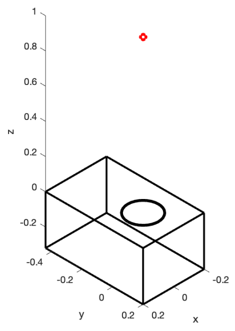
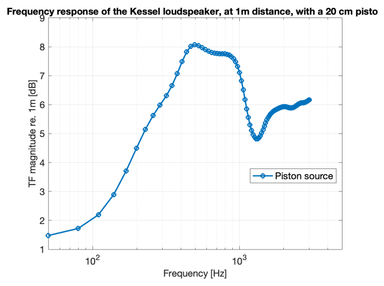
Edge diffraction Matlab toolbox
EDtoolbox v 0.501
Manual





Peter Svensson (peter.svensson@ntnu.no)

Acoustics group, Department of Electronic Systems, NTNU

March 25, 2024

Contents

1 Introduction

The edge diffraction Matlab toolbox, '`EDtoolbox`', presented here computes the scattered sound pressure, for polyhedral scattering objects with rigid surfaces (Neumann boundary condition). The scattered sound pressure is modeled as a sum of geometrical acoustics components, and first- and higher-order diffraction components. In the current version,'`EDtoolbox`' v 0.501, external scattering problems for convex bodies can be studied, with time- or frequency-domain computations. The sources can either be monopoles or pistons, represented by polygons. One goal for this toolbox is that compact scripts can document precisely how a computation was done, which should help to make computed results more reproducible and transparent. Furthermore, several different example scripts are supplied, so that it should be easy to get started.

The author has worked with the development of "Edge diffraction Matlab toolboxes" since around 1999. Previous versions (EDB1, EDB2, ESIE0, ESIE1, ESIE2) had evolved into quite a huge set of functions of mixed software quality. Those toolboxes tried to handle external as well as internal scattering problems with a single main program, which contributed to the complexity. So, in the late fall of 2017 during a sabbatical at UCL, the department of mathematics, which was hosted by Dr. David Hewett, quite a thorough clean-up process was started and a first version of the toolbox was focused on a restricted set of problems. In addition, the toolbox was made available at github.com.

1.1 License

This Matlab toolbox is offered under the BSD license, that is, the same as all files that are shared in Mathworks File Exchange. The license text is as follows:

1.2 Acknowledgements

Many people have contributed on a smaller and larger scale during more than twenty years of work. Particularly large contributions to the toolbox and/or to the edge diffraction research have been made by John Vanderkooy, Rendell Torres, Paul Calamia, Chris Strahm, Andreas Asheim, Jason Summers, David Hewett, Sara Martin. I have also had helpful contributions by, and discussions with, Roger Fred, Djamel Ouis (who first pointed out the Biot-Tolstoy solution to me), Herman Medwin, Bengt-Inge Dalenbäck, Jonathan Hargreaves, Jan Slechta, Bjørn Kolbrek, Stephan Ewert.

Parts of this work have been financed through Sara Martin's project "Hybrid methods" funded by the Research Council of Norway, and through Jan Slechta's stipend from the ERCIM Alain Bensoussan Fellowship Programme.

2 The ED toolbox

2.1 Installation

At `github.com`, you can find the last version of the `EDtoolbox` at

`https://github.com/upsvensson/Edge-diffraction-Matlab-toolbox`.

On that page, you can press `"Clone or download"` and get the full set of Matlab files, including this manual. It is recommended that you store all the files in a folder called `"EDtoolbox"`, and that you include the path to that folder in Matlab's path. You will also need to include the path to some subfolders, as described below. The cloning should generate these files in the main folder:

- README.md
- LICENSE.md
- EDtoolbox_manual_v0501.pdf (which is this pdf file)
- Directory `toolbox`

– `EDmain_convex.m`	This is the main function to run
– `EDplotmodel.m`	This is a function which generates a plot of the model
– Directory `internal`	This directory contains the functions/subroutines used by `EDmain_convex`
• Directory `examples`	This directory contains a number of example scripts.

You must also download two functions from Matlab Central. These two functions are required for `EDtoolbox`, and they must be stored in a folder which is added to Matlab's list of paths. Those two functions are

- `DataHash`, developed by Jan Simon. This function is used to create a hash, a 32-bit character that is unique for all the calculation settings and the specific toolbox version number. That hash is stored in each intermediate file which is (optionally) stored. At a subsequent run of the main EDtoolbox program, the directory with result files is scanned for existing files with the same hash value, and if a result file is found with the right hash stored, that existing file will automatically be reused instead of calculating a new.
- `lgwt`, developed by Greg von Winckel. This function gives the Gauss-Legendre nodes and weights.

2.2 Overview

The EDtoolbox computes the scattered sound pressure for a scattering object with rigid surfaces (Neumann boundary condition). In the current version of the toolbox, only external scattering problems can be studied, and the underlying theory is well-developed for convex-shaped scattering objects. The scattered sound pressure is decomposed into several terms, illustrated in Fig. 1,

$$\begin{aligned} p_{\text{total}} &= p_{\text{direct}} + p_{\text{specular}} + p_{\text{1. order diffraction}} \\ &+ p_{\text{2. order diffraction}} + p_{\text{3. order diffraction}} + \cdots \\ &= p_{\text{direct}} + p_{\text{specular}} + p_{\text{1. order diffraction}} + p_{\text{HOD}}. \end{aligned} \tag{1}$$

As of version 0.41, the EDtoolbox handles:

- convex geometries (type 0 in Table 1) for external scattering problems,
- ideally rigid surfaces (Neumann boundary condition)[a],
- practically arbitrarily high diffraction orders with the ESIE method in the frequency domain (FD) [?], or
- low diffraction orders with the "separate diffraction orders" method in the time domain (TD) [?], or
- the free-field case, that is, without any scattering object. This case makes it easy to generate a large number of impulse responses or transfer functions.

Sources can be

- Monopoles, or
- Pistons (as parts of the planes that form the scattering object), represented by polygons.

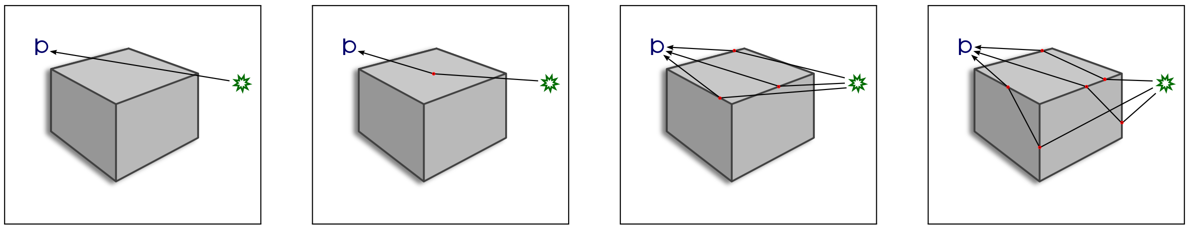
Above version 0.213, but below version 0.300, also these cases are handled by preliminary functions:

- non-convex geometries of type 1 (see Table 1) for external scattering problems for low-diffraction-order TD calculations using the "separate diffraction orders" method [?].

These non-convex cases will be implemented again in updates to version 0.41.

[a]The Dirichlet boundary condition can also be handled for specular reflections and first-order diffraction, but not for higher-order diffraction.

It can be noted that for a convex scattering body, combinations of specular reflections and edge diffractions can not occur. Furthermore, only first-order specular reflections are possible, which simplifies greatly the identification of possible paths.



(a) (b) (c) (d)

Figure 1: Illustration of the decomposition of the scattered sound field for a convex-shaped polyhedral scatterer into components: (a) Direct sound, (b) Specular reflection, (c) First-order diffraction, (d) Second-order diffraction

Table 2 summarizes the cases that have been implemented. The method denoted `ESIEBEM` uses the ESIE approach to compute the sound pressure at intermediate receiver positions at the surface of the scattering object, and then propagates this surface sound pressure to the external receiver positions using the Helmholtz integral, just like the boundary element method (BEM) does [?].

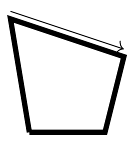
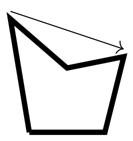
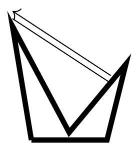
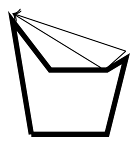
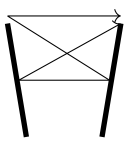
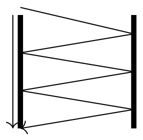
Geometry case:	**Geometry examples**	
Edge-to-edge paths can have no specular reflections	 Type 0: Convex	 Type 1: Non-convex, but generates no specular reflections in edge-to-edge paths
Edge-to-edge paths can have maximum one specular reflection	 Type 2a: Non-convex, specular back-reflection possible	 Type 2b: Non-convex, specular reflection possible in forward path and in back-reflections
Edge-to-edge paths can have a finite number of specular reflections	 Type 3: Non-convex, finite number of specular reflections possible	
Edge-to-edge paths can have an infinite number of specular reflections	 Type 4: Non-convex, infinite number of specular reflections possible	

Table 1: Types of geometry for external scattering problems

Table 2: EDtoolbox - Implemented cases as of version 0.501

Problem type **Geometry type** **Method (FD/TD)**	**Main function** **Parameter setting**	**Result variables**
External problems, convex objects Type 0	`EDmain_convex`	

Method: ESIE
(FD)
	`controlparameters.docalctf`=1	`tfdirect, tfgeom,`
		`tfdiff, tfinteqdiff`

Method: ESIEBEM (FD)	`controlparameters.docalctf_ESIEBEM`=1	`tftot_ESIEBEM`

Method: separate
diffraction orders (TD)
	`controlparameters.docalcir`=1	`irdirect, irgeom,`
		`irdiff, irhod`

External problems, non-convex objects Type 1	Preliminary TD version (above 0.213, below 0.300): `EDmain_nonconvex_time`	
External problems, non-convex objects Type 2-4	Not implemented yet (was implemented in `EDB1`)	
Internal problems	Not implemented yet (was implemented in `EDB1`)	
Free-field	`EDmain_convex` `geoinputdata.freefieldcase`=1	

	`controlparameters.docalctf`=1	`tfdirect, tfgeom,`
		`tfdiff, tfinteqdiff`

	`controlparameters.docalcir`=1	`irdirect, irgeom,`
		`irdiff, irinteqdiff`

The EDtoolbox gives the value of the sound pressure at a receiver *for a normalized source amplitude of 1*; that is, the result could be viewed as a transfer function (or an impulse response), which is why the output variables have names such as `tfdirect` etc. The transfer functions (TF) are defined such that a free-field radiating monopole has the transfer function

$$\mathrm{TF}_{\mathrm{free-field}} = \frac{\mathrm{e}^{-\mathrm{j}kr}}{r}$$

and all other transfer functions are scaled accordingly. For time-domain (TD) calculations, the corresponding free-field impulse response is

$$\mathrm{IR}_{\mathrm{free-field}}(t) = \frac{1}{r}\delta\left(t - \frac{r}{c}\right)$$

However, this impulse response is a function of continuous time, t, and the computed ones are functions of discrete time. Thus, the free-field impulse response is actually computed as described in Section 3.1.

