SFS Toolbox - Matlab Documentation

Release <unknown>

SFS Toolbox Developers

Sep 18, 2017

Contents

CHAPTER 1

Sound Field Synthesis Toolbox for Matlab

The SFS Toolbox for Matlab gives you the possibility to play around with sound field synthesis methods like wave field synthesis (WFS) or near-field compensated higher order Ambisonics (NFC-HOA). There are functions to simulate monochromatic sound fields for different secondary source (loudspeaker) setups, time snapshots of full band impulses emitted by the secondary source distributions, or even generate binaural room scanning (BRS) impulse response sets in order to generate binaural simulations of the synthesized sound fields with the SoundScape Renderer.

Theory: http://sfstoolbox.org/

Documentation: http://matlab.sfstoolbox.org/

Source code and issue tracker: http://github.com/sfstoolbox/sfs-matlab/

SFS Toolbox for Python: http://python.sfstoolbox.org/

License: MIT – see the file `LICENSE` for details.

Installation

Download the Toolbox, go to the main path of the Toolbox and start it with `SFS_start` which will add all needed paths to Matlab/Octave. If you want to remove them again, run `SFS_stop`.

Requirements

Matlab: You need Matlab version R2011b or newer to run the Toolbox. On older versions the Toolbox should also work, but you need to add narginchk.m to the `SFS_helper` directory.

Octave: You need Octave version 3.6 or newer to run the Toolbox. In addition, you will need the `audio` and `signal` packages from octave-forge.

audioread: If `audioread()` is not available in your Matlab or Octave version, you can replace it by `wavread()`. It is used in the two functions `auralize_ir()` and `compensate_headphone()`.

Impulse responses: The Toolbox uses the SOFA file format for handling impulse response data sets like HRTFs. If you want to use this functionality you also have to install the SOFA API for Matlab/Octave, which you can add to your paths by executing `SOFAstart`.

Backward compatibility: Since version 2.0.0 the SFS Toolbox incorporates SOFA as file format for HRTFs which replaces the old irs file format formerly used by the Toolbox. If you still need this you should download the latest version with irs file support.

Getting started

In order to make a simulation of the sound field of a monochromatic point source with a frequency of 800 Hz placed at (0,2.5,0) m synthesized by WFS run

```
conf = SFS_config;
conf.plot.normalisation = 'center';
sound_field_mono_wfs([-2 2],[-2 2],0,[0 2.5 0],'ps',800,conf)
```

To make a simulation of the same point source - now producing a broadband impulse - in the time domain at a time of 5 ms after the first loudspeaker activity run

```
conf = SFS_config;
conf.plot.normalisation = 'max';
sound_field_imp_wfs([-2 2],[-2 2],0,[0 2.5 0],'ps',0.005,conf)
```

After that have a look at `SFS_config.m` for the default settings of the Toolbox. Please don't change the settings directly in `SFS_config.m`, but create an extra function or script for this, that can look like this:

```
conf = SFS_config;
conf.fs = 48000;
```

For a detailed description of all available features the SFS Toolbox, have a look at the online documentation.

Credits and feedback

If you have questions, bug reports or feature requests, please use the Issue Section to report them.

If you use the SFS Toolbox for your publications please cite our AES Convention e-Brief and the DOI for the used Toolbox version, you will find at the official releases page:

H. Wierstorf, S. Spors - Sound Field Synthesis Toolbox. In the Proceedings of *132nd Convention of the Audio Engineering Society*, 2012 [pdf] [bibtex]

CHAPTER 2

Secondary Sources

The Toolbox comes with a function which can generate different common shapes of loudspeaker arrays for you. At the moment linear, circular, box shaped and spherical arrays are supported.

Before showing the different geometries, we start with some common settings. First we get a configuration struct and set the array size/diameter to 3 m.

```
conf = SFS_config;
conf.secondary_sources.size = 3;
```

Linear array

```
conf = SFS_config;
conf.secondary_sources.geometry = 'line'; % or 'linear'
conf.secondary_sources.number = 21;
x0 = secondary_source_positions(conf);
figure;
figsize(conf.plot.size(1),conf.plot.size(2),conf.plot.size_unit);
draw_loudspeakers(x0,conf);
axis([-2 2 -2 1]);
%print_png('img/secondary_sources_linear.png');
```

Circular array

```
conf = SFS_config;
conf.secondary_sources.geometry = 'circle'; % or 'circular'
conf.secondary_sources.number = 56;
x0 = secondary_source_positions(conf);
figure;
figsize(540,404,'px');
```

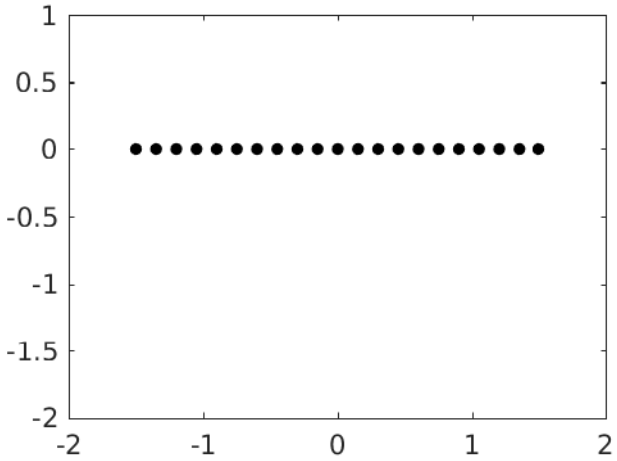

Fig. 2.1: Linear loudspeaker array with a length of 3m consiting of 21 loudspeakers.

```
draw_loudspeakers(x0,conf);
axis([-2 2 -2 2]);
%print_png('img/secondary_sources_circle.png');
```

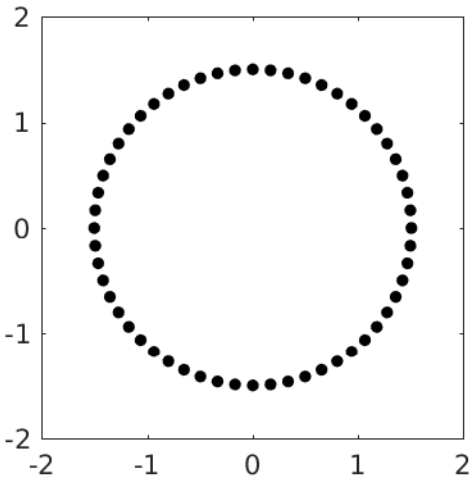

Fig. 2.2: Circular loudspeaker array with a diameter of 3m consiting of 56 loudspeakers.

Box shaped array

```
conf = SFS_config;
conf.secondary_sources.geometry = 'box';
conf.secondary_sources.number = 84;
x0 = secondary_source_positions(conf);
figure;
figsize(540,404,'px');
```

```
draw_loudspeakers(x0,conf);
axis([-2 2 -2 2]);
%print_png('img/secondary_sources_box.png');
```

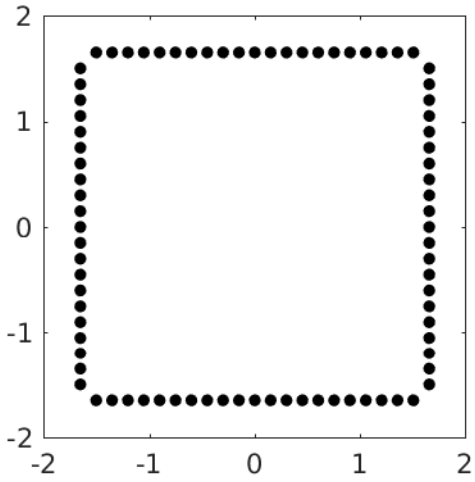

Fig. 2.3: Box shaped loudspeaker array with a diameter of 3m consisting of 84 loudspeakers.

Box shaped array with rounded edges

`conf.secondary_sources.edge_radius` defines the bending radius of the corners. It can be chosen in a range between `0.0` and the half of `conf.secondary_sources.size`. While the prior represents a square box the latter yields a circle. Note that the square box behaves it little bit different than the Box Shaped Array since loudspeakers might also be place directly in the corners of the box.

```
conf = SFS_config;
conf.secondary_sources.geometry = 'rounded-box';
conf.secondary_sources.number = 84;
conf.secondary_sources.corner_radius = 0.3;
x0 = secondary_source_positions(conf);
figure;
figsize(540,404,'px');
draw_loudspeakers(x0,conf);
axis([-2 2 -2 2]);
%print_png('img/secondary_sources_rounded-box.png');
```

Spherical array

For a spherical array you need a grid to place the secondary sources on the sphere. At the moment we provide grids with the Toolbox, that can be found in the corresponding folder of the data repository. You have to specify your desired grid, for example `conf.secondary_sources.grid = 'equally_spaced_points'`. The `secondary_source_positions()` functions will then automatically download the desired grid from that web page and stores it under `<$SFS_MAIN_PATH>/data`. If the download is not working (which can happen especially under Matlab and Windows) you can alternatively checkout or download the whole data repository to the data folder.

