Abersim 2.x
Reference manual with tutorials

Halvard Kaupang

2008

Contents

Chapter 1

Intoduction to Abersim

Abersim is a computer program designed to simulate ultrasonic wave propagation in nonlinear, heterogeneous and absorbing materials (*i.e.,* soft tissue). The program has been developed at the Department of Circulation and Medical Imaging (ISB) at the Norwegian University of Science and Technology (NTNU) starting in 1999 and up to today. The main contributors to this program, from now on referred to as the Abersim Team, has been Bjørn Angelsen, Martijn Frijlink, Tonni F. Johansen, Halvard Kaupang, Rune Hansen, Svein-Erik Måsøy, Peter Näsholm, Gunnar Taraldsen, Thor-Andreas Tangen and Trond Varslot.

1.1 Motivation and History

Over the last decades, simulations have been widely used to study different phenomenas related to acoustical wave propagation. First, linear behavior was studied, and then, as the comuters became more and more powerful, nonlinear beahavior started to become an issue. Later, the effects of heterogeneous media became interesting.

Abersim was constructed as a tool to simulate forward wave propagation to study effects of aberration. Tonni Franke Johansen started the development in 1999, but Svein-Erik Måsøy, Trond Varslot and Gunnar Taraldsen soon started to work more on the different parts of the code. Svein-Erik Måsøy, especially, and Trond Varslot worked with the heterogeneous part related to aberration, and Trond Varslot and Gunnar Taraldsen worked on the nonlinear parts. Version 1 of Abersim was never released as anything more than a collection of files. Most of the code was written for 2D applications, but was extended to also work in 3D around 2004. The last version of the 1.X series was version 1.5.3 which was released in 2004. The last years, Halvard Kaupang has worked out the Abersim 2 version partially based on the old code. The program is also ported to C and a link through Matlab's MEX interface is established. The rest of the Abersim Team has contributed in form of specifc functionalities and tuning for applications. Abersim 2.0 is released under the GNU General Public License, version 3 [5]. This license is included in Ch. 5.

1.2 Short description

Abersim is a program for solving directional wave propagation in retarded time. Physically, the wave equation has two solutions propagating in opposite directions. Abersim

will choose the solution only propagating into the body, *i.e.,* the positive z-direction. The introduction of retarded time couples the depth coordinate z to the time coordinate t, and one may say that Abersim simulates the solution propagating in positive *time*. This makes it possible to simulate wave fields propagating either away from or towards the transducer.

Choosing only one of the two solutions, the governing equation is a directional wave equation. The medium in which the wave propagates is mostly soft tissues. Soft tissue is shown to be attenuating and have a nonlinear elasticity [1,2]. Soft tissue is also heterogeneous, causing scattering, reflection and refraction effects. Since the wave equation used in Abersim is directional, Abersim has no way to resolve both the *forward* propagating and back scattered wave in one simulation. Simple back scatter studies using one or a smaller group of point scatterers may be performed in Abersim simulation by first simulating the forward propagating field, and then simulate the scattered field back to the transducer.

The effects of nonlinearity and attenuation is modeled as terms of the wave equation. Numerically, Abersim solves the diffraction, nonlinearity and attenuation using an operator splitting approach. This makes it easy to switch nonlinearity and attenuation on and off to simulate wave propagationg in linear or lossless media. The final differential equation is deducted from the Westervelt equation following somewhat the same procedure as for the deduction of the classic KZK equation (a directional, parabolic approximation to the Westervelt equation). Abersim, however, compared to other KZK simulation tools, has the ability to solve the directional Westervelt equations in cartesian 2D and 3D grids, and not the approximative KZK equation. Even though the diffraction for cartesian setups is solved exactly, there are still directional approximations made. To solve the nonlinear part, a plane wave approximation is made, and the nonlinearity is only accounted for for wave travelling parallel to the z-axis. This introduces errors for steered beams and sharply curved wave fronts.

The structure of Abersim is a folder tree consisting of several functions. This is why it may be referred to as a computer *package* as well as a *program*. These functions may be combined into new simulation scripts, or programs, specifically tuned to meet the needs and requirements of the user. This package contains the basic functions, but the user is encouraged to make their own scripts and functions, and to possibly improve the program. The core routines concerning the propagation of wave fields, should not be altered without publishing further verification and validation. The routines in this package is verified in the papers of Varslot and Taraldsen [10] and Varslot and Måsøy [9]. When publishing studies where Abersim has been used, the user is encouraged to refer to one of these two papers.

1.3 This Manual

This manual contains information on the use of Abersim as well as simple tutorials. The tutorials are presented in Chapter 2 and a more thorough reference of the functions and variables of Abersim is found in Chapter 3. Chapter 4 contains information about how to install Abersim.

Chapter 2

Tutorials

This chapter contains a few simple tutorials for Abersim 2. For more thorough information on the different routines and functions please consult Chapter 3.

The available tutorials in this section are the following:

- Getting started with Abersim
- 2D linear homogeneous beam simulation in muscle tissue
- 3D linear homogeneous beam simulation in muscle tissue
- 3D non-linear homogeneous beam simulation with circular symmetry in water
- 3D non-linear beam simulation in heterogeneous muscle tissue

The tutorials are meant to provide some simple setups of different simulations. As you go along, you will find that there are many ways to modify the setups and create your own export routines, filters and such. How to specify a wave field at the transducer surface is an art in itself, but initial fields may be generated using `pulsegenerator` or obtained from other simulation tools such as Field II or measurements. Abersim and the function `beamsim` handles transducers of arbitrary geometry and with arbitrary pulse shapes. As a user, you are encouraged to modify and create what ever Abersim does that does *not* concern the propagation itself.

2.1 Tutorial 1: Getting started with Abersim

The goal of this tutorial is to get used to the control struct `Propcontrol` *and how to set up wave fields, export beamprofiles and simple visualization of these.*

Make sure the Abersim package is installed on your computer, and start up Matlab from the Abersim directory. For help on the installation see Chapter 4. Then run `startup_abersim`. This will add the Abersim folders to the Matlab path. Then run this tutorial using the testscript `tutorial_basic.m` (located in the test folder). You don't have to be in the test folder to run the tutorial, but the test script will change the active folder to the test folder for storage of files etc.

2.1.1 Create Propcontrol

Abersim simulations depend on the control struct `Propcontrol`. For more information on the different variables in `Propcontrol`, please consult Section 3.1. This struct is initialized by calling `initpropcontrol`, and the output should look something like:

```
Start of part 1: Creating Propcontrol


The Propcontrol is created using initpropcontrol
Propcontrol = initpropcontrol


Propcontrol =

         simname: '20080817_101841_beamsim'
           ndims: 2
              nx: 512
              ny: 1
              nt: 1024
        PMLwidth: 0
       diffrflag: 1
     nonlinflag: 1
       lossflag: 1
         abflag: 0
        annflag: 0
    equidistflag: 0
     historyflag: 1
        stepsize: 0.0013
       shockstep: 0.5000
       endpoint: 0.1000
     currentpos: 0
       storepos: 0.0600
       material: [1x1 struct]
              d: 0.0200
         offset: [2x1 double]
     numscreens: 8
          abamp: [16x1 double]
       ablength: [16x2 double]
         abfile: 'randseq3'
             Fs: 40000000
             dx: 1.1667e-04
             dy: 1.2903e-04
             dz: 8.2225e-05
             dt: 2.5000e-08
             fc: 1500000
      bandwidth: 750000
             Np: 2.5000
      amplitude: 0.5000
       harmonic: 2
         filter: [2x1 double]
             Dx: 0.0224
             Dy: 0.0120
             Fx: 0.0600
             Fy: 0.0600
       cchannel: [2x1 double]
            Nex: 64
            Ney: 1
         esizex: 3.5000e-04
         esizey: 0.0120
```

To get used to `Propcontrol`, type `help initpropcontrol` and run the function `initpropcontrol` with different input arguments. The fields of `Propcontrol` may be altered by the user directly, but this should be done with much care, and make sure that you fully understand the effects of your alterations before running larger studies. All varaibles regarding the computational domain, the transducer and wave fields, and the material and its properties may be altered in `Propcontrol`.

2.1.2 Generation and visualization of pulses

To generate a simple wave field call the function `pulsegenerator` with `Propcontrol` as the argument. This will return the wave field from a transducer specified in `Propcontrol`. The default is a 1.5 MHz pulse transmitted from a linear array with 64 elements and total aperture of 22.4 mm. Then, visualize the wave field by calling `plot_pulse` with the wave field and `Propcontrol` as arguments. Use the help function or consult Section 3.2.4 for more information on visualization. The returned wave field may be seen in Figure 2.1. The output of the tutorial should look something like:

```
Start of part 2: Creating a wave field at the transducer


The pulse is created using pulsegenerator
u = pulsegenerator(Propcontrol,'transducer');


Press any key to plot pulse
plot_pulse(u, Propcontrol);
```

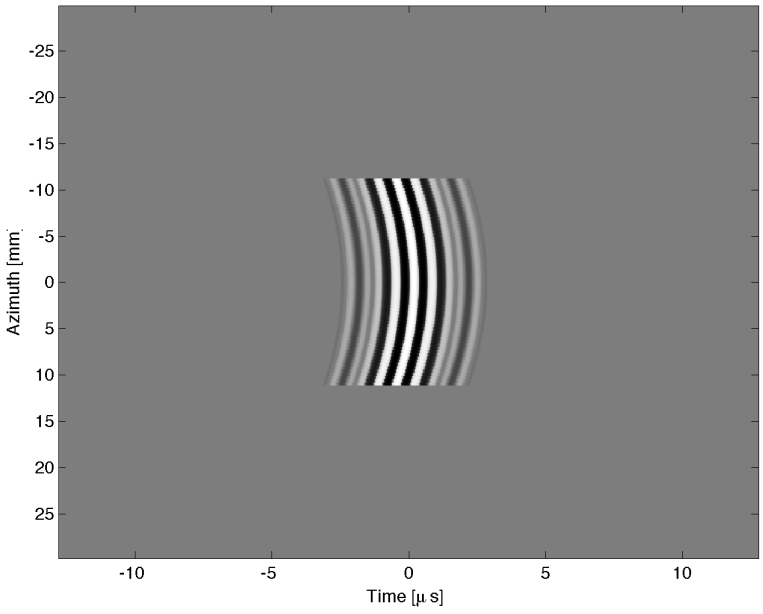


Figure 2.1: The generated wave field at the transducer surface.

2.1.3 Export and visualization of beam profiles

To generate a beamprofile the function `export_beamprofile` is called with the wave field and `Propcontrol` as arguments. The function `export_beamprofile` returns a 4D-variable containing the beamprofiles. This variable is visualized using `plot_beamprofile` in Figure 2.2. The output of the tutorial should look something like:

```
Start of part 3: Creating a beam profile


The beam profile is exported by calling export_beamprofile
prof = export_beamprofile(u, Propcontrol);


Press any key to plot beam profile
plot_beamprofile(u, Propcontrol);
```

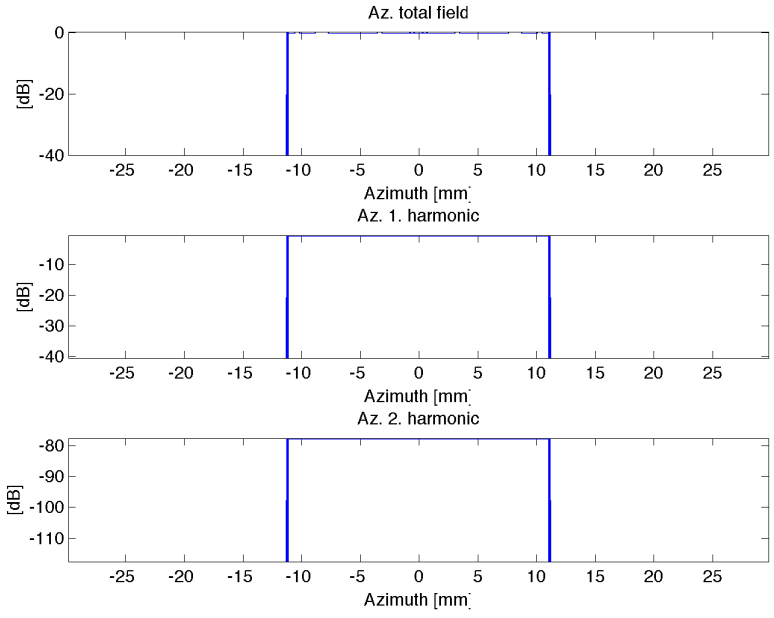


Figure 2.2: The beam profile of the generated wave field at the transducer surface.

2.2 Tutorial 2: 2D linear homogeneous beam simulation

The goal of this tutorial is to set up a simple simulation and run it using the simulation script `beamsim`*, and visualizing the output of the simulation.*

This tutorial will set up a 2D beam simulation using a *1D* linear array probe with infinite extenision in the elevation direction. The default medium is muscle at 37 degrees Celcius, which now is assumed to be linear and homogeneous. The diffraction is solved in the frequency domain using the Angular Spectrum Method (see Sec. 3.3.2). Attenuation of the medium is for linear simulations using Angular Spectrum baked into the wave number operator (see Sec. 3.3.3). The tutorial follows the test script `tutorial_2Dlinhom.m`.

2.2.1 Preparing a simulation

The `Propcontrol` is an important part of any Abersim simulation. For a simulation it is often necessay to specify input arguments for `initpropcontrol`. For this simulation we will use the input arguments presented in Table 2.1. For a more thorough description of the possible input variables and their effect, please consult Section 3.1.

Table 2.1: Input arguments for `initpropcontrol` for a 2D simulation in a linear, homogeneous medium.

Input variable	Value	Desciption
name	'test_2Dlinhom'	Name of simulation for storage etc.
ndim	2	Number of dimensions
flags	[1,0,1,0,0,0,1]	
diffrflag	1	Type of diffraction (Ang. Spect.)
nonlinflag	0	Nonlinearity (off)
lossflag	1	Attenuation (on)
abflag	0	Heterogeneous medium (off)
annflag	0	Annular transducer (off)
equidistflag	0	Equidistant steps (on)
historyflag	1	History (save profiles)
harm	1	Fundamental imaging

The default imaging setup by `initpropcontrol` is fundamental imaging at 3 MHz, focused at 60 mm and an azimuth aperture 22.4 mm. The endpoint of the simulation is at 100 mm. A wave field similar to the one in Figure 2.1 is generated, and the simulation is run using the simulation function `beamsim` (see Sec. 3.2.2).

2.2.2 Running the simulation

The function `beamsim` is the main simulation function. This will setup a simulation. The script and calculate the number of steps needed, export beam profiles and on-axis pulses, store wave fields, applies eventual aberration and/or aberration correction. The beam profiles, the on-axis pulse for each step and the wave field at the endpoint is output variables from `beamsim`. The stored wave fields are stored in files with a filename specified by the `Propcontrol` field *simname* and a further extension describing the position at which the wave field was stored. The output produced from the first two parts of the tutorial is:

```
Start of part 1: Preparing the simulation


The Propcontrol is created using initpropcontrol with a set of arguments
The arguments that are important are such as the flags, center frequency,
the number of dimensions and a name for the simulation. This test script
sets up a 2D simulation in a linear medium. The imaging mode is thus
fundamental, or first-harmonic, imaging. The imaging frequency, aperture,
and focal depth are as default by initpropcontrol.

Press any key to continue

Input variables for Propcontrol:
name      = 'test_2Dlinhom';
ndim      = 2;
flags     = [1,0,1,0,0,0,1];
harm      = 1;

Press any key to continue

Generate Propcontrol
Propcontrol = initpropcontrol(name, ndim, flags, harm);

Generate a wave field at the transducer
u = pulsegenerator(Propcontrol,'transducer');


Press any key to continue to the next part


Start of part 2: Running the simulation


The simulation itself is run using the function beamsim. This will
propagate the initial wave field up to a certain distance and export
the beam profile for each step. These profiles will be returned by
beamsim. Propcontrol is also returned in a updated manner (that is: the
current position is not longer 0 (at the transducer), but has the value
of the endpoint.

Press any key to continue

[u_out, Propcontrol, rmspro, maxpro, axplse] = beamsim(Propcontrol, u);
Starting linear simulation of size 512 x 512
Simulation to final endpoint is 10.00 percent complete (st.   8/ 80), ETA 10:18
Simulation to final endpoint is 20.00 percent complete (st.  16/ 80), ETA 10:18
Simulation to final endpoint is 30.00 percent complete (st.  24/ 80), ETA 10:18
Simulation to final endpoint is 40.00 percent complete (st.  32/ 80), ETA 10:18
Simulation to final endpoint is 50.00 percent complete (st.  40/ 80), ETA 10:18
Simulation to final endpoint is 60.00 percent complete (st.  48/ 80), ETA 10:18
Simulation to final endpoint is 70.00 percent complete (st.  56/ 80), ETA 10:19
Simulation to final endpoint is 80.00 percent complete (st.  64/ 80), ETA 10:19
Simulation to final endpoint is 90.00 percent complete (st.  72/ 80), ETA 10:19
Simulation finished in    0.38 min using an average of    0.28 sec pr step
```

2.2.3 Visualizing beam profiles

Now we have a set of *nsteps* 1D beam profiles of the kind we plotted in Tutorial 1. The beam profiles plots show the profile for the total field and for each of the harmonic components. The harmonic components are obtained by bandpass filtering the total field

in the frequency domain. The center frequency of the filter is nf_c for the n'th harmonic frequency, and the relative bandwidth of the filter is specified in the `Propcontrol` field *filter*. A set of such beam profiles is in fact a 2D beam profile. This may be visualized as an image as presented in Figure 2.3. The focal point beam profile is shown as an own 1D bemprofile in Figure 2.4, and the sudden fall of the profile around 15-20 mm comes from the windowing of the solution to avoid wrap-around effects (see Sec. 3.2.2). The on-axis pulse may be exported and plotted using the `plot_pulse` function with the input argument *axflag* set to one. The on-axis pulse with its frequency spectrum is shown in Figure 2.5. The output from the third part of the tutorial should look something like:

```
Start of part 3: Visualization of results


In this part we will plot the beam profiles obtained by beamsim.
Since this is a 2D simulation, only the (x,z)-plane profile is
available. We will also plot the focal profile and the axial
pulse for each step.

Press any key to plot beam profile
plot_beamprofile(rmspro,Propcontrol);

Press any key to plot focal point beam profile
Find index of focal profile
idx = round(Propcontrol.Fx / Propcontrol.stepsize) + 1;
plot_beamprofile(rmspro(:,:,idx,:),Propcontrol);

Press any key to plot on-axis pulses
plot_pulse(axplse,Propcontrol,1);
```

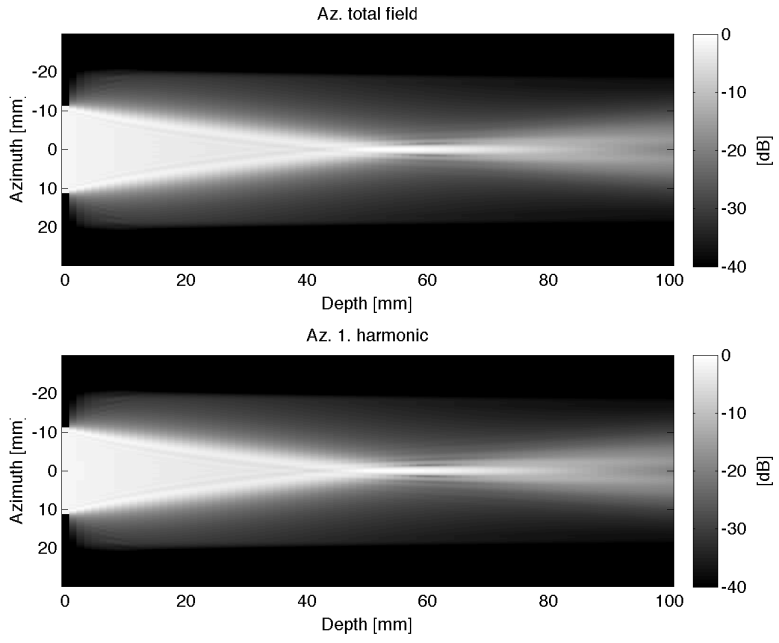


Figure 2.3: The temporal RMS-value of the radiated ultrasound beam.