As a consequence of the transfer function/impulse response view described above, it could also be interpreted that the EDtoolbox gives the sound pressure at the receiver if the monopole's source signal amplitude is 1, and this source signal, Q_{M}, is, for frequency-domain (FD) calculations,

$$Q_{\mathrm{M}} = \frac{\mathrm{j}\omega\rho_0 U_0}{4\pi}$$

where U_0 is the volume velocity of the monopole. For TD calculations the monopole source signal is

$$Q_{\mathrm{M}}(t) = \frac{\rho_0}{4\pi}\frac{\mathrm{d}}{\mathrm{d}t}U_0(t)$$

If one prefers, it is also possible to call the output quantities "sound pressure re. 1m free-field", with the additional information that the phase reference is determined by one parameter setting, called `controlparameters.Rstart`, expecting a value in meters. The default value is zero, which would yield an `irdirect`, for the case of a receiver 1 m from a source, as being a pulse of amplitude 1, at the time slot that corresponds to the propagation time for 1m.

New in version 0.400 The EDtoolbox up to version 0.303 used only monopoles, but from version 0.400, polygonal pistons can also be used as sources. The amplitude scaling of piston sources is handled such that a piston source, and a monopole placed very close to the scattering object, will give the same transfer function amplitude for low frequencies (where the piston source is non-directional), regardless of size of the piston.

A plane wave can be emulated by doing those steps:

1. Place a monopole arbitrarily far away, at a distance of, say, 10^6 m.
2. Set `controlparameters.Rstart` to the distance to the monopole. This implies that the incident sound wave has the phase 0 at the origin. This is especially important for TD calculations with sources far away since otherwise, an impulse response with extremely many initial zeros will be generated.
3. Set `Sinputdata.sourceamplitudes` to the distance to the monopole.

Sources and receivers can be placed at surfaces, but they must be positioned a tiny little distance from the surface of the scattering object; 10^{-5} m or 10^{-4} m suffices. The reason for this is that the functions must be able to determine which side of a plane that the sources and receivers are.

2.3 How to run the `EDmain_convex` function

The `EDmain_convex` function is run by assigning values to six input structs, each with a number of fields, that are further described in Section 4.1,

- `geoinputdata`
- `Sinputdata`
- `Rinputdata`
- `controlparameters`
- `filehandlingparameters`
- `envdata`

and then call the main function, such as

```
EDmain_convex(geoinputdata,Sinputdata,Rinputdata,envdata,...
controlparameters,filehandlingparameters)
```

or

```
EDres = EDmain_convex(geoinputdata,Sinputdata,Rinputdata,envdata,...
controlparameters,filehandlingparameters);
```

The second version above returns the results (impulse responses or transfer functions) into the struct `EDres`, and saves the results in files. The first version above only saves the results in files.

Attempts have been made to give many default values to settings, such that it can be easy to get started.

2.3.1 A minimal working example - the frequency response of a loudspeaker

As one example, the script below (available also as `EDexample_LspKessel_minimal.m` inside the `examples` folder in the `EDtoolbox` folder) defines a shoebox-shaped loudspeaker box[1], a point source right at the box (at a distance of 10^{-5} m from the surface of the box), and a receiver 1 m away, see Fig. 2 (a). The function `EDmain_convex` is run, and after the calculations have been run, the frequency response is plotted. Fig. 2 (b) shows the resulting output.

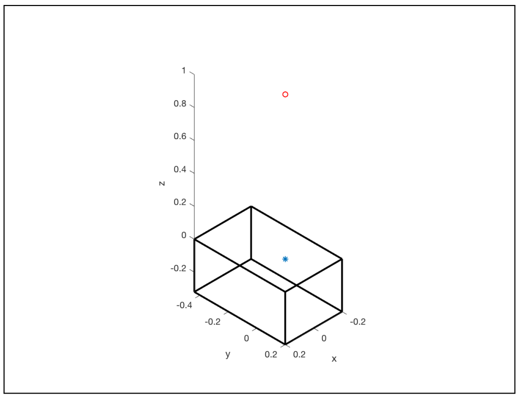
The response in Fig. 2 (b) demonstrates the typical "baffle-step" in the response, that is, a step-up by 6 dB from low to high frequencies, with interference ripple effects. At low frequencies (LF), the magnitude of the transfer function/frequency response should be $1/r$ where r is the distance. For short distances, this will not be true, and we can see that the response does not quite tend towards 0 dB. For receivers at a very long distance, however, the LF response magnitude will indeed come very close to $1/r$.

The response is computed up to 3000 Hz. For higher frequencies (HF), a larger number of edge points (setting `controlparameters.ngauss`) would be needed. One can estimate that 3 edge points per shortest wavelength are needed. The response at HF can be computed much more efficiently with TD calculations, as demonstrated in Section 6.4.

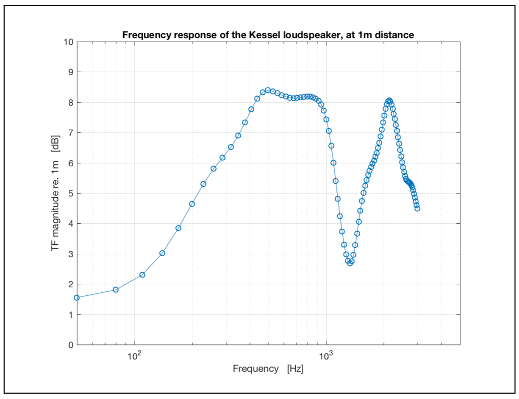
```
% EDexample_LspKessel_minimal.m

mfile = mfilename('fullpath');
[infilepath,filestem] = fileparts(mfile);

%---------------------------------------------------------------------
% Here the size of the shoebox-shaped enclosure is defined
% The function EDmakegeo_shoebox centers the box around the origin, with the
% front baffle at z = 0 but we
% shift it in the y-direction so that the origin (where the source will be)
% has 0.2m to all edges.

[corners,planecorners] = EDmakegeo_shoebox(0.4,0.64,0.32);
corners(:,2) = corners(:,2) - 0.12;
geoinputdata = struct('corners',corners,'planecorners',planecorners);

%---------------------------------------------------------------------
% The source (single point) and receiver are defined

sourcecoordinates = [0 0 0.00001];
Sinputdata = struct('coordinates',sourcecoordinates);

receivercoordinates = [0 0 1];
Rinputdata = struct('coordinates',receivercoordinates);

%---------------------------------------------------------------------
% Set some calculation parameter values

fvec = linspace(50,3000,100);
controlparameters = struct('frequencies',fvec);
controlparameters.docalctf = 1;

% The settings below could be changed, if desired
% .ngauss could be increased for higher frequencies, and/or higher accuracy
% .difforder could be reduced for faster calculations (the default is 10)
% controlparameters.ngauss = 24;
% controlparameters.difforder = 6;

%---------------------------------------------------------------------
% Specify output file names and location

% If the directory doesn't exist; it will be created
```

[1]As described in Section 4.1, the matrices `corners` and `planecorners` can be defined "by hand", or using a function such as `EDmakegeo_shoebox`, and the latter alternative is used here.

```
outputdirectory = [infilepath,filesep,'results'];
filehandlingparameters = struct('outputdirectory',outputdirectory);
filehandlingparameters.filestem = filestem;

%------------------------------------------------------------------------
% Run the calculations

EDres = EDmain_convex(geoinputdata,Sinputdata,...
Rinputdata,struct,controlparameters,filehandlingparameters);

%------------------------------------------------------------------------
% Present the results

figure(1)
semilogx(controlparameters.frequencies,20*log10(abs(EDres.tftot)),'-o')
xlabel('Frequency [Hz]');
ylabel('TF magnitude re. 1m free-field [dB]')
title('Frequency response of the Kessel loudspeaker, at 1m distance')
axis([50 5000 0 10]);
grid

EDplotmodel([filehandlingparameters.outputdirectory,filesep,...
filehandlingparameters.filestem,'_eddata.mat'],'plotoptions',3,'figurewindow',2);
```



(a)



(b)

Figure 2: The "Kessel" loudspeaker example [?]. (a) The loudspeaker model, with source and receiver positions. (b) The frequency response at 1m distance, for a point source, computed with `EDmain_convexESIE`.

2.4 Known bugs and limitations

Up to version 0.215, the following two bugs were known, as revealed by test cases 7 and 8 in the test function `EDverify`. These bugs were solved for version 0.216:

- If the receiver is hidden behind the scattering object, but the direct sound path happens to pass exactly through two edges, then the direct sound path was previously not detected as obstructed. This failure was revealed by the test function `EDverify`, test case number 8 (see Section 5.2). From v. 0.216 the direct sound path is correctly found to be obstructed for this special case.

- Similarly, if the receiver is hidden behind a scattering object and the direct sound path passes exactly through two corners, the path was previously not detected as obstructed. This failure was revealed by the test function `EDverify`, test case number 7 (see Section 5.2). From v. 0.216 the direct sound path is correctly found to be obstructed for this special case.

Furthermore, up to version 0.215, yet another bug was known, as revealed by test case 4 in the test function `EDverify`. This bug was also solved for version 0.216 (even though the fix is a bit mysterious):

- If the direct sound path, or a specular reflection, passes exactly through a single corner, which means that the direct sound is right on the border of being visible, then a little jump resulted in the wavefield, as revealed by test case 4 in the test function `EDverify`.

This cornerhit-case is a special case of the more common situation that the direct sound passes exactly through *an edge*. For an edge hit, one of the four diffraction terms[2] becomes singular and is enforced to be

[2]See [?]: the β directivity factor is a sum of four terms.

zero, at the same time as the direct sound is scaled to get half the unobstructed direct sound wave amplitude[?]. This treatment gives a perfectly smooth total wavefield across the so-called *zone-boundaries* that result from such edge hits. However, when the direct sound passes exactly through a single *corner* (of 90 degrees), one could have expected that the direct sound, and specular reflection, should also be scaled by 1/2, but it turns out (empirically) that the direct sound should be scaled to 2/3 of the unobstructed direct sound wave amplitude and the specular reflection should be scaled to 1/3. For other than 90-degree corners, there is still a small jump in the total wave-field.

One missing implementation, is that detailed timing data is not saved for the integral equation, when there are several sources, but `doaddsources = 0`.

Another bug is that for the `ESIEBEM` main function, the parameter `.doallSRcombinations` (see Table 4) has no effect - all source/receiver combinations are always computed.