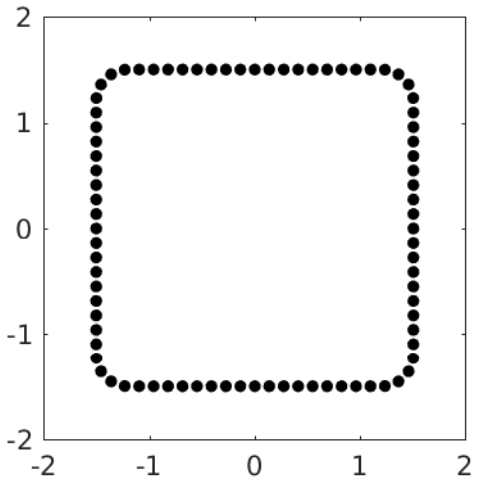

Fig. 2.4: Box shaped loudspeaker array with rounded edges. It has again a diameter of 3m, consists of 84 loudspeakers and has a edge bending factor of 0.3.

```
conf = SFS_config;
conf.secondary_sources.size = 3;
conf.secondary_sources.geometry = 'sphere'; % or 'spherical'
conf.secondary_sources.grid = 'equally_spaced_points';
conf.secondary_sources.number = 225;
x0 = secondary_source_positions(conf);
figure;
figsize(540,404,'px');
draw_loudspeakers(x0,conf);
axis([-2 2 -2 2]);
%print_png('img/secondary_sources_sphere.png');
```

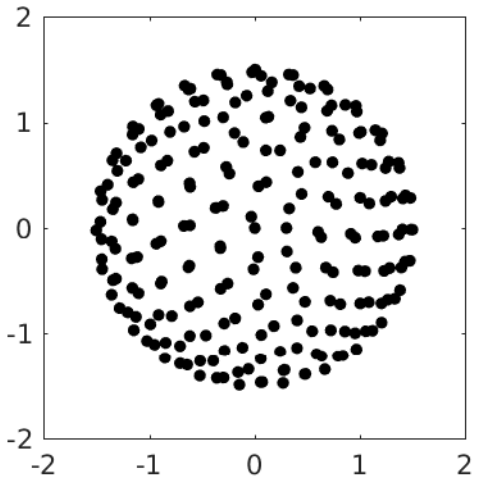

Fig. 2.5: Spherical loudspeaker array with a diameter of 3m consiting of 225 loudspeakers arranged on a grid with equally spaced points.

Arbitrary shaped arrays

You can create arbitrarily shaped arrays by setting `conf.secondary_sources.geometry` to `'custom'` and define the values of the single loudspeaker directly in the `conf.secondary_sources.x0` matrix. The rows of the matrix contain the single loudspeakers and the six columns are `[x y z nx ny nz w]`, the position and direction and weight of the single loudspeakers. The weight `w` is a factor the driving function of this particular loudspeaker is multiplied with in a function that calculates the sound field from the given driving signals and secondary sources. For WFS (Wave Field Synthesis) `w` could include the tapering window, a spherical grid weight, and the $r^2 \cos(\theta)$ weights for integration on a sphere.

```
conf = SFS_config;
% create a stadium like shape by combining two half circles with two linear
% arrays
% first getting a full circle with 56 loudspeakers
conf.secondary_sources.geometry = 'circle';
conf.secondary_sources.number = 56;
conf.secondary_sources.x0 = [];
x0 = secondary_source_positions(conf);
% store the first half cricle and move it up
x01 = x0(2:28,:);
x01(:,2) = x01(:,2) + ones(size(x01,1),1)*0.5;
% store the second half circle and move it down
x03 = x0(30:56,:);
x03(:,2) = x03(:,2) - ones(size(x03,1),1)*0.5;
% create a linear array
conf.secondary_sources.geometry = 'line';
conf.secondary_sources.number = 7;
conf.secondary_sources.size = 1;
x0 = secondary_source_positions(conf);
% rotate it and move it left
R = rotation_matrix(pi/2);
x02 = [(R*x0(:,1:3)')' (R*x0(:,4:6)')'];
x02(:,1) = x02(:,1) - ones(size(x0,1),1)*1.5;
x02(:,7) = x0(:,7);
% rotate it the other way around and move it right
R = rotation_matrix(-pi/2);
x04 = [(R*x0(:,1:3)')' (R*x0(:,4:6)')'];
x04(:,1) = x04(:,1) + ones(size(x0,1),1)*1.5;
x04(:,7) = x0(:,7);
% combine everything
conf.secondary_sources.geometry = 'custom';
conf.secondary_sources.x0 = [x01; x02; x03; x04];
% if we gave the conf.x0 to the secondary_source_positions function it will
% simply return the defined x0 matrix
x0 = secondary_source_positions(conf);
figure;
figsize(540,404,'px');
draw_loudspeakers(x0,conf);
axis([-2 2 -2.5 2.5]);
%print_png('img/secondary_sources_arbitrary.png');
```

Plot loudspeaker symbols

For two dimensional setups you can plot the secondary sources with loudspeaker symbols, for example the following will replot the last array.

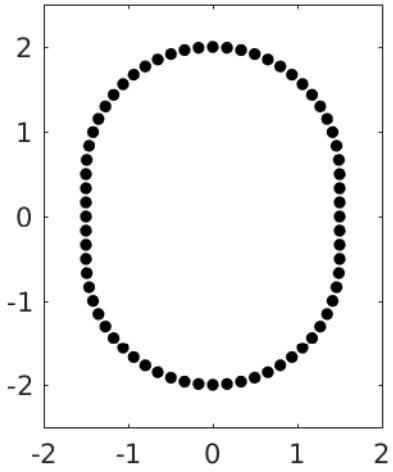

Fig. 2.6: Custom arena shaped loudspeaker array consiting of 70 loudspeakers.

```
conf.plot.realloudspeakers = true;
figure;
figsize(540,404,'px');
draw_loudspeakers(x0,conf);
axis([-2 2 -2.5 2.5]);
%print_png('img/secondary_sources_arbitrary_realloudspeakers.png');
```

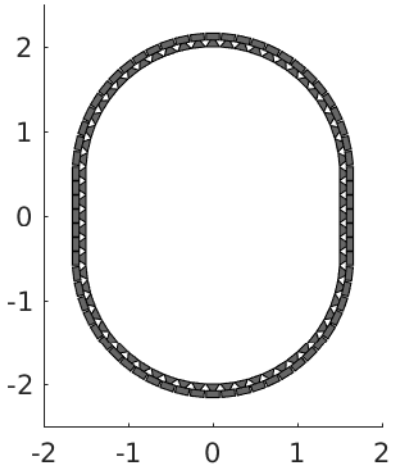

Fig. 2.7: Custom arena shaped loudspeaker array consiting of 70 loudspeakers, plotted using loudspeaker symbols instead of circles for the single loudspeakers.

CHAPTER 3

Frequency Domain

With the files in the folder `SFS_monochromatic` you can simulate a monochromatic sound field in a specified area for different techniques like WFS and NFC-HOA (Near-Field Compensated Higher Order Ambisonics). The area can be a 3D cube, a 2D plane, a line or only one point. This depends on the specification of `X,Y,Z`. For example `[-2 2],[-2 2],[-2 2]` will be a 3D cube; `[-2 2],0,[-2 2]` the xz-plane; `[-2 2],0,0` a line along the x-axis; `3,2,1` a single point. If you present a range like `[-2 2]` the Toolbox will create automatically a regular grid from this ranging from -2 to 2 with `conf.resolution` steps in between. Alternatively you could apply a *custom grid* by providing a matrix instead of the `[min max]` range for all active axes.

For all 2.5D functions the configuration `conf.xref` is important as it defines the point for which the amplitude is corrected in the sound field. The default entry is

```
conf.xref = [0 0 0];
```

Wave Field Synthesis

The following will simulate the field of a virtual plane wave with a frequency of 800 Hz going into the direction of (0 -1 0) synthesized with 3D WFS.

```
conf = SFS_config;
conf.dimension = '3D';
conf.secondary_sources.size = 3;
conf.secondary_sources.number = 225;
conf.secondary_sources.geometry = 'sphere';
% [P,x,y,z,x0,win] = sound_field_mono_wfs(X,Y,Z,xs,src,f,conf);
sound_field_mono_wfs([-2 2],[-2 2],0,[0 -1 0],'pw',800,conf);
%print_png('img/sound_field_wfs_3d_xy.png');
sound_field_mono_wfs([-2 2],0,[-2 2],[0 -1 0],'pw',800,conf);
%print_png('img/sound_field_wfs_3d_xz.png');
sound_field_mono_wfs(0,[-2 2],[-2 2],[0 -1 0],'pw',800,conf);
%print_png('img/sound_field_wfs_3d_yz.png');
```

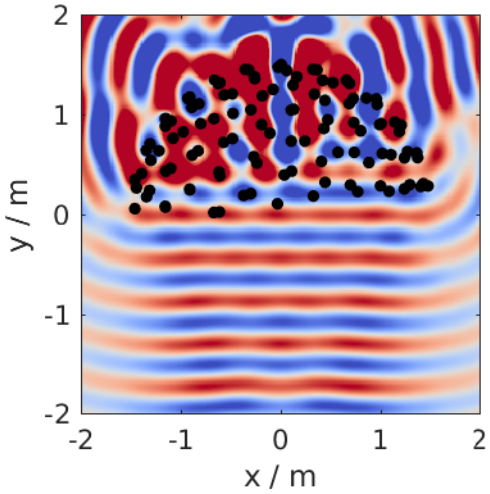

Fig. 3.1: Sound pressure of a mono-chromatic plane wave synthesized by 3D WFS. The plane wave has a frequency of 800Hz and is travelling into the direction (0,-1,0). The plot shows the xy-plane.