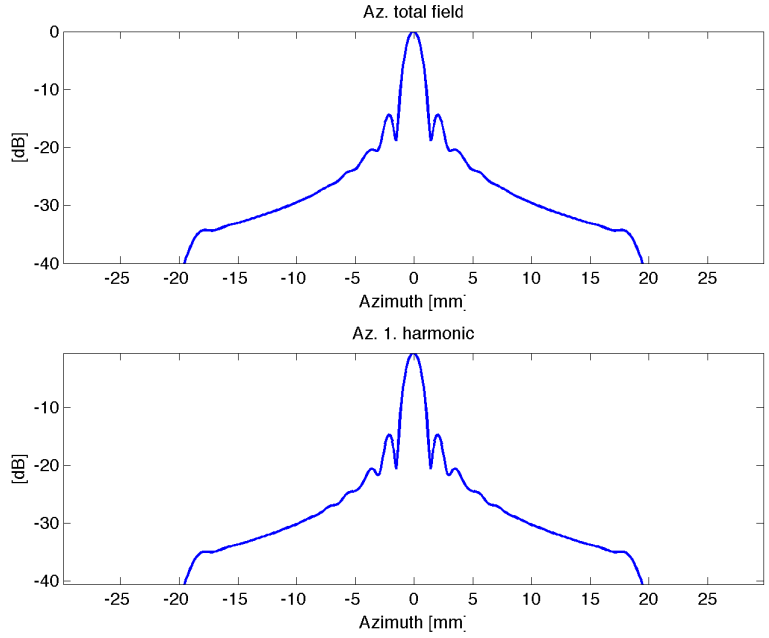


Figure 2.4: The temporal RMS-value of the radiated ultrasound beam at the focal point as a function of x.

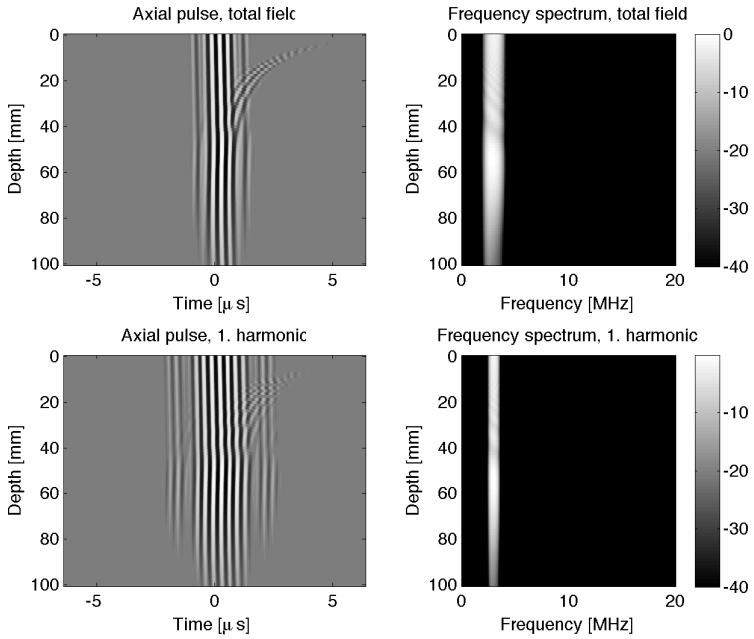


Figure 2.5: The on-axis pulse for each depth (right) and its frequency spectrum (right).

2.3 Tutorial 3: 3D linear homogeneous beam simulation

The goal of this tutorial is to set up and run a simulation in 3D using a rectangular transducer with apodization, and to export and visualize 3D wave fields and beam profiles.

This tutorial will set up a 3D beam simulation using a 1D linear array probe with lens focusing in the elevation direction. The medium is still muscle at 37 degrees, which is assumed to be linear and homogeneous. The diffraction is solved in the frequency domain using the Angular Spectrum Method (see Sec. 3.3.2). Attenuation of the medium is for linear simulations using Angular Spectrum baked into the wave number operator (see Sec. 3.3.3). The tutorial follows the test script `tutorial_3Dlinhom.m`. 3D simulations are quite memory consuming and should be run on computer with a fairly large memory. The memory consumption for this simulation is about 1 GB.

2.3.1 Preparing the simulation

The `Propcontrol` is prepared as described in Tutorial 2 (Sec. 2.2 and Tab. 2.1), but now the number of dimensions is three. The standard imaging setup of fundamental imaging at 3 MHz, focused at 60 mm in, now both, azimuth and elevation direction, and the aperture sizes are 22.4 mm and 12 mm in azimuth and elevation respectively. The endpoint of the simulation is at 100 mm. In azimuth a rectangular apodization is used, and in elevation a cosine apodization. A pulse similar to the one in Figure 2.1, but now in 3D, is generated, and the simulation is run using `beamsim`. A 3D wave field may be plotted and visualized in the same way as 2D pulses, and a 3D wave field is seen in Fig. 2.6.

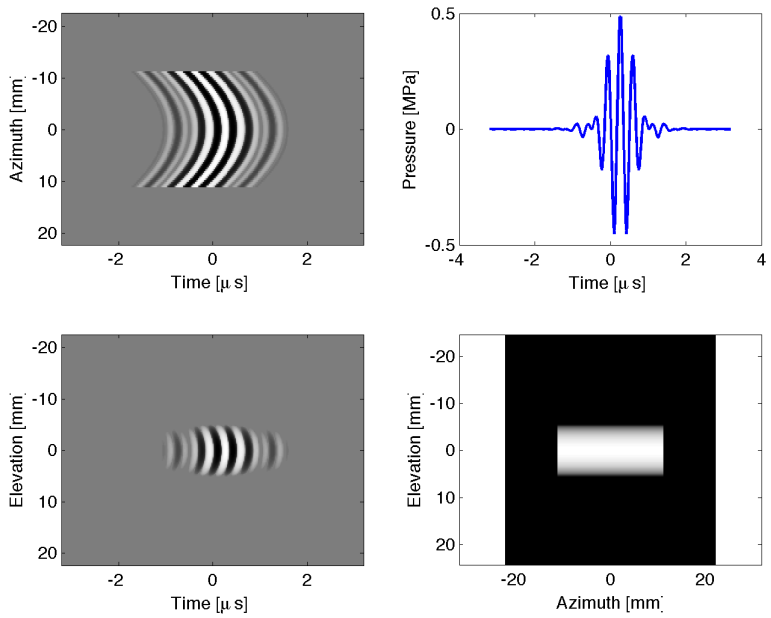


Figure 2.6: The generated 3D wave field at the transducer surface. The elevation direction uses a lens focusing and a cosine apodization. The top left panel is the pulse in the (x,t)-frame, the bottom the in the (y,t)-frame. The right, top panel is the center channel pulse and the bottom if the RMS intensity of the pulse in the cross sectional (x,y)-plane

2.3.2 Running the simulation

3D simulations are more computationally demanding than 2D. The sampling is therefore slightly coarser, but still sufficient. The padding with free space on either side of the pulse is also reduced for the 3D grid compared to the 2D. The output produced from the first two parts of the tutorial is:

```
Start of part 1: Preparing the simulation


The Propcontrol is created using initpropcontrol with a set of arguments
similar to the 2D tutorial simulation.

Press any key to continue

Input variables for Propcontrol:
name      = 'test_3Dlinhom';
ndim      = 3;
flags     = [1,0,1,0,0,0,1];
harm      = 1;

Press any key to continue

Generate Propcontrol
Propcontrol = initpropcontrol(name, ndim, flags, harm);

Generate a wave field at the transducer with a cosine apodization in elevation
u = pulsegenerator(Propcontrol,'transducer',[0 1]);

Press any key to plot pulse
3D pulses may also be plotted
plot_pulse(u, Propcontrol);


Press any key to continue to the next part


Start of part 2: Running the simulation


The simulation itself is run using the function beamsim as for the
linear 2D simulation.

Press any key to continue

[u_out, Propcontrol, rmspro, maxpro, axplse] = beamsim(Propcontrol, u);
Starting linear simulation of size 256 x 192 x 256
Simulation to final endpoint is 10.00 percent complete (st.   8/ 80), ETA 10:41
Simulation to final endpoint is 20.00 percent complete (st.  16/ 80), ETA 10:41
Simulation to final endpoint is 30.00 percent complete (st.  24/ 80), ETA 10:41
Simulation to final endpoint is 40.00 percent complete (st.  32/ 80), ETA 10:41
Simulation to final endpoint is 50.00 percent complete (st.  40/ 80), ETA 10:41
Simulation to final endpoint is 60.00 percent complete (st.  48/ 80), ETA 10:42
Simulation to final endpoint is 70.00 percent complete (st.  56/ 80), ETA 10:41
Simulation to final endpoint is 80.00 percent complete (st.  64/ 80), ETA 10:41
Simulation to final endpoint is 90.00 percent complete (st.  72/ 80), ETA 10:42
Simulation finished in   22.97 min using an average of   17.23 sec pr step
```

2.3.3 Visualizing beam profiles

We have a set of *nsteps* 2D beam profiles. This may be visualized using `plot_beamprofile` as presented in Figure 2.7. Cross sections of the beam, or other more specific cross sectional profiles, may also be plotted using `plot_beamprofile`. The focal profile in the (x,y)-plane, or azimuth and elevation direction, is presented in Figure 2.8. The output from the third part of the tutorial should look something like:

```
Start of part 3: Visualization of results


In this part we will plot the beam profiles obtained by beamsim.
Since this is a 3D simulation, both the (x,z)-plane profile and
the (y,z)-plane profiles are available. We will also plot the focal
profile in the (x,y)-plane.

Press any key to plot beam profile
plot_beamprofile(rmspro,Propcontrol);

Press any key to plot focal point beam profile
Find index of focal profile
idx = Propcontrol.Fx / Propcontrol.stepsize + 1;
plot_beamprofile(rmspro(:,:,idx,:),Propcontrol);
```

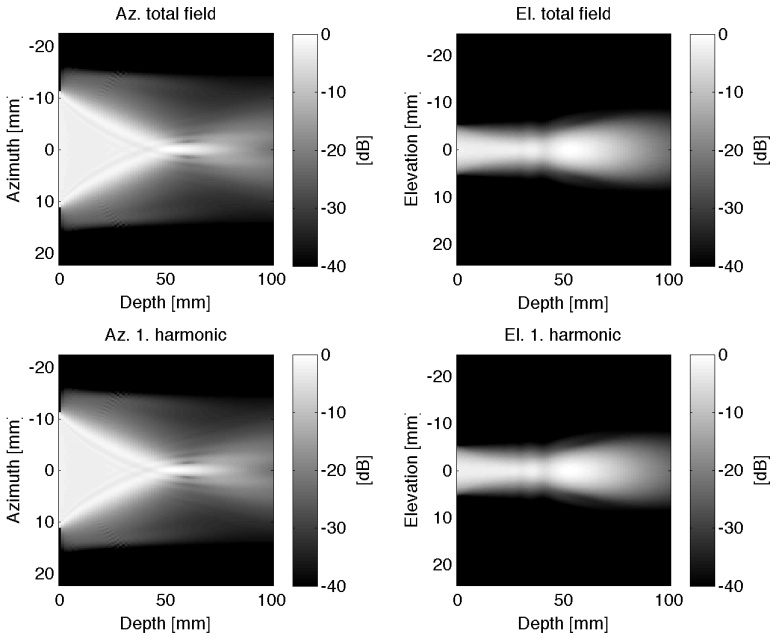


Figure 2.7: The temporal RMS-value of the radiated ultrasound beam in depth over the azimuth and elevation direction.

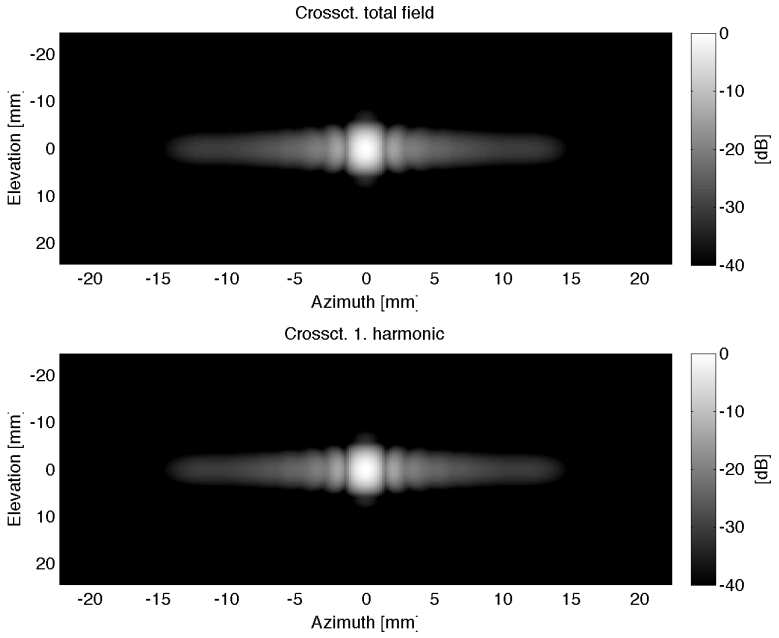


Figure 2.8: The temporal RMS-value of the radiated ultrasound beam at the focal point over the (x,y)-plane.

2.4 Tutorial 4: 3D non-linear homogeneous beam simulation with rotational symmetry

The goal of this tutorial is to set up and run a non-linear simulation using a circular-symmetric transducer in another material than the default muscle, and to visualize wave fields and beam profiles when higher harmonic frequencies are present.

This tutorial will set up a 3D beam simulation with circular symmetry using an circular piston transducer. The medium is now water, which is non-linear and homogeneous, and the latter enables the use of a symmetric simulation. Although the transducer itself is circular-symmetric regardless of the medium, a fully heterogeneous medium is not symmetric in any way. This is why circular-symmetric simulation requires homogeneous media. Piston transducers may, however, be used for heterogeneous medias as well, but then the computation is performed in full 3D. Using circular symmtry, the diffraction is no longer solved using the Angular Spectrum Method, but with a finite difference discretization of the parabolic approximation of the wave equation, the classic KZK equation. The tutorial follows the script in `tutorial_3Dnonsym`.

2.4.1 Preparing the simulation

The input variables to `initpropcontrol` is much the same as presented in Table 2.1, but we add quite a few input arguments. For now, the flags *diffrflag* is set to 4 (or 5) and *nonlinflag* is 1. The difference between *diffrflag* 4 and 5 is described in Section 3.1. Note that *ndim* is kept as 2. This denotes the dimensions of the computational domain, not the physical dimensions. Because of the symmetry conditions, the wave field is only treated in the half-plane. Further we intend to do fifth harmonic imaging at 7.5 MHz, requiring a transmit frequency of 1.5 MHz. One thing to keep in mind is that the *input* argument *fi* in `Propcontrol` denotes the *imaging* frequency, whereas the field *fc* denotes the center frequency of the *transmitted* pulse. This should on input be specified then as *fi* of 7.5 MHz and *harm* as 5 denoting the fifth harmonic. The number of periods in the transmitted pulse is 8 to ensure a high degree of generation. The transmit pressure is set to 1 MPa, and no apodization is used. The number of elements in azimuth (and elevation) is one (single element piston transducer), and the aperture size is 22.4 mm. The transducer is focused at 60 mm, and the endpoint of the simulation is at 100 mm. Using circular symmetry, the computation is performed in the half domain, and a wave field in the half domain is plotted below in Figure 2.9.

