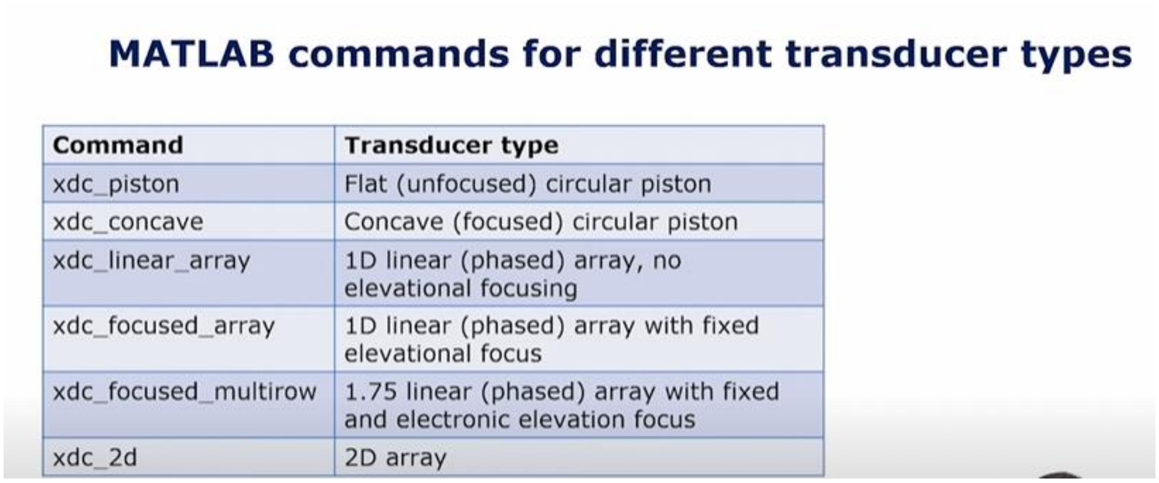


Simulation of acoustic field

Steps:

1. Define transducer type and create transducer "handle"
2. Define electromechanical impulse response
3. Define excitation signal
4. Calculate field at each point of interest

So first we would define our transducer type and create a transducer handle. I'll explain more about that in the next slide. Following, we would define the electromechanical impulse response of our imaging system. Then we would define the excitation signal that will be sent to the transducer elements, and finally calculating the field at each point of interest. Now here are some transducer types that can be simulated in the MATLAB command.

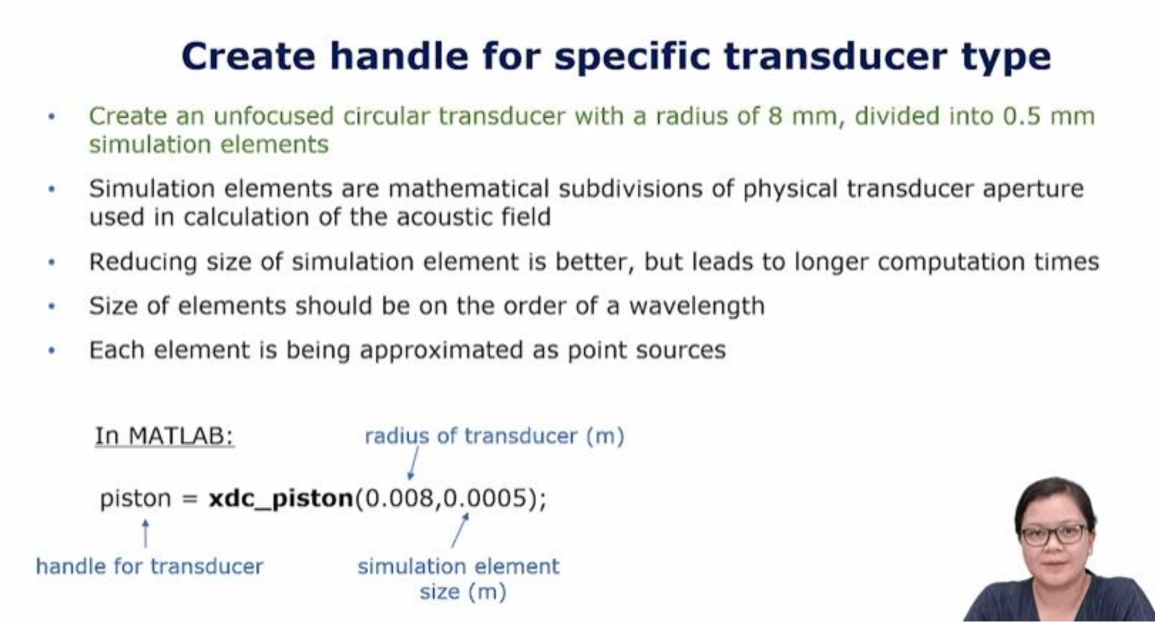


MATLAB commands for different transducer types

Command	Transducer type
xdc_piston	Flat (unfocused) circular piston
xdc_concave	Concave (focused) circular piston
xdc_linear_array	1D linear (phased) array, no elevational focusing
xdc_focused_array	1D linear (phased) array with fixed elevational focus
xdc_focused_multirow	1.75 linear (phased) array with fixed and electronic elevation focus
xdc_2d	2D array

So here on the left first column is the commands associated with the different transducer types. We have here the xdc_piston. This corresponds to the command for simulating a flat, unfocused, circular, single element transducer, or also called the piston transducer. We also have xdc_concave here, which simulates a focused, concave, circular piston, single element transducer as well. And we also have xdc_linear_array here that will simulate a 1D linear phased array. In this case, the code does not allow it to focus on the elevational direction. Whereas xdc_focused_array, allows you to simulate with a fixed elevational focus. So it actually depends on what your transducer parameters are, which you are simulating. We can also simulate a 1.75 phased array here with a fixed electronic elevation focus as well as a 2D array.

So you can see that this Field II simulation software has versatility in terms of simulating different types of transducer. Now, how do we specify a transducer type in Matlab? Here we can create a handle for that specific type.