See also section 3.6.

3 A little theory

As indicated by Eq. (1), and Fig. 1, the sound pressure is decomposed into four types of components, and they are computed by different methods as follows.

3.1 Direct sound

The direct sound is, in the FD, given by

$$\mathrm{TF}_{\mathrm{dir}} = \frac{\mathrm{e}^{-\mathrm{j}kr}}{r}\mathrm{V}_{\mathbf{x}_\mathrm{R},\mathbf{x}_\mathrm{S}}$$

where $r = |\mathbf{x}_\mathrm{R} - \mathbf{x}_\mathrm{S}|$ and $\mathrm{V}_{\mathbf{x}_2,\mathbf{x}_1}$ is a visibility function[3]:

$$\mathrm{V}_{\mathbf{x}_2,\mathbf{x}_1} = \begin{cases} 1 & \text{if the line from } \mathbf{x}_1 \text{ to } \mathbf{x}_2 \text{ is unobstructed, i.e., does not hit any of the} \\ & \text{finite planes of the scattering object} \\ 2/3 & \text{if the line from } \mathbf{x}_1 \text{ to } \mathbf{x}_2 \text{ hits the corner of some finite plane} \\ 0.5 & \text{if the line from } \mathbf{x}_1 \text{ to } \mathbf{x}_2 \text{ hits the edge of some finite plane} \\ 0 & \text{if the line from } \mathbf{x}_1 \text{ to } \mathbf{x}_2 \text{ is obstructed, i.e., passes through the interior} \\ & \text{of some finite plane} \end{cases}$$

The obstruction check is done by first identifying which polygons are potentially obstructing: those for which $\mathbf{x}_1$ and $\mathbf{x}_2$ are on opposite sides of the infinite plane that the polygon belongs to. For the potentially obstructing polygons, the hit-point is computed, that is, the point where the ray from $\mathbf{x}_1$ to $\mathbf{x}_2$ crosses the infinite plane. Then the ray-shooting algorithm is used, for a 2D-projection of the 3D-polygon to be tested.
In the time domain, the continuous-time impulse response expression is

$$\mathrm{TF}_{\mathrm{dir}}(t) = \frac{\delta\left(t - \frac{r}{c}\right)}{r}\mathrm{V}_{\mathbf{x}_\mathrm{R},\mathbf{x}_\mathrm{S}}$$

In the EDtoolbox, a discrete-time IR is computed, and then the Dirac delta function is represented as

$$h(t) = A \cdot \delta(t - t_0) \rightarrow h(n) = A(1-a)\cdot \mathrm{d}(n - n_0) + A \cdot a \cdot \mathrm{d}(n - n_0 - 1)$$

where $\mathrm{d}(n - n_0)$ is a unit pulse function, n_0 is the last discrete-time sample before t_0,

$$n_0 = \lfloor f_S \cdot t_0 \rfloor$$

and $a \in [0, 1]$ specifies where in the discrete-time sample slot that t_0 is:

$$a = t_0 - \frac{n_0}{f_S}$$

This discrete-time representation of the Dirac pulse has two pulses and gives zero-phase error, but quite a significant magnitude low-pass filter effect. Therefore, one might want to use a high sampling frequency in order to reduce this low-pass filter effect.

3.2 Specular reflection

The specular reflection is computed very similarly to the direct sound. We show only the FD version here:

$$\mathrm{TF}_{\mathrm{spec}} = \frac{\mathrm{e}^{-\mathrm{j}kr}}{r}\mathrm{V}'_{\mathbf{x}_\mathrm{R},\mathbf{x}_\mathrm{IS}}$$

where $r = |\mathbf{x}_\mathrm{R} - \mathbf{x}_\mathrm{IS}|$ and $\mathrm{V}'_{\mathbf{x}_2,\mathbf{x}_1}$ is a reflection visibility function:

$$\mathrm{V}'_{\mathbf{x}_2,\mathbf{x}_1} = \begin{cases} 1 & \text{if the line from } \mathbf{x}_1 \text{ to } \mathbf{x}_2 \text{ is unobstructed, i.e., does not hit any of the} \\ & \text{finite planes of the scattering object} \\ 0.5 & \text{if the line from } \mathbf{x}_1 \text{ to } \mathbf{x}_2 \text{ hits the edge of some finite plane} \\ 1/3 & \text{if the line from } \mathbf{x}_1 \text{ to } \mathbf{x}_2 \text{ hits the corner of some finite plane} \\ 0 & \text{if the line from } \mathbf{x}_1 \text{ to } \mathbf{x}_2 \text{ is obstructed, i.e., passes through the interior} \\ & \text{of some finite plane} \end{cases}$$

[3]The factor of 2/3 for a corner hit gives a smooth total field for a 90-degree thin-plate corner, but for other corners this factor should get another value, which has not been investigated further, and therefore not implemented. See also Section 2.4

3.3 First-order diffraction

First-order diffraction is computed in the FD with the expressions given in [?] and in the TD with the expressions given in [?]. For both FD and TD computations, the first step is to identify which edges are visible from the source and from the receiver. For convex polyhedra, this is an easy step: each edge is defined by two connected polygons, and the two polygons each have a frontal side/face towards the spatial domain of interest (the domain exterior to the scattering object). An edge of a convex polyhedron is visible from a point if that point is in front of at least one of the two planes that define the edge.
Then, for each visible edge, the integral is computed numerically. Special care needs to be applied when the receiver is exactly at a zone boundary, which is the plane at which the direct sound visibility factor, or a specular reflection visibility factor has a jump. See also the discussion in 2.4. One way to handle these cases is described for TD computations in [?], and it is used, with small modifications, also for the FD computations.

3.4 Second- and higher-order diffraction, with the ESIE method

Second- and higher-order diffraction is computed in the FD with the ESIE method described in [?]. The computational cost for the ESIE method has a dependence on diffraction order like:

$$t_{\text{comp.,HOD}} = A + B \cdot n_{\text{diffr.order}} \tag{2}$$

which means a *linear* dependence on diffraction order. Therefore, almost arbitrarily high diffraction orders can be computed. The diffraction order needs to be set manually, and it should simply be set "high enough" so that the total sound pressure amplitude is not affected by adding higher orders. In future versions, an automatic truncation could be implemented.

The computational cost, and accuracy, is strongly affected by the number of discretization points along each edge. If $n_{\text{edge points}}$ denotes the total number of discretization points along the edges, then

$$t_{\text{comp.,HOD}} \propto n^3_{\text{edge points}}$$

It should be understood that the parameter B in Eq. (2) is $\propto n^3_{\text{edge points}}$, that is:

$$t_{\text{comp.,HOD}} = A + b \cdot n_{\text{diffr.order}} \cdot n^3_{\text{edge points}}$$

The ESIE method uses an integral equation formulation, and the solution to the integral equation is computed with the Nyström, or quadrature, method. Gauss-Legendre quadrature is used, which gives rapid convergence for the solution, but for problematic source and/or receiver positions, singularities affect the convergence of the higher-order diffraction, see [?] and section 3.6.
A TD-version of the ESIE method was presented in [?] but it has not been implemented in the EDtoolbox as of v. 0.400.

3.5 Second- and higher-order diffraction, with the "separate diffraction orders" method

In [?], a TD formulation of second-order diffraction was presented, with a two-dimensional integral, corresponding to an integration over the points along two edges. In the EDtoolbox, this is extended to third-, fourth-, etc up to sixth-order diffraction, where the latter corresponds to a six-dimensional integral. These multi-dimensional integrals are computed with the mid-point method in the EDtoolbox, and the total number of edge points, $n_{\text{edge points}}$, directly affects the computational cost. In addition, each diffraction order has to be computed separately, so that the cost for the highest diffraction order is approximately

$$t_{\text{comp.},n_{\text{diffr. order}}} = \alpha + \gamma \cdot n_{\text{edge points}}^{n_{\text{diffr.order}}},$$

where α and γ are some constants. This corresponds to exponential growth (as function of diffraction order), which makes the computation of very high diffraction orders prohibitively expensive.
A FD formulation could easily be written down and implemented for this "separate diffraction orders method" but it would be practically useless because multidimensional integrals with oscillating integrands will be much more expensive than the TD counterparts. FD diffraction with the "separate diffraction orders" method has not been implemented (beyond first order) in the ED toolbox.

3.6 Numerical challenges

The edge diffraction-based modeling of scattering, as illustrated in Fig. 1, has some numerical challenges that lead to slow convergence of the underlying diffraction integrals, and ultimately singularities, for specific source and/or receiver positions. The reason for these challenges is the fact that diffraction components must compensate for the discontinuities of the geometrical acoustics components - and as a result, the diffraction component also get discontinuous in space. In the following, we comment on these discontinuities, or jumps, for three types of components.

An additional numerical challenge exists: whenever a source or receiver comes very close to an edge, the diffraction integrals get challenges as well.

3.6.1 The discontinuities of the GA components

The GA components are Dirac pulses in the time domain for the Neumann and Dirichlet BCs, and with simple frequency-domain expressions as well. As described in sections 3.1 and 3.2, the discontinuties are handled via the visibility factors, V, which are straightforward to handle - they act as simple scaling factors that take values 0 and 1, and a few inbetween.

Since an ideal monopole source is assumed in the derivations of the relationships, a receiver can not be placed at a distance zero from a source, since an infinite direct sound amplitude would result. The illustration on the front of this manual shows that a piston can be represented by a sum of discrete monopoles, but of course, this representation breaks down if the receiver comes very close to the individual monopoles. It could be perfectly possible to implement other sources such as pistons in more proper ways, but it has not been done yet in the EDtoolbox.

3.6.2 The challenges for the first-order diffraction

Figure 3 illustrates the two zone boundaries for one edge of a polyhedral scattering object: $\mathrm{ZB}_{\mathrm{direct}}$, where the direct sound has a discontinuity, or jump, and $\mathrm{ZB}_{\mathrm{specular}}$, where the specular reflection has a jump. The first-order diffraction integrals get numerically challenging whenever a receiver comes very close to one of those two boundaries, but using the approach in [?], together with Matlab's numerical integration, first-order diffraction can be computed accurately enough that a smooth total field results across zone boundaries.

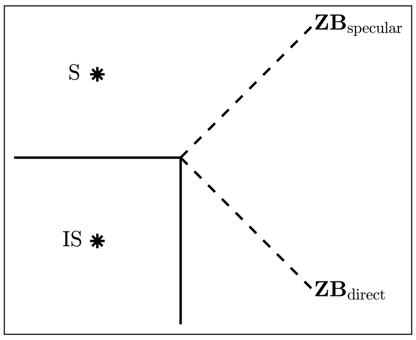

$\mathbf{ZB}_{\mathrm{specular}}$
S
IS
$\mathbf{ZB}_{\mathrm{direct}}$

Figure 3: Illustration of the zone boundaries for the direct sound and specular reflection that are created by a source and a single wedge. At these two zone boundaries, the geometrical acoustics wavefield will have a jump, and the first-order diffraction wavefield has a corresponding, exactly compensating, jump.