2.4.2 Running the simulation

Non-linear simulations are more computationally demanding than linear due to the need for several substeps through the introduction of operator splitting. The output produced from the first two parts of the tutorial is:

```
Start of part 1: Preparing the simulation


The Propcontrol is created using initpropcontrol with a set of arguments
similar to the 2D tutorial simulation. A 3D simulation with rotational
symmetry allows for a 2D calculation domain.

Press any key to continue
```

```
Input variables for Propcontrol:
name      = 'test_3Dnonsym';
ndim      = 2;
flags     = [4,0,1,0,1,0,1];
harm      = 5;
fi        = 7.5;
bandwidth = 0.5;
Np        = 8;
amp       = 1;
material  = WATER;
temp      = 20;
endpoint  = 0.1;
Faz       = 0.06;
Nelaz     = 1;
dimelaz   = 22.4;

Press any key to continue

Generate Propcontrol and pulse

Press any key to plot pulse
3D symmetric pulses is shown in half domain
plot_pulse(u, Propcontrol);


Press any key to continue to the next part


Start of part 2: Running the simulation


The simulation itself is run using the function beamsim as for the
linear 2D simulation.

Press any key to continue

[u_out, Propcontrol, rmspro, maxpro, axplse] = beamsim(Propcontrol, u);
Starting non-linear simulation of size 384 x 3072
Simulation to final endpoint is 10.00 percent complete (st.  20/200), ETA 12:21
Simulation to final endpoint is 20.00 percent complete (st.  40/200), ETA 12:22
Simulation to final endpoint is 30.00 percent complete (st.  60/200), ETA 12:21
Simulation to final endpoint is 40.00 percent complete (st.  80/200), ETA 12:21
Simulation to final endpoint is 50.00 percent complete (st. 100/200), ETA 12:21
Simulation to final endpoint is 60.00 percent complete (st. 120/200), ETA 12:21
Simulation to final endpoint is 70.00 percent complete (st. 140/200), ETA 12:21
Simulation to final endpoint is 80.00 percent complete (st. 160/200), ETA 12:21
Simulation to final endpoint is 90.00 percent complete (st. 180/200), ETA 12:21
Simulation finished in   99.76 min using an average of   29.93 sec pr step
```

2.4.3 Visualizing beam profiles

Again, we have a set of *nsteps* 1D beam profiles, each over r for each depth z. The beam profile for the total field and all the harmonic frequencies are visualized using `plot_beamprofile` and presented in Figure 2.10. The 1D focal beam profile for all the harmonics are presented in Figure 2.11. The on-axis pulse for a nonlinear simulation may be seen in Figure 2.12. The output from the third part of the tutorial should look something like:

```
Start of part 3: Visualization of results
```

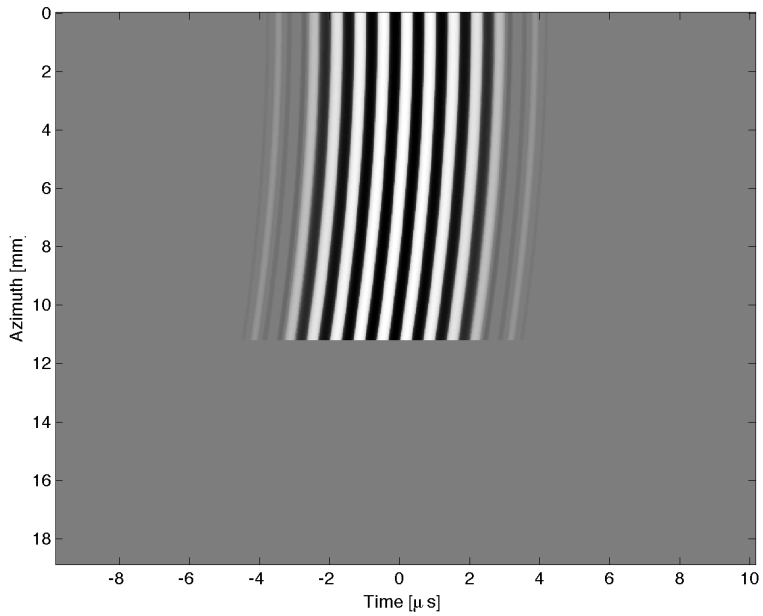


Figure 2.9: The generated half domain wave field at the transducer surface.

```
In this part we will plot the beam profiles obtained by beamsim.
Since this is a 3D symmetric simulation, only the (r,z)-plane
profile is available. We will also plot the focal profile and
the on-axis pulse for each step.

Press any key to plot beam profile
plot_beamprofile(rmspro,Propcontrol);

Press any key to plot focal point beam profile
Find index of focal profile
idx = Propcontrol.Fx / Propcontrol.stepsize + 1;
plot_beamprofile(rmspro(:,:,idx,:),Propcontrol);

Press any key to plot on-axis pulses
plot_pulse(axplse,Propcontrol,1);
```

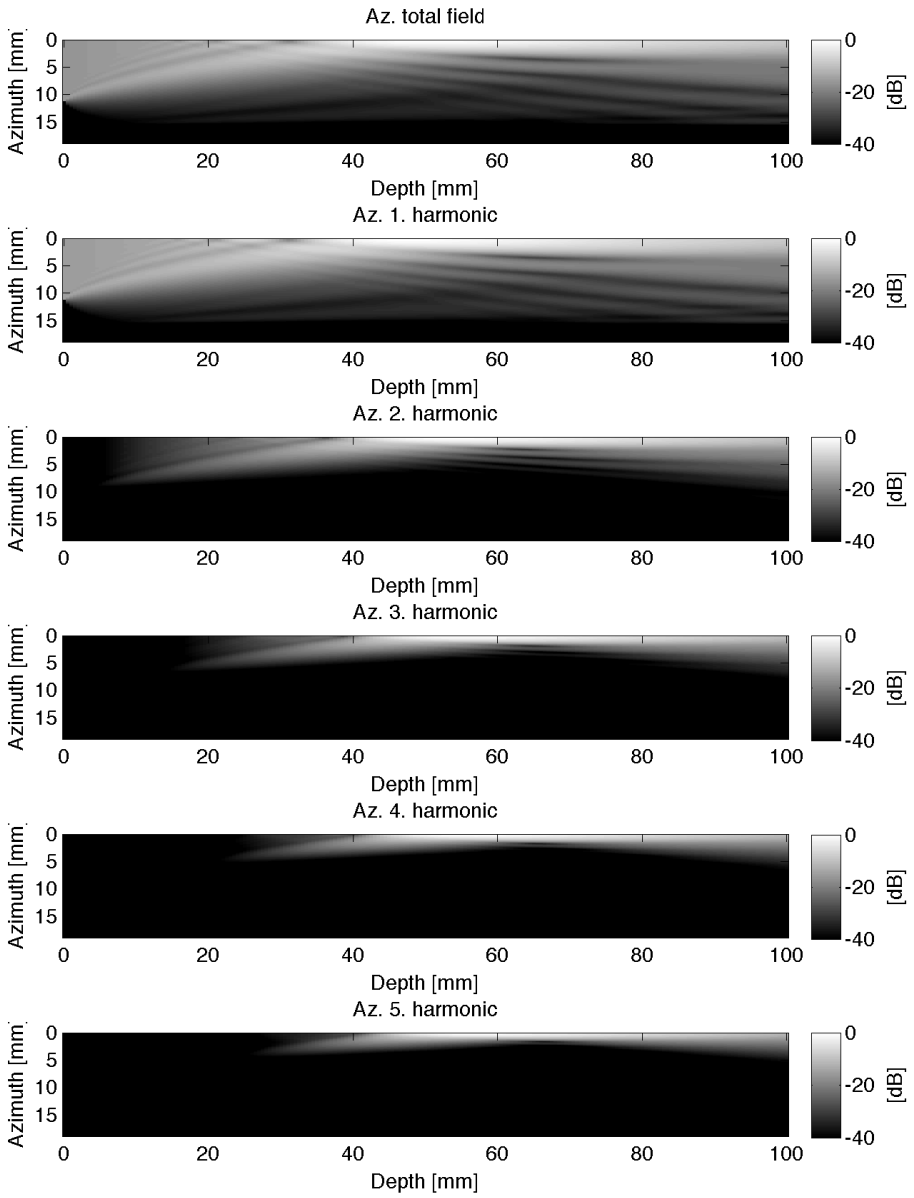


Figure 2.10: The temporal RMS-value of the radiated ultrasound beam in depth over the radial direction.

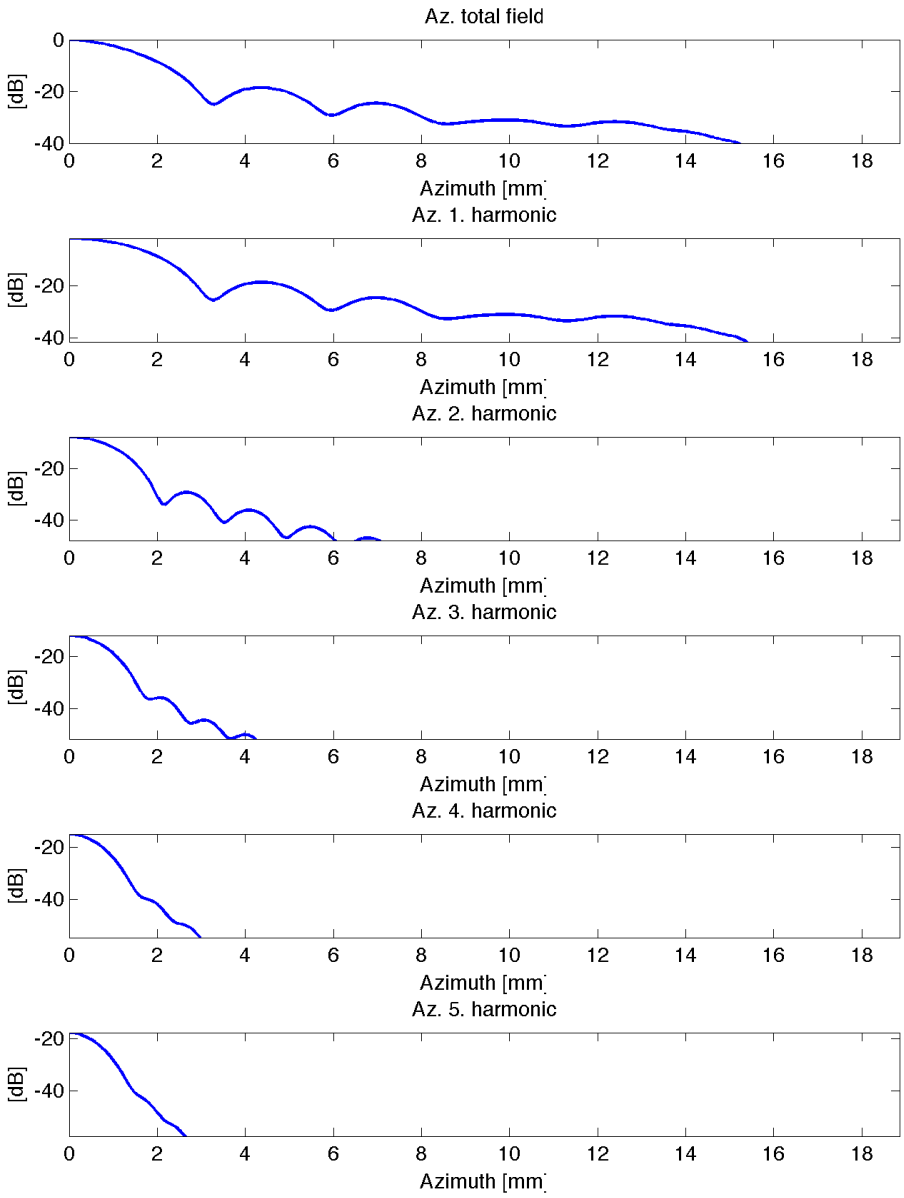


Figure 2.11: The temporal RMS-value of the radiated ultrasound beam at the focal point as a function of r.

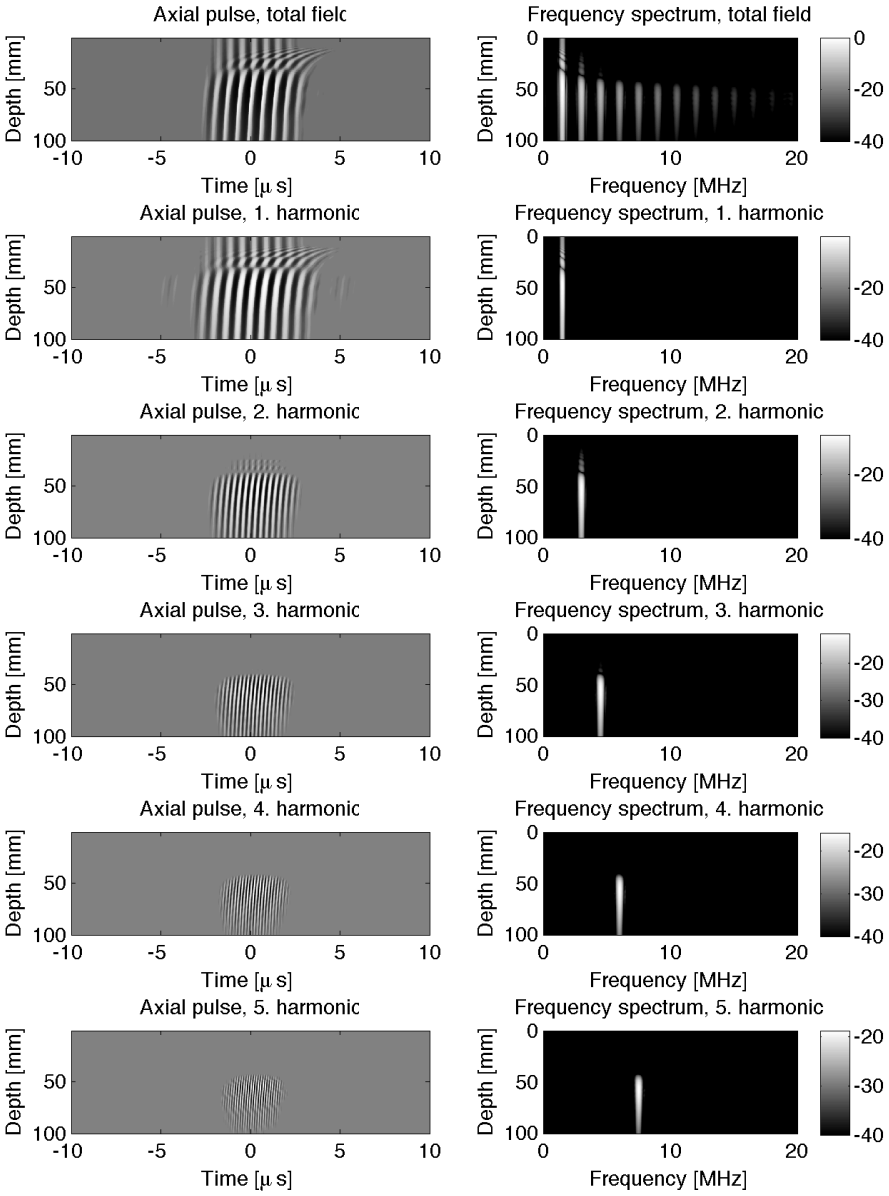


Figure 2.12: The on-axis pulse for each depth (right) and its frequency spectrum (right).

2.5 Tutorial 5: 3D non-linear heterogeneous beam simulation from an arbitrary transducer

The goal of this tutorial is to set up and run a non-linear 3D simulation using a rectangular transducer in a heterogeneous material, and to store away the wave field at some predefined positions.

This tutorial will set up a 3D beam simulation using a *1.75D* array probe. The medium is non-linear and heterogeneous muscle tissue at 37 degrees. The aberrating body wall is contructed as a delay screen body wall generated from a set of random numbers, and is set to be 20 mm thick by default. The diffraction is solved in the frequency domain using the Angular Spectrum Method, and to ensure a further speed-up of the calculations, the *equidist* flag is specified. Attenuation is for a non-linear medium *not* baked into the wave number operator, but is solved separately as a part of the operator splitting scheme. The tutorial follows the test script `tutorial_3Dnonhet.m`.

2.5.1 Preparing the simulation

The `Propcontrol` is prepared as described in Tutorial 3 (Sec. 2.3), but the *nonlinflag* and *abflag* are set to one. The default imaging frequency is still 3 MHz, but now a *second-harmonic* imaging scheme is prepared, *i.e.,* the parameter *harmonic* is set to 2 and the transmit frequency is 1.75 MHz. The focal depths are in azimuth and elevation 70 mm and 50 mm respectively. The transducer uses delay focusing for both azimuth and elevation, and the transducer consist of 64 elements in azimuth and 5 elements in elevation. All elements are equal in size, and the total apertures are 22.4 mm and 12 mm for azimuth and elevation respectively. A rectangular apodization if used for both azimuth and elevation.

The transmitted wave field may be seen in Figure 2.13, and this is similar, but not equal to the one used in Tutorial 3 in Figure 2.6.