Create handle for specific transducer type

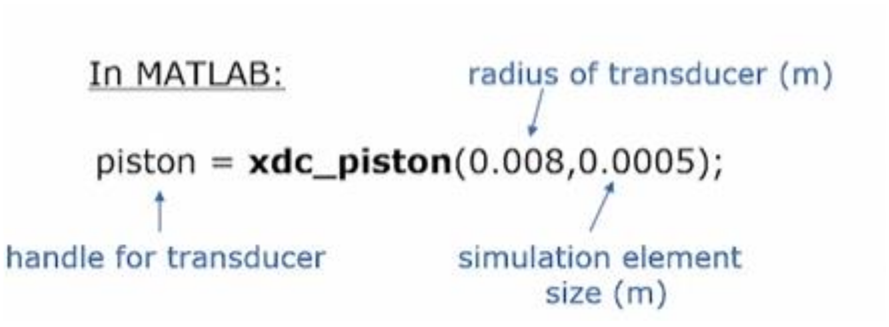
- Create an unfocused circular transducer with a radius of 8 mm, divided into 0.5 mm simulation elements
- Simulation elements are mathematical subdivisions of physical transducer aperture used in calculation of the acoustic field
- Reducing size of simulation element is better, but leads to longer computation times
- Size of elements should be on the order of a wavelength
- Each element is being approximated as point sources

In MATLAB:

```
piston = xdc_piston(0.008,0.0005);
```

So let's say I give this example where we'd like to create an unfocused circular or single element transducer with a radius of 8 mm. And we divide the element of this transducer into 0.5 mm simulation elements. So what are these simulation elements? So these are mathematical subdivisions of the actual physical transducer aperture, and that's used to calculate the acoustic field. So this is similar to when we are doing finite element modeling. For instance. where you have an object and you divide that into multiple smaller objects, to be able to simulate the properties or the response of that object. So similarly, we subdivide one aperture into multiple smaller elements. We call that “simulation elements”. So if we reduce the size of this simulation elements, much better the results would be produced. You will get finer results, but it will lead to longer computation times.

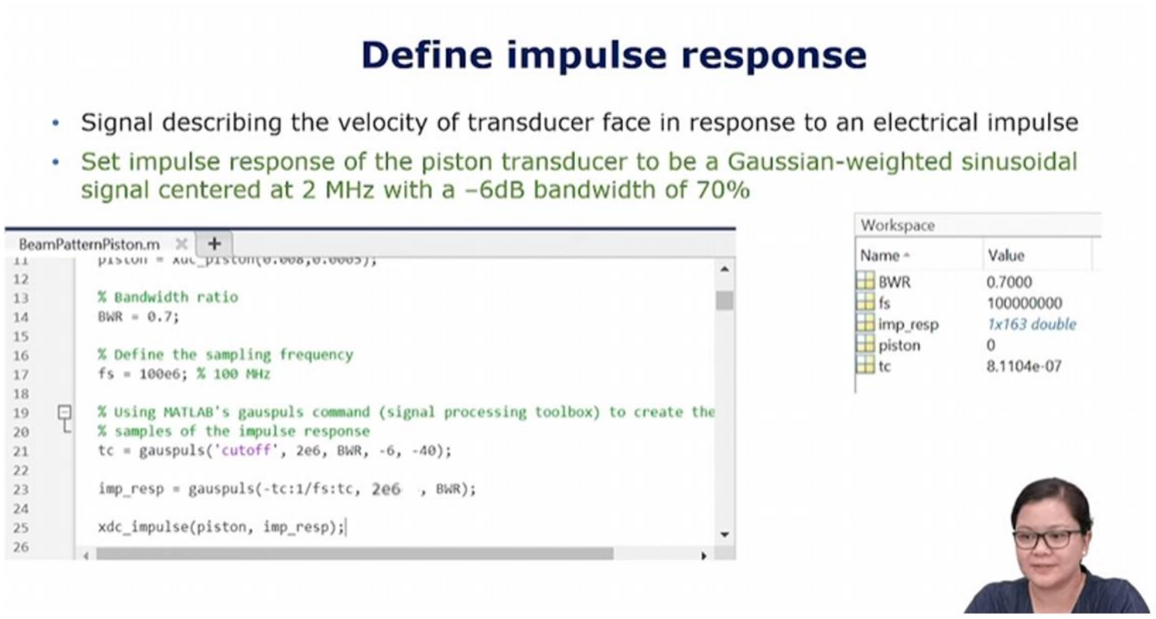
So there's a rule of thumb that the size of these simulation elements should be on the order of the wavelength. And in this case, each element is being approximated as point sources. So for example, similar to what you would observe when you apply Huygens principle, the way you would specify the transducer type is you would create a handle. In this case, in your MATLAB script or your command window, you can type this code where you name a handle for your transducer.



```
piston = xdc_piston(0.008,0.0005);
```

In this example, I call it piston. So piston is the handle. And that will equal to your command, xdc_ piston. Now, this will change if you have a linear array. So depending on the type of transducer you would like to simulate, then you would change this command. And the inputs to this xdc_piston command would be the radius of the transducer. So I had asked for an 8 mm radius. And in this case, since the field two environment operates in MKS units, here we write down 0.008 in terms of meters. And we would like that this aperture be divided into 0.5 mm simulation elements. So the next input you would have here is the size of that simulation element in meters.

After we define our transducer, the next step is to define the impulse response. So we had discussed earlier in a previous lecture what the impulse response is.



Define impulse response

- Signal describing the velocity of transducer face in response to an electrical impulse
- Set impulse response of the piston transducer to be a Gaussian-weighted sinusoidal signal centered at 2 MHz with a –6dB bandwidth of 70%

```
piston = xdc_piston(0.008,0.0005);

% Bandwidth ratio
BWR = 0.7;

% Define the sampling frequency
fs = 100e6; % 100 MHz

% Using MATLAB's gauspuls command (signal processing toolbox) to create the
% samples of the impulse response
tc = gauspuls('cutoff', 2e6, BWR, -6, -40);

imp_resp = gauspuls(-tc:1/fs:tc, 2e6   , BWR);

xdc_impulse(piston, imp_resp);
```

Workspace

Name	Value
BWR	0.7000
fs	100000000
imp_resp	*1x163 double*
piston	0
tc	8.1104e-07

And this is the type of signal that describes the velocity of the transducer space in response to an electronic impulse. So if we would like to set the impulse response of the piston transducer to be a Gaussian weighted sinusoidal signal, and we'd like that to be centered at 2 MHz frequency with a negative 6 dB bandwidth of 70%, our code would be something that looks like this. I had created a script called BeamPatternPiston here and saved that as an m file.

