Users' guide for the Field II program

Release 3.30, April 5, 2021

Jørgen Arendt Jensen

April 5, 2021

Jørgen Arendt Jensen
April 5, 2021
Department of Health Technology, Build. 349,
Technical University of Denmark
DK-2800 Lyngby, Denmark
E-mail: jaje@dtu.dk
Web: http://field-ii.dk/

CONTENTS

LIST OF FIGURES

CHAPTER

ONE

Introduction

This is the user guide for the version 3.30 of April 5, 2021 of the Field II program. This version of the program runs under Matlab 2021a, 2020 and 2019[1], and it can simulate all kinds ultrasound transducers and the associated images. The focusing and apodization of the transducers can be controlled dynamically, and it is, thus, possible to simulate all kinds of ultrasound imaging systems. The versions can also be used for synthetic aperture imaging.

The program is free for use, if you make a proper reference to the papers describing the program, when you publish results from its use. The reference are [1] and [2]. Also the name of the program (Field II) should be mentioned in the publication. Some unfortunately forget this, and the program will only stay in the public domain, if people continue to properly acknowledge its use.

This guide is intended as a presentation of the currently available routines. It includes a few examples and gives a small amount of background information. It is, however, not intended as an introduction to ultrasound scanning, and the reader should consult the extensive literature on this.

The program executables can be downloaded from the Web-site for the program:

http://field-ii.dk/

It currently exist for a number of platforms like Windows, Linux, Mac OS-X. Versions for other operating systems are generally discontinued due to lack of demand. A parallel Pro version also exits. Information about this can be found on the web-site above.

The web site also contains more extensive examples than are given in this guide, and up-to-date references and papers can also be found on the web-site.

The manual is made as a clickable pdf document with hyperlinks. All links are indicated in blue, and when clicked on will lead to the indicated references, which can be a web-site, figure, equation, etc.

The manual is organized as follows: Chapter 2 gives an overview of the organization of the program and how it is connected to Matlab. Chapter 4 details the installation from the programs on the web-site. A listing of all procedures callable in the program is given in Chapter 5 and finally a few examples are given in 6. More can be found on the Field II web-site.

Jørgen Arendt Jensen
April 5, 2021
Department of Health Technology, Build. 349,
Technical University of Denmark
DK-2800 Lyngby, Denmark
E-mail: jaje@dtu.dk
Web: http://field-ii.dk/

[1] Older versions can be found on the web-site for Matlab 4-7

CHAPTER

TWO

Program organization

The program consists of a C program and a number of Matlab m-functions that calls this program. All calculations are performed by the C program, and all data is kept by the C program.

Three types of m-functions are found. The are used for initializing the program, defining and manipulating transducers, and for performing calculations. The initializing routines are preceeded by field_, the transducer commands by xdc_, and the calculation routines by calc_. Help on use of the routines can be obtained by typing help <routine name>. Each of the routines are described in the following section and then three examples of use are given. The first shows how a phased array image is generated, the second simulates a flow system, and the last example is for a linear array system. The last example uses a computer generated phantom. The m-file for this phantom is also given in the example section.

CHAPTER

THREE

Method of simulation

3.1 The spatial impulse response

The Field program system uses the concept of spatial impulse responses as developed by Tupholme and Stepanishen in a series of papers [9, 10, 11]. The approach relies on linear systems theory to find the ultrasound field for both the pulsed and continuous wave case. This is done through the spatial impulse response. This response gives the emitted ultrasound field at a specific point in space as function of time, when the transducer is excitated by a Dirac delta function. The field for any kind of excitation can then be found by just convolving the spatial impulse response with the excitation function. The impulse response will vary as a function of position relative to the transducer, hence the name spatial impulse response.

The received response from a small oscillating sphere can be found by acoustic reciprocity. The spatial impulse response equals the received response for a spherical wave emitted by a point. The total received response in pulse-echo can, thus, be found by convolving the transducer excitation function with the spatial impulse response of the emitting aperture, with the spatial impulse response of the receiving aperture, and then taking into account the electro-mechanical transfer function of the transducer to yield the received voltage trace. An explanation and rigorous proof of this can be found in [14] and [15].

Any excitation can be used, since linear systems theory is used. The result for the continuous wave case is found by Fourier transforming the spatial impulse response for the given frequency. The approach taken here can, thus, yield all the diffent commenly found ultrasound fields for linear propagation.

3.2 Simulation

A number of different authors have calculated the spatial impulse response for different transducer geometries. But in general it is difficult to calculate a solution, and especially if apodization of the transducer is taken into account. Here the transducer surface does not vibrate as a piston, e.g. the edges might vibarte less then the center. The simulation program circumvents this problem by dividing the transducer surface into squares and the sum the response of these squares to yield the response. Thereby any tranducer geometry and any apodization can be simulated. The approach is described in [1].

The time for one simulation is also of major concern. As the squares making up the tranducer apertue are small, it is appropriate to use a far-field approximation, making simulation simple. Another issue in keeping the simulation time down is to use a low sampling frequency. Often spatial impulse responses are calculated using sampling frequencies in the GHz range due to the sharp discontinuities of the responses. These discontinuities are handled in the Field programs by accurately keeping track of the time position of the responses and uses the integrated spatial impulse response as an intermediate step in the calculations. Thereby no energy is lost in the response, which is far more important than having an exact shape of the spatial impulse response. Hereby the Field program ususally does better using 100 MHz sampling and approximate calculations, than using the exact analytic expression and GHz sampling.

3.3 Focusing and apodization

The focusing and apodization is handled in the program through time lines as:

Focusing: From time	Focus at
0	x_1, y_1, z_1
t_1	x_1, y_1, z_1
t_2	x_2, y_2, z_2
$\vdots$	$\vdots$

Apodization: From time	Apodize with
0	$a_{1,1}, a_{1,2}, \cdots a_{1,N_e}$
t_1	$a_{1,1}, a_{1,2}, \cdots a_{1,N_e}$
t_2	$a_{2,1}, a_{2,2}, \cdots a_{2,N_e}$
t_3	$a_{3,1}, a_{3,2}, \cdots a_{3,N_e}$
$\vdots$	$\vdots$

For each focal zone there is an associated focal point and the time from which this focus is used. The arrival time from the field point to the physical transducer element is used for deciding which focus is used. The focusing can also be set to be dynamic, so that the focus is changed as a function of time and thereby depth. The focusing is then set as a direction defined by two angles and a starting point on the aperture.

All the time values for focusing are calculated relative to a point on the aperture. Initially this is set to (0, 0, 0). It can be set to other values through the procedure xdc_center_focus. This is used in linear array imaging, where the origin of the emitted and received beam is moved over the aperture. The focusing values are calculated by:

$$t_i = \frac{1}{c}\left(\sqrt{(x_c - x_f)^2 + (y_c - y_f)^2 + (z_c - z_f)^2} - \sqrt{(x_i - x_f)^2 + (y_i - y_f)^2 + (z_i - z_f)^2}\right) \tag{3.1}$$

where (x_f, y_f, z_f) is the position of the focal point, (x_c, y_c, z_c) is the reference center point on the aperture for the focus as set by xdc_center_focus, (x_i, y_i, z_i) is the center for the physical element number i, c is the speed of sound, and t_i is the calculated delay time. The value is then quantized, if that is set for the aperture.

The time line method is employed for the apodization, where the time decides which apodization vector is used. The vector holds one apodization value for each physical element.

3.4 Attenuation

Frequency dependent attenuation can be included in the simulation by using the procedure set_field. The attenuation is included through a frequency dependent term and a frequency independent term. The frequency dependent term is linearized through a center frequency att_f0, so that the attenuation is zero dB at att_f0. This is done to make the inclusion of the attenuation computationally efficient. The variation in distance over the aperture of the frequency dependent attenuation is usually not significant, and therefore only the frequency independent attenuation is varied over the aperture. The frequency dependent attenuation is then included on the response by using the mean distance to the aperture.

The attenuation is assumed to be minimum phase.

CHAPTER

FOUR

Installation

The excutable code for the program can be obtained free of charge from the web-site:

http://field-ii.dk/

Here the mex-file to run under Matlab and the m-files for calling the mex-files can be found. Versions are currently found for Linux (Intel processors, 64 bits), MAC OS and Windows 64 bits. The latest version is only found for 64 bits processors. Older versions are found for HP-UX (PA-RISC processors), SUN (OS4.1 and Solaris), DEC ALPHA, Silicon Graphics, IBM AIX, but they are no longer supported. Matlab 2019 or higher is required to run the program, but older versions from Matlab 5 and on are also found on the web-site.

The individual files can be found at the web-site along with compressed Unix-style tar-files. A zip file also exits for the windows version. 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.Z
tar -xvf <name_of_tar_file>.tar
```

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

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

```
addpath('/home/user/field_II/m_files');
field_init
```

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

CHAPTER

FIVE

Description of Matlab procedures

5.1 List of current procedures

General commands

Transducer commands

Function name	Purpose	Page
xdc_lines	Create an aperture bounded by a set of lines.	34
xdc_piston	Define a round, flat aperture.	36
xdc_quantization	Set quantization of the phase delays.	37
xdc_rectangles	Procedure for creating an aperture consisting of rectangles.	38
xdc_show	Show information about an aperture.	39
xdc_times_focus	Creating a focus time line for an aperture with all delay values supplied by the user.	40
xdc_triangles	Make a multi-element aperture consisting of triangles.	41
xdc_2d_array	Create a two-dimensional array transducer.	41

Element manipulation commands

Function name	Purpose	Page
ele_apodization	Set the apodization for individual mathematical elements.	45
ele_delay	Set the delay for individual mathematical elements.	46
ele_waveform	Set the waveform for individual physical elements.	47

Field calculation commands

Function name	Purpose	Page
calc_h	Calculate the spatial impulse response.	50
calc_hhp	Calculate the pulse echo field.	51
calc_hp	Calculate the emitted field.	52
calc_scat	Calculate the received signal from a collection of scatterers.	53
calc_scat_all	Calculate the received signals from a collection of scatterers for all transmit and receive elements in the aperture.	53
calc_scat_multi	Calculate the received signals from a collection of scatterers for all the elements in the aperture.	55

5.2 Procedures for Field initialization

Field II user's guide **field_debug**

Purpose: Procedure for initialize debugging. This will print out various information about the programs inner working. Initially no debugging is done.

Calling: field_debug(state)

Input: State - 1: debugging, 0: no debugging.

Output: none.

Field II user's guide **field_end**

Purpose: Procedure for terminating the Field II program system and releasing the storage.

Calling: field_end ;

Input: none.

Output: none.

Field II user's guide **field_guide**

Purpose: Procedure for displaying the Field II users' guide (this guide) using the Adobe acrobat reader.

Calling: field_guide

Input: none.

Output: The Field II guide is displayed in a separate window using acrobat reader.

Note that the Adobe pdf reader must be installed on the system, and it must be accessible under Matlab under the name acroread. The users guide should be in the search path of Matlab, preferrably in the same directory as the m-files for Field II with the name users_guide.pdf.

Field II user's guide **field_info**

Purpose: Procedure for showing information about the Field II program. The information is printed in the Matlab window.

Calling: field_info

Input: None.

Output: Information is printed in the Matlab window.

For boolean variables a value of 1 indicates true and 0 for false.

Example: Print the information:

```
field_info

Current Field II configuration:

    Version 2.88, December 14, 2001 (Matlab version)

    Number of apertures in operation:            3
    Apertures to be defined uses rectangles:     0
    Apertures to be defined uses triangles:      0
    Apertures to be defined uses bounding lines: 1

    Program uses accuracte time calculation for rectangles: 0
    Program uses fast integration for lines and triangles:  1

    Speed of sound:      1540.0000 m/s
    Sampling frequency:   100.0000 MHz
    Whether to use attenuation: 0
    Frequency independent attenuation is:      0.0000 dB/m
    Frequency dependent attenuation around     0.0000 MHz is     0.0000 dB/[m Hz]
    Constant tau_m used in attenuation calculation:         20.0000

    Number of bytes reserved:                       12024
    Maximum number of bytes that has been reserved: 22924
    Number for next signal to be used:              60656
    Internal state of the program:                  1
    Debug mode enabled:                             0
    Last calculation type done:                     0
    Whether calculation time should be shown:       1
    Seconds between showing times                   5 s