2.5.2 Running the simulation

3D non-linear simulations are the most computationally demanding. The sampling and size of the computational domain is similar to the one used in Tutorial 3. The output produced from the first two parts of the tutorial is:

```
Start of part 1: Preparing the simulation


The Propcontrol is created using initpropcontrol with a set of arguments
similar to the 3D tutorial simulation.

Press any key to continue

Input variables for Propcontrol:
name      = 'test_3Dnonhet';
ndim      = 3;
flags     = [1,1,1,1,0,1,2];
harm      = 2;
fi        = 3.0;
bandwidth = 0.5;
Np        = 2.5;
```

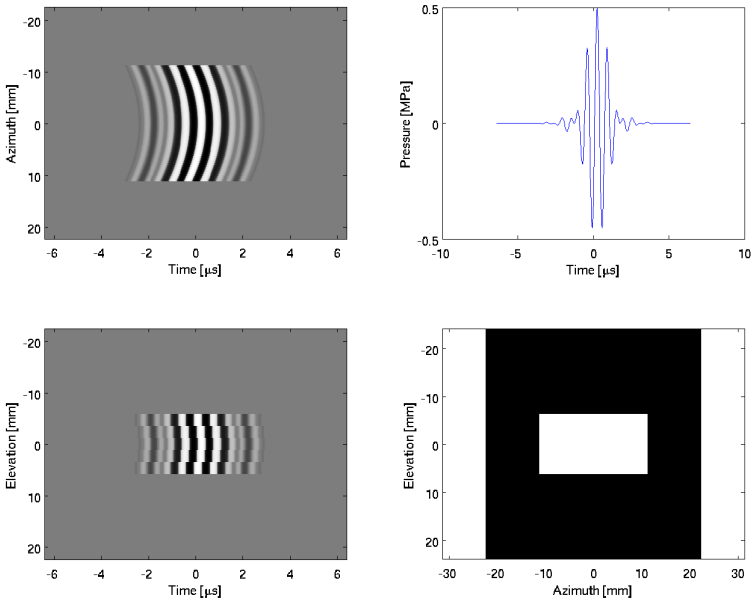


Figure 2.13: The generated 3D wave field at the transducer surface. The elevation direction uses a delay focusing and no apodization.

```
amp        = 0.5;
material   = MUSCLE;
temp       = 37;
endpoint   = 0.1;
Faz        = 0.07;
Nelaz      = 64;
dimelaz    = 3.5e-4;
Fel        = 0.05;
Nelel      = 5;
dimelel    = 2.5e-3;

Press any key to continue

Generate Propcontrol and pulse

Press any key to plot pulse
plot_pulse(u, Propcontrol);



Start of part 2: Running the simulation


The simulation itself is run using the function beamsim as for the
linear 2D simulation.

Press any key to continue

[u_out, Propcontrol, rmspro, maxpro, axplse] = beamsim(Propcontrol, u);
Starting non-linear simulation of size 256 x 192 x 512
Entering body wall
Simulation to end of bodywall is  6.25 percent complete (st.   1/ 16), ETA 18:27
Simulation to end of bodywall is 12.50 percent complete (st.   2/ 16), ETA 18:26
Simulation to end of bodywall is 18.75 percent complete (st.   3/ 16), ETA 18:24
```

```
Simulation to end of bodywall is 25.00 percent complete (st.   4/ 16), ETA 18:24
Simulation to end of bodywall is 31.25 percent complete (st.   5/ 16), ETA 18:24
Simulation to end of bodywall is 37.50 percent complete (st.   6/ 16), ETA 18:26
Simulation to end of bodywall is 43.75 percent complete (st.   7/ 16), ETA 18:26
Simulation to end of bodywall is 50.00 percent complete (st.   8/ 16), ETA 18:34
Simulation to end of bodywall is 56.25 percent complete (st.   9/ 16), ETA 18:35
Simulation to end of bodywall is 62.50 percent complete (st.  10/ 16), ETA 18:34
Simulation to end of bodywall is 68.75 percent complete (st.  11/ 16), ETA 18:34
Simulation to end of bodywall is 75.00 percent complete (st.  12/ 16), ETA 18:34
Simulation to end of bodywall is 81.25 percent complete (st.  13/ 16), ETA 18:33
Simulation to end of bodywall is 87.50 percent complete (st.  14/ 16), ETA 18:34
Simulation to end of bodywall is 93.75 percent complete (st.  15/ 16), ETA 18:33
save pulse to test_3Dnonhet_02000.mat
Done with body wall
Simulation to final endpoint is  1.56 percent complete (st.   1/ 64), ETA 17:19 (+1)
Simulation to final endpoint is  3.12 percent complete (st.   2/ 64), ETA 17:19 (+1)
Simulation to final endpoint is  4.69 percent complete (st.   3/ 64), ETA 17:20 (+1)
Simulation to final endpoint is  6.25 percent complete (st.   4/ 64), ETA 17:20 (+1)
Simulation to final endpoint is  7.81 percent complete (st.   5/ 64), ETA 17:21 (+1)
Simulation to final endpoint is  9.38 percent complete (st.   6/ 64), ETA 17:21 (+1)
Simulation to final endpoint is 10.94 percent complete (st.   7/ 64), ETA 17:21 (+1)
Simulation to final endpoint is 12.50 percent complete (st.   8/ 64), ETA 17:22 (+1)
Simulation to final endpoint is 14.06 percent complete (st.   9/ 64), ETA 17:22 (+1)
Simulation to final endpoint is 15.62 percent complete (st.  10/ 64), ETA 17:59 (+1)
Simulation to final endpoint is 17.19 percent complete (st.  11/ 64), ETA 18:31 (+1)
Simulation to final endpoint is 18.75 percent complete (st.  12/ 64), ETA 18:40 (+1)
Simulation to final endpoint is 20.31 percent complete (st.  13/ 64), ETA 18:36 (+1)
Simulation to final endpoint is 21.88 percent complete (st.  14/ 64), ETA 18:33 (+1)
Simulation to final endpoint is 23.44 percent complete (st.  15/ 64), ETA 18:28
Simulation to final endpoint is 25.00 percent complete (st.  16/ 64), ETA 18:23
Simulation to final endpoint is 26.56 percent complete (st.  17/ 64), ETA 18:20
Simulation to final endpoint is 28.12 percent complete (st.  18/ 64), ETA 18:16
Simulation to final endpoint is 29.69 percent complete (st.  19/ 64), ETA 18:13
Simulation to final endpoint is 31.25 percent complete (st.  20/ 64), ETA 18:11
Simulation to final endpoint is 32.81 percent complete (st.  21/ 64), ETA 18:08
Simulation to final endpoint is 34.38 percent complete (st.  22/ 64), ETA 18:07
Simulation to final endpoint is 35.94 percent complete (st.  23/ 64), ETA 18:05
save pulse to test_3Dnonhet_05000.mat
Simulation to final endpoint is 37.50 percent complete (st.  24/ 64), ETA 18:03
Simulation to final endpoint is 39.06 percent complete (st.  25/ 64), ETA 18:02
Simulation to final endpoint is 40.62 percent complete (st.  26/ 64), ETA 18:00
Simulation to final endpoint is 42.19 percent complete (st.  27/ 64), ETA 17:59
Simulation to final endpoint is 43.75 percent complete (st.  28/ 64), ETA 17:57
Simulation to final endpoint is 45.31 percent complete (st.  29/ 64), ETA 17:57
Simulation to final endpoint is 46.88 percent complete (st.  30/ 64), ETA 17:55
Simulation to final endpoint is 48.44 percent complete (st.  31/ 64), ETA 17:55
Simulation to final endpoint is 50.00 percent complete (st.  32/ 64), ETA 17:54
Simulation to final endpoint is 51.56 percent complete (st.  33/ 64), ETA 17:52
Simulation to final endpoint is 53.12 percent complete (st.  34/ 64), ETA 17:52
Simulation to final endpoint is 54.69 percent complete (st.  35/ 64), ETA 17:51
Simulation to final endpoint is 56.25 percent complete (st.  36/ 64), ETA 17:51
Simulation to final endpoint is 57.81 percent complete (st.  37/ 64), ETA 17:50
Simulation to final endpoint is 59.38 percent complete (st.  38/ 64), ETA 17:50
Simulation to final endpoint is 60.94 percent complete (st.  39/ 64), ETA 17:49
save pulse to test_3Dnonhet_07000.mat
Simulation to final endpoint is 62.50 percent complete (st.  40/ 64), ETA 17:49
Simulation to final endpoint is 64.06 percent complete (st.  41/ 64), ETA 17:48
Simulation to final endpoint is 65.62 percent complete (st.  42/ 64), ETA 17:48
Simulation to final endpoint is 67.19 percent complete (st.  43/ 64), ETA 17:47
Simulation to final endpoint is 68.75 percent complete (st.  44/ 64), ETA 17:47
Simulation to final endpoint is 70.31 percent complete (st.  45/ 64), ETA 17:46
Simulation to final endpoint is 71.88 percent complete (st.  46/ 64), ETA 17:46
```

```
Simulation to final endpoint is 73.44 percent complete (st.  47/ 64), ETA 17:45
Simulation to final endpoint is 75.00 percent complete (st.  48/ 64), ETA 17:44
Simulation to final endpoint is 76.56 percent complete (st.  49/ 64), ETA 17:45
Simulation to final endpoint is 78.12 percent complete (st.  50/ 64), ETA 17:44
Simulation to final endpoint is 79.69 percent complete (st.  51/ 64), ETA 17:44
Simulation to final endpoint is 81.25 percent complete (st.  52/ 64), ETA 17:43
Simulation to final endpoint is 82.81 percent complete (st.  53/ 64), ETA 17:43
Simulation to final endpoint is 84.38 percent complete (st.  54/ 64), ETA 17:43
Simulation to final endpoint is 85.94 percent complete (st.  55/ 64), ETA 17:43
Simulation to final endpoint is 87.50 percent complete (st.  56/ 64), ETA 17:42
Simulation to final endpoint is 89.06 percent complete (st.  57/ 64), ETA 17:42
Simulation to final endpoint is 90.62 percent complete (st.  58/ 64), ETA 17:42
Simulation to final endpoint is 92.19 percent complete (st.  59/ 64), ETA 17:42
Simulation to final endpoint is 93.75 percent complete (st.  60/ 64), ETA 17:41
Simulation to final endpoint is 95.31 percent complete (st.  61/ 64), ETA 17:41
Simulation to final endpoint is 96.88 percent complete (st.  62/ 64), ETA 17:41
Simulation to final endpoint is 98.44 percent complete (st.  63/ 64), ETA 17:40
Simulation finished in 1387.00 min using an average of 1300.31 sec pr step
```

2.5.3 Visualizing beam profiles

As for the linear simulation, we have a set of *nsteps* 2D beam profiles. This is visualized using `plot_beamprofile` and presented in Figure 2.14. The cross sectional focal profiles for the azimuth and elevation foci are shown in Figure 2.15 and Figure 2.16. Using a *historyflag* of 2, the wave field is also stored at certain position pre-defined in the field *storepos*. The wave field at the end of the body wall (at z=d=20 mm) is loaded and visualized in Figure 2.17. The output from the third part of the tutorial should look something like:

```
Start of part 3: Visualization of results


In this part we will plot the beam profiles obtained by beamsim.
We will plot the (x,z) and (y,z)-plane beam profiles, the (x,y)-
plane focal profile for both the azimuth and elevation foci. The
pulse at the end of the body wall will also be visualized.

Press any key to plot beam profile
plot_beamprofile(rmspro,Propcontrol);

Press any key to plot focal point beam profiles
Find index of azimuth focal profile
idx = Propcontrol.Fx / Propcontrol.stepsize + 1;
plot_beamprofile(rmspro(:,:,idx,:),Propcontrol);
Find index of elevation focal profile
idx = Propcontrol.Fy / Propcontrol.stepsize + 1;
plot_beamprofile(rmspro(:,:,idx,:),Propcontrol);

Press any key to plot the pulse at the end of the body wall
Find the filename of the pulse
fn = [Propcontrol.simname get_strpos(Propcontrol.d*1e3) '.mat'];
load(fn,'u_z');
plot_pulse(u_z, Propcontrol);
```

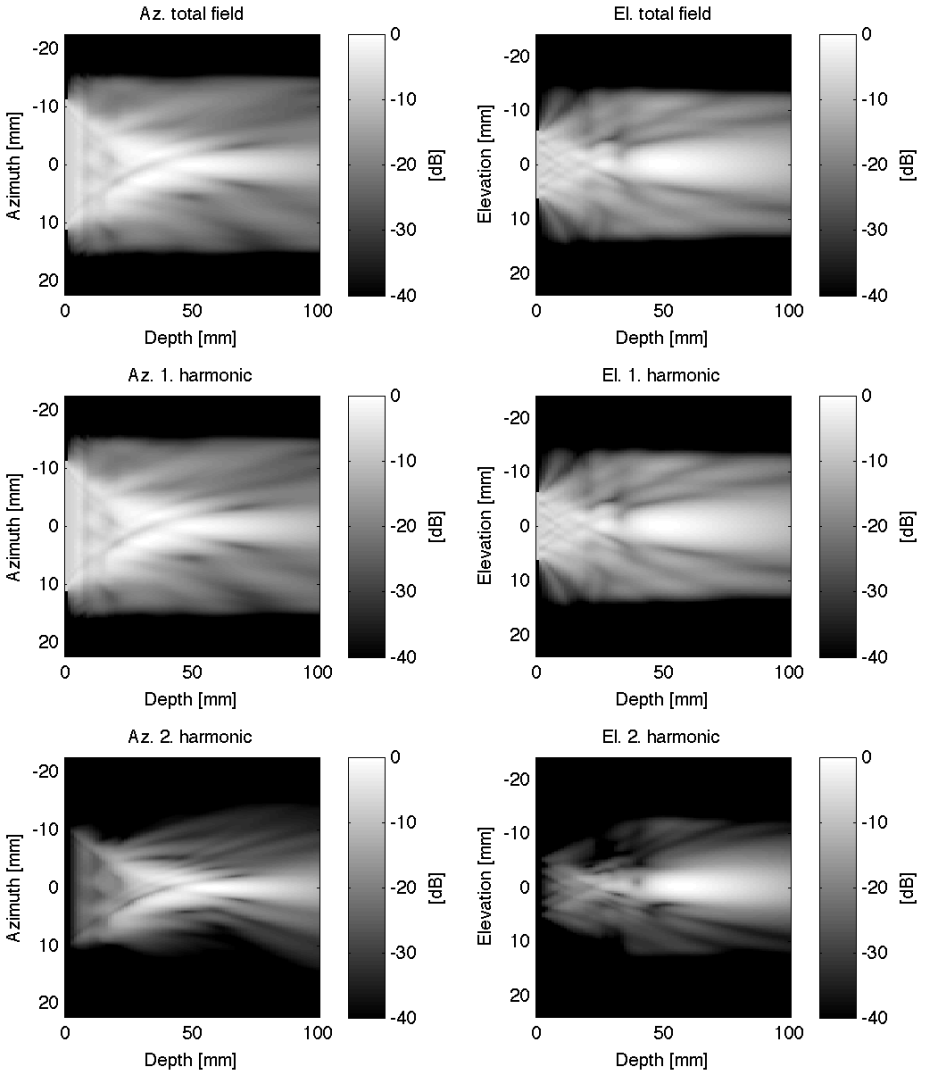


Figure 2.14: The temporal RMS-value of the radiated ultrasound beam in depth.

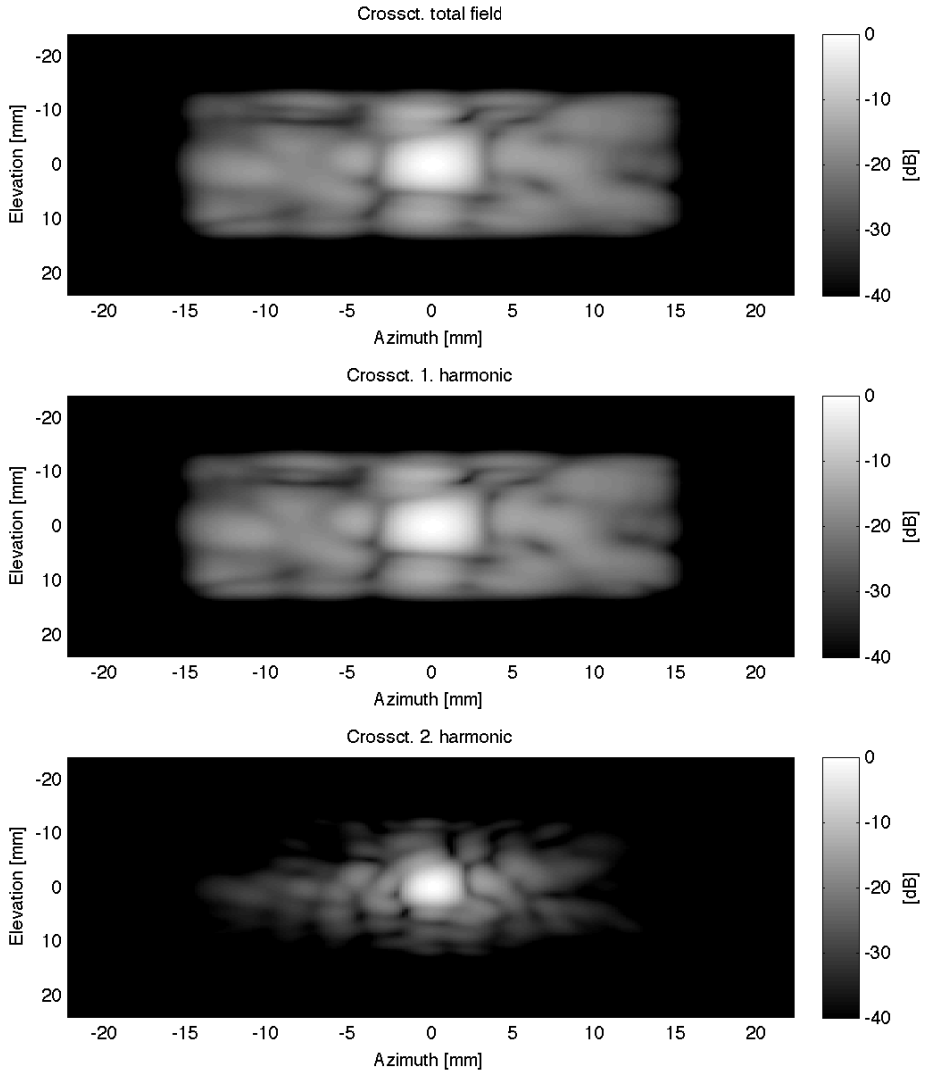


Figure 2.15: The temporal RMS-value of the cross sectional profile at the azimuth focal point. Note: the scale of the x- and y-axis are not equal.