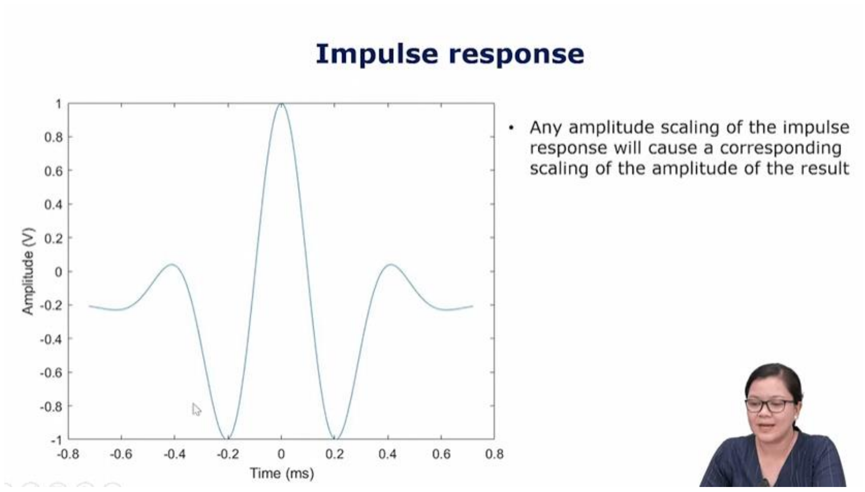
So I had already assigned my transducer handle. And next I will define the bandwidth ratio in the variable, BWR. And then I'd also like to define the sampling frequency.

Then this Gauss pulse command is in the signal processing toolbox of MATLAB, and this will help you simulate the samples of the impulse response (Line number 21). So the inputs of this Gauss pulse would be, you would type “cutoff”, the center frequency, bandwidth ratio, -6 dB corresponds to the full width half maximum of that bandwidth, which is what we would like, and the -40 stands for the dynamic range of your impulse response in dB.

So we would get this and create the frequencies components (Line number 23). The range of the values in the time domain would be added as input using the sampling frequencies and the time components. Then in this case, we like to simulate, a 2 MHz and then the bandwidth ratio. You would run this impulse response.

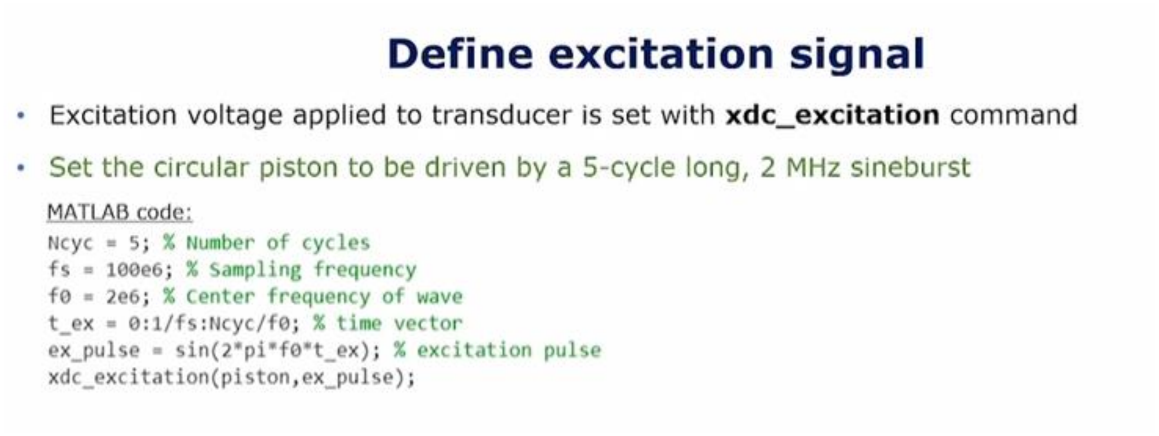
The impulse response would be inputted in the command in line 25 where you'd also look into the transducer handle. The command for generating the impulse response is xdc_impulse. And in your workspace, you will get these important variables that will represent your impulse response on your transducer.

Now this is what an impulse response would look like. You have some Gaussian weighted sinusoid which you can observe here. And in field two, if you multiply the amplitude by a certain factor, then the impulse response would scale according to that factor. So depending on your application, one would simulate it that way.


Impulse response

- Any amplitude scaling of the impulse response will cause a corresponding scaling of the amplitude of the result

After you have created your impulse response, the next is to define your excitation signal. And this is actually the voltage signal that is being applied to the transducer. And you would do this by setting using the command, xdc_excitation.



Define excitation signal

- Excitation voltage applied to transducer is set with **xdc_excitation** command
- Set the circular piston to be driven by a 5-cycle long, 2 MHz sineburst

MATLAB code:

```
Ncyc = 5; % Number of cycles
fs = 100e6; % Sampling frequency
f0 = 2e6; % Center frequency of wave
t_ex = 0:1/fs:Ncyc/f0; % time vector
ex_pulse = sin(2*pi*f0*t_ex); % excitation pulse
xdc_excitation(piston,ex_pulse);
```

So we would like to set the circular piston to be driven by a five cycle long, 2MHz sine burst. Here, Ncyc=5 is the number of cycles. We can find the sampling frequency, and the center frequency of the wave in this case is 2MHz. And we look at the time vector which would be used for the timing of our excitation pulse. And then we would generate the excitation pulse. Now this is a sine burst we had defined. So it's just the sine function of two pi center frequency times that time vector. So you had created the excitation pulse, then you would put it in this xdc_excitation command, wherein you point the handle to your piston transducer that you defined earlier, and then you put the excitation pulse signal here.