A boolean value of 1 indicates true, 0 indicates false
```


Purpose: Procedure for initializing the Field II program system. Must be the first routine that is called before using the system.

Calling: field_init (suppress) ;

Input: suppress An optional argument suppress with a value of zero can be given to suppress the display of the initial field screen.
No ACII ouput will be given, if the argument is -1. Debug messages will be written if enable by field_debug, and all error messages will also be printed.

Output: none.

Initial values: The following initial values are used by the program after field_init has been called:

Variable	Content	Value
c	Speed of sound	1540 m/s
fs	Sampling frequency	$100 \cdot 10^6$ Hz
show_times	Whether to print information about the time taken for the calculation (yes = any positive numer). A number large than 2 is taken as the time in seconds between the printing of estimates.	
debug	Whether to show debuging information	0 (no)
use_att	Whether to use attenuation	0 (no)
att	Frequency independent attenuation	0.0 dB/m.
freq_att	Frequency dependent attenuation in around the center frequency att_f0	0.0 dB/[m Hz]
att_f0	Attenuation center frequency in Hz	0.0 Hz
use_rectangles	Whether to use rectangles for describing apertures	1 (yes)
use_triangles	Whether to use triangles for describing apertures	0 (no)
use_lines	Whether to use lines for describing apertures	0 (no)
no_ascii_output	Whether ASCII output is not printed	0 (no, output is printed)
fast_integration	Whether to use fast integration for bound lines and triangles	0 (no)

Initially the program is set to use rectangles for the modeling of transducers. All of the options can be changed by the procedure set_field.

Example: Include the Field II m-files in Matlab's search path and start the Field II simulation system:

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


Purpose: Set the sampling frequency the system uses.

Remember that the pulses used in all apertures must be reset for the new sampling frequency to take effect.

This procedure has been superseed by set_field, and it is for portability reasons better to use set_field.

Calling: set_sampling (fs);

Input: fs - The new sampling frequency.

Output: none.

Purpose: Set various parameters that determins the function of the program.

Calling: set_field (option_name, value);

Input:

use_att	Whether to use attenuation (<> 0 for attenuation)
att	Frequency independent attenuation in dB/m.
freq_att	Frequency dependent attenuation in dB/[m Hz] around the center frequency att_f0.
att_f0	Attenuation center frequency in Hz.
debug	Whether to print debug information (1 = yes)
c	Set the speed of sound in m/s.
fs	Set the sampling frequency.
show_time	Show calculation times during calculation. (yes = any positive numer). A number large than 2 is taken as the time in seconds between the printing of estimates.
use_rectangles	Use rectangles for the apertures. (1 = yes)
use_triangles	Use triangles for describing apertures. (1 = yes)
use_lines	Use lines for describing apertures. (1 = yes)
fast_integration	Whether to use fast integration (1) of the responses for bound lines and triangles. Fast integration uses a simple trapezoidal time integration of the responses, else a Romberg integration, as described in Numerical Receipes, are used.

Output: none.

Example: Set the attenuation to 1.5 dB/cm and 0.5 dB/[MHz cm] around 3 MHz and use this:

```
set_field ('att',1.5*100);
set_field ('Freq_att',0.5*100/1e6);
set_field ('att_f0',3e6);
set_field ('use_att',1);
```

Note that the frequency independent and the frequency dependent terms should correspond, so that the frequency independent attenuation is the same as the frequency dependent term at the center frequency set. This is ensured if `att = Freq_att*att_f0`, else the attenuation can be too big or too low at large depths in tissue.

5.3 Procedures for transducer definition

Field II user's guide **xdc_apodization**

Purpose: Procedure for creating an apodization time line for an aperture

Calling: xdc_apodization (Th, times, values);

Input:

Th	Pointer to the transducer aperture.
times	Time after which the associated apodization is valid.
values	Apodization values. Matrix with one row for each time value and a number of columns equal to the number of physical elements in the aperture. At least one apodization value in each row must be different from zero.

Output: none.

Field II user's guide **xdc_baffle**

Purpose: Procedure for setting the baffle condition for the aperture.

Calling: xdc_baffle (Th, soft_baffle);

Input:

Th	Pointer to the transducer aperture.
soft_baffle	Whether to use the soft-baffle condition: 1 - using soft baffle 0 - using rigid baffle (default for apertures)

Output: none.

Implementation:

For a soft baffle, in which the pressure on the baffle surface is zero, the Rayleigh-Sommerfeld integral is used instead of the standard Rayleigh integral. This is:

$$h_s(\vec{r}_1, t) = \int_S \frac{\delta(t - \frac{|\vec{r}_1 - \vec{r}_2|}{c})}{2\pi \mid \vec{r}_1 - \vec{r}_2 \mid} \cos\varphi \, dS \tag{5.1}$$

Here $\cos\varphi$ is the angle between the line through the field point orthogonal to the aperture plane and the radius of the spherical wave. The angles φ is fixed for a given radius of the projected spherical wave and thus for a given time. It is given by

$$\cos\varphi = \frac{z_p}{R} = \frac{z_p}{ct} \tag{5.2}$$

I can be shown that

$$h_s(\vec{r}_1, t) = \frac{z_p}{ct} h(\vec{r}_1, t). \tag{5.3}$$

where $h(\vec{r}_1, t)$ is the standard spatial impulse response. The spatial impulse response for the soft baffle case is, thus, be found from the normal spatial impulse response by multiplying with $z_p/(ct)$, which is the method employed by the Field II program.

Example:

Create a 16 elements linear array, and divide the physical elements into 2 by 3 mathematical elements to increase the accuracy of the simulation. Then set the soft-baffle boundary condition.

```
%  Set initial parameters

height=5/1000;         %  Height of element [m]
width=1/1000;          %  Width of element [m]
kerf=width/4;          %  Distance between transducer elements [m]
N_elements=16;         %  Number of elements
focus=[0 0 40]/1000;   %  Initial electronic focus

%  Define the transducer

Th = xdc_linear_array (N_elements, width, height, kerf, 2, 3, focus);

%  Set the soft-baffle option

xdc_soft_baffle (Th, 1);
```

Field II user's guide — xdc_center_focus

Purpose: Procedure for setting the center point for the focusing. This point is used as a reference for calculating the focusing delay times and as a starting point for dynamic focusing.

Calling: xdc_center_focus (Th, point);

Input:
Th	Pointer to the transducer aperture.
point	Focus center point.

Output: none.

Field II user's guide — xdc_concave

Purpose: Procedure for creating a concave transducer

Calling: Th = xdc_concave (radius, focal_radius, ele_size);

Input:
radius	Radius of physical elements.
focal_radius	Focal radius.
ele_size	Size of mathematical elements.

Output: Th A pointer to this transducer aperture.

Example of transducer definition:

Create a concave, round transducer with an 8 mm radius and a focal radius of 20 mm and divided it into 1 mm mathematical elements.

```
%  Set initial parameters
```

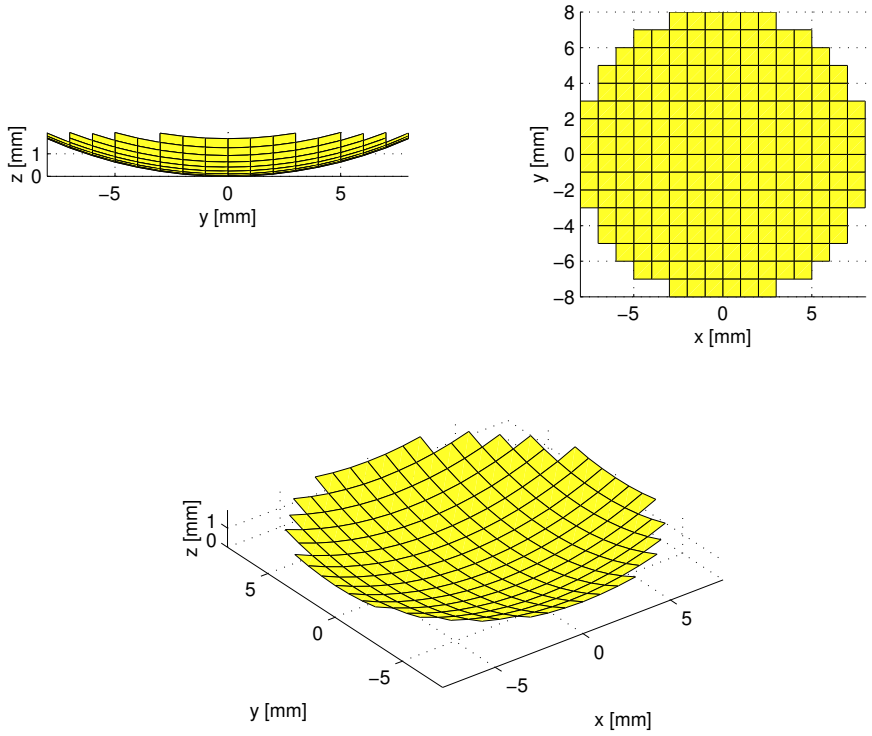

Figure 5.1: Concave, round transducer with a radius of 8 mm divided into 1 by 1 mm mathematical elements.

```
  R=8/1000;               %  Radius of transducer
  Rfocal=20/1000;         %  Focal radius of transducer
  ele_size=1/1000;        %  Size of mathematical elements

%  Define the transducer

  Th = xdc_concave (R, Rfocal, ele_size);
```

The resulting transducer is shown in Fig. 5.1.

Purpose: Procedure for converting an aperture from a rectangular description to a triangular description.

Calling: xdc_convert (Th);

Input: Th Aperture to be converted.

Output: None.

Note: The number of mathematical elements gets to be twice as large since one rectangle is modeled by two triangles.

Purpose: Procedure for creating a convex array aperture.

Calling: Th = xdc_convex_array (no_elements, width, height, kerf, Rconvex, no_sub_x, no_sub_y, focus);

Input:

no_elements	Number of physical elements.
width	Width in x-direction of elements.
height	Width in y-direction of elements.
kerf	Distance in x-direction between elements.
Rconvex	Convex radius.
no_sub_x	Number of sub-divisions in x-direction of elements.
no_sub_y	Number of sub-divisions in y-direction of elements.
focus[]	Fixed focus for array (x,y,z). Vector with three elements.

Output: Th A pointer to this transducer aperture.

Example of transducer definition:

Create a 16 element convex array with a convex radius of 20 mm:

```
%  Set initial parameters

  height=5/1000;         %  Height of element [m]
  width=1/1000;          %  Width of element [m]
  kerf=width/4;          %  Distance between transducer elements [m]
  N_elements=16;         %  Number of elements
  Rconvex=20/1000;       %  Convex radius [m]
  focus=[0 0 40]/1000;   %  Initial electronic focus

%  Define the transducer

  Th = xdc_convex_array (N_elements, width, height, kerf,
                         Rconvex, 1, 5, focus);
```

Note that the radii are quite small in order to show the aperture curvature. The resulting aperture is shown below.

Purpose: Procedure for creating a mechanical elevation focused convex array aperture.

Calling: Th = xdc_convex_focused_array (no_elements, width, height, kerf, Rconvex, Rfocus, no_sub_x, no_sub_y, focus);

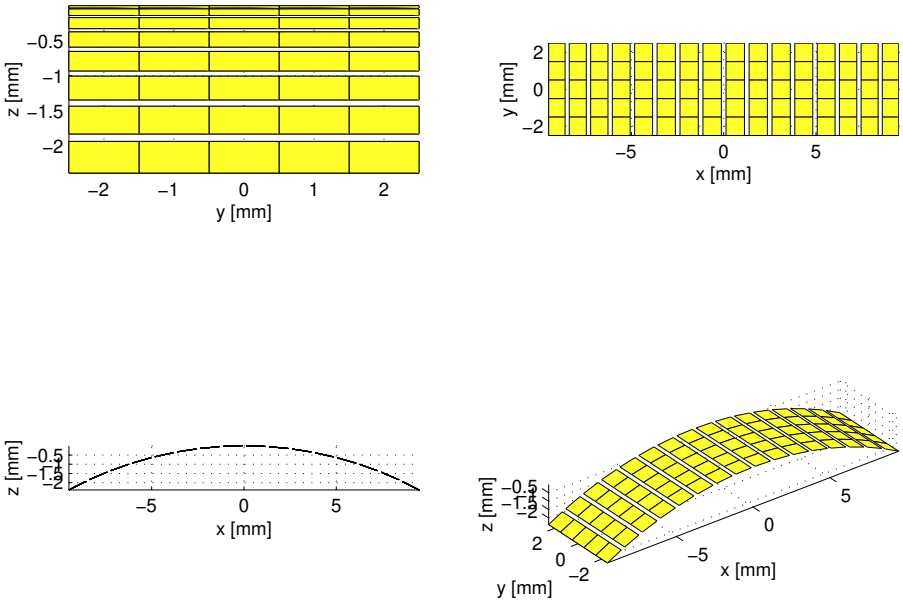

Figure 5.2: Rectangles for a convex array with Rconvex equal to 20 mm.

Input:

	no_elements	Number of physical elements.
	width	Width in x-direction of elements.
	height	Width in y-direction of elements.
	kerf	Distance in x-direction between elements.
	Rconvex	Convex radius.
	Rfocus	Radius of elevation focus.
	no_sub_x	Number of sub-divisions in x-direction of elements.
	no_sub_y	Number of sub-divisions in y-direction of elements.
	focus[]	Fixed focus for array (x,y,z). Vector with three elements.

Output: Th A pointer to this transducer aperture.

Limitations: The kerf and width of the elements must lie within the range: π*Rconvex<=(kerf*(no_elements-1)+width*no_elements). Also all parameters for physical dimensions (width, height, kerf, Rconvex, Rfocus) must be positive.

Example of transducer definition: :

Create a 32 element elevation focused, convex array with an elevation focus at 10 mm and a convex radius of 30 mm:

```
%  Set initial parameters

  height=10/1000;       %  Height of element [m]
  width=1.9/1000;       %  Width of element [m]
  kerf=width/2;         %  Distance between transducer elements [m]
  N_elements=32;        %  Number of elements
  Rfocus=5/1000;        %  Elevation focus [m]
  Rconvex=30/1000;      %  Convex radius [m]
  focus=[0 0 70]/1000;  %  Initial electronic focus

%  Define the transducer
```

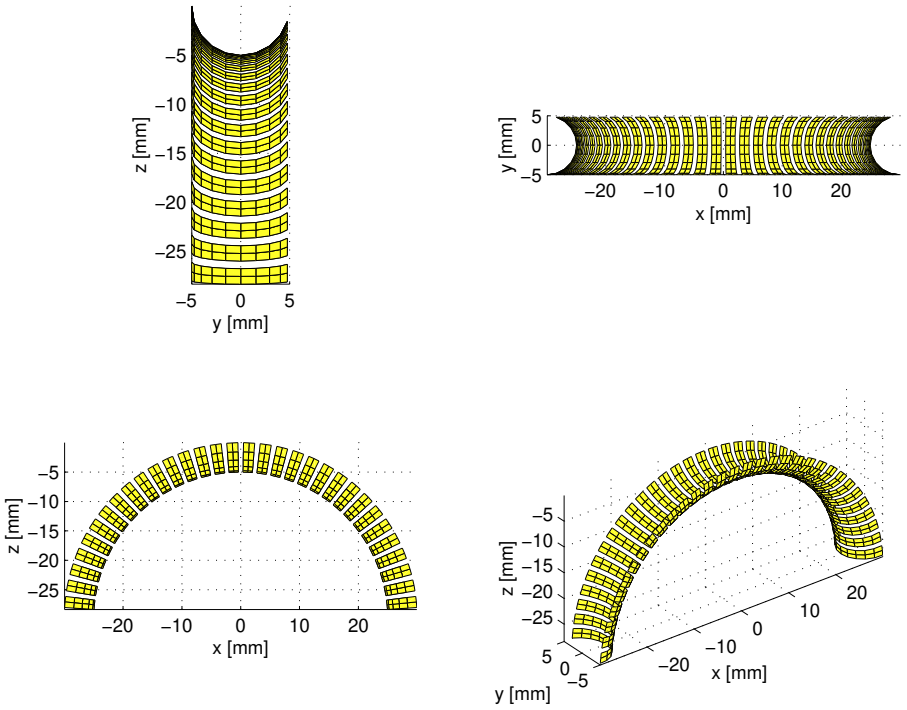

Figure 5.3: Rectangles for an elevation focused, convex array with Rfocus equal to 10 mm and Rconvex equal to 30 mm.