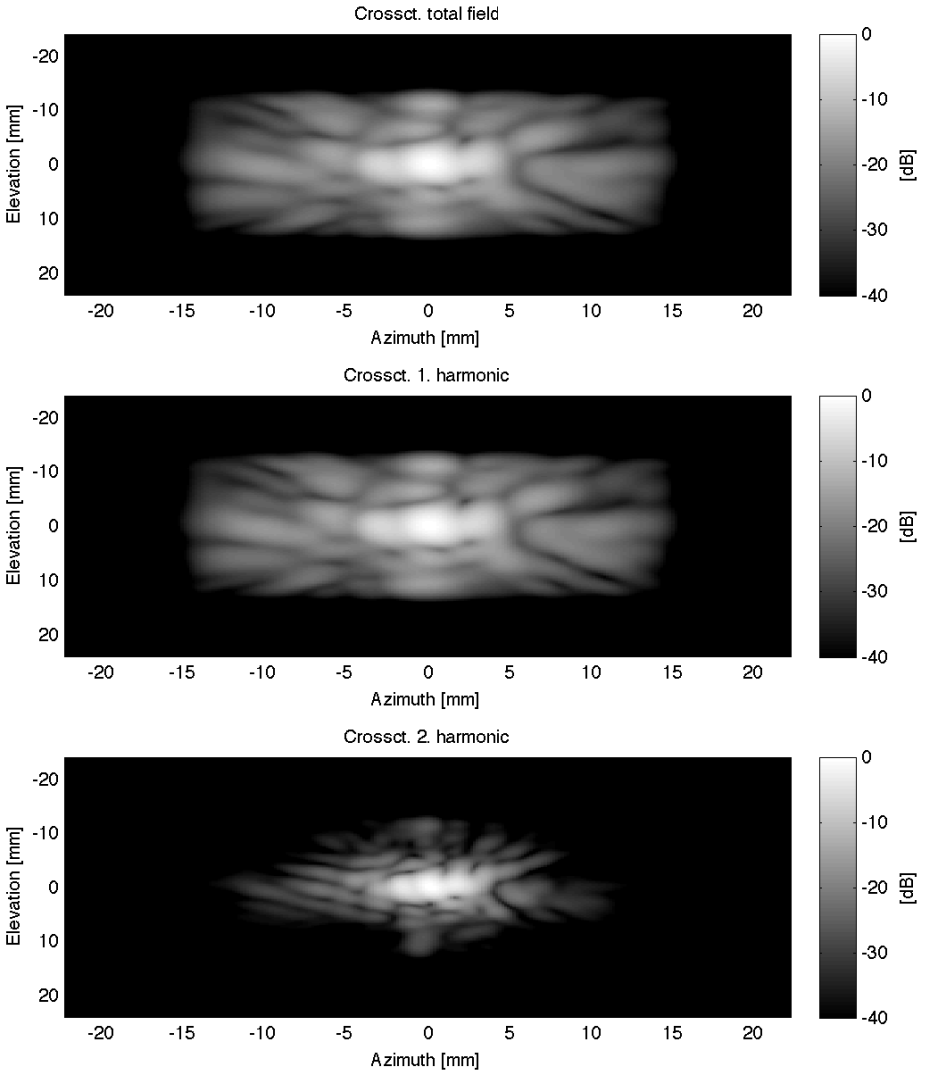


Figure 2.16: The temporal RMS-value of the cross sectional profile at the elevation focal point. Note: the scale of the x- and y-axis are not equal.

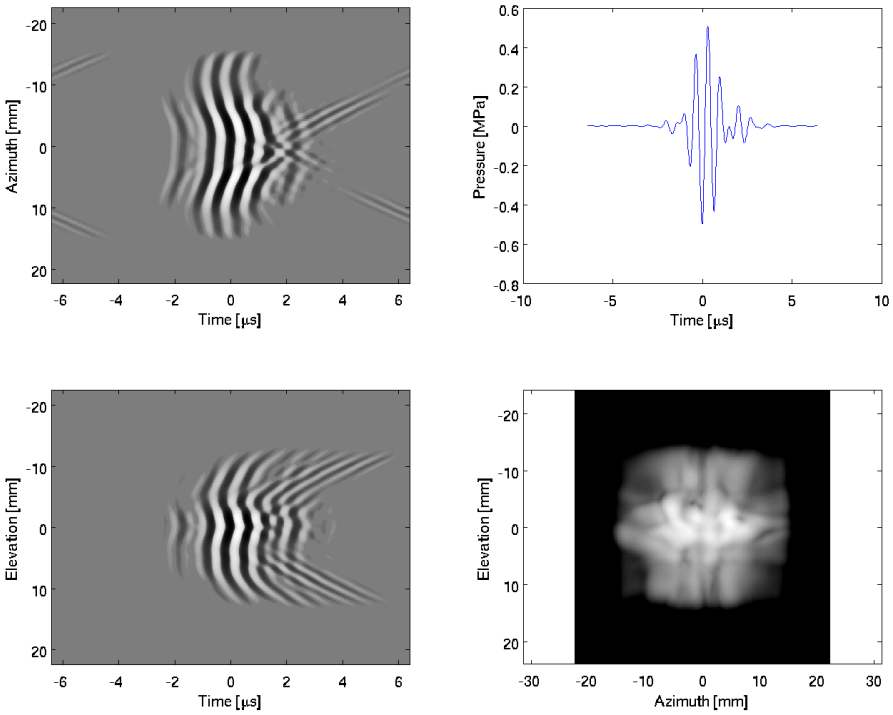


Figure 2.17: The wave field at the inner boundary of the body wall z=d.

Chapter 3

Function Reference

This chapter is a reference chapter for the main functions of Abersim. The functions may be divided into categories related to the preparation and running of a simulation, propagation core routines, material library and aberration functions.

3.1 The Propcontrol Struct

The `Propcontrol` is a struct consisting of several variables controlling the simulations in Abersim. `Propcontrol` is passed on to functions as an argument, and will be updated throughout the simulation and stored together with exported pulses and profiles. The latter is to ensure that when a pulse is loaded, the `Propcontrol` stored together with it will provide information about the position, sampling etc. of the pulse. The `Propcontrol` has several *fields* described below. All field varaible names will be given in *italic* fonts.

3.1.1 The Fields of 'Propcontrol'

A description of all fields of `Propcontrol` and their effects is provided below.

simname The name of the simulation. The *simname* is used as a root for all storage to disk. Depending on the specific export, some additional extenisions will be added, but the root of the filename is allways *simname*.

ndims The number of dimensions for the calculation domain. A 3D axis-symmetric simulation has a 2D computational domain, and *ndims* is therefore set to 2 in this case.

nx, ny, nt The number of points along each dimension. This is also the length of the Fourier transforms used. The first dimension (azimuth) is x, the second (elevation) is y, and t is the temporal dimension. The number of points are calculated automatically by `initpropcontrol` to yield a sufficiently large calculation domain, and is adjusted upward to yield number consisting of low factors to make the Fourier transform more efficient. Because Abersim integrates in the depth coordinate z, there is no need for a variable for nz. For 2D simulations ny is set to 1. Please note that the wave field itself is stored using Matlab column major ordering, making the temporal dimension to be the first, the elevation the second and the azimuth the third. For 2D grids, azimuth will be the second dimension.

PMLwidth The width of the absorbing boundary layer. Abersim has a Perfectly Matched Layer (PML) implemented, but this is more or less phased out and replaced by a regular window function to ensure better periodicity of the wave field and reduce wrap around effects introduced by the FFT.

diffrflag Flag determining the type of diffraction solver. The table below describes the different possibilities:

0 No diffraction.

1 Diffraction using the Angular Spectrum Method, or the pseudo-diferential model, described by Varslot and Taraldsen in Ref. [10] or Varslot and Måsøy in Ref. [9]. This solves the diffraction term exactly, but requires periodic boundary conditions (because of the Fourier transform). The wave number operator is fully contructed and enables the use of equidistant steps.

2 The same as 1, but the wave number operator is not contructed explicitly to save memory. Equidistant steps may not be used.

3 Not yet available.

4 Diffraction using a finite difference scheme presented by Varslot and Taraldsen in Ref. [10]. Here, the parabolic approximation is made, and the diffraction is not solved exactly.

5 The same as 4, but all matrices are made banded so that banded BLAS may be utilized to speed up the computation.

nonlinflag Switch non-linearity on or off. The value 1 denotes on.

lossflag The *lossflag* switches on (1) or off (0) frequency dependent attenuation. See Section 3.3.6 for more information about the attenuation itself.

abflag Flag for heterogeneities. The different types of heterogeneities are described below:

0 No aberration and homogeneous medium.

1 Aberration from a delay screen body wall. The body wall is contructed from a series of random numbers filtered to yield the desired statistics. See Section 3.5.4 for more information.

annflag Flag for annular transducer. If the transducer surface is circular, the *annflag* should be set to 1. This enables the use of a 2D computation domain if the medium is homogeneous. For arbitrary transducer geometries *annflag* must be set to 0.

equidistflag This flag should be switched on (set to 1) if the stepsize throughout the *whole* simulation is equal. When this flag is set, all steps are assumed to be of equal length, and the computation may be performed faster. See Section 3.3.2 for more information.

historyflag Flag for simulation export and history. Abersim may export profiles, pulses or both dependent on this flag. For more information on profiles and history, please see Section 3.2.3. The different options are:

0 No history. No profiles or wave fields are exported or stored to disk except for the wave field at the endpoint.

1 Profile history. The temporal maximum pressure and temporal RMS value for all harmonics up to *harm* (see further down in this section) are calculated and collected for each step. The on-axis pulse is also exported for each step. The profiles are stored at the end of the simulation along with the pulse at the endpoint, and is returned as output variables from `beamsim`.

2 Wave field history. The wave field is stored for each step of the simulation. No filtering or calculation of profiles are performed.

stepsize The stepsize, Δz, of the simulation. This should not be confused with the substeppsing dz presented later on. The computation time per step is for linear simulations constant. Because of this, the total computation time will descrease with increasing stepsize. An increase in stepsize should, however, must be handled with care to avoid "reflections" from the periodic boundary conditions. For non-linear simulations, an operator splitting is performed, and the substepsize dz is fixed. Because of this, the number of substeps increases for increasing stepsize, but the total computation time will be less dependent on the length of Δz.

shockstep A factor used to avoid shock formation. The non-linear solver does not have a shock-capturing routine, and if the distortion is too large, then the stepsize is shortened. This factor influence the threshold of when to sub-divide the original substep of the non-linear propagation.

endpoint The endpoint z_e of the simulation. For simulation with no export, the endpoint should be set to the point of interest, *e.g.,* the focal point.

currentpos The current position in depth z of a wave field. Is updated for each step and works as a position indicator for stored wave fields.

storepos The positions at which to store wave fields. This is set to be both the azimuth and the elevation focal point as well as the inner boundary of the body wall for heterogeneous simulations.

material The `Material` struct for the simulation. Each material has certain properties, and this struct contains all these properties. For more information on the material handling in Abersim, please consult Section 3.4. The struct has the followin fields:

- *i* The material indentifier. All materials are equipped wih an indentifier.
- *temp* The temperature of the material in Celcius.
- *eps* A vector containing $[\epsilon_n, \epsilon_a, \epsilon_B]$.
- *c0* The background speed of sound c_0.
- *rho* The mass density ρ.
- *kappa* The compressibility κ_∞.
- *betan* The coefficient of non-linearity β_n.
- *a* The attenuation constant a.
- *b* The attenuation frequency exponent b.

d The thickness of the body wall (or aberrator). See Section 3.5 for more information.

offset The offset of the transducer relative to a reference position. This is used for heterogeneous mediums when adjacent beams are to be compared. The aberrator should thus not change significantly, but be shifted a small amount. See Section 3.5 for more information.

numscreens The number of screens used in the delay screen body wall model. See Section 3.5.4 for more information.

abamp The amplitude of each delay screen. This may be tuned to construct weak and strong aberrators. See Section 3.5.4 for more information.

ablength The length scales of each delay screen. The screen is constructed by filtering random numbers using a spatial bandpass filter. The *ablength* defines the shortest and longest length scale of the heterogeneities, *i.e.,* the filter band. See Section 3.5.4 for more information.

abfile The filename of the file where the aberrator is stored. For the delay screen model, this is the file of random numbers. For the aberration phantom it is the file containing the list of the coordinates and radii of all the spheres. See Section 3.5 for more information.

Fs The sampling frequency. This is set to be the maximum of either 40 MHz or $10 \cdot f_c$.

dx, dy, dz, dt The resolution in space $\mathrm{d}x$, $\mathrm{d}y$ and time $\mathrm{d}t$. The spatial resolution is given as $\frac{\lambda}{2}$ where $\lambda = \frac{c_0}{f_i}$ where f_i is the *imaging* frequency. Then the spatial resolution is adjusted downward to yield an integer number of resolution points per element size e defined in the fields *esizex* and *esizey*. The temporal resoltion is given as $\mathrm{d}t = \frac{1}{F_s}$. The $\mathrm{d}z$ is the sub-cycling step of the non-linear propagation and is given by the wave number $\mathrm{d}z = k_t = \frac{c_0}{2\pi f_c}$ where f_c is the transmitted frequency. See Section 3.3 and 3.2.1 for more information.

fc The *transmitted* center frequency. Abersim generated pulses have a Gaussian shape and a center frequency of f_c. This frequency related to the n'th harmonic *imaging* frequency by $f_i = n_h f_c$ where n_h denotes the n'th harmonic.

bandwidth The relative bandwidth of the transmitted *pulse*, not the *transducer*, and is defined as a fraction, *e.g.,* 0.5 equals 50% bandwidth. See Section 3.2.1 for more information.

Np The number of periods in the transmitted pulse. See Section 3.2.1 for more information.

amplitude The pressure amplitude of the transmitted pulse in MPa. See Section 3.2.1 for more information.

harmonic The harmonic frequency used for imaging, *i.e.,* if *harm* is set to 2, then second-harmonic imaging is performed. If *harm* is set to 3, then third-harmonic and so forth. See Section 3.2.1 for more information.

filter A vector containing filter coefficients for each harmonic frequency. Defined as fractional bandwidth of the frequency of interest. See Section 3.2.3 for more information.

Dx, Dy Aperture size in x (azimuth) and y (elevation) direction. See Section 3.2.1 for more information.

Fx, Fy Focal depth in x (azimuth) and y (elevation) direction. See Section 3.2.1 for more information.

cchannel Center channel of transducer. Determines the positioning of the pulse within the computational domain. See Section 3.2.1 for more information.

Nex, Ney The number of transducer elements along each dimension. See Section 3.2.1 for more information.

esizex, esizey The size of the transducer elements along each dimension. See Section 3.2.1 for more information.

3.1.2 The Function 'initpropcontrol'

This function initiates the `Propcontrol`. The function takes several input arguments, and the default values of all fields may be found in Tutorial 1 (Sec. 2.1). The help file for the `initpropcontrol` is quite self-explainable.

3.2 The Simulation Itself

Abersim simulates *forward* propagation of acoustic pulses. By forward, we mean only one direction. The solution of the wave equation has two waves propagating in opposite directions, but Abersim chooses one of them, and forgets the other. This makes back scattering simulations somewhat troublesome in Abersim as they have to be split up in several simulations, but it is possible to perform this type of simulations.

Abersim integrates the wave field in the depth coordinate z, and this is possbile using the concept of retarded time (see Section 3.3). This is equivalent to following the propagating wave field. The wave field itself does not propagate in depth *within* the moving frame of reference. Within this frame of reference, no approximations are made. The wave equation in retarded time is then solved keeping only the forward component.

This section seek to present the most important functions involved in a full beam simulation. These functions are:

- `pulsegenerator`
- `beamsim`
- `pointsourcesim`
- `export_beamprofile`
- `plot_pulse`
- `plot_beamprofile`

The simulation consist of three steps: Preparation, propagation and visualization. The preparation part is covered by setting up `Propcontrol` and the wave field at the transducer. The latter may be done using `pulsegenerator`. For more complex wave fields, `pulsegenerator` will be insufficient, and the user must specify the wave field manually. The simulation itself is run using `beamsim`, and the export and visualization routines are also decribed in this section.

3.2.1 The Function 'pulsegenerator'

This function will generate initial wave fields in Abersim. Both pulse shape and transducer geometry may be arbitrary. One important physical interpretation is that the transducer *surface* is plane. All focusing is assumed to be either delay or lens focusing.

The Computational Domain

The computational domain in Abersim is a moving reference frame, or a retarded time frame. The retarded time frame consist of a plane and an interval in time. The plane extends into the (x, y)- or (r, θ)-plane, and is positioned at depth z, or in retarded time as $\tau{=}t{-}\frac{z}{c}{=}0$. The extension in time may be described as a recording of all wave passing this plane over a time interval $\tau{=}[-\frac{n_t}{2}\mathrm{d}t, \frac{n_t}{2}\mathrm{d}t]$. The reference frame is thus a domain with dimensions (x, t) or (r, t) for 2D simulations and (x, y, t) for 3D. The extensions of the spatial dimensions are either $[-\frac{n_x}{2}\mathrm{d}x, \frac{n_x}{2}\mathrm{d}x]$ (and similar for y) for 2D and 3D simulations with no symmetry, and $[0, n_x\mathrm{d}x]$ for 3D circular symmetric simulations.