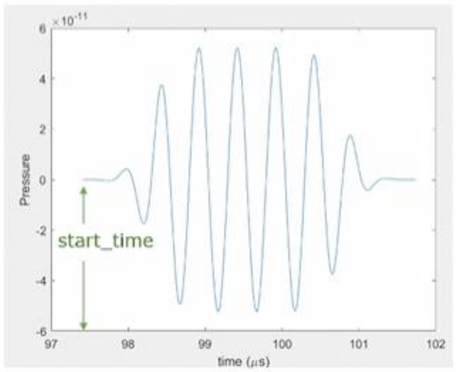
If you would like to display what the signal would look like at a point located 15 centimeters on the axis, then you can code something like this.

Calculate field at each point

- Display the signal at a point located at 15 cm on axis

MATLAB code:

```
[hp, start_time]=calc_hp(piston, [0 0 0.15]);
times=(1:length(hp))/fs+start_time;
figure(2)
plot(times*1e6,hp),xlabel('time (\mus)'); ylabel('Pressure');
```



calc_hp command returns:

i. Vector of pressure samples, "hp"

ii. Time corresponding to the sample in the vector

Output: start_time = 97.4 μs

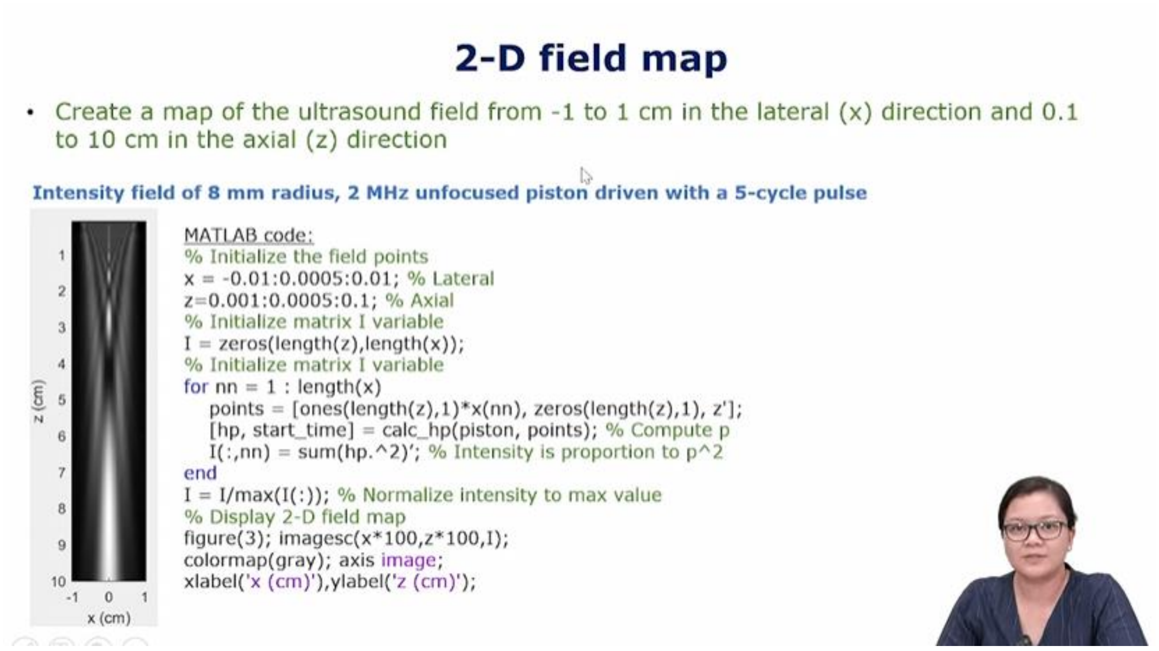
So our calc_hp function, actually computes the pressure amplitudes at different points in space. So depending on the location we'd like to compute, we would input the location in the vector parameter. So the calc_hp command returns a vector of pressure amplitudes, which here we call hp, and a time vector corresponding to the sample in the vector. You can also compute the start time, and we'll go through that a little bit further.

In terms of the inputs of calc_hp,   we would use this handle to point to our transducer. And then the location is, in terms of x, y, and z. So the first input of this vector would be x, second y, and third is z. Since we're looking 15 centimeters on axis, that's in the axial direction, which is our z direction. And this is in meters. So we use the length of the pressure samples that we would get, divide that by the sampling frequency of 100 MHz and to that, the start time is added.

What the start_time does is that when you generate the actual pressure signal as a function of time then the first sample would start at the start time. The start time actually corresponds to the 15 centimeter distance, where your point is located.

So that start time is needed to shift the pulse at the right timing. So here you just plot the signal according to this basic plotting function here. And so this is what we get. We get five cycles. Of course, it's not a complete sine burst that is evenly distributed, because it also depends on our impulse response. But this is what the pressure field would look like at that point in space.

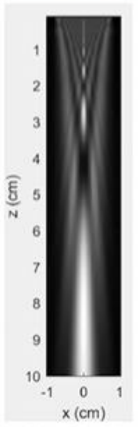
Now if you want to create a two-dimensional map of that pressure field, say from minus 1 to 1 centimeter in the lateral X direction and 0.1 to 10 centimeters in the axial Z direction. So this is the following MATLAB code here.