```
Th = xdc_convex_focused_array (N_elements, width, height, kerf,
                               Rconvex, Rfocus, 2, 10, focus);
```

Note that the radii are quite small in order to show the aperture curvature. The resulting aperture is shown below.

Notice also that the physical elements must be subdised in order to model the curvature of the array.

Field II user's guide	**xdc_convex_focused_multirow**

Purpose: Procedure for creating a mechanical elevation focused convex array, where the array has been divided into a number of rows.

Calling: Th = xdc_convex_focused_multirow (no_elements, width, heights, kerf, Rconvex, Rfocus, no_sub_x, no_sub_y, focus);

Input:

no_elem_x	Number of physical elements in x-direction.
width	Width in x-direction of elements.
no_elem_y	Number of physical elements in y-direction.
heights[]	Heights of the element rows in the y-direction. Vector with no_elem_y values.
kerf_x	Width in x-direction between elements.
kerf_y	Gap in y-direction between elements.
Rconvex	Convex radius.
Rfocus	Radius of elevation focus.
no_sub_x	Number of sub-divisions in x-direction of elements.
no_sub_y	Number of sub-divisions in y-direction of elements.
focus[]	Fixed focus for array (x, y, z). Vector with three elements.

Output: Th A pointer to this transducer aperture.

Limitations: The kerf and width of the elements must lie within the range: π*Rconvex<=(kerf*(no_elements-1)+width*no_elements). The combined heghts mus obey: sum(heights)+(no_elem_y-1)*kerf_y)>2*Rfocus. Also all parameters for physical dimensions (width, height, kerf, Rconvex, Rfocus) must be positive.

Example of transducer definition: :

Create a 20 element elevation focused, convex array with 5 rows. The elevation focus is at 10 mm and the convex radius is 30 mm:

```
%  Set initial parameters

  heights=[1 2 3 2 1]/1000;  %  Height of element [m]
  width=3/1000;              %  Width of element [m]
  kerf_x=width/3;            %  Distance between transducer elements [m]
  kerf_y=1/1000;             %  Distance between transducer elements [m]
  N_elem_x=20;               %  Number of elements in x-direction
  Rconvex=30/1000;           %  Convex radius [m]
  Rfocus=7/1000;             %  Elevation focus [m]
  focus=[0 0 70]/1000;       %  Initial electronic focus [m]

%  Define the transducer

  Th = xdc_convex_focused_multirow (N_elem_x, width, length(heights), ...
             heights, kerf_x, kerf_y, Rconvex, Rfocus, 2, 3, focus);
```

Note that the radii are quite small in order to show the aperture curvature. The resulting aperture is shown below.

Notice also that the physical elements must be subdised in order to model the curvature of the array.

Purpose: Procedure for using dynamic focusing for an aperture.

Calling: xdc_dynamic_focus (Th, time, dir_zx, dir_zy);

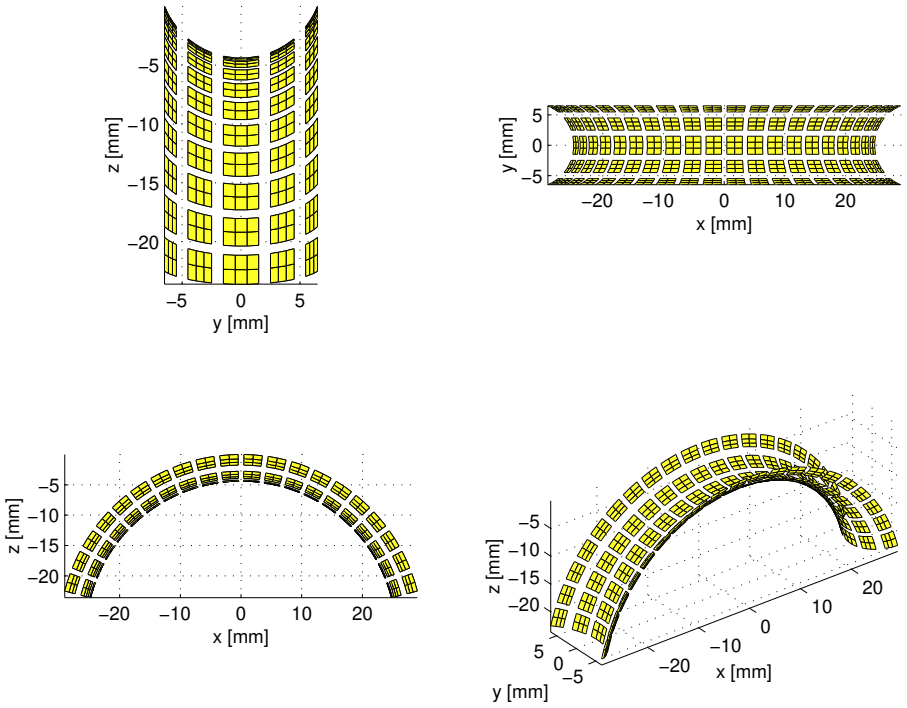

Figure 5.4: Rectangles for an elevation focused, multi-row, convex array with Rfocus equal to 7 mm and Rconvex equal to 30 mm.

Input:

Th	Pointer to the transducer aperture.
time	Time after which the dynamic focus is valid.
dir_zx	Direction (angle) in radians for the dynamic focus. The direction is taken from the center for the focus of the transducer in the z-x plane.
dir_zy	Direction (angle) in radians for the dynamic focus. The direction is taken from the center for the focus of the transducer in the z-y plane.

Output: none.

Field II user's guide — **xdc_excitation**

Purpose: Procedure for setting the excitation pulse of an aperture

Calling: xdc_excitation (Th, pulse);

Input:

Th	Pointer to the transducer aperture.
pulse	Excitation pulse of aperture as row vector

Output: none.

Field II user's guide — **xdc_focus**

Purpose: Procedure for creating a focus time line for an aperture

Calling: xdc_focus (Th, times, points);

Input:

Th	Pointer to the transducer aperture.
times	Time after which the associated focus is valid.
points	Focus points. Vector with three columns (x,y,z) and one row for each field point.

Output: none.

Purpose: Procedure for creating an elevation focused linear array transducer.

Calling: Th = xdc_focused_array (no_elements, width, height, kerf, Rfocus, no_sub_x, no_sub_y, focus);

Input:

no_elements	Number of physical elements.
width	Width in x-direction of elements.
height	Width in y-direction of elements.
kerf	Distance in x-direction between elements.
Rfocus	Radius of elevation focus.
no_sub_x	Number of sub-divisions in x-direction of elements.
no_sub_y	Number of sub-divisions in y-direction of elements.
focus[]	Fixed focus for array (x,y,z). Vector with three elements.

Output: Th A pointer to this transducer aperture.

Example:

Create a 20 element elevation focused, linear array with an elevation focus at 15 mm:

```
%  Set initial parameters

  height=15/1000;       %  Height of element [m]
  width=1.9/1000;       %  Width of element [m]
  kerf=width/3;         %  Distance between transducer elements [m]
  N_elements=20;        %  Number of elements
  Rfocus=15/1000;       %  Elevation focus [m]
  focus=[0 0 70]/1000;  %  Initial electronic focus

%  Define the transducer

  Th = xdc_focused_array (N_elements, width, height, kerf, ...
                          Rfocus, 1, 10, focus);
```

Note that the radii are quite small in order to show the aperture curvature. The resulting aperture is shown below.

Notice also that the physical elements must be subdised in order to model the curvature of the array.

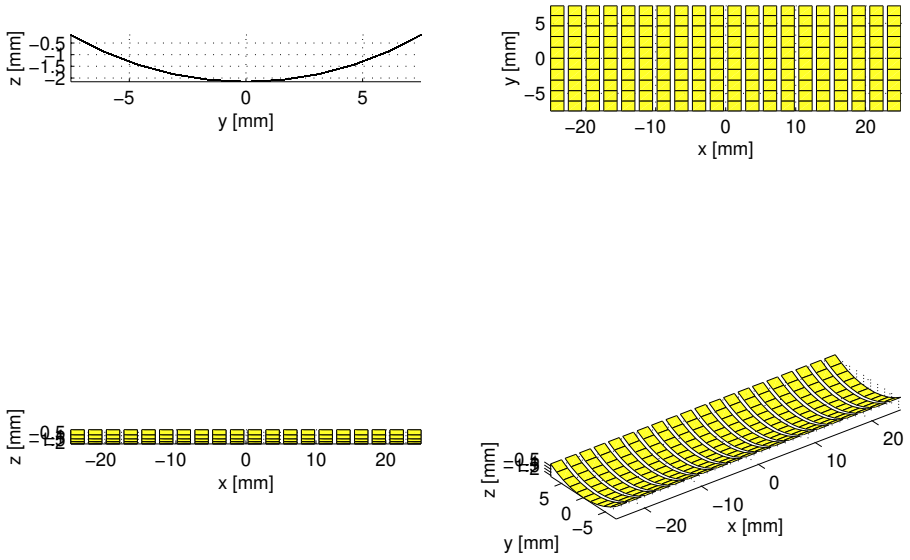

Figure 5.5: Rectangles for an elevation focused, linear array with Rfocus equal to 15 mm.

Purpose: Procedure for creating a linear, elevation focused array transducer with an number of rows (1.5D array)

Calling: Th = xdc_focused_multirow (no_elem_x, width, no_elem_y, heights, kerf_x, kerf_y, Rfocus, no_sub_x, no_sub_y, focus);

Input:

no_elem_x	Number of physical elements in x-direction.
width	Width in x-direction of elements.
no_elem_y	Number of physical elements in y-direction.
heights[]	Heights of the element rows in the y-direction. Vector with no_elem_y values.
kerf_x	Width in x-direction between elements.
kerf_y	Gap in y-direction between elements.
Rfocus	Radius of elevation focus.
no_sub_x	Number of sub-divisions in x-direction of elements.
no_sub_y	Number of sub-divisions in y-direction of elements.
focus[]	Fixed focus for array (x, y, z). Vector with three elements.

Output: Th A pointer to this transducer aperture.

Example:

Create a 6 element elevation focused, multi-row, linear array with an elevation focus at 10 mm:

```
%  Set initial parameters

  heights=[1 2 3 2 1]/1000;  %  Height of element [m]
  width=1.9/1000;            %  Width of element [m]
  kerf_x=width/5;            %  Distance between transducer elements [m]
```

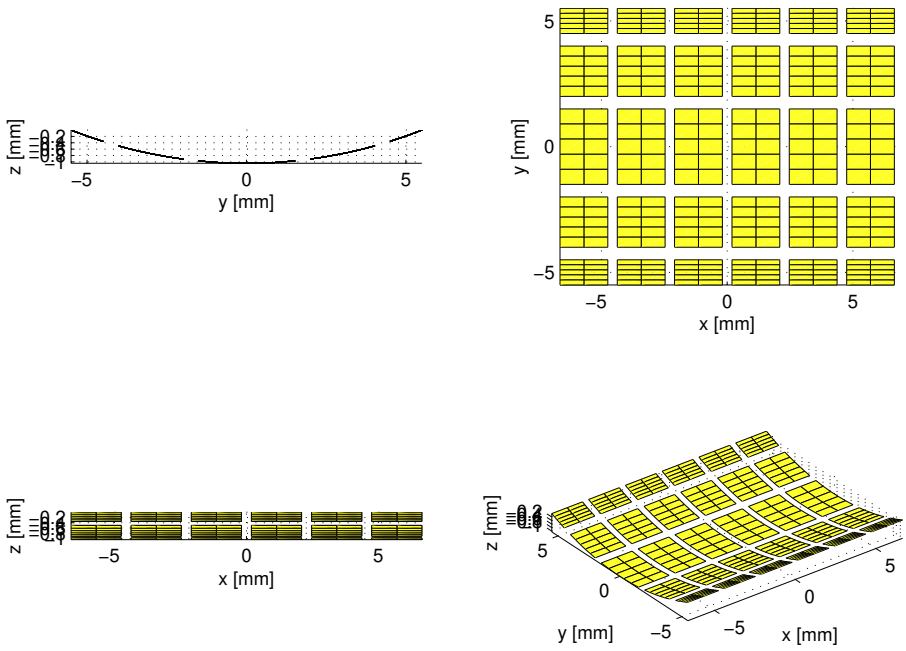

Figure 5.6: Rectangles for an elevation focused, multi-row linear array with Rfocus equal to 10 mm and 5 rows.