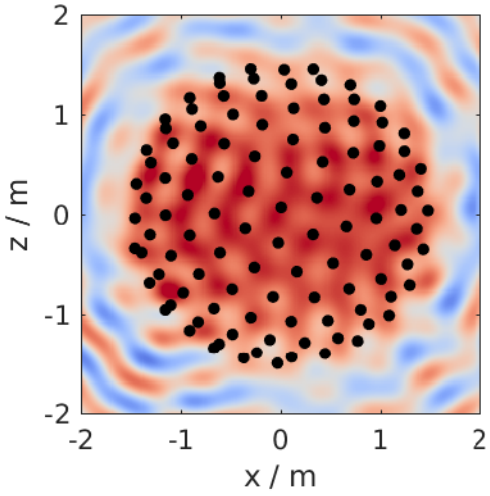

Fig. 3.2: The same as in the figure before, but now showing the xz-plane.

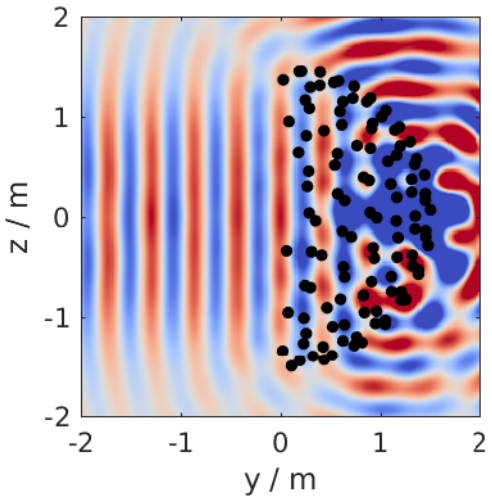

Fig. 3.3: The same as in the figure before, but now showing the yz-plane.

You can see that the Toolbox is now projecting all the secondary source positions into the plane for plotting them. In addition the axis are automatically chosen and labeled.

It is also possible to simulate and plot the whole 3D cube, but in this case no secondary sources will be added to the plot.

```
conf = SFS_config;
conf.dimension = '3D';
conf.secondary_sources.size = 3;
conf.secondary_sources.number = 225;
conf.secondary_sources.geometry = 'sphere';
conf.resolution = 100;
sound_field_mono_wfs([-2 2],[-2 2],[-2 2],[0 -1 0],'pw',800,conf);
%print_png('img/sound_field_wfs_3d_xyz.png');
```

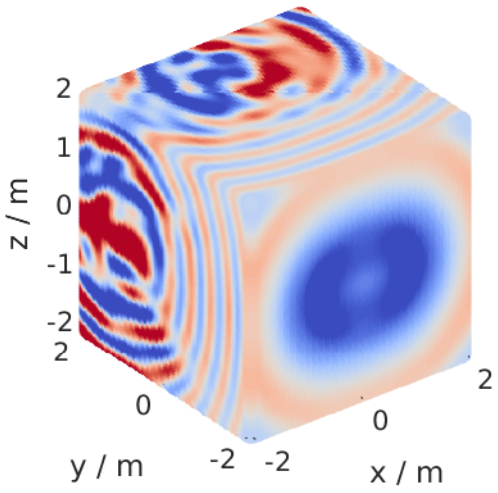

Fig. 3.4: Sound pressure of a mono-chromatic plane wave synthesized by 3D WFS. The plane wave has a frequency of 800Hz and is travelling into the direction (0,-1,0). All three dimensions are shown.

In the next plot we use a two dimensional array, 2.5D WFS and a virtual point source located at (0 2.5 0) m. The 3D example showed you, that the sound fields are automatically plotted if we specify now output arguments. If we specify one, we have to explicitly say if we want also plot the results, by `conf.plot.useplot = true;`.

```
conf = SFS_config;
conf.dimension = '2.5D';
conf.plot.useplot = true;
conf.plot.normalisation = 'center';
% [P,x,y,z,x0] = sound_field_mono_wfs(X,Y,Z,xs,src,f,conf);
[P,x,y,z,x0] = sound_field_mono_wfs([-2 2],[-2 2],0,[0 2.5 0],'ps',800,conf);
%print_png('img/sound_field_wfs_25d.png');
```

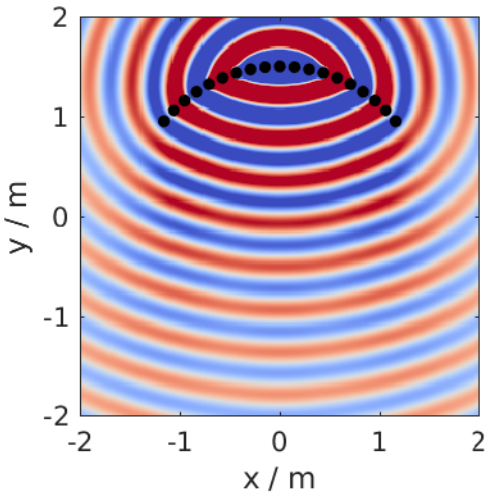

Fig. 3.5: Sound pressure of a mono-chromatic point source synthesized by 2.5D WFS. The point source has a frequency of 800Hz and is placed at (0 2.5 0)m. Only the active loudspeakers of the array are plotted.

If you want to plot the whole loudspeaker array and not only the active secondary sources, you can do this by adding these commands. First we store all sources in an extra variable `x0_all`, then we get the active ones `x0` and the corresponding indices of these active ones in `x0_all`. Afterwards we set all sources in `x0_all` to zero, which are inactive and only the active ones to the loudspeaker weights `x0(:,7)`.

```
x0_all = secondary_source_positions(conf);
[~,idx] = secondary_source_selection(x0_all,[0 2.5 0],'ps');
x0_all(:,7) = zeros(1,size(x0_all,1));
x0_all(idx,7) = x0(:,7);
plot_sound_field(P,[-2 2],[-2 2],0,x0_all,conf);
%print_png('img/sound_field_wfs_25d_with_all_sources.png');
```

Near-Field Compensated Higher Order Ambisonics

In the following we will simulate the field of a virtual plane wave with a frequency of 800 Hz traveling into the direction (0 -1 0), synthesized with 2.5D NFC-HOA.

```
conf = SFS_config;
conf.dimension = '2.5D';
% sound_field_mono_nfchoa(X,Y,Z,xs,src,f,conf);
```

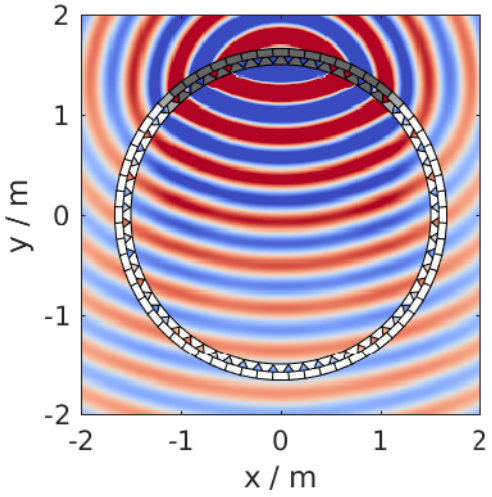

Fig. 3.6: Sound pressure of a mono-chromatic point source synthesized by 2.5D WFS. The point source has a frequency of 800Hz and is placed at (0 2.5 0)m. All loudspeakers are plotted. Their color correspond to the loudspeaker weights, where white stands for zero.

```
sound_field_mono_nfchoa([-2 2],[-2 2],0,[0 -1 0],'pw',800,conf);
%print_png('img/sound_field_nfchoa_25d.png');
```

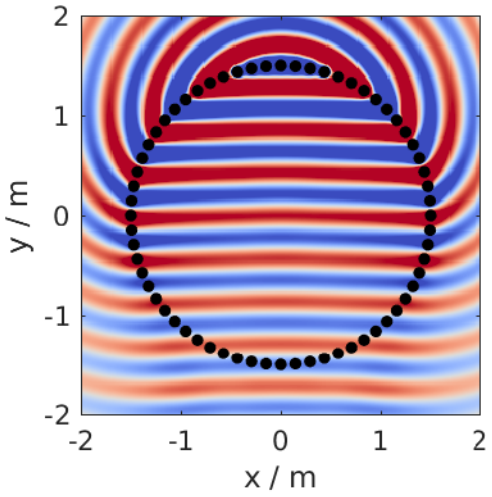

Fig. 3.7: Sound pressure of a monochromatic plane wave synthesized by 2.5D NFC-HOA. The plane wave has a frequency of 800 Hz and is traveling into the direction (0,-1,0).

Local Wave Field Synthesis

In NFC-HOA the aliasing frequency in a small region inside the listening area can be increased by limiting the used order. A similar outcome can be achieved in WFS by applying so called local Wave Field Synthesis. In this case the original loudspeaker array is driven by WFS to create a virtual loudspeaker array consisting of focused sources which can then be used to create the desired sound field in a small area. The settings are the same as for WFS, but a new

struct `conf.localwfs` has to be filled out, which for example provides the settings for the desired position and form of the local region with higher aliasing frequency, have a look into `SFS_config.m` for all possible settings.

```
X = [-1 1];
Y = [-1 1];
Z = 0;
xs = [1 -1 0];
src = 'pw';
f = 7000;
conf = SFS_config;
conf.resolution = 1000;
conf.dimension = '2D';
conf.secondary_sources.geometry = 'box';
conf.secondary_sources.number = 4*56;
conf.secondary_sources.size = 2;
conf.localwfs_vss.size = 0.4;
conf.localwfs_vss.center = [0 0 0];
conf.localwfs_vss.geometry = 'circular';
conf.localwfs_vss.number = 56;
sound_field_mono_localwfs_vss(X,Y,Z,xs,src,f,conf);
axis([-1.1 1.1 -1.1 1.1]);
%print_png('img/sound_field_localwfs_2d.png');
```

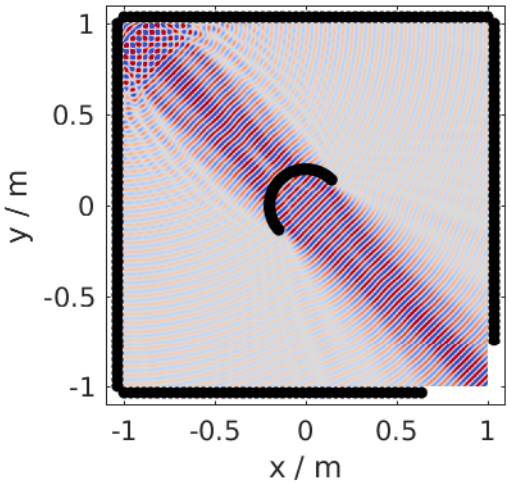

Fig. 3.8: Sound pressure of a monochromatic plane wave synthesized by 2D local WFS. The plane wave has a frequency of 7000 Hz and is traveling into the direction (1,-1,0). The local WFS is created by using focused sources to create a virtual circular loudspeaker array in he center of the actual loudspeaker array.