The formulations from [?], together with Matlab's numerical integration, handle source and receiver positions very close to edges quite well. Note, however, that a source or receiver can never be placed *exactly* at an edge, because a source and a receiver must create well-defined zone boundaries, and the zone boundary definitions break down for a source exactly at an edge.

3.6.3 The challenges for second- and higher-order diffraction

The edge source integral equation leads to numerical challenges that are related to zone boundaries, and distances to edges. Whereas the zone boundaries are defined by the source position relative to *a single* edge (and its connected planes), the zone boundaries for higher-order diffraction are defined by pairs of edges: each face/polygon of a polyhedral scatterer is part of an infinite plane, and this infinite plane creates a challenging zone boundary.

1. The source position: if the source is very close to one of the infinite planes defined by the polyhedron faces, then the computation of the source term in the ESIE converges slowly - but not if the source is *inside* the polyhedron face. In Fig. 4 source S_1 is thus unproblematic whereas source S_2 will lead to convergence problems. This challenge could be handled with the so-called *Locally Corrected Nyström* method, as

demonstrated in [?], but it has not been implemented in the EDtoolbox yet.

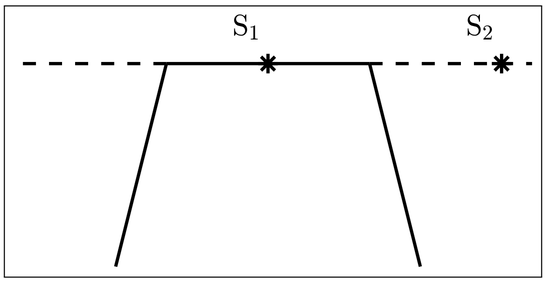


Figure 4: Illustration of the zone boundaries for the source term in higher-order diffraction. Source S_1 is unproblematic whereas source S_2 will lead to convergence problems. Source positions like S_2 could be handled with the so-called *Locally Corrected Nyström* method, which has not been implemented in the EDtoolbox yet.

Also if the source is very close to one edge, similar numerical challenges result.

2. The receiver position: if the receiver is very close to one of the infinite planes defined by the polyhedron faces, then the computation of the propagation integral (using the edge source amplitudes that result from the solving of the integral equation) is inaccurate and requires a finer discretization of the edge source integral equation. A method to overcome this challenge was suggested as the *ESIEBEM* approach in [?]. In the same way as for the source position challenge, a receiver which is *inside* a polyhedral face does not lead to numerical challenges, and this fact is behind the ESIEBEM approach. Fig. 5 illustrates the situation which is exactly corresponding to the source-related challenge in Fig. 4.

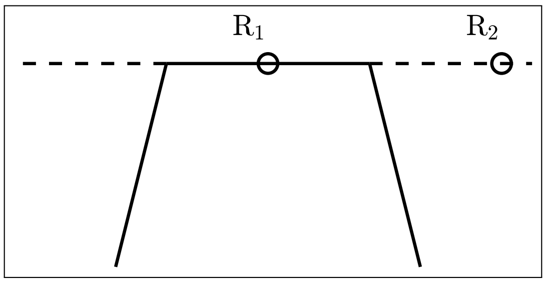


Figure 5: Illustration of the zone boundaries for the receiver in higher-order diffraction. Receiver R_1 is unproblematic whereas receiver R_2 will lead to convergence problems. Receiver positions like R_2 can be handled with the *ESIEBEM* approach, which has been implemented in the EDtoolbox.

3. The wedge angles of the polyhedral scattering object: if the polyhedron has some wedge angles that are close to 180 degrees, that is, if one tries to approximate a smooth surface with a polyhedral approximation, as illustrated in Fig. 6, then the integral equation will converge more slowly.

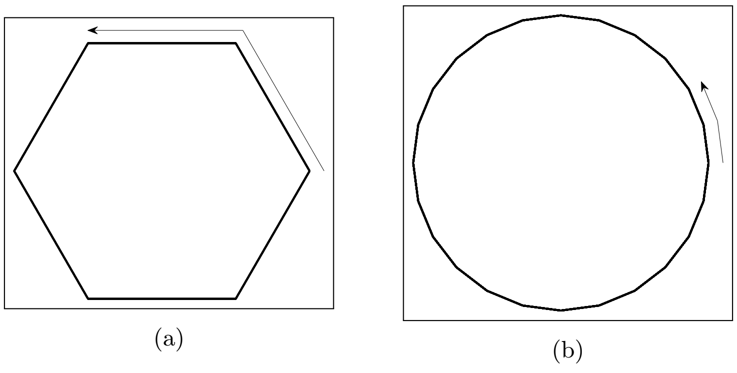


Figure 6: Illustration of a (a) 6-sided cylinder and a (b) 24-sided cylinder. The closer the angle of this edge-to-edge-to-edge path is to 180 degrees, the slower is the convergence of the integral equation.

4 Input data

4.1 General

The main function, `EDmain_convex`, is run with six structs containing all input parameters:

```
EDmain_convexESIE(geoinputdata,Sinputdata,Rinputdata,envdata,...
controlparameters,filehandlingparameters)
```

These six structs are described in the following subsections and in Tables 3 - 8. A function, `EDcheckinputstructs`, can be used to check these six structs and returns all six, with field values checked, and completed with default values, if needed. The function is used like this:

```
   [geoinputdata,Sinputdata,Rinputdata,envdata,controlparameters,filehandlingparameters] = ...
EDcheckinputstructs(geoinputdata,Sinputdata,Rinputdata,envdata,controlparameters,filehandlingparameters)
```

The main function, `EDmain_convexESIE`, calls `EDcheckinputstructs` as one of the first steps.

4.2 Geometry format: `geoinputdata`

The `EDtoolbox` handles only polyhedra, including polygonally shaped thin discs/ plates. A polyhedron is defined here in terms of `corners` (vertices) and `planes` (faces/polygons). There are four ways to specify the geometry[4]:

1. Directly specifying the numerical values into matrices `.corners` and `.planecorners`, which are then given as the fields `.corners` and `.planecorners` of the input struct `geoinputdata`
2. Using the function `EDmakegeo_shoebox` (or other upcoming `EDmakegeo_XXXX` functions), which returns matrices `.corners` and `.planecorners` on the right format, which are then given as the fields `.corners` and `.planecorners` of the input struct `geoinputdata`.
3. As an alternative to point 2 above, you can write your own Matlab function which generates matrices `.corners` and `.planecorners` on the required format, and give them as the fields `.corners` and `.planecorners` of the input struct `geoinputdata`
4. Specify a file name for a file with a supported geometry file format as the field `.geoinputfile` of the input struct `geoinputdata`. As of version 0.400, the .cad-format, a text file format exported by the CATT-Acoustic software [?], is the only supported file format. The EDtoolbox will then internally read the file and convert to the `.corners` and `.planecorners` matrices.

Fig. 7 shows a simple example: a shoebox-shaped box.

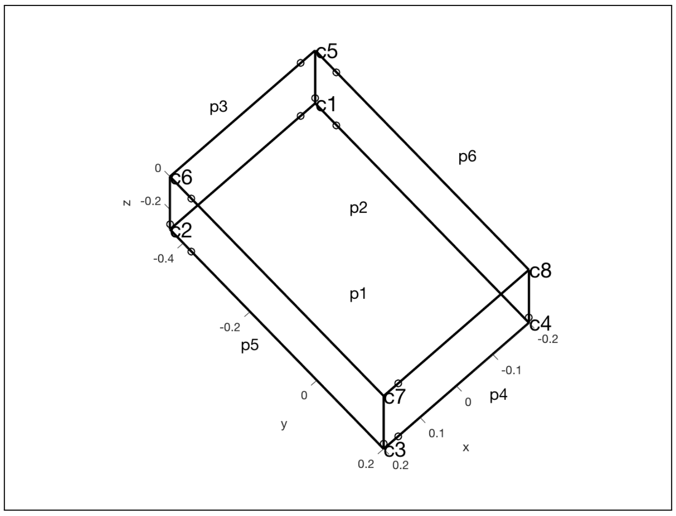


Figure 7: Illustration of a shoebox-shaped scattering object. Corner numbers and plane numbers are indicated.

4.2.1 Corners

The `.corners` field is straightforward: it should be a matrix of size `[ncorners,3]` where row n contains the x-, y- and z-coordinates of corner number n. For the example in Fig. 7, this matrix would have the first few lines as

```
geoinputdata.corners = [−0.2  −0.44 −0.32;...
0.2  −0.44 −0.32; 0.2 0.2 −0.32; ...
```

If the geometry of the scattering object is instead defined in a .cad-file, see section 4.2.3, the `.corners` field will be created by the function `EDreadcad`, which is called automatically inside `EDmain_convex` if a cad-file has been specified as input. The corner numbers in the cad-file will be the same in the `.corners` field. However, if the cad-file had a non-contiguous numbering of the corners, or the cad-file corner numbers did not start with 1, a renumbering will be done for the EDtoolbox, starting with number 1 and giving a contiguous list.

[4]Note that if the field `geoinputdata.freefieldcase` is defined and given the value 1, then no scattering object is defined.

4.2.2 Planes

The `.planecorners` field should be a matrix of size `[nplanes,nmaxcp]`, `nmaxcp` standing for "nmaxcornersperplane", where row n gives the corners that define plane n.
The example in Fig. 7 would have its planes defined as

```
geoinputdata.planecorners = [1 4 3 2;5 6 7 8; ...
```

Three different boundary conditions are possible to specify[5], by specifying the reflection factor. These are:

- Reflection factor 1 corresponds to a Neumann boundary condition, that is, a perfectly rigid surface. This is the default value which results if one does not specify a field `geoinputdata.planerefltypes`.
- Reflection factor 0 is not a well-defined boundary condition here, but it turns off a specular reflection completely, and it also turns off the edge diffraction for all edges that connect to this plane. This might be the most useful feature of this value.
- Reflection factor -1 corresponds to a Dirichlet boundary condition, that is, a "soft", or a "pressure-release" surface. This has been implemented to a limited extent (specular reflections, and first-order diffraction)

A few important rules must be followed:

- The corners must be defined in a counter-clockwise order, as seen from the frontal side of the plane. You can use a right-hand rule: if you place your right hand on the frontal side of the surface, with your thumb pointing in the direction of the (imagined) plane normal vector, than your curved fingers should indicate the order to specify the corners.
- Please note that for thin planes, both sides of the plane must be specified.
- If not all planes have the same number of corners, you must add zeros to the end of each row with fewer corners, so that each row gets the same number of values[6].
- Some geometry generating software splits up polygons into triangles, but `EDtoolbox` can not handle co-planar triangles. As large polygons as possible must be constructed for each face of the polyhedron.