```
  kerf_y=0.5/1000;            %  Distance between transducer elements [m]
  N_elem_x=6;                 %  Number of elements in x-direction
  Rfocus=15/1000;             %  Elevation focus
  focus=[0 0 70]/1000;        %  Initial electronic focus

%  Define the transducer

  Th = xdc_focused_multirow (N_elem_x, width, length(heights), heights, ...
                             kerf_x, kerf_y, Rfocus, 2, 5, focus);
```

Note that the radii are quite small in order to show the aperture curvature. The resulting aperture is shown below.

Notice also that the physical elements must be subdised in order to model the curvature of the array.

Field II user's guide — xdc_focus_times and xdc_times_focus

Purpose: Procedure for creating a focus time line for an aperture. All the delay values are supplied by the user. The previous time line is replaced by this time line.

Note that the two procedures perform the same operation. xdc_times_focus has been adde due to compatability with the PC version of Field, and should be the procedure generally used.

Calling: xdc_focus_times (Th, times, delays); or xdc_times_focus (Th, times, delays);

Input:

Th	Pointer to the transducer aperture.
times	Time after which the associated focus is valid.
delays	Delay values. Matrix with one row for each time value and a number of columns equal to the number of physical elements in the aperture.

Output: none.

Field II user's guide — xdc_free

Purpose: Procedure for freeing the storage occupied by an aperture

Calling: xdc_free(Th);

Input: Th Pointer to the transducer aperture.

Output: none.

Field II user's guide — xdc_get

Purpose: Procedure for getting data for an aperture

Calling: data = xdc_get(Th, info_type);

Input:

Th	Pointer to the transducer aperture.
info_type	Which information to get (text string). The possibilities are:
rect	information about rectangular elements
tri	information about triangular elements
focus	focus time line
apo	apodization time line

Output: data data about the aperture

For each mathematical rectangle in the aperture is returned one column of data containing:

Row number	Information
1	Number for the physical element in the aperture.
2	Number for the mathematical element in this physical element.
3	Width of the mathematical element [m]
4	Height of the mathematical element [m].
5	Apodization of the *mathematical* element [m]. Note this is the fixed apodization value set on the mathematical element and not the dynmaic one from the apodization time line set by xdc_apodization.
6	Tangens of the angle with the xz-plane.
7	Tangens of the angle with the yz-plane.
8-10	Position of center (x, y, z) of the mathematical element [m].
11-22	Corners (x, y, z) of the mathematical element [m].
23	Delay value of the mathematical element [s].
24-26	Position of center (x, y, z) of the physical element [m].

For each mathematical triangle in the aperture is returned one column of data containing:

Row number	Information
1	Number for the physical element in the aperture.
2	Number for the mathematical element in this physical element.
3	Apodization of the mathematical element [m].
4-6	Position of center (x, y, z) of the mathematical element [m].
7-15	Corners (x, y, z) of the mathematical element [m].

A matrix with the focusing information is returned, when info_type=”focus”. The matrix contains one column for each focal zone, with the first element indicating the starting time for the focus values and the values following are the time delays for each of the physical elements.

A matrix with the apodizations is returned, when info_type=”apo”. The matrix contains one column for each apodization zone, with the first element indicating the starting time for the apodization selection and the data following are the apodization value for each of the physical elements.

Example:

The geometry and static apodization of an aperture can be shown with the following code:

```
%  Show the transducer surface in a surface plot
%
%  Calling: show_xdc(Th)
%
%  Argument: Th - Transducer handle
%
%  Return:   Plot of the transducer surface on the current figure
%
%  Bote this version onlys shows the defined rectangles
%
%  Version 1.1, June 29, 1998, JAJ

function res = show_xdc (Th)

%  Do it for the rectangular elements

colormap(cool(128));
data = xdc_get(Th,'rect');
[N,M]=size(data);

%  Do the actual display
```

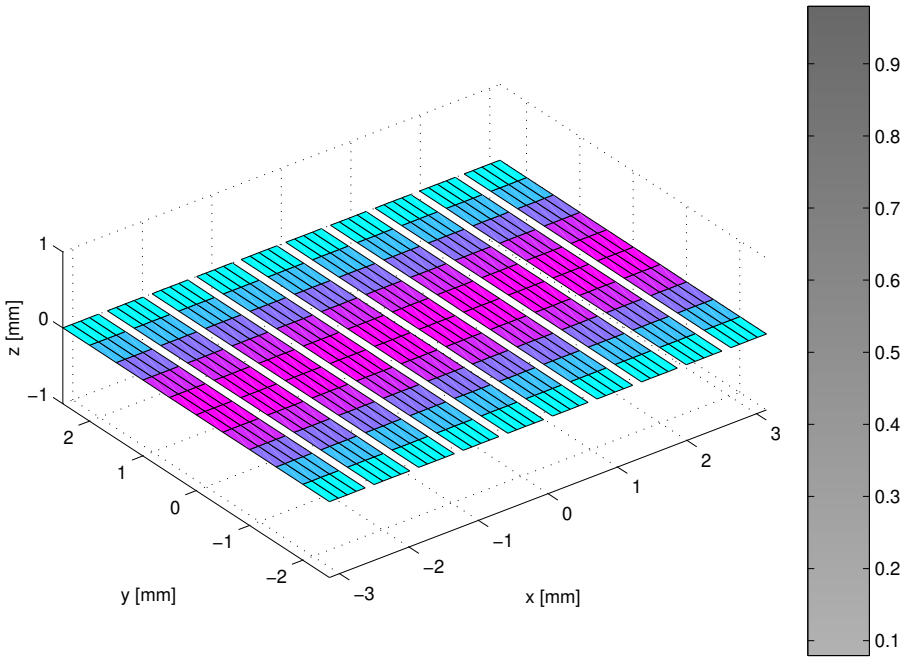

Figure 5.7: Display of the geometry and apodization of a linear array transducer.

```
for i=1:M
  x=[data(11,i), data(20,i); data(14,i), data(17,i)]*1000;
  y=[data(12,i), data(21,i); data(15,i), data(18,i)]*1000;
  z=[data(13,i), data(22,i); data(16,i), data(19,i)]*1000;
  c=data(5,i)*ones(2,2);
  hold on

  surf(x,y,z,c)
  end

%  Put som axis legends on

Hc = colorbar;
view(3)
xlabel('x [mm]')
ylabel('y [mm]')
zlabel('z [mm]')
grid
axis('image')
hold off
```

An example of the output is shown in Fig. 5.7.

Purpose: Procedure for setting the impulse response of an aperture.

Calling: xdc_impulse (Th,pulse);

Input:

Th	Pointer to the transducer aperture.
pulse	Impulse response of aperture as row vector.

Output: none.

Purpose: Procedure for creating a linear array aperture.

Calling: Th = xdc_linear_array (no_elements, width, height, kerf, no_sub_x, no_sub_y, focus);

Input:

no_elements	Number of physical elements.
width	Width in x-direction of elements.
height	Width in y-direction of elements.
kerf	Distance in x-direction between elements.
no_sub_x	Number of sub-divisions in x-direction of elements.
no_sub_y	Number of sub-divisions in y-direction of elements.
focus[]	Fixed focus for array (x,y,z). Vector with three elements.

Output: Th A pointer to this transducer aperture.

Example of transducer definition:

Create a 16 elements linear array, and divide the physical elements into 2 by 3 mathematical elements to increase the accuracy of the simulation.

```
%  Set initial parameters

height=5/1000;         %  Height of element [m]
width=1/1000;          %  Width of element [m]
kerf=width/4;          %  Distance between transducer elements [m]
N_elements=16;         %  Number of elements
focus=[0 0 40]/1000;   %  Initial electronic focus

%  Define the transducer

Th = xdc_linear_array (N_elements, width, height, kerf, 2, 3, focus);
```

The resulting array without the subdivions is shown in Fig. 5.8.

Purpose: Procedure for creating a linear multi-row array aperture, where the transducer has been diced to create a two-dimensional matrix of elements. The individual rows can have different heights.

Calling: Th = xdc_linear_multirow (no_elem_x, width, no_elem_y, heights, kerf_x, kerf_y, no_sub_x, no_sub_y, focus);

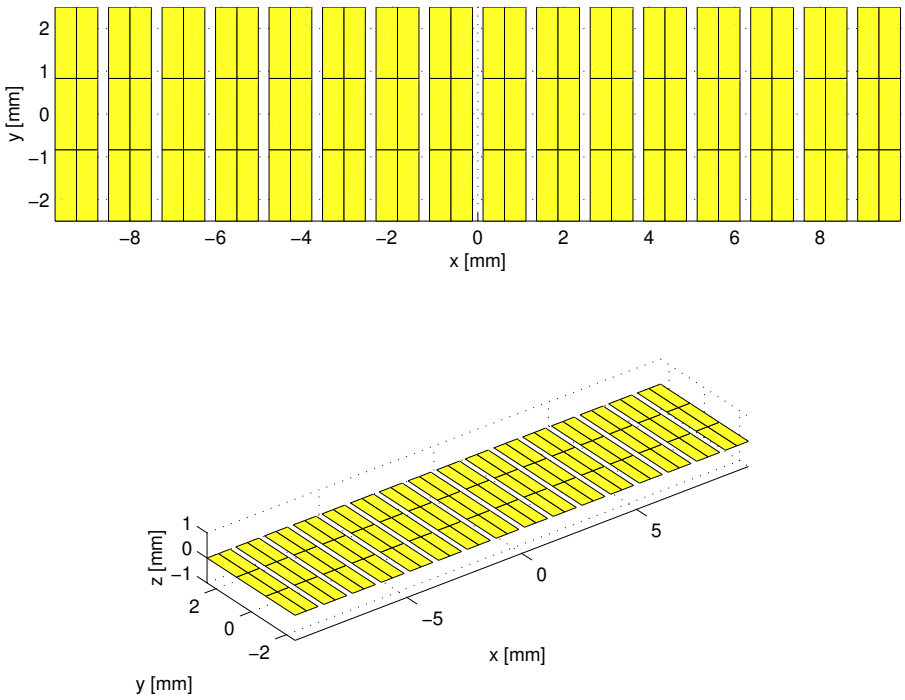

Figure 5.8: Rectangles for a 16 elements linear array transducer.

Input:

no_elem_x	Number of physical elements in x-direction.
width	Width in x-direction of elements.
no_elem_y	Number of physical elements in y-direction.
heights[]	Heights of the element rows in the y-direction. Vector with no_elem_y values.
kerf_x	Width in x-direction between elements.
kerf_y	Gap in y-direction between elements.
no_sub_x	Number of sub-divisions in x-direction of physical elements.
no_sub_y	Number of sub-divisions in y-direction of physical elements.
focus[]	Fixed focus for array (x, y, z). Vector with three elements.

Output: Th A pointer to this transducer aperture.

Geometry:

The geometry of the transducer is shown in Fig. 5.9 with definitions of the relevant parameters. The physical elements are numbered consecutively starting with the one at the most negative x and y coordinate. The element number then increase with increasing x and then with increasing y as shown on the figure. The same numbering scheme is used for the mathematical elements that models the physical elements.

Example of transducer definition:

Create a 16 by 5 elements multirow array, and divide the physical elements into 2 by 3 mathematical elements to increase the accuracy of the simulation.

```
%  Set initial parameters

heights=[1 2 3 2 1]/1000;  %  Height of element [m]
width=1.9/1000;            %  Width of element [m]
kerf_x=width/5;            %  Distance between transducer elements [m]
```

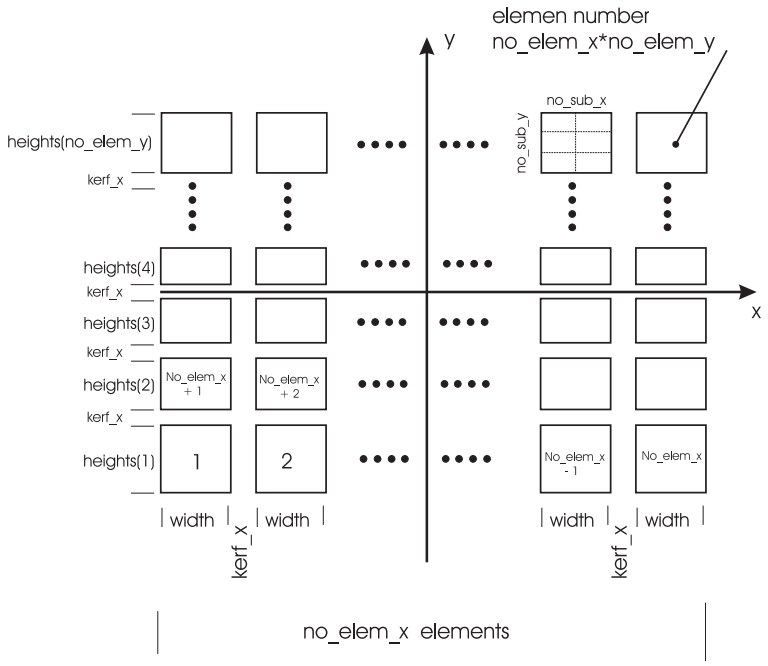

Figure 5.9: Geomtery of multi-row linear array transducer. Currently x and y has been switched.