Stereo

The Toolbox includes not only WFS and NFC-HOA, but also some generic sound field functions that are doing only the integration of the driving signals of the single secondary sources to the resulting sound field. With these function you can for example easily simulate a stereophonic setup. In this example we set the `conf.plot.normalisation = 'center';` configuration manually as the amplitude of the sound field is too low for the default `'auto'` setting to work.

```
conf = SFS_config;
conf.plot.normalisation = 'center';
x0 = [-1 2 0 0 -1 0 1;1 2 0 0 -1 0 1];
% [P,x,y,z] = sound_field_mono(X,Y,Z,x0,src,D,f,conf)
sound_field_mono([-2 2],[-1 3],0,x0,'ps',[1 1],800,conf)
%print_png('img/sound_field_stereo.png');
```

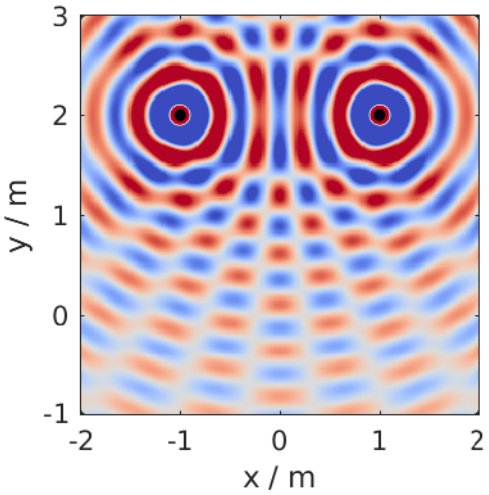

Fig. 3.9: Sound pressure of a monochromatic phantom source generated by stereophony. The phantom source has a frequency of 800 Hz and is placed at (0,2,0) by amplitude panning.

CHAPTER 4

Time Domain

With the files in the folder `SFS_time_domain` you can simulate snapshots in time of an impulse originating from your WFS or NFC-HOA system.

In the following we will create a snapshot in time after 5 ms for a broadband virtual point source placed at (0 2 0) m for 2.5D NFC-HOA.

```
conf = SFS_config;
conf.dimension = '2.5D';
conf.plot.useplot = true;
% sound_field_imp_nfchoa(X,Y,Z,xs,src,t,conf)
[p,x,y,z,x0] = sound_field_imp_nfchoa([-2 2],[-2 2],0,[0 2 0],'ps',0.005,conf);
%print_png('img/sound_field_imp_nfchoa_25d.png');
```

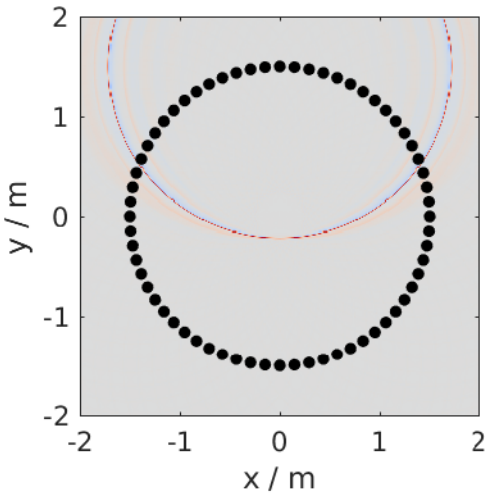

Fig. 4.1: Sound pressure of a broadband impulse point source synthesized by 2.5D NFC-HOA. The point source is placed at (0,2,0) m and the time snapshot is shown 5 ms after the first secondary source was active.

The output can also be plotted in dB by setting `conf.plot.usedb = true;`. In this case the default color map is changed and a color bar is plotted in the figure. For none dB plots no color bar is shown in the plots. In these cases the color coding goes always from -1 to 1, with clipping of larger values.

```
conf.plot.usedb = true;
plot_sound_field(p,[-2 2],[-2 2],0,x0,conf);
%print_png('img/sound_field_imp_nfchoa_25d_dB.png');
```

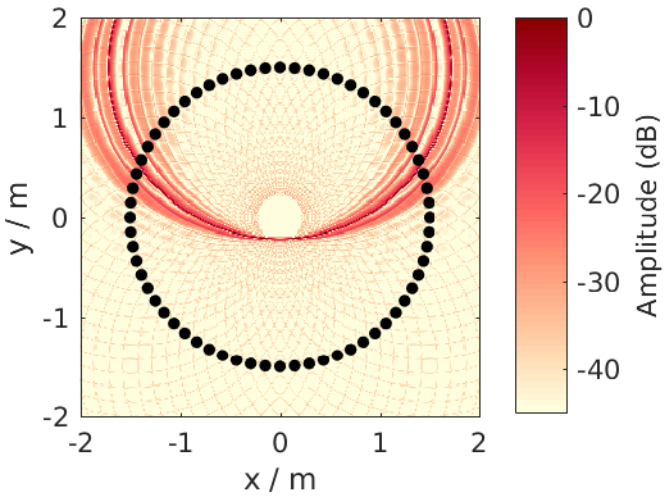

Fig. 4.2: Sound pressure in decibel of the same broadband impulse point source as in the figure above.

You could change the color map yourself doing the following before the plot command.

```
conf.plot.colormap = 'jet'; % Matlab rainbow color map
```

If you want to simulate more than one virtual source, it is a good idea to set the starting time of your simulation to start with the activity of your virtual source and not with the secondary sources, which is the default behavior. You can change this by setting `conf.t0 = 'source'`.

```
conf.plot.useplot = false;
conf.t0 = 'source';
t_40cm = 0.4/conf.c; % time to travel 40 cm in s
t0 = 0.0005; % start time of focused source in s
[p_ps,~,~,~,x0_ps] = ...
    sound_field_imp_wfs([-2 2],[-2 2],0,[1.9 0 0],'ps',t0+t_40cm,conf);
[p_pw,~,~,~,x0_pw] = ...
    sound_field_imp_wfs([-2 2],[-2 2],0,[1 -2 0],'pw',t0-t_40cm,conf);
[p_fs,~,~,~,x0_fs] = ...
    sound_field_imp_wfs([-2 2],[-2 2],0,[0 -1 0 0 1 0],'fs',t0,conf);
plot_sound_field(p_ps+p_pw+p_fs,[-2 2],[-2 2],0,[x0_ps; x0_pw; x0_fs],conf)
hold;
scatter(0,0,'kx');   % origin of plane wave
scatter(1.9,0,'ko'); % point source
scatter(0,-1,'ko');  % focused source
hold off;
%print_png('sound_field_imp_multiple_sources_dB.png');
```

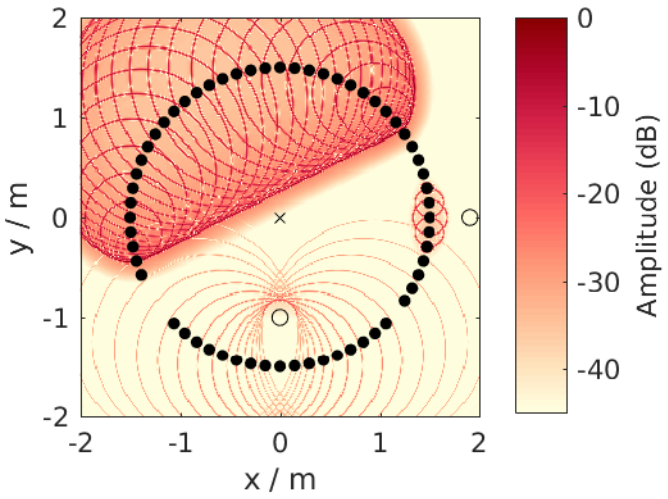

Fig. 4.3: Sound pressure in decibel of a boradband impulse plane wave, point source, and focused source synthesized all by 2.5D WFS. The plane wave is traveling into the direction (1,-2,0) and shown 0.7 ms before it starting point at (0,0,0). The point source is placed at (1.9,0,0) m and shown 1.7 ms after its start. The focused source is placed at (0,-1,0) m and shown 0.5 ms after its start.