4.2.3 The cadfile format

The cadfile format is a very simple format defined by the CATT-Acoustic software. It is a textfile with four sections, marked with textlines `%CORNERS`, `%PLANES`, `%SOURCES`, `%RECEIVERS`. For the use in the EDtoolbox, only the first two are used. Thus, the two sections should have the format given below, exemplifying for the same box as in Fig. 7 (the first line is optional but quite useful):

```
%LSP_Kessel.CAD
%CORNERS
   1   −0.20   −0.44   −0.32
   2   0.20   −0.44   −0.32
   3 ...

%PLANES
  1 / /RIGID
  1  4  3  2

  2 / /RIGID
  5  6  7  8
```

4.2.4 Excluding some edges

There are four possibilities to create geometrical models where only some of all the edges are included in the calculations (see also Table 3 for more information) :

1. If a plane is given the material type `TOTABS` in a CAD input file, or assigned a refltype = 0 using the optional parameter optional parameter `geoinputdata.planerefltype`, then all edges that belong to that plane are deactivated.

[5] as of version 0.300

[6] This is a consequence of using Matlab's matrix variable type for specifying planecorners. The toolbox could have choosen a cell structure instead, which could allow different numbers of elements for each plane number, but the simpler matrix format was chosen for `EDtoolbox`

2. The optional parameter `geoinputdata.listofedgestoskip` can be single-column or double-column list. A single-column list is interpreted as a list of edge numbers which should be deactivated. Please note that the edge numbers are not known before `EDmain_convex` has been run once. The edge numbers can be found using the function `EDplotmodel`. If `geoinputdata.listofedgestoskip` is instead specified as a two-column list, then each row in that list is interpreted as a corner pair, and the edge which connects those two corners will be deactivated. This parameter, `geoinputdata.listofedgestoskip`, takes precedence over the two following optional parameters.

3. If the optional parameter `geoinputdata.firstcornertoskip` is given a value, then all edges with at least one corner which has a number like that parameter value, or higher, will be deactivated.

4. If the optional parameter `geoinputdata.listofcornerstoskip` is given a value or a list of values, then all edges with at least one corner which has a number that belongs to that list, will be deactivated.

Using these techniques, it is possible to simulate, e.g., just the top edges of a noise barrier.

Table 3: Input data struct `geoinputdata`

Field name[1]	**Description**
`.geoinputfile`	(Optional) This field should be given the file name (with path) of a .cad-file
`.corners`	(Optional) Format as in Section 4.2.1
`.planecorners`	(Optional) Format as in Section 4.2.2
`.planerefltypes`	(Optional) Allowed values are 1 (default), 0, and -1.
`.listofedgestoskip`	(Optional) As described in Section 4.2.4, some edges can be deactivated by using this parameter. Default value [] (empty).
`.firstcornertoskip`	(Optional) As described in Section 4.2.4, some edges can be deactivated by using this parameter. Default value 1e6.
`.listofcornerstoskip`	(Optional) As described in Section 4.2.4, some edges can be deactivated by using this parameter. Default value [] (empty).
`.planeseesplanestrategy`	(Optional) This parameter is irrelevant for convex-shaped scattering objects but is preparing for non-convex objects. If this parameter is given the value 1, then a plane-to-plane visibility check is done by checking plane-midpoint to plane-midpoint visibility. Default value 0.
`.freefieldcase`	(Optional) If this parameter is given the value 1, then a free-field case will be run and no scattering object will be defined. Default value 0.

[1] Four alternatives exist for specifying the struct `geoinputdata`
A. An external `.cad`-file is specified in the field `.geoinputfile`
B. If the field `.geoinputfile` is not specified, then the fields `corners` and `planecorners` can give the geometry data.
C. If neither of the two alternatives above apply (e.g., if the entire struct is left empty), then a file opening window will appear, and a `.cad`-file can be selected.
D. If both alternatives A and B are given, priority will be given to the `.geoinputfile`.
See section 4.2 for more information on the geometry format.

4.3 Source specification: `Sinputdata`

As of v0.400 of `EDtoolbox`, monopoles or polygonal pistons can be used as sources. If more than one source is specified (one polygonal piston counts as one source), one can choose to add their contributions or save the individual sources' transfer functions/impulse responses. As suggested in Section 2.2, a plane wave can be emulated by setting a monopole very far away, at a distance of 10^6 m or so, and then specify the `Sinputdata.sourceamplitudes` to be "distance" instead of the default value 1. Furthermore, one can set the phase of the incident sound field to be zero at the origin by setting `controlparameters.Rstart = 0` - but not for impulse responses since that would generate very long impulse responses.

As described in section 2.2, the scaling of the resulting transfer functions and impulse responses is such that the amplitude, from a monopole in free-field, at a distance r m is $1/r$. That means that the quantity $\mathrm{j}\omega\rho_0 U_0/(4\pi)$ of the source is 1 kg/s^2. The polygonal piston gets an amplitude which corresponds to a monopole at an infinite baffle, which means that when `.sourceamplitudes` $=1$ (the default), then the quantity $\mathrm{j}\omega\rho_0 u_{\mathrm{piston}} S_{\mathrm{piston}}/(2\pi)$ $=1$, which means that $u_{\mathrm{piston}} = 2\pi/(\mathrm{j}\omega\rho_0 u_{\mathrm{piston}} S_{\mathrm{piston}})$.

Piston sources

For piston sources, two different computational approaches are used for the direct sound and for the diffraction components:

- For the direct sound, the Rayleigh integral gives the radiated sound pressure and here, the TD-formulation of the piston edge wave approach in [?] is used here, with an adaptation to the FD. By using the edge wave approach, *it is unproblematic to put receivers immediately at the piston surface.*
- For the first- and higher-order diffraction terms, the piston is replaced by a number of quadrature points, and how they are distributed differs between triangular, rectangular, and regular pistons, see details in Table 4. For the diffraction terms it is also unproblematic to place receivers immediately at a piston surface.

Table 4: Input data struct `Sinputdata`

Field name	**Description**
`.coordinates`	(Required) A matrix of size [nsources,3], giving the x-, y-, and z- coordinates of each source.[1]
`.doaddsources`	(Optional) If this value is set to 1, the contributions from all sources will be added and saved in a single transfer function or impulse response, after being multiplied by the values in the matrix `.sourceamplitudes`.[2]Default value 0.
`.sourceamplitudes`	(Optional) A matrix of amplitudes, size `[nsources,nfreq]`, that each source is multiplied with. Using this, together with `.doaddsources`=1, a vibration pattern on a surface can be simulated. [3]Default value `ones(nsources,nfreq)`.
`.doallSRcombinations`	(Optional) If n sources and n receivers are specified, one can choose to compute the response only for source 1 to receiver 1, source 2 to receiver 2, etc by setting this parameter to 0. This is relevant for computing monostatic backscattering. Default value 1.
`.sourcetype`	(Optional) Can be either `monopole` (default) or `polygonpiston`.[4]
`.pistoncornercoordinates`	(Optional) but (Required) if `.sourcetype` is `polygonpiston`. Format as for `geoinputdata.corners`, that is, x, y, z of all `npistoncorners` piston corners, stored in a matrix of size [`npistoncorners`,3]. Please note that these coordinates must be given values that fulfill the plane equation of the plane specified in the field `.pistonplanes`. The EDtoolbox does not (as of version 0.400) check this for you.
`.pistoncornernumbers`	(Optional) but (Required) if `.sourcetype` is `polygonpiston`. Matrix of size [`npistons`,`maxncornersperpiston`], where each row contains the 3,4,5,.... corner numbers for each piston. The order must follow the same right-hand rule as the planecorners: if you put your right hand to point in the direction of the plane and piston normal vector, then your curved fingers should point in the direction of the corner order. This can also be expressed as counterclockwise order when one is looking at the piston from the outside of the scattering object. Please note that if the pistons have different numbers of corners, the matrix in `.pistoncornernumbers` must get zeros at the end of each row which has fewer than `maxncornersperpiston` values.
`.pistonplanes`	(Optional) but (Required) if `.sourcetype` is `polygonpiston`. List of size [`npistons`,1], which gives the plane number (defined in the list `geoinputdata.planecorners`) that each piston belongs to.[5]
`.pistongaussorder`	(Optional); only relevant if `.sourcetype` is `polygonpiston`. A number which will have slightly different meanings for different piston types. The default value is 3. • For a rectangular piston, then `.pistongaussorder` is the number of gauss quadrature points along one dimension of each piston. The total number of gauss quadrature points is thus `.pistongaussorder`^{2}. • For a triangular piston, then `.pistongaussorder` with values [1, 2, 3, 4, 5, 6, 7, 8] gives, respectively, [1, 3, 4, 6, 7, 12, 16, 16] quadrature points, according to the numerical values given in [?] and [?]. • For a regular polygon (approximating a circle), quadrature points will be distributed in `.pistongaussorder` concentric circles with linearly distributed radii. For each radius, the number of quadrature points is [1, 8, 16, 24, 32, ...] so that for `.pistongaussorder` = [1, 2, 3, 4, 5, ...], the total number of points is, respectively, [1, 9, 25,49, 81, ...].

[1] If a source is placed at a surface, it needs to be placed a tiny distance away from the surface, say 10^{-5} m.
[2] This is a straightforward way to simulate extended sources, or vibration patterns. See section 4.8 for a description of the scale values.
[3] To simulate an incoming plane wave with amplitude 1 at the origin, then `.sourceamplitudes` should be given the value "distance", where "distance" is the distance to the far-away monopole source. See also the description for the input data struct `controlparameters` and section 2.2.
[4] The term 'polygonpiston' is used to give the possibility to define 'circularpiston' in future versions.
[5] Why does one have to specify which plane number that each piston belongs to - can't the EDtoolbox find out that by itself? Yes, it could but this way mistakes can be avoided when a piston belongs to a thin plane.

4.4 Receiver specification: `Rinputdata`

Any number of point receivers can be specified in a long list of x-,y-,z-coordinates. In order to make an animation, one could create a list of receivers which together form a grid of receivers in one plane. By computing impulse responses, and plotting one time sample impulse response value *for all receiver positions*, an animation is straightforward to generate. No code for this kind of postprocessing is presented here, however. A tip is to first convolve all impulse responses with some kind of low-pass filter, where the cut-off frequency is linked to the grid point spacing.

Table 5: Input data struct `Rinputdata`

Field name	**Description**
`.coordinates`	(Required) A matrix of size [nreceivers,3], giving the x-, y-, and z- coordinates of each receiver.[1]

[1] If a receiver is placed at a surface, it needs to be placed a tiny distance away from the surface, say 10^{-5} m.