```
kerf_y=0.5/1000;           %  Distance between transducer elements [m]
no_elem_x=16;              %  Number of elements in x-direction
no_elem_y=length(heights); %  Number of elements in y-direction
focus=[0 0 70]/1000;       %  Initial electronic focus

%  Define the transducer

Th = xdc_linear_multirow (no_elem_x, width, no_elem_y, heights, ...
                          kerf_x, kerf_y, 2, 3, focus);
```

The resulting array is shown in Fig. 5.10.

Example of setting apodization:

Setting a Hanning apodization for the array in the x-direction can be done by:

```
%  The apodization for the aperture

  apo=reshape(hanning(no_elem_x)*ones(1,no_elem_y),1,no_elem_x*no_elem_y);
  xdc_apodization(Th, 0, apo);
```

Note how the apodization values have been packed with one value for each physical element. First a matrix of size no_elem_x by no_elem_y is created, so that the Hanning weighting is the same for the elements in the y-direction and varies in the x-direction. The matrix is then reshaped to a vector with no_elem_y*no_elem_x elements, that can be used by the apodization routine. The setting of focus time values can be done in a similar fashion.

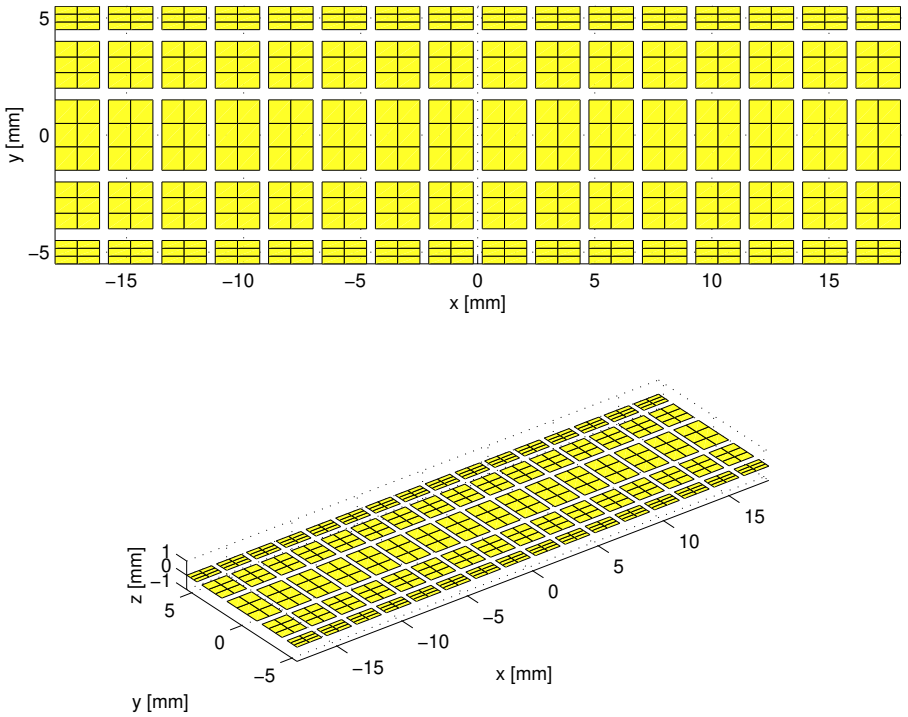

Figure 5.10: Rectangles for a 16 by 5 elements multi-row transducer.

Purpose: Procedure for creating an aperture bounded by a set of lines.

Calling: Th = xdc_lines (lines, center, focus);

Input: lines Information about the lines. One row for each line. The contents is:

Index	Variable	Value
1	no_phys	The number for the physical element starting from one
2	no_mat	The number for the mathematical element starting from one
3	slope	Slope of line (NaN is infinity slope)
4	infinity	True if slope is infinity
5	intersect	Intersection with y-axis (slope<>NaN) or x-axis if slope is infinity
6	above	Whether the active aperture is above or to the left (for infinite slope) of the line

center The center of the physical elements. One line for each element starting from 1.
focus The fixed focus for this aperture.

All dimensions are in meters.

Notice that this procedure will only work for flat elements positioned in the x-y plane.

Output: A handle Th as a pointer to this transducer aperture.

Example: Make a two element transducer aperture with elements 4 mm wide and 10 mm tall. The elements are center at (0,0,0) mm and (4.5, 0, 0) mm. Display information about the aperture after it has been created:

```
lines =[1   1   NaN    1   2/1000  1
        1   1    0     0   5/1000  0
        1   1   NaN    1  -2/1000  0
        1   1    0     0  -5/1000  1
        2   1   NaN    1   6.5/1000  1
        2   1    0     0   5/1000  0
        2   1   NaN    1   2.5/1000  0
        2   1    0     0  -5/1000  1];

center=[0 0 0; 4.5/1000 0 0];
focus=[0 0 70]/1000;
Th = xdc_lines (lines, center, focus);
xdc_show(Th)
```

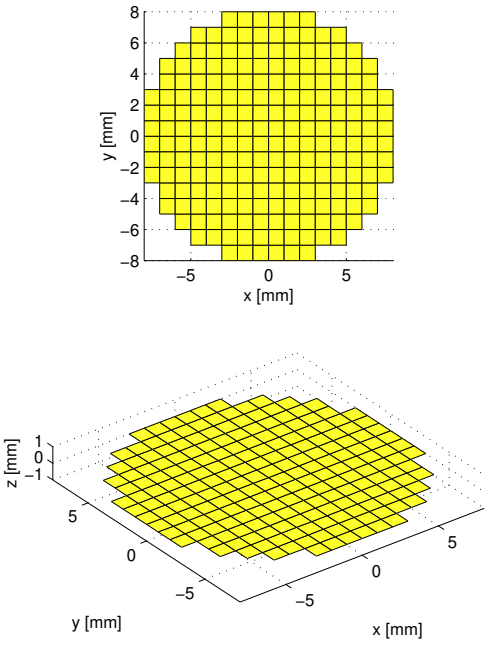

Figure 5.11: Piston transducer with a radius of 8 mm divided into 1 by 1 mm mathematical elements.

Purpose: Procedure for creating a flat, round transducer

Calling: Th = xdc_piston (radius, ele_size);

Input: radius Radius of physical elements.
ele_size Size of mathematical elements.

Output: Th A pointer to this transducer aperture.

Example of transducer definition:

Create a piston transducer with an 8 mm radius and divided into 1 mm mathematical elements.

```
%  Set initial parameters

  R=8/1000;              %  Radius of transducer
  ele_size=1/1000;       %  Size of mathematical elements

%  Define the transducer

  Th = xdc_piston (R,ele_size);
```

The resulting transducer is shown in Fig. 5.11.

Purpose: Procedure for setting the minimum quantization interval that can be used when phasing the transducer.

Remember that the focus time lines must be set again for the quantization to take effect. This setting does not affect the user calculated delays.

Calling: xdc_quantization (Th, min_delay);

Input:

Th	Pointer to the transducer aperture.
min_delay	The smallest delay in seconds that can be used by the system. No quantization is used, if this delay is set to zero.

Output: none.

Purpose: Procedure for creating an aperture consisting of rectangles.

Calling: Th = xdc_rectangles (rect, center, focus);

Input: rect Information about the rectangles. One row for each rectangle. The contents is:

Index	Variable	Value
1	no	The number for the physical aperture starting from one
2-4	x1,y1,z1	First corner coordinate. Must be the lower left corner of the rectangle.
5-7	x2,y2,z2	Second corner coordinate
8-10	x3,y3,z3	Third corner coordinate
11-13	x4,y4,z4	Fourth corner coordinate
14	apo	Apodization value for this element.
15	width	Width of the element (x direction)
16	heigth	Height of the element (y direction)
17-19	c1,c2,c2	Center point of the rectangle

The corner coordiantes points must be in a sorted order, so that they are meet in the clockwise order when going from 1 to 2 to 3 to 4. The rectangle number given must also be in increasing order.

center The center of the physical elements. One line for each element starting from 1.

focus The fixed focus for this aperture.

All dimensions are in meters.

Output: A handle Th as a pointer to this transducer aperture.

Example: Make a one element transducer aperture with a 4 mm wide and 10 mm tall element. The element is center at (0,0,0) mm. Display information about the aperture after it has been created:

```
rect=[1  0/1000  0/1000 0 2/1000  0/1000 0  2/1000 5/1000 0  0/1000  5/1000 0 ...
           1 2/1000 5/1000  1/1000   2.5/1000 0
      1 -2/1000  0/1000 0 0/1000  0/1000 0  0/1000 5/1000 0 -2/1000  5/1000 0 ...
           1 2/1000 5/1000 -1/1000   2.5/1000 0
      1 -2/1000 -5/1000 0 0/1000 -5/1000 0  0/1000 0/1000 0 -2/1000  0/1000 0 ...
           1 2/1000 5/1000 -1/1000  -2.5/1000 0
      1  0/1000 -5/1000 0 2/1000 -5/1000 0  2/1000 0/1000 0  0/1000  0/1000 0 ...
           1 2/1000 5/1000  1/1000  -2.5/1000 0];

  center=[0 0 0];
  focus=[0 0 70]/1000;
  Th = xdc_rectangles (rect, center, focus);
  xdc_show(Th)
```

Purpose: Procedure for showing information about an aperture.

Calling: xdc_show(Th, info_type);

Input: Th Pointer to the transducer aperture.
info_type Which information to show (text string). The possibilities are:
- elements - information about elements
- focus - focus time line
- apo - apodization time line
- all - all information is shown

The argument is optional, and by default all information is shown.

Output: ASCII output on the screen about the aperture

Example: The call to show the focus delays is: xdc_show(Th,'focus');

Purpose: Procedure for creating a focus time line for an aperture. All the delay values are supplied by the user. The previous time line is replaced by this time line.

Note that the two procedures perform the same operation. xdc_times_focus has been adde due to compatability with the PC version of Field, and should be the procedure generally used.

Calling: xdc_focus_times (Th, times, delays); or xdc_times_focus (Th, times, delays);

Input:

Th	Pointer to the transducer aperture.
times	Time after which the associated focus is valid.
delays	Delay values. Matrix with one row for each time value and a number of columns equal to the number of physical elements in the aperture.

Output: none.

Purpose: Procedure for creating a multi-element aperture consisting of triangles.

Calling: Th = xdc_triangles (data, center, focus);

Input: data Information about the triangles. One row for each triangle. The contents is:

Index	Variable	Value
1	no	The number for the physical aperture starting from one
2-4	x1,y1,z1	First corner coordinate
5-7	x2,y2,z2	Second corner coordinate
8-10	x3,y3,z3	Third corner coordinate
11	apo	Apodization value for this element.

The physical element number given must be in increasing order.

center The center of the physical elements. One line for each element starting from 1.

focus The fixed focus for this aperture.

All dimensions are in meters.

Output: A handle Th as a pointer to this transducer aperture.

Example: Make a two element transducer aperture with a 4 mm wide and 10 mm tall elements. Display information about the aperture after it has been created:

```
data=[
1   -0.00405   -0.0050         0   -0.00405         0         0   -0.00205        0         0 1
1   -0.00405   -0.0050         0   -0.00205   -0.0050         0   -0.00205        0         0 1
1   -0.00205   -0.0050         0   -0.00205         0         0   -0.00005        0         0 1
1   -0.00205   -0.0050         0   -0.00005   -0.0050         0   -0.00005        0         0 1
1   -0.00405         0         0   -0.00405    0.0050         0   -0.00205    0.0050        0 1
1   -0.00405         0         0   -0.00205         0         0   -0.00205    0.0050        0 1
1   -0.00205         0         0   -0.00205    0.0050         0   -0.00005    0.0050        0 1
1   -0.00205         0         0   -0.00005         0         0   -0.00005    0.0050        0 1
2    0.00005   -0.0050         0    0.00005         0         0    0.00205        0         0 1
2    0.00005   -0.0050         0    0.00205   -0.0050         0    0.00205        0         0 1
2    0.00205   -0.0050         0    0.00205         0         0    0.00405        0         0 1
2    0.00205   -0.0050         0    0.00405   -0.0050         0    0.00405        0         0 1
2    0.00005         0         0    0.00005    0.0050         0    0.00205    0.0050        0 1
2    0.00005         0         0    0.00205         0         0    0.00205    0.0050        0 1
2    0.00205         0         0    0.00205    0.0050         0    0.00405    0.0050        0 1
2    0.00205         0         0    0.00405         0         0    0.00405    0.0050        0 1];

center=[   -2.0500,     0.0000,     0.0000
            2.0500,     0.0000,     0.0000]/1000;
focus=[0 0 70]/1000;
Th = xdc_triangles (data, center, focus);
xdc_show(Th)
```


Purpose: Procedure for creating a two-dimensional (sparse) array aperture.

Calling: Th = xdc_2d_array (no_ele_x, no_ele_y, width, height, kerf_x, kerf_y, enabled, no_sub_x, no_sub_y, focus);

Input:

no_ele_x	Number of physical elements in x-direction.
no_ele_y	Number of physical elements in y-direction.
width	Width in x-direction of elements.
height	Width in y-direction of elements.
kerf_x	Distance in x-direction between elements.
kerf_y	Distance in y-direction between elements.
enabled	Matrix of size (no_ele_x, no_ele_y) indicating whether the physical element is used. A 1 indicates an enabled element and zero that it is not. enable(1,1) determines the state of the lower left element of the transducer when seen in the $x-y$ plane.
no_sub_x	Number of sub-divisions in x-direction of elements.
no_sub_y	Number of sub-divisions in y-direction of elements.
focus[]	Fixed focus for array (x, y, z). Vector with three elements.

Output: Th A pointer to this transducer aperture.

Example of transducer definition:

Create a fully populated two-dimensional array with 11 by 13 elements:

```
%  Set initial parameters

  height=1.5/1000;      %  Height of element [m]
  width=1/1000;         %  Width of element [m]
  kerf_x=width/5;       %  Distance between transducer elements [m]
  kerf_y=height/2;      %  Distance between transducer elements [m]
  no_ele_x=11;          %  Number of elements in x-direction
  no_ele_y=13;          %  Number of elements in y-direction
  focus=[0 0 60]/1000;  %  Initial electronic focus [m]

%  Find which elements to use

  enabled=ones(no_ele_x, no_ele_y);

%  Define the transducer

  Th = xdc_2d_array (no_ele_x, no_ele_y, width, height, ...
                     kerf_x, kerf_y, enabled, 1, 1, focus);
```

The resulting transducer is shown in Fig. 5.12.

Create a sparsely populated two-dimensional array arranged in the form of a cross:

```
%  Set initial parameters

  height=1.5/1000;      %  Height of element [m]
  width=1/1000;         %  Width of element [m]
  kerf_x=width/5;       %  Distance between transducer elements [m]
  kerf_y=height/2;      %  Distance between transducer elements [m]
  no_ele_x=11;          %  Number of elements in x-direction
  no_ele_y=13;          %  Number of elements in y-direction
  focus=[0 0 60]/1000;  %  Initial electronic focus [m]

%  Find which elements to use

  enabled=zeros(no_ele_x, no_ele_y);
  enabled(:,7)=ones(no_ele_x,1);
```

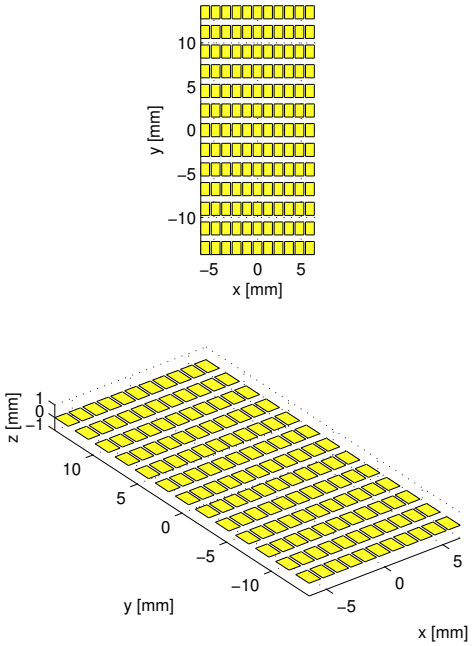

Figure 5.12: Fully populated two-dimensional array with 11 by 13 elements.