CHAPTER 5

Modal Windows

In the time-domain implementation of NFC-HOA it is possible to apply different weighting factors to each mode of the driving signals in the circular harmonics domain by changing the `conf.modal_window` configuration parameter. Using a weighting window that is smoother than the default rectangular window, causes a concentration of energy to fewer loudspeakers.

```
conf = SFS_config;
conf.dimension = '2.5D';
conf.secondary_sources.number = 16;
conf.secondary_sources.geometry = 'circular';
conf.secondary_sources.size = 3;
conf.resolution = 300;
conf.plot.usedb = true;
conf.t0 = 'source';
X = [-2,2];
Y = [-2,2];
Z = 0;
conf.modal_window = 'rect';  % default
sound_field_imp_nfchoa(X,Y,Z,[0 -1 0],'pw',0,conf);
%print_png('sound_field_imp_nfchoa_25d_dB_rect.png');
conf.modal_window = 'max-rE';
sound_field_imp_nfchoa(X,Y,Z,[0 -1 0],'pw',0,conf);
%print_png('sound_field_imp_nfchoa_25d_dB_max-rE.png');
conf.modal_window = 'kaiser';
conf.modal_window_parameter = 1.0;
sound_field_imp_nfchoa(X,Y,Z,[0 -1 0],'pw',0,conf);
%print_png('sound_field_imp_nfchoa_25d_dB_kaiser.png');
conf.modal_window = 'tukey';
conf.modal_window_parameter = 0.5;
sound_field_imp_nfchoa(X,Y,Z,[0 -1 0],'pw',0,conf);
%print_png('sound_field_imp_nfchoa_25d_dB_tukey.png');
```

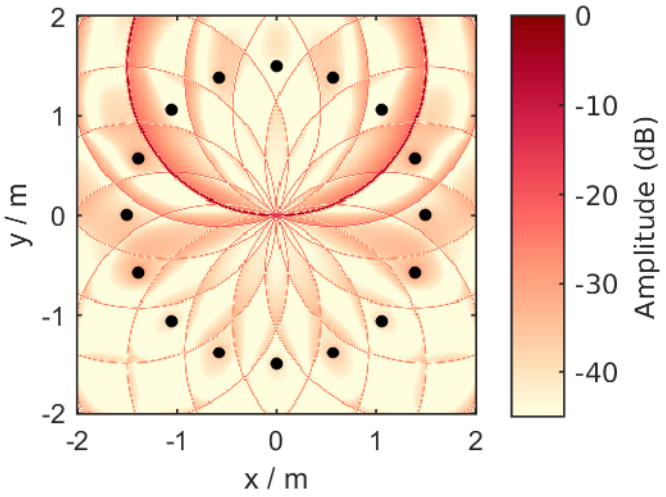

Fig. 5.1: Sound pressure in decibel of a broadband impulse plane wave synthesized by 2.5D NFC-HOA using a rectangular window. The plane wave propagates into the direction of (0,-1,0).

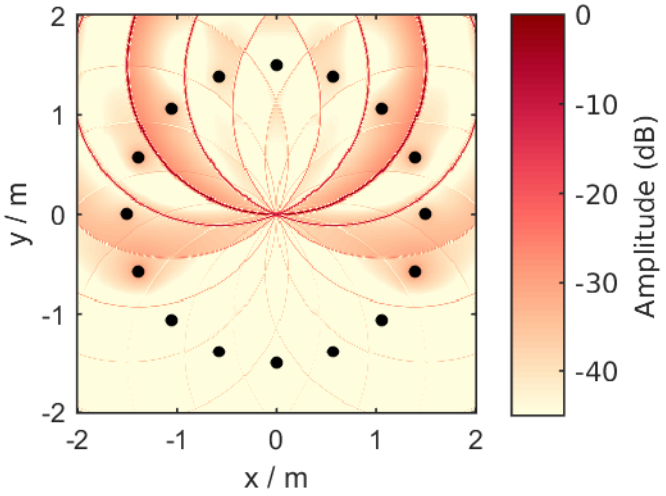

Fig. 5.2: Sound pressure in decibel of a broadband impulse plane wave synthesized by 2.5D NFC-HOA using a max-rE window. The plane wave propagates into the direction of (0,-1,0).

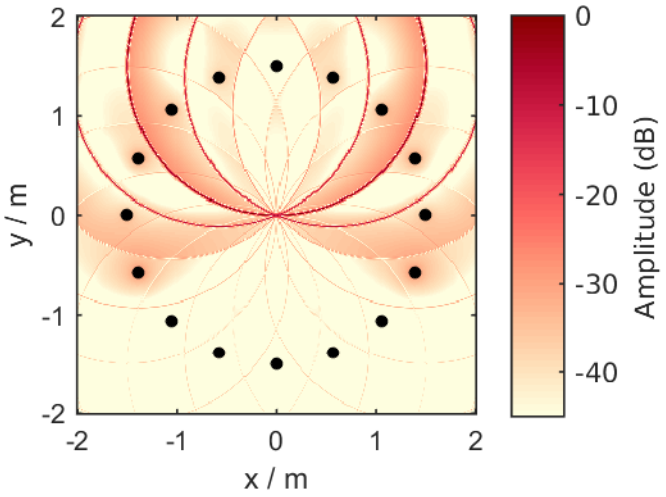

Fig. 5.3: Sound pressure in decibel of a broadband impulse plane wave synthesized by 2.5D NFC-HOA using a Kaiser window. The plane wave propagates into the direction of (0,-1,0).

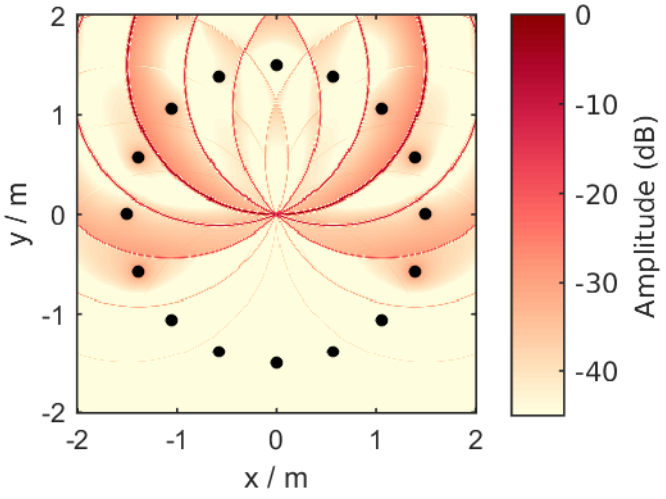

Fig. 5.4: Sound pressure in decibel of a broadband impulse plane wave synthesized by 2.5D NFC-HOA using a modified Tukey window. The plane wave propagates into the direction of (0,-1,0).

CHAPTER 6

Custom Grids

As stated earlier you can provide the sound field simulation functions a custom grid instead of the `[min max]` ranges. Again, you can provide it for one dimension, two dimensions, or all three dimensions.

```
conf = SFS_config;
conf.dimension = '3D';
conf.secondary_sources.number = 225;
conf.secondary_sources.geometry = 'sphere';
conf.resolution = 100;
conf.plot.normalisation = 'center';
X = randi([-2000 2000],125000,1)/1000;
Y = randi([-2000 2000],125000,1)/1000;
Z = randi([-2000 2000],125000,1)/1000;
sound_field_mono_wfs(X,Y,Z,[0 -1 0],'pw',800,conf);
%print_png('img/sound_field_wfs_3d_xyz_custom_grid.png');
conf.plot.usedb = true;
conf.dimension = '2.5D';
conf.secondary_sources.number = 64;
conf.secondary_sources.geometry = 'circle';
sound_field_imp_nfchoa(X,Y,0,[0 2 0],'ps',0.005,conf);
%print_png('img/sound_field_imp_nfchoa_25d_dB_custom_grid.png');
```

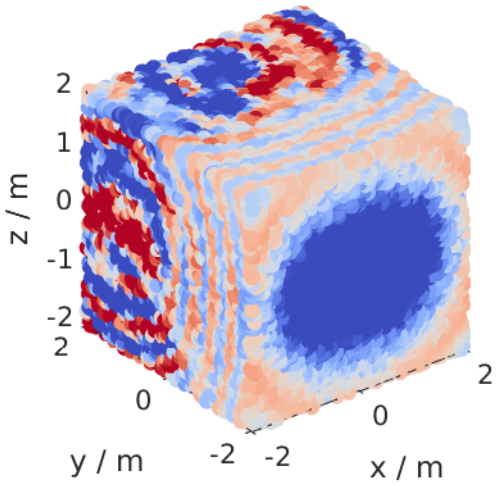

Fig. 6.1: Sound pressure of a monochromatic point source synthesized by 3D WFS. The plane wave has a frequency of 800 Hz and is travelling into the direction (0,-1,0). The sound pressure is calculated only at the explicitly provided grid points.

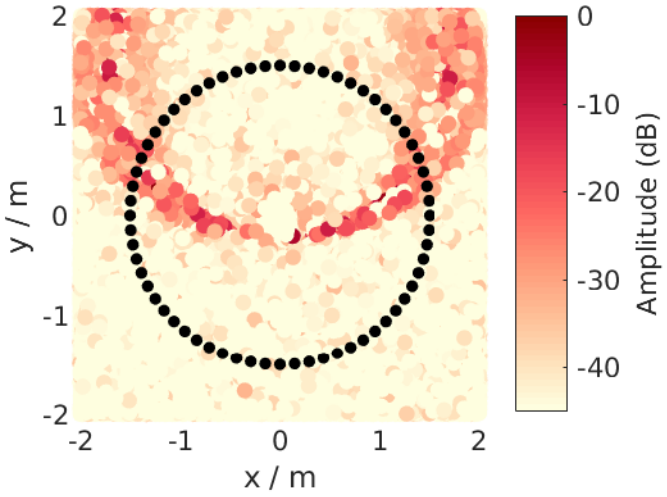

Fig. 6.2: Sound pressure in decibel of a broadband impulse point source synthesized by 2.5D NFC-HOA. The point source is placed at (0,2,0) m and a time snapshot after 5 ms of the first active secondary source is shown. The sound pressure is calculated only at the explicitly provided grid points.