4.5 Environmental data: `envdata`

Table 6: Input data struct `envdata`

Field name	Description
`.cair`	(Optional) Should be the speed of sound in m/s. Default value 344.
`.rhoair`	(Optional) Should be the density of the medium, in kg/m^3. Default value 1.21.

[1] The air density is not used in any of the calculations in the current version (0.300) of the EDtoolbox, but might be used in future versions.

4.6 Calculation settings: `controlparameters`

Table 7: Input data struct `controlparameters`

Field name	**Description**
One of the three following parameters must be set to one:	
`.docalctf`	Determines if transfer functions will be computed, with the ESIE method for higher-order diffraction.[*]
`.docalctf_ESIEBEM`	Determines if transfer functions will be computed, with the ESIEBEM method.[*]
`.docalcir`	Determines if impulse responses will be computed, with the "separate diffraction orders" method.[*]
`.frequencies`	(Required if `.docalctf = 1` or `.docalctf_ESIEBEM`=1). A vector of frequency values that computations will be made for.
`.directsound`	(Optional) If this parameter is set to 0, the direct sound will not be computed and the result variable `tfdirect` or `irdirect` will be empty. Default value 1.
`.skipfirstorder`	(Optional) If this parameter is set to 1, the direct sound, specular reflection, and first-order diffraction will not be computed and their respective output variables will be empty. Default value 0.
`.Rstart`	(Optional) Determines the phase of the final transfer functions (or the definition of the time zero in TD calculations).[1] Default value 0.
`.difforder`	(Optional) Specifies how many orders of diffraction should be included.[2]Default value 10.
`.discretizationtype`	Determines how the edges will be discretized: 0 $\Rightarrow$ a uniform discretization. 2 $\Rightarrow$ Gauss-Legendre discretization. The value 1 is obsolete/not used. Default value 2.
`.ngauss`	Specifies the number of quadrature points along the longest edge, for the ESIE method. It will be scaled down linearly based on the length of each edge, with a minimum of 2. Recommended value is at least 3 quadrature points per wavelength, which has to be converted manually to a number of quadrature points. Default value 16.
`.surfacegaussorder`	For the ESIEBEM method, this parameter determines how many intermediate receivers will be placed across each plane. A value of 5 implies 5*5 surface receivers for the largest plane, and progressively fewer for smaller planes. Default value 5.
`.fs`	Determines the sampling frequency. Default value 44100.
`.savealldifforders`	If this parameter is set to 1, then results are stored for each diffraction order. Default value 0.

[*] One of these three must be set to one, and only one can be set to one.
[1] To simulate an incoming plane wave with amplitude 1, and phase zero, at the origin, then `.Rstart` should be set to the distance to the far-away monopole source. See also the description for the input data struct `Sinputdata` and the description in Section 2.2.
[2] For time-domain (impulse response) calculations, `difforder` can not be higher than 6.

4.7 File-related settings: `filehandlingparameters`

Table 8: Input data struct `filehandlingparameters`

Field name	**Description**[1]
`.outputdirectory`	(Required). All result files will be saved in this directory. If not specified, a folder named "results" will be made in the folder of the .cad-file.
`.filestem`	(Required). All result files will start with this text string. If not specified, this field will be given the name of the .cad-file.
`.suppressresultrecycling`...	(Optional) 0 ⇒ all result files will be inspected for possible recycling[2] Default value 0.
`.savecadgeofile`	(Optional) 1 ⇒ the contents of the .cad-file will be saved in a `_cadgeo.mat` file. Default value 0.
`.saveSRdatafiles`	(Optional) 1 ⇒ the visibility of planes and edges, as seen from sources and receivers, is stored in `_Sdata.mat` and `_Rdata.mat` files. Default value 1.
`.saveeddatafile`	(Optional) 1 ⇒ the `edgedata` struct is saved in an `_eddata.mat` file. Default value 1.
`.saveed2datafile`	(Optional) 1 ⇒ the `edgetoedgedata` struct is saved in an `_ed2data.mat` file. Default value 1.
`.savesubmatrixdata`	(Optional) 1 ⇒ the `submatrixdata` struct is saved in a `_submatrixdata.mat` file. Default value 1.
`.saveinteqsousigs`	(Optional) 1 ⇒ the edge source signals are saved in a `_sousigs.mat` file. Default value 0.
`.loadinteqsousigs`	(Optional) 1 ⇒ previously calculated, and saved, edge source signals are loaded and re-used. Default value 0.
`.savepathsfile`	(Optional) 1 ⇒ the lists of possible direct sound, specular reflections, and first-order diffractions, are saved in a `_paths.mat` file. Default value 1.
`.savehodpaths`	(Optional) 1 ⇒ the lists of possible higer-order diffractions are saved in a `_hodpaths.mat` file. Used only by `EDmain_convex_time`. Default value 0.
`.savelogfile`	(Optional) 1 ⇒ a log text-file is saved, see Section 4.9. Default value 1.
`.savediff2result`	(Optional) 1 ⇒ the results for second-order diffraction are saved separately, in the form of the variable `extraoutputdata.tfinteqdiff_nodiff2`. Default value 0.
`.showtext`	(Optional) 1 ⇒ some progression text is printed out on screen. If the value is set to 0, no text is printed out on the screen. If values higher than 1 are set, then even more detailed information is printed out on screen.

[1] See section 4.10 for a further description of the output files and the data stored therein.
[2] If possible, previously computed result files are reused. All the relevant input data to a function is checked, and if the settings for the current calculation match exactly the settings of a previous calculation, the results from the previous calculation will be reused. Note that the toolbox version number will be checked, so if the toolbox version has been changed, all result files will be calculated anew. All the relevant input data are encoded into a hash, which is stored together with the result variables. During the inspection, only this hash is loaded from each available result file, and checked against the hash of the current wanted settings.

4.8 Output data in the `_tf.mat` and `_tf_ESIEBEM.mat` and `_ir.mat` files

4.8.1 The resulting sound pressure/transfer functions

The main function, `EDmain_convex`, can generate several types of optional output files, as indicated in table 8. The primary output is, however, the resulting sound pressure in the form of transfer functions or impulse responses. They are stored as variables in files described in Table 9. These files are always generated.

Table 9: Primary output files and result variables

Main function setting	**Result file**	**Result variables**
`.docalctf = 1`	`_tf.mat`	`tfdirect, tfgeom` `tfdiff`
	`_tfinteq.mat`	`tfinteqdiff`
`.docalctf_ESIEBEM = 1`	`_tfesiebem.mat`	`tftot_ESIEBEM`
`.docalcir = 1`	`_ir.mat`	`irdirect, irgeom,` `irdiff, irhod`

Typically, you don't have to interact with these files yourself since you can use the function `EDgetresults`, which will reurn a struct, `EDres`, with all the relevant result variables. See table 10 for which results are contained in the `EDres` struct. In addition, if `doallSRcombinations` is given the value 0, then the tf-matrices will contain values only along the diagonal. Obviously, this applies only to square matrices, that is, when `nR` = `nS`.
The value of `nirsamples` depends on the geometry of the scattering object, the sampling frequency, and the value of `.Rstart`.

Table 10: Contents of the EDres struct.

If `.docalctf`=1	If `.docalctf_ESIEBEM`=1	If `.docalcir`=1

`.tfdirect`
`.tfgeom`
`.tfdiff`
`.tfinteqdiff`
`.tftot = tfdirect + tfgeom + tfdiff + tfinteqdiff`
`.timingstruct`
	`.tftot_ESIEBEM`
	`.timingstruct`
		`.irdirect`
		`.irgeom`
		`.irdiff`
		`.irhod`
		`.irtot = irdirect + irgeom + irdiff + irhod`
		`.timingstruct`

• if `.doaddsources = 0`, then all `tf` have the size [`nfreq,nR,nS`] • if `.doaddsources = 1`, then all `tf` have the size [`nfreq,nR`]	Same as ←	• if `.doaddsources = 0`, then all `ir` have the size [`nirsamples,nR,nS`] • if `.doaddsources = 1`, then all `ir` have the size [`nirsamples,nR`]

The variable `irhod` contains the higher-order diffraction. This is a cell variable, which has a structure which is determined by the parameter `.savealldifforders`:

$$\texttt{irhod} = \begin{cases} \text{single cell}, \texttt{irhod\{1\}}, & \text{if } \texttt{.savealldifforders = 0} \\ \text{one cell for each diffraction order } \texttt{n}, \texttt{irhod\{n\}}, & \text{if } \texttt{.savealldifforders = 1} \end{cases}$$

Each `irhod{n}` has the size:

$$\texttt{size(irhod\{n\})} = \begin{cases} [\texttt{nirsamples,nR,nS}], & \text{if } \texttt{doaddsources == 0} \\ [\texttt{nirsamples,nR}], & \text{if } \texttt{doaddsources == 1} \end{cases}$$

4.8.2 Timing data

When the function `EDgetresults` is used to load the result files, the output struct `EDres` will have a field `.timingstruct`, which has a number of fields that give the time consumed by the different parts of the function `EDmain_convex`. Table 11 shows some example values from a calculation. The two columns show values for a first calculation and a re-run of the setup script with the same settings, respectively. For a re-run of a calculation, two values might ve given. Then the first value is from the latest run and the second value is the older, previous run.

The function `EDmain_convex` checks, for every sub-function, if a result file already exists, with the correct settings. If a result file already exists, then that particular sub-function will not be re-run. One would expect that a re-run is faster because one or more sub-function will not have to be run. However, for sub-functions that run very fast, the extra overhead involved in checking existing files might make the effort to re-use resutls more time-consuming. Still, for the sub-functions that consume significant amounts of time, the re-run will be substantially faster. See the timing values for the field names `maketfs` and `integralequation`.

Table 11: Output data struct `EDres.timingstruct`, with example timing values

Field name	**Function which is timed**	**Values after a first run**	**Values after a re-run**
`geoinput`	`EDreadcad`, or `EDreadgeomatrices`	0.0040	0.0020
`edgedata`	`EDedgeo`	0.0144	0.0087
`Sdata`	`EDSorRgeo`	0.0084	[0.0099 0.0084]
`Rdata`	`EDSorRgeo`	0.0058	[0.0097 0.0058]
`findpaths`	`EDfindconvexGApaths`	0.0077	[0.0112 0.0077]
`maketfs`	`EDmakefirstordertfs`[1]	0.1562	[0.0105 0.1562]
`makeirs`	`EDmakefirstorderirs`	0	0
`edgetoedgedata`	`EDed2geo`	0.0148	[0.0127 0.0148]
`submatrixdata`	`EDinteg_submatrixstructure`	0.0091	[0.0076 0.0091]
`integralequation`	`EDintegralequation_convex_tf`[1]	32.3621	[0.0134 32.3621]
`esiebem`	`EDpropagateESIEBEM`	0	0
`hodpaths`	`EDfindHODpaths`	0	0
`irhod`	`EDmakeHODirs`	0	0

[1] More detailed information about this timing value can be find in the log file, see Section 4.9.