```
  enabled(6,:)=ones(1,no_ele_y);

%  Define the transducer

  Th = xdc_2d_array (no_ele_x, no_ele_y, width, height, ...
                     kerf_x, kerf_y, enabled, 1, 1, focus);
```

The resulting transducer is shown in Fig. 5.13.

The elements for a sparsely populated array is stored in order starting with the element with the most negative x and y coordinate, and then first increasing the x coordinate and then the y coordinate. This is the ordering that should be used for delay values and apodization values. The ordering can also be viewed by the routine xdc_show.

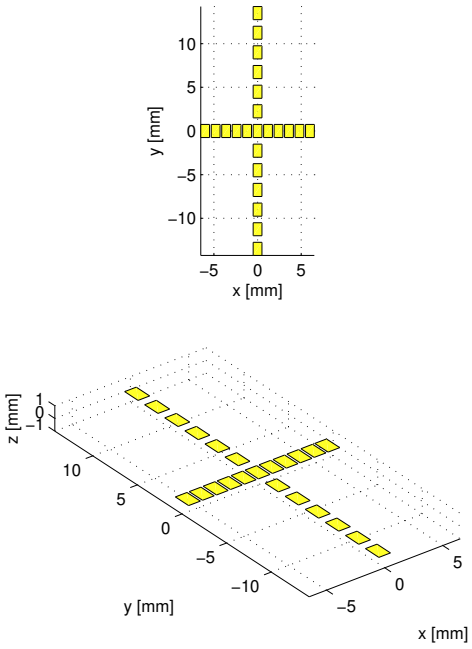

Figure 5.13: Partially populated two-dimensional array with 23 elements.

5.4 Procedures for element manipulation

Field II user's guide **ele_apodization**

Purpose: Procedure for setting the apodization of individual mathematical elements making up the transducer. This apodization is also multiplied onto the spatial impulse response for the mathematical element regardless of the value of the apodization of the physical element and its dynamic apodization.

Calling: ele_apodization (Th, element_no, apo);

Input:

Th	Pointer to the transducer aperture.
element_no	Column vector with one integer for each physical element to set apodization for.
apo	Apodization values. Matrix with one row for each physical element and a number of columns equal to the number of mathematical elements in the aperture.

Output: none.

Example: Define a linear array with 10 elements, where each element is divided into 4 by 10 mathematical elements. Then set the apodization of the mathematical elements to have a Hanning window apodization in the y-direction.

```
%  Set initial parameters

  f0=3e6;                %  Transducer center frequency [Hz]
  fs=100e6;              %  Sampling frequency [Hz]
  c=1540;                %  Speed of sound [m/s]
  lambda=c/f0;           %  Wavelength [m]
  height=5/1000;         %  Height of element [m]
  width=lambda;          %  Width of element [m]
  kerf=width/4;          %  Distance between transducer elements [m]
  N_elements=10;         %  Number of elements
  no_sub_x=4;            %  Number of sub-divisions in x-direction of elements.
  no_sub_y=10;           %  Number of sub-divisions in y-direction of elements.
  focus=[0 0 40]/1000;   %  Initial electronic focus

%  Define the transducer

  Th = xdc_linear_array (N_elements, width, height, kerf, ...
                         no_sub_x, no_sub_y, focus);

%  Set the apodization for the individual mathematical elements

  element_no=(1:N_elements)';
  hann=hanning(no_sub_y)';
  apo=ones(N_elements,1)*reshape(ones(no_sub_x,1)*hann, 1, no_sub_x*no_sub_y);
  ele_apodization (Th, element_no, apo);
```

The resulting transducer and apodization is shown in Fig. 5.14. The colorbar on the right hand side of the figure indicates the apodization value.

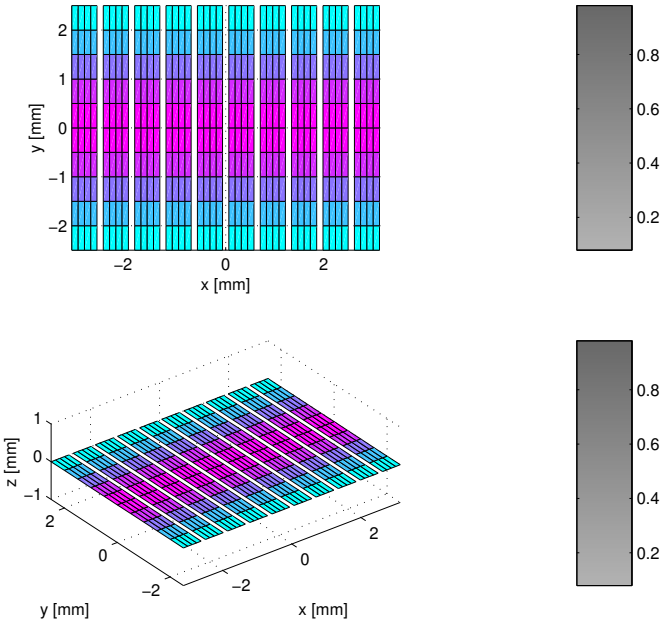

Figure 5.14: Linear array transducer with a fixed apodization of the mathematical elements.

Purpose: Procedure for setting the delay of individual mathematical elements making up the transducer. This can be used to model a fixed lens in front of the aperture.

Calling: ele_delay (Th, element_no, delays);

Input:

Th	Pointer to the transducer aperture.
element_no	Column vector with one integer for each physical element to set delay for.
delays	Delay values. Matrix with one row for each physical element and a number of columns equal to the number of mathematical elements in the aperture.

Output: none.

Example: Define a linear array with 10 elements, where each element is divided into 1 by 21 mathematical elements. Then set the delay of the mathematical elements to have an elevation focus at 30 mm from the aperture.

```
%  Set initial parameters

  f0=3e6;                 %  Transducer center frequency [Hz]
  fs=100e6;               %  Sampling frequency [Hz]
  c=1540;                 %  Speed of sound [m/s]
  lambda=c/f0;            %  Wavelength [m]
  height=10/1000;         %  Height of element [m]
  width=lambda;           %  Width of element [m]
  kerf=width/4;           %  Distance between transducer elements [m]
  N_elements=32;          %  Number of elements
  no_sub_x=1;             %  Number of sub-divisions in x-direction of elements.
  no_sub_y=21;            %  Number of sub-divisions in y-direction of elements.
  focus=[0 0 80]/1000;    %  Initial electronic focus
```

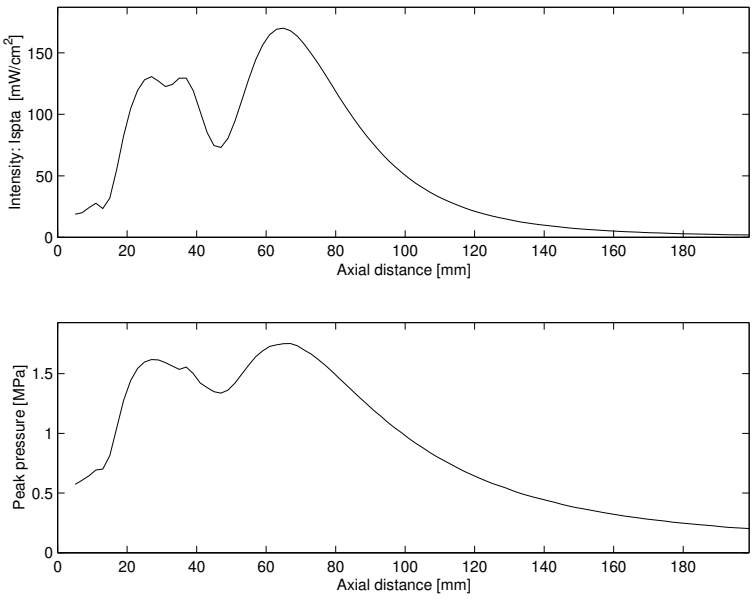

Figure 5.15: Intensity profile for linear array transducer with an elevation focus lens.

```
%  Define the transducer

  Th = xdc_linear_array (N_elements, width, height, kerf, ...
                         no_sub_x, no_sub_y, focus);

%  Set the apodization for the individual mathematical elements

  element_no=(1:N_elements)';
  y=((0:(no_sub_y-1))-(no_sub_y-1)/2)/no_sub_y*height;
  zf=30/1000;
  basic_delay=(zf-sqrt(y.^2+zf.^2))/c;
  delays=ones(N_elements,1)*reshape(ones(no_sub_x,1)*basic_delay,
                                    1, no_sub_x*no_sub_y);
  ele_delay (Th, element_no, delays);
```

An example of the resulting intensity profile is shown in Fig. 5.15.

Field II user's guide **ele_waveform**

Purpose: Procedure for setting the waveform of individual physical elements of the transducer. This can be used to model that the different elements are excitet by different waveforms.

Calling: ele_waveform (Th, element_no, samples);

Input:

Th	Pointer to the transducer aperture.
element_no	Column vector with one integer for each physical element to set delay for.
samples	Sample values for waveform. Matrix with one row for each physical element and a number of columns equal to the number of samples in the waveforms.

Output: none.

Example: Define a linear array with 2 elements, where each physical element is excitet differently.

```
  f0=3e6;                %  Transducer center frequency [Hz]
  f1=1e6;                %  Test frequency 1 [Hz]
  f2=10e6;               %  Test frequency 2 [Hz]
  fs=100e6;              %  Sampling frequency [Hz]
  c=1540;                %  Speed of sound [m/s]
  lambda=c/f0;           %  Wavelength [m]
  height=10/1000;        %  Height of element [m]
  width=lambda;          %  Width of element [m]
  kerf=width/4;          %  Distance between transducer elements [m]
  N_elements=2;          %  Number of elements
  no_sub_x=1;            %  Number of sub-divisions in x-direction of elements.
  no_sub_y=1;            %  Number of sub-divisions in y-direction of elements.
  focus=[0 0 80]/1000;   %  Initial electronic focus

%  Define the transducer

  Th = xdc_linear_array (N_elements, width, height, kerf, ...
                         no_sub_x, no_sub_y, focus);

%  Set the waveforms

  waveform1=sin(2*pi*f1*(0:1/fs:4/f1));
  element_no=1;
  ele_waveform (Th, element_no, waveform1);
  waveform2=sin(2*pi*f2*(0:1/fs:4/f2));
  element_no=2;
  ele_waveform (Th, element_no, waveform2);