The center channel relates to the computational domain, and determines where the transducer should be placed in the retarded time frame. The center channel is given as a matrix index and *not* as a pysical position in space. The center channel is thus for regular 2D and 3D $\frac{n_{x/y}}{2}+1$ and 1 for both x and y for 3D circular symmetric simulations. This ensure the zero to be placed at the center of the computational domain, see the next paragraph for more information on transducer coordinates.

Coordinates of the Transducer

The coordinate system of the transducer is important when it comes to defining the transmitted wave field. As for all numerical purposes a discretization must be made, and the nodal value represents the measure of interest in a neighborhood around its position. Figure 3.1 shows how the nodal values are defined and their region of validity.

For a rectangular transducer the number of resolution points dx per element e may be arbitrary. For an annular transducer, however, the center element must have an odd number of resolution points to ensure symmetry. The other elements may consist of an arbitrary number of points. The bandwidth of the transducers used in Abersim is assumed to be infinite, and the sensitivity is assumed to be 1 for all frequencies. All bandwidths used in Abersim relates to the *pulses*, not the transducer.

Pulse Shape

The default pulse shape produced by Abersim is a cosine signal of frequency f_c with a half-period cosine envelope of length equal to N_p (specified in field *Np*) periods of the form:

$$p = \cos\left(2\pi f_c t\right)\cos\left(\pi\frac{f_c}{N_p}t\right), \quad t\in[-\frac{N_p}{2}T_p, \frac{N_p}{2}T_p],$$

where $T_p{=}\frac{1}{f_c}$ is the period of the signal. The length N_p does not have to be an integer.

The pulse is then filtered in the frequency domain with a Gaussian filter of the form

$$H(f) = e^{\left(2a\frac{||f|-f_c|}{Bf_c}\right)^4}$$

where $a{=}\log_{10}10^{\frac{6}{20}}$, f_c the center frequency of the pulse, f the frequencies covered by the filter and B the -6 dB fractional bandwidth of the signal given by the `Propcontrol` field *bandwidth*. The amplitude of the pulse is then scaled so that the maximum of the *envelope* equals P_0, the *amplitude* field specifed in `Propcontrol`.

An arbitrary pulse shape provided by the user is possible. This must be specified as a vector of length shorter or equal to n_t. The vector should be tapered to zeros at the endpoints. If the vector is shorter than n_t, it is zero-padded to length n_t.

Transducer Geometry and Apodization

The transducer geometry is either rectangular or circular by default. The aperture size, D (field Dx and Dy), of the transducer is given as:

$$D_x = N_x n_x \mathrm{d}x, \qquad D_y = N_y n_y \mathrm{d}y,$$

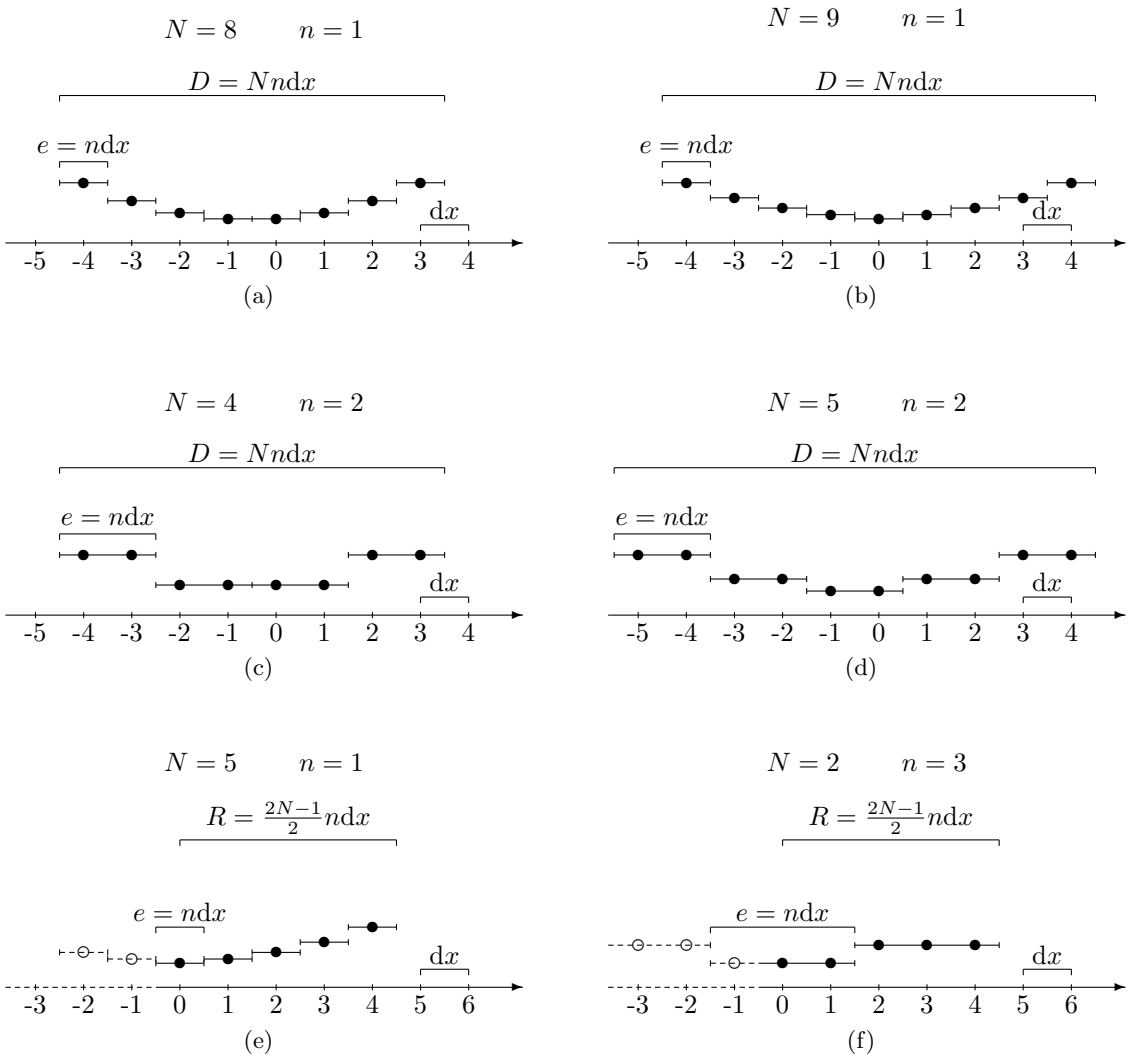


Figure 3.1: Definition of transducer coordinates. The filled circles indicate pressure nodes and their location. N denotes the number of elements, n the number of resolution points per element and e the size of one element. Subfigure (a)-(d) describes rectangular transducers and (e)-(f) annular transducers. For the annular transducers, the dashed line indicates symmetry, and the nodes marked with an empty circle is a mirror node of the filled node at the opposite, or symmetric, position.

where x denotes azimuth, y elevation, N is the number of elements (field *Nex* and *Ney*), n if the number of resolution points per element and dx and dy is the spatial resolution. For annular transducers D_x and D_y must be equal. The focal depths F_x and F_y are also required to be equal for annular transducers.

Apodization is set by the input argument *apod*. Default apodization is rectangular, which is equivalent to a matrix of ones over all active transducer elements. This corresponds to setting *apod* to 0. The default transducer is assumed to have no kerfs and transmits on adjacent elements, *i.e.,* no sparse apertures. Annular transducers return a circular shaped apodization of value one for full 3D simulations. The focusing bowl is calculated independent of the apodization, and this is why it is possible to have different focusing in azimuth and elevation direction for annular transducers. This is something to be aware of.

Another implemented feature is the *cosine* apodization. This is called when *apod* is set to an integer s indicating an apodization of the form:

$$A = \frac{1}{2}\left(\cos\left(\pi\sqrt{x^{2s}}\right) + 1\right), \quad x \in [-1, 1],$$

where x is either a scaled length measure across one of the transducer dimensions or the radius for annular transducer.

If the input apodization is specified as a matrix, the geometric shape of the transducer may be completely arbitrary. This is the way to generate wave fields from transducers with complex shapes. This apodization matrix will *redefine* the size of the tranducer according to what is specified in `Propcontrol`. To maintain the the desired size of the transducer, the apodization matrix should then be of proper size according to the aperture D, the element size and number of elements. The midpoint in x and y of the apodization is the point that will collocate with the specific center channel (*cchannel*) in `Propcontrol`. When using heterogeneous mediums, small shifts may be produced by adjusting the center channel. Larger shifts should be introduced through the *offset* field. This is to make sure that the transducer does not come too close to the boundaries of the computational domain.

3.2.2 The function 'beamsim'

The function `beamsim` is the main simulation script. This function will start up a simulation and propagate a (possibly given) wave field to a desired endpoint. The functions input arguments are `Propcontrol`, a wave field, a correction screen and a domain window. If the wave field is not given as input, it is generated using `pulsegenerator`. The screen input is used for aberration correction, and if this argument is empty or omitted, no correction is performed. The window input argument is the window used for tapering the wave field to zero at the boundaries to reduce wrap-around effects.

The pseudo code for beamsim is as follows:

1. Generate wave field (if not specified)
2. Calculate number of steps needed for simulation
3. Introduce aberration correction (if *screen* is specified)
4. Generate wave number operator for propagation (if *diffrflag* is 1, 2 or 3)
5. Generate window used to taper wave field to zero at the boundaries (if not specified)
6. Export start profile
7. Run `bodywall` (Sec. 3.5.1) if the medium is heterogeneous
8. Run *nsteps* of propagation using
   (a) Propagation
   (b) Windowing of solution
   (c) Export of history
9. Save profiles and wave field

The function `beamsim` should cover most beam simulations performed with Abersim. It is not recommended to do changes in `beamsim` since everything that deals with the wave field, correction and windowing may be specified as input arguments. When it comes to export, modifications should be done in `export_beamprofile` rather than in `beamsim`.

3.2.3 The function 'export_beamprofile'

This function governs the calculation and export of beam profiles and wave fields. Wave fields are allways stored in raw format, that is: No filtering has been performed prior to storage.

Filtering of Harmonic Components

The harmonic components of a wave field are obtained through filtering. The filter routines are far from optimal, and the user is encouraged to find and use own filters. The field *filter* in `Propcontrol` is a vector containing a set of fractions used for filtering out harmonic components. The vector has length n_h, where n_h is the highest desired harmonic frequency.

The filter used by default is a frequency domain Gaussian bandpass filter of the kind

$$H(f) = e^{\left(2a\frac{||f|-h_n f_t|}{B_n h_n f_t}\right)^4}$$

where a=$\log_{10} 10^{\frac{6}{20}}$, f_t the transmit frequency, f the frequencies covered by the filter, h_n is the integer denoting the n'th harmonic, and B_n is the -6 dB fractional bandwidth of the signal specified as the n'th component of the *filter* vector.

Calculation of profiles

The maximum temporal pressure profile is found as the maximum of the *envelope* of the filtered signal. The envelope of the filtered signal is found using:

$$p_e(t) = |p(t) + \mathcal{H}\{p(t)\}|$$

where $\mathcal{H}\{\cdot\}$ denotes the Hilbert transform.

The maximum pressure is then found as

$$p_{\max} = \max_i p_e(t_i),$$

where t_i are the temporal nodal points and i=$1, \ldots, n_t$.

The temporal RMS value is calculated from the signal itself and not the envelope as:

$$p_{\text{rms}} = \sqrt{\frac{1}{n_t}\sum_{i=1}^{n_t} |p(t_i)|^2}.$$

Storage of Wave Fields

The wave fields are stored in separate files with a name specified by *simname* and an extension specifying the position of the wave field. This extension is given as a underscore and a five digit number. The number is the position given in units of 10^{-5} m, or the right digit is 1/100 of a mm. The position `_99999` denotes 99.999 cm, or just below one meter.

3.2.4 The Function 'plot_pulse'

The `plot_pulse` function visualize wave fields in Abersim. The function recognizes whether the pulse is a 2D or 3D wave field. Wave fields are visualized in the (x,t) (or (y,t) or (r,t)) retarded time frame, *i.e.,* the computational domain, using a two-sided logarithmic scale. The dynamic range is given as the input argument *dyn*, and the default is 40 dB. A two-sided logarithmic scale with X dB dynamic range means that the visual parts of the wave field will range from $+X$ dB to $-X$ dB, where the peak positive pressure will be at $+X$ dB and the maximum negative pressure at $-X$ dB. All wave field visualizations in the $(x/y/r,t)$-frame is plotted in a two-sided logarithmic scale unless *dyn* is set to zero, then the scale is linear. See the tutorials in Chapter 2 and the help file for more information.

For 2D wave fields, the wave field itself is plotted as a one-panel figure. For 3D wave fields, the figure will have four panels. The top left panel shows the wave field over the (x,t)-frame (azimuth direction) at the elevation center channel given by the second element of the center channel position (*cchannel(2)*). The bottom left panel shows the (y,t)-frame (elevation direction) at the azimuth center channel (*cchannel(1)*). The top right panel shows the center channel pulse as a function of time in a *linear* scale. The bottom right panel show the RMS profile of the wave field over the azimuth and elevation direction. This is the RMS intensity over the cross-section of the beam at position z, and is the same as the beamprofile of the *unfiltered* wave field at this depth.

3.2.5 The Function 'plot_beamprofile'

The `plot_beamprofile` function visualizes beam profiles from Abersim. The beam profiles are 4D arrays of dimensions (n_x, n_y, n_z, n_h) where n_z are the number of depths the profiles are taken from, and n_h the number of harmonic profiles. The beam profiles solemnly consist of positive numbers, and the visualization is performed using a one-sided logarithmic scale with dynamic range defined by *dyn*. If *dyn* is 0, a linear scale i used.

A full 4D beam profile will be plotted in a figure with $n_h \times 2$ panels. The left column is the azimuth profiles of the harmonic components, and the right is the elevation. If the size n_z=1, a cross-sectional profile is visualized. If both n_z and n_x or n_y are 1, 1D profiles are plotted. See the tutorials in Chapter 2 and the help file for more information.

3.3 Propagation

The propagation of wave fields is the main part of Abersim. The propagation is the most time consuming part of the code, and the one that benefits the most from optimization. The five main functions related to propagation are:

- `propagate`
- `get_wavenumbers`
- `nonlinearpropagate`
- `burgerssolve`
- `attenuationsolve`

3.3.1 Short Mathematical Description

The underlying differential equation for wave propagation is the Westervelt equation

$$\nabla^2 p - \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = -\frac{\beta_n \kappa}{c^2}\frac{\partial^2 p^2}{\partial t^2} - \frac{1}{c^2}\frac{\partial^2 \mathcal{L}p}{\partial t^2}, \quad c = \frac{1}{\sqrt{\rho\kappa}},$$

where p is the acoustic pressure, c the speed of sound, β_n the coefficient of nonlinearity and $\mathcal{L}$ the kernel of a convolution operator accounting for attenuation.

According to Varslot and Taraldsen in Ref. [10], the Westervelt equation may be transformed into a directional wave equation through the introduction of retarded time coordinate $\tau{=}t{-}z/c$. The resulting directional Westervelt equation is then:

$$\frac{\partial^2 p}{\partial \tau \partial z} = \frac{1}{2}\left(\nabla^2 - g\right) p - \epsilon_t \frac{\partial^2 p}{\partial \tau^2} + \frac{\epsilon_n}{2}\frac{\partial^2 p^2}{\partial \tau^2} + \epsilon \frac{\partial^2 \mathcal{L}p}{\partial \tau^2}.$$

where ϵ_t, ϵ_n and ϵ are scaling constants. Integrating this with respect to the retarded time coordinate τ we get:

$$\frac{\partial p}{\partial z} = \frac{1}{2}\int_{-\infty}^{\tau} \left(\nabla^2 - g\right) p \,\mathrm{d}\tau_0 + \left(\epsilon_n p - \epsilon_t\right)\frac{\partial p}{\partial \tau} + \epsilon \frac{\partial^2 \mathcal{L}p}{\partial \tau^2}.$$

Please note that an introduction of the parabolic approximation through $\frac{\partial^2 p}{\partial z^2}{\approx}0$ will result in the classic KZK equation as:

$$\frac{\partial p}{\partial z} = \frac{1}{2}\int_{-\infty}^{\tau} \left(\nabla_\perp^2 - g\right) p \,\mathrm{d}\tau_0 + \left(\epsilon_n p - \epsilon_t\right)\frac{\partial p}{\partial \tau} + \epsilon \frac{\partial^2 \mathcal{L}p}{\partial \tau^2}.$$

The only difference between the directional Westervelt equation and the KZK equation is in the diffraction term where we have *not* neglected diffraction in the z direction for the Westervelt case.

Operator splitting enables the use of separate solutions for each of the three terms of both the directional Wastervelt equation and the KZK equation. The solution of these terms will be treated separately in the following sections (Secs. 3.3.2-3.3.6).

3.3.2 Linear Propagation and Diffraction

Focusing on the diffraction term in the Westervelt equation in retarded time, it is obvious that this is the diffraction for a linear wave equation in a lossless medium. The linear wave equation, or the diffraction term is:

$$\frac{\partial^2 p}{\partial \tau \partial z} = \frac{1}{2}\left(\nabla^2 - g\right) p,$$

where $\nabla_\perp^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ and g accounts for heterogeneities.

Abersim introduces heterogeneities using a delay screen body wall, så all *propagation* is in fact homogeneous. The heterogeneous effects are introduced as phase shifts between propagation steps (see Section 3.5). This way, g will be zero for propagation between two screens. The diffraction term then simplifies to:

$$\frac{\partial^2 p}{\partial \tau \partial z} = \frac{1}{2}\nabla^2 p,$$

All these arguments are also valid for the approximative KZK equation. Expanding the operators and rearranging this we find the following differential equation:

$$\frac{\partial^2 p}{\partial z^2} - \frac{2}{c}\frac{\partial^2 p}{\partial \tau \partial z} + \nabla_\perp^2 p = 0.$$

which is a *second-order* partial differential equation with coupling in both x, y and τ.