2-D field map

- Create a map of the ultrasound field from -1 to 1 cm in the lateral (x) direction and 0.1 to 10 cm in the axial (z) direction

Intensity field of 8 mm radius, 2 MHz unfocused piston driven with a 5-cycle pulse



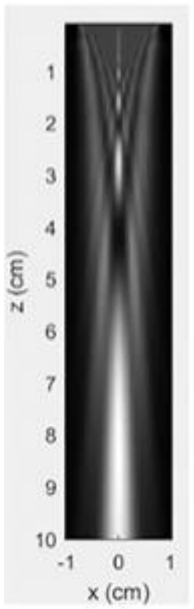
MATLAB code:

```
% Initialize the field points
x = -0.01:0.0005:0.01; % Lateral
z=0.001:0.0005:0.1; % Axial
% Initialize matrix I variable
I = zeros(length(z),length(x));
% Initialize matrix I variable
for nn = 1 : length(x)
    points = [ones(length(z),1)*x(nn), zeros(length(z),1), z'];
    [hp, start_time] = calc_hp(piston, points); % Compute p
    I(:,nn) = sum(hp.^2)'; % Intensity is proportion to p^2
end
I = I/max(I(:)); % Normalize intensity to max value
% Display 2-D field map
figure(3); imagesc(x*100,z*100,I);
colormap(gray); axis image;
xlabel('x (cm)'),ylabel('z (cm)');
```

The first thing we would need is to initialize the field points. So we would like calculations at several points in space. And in this case, for the lateral location, we generate a vector from -1 cm or -0.01 meter, all the way to 0.01 meter. The subdivision or sub-locations will be 0.5 millimeters. And then in the axial direction, similarly from 0.1 centimeters all the way up to 10 centimeters at 0.5 spatial difference. The next thing after we have our vector of points is that, we will then initialize the matrix of intensity values

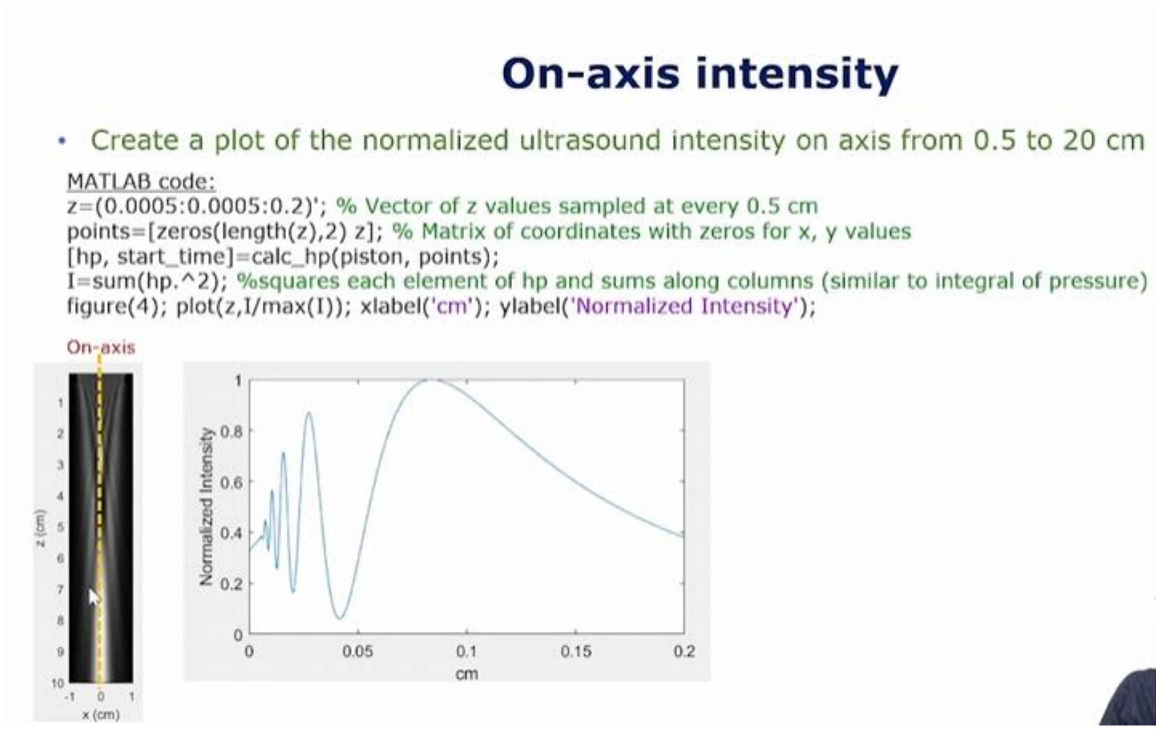
So later on, we'd like to create an intensity map. And the way we would do that is at each point in space. We have this for loop right here. That's going to go through each lateral location in X and it's going to calculate what the pressure field would be at that point in space. And since intensity is proportional to the pressure squared, then what we do is at that specific point in space, we would first square the pressure signal and sum over that vector pressure signal. So with that, it's similar to doing the integral of the pressure squared as we are calculating the intensity.

Then, we would also like to plot a normalized intensity. So we would just divide the entire intensity matrix by the maximum intensity values and we displayed a two-dimensional map of it. So the result is also in the figure below, where we have the z direction or the axial direction in the vertical axis, and we have the lateral direction in the horizontal axis. And so when we're looking at this, it's showing that the transducer is emitting the field from the top of the image.



You can see the Fresnel region, which is the near field region of your transducer. And then below that would be the far field region. So it's nice to be able to see, depending on the type of transducer you had. This simulation program would give you what the field you would expect to be.

You can also plot the on-axis intensity right. So you can just take the column at a point in space. When we talk about axis, It means that we start at the zero lateral location, which is the center of your beam, and then you just plot it across the axial depth. So if you want to create a on-axis intensity profile from 0.5 to 20 centimeters, then you would code it like this.