%  calculate the field to see the effekt

  [h,t] = calc_h (Th,[0 0 60]/1000);
  plot((0:length(h)-1)/fs+t,h)
  ylabel('Response')
  xlabel('Time [s]')
```

An example of the resulting response is shown in Fig. 5.16.

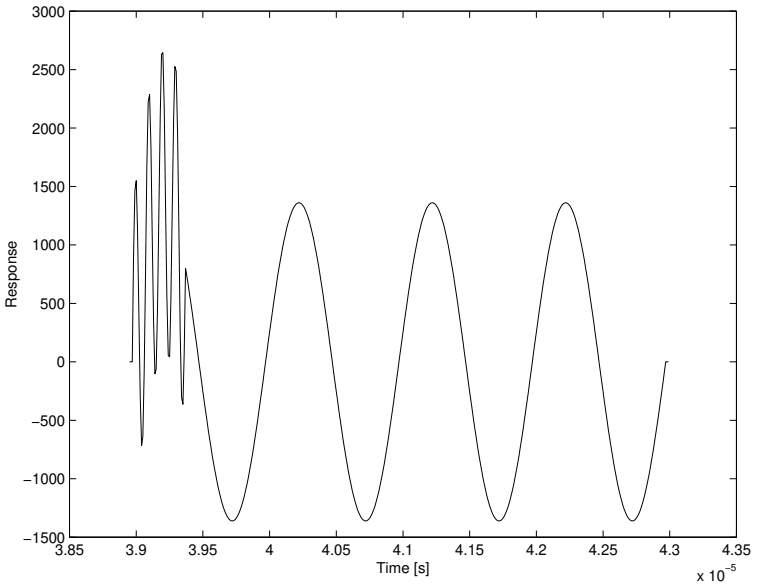

Figure 5.16: Example of calculated response when using different physical element excitations.

5.5 Procedures for field calculation

Field II user's guide **calc_h**

Purpose: Procedure for calculating the spatial impulse response for an aperture.

Calling: [h, start_time] = calc_h(Th,points);

Input:
- Th — Pointer to the transducer aperture.
- points — Field points. Vector with three columns (x,y,z) and one row for each field point.

Output:
- h — Spatial impulse response in m/s.
- start_time — The time for the first sample in h.

Purpose: Procedure for calculating the pulse echo field.

Calling: [hhp, start_time] = calc_hhp(Th1, Th2, points);

Input:

Th1	Pointer to the transmit aperture.
Th2	Pointer to the receive aperture.
points	Field points. Vector with three columns (x,y,z) and one row for each field point.

Output:

hhp	Received voltage trace.
start_time	The time for the first sample in hhp.

Purpose: Procedure for calculating the emitted field.

Calling: [hp, start_time] = calc_hp(Th, points);

Input:

Th	Pointer to the transmit aperture.
points	Field points. Vector with three columns (x,y,z) and one row for each field point.

Output:

hp	Emitted pressure field.
start_time	The time for the first sample in field.

Field II user's guide **calc_scat**

Purpose: Procedure for calculating the received signal from a collection of scatterers.

Calling: [scat, start_time] = calc_scat(Th1, Th2, points, amplitudes);

Input:

Th1	Pointer to the transmit aperture.
Th2	Pointer to the receive aperture.
points	Scatterers. Vector with three columns (x,y,z) and one row for each scatterer.
amplitudes	Scattering amplitudes. Row vector with one entry for each scatterer.

Output:

scat	Received voltage trace.
start_time	The time for the first sample in scat.

Field II user's guide **calc_scat_all**

Purpose: Procedure for calculating the received signal from a collection of scatterers and for each combination of ttransmit and receive elements in the aperture. This corresponds to a full synthetic aperture scan, with each element transmitting and all elements receiving. Note that the raw data is calculated. No focusing or apodization is employed on the data and this has to be done on the data afterwards.

Note that this routine can give a lot of data, when many elements are used. A 32 elements transducer gives 1024 signals. The data can therefore be decimated after calculation of the response. This still gives exactly the same response, but with fewer samples in the result. It just has to be ensured that the decimated sampling frequency (f_s/dec_factor) is large enough to not give aliasing in the response.

Calling: [scat, start_time] = calc_scat_all (Th1, Th2, points, amplitudes, dec_factor);

Input:

Th1	Pointer to the transmit aperture.
Th2	Pointer to the receive aperture.
points	Scatterers. Vector with three columns (x,y,z) and one row for each scatterer.
dec_factor	Decimation factor for the output sampling rate. The sampling frequency is then fs/dec_factor, where fs is the sampling frequency set in the program. The factor must be an integer.
amplitudes	Scattering amplitudes. Row vector with one entry for each scatterer.

Output:

scat	Received voltage trace. The matrix is organized with one received signal for each receiving element and this is repeated for all transmitting element, so the first signal is transmitting with element one and receiving with element one. The transmitting with element one receiving with element two and so forth. The it is repeated with transmitting element 2, etc.
start_time	The time for the first sample in scat.

Example: Calculate the received response from all elements of a linear array with 3 transmitting and 16 receiving elements and plot the responses and the summed response (see Fig. 5.17).

```
%  Set initial parameters

f0=3e6;                 %  Transducer center frequency [Hz]
fs=100e6;               %  Sampling frequency [Hz]
c=1540;                 %  Speed of sound [m/s]
lambda=c/f0;            %  Wavelength [m]
```

```
height=5/1000;        %  Height of element [m]
width=1/1000;         %  Width of element [m]
kerf=width/5;         %  Distance between transducer elements [m]
N_elements=3;         %  Number of elements
N_elements2=16;       %  Number of elements
focus=[0 0 40]/1000;  %  Initial electronic focus

%  Define the transducers

Th = xdc_linear_array (N_elements, width, height, kerf, 2, 3, focus);
Th2 = xdc_linear_array (N_elements2, width, height, kerf, 2, 3, focus);

%  Set the impulse response and excitation of the emit aperture

impulse_response=sin(2*pi*f0*(0:1/fs:2/f0));
impulse_response=impulse_response.*hanning(max(size(impulse_response)))';
xdc_impulse (Th, impulse_response);
xdc_impulse (Th2, impulse_response);

excitation=sin(2*pi*f0*(0:1/fs:2/f0));
xdc_excitation (Th, excitation);

%  Define a small phantom with scatterers

N=200;              %  Number of scatterers
x_size = 20/1000;   %  Width of phantom [mm]
y_size = 10/1000;   %  Transverse width of phantom [mm]
z_size = 20/1000;   %  Height of phantom [mm]
z_start = 5/1000;   %  Start of phantom surface [mm];

%  Create the general scatterers

x = (rand (N,1)-0.5)*x_size;
y = (rand (N,1)-0.5)*y_size;
z = rand (N,1)*z_size + z_start;
positions=[x y z];

%  Generate the amplitudes with a Gaussian distribution

amp=randn(N,1);

%  Do the calculation

[v,t]=calc_scat_all (Th, Th2, positions, amp, 1);

%  Plot the individual responses

[N,M]=size(v);
scale=max(max(v));
v=v/scale;
for i=1:M
  plot((0:N-1)/fs+t,v(:,i)+i,'b'), hold on
  end
hold off
```

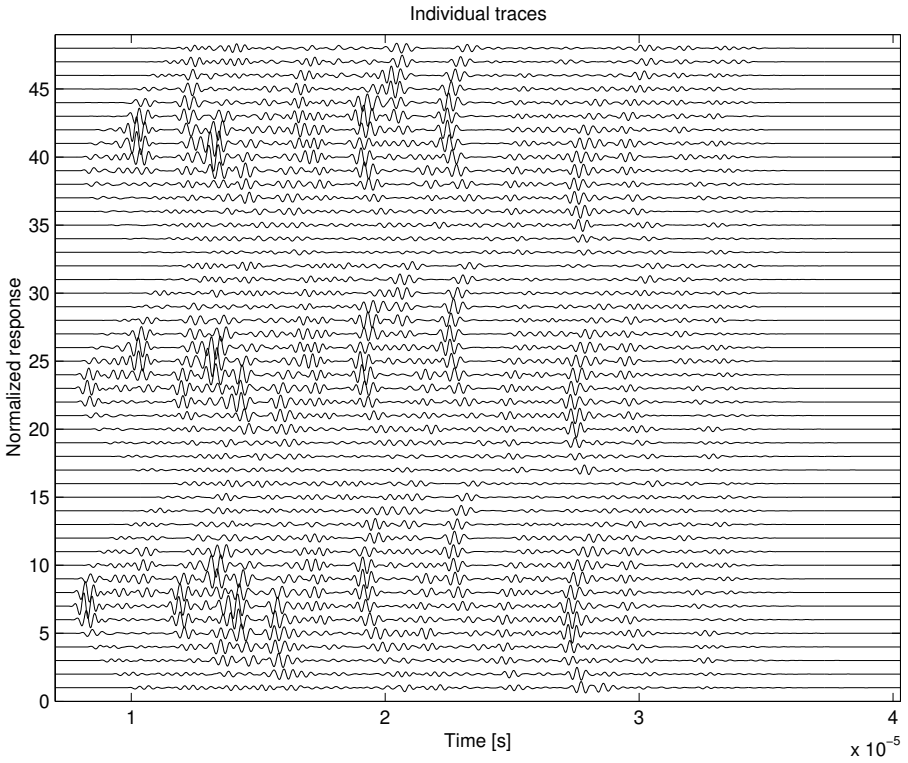

Figure 5.17: Received voltage traces from the individual elements of a 16 elements linear array transducer, when transmitting with three different elements.

```
title('Individual traces')
xlabel('Time [s]')
ylabel('Normalized response')
axis([t t+N/fs 0 M+1])
```


Purpose: Procedure for calculating the received signal from a collection of scatterers and for each of the elements in the receiving aperture.

Calling: [scat, start_time] = calc_scat_multi(Th1, Th2, points, amplitudes);

Input:

Th1	Pointer to the transmit aperture.
Th2	Pointer to the receive aperture.
points	Scatterers. Vector with three columns (x,y,z) and one row for each scatterer.
amplitudes	Scattering amplitudes. Row vector with one entry for each scatterer.

Output:

scat	Received voltage trace. One signal for each physical element in the receiving aperture.
start_time	The time for the first sample in scat.

Example: Calculate the received response from all elements of a linear array and plot the responses and the summed response (see Fig. 5.18).

```
%  Set initial parameters

f0=3e6;               %  Transducer center frequency [Hz]
fs=100e6;             %  Sampling frequency [Hz]
c=1540;               %  Speed of sound [m/s]
lambda=c/f0;          %  Wavelength [m]
height=5/1000;        %  Height of element [m]
width=1/1000;         %  Width of element [m]
kerf=width/4;         %  Distance between transducer elements [m]
N_elements=32;        %  Number of elements
focus=[0 0 40]/1000;  %  Initial electronic focus

%  Define the transducer

Th = xdc_linear_array (N_elements, width, height, kerf, 2, 3, focus);

%  Set the impulse response and excitation of the emit aperture

impulse_response=sin(2*pi*f0*(0:1/fs:2/f0));
impulse_response=impulse_response.*hanning(max(size(impulse_response)))';
xdc_impulse (Th, impulse_response);

excitation=sin(2*pi*f0*(0:1/fs:2/f0));
xdc_excitation (Th, excitation);

%  Do the calculation

[v,t]=calc_scat_multi (Th, Th, [0 0 20]/1000, 1);

%  Plot the individual responses

subplot(211)
[N,M]=size(v);
v=v/max(max(v));
for i=1:N_elements
  plot((0:N-1)/fs+t,v(:,i)+i), hold on
  end
hold off
title('Individual traces')
xlabel('Time [s]')
ylabel('Normalized response')
subplot(212)
plot((0:N-1)/fs+t,sum(v'))
title('Summed response')
xlabel('Time [s]')
ylabel('Normalized response')
```

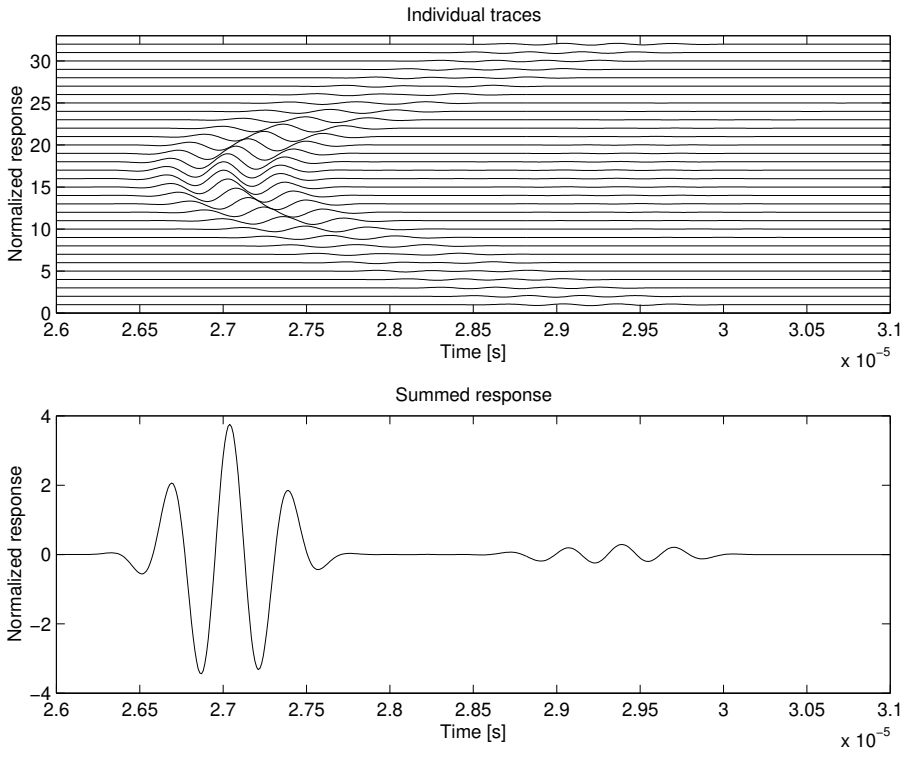

Figure 5.18: Received voltage traces from the individual elements of a linear array transducer (top) and the sum of all the individual responses (bottom).

CHAPTER

SIX

Examples

6.1 Phased array imaging

This examples shows how the procedures can be used for making a phased array scan of a point target.

```
%  Example of use of the new Field II program running under Matlab
%
%  This example shows how a phased array B-mode system scans an image
%
%  This script assumes that the field_init procedure has been called
%
%  Example by Joergen Arendt Jensen, Nov. 28, 1995.