CHAPTER 7

Binaural Simulations

If you have a set of HRTF (Head-Related Transfer Function)s or BRIR (Binaural Room Impulse Response)s you can simulate the ear signals reaching a listener sitting at a given point in the listening area for different spatial audio systems.

In order to easily use different HRTF or BRIR sets the Toolbox uses the SOFA file format. In order to use it you have to install the SOFA API for Matlab/Octave and run `SOFAstart` before you can use it inside the SFS Toolbox. If you are looking for different HRTFs and BRIRs, a large set of different impulse responses is available: http://www.sofaconventions.org/mediawiki/index.php/Files.

The files dealing with the binaural simulations are in the folder `SFS_binaural_synthesis`. Files dealing with HRTFs and BRIRs are in the folder `SFS_ir`. If you want to extrapolate your HRTFs to plane waves you may also want to have a look in the folder `SFS_HRTF_extrapolation`.

In the following we present some examples of binaural simulations. For their auralization an anechoic recording of a cello is used, which can be downloaded from anechoic_cello.wav.

Binaural simulation of arbitrary loudspeaker arrays

If you use an HRTF data set, it has the advantage that it was recorded in anechoic conditions and the only parameter that matters is the relative position of the loudspeaker to the head during the measurement. This advantage can be used to create every possible loudspeaker array you can imagine, given that the relative locations of all loudspeakers are available in the HRTF data set. The above picture shows an example of a HRTF measurement. You can download the corresponding QU_KEMAR_anechoic_3m.sofa HRTF set, which we can directly use with the Toolbox.

The following example will load the HRTF data set and extracts a single impulse response for an angle of 30° from it. If the desired angle of 30° is not available, a linear interpolation between the next two available angles will be applied. Afterwards the impulse response will be convolved with the cello recording by the `auralize_ir()` function.

```
X = [0 0 0];
head_orientation = [0 0];
xs = [rad(30) 0 3];
coordinate_system = 'spherical';
hrtf = SOFAload('QU_KEMAR_anechoic_3m.sofa');
```

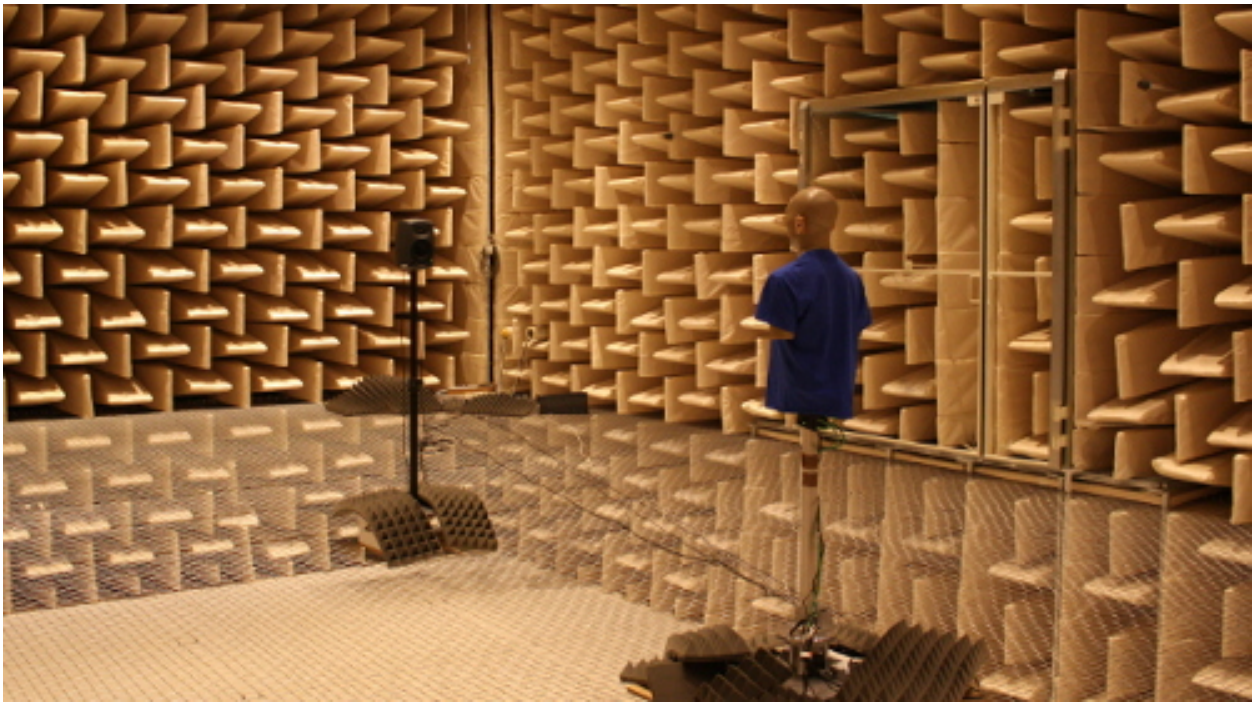

Fig. 7.1: Setup of the KEMAR (Knowles Electronics Manikin for Acoustic Research) and a loudspeaker during a HRTF measurement.

```
conf = SFS_config;
ir = get_ir(hrtf,X,head_orientation,xs,coordinate_system,conf);
cello = wavread('anechoic_cello.wav');
sig = auralize_ir(ir,cello,1,conf);
sound(sig,conf.fs);
```

To simulate the same source as a virtual point source synthesized by WFS and a circular array with a diameter of 3 m, you have to do the following.

```
X = [0 0 0];
head_orientation = [pi/2 0];
xs = [0 3 0];
src = 'ps';
hrtf = SOFAload('QU_KEMAR_anechoic_3m.sofa');
conf = SFS_config;
conf.secondary_sources.size = 3;
conf.secondary_sources.number = 56;
conf.secondary_sources.geometry = 'circle';
conf.dimension = '2.5D';
ir = ir_wfs(X,head_orientation,xs,src,hrtf,conf);
cello = wavread('anechoic_cello.wav');
sig = auralize_ir(ir,cello,1,conf);
```

If you want to use binaural simulations in listening experiments, you should not only have the HRTF data set, but also a corresponding headphone compensation filter, which was recorded with the same dummy head as the HRTFs and the headphones you are going to use in your test. For the HRTFs we used in the last example and the AKG K601 headphones you can download QU_KEMAR_AKGK601_hcomp.wav. If you want to redo the last simulation with headphone compensation, just add the following lines before calling `ir_wfs()`.

```
conf.ir.usehcomp = true;
conf.ir.hcompfile = 'QU_KEMAR_AKGK601_hcomp.wav';
conf.N = 4096;
```

The last setting ensures that your impulse response will be long enough for convolution with the compensation filter.

Binaural simulation of a real setup

Fig. 7.2: Boxed shaped loudspeaker array at the University Rostock.

Besides simulating arbitrary loudspeaker configurations in an anechoic space, you can also do binaural simulations of real loudspeaker setups. In the following example we use BRIRs from the 64-channel loudspeaker array of the University Rostock as shown in the panorama photo above. The BRIRs and additional information on the recordings are available for download, see doi:10.14279/depositonce-87.2. For such a measurement the SOFA (Spatially Oriented Format for Acoustics) file format has the advantage to be able to include all loudspeakers and head orientations in just one file.

```
X = [0 0 0];
head_orientation = [0 0];
xs = [3 0 0];
src = 'ps';
brir = 'BRIR_AllAbsorbers_ArrayCentre_Emitters1to64.sofa';
conf = SFS_config;
conf.secondary_sources.geometry = 'custom';
conf.secondary_sources.x0 = brir;
conf.N = 44100;
ir = ir_wfs(X,head_orientation,xs,src,brir,conf);
cello = wavread('anechoic_cello.wav');
sig = auralize_ir(ir,cello,1,conf);
```

In this case, we don't load the BRIRs into the memory with `SOFAload()` as the file is too large. Instead, we make use of the ability that SOFA can request single impulse responses from the file by just passing the file name to the `ir_wfs()` function. In addition, we have to set `conf.N` to a reasonable large value as this determines the length of the impulse response `ir_wfs()` will return, which has to be larger as for the anechoic case as it should now include the room reflections. Note, that the head orientation is chosen to be `0` instead of `pi/2` as in the HRTF examples due to a difference in the orientation of the coordinate system of the BRIR measurement.

Impulse response of your spatial audio system

Binaural simulations are also an interesting approach to investigate the behavior of a given spatial audio system at different listener positions. Here, we are mainly interested in the influence of the system and not the HRTFs so we simply use a Dirac impulse as HRTF as provided by `dummy_irs()`.

```
X = [0 0 0];
head_orientation = [0 0];
xs = [2.5 0 0];
src = 'ps';
t = (1:1000)/conf.fs*1000;
hrtf = dummy_irs(conf);
conf = SFS_config;
conf.t0 = 'source';
[ir,~,delay] = ir_wfs(X,head_orientation,xs,src,hrtf,conf);
figure;
figsize(540,404,'px');
plot(t,ir(1:1000,1),'-g');
hold on;
offset = round(delay*conf.fs);
plot(t,ir(1+offset:1000+offset,1),'-b');
hold off;
xlabel('time / ms');
ylabel('amplitude');
%print_png('img/impulse_response_wfs_25d.png');
```

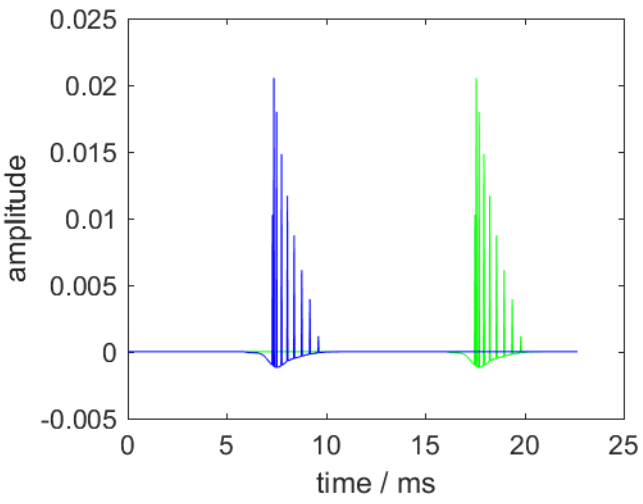

Fig. 7.3: Sound pressure of an impulse synthesized as a point source by 2.5D WFS at (2.5, 0, 0) m. The sound pressure is observed by a virtual microphone at (0, 0, 0) m. The impulse is plotted including the delay offset of the WFS driving function (green) and with a corrected delay that corresponds to the source poisition (blue).