On-axis intensity

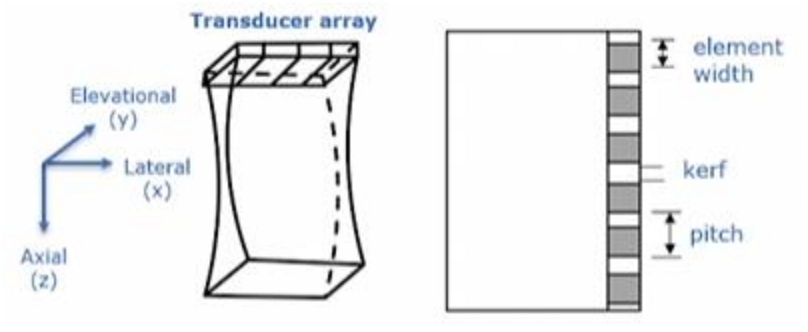
- Create a plot of the normalized ultrasound intensity on axis from 0.5 to 20 cm

MATLAB code:

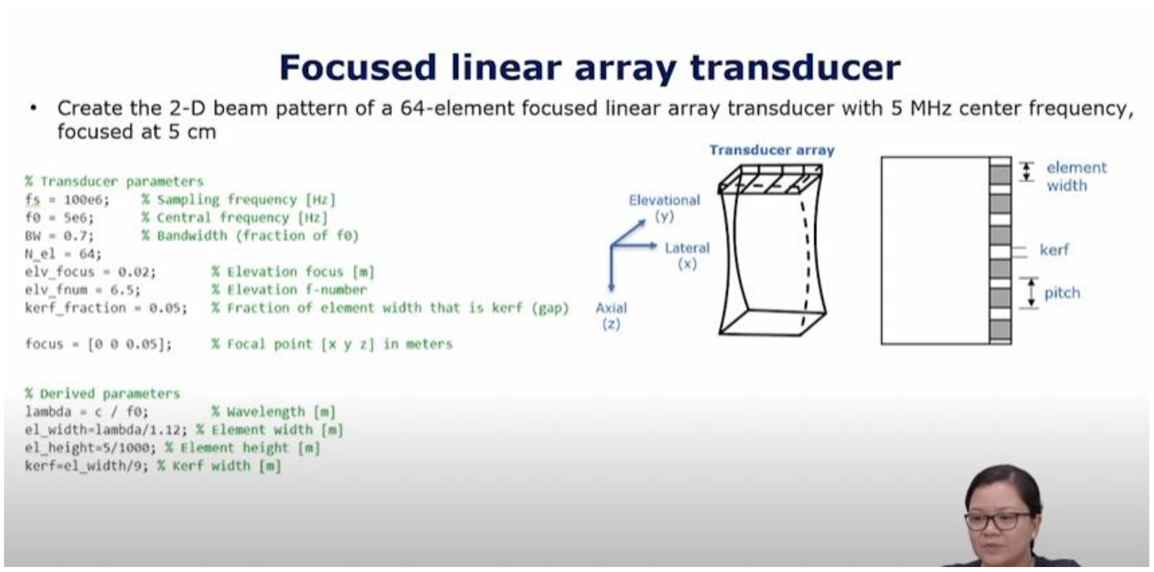
```
z=(0.0005:0.0005:0.2)'; % Vector of z values sampled at every 0.5 cm
points=[zeros(length(z),2) z]; % Matrix of coordinates with zeros for x, y values
[hp, start_time]=calc_hp(piston, points);
I=sum(hp.^2); %squares each element of hp and sums along columns (similar to integral of pressure)
figure(4); plot(z,I/max(I)); xlabel('cm'); ylabel('Normalized Intensity');
```

So first you would specify the points in space. So in this case, we're talking about the z-axis. So we're going from 0.5 centimeters in 0.5 centimeter intervals all the way up to 20 centimeters right here. Again, these are in meter units. And the points can be such that you can have your matrix of coordinates with zero values for x and y since we're only looking at the z direction. We use again the calc_hp function to compute the pressure field at that location. Use start time again. And we would sum the pressure squared, the hp squared for doing our integral. And after that, we get an I value, which corresponds to our intensity. And we would plot the normalized intensity to get the output. You will see that the x-axis normalized intensity as a function of axial depth right and you can see that when you are in the near field, the on-axis intensity goes up and down really fast in the spatial dimension in the near field right. But as you go towards the far field, you notice that the intensity will climb up all the way to the last axial maxima. and the intensity would attenuate because the beam is also spreading as you go further into the far field. So this is a nice way to visualize what your field would look like on axis.

So that is an example of a single element circular piston transducer. Now I'll give an example of what a focused linear array transducer would look like in terms of its beam pattern. So what we want to do is we want to create a two-dimensional beam pattern of a 64 element focused linear array transducer with 5 MHz center frequency and it's focused at 5 cm. So here's just reminding you again of what the coordinates would look like. So we have our transducer array. We'll have 64 elements in this case. There is only four here in the schematic, but we'll have 64 elements in our example. So the beam will look something like this. And you'll have an elevational direction. You have lateral and an axial direction. Also, just to remind you that if you remember in the transducer array lecture where we talked about element width, the kerf, which is the space between the elements, and then the pitch, which is the kerf plus the element width. So these parameters will become important when you are simulating your linear array transducer.



So let's go back to the parameters that you will define in your MATLAB code. So first you have your sampling frequency which is, 100 MHz. The center frequency we would like to be 5 MHz, so 5e6. Bandwidth ratio, we'll keep it at 70%. The number of elements are defined here as 64.