%  Generate the transducer apertures for send and receive

f0=3e6;                  %  Transducer center frequency [Hz]
fs=100e6;                %  Sampling frequency [Hz]
c=1540;                  %  Speed of sound [m/s]
lambda=c/f0;             %  Wavelength
element_height=5/1000;   %  Height of element [m]
kerf=0.1/1000;           %  Kerf [m]
focus=[0 0 70]/1000;     %  Fixed focal point [m]

%  Generate aperture for emission

emit_aperture = xdc_linear_array (128, lambda/2, element_height, kerf, 1, 1,focus);

%  Set the impulse response and excitation of the emit aperture

impulse_response=sin(2*pi*f0*(0:1/fs:2/f0));
impulse_response=impulse_response.*hanning(max(size(impulse_response)))';
xdc_impulse (emit_aperture, impulse_response);

excitation=sin(2*pi*f0*(0:1/fs:2/f0));
xdc_excitation (emit_aperture, excitation);

%  Generate aperture for reception

receive_aperture = xdc_linear_array (128, lambda/2, element_height, kerf, 1, 1,focus);

%  Set the impulse response for the receive aperture

xdc_impulse (receive_aperture, impulse_response);


%   Do phased array imaging

point_position=[0 0 70]/1000;      %  Position of the point to be imaged
no_lines=50;                       %  Number of A-lines in image
sector=20 * pi/180;                %  Size of image sector
```

```
d_theta=sector/no_lines;           %  Increment in angle for 90 deg. image

%  Pre-allocate some storage

image_data=zeros(800,no_lines);

theta= -sector/2;
for i=1:no_lines

   %   Set the focus for this direction

  xdc_focus (emit_aperture, 0, [70*sin(theta) 0 70*cos(theta)]/1000);
  xdc_focus (receive_aperture, 0, [70*sin(theta) 0 70*cos(theta)]/1000);

  %   Calculate the received response

  [v, t1]=calc_scat(emit_aperture, receive_aperture, point_position, 1);

  %  Store the result

  image_data(1:max(size(v)),i)=v';
  times(i) = t1;

  %  Steer in another angle

  theta = theta + d_theta;
  end

%  Here the display of the data is inserted

plot(image_data)
```

6.2 Linear array imaging

This examples shows how the procedures can be used for making a linear array scan of an artificial phantom.

```
%  Example of use of the new Field II program running under Matlab
%
%  This example shows how a linear array B-mode system scans an image
%
%  This script assumes that the field_init procedure has been called
%
%  Example by Joergen Arendt Jensen, Version 2.0, March 22, 2011.

%  Generate the transducer apertures for send and receive

f0=3e6;                 %  Transducer center frequency [Hz]
fs=100e6;               %  Sampling frequency [Hz]
c=1540;                 %  Speed of sound [m/s]
lambda=c/f0;            %  Wave length [m]
width=lambda;           %  Width of element
element_height=5/1000;  %  Height of element [m]
kerf=width/20;          %  Kerf [m]
focus=[0 0 50]/1000;    %  Fixed focal point [m]
N_elements=192;         %  Number of elements in the transducer
N_active=64;            %  Active elements in the transducer

%  Set the sampling frequency

set_sampling(fs);

%  Generate aperture for emission

emit_aperture = xdc_linear_array (N_elements, width, element_height, kerf, 1, 5, focus);

%  Set the impulse response and excitation of the emit aperture

impulse_response=sin(2*pi*f0*(0:1/fs:2/f0));
impulse_response=impulse_response.*hanning(max(size(impulse_response)))';
xdc_impulse (emit_aperture, impulse_response);

excitation=sin(2*pi*f0*(0:1/fs:2/f0));
xdc_excitation (emit_aperture, excitation);

%  Generate aperture for reception

receive_aperture = xdc_linear_array (N_elements, width, element_height, kerf, 1, 5, focus);

%  Set the impulse response for the receive aperture

xdc_impulse (receive_aperture, impulse_response);

%   Load the computer phantom

[phantom_positions, phantom_amplitudes] = cyst_phantom(10000);

%   Do linear array imaging

no_lines=N_elements-N_active+1;          %  Number of A-lines in image
dx=width;                                %  Increment for image
z_focus=50/1000;

%  Pre-allocate some storage

image_data=zeros(1,no_lines);

for i=1:no_lines
i
```

```
   %  Find position for imaging

   x=(i-1-no_lines/2)*dx;

   %   Set the focus for this direction

  xdc_center_focus (emit_aperture, [x 0 0]);
  xdc_focus (emit_aperture, 0, [x 0 z_focus]);
  xdc_center_focus (receive_aperture, [x 0 0]);
  xdc_focus (receive_aperture, 0, [x 0 z_focus]);

  %  Set the active elements using the apodization

  apo=[zeros(1, i-1) hamming(N_active)' zeros(1, N_elements-N_active-i+1)];
  xdc_apodization (emit_aperture, 0, apo);
  xdc_apodization (receive_aperture, 0, apo);

  %   Calculate the received response

  [v, t1]=calc_scat(emit_aperture, receive_aperture, phantom_positions, phantom_amplitudes);

  %  Store the result

  image_data(1:max(size(v)),i)=v;
  times(i) = t1;
  end

%   Free space for apertures

xdc_free (emit_aperture)
xdc_free (receive_aperture)

%  Adjust the data in time and display it as
%  a gray scale image

min_sample=min(times)*fs;
for i=1:no_lines
  rf_env=abs(hilbert([zeros(round(times(i)*fs-min_sample),1); image_data(:,i)]));
  env(1:size(rf_env,1),i)=rf_env;
  end

%  make logarithmic compression to a 60 dB dynamic range
%  with proper units on the axis

env_dB=20*log10(env);
env_dB=env_dB-max(max(env_dB));
env_gray=127*(env_dB+60)/60;
depth=((0:size(env,1)-1)+min_sample)/fs*c/2;
x=((1:no_lines)-no_lines/2)*dx;
image(x*1000, depth*1000, env_gray)
xlabel('Lateral distance [mm]')
ylabel('Depth [mm]')
axis('image')
colormap(gray(128))
title('Image of cyst phantom (60 dB dynamic range)')
```

6.2.1 Computer cyst phantom

Code for generating an artificial phantom with point scatterers and a cyst.

```
%  Create a computer model of a cyst phantom. The phantom contains
%  five point targets separated by 5 mm and a 10 mm water filled cyst.
%  All scatterers are situated in a box of (x,y,z)=(40,10,50) mm.
```

```
%
%   Calling: [positions, amp] = cyst_phantom (N);
%
%  Parameters:  N - Number of scatterers in the phantom
%
%  Output:      positions  - Positions of the scatterers.
%               amp        - amplitude of the scatterers.
%
%  Version 1.1, March 22, 2011 by Joergen Arendt Jensen

function [positions, amp] = cyst_phantom (N)

x_size = 40/1000;   %  Width of phantom [m]
y_size = 10/1000;   %  Transverse width of phantom [m]
z_size = 50/1000;   %  Height of phantom [m]
z_start = 30/1000;  %  Start of phantom surface [m];

%  Create the general scatterers

x = (rand (N,1)-0.5)*x_size;
y = (rand (N,1)-0.5)*y_size;
z = rand (N,1)*z_size + z_start;

%  Generate the amplitudes with a Gaussian distribution

amp=randn(N,1);

%  Make the cyst and set the amplitudes to zero inside

r=5/1000;    %  Radius of cyst [m]
xc=0/1000;    %  Place of cyst [m]
zc=25/1000+z_start;

inside = ( ((x-xc).^2 + (z-zc).^2) < r^2);
amp = amp .* (1-inside);

%  Place the point scatterers in the phantom

dz=z_size/10;
for i=N-9:N
  x(i) = -15/1000;
  y(i) = 0;
  z(i) = z_start + (i-N+9)*dz;
  amp(i) = 100;
  end

%  Return the variables
positions=[x y z];
end
```

6.3 Flow data generation

This examples shows how the procedures can be used for making flow data from a number of scatteres in a tube.

```
%  Example of use of the new Field II program running under Matlab
%
%  This example shows how flow can simulated
%
%  This script assumes that the field_init procedure has been called
%
%  Example by Joergen Arendt Jensen, March 22, 2011.

%  Generate the transducer apertures for send and receive

f0=3e6;                 %  Transducer center frequency [Hz]
fs=100e6;               %  Sampling frequency [Hz]
c=1540;                 %  Speed of sound [m/s]
lambda=c/f0;            %  Wavelength
element_height=5/1000;  %  Height of element [m]
kerf=0.1/1000;          %  Kerf [m]
focus=[0 0 70]/1000;    %  Fixed focal point [m]

%  Generate aperture

aperture = xdc_linear_array (128, lambda/2, element_height, kerf, 1, 1,focus);

%  Set the impulse response and excitation of the emit aperture

impulse_response=sin(2*pi*f0*(0:1/fs:2/f0));
impulse_response=impulse_response.*hanning(max(size(impulse_response)))';
xdc_impulse (aperture, impulse_response);

excitation=sin(2*pi*f0*(0:1/fs:8/f0));
xdc_excitation (aperture, excitation);

%  Set the seed of the random number generator

randn('seed',sum(100*clock))

%  Initialize the ranges for the scatterers
%  Notice that the coordinates are in meters

x_range=0.015;   %  x range for the scatterers  [m]
y_range=0.015;   %  y range for the scatterers  [m]
z_range=0.015;   %  z range for the scatterers  [m]
z_offset=0.70;   %  Offset of the mid-point of the scatterers [m]
R=0.005;         %  Radius of blood vessel [m]

%  Set the number of scatterers. It should be roughly
%  10 scatterers per resolution cell

c=1540;    %  Ultrasound propagation velocity [m/s]
f0=3e6;    %  Center frequency of transducer  [Hz]
lambda=c/f0;
N=round(10*x_range/(5*lambda)*y_range/(5*lambda)*z_range/(lambda*2));
disp([num2str(N),' Scatterers'])

%  Generate the coordinates and amplitude
%  Coordinates are rectangular within the range.
%  The amplitude has a Gaussian distribution.

x=x_range*(rand(1,N)-0.5);
y=y_range*(rand(1,N)-0.5);
z=z_range*(rand(1,N)-0.5);

%  Find which scatterers that lie within the blood vessel
```

```
r=(y.^2+z.^2).^0.5;
within_vessel= (r < R)';

%  Assign an amplitude and a velocity for each scatterer

v0=0.5;   %  Largest velocity of scatterers [m/s]
velocity=v0*(1-(r/R).^2).*within_vessel';

blood_to_stationary= 0.1;   %  Ratio between amplitude of blood to stationary tissue
amp=randn(N,1).*((1-within_vessel) + within_vessel*blood_to_stationary);

%  Calculate a suitable Tprf

theta=45/180*pi;
f_max=2*v0*cos(theta)/c*f0;
fprf=3*f_max

Tprf=1/fprf;  %  Time between pulse emissions  [s]
Nshoots=128;  %  Number of shoots

%  Find the response by calling field

for i=1:Nshoots
i

  %  Generate the rotated and offset block of sample

  xnew=x*cos(theta)+z*sin(theta);
  znew=z*cos(theta)-x*sin(theta) + z_offset;
  scatterers=[xnew; y; znew;]' ;

  %   Calculate the received response

  [v, t1]=calc_scat(aperture, aperture, scatterers, amp);

  %  Store the result

  image_data(1:max(size(v)),i)=v;
  times(i) = t1;

  %  Propagate the scatterers and alias them
  %  to lie within the correct range

  x=x + velocity*Tprf;
  outside_range= (x > x_range/2);
  x=x - x_range*outside_range;
  end

%  Here the display of the data is inserted

plot(image_data)
```

BIBLIOGRAPHY

[1] J. A. Jensen and N. B. Svendsen. Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers. *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, 39:262–267, 1992.

[2] J. A. Jensen. Field: A program for simulating ultrasound systems. *Med. Biol. Eng. Comp.*, 10th Nordic-Baltic Conference on Biomedical Imaging, Vol. 4, Supplement 1, Part 1:351–353, 1996b.

[3] S. A. Goss, R. L. Johnston, and F. Dunn. Comprehensive compilation of empirical ultrasonic properties of mammalian tissues. *J. Acoust. Soc. Am.*, 64:423–457, 1978.

[4] S. A. Goss, R. L. Johnston, and F. Dunn. Compilation of empirical ultrasonic properties of mammalian tissues II. *J. Acoust. Soc. Am.*, 68:93–108, 1980.

[5] M. J. Haney and W. D. O'Brien. Temperature dependency of ultrasonic propagation properties in biological materials. In J. F. Greenleaf, editor, *Tissue Characterization with Ultrasound*. CRC Press, Boca Raton, Fla., 1986.

[6] J. A. Jensen. *Estimation of Blood Velocities Using Ultrasound: A Signal Processing Approach*. Cambridge University Press, New York, 1996.

[7] R. F. Wagner, S. W. Smith, J. M. Sandrick, and H. Lopez. Statistics of speckle in ultrasound B-scans. *IEEE Trans. Son. Ultrason.*, 30:156–163, 1983.

[8] L. E. Kinsler, A. R. Frey, A. B. Coppens, and J. V. Sanders. *Fundamentals of Acoustics*. John Wiley & Sons, New York, third edition, 1982.

[9] G. E. Tupholme. Generation of acoustic pulses by baffled plane pistons. *Mathematika*, 16:209–224, 1969.

[10] P. R. Stepanishen. The time-dependent force and radiation impedance on a piston in a rigid infinite planar baffle. *J. Acoust. Soc. Am.*, 49:841–849, 1971.

[11] P. R. Stepanishen. Transient radiation from pistons in an infinite planar baffle. *J. Acoust. Soc. Am.*, 49:1629–1638, 1971.

[12] A. D. Pierce. *Acoustics, An Introduction to Physical Principles and Applications*. Acoustical Society of America, New York, 1989.

[13] P. M. Morse and K. U. Ingard. *Theoretical Acoustics*. McGraw-Hill, New York, 1968.

[14] P. R. Stepanishen. Pulsed transmit/receive response of ultrasonic piezoelectric transducers. *J. Acoust. Soc. Am.*, 69:1815–1827, 1981.

[15] J. A. Jensen. A model for the propagation and scattering of ultrasound in tissue. *J. Acoust. Soc. Am.*, 89:182–191, 1991a.