The figure includes two versions of the impulse response at two different time instances. The green impulse response includes the processing delay that is added by `driving_function_imp_wfs()` and other functions performing filtering and delaying of signals. This delay is returned by `ir_wfs()` as well and can be used to correct it during plotting. The blue impulse response is the corrected one, which is now placed at 7.3 ms which corresponds to the actual distance of the synthesized source of 2.5 m.

The impulse response can also be calculated without involving functions for binaural simulations, but by utilizing directly `sound_field_imp()` related function.

```
X = [0 0 0];
head_orientation = [0 0];
xs = [2.5 0 0];
src = 'ps';
conf = SFS_config;
```

```
conf.N = 1000;
conf.t0 = 'source';
time_response_wfs(X,xs,src,conf)
axis([0 25 -0.005 0.025]);
%print_png('img/impulse_response_wfs_25d_imp.png');
```

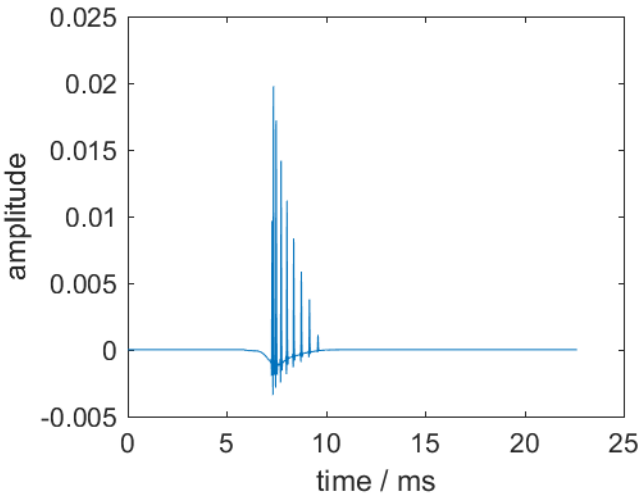

Fig. 7.4: Sound pressure of an impulse synthesized as a point source by 2.5D WFS at (2.5, 0, 0) m. The sound pressure is observed by a virtual microphone at (0, 0, 0) m.

This time the delay offset of the driving function is automatically corrected for and the involved calculation uses inherently a fractional delay filter. The downside is that the calculation takes longer and the amplitude is slightly lower by the involved fractional delay method.

Frequency response of your spatial audio system

Binaural simulations are also a nice way to investigate the frequency response of your reproduction system. The following code will investigate the influence of the pre-equalization filter in WFS on the frequency response. For the red line the pre-filter is used and its upper frequency is set to the expected aliasing frequency of the system (above these frequency the spectrum becomes very noise as you can see in the figure).

```
X = [0 0 0];
head_orientation = [pi/2 0];
xs = [0 2.5 0];
src = 'ps';
hrtf = dummy_irs(conf);
conf = SFS_config;
conf.ir.usehcomp = false;
conf.wfs.usehpre = false;
[ir1,x0] = ir_wfs(X,head_orientation,xs,src,hrtf,conf);
conf.wfs.usehpre = true;
conf.wfs.hprefhigh = aliasing_frequency(x0,conf);
ir2 = ir_wfs(X,head_orientation,xs,src,hrtf,conf);
[a1,p,f] = spectrum_from_signal(norm_signal(ir1(:,1)),conf);
a2 = spectrum_from_signal(norm_signal(ir2(:,1)),conf);
figure;
figsize(540,404,'px');
```

```
semilogx(f,20*log10(a1),'-b',f,20*log10(a2),'-r');
axis([10 20000 -80 -40]);
set(gca,'XTick',[10 100 250 1000 5000 20000]);
legend('w/o pre-filter','w pre-filter');
xlabel('frequency / Hz');
ylabel('magnitude / dB');
%print_png('img/frequency_response_wfs_25d.png');
```

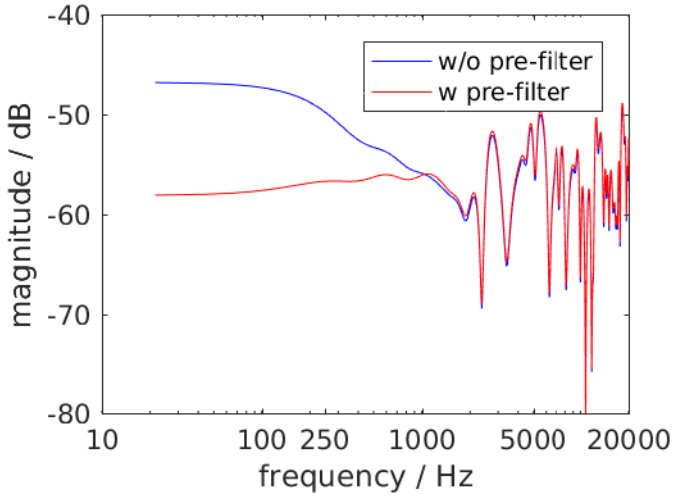

Fig. 7.5: Sound pressure in decibel of a point source synthesized by 2.5D WFS for different frequencies. The 2.5D WFS is performed with and without the pre-equalization filter. The calculation is performed in the time domain.

The same can be done in the frequency domain, but in this case we are not able to set a maximum frequency of the pre-equalization filter and the whole frequency range will be affected.

```
X = [0 0 0];
xs = [0 2.5 0];
src = 'ps';
conf = SFS_config;
freq_response_wfs(X,xs,src,conf);
axis([10 20000 -40 0]);
%print_png('img/frequency_response_wfs_25d_mono.png');
```

Using the SoundScape Renderer with the SFS Toolbox

In addition to binaural synthesis, you may want to apply dynamic binaural synthesis, which means you track the position of the head of the listener and switches the used impulse responses regarding the head position. The SoundScape Renderer (SSR) is able to do this. The SFS Toolbox provides functions to generate the needed wav files containing the impulse responses used by the SoundScape Renderer. All functions regarding the SSR (SoundScape Renderer) are stored in folder `SFS_ssr`.

```
X = [0 0 0];
head_orientation = [pi/2 0];
xs = [0 2.5 0];
src = 'ps';
hrtf = SOFAload('QU_KEMAR_anechoic_3m.sofa');
```

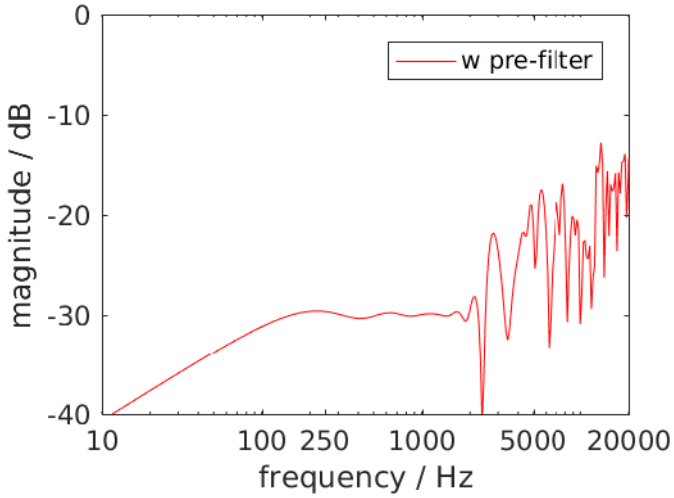

Fig. 7.6: Sound pressure in decibel of a point source synthesized by 2.5D WFS for different frequencies. The 2.5D WFS is performed only with the pre-equalization filter active at all frequencies. The calculation is performed in the frequency domain.

```
conf = SFS_config;
brs = ssr_brs_wfs(X,head_orientation,xs,src,hrtf,conf);
wavwrite(brs,conf.fs,16,'brs_set_for_SSR.wav');
```

CHAPTER 8

Helper Functions

The Toolbox provides you also with a set of useful small functions. Here the highlights are angle conversion with `rad()` and `deg()`, FFT (Fast Fourier Transform) calculation and plotting `spectrum_from_signal()`, rotation matrix `rotation_matrix()`, multi-channel fast convolution `convolution()`, nearest neighbour search `findnearestneighbour()`, even or odd checking `iseven() isodd()`, spherical Bessel functions `sphbesselh() sphbesselj() sphbessely()`.

CHAPTER 9

Plotting

The Toolbox provides you with a function for plotting your simulated sound fields (`plot_sound_field()`) and adding loudspeaker symbols to the figure (`draw_loudspeakers()`). If you have gnuplot installed, you can use the functions `gp_save_matrix()` and `gp_save_loudspeakers()` to save your data in a way that it can be used with gnuplot. An example use case can be found at this plot of a plane wave which includes the Matlab/Octave code to generate the data and the gnuplot script for plotting it.