Focused linear array transducer

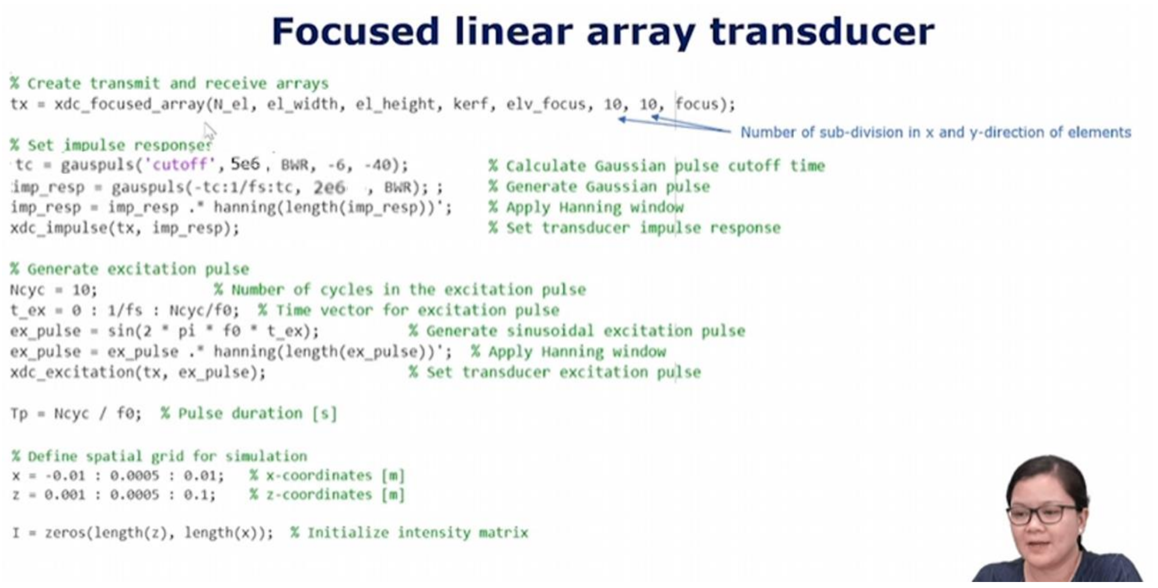
- Create the 2-D beam pattern of a 64-element focused linear array transducer with 5 MHz center frequency, focused at 5 cm

```
% Transducer parameters
fs = 100e6;     % Sampling frequency [Hz]
f0 = 5e6;       % Central frequency [Hz]
BW = 0.7;       % Bandwidth (fraction of f0)
N_el = 64;
elv_focus = 0.02;        % Elevation focus [m]
elv_fnum = 6.5;          % Elevation f-number
kerf_fraction = 0.05;    % Fraction of element width that is kerf (gap)

focus = [0 0 0.05];      % Focal point [x y z] in meters

% Derived parameters
lambda = c / f0;         % Wavelength [m]
el_width=lambda/1.12; % Element width [m]
el_height=5/1000; % Element height [m]
kerf=el_width/9; % Kerf width [m]
```

And then we also would like to define the elevation focus of two centimeters, so 0.02 meters. Depending on the transducer that you're trying to simulate, the manufacturer of that transducer would give you what the elevation focus is. Also, elevation f number here is 6.5. The kerf fraction, 0.05. So fraction of element width that is kerf. So the fraction of this element width or the pitch that is actually the kerf is the kerf fraction.

And we would like to see the beam at the focus. So here our focal point is 5 cm, so 0.05. And since we're only looking at a focal point on axis, your X and Y locations are set to zero. There are also some other parameters here that will be important when you are going to calculate others. For instance, the wavelength here would be computed by your speed of sound and your center frequency. Also, to define the element width, you would need that wavelength parameter here, as well as you can define an element height and the kerf as well.

Now let's create the transducer handle. To create a focused linear array transducer, we would use this command, xdc_focused_array. Depending on your transducer, you would go back to that table that I had described earlier, which looks at the different commands for the transducer type. So the inputs for this function will be the number of elements, the element width, element height, as well as the kerf, elevation, focus, and there are two others here that relate to the number of subdivisions, so simulation elements within that transducer element. So here I type 10 in the X direction and 10 in the Y direction, meaning that for one element I'm dividing it into 10 smaller mathematical elements in the X as well as in the Y direction. And another input is the focus.



Focused linear array transducer

```
% Create transmit and receive arrays
tx = xdc_focused_array(N_el, el_width, el_height, kerf, elv_focus, 10, 10, focus);

% Set impulse response
tc = gauspuls('cutoff', 5e6 , BWR, -6, -40);         % Calculate Gaussian pulse cutoff time
imp_resp = gauspuls(-tc:1/fs:tc, 2e6   , BWR); ;     % Generate Gaussian pulse
imp_resp = imp_resp .* hanning(length(imp_resp))';   % Apply Hanning window
xdc_impulse(tx, imp_resp);                           % Set transducer impulse response

% Generate excitation pulse
Ncyc = 10;              % Number of cycles in the excitation pulse
t_ex = 0 : 1/fs : Ncyc/f0;  % Time vector for excitation pulse
ex_pulse = sin(2 * pi * f0 * t_ex);           % Generate sinusoidal excitation pulse
ex_pulse = ex_pulse .* hanning(length(ex_pulse))';  % Apply Hanning window
xdc_excitation(tx, ex_pulse);                 % Set transducer excitation pulse

Tp = Ncyc / f0;  % Pulse duration [s]

% Define spatial grid for simulation
x = -0.01 : 0.0005 : 0.01;   % x-coordinates [m]
z = 0.001 : 0.0005 : 0.1;    % z-coordinates [m]

I = zeros(length(z), length(x));  % Initialize intensity matrix
```

So after we create the handle to our transducer, in this case my handle is now called tx, then we will set the impulse response, similar to what it was previously. So in this case, Gauss pulse cut off is 5 MHz.

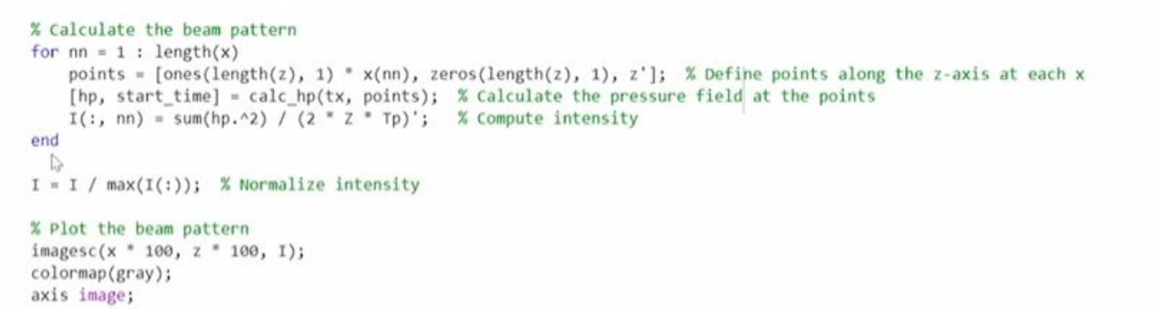
Then you should add -6 dB, Similar to how Gauss pulse was in the single element transducer part. So we would create the impulse response here, from -tc to 1/fs. to tc , 5e6 and the bandwidth ratio. Afterwards, let's say you wanted to weight the type of pulse.