4.8.3 Other output data in the `_tfinteq.mat` file

The `_tf` and `_tfinteq` files will also contain some more variables:

- `EDdatainputhash`, which is a 32-bit hex character string which uniquely describes all the settings of the relevant input parameters, and EDtoolbox version. It is like a compact fingerprint of the contents in `EDsettings` described above. The settings can, however, not be extracted from this hash; it is just used to check if an existing file can be recycled for a subsequent calculation.

- `recycledresultsfile`, which is either empty, if the calculations have been run, or contains a file-name, where the results have been copied from.

4.9 Log file

If the parameter `filehandlingparameters.savelogfile` is set to 1, a log file will be generated as a plain text file. Its contents will be as below. When some result files have been recycled, the detailed timing data (for the functions `EDmakefirstordertfs` and `EDintegralequation_convex_tf`) will be given for the original files.

```
EDmain_convex, v.0.301 (3Oct2023)
   filestem for results: Lsp_Kessel_minimalexample
   directory for results: /Users/svensson/Documents/LSP_Kessel/results
Calculating transfer functions with the ESIE method

EDreadgeomatrices: 8 corners and 6 planes. Time: 0.001979 s
EDedgeo: 12 edges. Time: 0.012741 s
EDSorRgeo: 1 sources. Time: 0.007599 s
EDSorRgeo: 1 receivers. Time: 0.004871 s
EDfindconvexGApaths: 1 spec. paths and 12 diff. paths. Time: 0.006725 s
EDmakefirstordertfs: 100 frequencies. Time: 0.16571 s
   Direct sound: 0.000283s, spec.refl.: 9.1e-05s, first-order diff.: 0.1605s
EDed2geo:  Time: 0.012971 s
EDinteg_submatrixstructure: 252 submatrices, out of 432, to compute. Time: 0.006255 s
   Edges discretized with: 8-16 gauss points. 9248 edge source signals to compute.
   The IE matrix has 628864 non-zero elements. Some may be identical due to symmetries
EDintegralequation_convex_tf: 100 frequencies. Diffraction order: 10. Time: 28.1315 s
   H-matrix: 0.10742 s. Q_firstterm: 0.006956 s (per freq.)
   Qfinal: 0.25697 s. P at receiver: 0.006164 s.
```

4.10 Other output files

The contents in the optional output files are described in table 12, and some of the structs are further detailed in tables 13 - 18 below. Whether or not these files are generated is set by the various fields of `filehandlingparameters`, as specified in the last column in Table 12 and further in Table 8.

Table 12: Contents of the optional output files

Output file[1]	**Contents**[2]	**filehandling-parameters**
`_cadgeo`	`planedata`, `extraCATTdata`, `elapsedtimecadgeo`[3]	`.savecadgeofile`
`_eddata`	`planedata`, `edgedata`, `elapsedtimeedgeo`[3], `EDinputdatahash`[4]	`.saveeddatafile`
`_Sdata, _Rdata`	`Sdata` or `Rdata`, `elapsedtimeSgeo` or `elapsedtimeRgeo`[3], `EDinputdatahash`[4]	`.saveSRdatafiles`
`_ed2data`	`planedata`, `edgedata`, `edgetoedgedata`, `elapsedtimeed2geo`[3], `EDinputdatahash`[4]	`.saveed2datafile`
`_paths`	`firstorderpathdata`, `elapsedtimefindpaths`[3], `EDinputdatahash`[4]	`.savepathsfile`
`_submatrixdata`	`Hsubmatrixdata`, `elapsedtimesubmatrix`[3], `EDinputdatahash`[4]	`.savesubmatrixdata`

[1] The output files will all start with the text label in `filestem`, followed by an underscore and a label, as given in this table.
[2] The contents will all be structs unless described by footnote 3 or 4.
[3] One or two numeric values.
[4] The text string in `EDinputdatahash` will be determined from different subsets of all input parameter settings for each output file.

As one example of an extra result file, the `edgedata` struct can be mentioned. Internally in the calculations, edges are generated as illustrated in Fig. 8, by the function `EDedgeo`. The edge numbers, and other edge-related parameters, are not immediately relevant for the end results, but can be found in the optional output file `xxx_eddata.mat`.

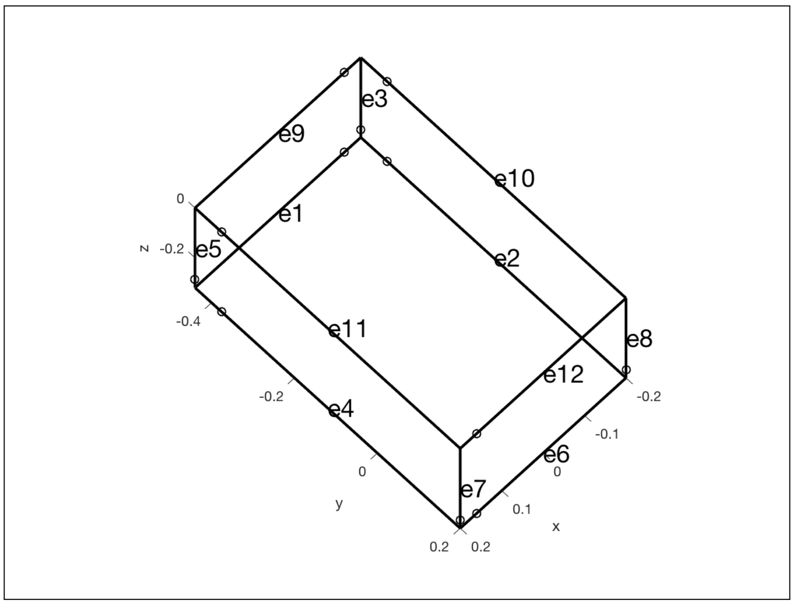


Figure 8: Illustration of a shoebox-shaped scattering object, with the derived edge numbers indicated. For each edge, the little circle indicates the starting end.

Table 13: The `planedata` struct, generated by `EDreadcad` or `EDreadgeomatrices`

Field name	**Size**	**Description**
`.corners`	[ncorners,3]	x, y and z for each corner, taken from input data.
`.planecorners`	[nplanes,nmax]	For each plane, the corner numbers that define the plane, taken from input data. Zeros fill each row to give nmax columns, where nmax is the max number of corners per plane.
`.planeabstypes`	sparse([nplanes,nn])	For each plane, its absorber type. The values are either taken from the cad-file (in `EDreadcad`), or given the value 'RIGID' for each plane (in `EDreadgeomatrices`).
`.planeeqs`	[nplanes,4]	For each plane, the A, B, C, D parameters of its plane equation on the form $Ax + By + Cy = D$ where A, B, C are normalized to give the plane normal vector.
`.ncornersperplanevec`	[nplanes,1]	The number of corners per plane.
`.minvals` `.maxvals`	[nplanes,3] [nplanes,3]	$[\min(x_i), \min(y_i), \min(z_i)]$ $[\max(x_i), \max(y_i), \max(z_i)]$ These min- and max-values give each plane's "axis-aligned bounding box (AABB)" .
`.planehasindents`	[nplanes,1]	For each plane, 0 or 1, telling if the plane has any indents.
`.indentingcorners`	[nplanes,nmax]	For each plane, the number to the first corner in a corner triplet which identifies an indenting corner. The number of the corner is the order given in `planecorners` for that plane.
`.cornerinfrontofplane`	[nplanes,ncorners]	-1, 0 or 1. These values specify: 1 means that a corner is in front of the plane 0 means that a corner is aligned with the plane, including belonging to the plane -1 means that a corner is behind the plane
`.modeltype`	—	'convex_ext' or 'convex_int' or 'singleplate' or 'thinplates' or 'other'

Table 14: The fields added to the `planedata` struct by `EDedgeo`

Field name	**Size**	**Description**
`.planeisthin`	[nplanes,1]	For each plane, 0 or 1, telling if the plane is thin
`.planeseesplane`	[nplanes,nplanes]	For each plane-plane pair, -2,-1,0,1: 1 means that a plane is in front of the other plane, butobstruction has not been checked 0 means that a plane is completely behind the other plane -1 means that a plane is aligned with the other plane -2 means that a (thin) plane is back-to-back with the other (thin) plane
`.rearsideplane`	[nplanes,1]	For each plane, the number of the plane on the other side. Only relevant for thin planes.
`canplaneobstruct`	[nplanes,1]	States whether or not a plane has the potential to obstruct (in a plane-to-plane path; which is irrelevant for external convex problems)
`.reflfactors`	[nplanes,1]	-1 (SOFT), 0 (TOTABS), 1 (RIGID)

Table 15: The `edgedata` struct, generated by `EDedgeo`

Field name	**Size**	**Description**
`.edgecorners`	[nedges,2]	For each edge, a starting corner (column 1) and an ending corner (column 2). This direction is maintained through all calculations.
`.closwedangvec`	[nedges,1]	For each edge, the "closed wedge angle" is given, in radians. For a 90-degree external corner of a scattering box, the value would be $\pi/2$.
`.edgestartcoords`	[nedges,3]	For each edge, the x,y,z of the starting corner. Redundant; could have been found from `planedata.corners(edgecorners(:,1),:)`
`.edgeendcoords`	[nedges,3]	For each edge, the x,y,z of the starting corner. Redundant; could have been found from `planedata.corners(edgecorners(:,2),:)`
`.edgelengthvec`	[nedges,1]	For each edge, the edge length. Redundant, could have been computed from the starting and ending coordinates for each edge.
`.offedges`	[noffedges,1]	A list with all the edges that are not active; either because they have an open angle of 90 degrees (or 60 or 45 or ...), or because `firstskipcorner` has been given a value which turns off some corners, and therefore edges.
`.edgenvecs`	[nedges,3]	For each edge, the normal vector of its reference plane/face. The ref. plane is defined by a right-hand rule: if the right-hand thumb is aligned with the direction of the edge (as defined by the two corners in`.edgecorners`), then the right-hand fingers will come out of the reference plane, and thereby point in the direction of the normal of the reference plane.
`.edgenormvecs`	[nedges,3]	For each, a normalized vector in the direction of the edge.
`.edgestartcoordsnudge`	[nedges,3]	Same as `.edgestartcoords`, but moved a short distance away from the edge start point.
`.edgeendcoordsnudge`	[nedges,3]	Same as `.edgeendcoords`, but moved a short distance away from the edge endpoint.
`.edgerelatedcoord...` `sysmatrices`	[nedges,9]	For each edge, 9 values that form a matrix which can be used to compute the coordinates of points in the edge-related coordinate system. Each row has to be reshaped: `Bmatrix =reshape(edgerelatedcoordsys...` `matrices(edgenumber,:),3,3);`
`.indentingedgepairs`	[nn,2]	A list of edge pairs that form an indent.
`.planesatedge`	[nedges,2]	For each edge, the two connected planes.
`.edgesatplane`	[nplanes,nmaxcp]	For each plane, the connected edges. The value `nmaxcp` is the largest number of edges for one plane.
`.edgeseesplane`	[nplanes,nedges]	-2,-1,0 or 1 For each plane-edge pair: -2 means that the edge belongs to the plane -1 means that the edge is aligned with the plane, but does not belong to it 0 means that the edge is completely behind the plane 1 means that the edge has at least some point in front of the plane, but obstruction has not been checked