Applying the Fourier transform in time (τ) to this equation we obtain:

$$\frac{\partial^2 \hat{p}}{\partial z^2} - 2ik_t\frac{\partial \hat{p}}{\partial z} + \nabla_\perp^2 \hat{p} = 0, \quad k_t = \frac{\omega}{c},$$

where $\hat{p}$ denotes the temporal Fourier transform of p, and k_t the wavenumber in time. This equation is a linear stationary Helmholtz equation and is decoupled in time, but still coupled in space.

A decoupling in space may be performed using a spatial Fourier transform, and this will result in a fully decoupled second-order ordinary differential equations (ODE) as:

$$\frac{\partial^2 \tilde{p}}{\partial z^2} - 2ik_t\frac{\partial \tilde{p}}{\partial z} - \left(k_x^2 + k_y^2\right)\tilde{p}.$$

where $\tilde{p}$ denotes the Fourier transform in time and all spatial dimensions except the spatial integration varaible z, and k_x and k_y the wavenumbers in x and y respectively. For the discretized equation, we will have a system of ODEs, one for each combination of k_x, k_y and k_t, that needs to be solved.

This differential equation has a solution of the form

$$p(z_0 + h) = Ae^{i\left(\sqrt{k_t^2 - k_x^2 - k_y^2} - k_t\right)h} + Be^{-i\left(\sqrt{k_t^2 - k_x^2 - k_y^2} - k_t\right)h},$$

where h is the stepsize.

Choosing only the solution propagating in the positive direction (*within* the retarded time frame), we may find the solution to be:

$$\tilde{p}(z_0 + h) = \tilde{p}(z_0)e^{i\mathcal{K}_z h}, \quad \mathcal{K}_z = \sqrt{k_t^2 - k_x^2 - k_y^2} - k_t,$$

where $\mathcal{K}_z$ is a wave-number (scalar) operator. The solution at point $z_0 + h$ is thus found throught the inverse Fourier transforms in x, y and τ of $\tilde{p}$. This approach is referred to as the pseudo-differential model in Ref. [10] and [9]. Using this approach, the forward component of the propagating wave is solved exactly in retarded time coordinates. The KZK equation may now be modified according to this, and the parabolic approximation may be omitted.

The algorithm for solving the diffraction (or in fact the linear wave equation) is:

1. Perform Fourier transform in space
2. Perform Fourier transform in time
3. Do multiplication with $\mathcal{K}_z$ for all wave numbers
4. Perform inverse Fourier transform in time
5. Perform inverse Fourier transform in space

The `Propcontrol` field *equidistflag* enables the pre-calculation of $e^{i\mathcal{K}_z h}$ for a constant h. This way, the multiplication step will be a single complex multiplication for all points in space and time instead of one complex multiplication and one complex function evaluation.

The function `propagate` will recognize whether the propagation is linear or non-linear based on the `Propcontrol` field *nonlinflag*. If the propagation is linear, the algorithm above is performed, and if the propagation is non-linear the function `nonlinearpropagate` is called to perform operator splitting and sub-cycling.

3.3.3 The Wave-Number Operator

The wave-number operator $\mathcal{K}_z$ from the pseudo-differential model plays an important role in the diffraction of the pulse. For linear propagation, the attenuation is baked into the wave number operator. This is not desirable to do for non-linear propagation. The splitting scheme for diffraction and non-linearity/attenuation uses a *fixed* substepsize based on dz, but the second splitting between non-linearity and attenuation may do even further sub-cycling to avoid shock formation by introducing attenuation more often. This is why the attenuation is kept outside the wave number operator for non-linear propagation.

Decoupling of the differential equation for regular 2D and 3D is performed using the Fourier transform in the Angular Spectrum Method. The wave-number operator is stored as an array of the same size as the wave field. The wave-number operator has the general form:

$$\mathcal{K}_z = \sqrt{k_t^2 - k_x^2 - k_y^2} - k_t.$$

The expression within the root may become negative and thus imaginary. These wave-numbers must be treated with special care because the function $e^{-i\mathcal{K}_z h}$ will be $e^{|\mathcal{K}_z| h}$ and amplify the solution. This effect is called evanescent waves. To avoid evanescent waves, the wave number operator is modified to be:

$$\mathcal{K}_z = \begin{cases} \sqrt{k_t^2 - k_x^2 - k_y^2} - k_t & \text{for } k_t^2 - k_x^2 - k_y^2 > 0 \\ -i\sqrt{k_x^2 + k_y^2 - k_t^2} - k_t & \text{for } k_t^2 - k_x^2 - k_y^2 < 0 \end{cases}$$

This way, evanescent waves will be suppressed and dampened out.

Using the Angular Spectrum Method without a fully constructed wave numer operator may be advantageous when there is a need for saving memory. This is for *diffrflag* equal to 2. Then the wave-number vectors k_t, k_x, k_y and the eventual attenuation are returned.

When the parabolic approximation and the KZK equation is solved, the wave number operator will be a long vector containing the difference matrices used by the finite difference scheme in `propagate`.

3.3.4 Non-Linear Propagation and Operator Splitting

The function `nonlinearpropagate` governs the second operator splitting scheme used in solving the directional Westervelt or KZK equation. The operator splitting for the full propagation is divided into two parts. First the diffraction and the non-linearity/attenuation are split using a first order Gudonov splitting [10]. To do this, with a sufficiently low splitting error, several substeps are taken. This substepsize is determined by the field *dz* in `Propcontrol`. Then, the non-linearity and attenuation are split using a second Strang splitting scheme of second-order. Here, another possible substepping is performed based on estimates of the shock formation of the pulse. If the temporal gradient at some point is higher than a certain limit, the substep is divided into even smaller steps. This is to avoid shock formation.

The diffraction is solved by calling `propagate` as if the propagation was linear. Then one step is performed, and `nonlinearpropagate` is not called recursively. This way, equidistant steps may be utilized with the substepsize as h instead of the full stepsize. The substepsize is fixed through the whole simulation, and if each regular stepsize is an integer number of substepsize, *equidistflag* may be used without further concern.

Non-linearity and attenuation are two phenomena with opposite effect. The non-linearity generates higher harmonic frequencies through forward distortion of the pulse. The attenuation tend to attenuate higher frequncies more than lower, and will make sure that a perfect mathematical shock never is formed. Beause of this, sub-cycling is performed to introduce attenuation as often as necessary.

3.3.5 Non-Linearity and Burgers' Equation

The term accounting for non-linearity in the directional Westervelt and KZK equation originally has the form:

$$\frac{\partial p}{\partial z} = (\epsilon_n p - \epsilon_t) \frac{\partial p}{\partial \tau}$$

where p is a function of x, y, z and t. Assuming plane wave propagation in the z direction, p will only be a function of z and t, and the above equation may be rearranged to form a modified Burgers equation on the standard form through a change of variables:

$$\frac{\partial p}{\partial z} = \epsilon_n p \frac{\partial p}{\partial \tau} + \epsilon \frac{\partial^2 p}{\partial \tau^2}.$$

Because if the plane wave approximation, the nonlinearity is only accounted for parallel to the z-axis.

The non-linear steepening of the pulse is solved using the method of characteristics. For small steps, a perturbation in the temporal grid points proportional to the pressure at each point is introduced, causing them to not longer be equidistant. This is a distortion of the 1D pulse causing the wave peaks to move faster than the throughs. The pressure is then resampled at equidistant time points, a process that introduces a small interpolation error, but for a sufficiently high sampling frequency, this error is neglible.

3.3.6 Frequency Dependent Attenuation

The attenuation term may be integrated in the same way as the non-lineaorty term to form:

$$\frac{\partial p}{\partial z} = \epsilon \frac{\partial \mathcal{L} p}{\partial \tau}.$$

The application of the convolution operator $\mathcal{L}$ follows a frequency dependent power law attenuation of the form:

$$\alpha(f) = a f^b,$$

and is defined through its Fourier transform as:

$$\mathcal{F}_\tau \left\{ \frac{\partial \mathcal{L} p}{\partial \tau} \right\} = - |\omega|^b \mathcal{F}_\tau \{p\}.$$

The constant ϵ is given as:

$$\epsilon = \frac{\ln 10}{20} \frac{a}{(2\pi)^b}$$

where a and b are the constants of the power law. This model in itself does not obey causailty, and is modified according to the Kramers-Kronig relations to form:

$$\mathcal{F}_\tau \left\{ \frac{\partial \mathcal{L} p}{\partial \tau} \right\} = \left(- |\omega|^b + i \mathcal{H} \left\{ |\omega|^b \right\} \right) \mathcal{F}_\tau \{p\},$$

where $\mathcal{H}\{\cdot\}$ denotes the Hilbert transform [9, 10]. This model is implemented in Abersim.

3.4 Implementation of Materials

Abersim handles a wide range of materials. Almost all of the implemented materials are soft tissue models. The materials are tuned according to Duck and Hamilton and Blackstock (Refs. [4,7]. The material is brought into the simulation through the `Material` struct, and this struct contains the parameters used by Abersim. The struct will get its parameter values from material files corresponding to specific materials.

Some important functions are:

- `Material`
- `MUSCLE`, `LIVER`, `FAT`, `BLOOD`, etc.
- `list_material`
- `get_matparam`

3.4.1 The Material Struct

The `Material` struct has several fields presented below:

i This is a material indentifier and is unique for all materials. The material indentifier 0 denotes water, indentifiers 1-10 indicates soft tissues, 11-20 tissue mimicking phantoms etc, 21-30 liquids, 31-40 gases and 41-50 solids. The function `list_material` displays the different implemented or reserved identifiers. The user is encouraged to implement own materials and to add ranges of material indentifiers.

temp The temperature of the material in Celcius degrees. For tissues 37 degree Celcius is the default.

eps A vector containing the ϵ values [ϵ_n, ϵ_a, ϵ_b]. These constants are scaled quantities used in the propagation and information about how they are calculated may be found in Ref. [10].

c0 The background speed of sound.

rho The mass density of the material at rest.

kappa The compressibility of the material at rest.

betan The coefficient of non-linearity. The β_n constant relates to the common $\frac{B}{A}$-parameter as β_n=1+$\frac{B}{2A}$.

a Abersim uses a power law absorption model of the form af^b (see Section 3.3.6). This parameter is the a in the constant, or not frequency dependent, part of the attenuation.

b This is the exponent b in the power law attenuation.

3.4.2 The Material File 'MUSCLE'

Material files are files containing *measurements* of different material parameters. The measureable parameters are such as speed of sound, mass density, coefficient of non-linearity and the two attenuation coefficients. The files are implemented with a selector that returns a specific parameter based on the input *param*. The input value *param* may be either of the following: `MATERIALID`, `WAVESPEED`, `DENSITY`, `BETAN`, `ATTCONST` and `ATTEXP`. The measurements are implemented as arrays named *temps* and *meas* corresponding to a measurement of one parameter at one specific temperature. The return parameter of the function at the desired temperature *temp*, specified at input, is found through interpolation.

To add new materials, a material file must be created and the function `list_material` must be updated. The function `list_material` contains arrays of material names and corresponding identifiers. The *name* of the new material must be added to this list, and the material file *must have the exact same name in all upper-case letters*. The extension `.m` has to be added at the end.

An example material function, `MUSCLE`, is included below:

```
function value = MUSCLE(param, temp)

%
% value = MUSCLE(param, temp)
%
% The function returns the value of a given parameter for 'MUSCLE'
% For no input arguments the material index is returned (equivalent to
% calling function with param=MATIDX).
%
%
% Variables in:
% param       The desired parameter. May be one of the following
%             _measureable_ material parameters:
%               MATIDX      Material identifier
%               WAVESPEED   Speed of sound
%               DENSITY     Mass density
%               BETAN       Coefficient of non-linearity
%               ATTCONST    Constant of freq. dep. attenuation (a*f^b)
%               ATTEXP      Exponent of freq. dep. attenuation (a*f^b)
%
% temp        Optional. The temperature.
%
%
% Variables out:
% value       The value of the desired parameter.
%

%
% Copyright (C) 2004, 2008 The Abersim group
% Copyright (C) 2004, 2008 Department of Circulation and Medical Imaging
% Copyright (C) 2004, 2008 Norwegian University of Science and Technology
%
% This file is a part of the Abersim computer package
%
% This Abersim computer package is free software; you can redistribute it
% and/or modify it under the terms of the GNU General Public License as
% published by the Free Software Foundation; either version 2 of the
% License, or (at your option) any later version.
%
% This Abersim computer package is distributed in the hope that it will be
% useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program; if not, please download it from
% <http://www.gnu.org/licenses>
%

%
% Created July 2008 by Halvard Kaupang, ISB
% Based on earlier works by Gunnar Taraldsen
%

% Material identifier (call list_material for avail. identifiers)
if nargin==0
  value = 1;
```

```
  return;
end
if nargin<2
  temp = 37;
end

switch param
  case MATIDX
    value = MUSCLE;
    return;

  case WAVESPEED
    % Duck'90, p.76.
    c0 = 1529; %  18 Celsius, cardiac
    scale = 1.1; % p.86
    temps = [18, 40]'; %Ext. acc. to Mast(part II simulations).
    meas = [c0; c0 + scale * (40-18)];

  case DENSITY
    % Duck'90, p.138 and p.141
    rho0 = 1060; % 37 Celsius
    temp0 = 37;
    alpha0 = 3.75e-4; % Volume expansion cow
    temps = (35:40)';
    meas = rho0 * exp(alpha0 * (temp0 - temps));

  case BETAN
    % Hamilton'98 p.36. Duck'90 Cardiac p.98
    temps = [36, 37]';
    meas = 1 + 0.5*[5.8; 5.8];

  case ATTCONST
    % Duck, tab. 4.16
    temps = [36; 37];
    meas = [.52; .52] ; % dB / cm

  case ATTEXP
    % Duck, table 4.16
    %aTM = [20; 30];
    temps = [20,40];%This is not correct!
    meas = [1.1; 1.1];

end
if length(meas) > 2
  method = 'spline';
else
  method = 'linear';
end

value = interp1(temps, meas, temp, method, 'extrap');
```

3.4.3 The Function 'list_material'

This function will list all material names and identifiers when run without input arguments. If a string is presented as the input, and if the string corresponds to a material name, the identifier for this material is returned. If an integer is presented as an input, the material name for this identifier (if any) will be returned.

3.4.4 The Function 'get_matparam'

The function `get_matparam` returns parameters directly from the material files. All other functions regarding material parameters will call this function. The function itself is very simple, and should *not* be altered.

3.4.5 Other Material Functions

A list of material functions are provided below. The top five functions return standard measureable quantities, the rest return quantities calculated from measured ones.

- `get_wavespeed`
- `get_density`
- `get_betan`
- `get_attconst`
- `get_attexp`
- `get_compressibility`
- `get_epsn`
- `get_epsa`
- `get_epsb`

3.5 Aberration and heterogeneities

Abersim was from the very beginning contructed to simulate forward distortion caused by heterogeneities, *i.e.,* phase front aberrations, or phase aberrations or just aberrations. This is an important part of the program, and more information about the theory of acoustics in heterogeneous media may be found in the book of Angelsen [1].

The heterogeneities are in Abersim assumed to be located withtin the body wall. Once the pulse has passed the body wall, the medium is assumed to be homogeneous. The body wall, however, may be of arbitrary thickness, allowing for simulation in fully heterogeneous media. The aberrator may also be of arbitrary strength.

Heterogeneities are incorporated in Abersim as a delay screen body wall. This is further described in Section 3.5.4.

The main functions related to aberration and heterogeneous media are:

- `bodywall`

- `timeshift`
- `aberrator`

3.5.1 The Function 'bodywall'

The function `bodywall` is a simulation script equivalent to `beamsim`. The function will plan and perform a simulation up to the endpoint specified by the field d in `Propcontrol`, which is the thickness of the body wall. The number of screens will determine the spacing between the screens. The screens are equally spaced over the thickness d, and there is never a screen at the transducer surface, and allways a screen at the inner boundary of the body wall at position d. The function has an input argument *dir* specifying a direction, and this is 1 for regular transmit beam simulation (`beamsim`) and -1 for point source simulations (`pointsourcesim`). If *dir* is 1, then the phase shifts are introduced *after* propagation of one or more steps, and if it is -1, the phase shifts are introduced prior to stepping.

The stepsize is adjusted to be equidistant within the body wall. Given a distance between two screens of 1.5 times the original stepsize, two steps are need to propagate from one screen to another. To enable the use of equidistant steps, the stepsize within the body wall will be adjusted such that the wave field will be propagated between screens using two *equidistant* stepsizes. By varying the thickness of the body wall and number of screens, this stepsize adjustment may be avoided such that the regular stepsize may be utilized for the whole simulation.

The return arguments are the wave field at position d or at the endpoint if the simulation is performed in a fully heterogeneous medium. The beam profiles (if any) will be returned as well, and incorporated in eventual beamprofiles from `beamsim`.

3.5.2 The Function 'timeshift'

The function `timeshift` introduces the phase shifts in the wave field. This is also used for delay focusing and is called by `pulsegenerator`. The phase shifts are introduced in the frequency domain according to the formula:

$$p_{\text{shifted}} = \mathcal{F}_t^{-1} \left\{ \mathcal{F}_t \{p\} \, e^{-i\pi\delta} \right\}$$

where δ is the phase shift in samples. To make shifts in the frequency domain allows for shift of sub-sample resolution.

3.5.3 The Function 'aberrator'

This function `aberrator` will calculate the phase shifts for the whole simulation. The function has `Propcontrol` as input argument.