So here in this case, you would like to apply a Hanning window. So we would just multiply that impulse response with the Hanning window of a similar length. Then we would calculate the impulse response again using the xdc_impulse. So after you set your impulse response, then you can generate an excitation pulse.

Here, the number of cycles is set to 10. So I'm exciting the transducer with a 10 cycle sine burst or sinusoidal excitation pulse. I also like to weight that sine pulse using a Hanning function and then calculate the excitation signal by inputting the transducer handle and the excitation pulse that we sent to the transducer. So setting the impulse response, creating the handle, generating the excitation pulse, these are the common steps you would do first before viewing the beam of the transducer. So then we'd also like to define the spatial grid for the simulation.

So in this case, the spatial grid would go from -1 to 1 cm with at 0.5 cm spatial intervals. And the Z direction will be simulating from 0.1 cm or 1 mm all the way up to 10 cm. The variable, Tp, the post duration will be relevant for the next stage of the simulation. We then also initialize the intensity matrix because here we'd like to plot the beam intensity of the focus linear array transducer.

Initializing variables helps to make the computation faster. So then now here, we're calculating the beam pattern right here, where similar to the 2D beam pattern of the single element transducer, we would have, it loop through the different lateral locations, and look at the points as a function of the axial locations as well.

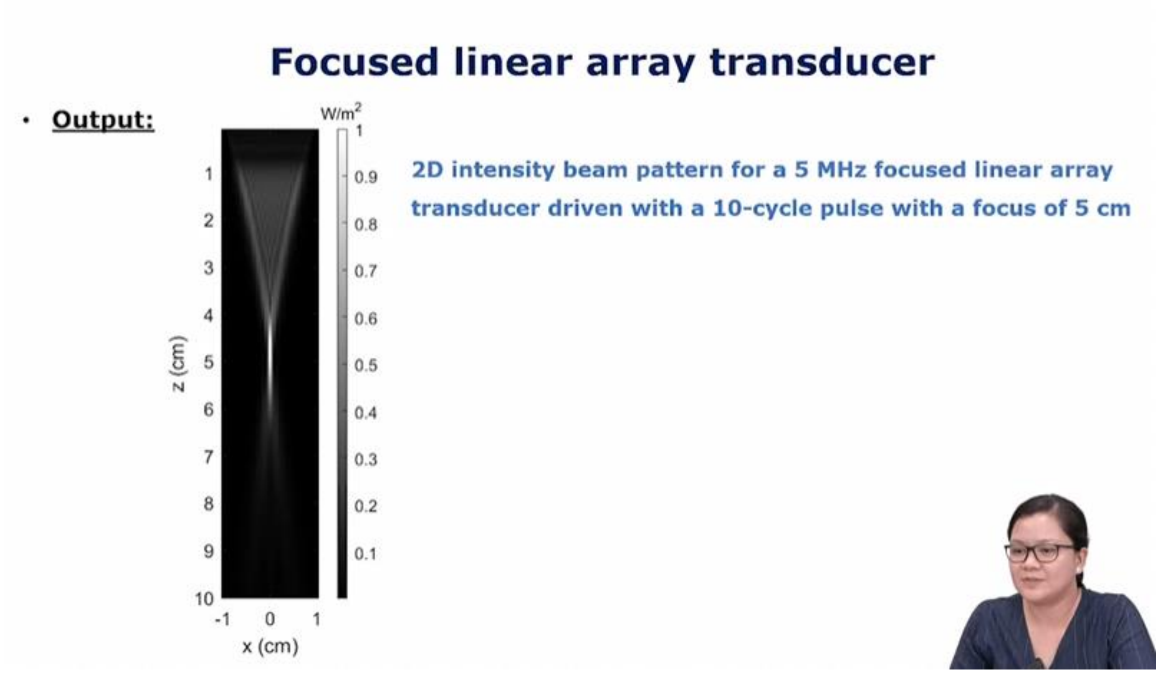


```
% Calculate the beam pattern
for nn = 1 : length(x)
    points = [ones(length(z), 1) * x(nn), zeros(length(z), 1), z'];  % Define points along the z-axis at each x
    [hp, start_time] = calc_hp(tx, points);  % Calculate the pressure field at the points
    I(:, nn) = sum(hp.^2) / (2 * Z * Tp)';   % Compute intensity
end

I = I / max(I(:));  % Normalize intensity

% Plot the beam pattern
imagesc(x * 100, z * 100, I);
colormap(gray);
axis image;
```

So using this code, we will be defining the points along the Z axis at each lateral x location. Then we would compute the pressure field at these points here using the calc_hp function again where we put the transducer handle and the points variable that looks to the locations in the field.

Afterwards, we would like to compute the intensity. If you would like close enough values to the intensity based on the pressure, you can divide by two times the acoustic impedance and the pulse duration right here. And then you would get the intensity that would be close enough based on a plane wave approximation. Then here you would normalize intensity. You would get a normalized intensity variable in I.

Then you would plot the beam pattern according to your spatial coordinates. So this is how a focused linear array transducer intensity beam would look like, where you can see that there's a nice focal spot at 5 cm, which would we have defined.



Focused linear array transducer

- **Output:**

2D intensity beam pattern for a 5 MHz focused linear array transducer driven with a 10-cycle pulse with a focus of 5 cm

And for any focused transducers that you would simulate, you would see a really high intensity beam, and the beam would be much thinner at the focal spot right. You can also see the focal region, the near field and the far field would be farther in the axial direction.

So I hope this lecture gives you a good overview of how to use Field II, how to set up the program in your MATLAB environment. We have defined the transducer. We also know how to define the impulse response as well as the excitation signal to the transducer. We learned how to generate 2D intensity field maps of a single element transducer and a focused linear array transducer. So based on these, the coding is very similar to what it would be for other types of transducers that is supported in the Field II environment. If you want to know more about how to code for different types of transducers, then I recommend going through the Field II website, as well as looking through the user guide to help guide you on how to code. We will see you in the next lecture. Thank you.