CHAPTER 10

Version History

2.4.1 (28. September 2017)

- add monochromatic implementation of LWFS using spatial bandwidth-limitation
- add monochromatic circular expansion functions for ps and pw
- add function for conversion from circular to plane wave expansion
- add freq_response_* and time_response_* for all LWFS methods
- add optional message arg to progress_bar()
- fix missing conf.N in freq_response_nfchoa()
- fix auralize_ir() for local files

2.4.0 (22. August 2017)

- improve references in SFS_config()
- update structure of configuration for LWFS methods
- fix off-center dummy head positions for HRTFs
- add elevation to head orientation for binaural synthesis
- fix sphbesselh_zeros() for high orders
- fix symmetric ifft for Octave
- add inverse Legendre transform
- fix integral weights for spherical secondary sources
- add 3D ps and pw driving functions for NFC-HOA
- add 'reference_circle' as new default for focused sources in 2.5D
- add max-rE and tukey modal weighting windows
- add time-domain implementation of LWFS using spatial bandwidth-limitation
- add circular expansion functions

- fix incorporation of tapering weights for LWFS
- remove x0 from interpolate_ir() call
- fix interpolate_ir() for special cases
- switch handling of time from samples to seconds
- add freq_response_line_source()
- add freq_response_point_source()
- add time_response_line_source()

2.3.0 (04. March 2017)

- default 2D WFS focused source is now a line sink
- improve point selection and interpolation of impulse responses
- speed up Parks-McClellan resampling method
- change default value of conf.usebandpass to false
- rename conf.wfs.t0 to conf.t0
- rename and improve easyffft() to spectrum_from_signal()
- rename and improve easyifft() to signal_from_spectrum()
- correct amplitude values of WFS and NFC-HOA in time domain
- fix default 2.5D WFS driving function in time domain
- add time_response_point_source()
- update amplitude and position of dirac in dummy_irs()
- fix missing secondary source selection in ssr_brs_wfs()
- add amplitude terms to WFS FIR pre-filter
- fix Gauss-Legendre quadrature weights
- add delay_offset as return value to NFC-HOA and ir funtions
- fix handling of delay_offset in WFS time domain driving functions

2.2.1 (22. August 2016)

- fix delayoffset for FIR fractional delay filter
- add findconvexcone()
- simplify convolution()
- add linear interpolation working in the frequency domain
- fix pm option for delayline()

2.2.0 (7. July 2016)

- fix impulse response interpolation for three points
- add the ability to apply modal weighting window to NFC-HOA in time domain
- change license to MIT
- update delayline() config settings
- add Lagrange and Thiran filters to delayline()

- replace wavread and warwrite by audioread and savewav
- convolution() excepts now two matrices as input
- allow headphone compensation filter to be a one- or two-channel wav file
- add new online doc at http://matlab.sfstoolbox.org/
- fix greens_function_mono() for plane wave and 3D
- replace conf.ir.useoriglength by conf.ir.hrirpredelay
- update default WFS driving functions
- add links to equations in online theory at http://sfstoolbox.org

2.1.0 (10. March 2016)

- make conf struct mandatory
- add new start message
- fix handling of 0 in least squares fractional delays
- fix NFC-HOA order for even loudspeaker numbers to N/2-1
- add conf.wfs.hpreFIRorder as new config option (was hard coded to 128 before)
- speed up secondary source selection for WFS
- rename chromajs colormap to yellowred
- fix tapering_window() for non-continuous secondary sources
- remove cubehelix colormap as it is part of Octave
- add conf.wfs.t0 option which is useful, if you have more than one virtual source
- virtual line sources are now available for monochromatic WFS and NFC-HOA
- allow arbitrary orders for time-domain NFC-HOA simulations

2.0.0 (26. October 2015)

- add support for SOFA
- add SOFA convention SimpleFreeFieldHRIR
- add SOFA convention MultiSpeakerBRIR
- calculate integration weights (x0(:,7)) of secondary sources based on their distances to their neighbours
- add rounded-box as new loudspeaker array geometry
- fix bugs in local sound field synthesis time domain implementation
- speedup local sound field synthesis processing by fewer calls to delayline()
- add heuristic to find a good local wave field synthesis pre-filter
- loudspeaker geometry can now be read from a SOFA file
- now custom grids can be used during sound field simulations
- add 3D plot routine
- change plot_sound_field(P,x,y,z) to plot_sound_field(P,X,Y,Z)
- normalization of sound field now only happens in plot_sound_field(); this comes with the new config option conf.plot.normalisation

- remove interaural_level_difference() and interaural_time_difference()
- change default config setting conf.ir.usehcomp to false
- lots of small bug fixes

1.2.0 (2. June 2015)

- add PDF documentation "Theory of Sound Field Synthesis"
- fix remaining usegnuplot config entry
- change default dB color map to chromajs
- add missing hgls2 functionality (fractional delays)
- add cubehelix and chromajs color maps
- remove noise() function, use the one from the LTFAT Toolbox instead

1.1.0 (2. April 2015)

- fix amplitude bug in get_ir() and ir_generic()
- remove direct gnuplot plotting
- add support for local Wave Field Synthesis
- the length of the dirac impulse response is now an option for dummy_irs()
- fix iseven(), isodd() for very large numbers
- correct the sign for Wave Field Synthesis driving functions

1.0.1 (4 August 2014)

- rms() works now also with row vectors in order to be compatible with the Auditory Modeling Toolbox
- fixed handling of number of secondary sources for a box shaped array
- fixed a bug in ir_auralize() regarding the contentfile configuration
- corrected NFC-HOA driving functions for off-center arrays

1.0.0 (27 March 2014)

- added references for all driving functions
- streamlined nested conf settings; e.g. now it is no longer neccessary to set conf.ir.hcompfile if conf.usehcomp == false
- added WFS driving functions from Völk et al. and Verheijen et al.
- removed secondary_source_number() and xy_grid, because they are no longer needed
- enabled pre-equalization filter of WFS as default in SFS_config_example()
- fixed sound_field_mono_sdm_kx()
- Green's function for line sources returns now real values
- correct y-direction of plane waves for 3D NFC-HOA
- updated the test functions in the validation folder
- several small fixes

1.0.0-beta2 (5 December 2013)

- rms() now works for arbitrary arrays

- speedup of delayline() and HRTF extrapolation
- delayline() now works with more than one channel
- fixed a critical bug in wfs_preequalization()
- fixed missing conf values in several functions
- fixed README
- changed location of sfs-data for automatic download, because github does not allow this
- several minor fixes

1.0.0-beta (26 August 2013)

- bandpass() can now handle arbritrary frequency limits
- sphbesselh_zeros() comes now with precomputed zeros for an order up to 1000
- renamed wave_field_* functions to sound_field_*
- the order for NFC-HOA can now be set manually via conf.nfchoa.order
- several performance improvements
- added missing driving functions for WFS and NFC-HOA
- added convolution() which is faster than conv and can handle multidimensional signals
- changed default plotting style of loudspeakers to conf.plot.realloudspeaker=false
- hann_window() now uses (2*n+1) instead of (2*n) to generate the window
- replaced the input parameter L by conf.secondary_sources.size
- the aliasing frequency is now calculated by the mean distance between the given secondary sources
- added nearest neighbour search and 3D interpolation to get_ir()
- moved the tapering window into x0(:,7), added new function secondary_source_tapering to achieve this
- added a seventh column to x0 which includes integrational weights
- added extra directory for SSR renderer functions
- added 3D HRTF extrapolation
- changed array configuration to use number of secondary sources instead of distance between them
- changed SFS_config to use substructs like conf.secondary_sources.*
- added the possibility to calculate the wave field for a arbritrary positioned plane in 3D
- added 3D WFS functions
- make the Toolbox work in 3D, which brakes backwards compability!
- now all monochromatic functions have a time_domain counterpart
- reordered the argouts for the wave field functions; now P is always the first argout
- automatically plotting of the wave fields if no argouts are wanted
- changed direction of focused source from the conf.xref vector directly into xs. For a focused source xs is now [1x6]

0.2.5 (12 July 2013)

- fixed a bug causing the wrong loudspeaker position in the output of generic_wfs()

0.2.4 (4 June 2013)

- added a documentation to the github README
- reworked the plotting, now simple saving to png is possible
- added a narginchk function for older Matlab versions
- replaced conf.frame with t in the imp functions
- lots of small bugs were fixed

0.2.3 (9 April 2013)

- summed up line, point, ... sources to green_function for mono and imp
- introduced global wave_field functions for mono and imp
- fixed binaural simulations for NFC-HOA
- removed compatibility for octave versions <3.6
- fixed a critical bug for the HRTF farfield extrapolation, due to the new secondary source selection behavior

0.2.2 (27 November 2012)

- added functions to calculate the sound pressure for monochromatic WFS at a single point in analogy to the point_source function
- changed the behavior of secondary_source_selection to returning a new x0 vector
- added compatibility for octave 3.6
- first fix of secondary source selection for focused sources (now they point always in the direction of the reference point)

0.2.1 (15 June 2012)

- added NFC-HOA 2.5D monochromatic
- added NFC-HOA 2.5D binaural simulations
- added SDM 2.5D monochromatic
- make NFC-HOA work under Octave
- fixed direction of plane waves and point sources for NFC-HOA time domain simulations
- changed syntax for wave_field_* and driving_* functions:
  - xs,f,src => xs,src,f
  - xs,L,src => xs,src,L
  - xs,L,f,src => xs,src,f,L

0.2.0 (25 April 2012)

- first public release (under the GPLv3+ license)