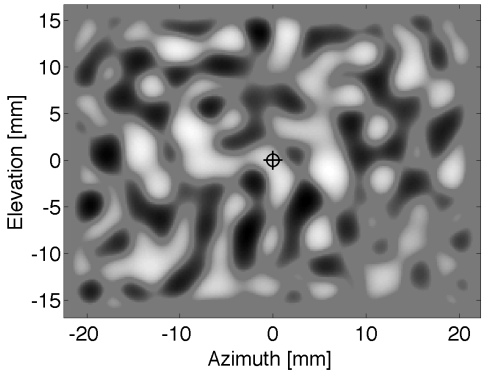
3.5.4 Delay screen body wall

The delay screen body wall is contructed from a sequnce of random numbers. This sequence may be downloaded from the Abersim website. The random number may be generated using a random number generator, but this should have the ability to return the

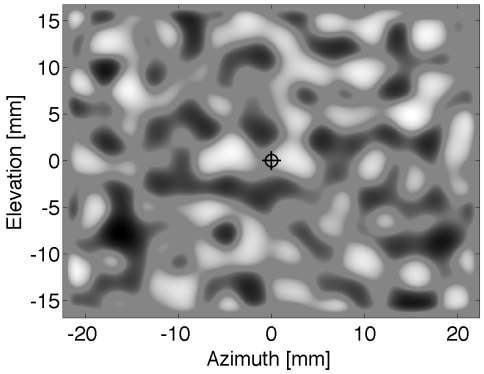
same numbers for a given seed or input. The numbers are bandpass filtered using a 2D spatial annular filter. The cut-off frequencies of the filter are defined in the `Propcontrol` field *ablength*. Examples of delay screens may be found in Figure 3.2.

The shortest length scale defining the upper cut-off frequency is set to be the smallest heterogeneous structure present, and is usually around the size of a few wavelengths. The shortest length scale is the most crucial as seen in Fig. 3.2. The longest length scale should be set to quite much larger, and is usually about the size of an organ or larger for soft tissue purposes.

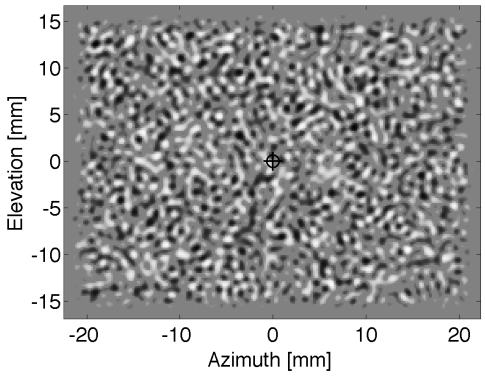
An increase in sampling rate in space should not affect the *physical* size of the heterogeneities. For panel a and b in Figure 3.2 we see that the heterogeneities have different patterns. This is because a longer Fourier transform, and thus filter, is used. To be able to use the very same aberrator for different samplings, the one with the finest sampling will have to be generated, and then the coarser delayscreen should be resampled or interpolated from this screen.



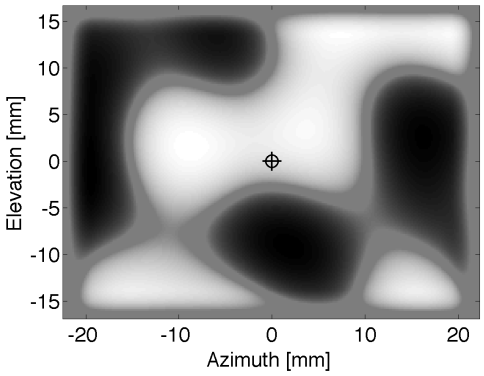
(a) 256×192 points, dx=dy=ds, *ablength* of 4 to 200 mm



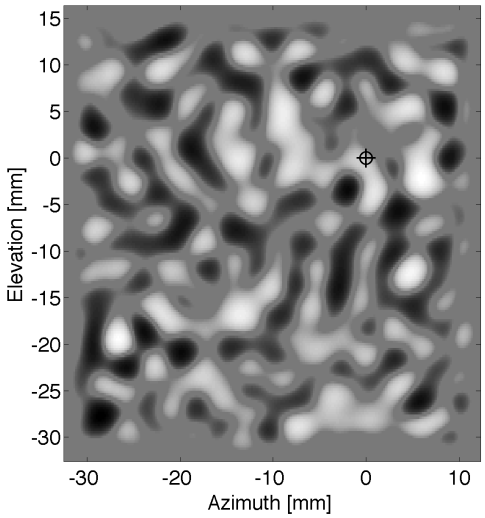
(b) 512×384 points, dx=dy=ds/2, *ablength* of 4 to 200 mm



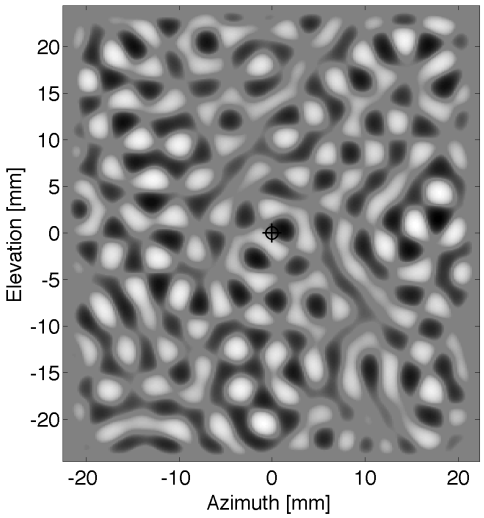
(c) 256×192 points, dx=dy=ds, *ablength* of 1 to 200 mm



(d) 256×192 points, dx=dy=ds, *ablength* of 20 to 200 mm



(e) 256×192 points, dx=ds_x, dy=ds_y, *ablength* of 4 to 2000 mm, *offset* of (-10,-8) mm



(f) 256×384 points, dx=ds_x, dy=ds_y/2, *ablength* of 3.9 to 4.1 mm

Figure 3.2: Six different delay screens with different sizes, sampling and length scales. A marker is set to mark the origin of the delay screen to show the effects of the `Propcontrol` field *offset*. The delay screen are visualized using a logarithmic compression to enhance contrast.

Chapter 4

Version and Installation Guide

4.1 Abersim 2.0 - Matlab version

Abersim 2.0 is a new, enhanced version of the old Matlab code used for version 1.X. Some functions are written in C, but they do not require linking to other libraries outside Matlab on UNIX based systems. Abersim and Matlab uses the same library "the Fastest Fourier Transform in the West" (FFTW) for FFTs. On UNIX systems, this is easily linked up to Abersim from the Matlab path. On Windows systems, this FFT library must be downloaded. For some newer Matlab implementations, errors have occurred when linking to FFTW through Matlab. If this happen, either fix the error or install FFTW on your machine.

To install Abersim 2.0 to your computer, please follow the intructions below.

If the computer does not have a C compiler at all, a fully Matlab implemented version is possible to use. For this option, unpack the files and start up Abersim with `startup_abersim`. If you *do* have a C compiler, please follow the instructions below.

4.1.1 Installation of FFTW

- Download the latest stable version of the FFTW library from *http://www.fftw.org*.
- On UNIX: Install this library using configure and make.
- On Windows: Download the latest dll-file and follow installation procedure as described at *http://www.fftw.org/install/windows.html*.
- Make sure the library files (`.so`/`.a` or `.dll`) are in the path of your computer.

4.1.2 Installation on UNIX systems

1. Unpack the files in the tar ball at the desired location
2. Start up Matlab
3. Run `mex -setup` if you want to specify a certain compiler or haven't done this before
4. Go to the Abersim 2.0 directory

5. Run the script `install_abersim`. As input arguments, the path of the FFTW library and mexfile may be specified. If the FFTW is installed somewhere else than in */usr/local/* this path must be specified as the first input argument. If you plan to use another *specific* mexopts file, another than the standard one setup up using `mex -setup`, the full filename of this may be specified as the second input argument.

4.1.3 Installation on Windows systems

If you do not have a C compiler you may download Visual Studio Express from the internet. The use of Matlab's `lcc` is strongly not recommended and should be avoided.

1. Unpack the files in the Abersim zip file at the desired location
2. Start up Matlab
3. Run `mex -setup` and specify the compiler you want to use
4. Open the newly created `mexopts.bat` and locate the variable `LINKFLAGS`
5. Add `/LIBPATH:"<fftw-lib-path>"` and `libfftw3-3.lib` to the end of the line
6. Go to the Abersim 2.0 directory
7. Run the script `install_abersim`. As input arguments, the path of the FFTW library and mexfile may be specified. The path to FFTW is ignored on Windows systems. If you plan to use another specific mexopts file, another than the standard one setup up using `mex -setup`, the full filename of this may be specified as the second input argument.

Abersim 2.0 should now be installed to you computer. When using Abersim 2.0 for simulations, please run the script `startup_abersim`. This will make sure that the folders of Abersim are in the Matlab path.

4.2 Abersim 2.0 - C version

The stand-alone C version is somewhat more troublesome to install, and requires some pre-installed libraries [3, 6, 8]. The C version is not dependent on Matlab in any way, but uses the Matlab file format `.mat` for everything that is stored to disk. This requires an own library. Make sure you have a C compiler installed on your computer. This guide is only for UNIX type systems, and there are no plans at the time of writing of making Windows installation guides for the stand-alone C version of Abersim 2.0. The required libraries needed by Abersim 2.0 are:

- CBLAS, LAPACK (ATLAS, *http://math-atlas.sourceforge.net/*)
- FFTW (*http://www.fftw.org/*)
- MATIO (*http://sourceforge.net/projects/matio*)

Install the above libraries following the procedure presented in their respective manuals and documentation. Abersim is compiled and built using Makefiles. The main Makefile will include a Makefile specific for your machine (`Makefile.in`) that contains information on where to find the libraries specified above etc.

4.2.1 Installation on UNIX systems

Make sure the Abersim 2.0 tar ball is unpacked. This is the case if have already installed the Matlab version. The source files for the C version is included in the subdirectory *cfiles/*.

The wrapper library `locblas` must be installed before building Abersim. This is installed through the following procedure:

1. Go to the *abersim-2.0/cfiles/locblas/* directory.
2. Modify `Makefile.YOURMACHINE`.
   - For Mac: Add `-D_USE_VECLIB` to the `LOCFLAGS` variable.
   - For Linux with Intel MKL: Add `-D_USE_MKL_CBLAS` to the `LOCFLAGS` variable.
   - For Linux with ATLAS: Add `-D_USE_ATLAS` to the `LOCFLAGS` variable.
   - Set the `LOCINCLS`, `LOCLDINCLS` to the `include` and `lib` folder of the BLAS distribution and `LOCLIBS` to the libraries that need to be linked in.
   - Set the `LOCDIR` variable to the path where the `locblas` library is to be installed.
3. Make a link from `Makefile.YOURMACHINE` to `Makefile.in`
4. Run `make` and `make install`

When the `locblas` library is built, the main program is built in a similar fashion. Make sure that all the libraries are in the run time path of your machine. This is specified in the shell variable `$LD_LIBRARY_PATH`.

Abersim 2.0 is installed throught the following procedure:

1. Go to the *abersim-2.0/cfiles/* directory.
2. Modify `Makefile.YOURMACHINE`.
   - For Mac: Add `-D_USE_VECLIB` to the `LOCALDEFINES` variable.
   - For Linux with Intel MKL: Add `-D_USE_MKL_CBLAS` to the `LOCALDEFINES` variable.
   - For Linux with ATLAS: Add `-D_USE_ATLAS` to the `LOCALDEFINES` variable.
   - Set the `MATIODIR`, `LOCBLASDIR` and `FFTWDIR` to the paths of the different libraries. The variable `MATLABDIR` may be specified if wanted
   - Set the `LOCALLIBS` variable to the proper libraries.
3. Make a link from `Makefile.YOURMACHINE` to `Makefile.in`
4. Run `make`

the C version of Abersim 2.0 should now be installed to you computer. When using this version, the simulations are run using initial data. These data should be exported from the Matlab version of Abersim 2.0 using `export_simulation`.

Chapter 5

License Agreement

Abersim is released under the GNU General Pulic License, version 3. The full license may be found below or downloaded from *http://www.gnu.org/copyleft/gpl* (Ref. [5]).

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license **for** software and other kinds of works.

The licenses **for** most software and other practical works are designed to take away your freedom to share and change the works. By **contrast**, the GNU General Public License is intended to guarantee your freedom to share and change **all** versions of a program—to make sure it remains free software **for all** its users. We, the Free Software Foundation, use the GNU General Public License **for** most of our software; it applies also to **any** other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge **for** them **if** you wish), that you receive source code or can **get** it **if** you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities **if** you distribute copies of the software, or **if** you modify it: responsibilities to respect the freedom of others.

For example, **if** you distribute copies of such a program, whether gratis or **for** a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can **get** the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty **for** this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products **for** individuals to use, **which** is precisely where it is most unacceptable. Therefore, we have designed this **version** of the GPL to prohibit the practice **for** those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions **for** copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to **version** 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to **any** copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt **all** or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified **version**" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable **for** infringement under applicable copyright law, except executing it on a **computer** or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means **any** kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a **computer** network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible

feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose

of having them make modifications exclusively **for** you, or provide you with facilities **for** running those works, provided that you comply with the terms of this License in conveying **all** material **for which** you do not control copyright. Those thus making or running the covered works **for** you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making **any** copies of your copyrighted material outside their relationship with you.

Conveying under **any** other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under **any** applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive **any** legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim **any** intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in **any** medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact **all** notices stating that this License and **any** non-permissive terms added in accord with section 7 apply to the code; keep intact **all** notices of the absence of **any** warranty; and give **all** recipients a copy of this License along with the Program.

You may charge **any** price or no price **for** each copy that you convey, and you may offer support or warranty protection **for** a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet **all** of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant **date**.

b) The work must carry prominent notices stating that it is released under this License and **any** conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact **all** notices".

c) You must license the entire work, as a whole, under this License to anyone **who** comes into possession of a copy. This License will therefore apply, along with **any** applicable section 7 additional terms, to the whole of the work, and **all** its parts, regardless of how they are packaged. This License gives no permission to license the work in **any** other way, but it does not invalidate such permission **if** you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no

charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove **any** additional permissions from that copy, or from **any** part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, **for which** you have or can give appropriate copyright permission.

Notwithstanding **any** other provision of this License, **for** material you add to a covered work, you may (**if** authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original **version**; or

d) Limiting the use **for** publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law **for** use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone **who** conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, **for** **any** liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or **any** part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to **find** the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including **any** patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are **all** patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor **version**, but do not include claims that would be infringed only as a consequence of further modification of the contributor **version**. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer **for** sale, import and otherwise run, modify and propagate the contents of its contributor **version**.

In the following three paragraphs, a "patent license" is **any** express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue **for** patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available **for** anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license **for** this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but **for** the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or **more** identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to **all** recipients of the covered work and works based on it.

A patent license is "discriminatory" **if** it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or **more** of the rights that are specifically granted under this License. You may not convey a covered work **if** you are a party to an arrangement with a third party that is in the business of distributing software, under **which** you make payment to the third party based on the extent of your activity of conveying the work, and under **which** the third party grants, to **any** of the parties **who** would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily **for** and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting **any** implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and **any** other pertinent obligations, then as a consequence you may not convey it at **all**. For example, **if** you agree to terms that obligate you to collect a royalty **for** further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding **any** other provision of this License, you have permission to link or combine **any** covered work with a work licensed under **version** 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part **which** is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present **version**, but may differ in detail to address new problems or concerns.

Each **version** is given a distinguishing **version** number. If the Program specifies that a certain numbered **version** of the GNU General Public License "or **any** later **version**" applies to it, you have the option of following the terms and conditions either of that numbered **version** or of **any** later **version** published by the Free Software Foundation. If the Program does not specify a **version** number of the GNU General Public License, you may choose **any version** ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide **which** future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a **version** permanently authorizes you to choose that **version for** the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on **any** author or copyright holder as a result of your choosing to follow a later **version**.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of **all** civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in **return for** a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software **which** everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" **line** and a pointer to where the **full** notice is found.

> <one **line** to give the program's name and a brief idea of **what** it does.>
> Copyright (C) <year> <name of author>
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either **version** 3 of the License, or (at your option) **any** later **version**.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License **for more** details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does **terminal** interaction, make it output a short notice like this when it starts in an interactive mode:

> <program> Copyright (C) <year> <name of author>
> This program comes with ABSOLUTELY NO WARRANTY; **for** details **type** 'show w'.
> This is free software, and you are welcome to redistribute it
> under certain conditions; **type** 'show c' **for** details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; **for** a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

Bibliography

[1] Bjørn A. J. Angelsen. *Ultrasound Imaging*, volume 1. Emantec, Trondheim, Norway, 2000.

[2] Bjørn A. J. Angelsen. *Ultrasound Imaging*, volume 2. Emantec, Trondheim, Norway, 2000.

[3] Christopher Hulbert. *MAT File I/O Library.* SourceForge, http://sourceforge.net/projects/matio.

[4] Francis A. Duck. *Physical Properties of Tissue.* Academic Press, London, 1990.

[5] Free Software Foundation, http://www.gnu.org/copyleft/gpl. *GNU General Public License.*

[6] Matteo Frigo and Steven G. Johnson. *FFTW 3.1.3.* Massachussetts Institute of Technology, www.fftw.org, 2003.

[7] Mark F. Hamilton and David T. Blackstock. *Nonlinear Acoustics.* Academic Press, San Diego, 1997.

[8] University of Tennessee, Knoxville, Tennessee, http://math-atlas.sourceforge.net/. *Automatically Tuned Linear Algebra Software*, 2008.

[9] Trond Varslot and Svein-Erik Måsøy. Forward Propagation of Acoustic Pressure Pulses in 3D Soft Biological Tissue. *Modeling, Identification and Control*, 27(3):181–200, 2006.

[10] Trond Varslot and Gunnar Taraldsen. Computer Simulation of Forward Propagation in Soft Tissue. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 52:1473–82, 2005.