Table 16: Output data struct `Sdata`, generated by `EDSorRgeo`

Field name[1]	**Size**	**Description**[1]
`.sources`	[nsources,3]	For each source, the x-, y-, z-coordinates[2]
`.visplanesfroms`	[nplanes,nsources]	For each plane-source pair: 0: S is behind a plane which is RIGID or TOTABS, 1: S is aligned with a plane, but outside the polygon that defines the plane, 2: S is in front of a plane which is reflective (which means that a specular reflection is possible), 3: S is in front of a plane which is TOTABS (which means that a specular reflection is not possible), 4: S is inside a plane which is RIGID, 5: S is inside a plane which is TOTABS
`.vispartedgesfroms`	[nedges,nsources]	For each edge-source pair, a value 0-$2^n - 1$. The value n is an integer for checking part-visibility of an edge. This visibility check is irrelevant for convex-shaped objects, but has been used in previous toolbox versions. 0: S can not see any part of the edge $2^n - 1$: S can see the entire edge other: S can see parts of an edge
`.soutsidemodel`	[1,nsources]	For each source, 0 or 1, specifying if the source is outside the entire model. This is used in cases where a huge grid of sources, or receivers, is generated with some of them being outside the domain of interest.
`.vispartedgesfroms_start`	[nedges,nsources]	For each edge-source pair: 0: S can not see the start-point of the edge 1: S can see the start-point of the edge
`.vispartedgesfroms_end`	[nedges,nsources]	For each edge-source pair: 0: S can not see the end-point of the edge 1: S can see the end-point of the edge
`.reftoshortlistS`	[nedges,nsources]	For each edge-source pair, a pointer to the 3 following short lists [3]
`.rSsho`	[nshortlist,3]	A list of unique values of rS [3]
`.thetaSsho`	[nshortlist,3]	A list of unique values of thetaS [3]
`.zSsho`	[nshortlist,3]	A list of unique values of zS [3]

[1] An equivalent version exists for receivers, with field names `.receivers, .visplanesfromr` etc.
[2] Direct copy from `Sinputdata.coordinates`
[3] Instead of storing all nedges*nsources values of rS,thetaS,zS, three more compact "short lists" are stored, with unique values, and nedges*nsources pointers to these short lists. Thus, the r-value of source 2, rel. to edge 7, is found as `rSsho(reftoshortlistS(7,2))`.

Table 17: Output data struct `firstorderpathdata`

Field name	**Size**	**Description**
`.specreflIScoords`	[nIS,3]	A list of all valid image sources (IS). For each IS, its x-, y- z-coordinates
`.specrefllist`	[nIS,3]	For each IS, three values: S,R,amp. S is the generating source, R is the receiver number, and amp is an amplitude factor.
`.diffpaths`	[nreceivers,nsources,... nedges]	For each receiver-source-edge triplet: 0 or 1, specifying whether or not that edge is visible
`.edgeisactive`	[nedges,1]	For each edge: 0 or 1, specifying whether or not that edge is at all active
`.directsoundlist`	[ndircombs,3]	A list of valid direct sound components. For each such component, three values: S,R,amp. S is the generating source, R is the receiver number, and amp is an amplitude factor.
`.ncomponents`	[1,3]	A list of number of components: [ndircombs,nIS,ndiffrcombs]

Table 18: Output data struct `Hsubmatrixdata`

Field name	Size	Description
`.nedgeelems`	[nedges,1]	For each edge, the number of edge elements = quadrature order.
`.edgepairlist`	[nedgepairs,2]	A list of all edge-sees-edge pairs of the polyhedron. Column 1 has the "to-edge" numbers and column 2 has the "from-edge" numbers. The edge source signals are stored in the edge source signal vector following this order.
`.bigmatrixstartnums`	[nedgepairs,1]	For each edgepair in `.edgepairlist`, this vector gives the start index in the edge source signal vector. Each edgepair occupies a number of elements, corresponding to the product of the values of `ngauss` for the two involved edges.
`.bigmatrixendnums`	[nedgepairs,1]	For each edgepair in `.edgepairlist`, this vector gives the end index in the edge source signal vector.
`.isthinplaneedgepair`	[nedgepairs,1]	For each edgepair in `.edgepairlist`, 0 or 1, indicating if the path from the "from-edge" to the "to-edge".
`.Hsubmatrix`	[nedgepairs,nedgepairs]	A matrix with pointers, which for row m, column n, gives the number of the separately stored Hsub matrix which has the transfer matrix elements for the transfer from edgepair number n, to edgepair number m. Scattering objects with symmetries will have a number of identical pointer values, indicating that not all submatrices need to be computed. There are nsub unique submatrices.
`.listofsubmatrices`	[nsub,3]	
`.edgetripletlist`	[nsub,3]	edgenumbers
`.submatrixcounter`		nsub
`.nuniquesubmatrices`		nuniquesub
`.reftoshortlist`	[nsub,1]	
`.isthinplanetriplet`	[nsub,1]	0 or 1
`.quadraturematrix_pos`	[ngauss,ngauss]	sparse matrix
`.quadraturematrix_...` `weights`	[ngauss,ngauss]	sparse matrix

5 Some auxiliary functions

In the following subsections, the help text of some of the auxiliary functions is given.

5.1 EDplotmodel

```
 EDplotmodel — Plots a model which is given in an eddatafile.

 Input parameters:
   eddatafile (optional)   If an input file is not specified, a file
                           opening window will be presented.
   plotoptions (optional)
                           The text–string 'plotoptions',with an integer can give extra options:
                           if bit0 = 1 (= 1) => plot sources
                           if bit1 = 1 (= 2) => plot Rdata.receivers
                           if bit2 = 1 (= 4) => plot plane normal vectors
                           if bit3 = 1 (= 8) => print plane numbers
                           if bit4 = 1 (=16) => print edge numbers (and
                           indicate start end each edge with a little
                           circle)
                           if bit5 = 1 (=32) => print corner numbers
                           if bit6 = 1 (=64) => print plane numbers using the
                                                CAD file numbering
                           if bit7 = 1 (=128)=> print corner numbers using the
                                                CAD file numbering
                           Example: the integer 11 = 1011 binary,
                           so bits 0,1, and 3 are set.
   'sounumbers',vector of source numbers (optional)
                           If the text–string 'sounumbers' is given,
                           followed by a vector of integers, only those
                           source numbers are plotted.
   'recnumbers',vector of source numbers (optional)
                           If the text–string 'recnumbers' is given,
                           followed by a vector of integers, only those
                           source numbers are plotted.
   'edgenumbers',vector of edge numbers (optional)
                           If the text–string 'edgenumbers' is given,
                           followed by a vector of integers, only those
                           edge numbers are plotted with a solid line.
                           The others are plotted with dashed lines.
   'figurewindow' a desired figure number. If not specified, figure number
   1 will be used.

 Output parameters:
   plothandles     A struct with fields that give handles to the various
                   parts of the figure:
                   .parent   To set the size of the axis values, change
                             the parameter 'FontSize' of this figure
                             handle etc.
                   .sources
                   .receivers
                   .edges        A list, one value for each edge
                   .edgecircles  A list, one value for each edge
                   .edgenumbers  A list, one value for each edge
                   .conumbers    A list, one value for each corner
                   .nvecs        A list, one value for each nvec
                   .nveccircles  A list, one value for each nvec
                   .planenumbers A list, one value for each plane

 Sources and Rdata.receivers are taken from an sdatafile and an rdatafile, the file name of
 which is assumed to be similar to the eddatafile.

 Uses functions EDstrpend, EDstrpblnk

  plothandles = EDplotmodel(eddatafile,varargin);
```

5.2 EDverify

```
EDverify runs EDtoolbox for one or several defined tests,
 and compares the results to expected results. A logfile is written.

 Input parameters:
   outputdirectory (optional) The directory were the logfile will be
                   stored. If this parameter is not given any value, no
                   logfile will be written, but the user will get a window
                   to specify a directory where the temporary calculation
                   results will be stored.
   runtest         (optional) A vector of 0 or 1, which for each position n
                   tells if test n should be run. Default value: all 1.
   showtext        (optional) 0 or 1; determines if the results should be printed on
                   the screen. The results are always written to the
```

```
                     logfile. Default value: 0.
      plotdiagrams    (optional) 0 or 1: determines if result plots will be
                     generated. Default value: 0.

    Output parameter:
      passtest        A vector with −1 (fail), 0 (not run), or 1 (pass) for
      all the tests.

    Tests:
    1. Response at cuboid surface, plane wave incidence, 0 Hz, no singularity
    problem. HOD test.
    2. Field continuity across zone boundaries, single edge, 100 Hz.
    Perpendicular edge hit. Diff1 test.
    3. Field continuity across zone boundaries, single edge, 100 Hz.
    Skewed edge hit. Diff1 test.
    4. Field continuity across corner zone boundaries, single edge, 100 Hz.
    Diff1 test.
    5. Field continuity for receivers close to a single edge, 100 Hz.
    Diff1 test.
    6. Replicate a non−centered internal monopole, at 0.1 Hz.
    7. Direct sound obscuring for a corner−on hit of an octahedron.
    8. Direct sound obscuring for an edge−on hit of a cube.

    Peter Svensson 7 Oct. 2023 (peter.svensson@ntnu.no)

    passtest = EDverify(outputdirectory,runtest,showtext,plotdiagrams);
```

5.3 EDcirclepoints

```
  EDcirclepoints distributes point coordinates across a circular surface.

    Input parameters
      radius              The piston radius, in meters
      numberofcircles     The number of circles of point sources. The value 1
                          will lead to 9 point sources. The value 2 gives 25
                          point sources etc

    Output parameters
      coordinates         A matrix, size [npoints,3], with the xyz−coordinate
                          of the npoints points. They all have the
                          z−coordinate 0.

    coordinates = EDcirclepoints(radius,numberofcircles)
```

5.4 EDmakegeo_shoebox

```
  This function generates the corners and planecorners matrices on the form
    that is used in the EDtoolbox for an external shoebox. By default, the
    corners are symmetrically distributed in the x− and y−directions, but
    placed at z = 0 and z = −length_z.

    Input parameters:
      length_x,length_y, length_z     The sizes in meters

    Output parameters:
      corners             A matrix of size [8,3] giving the x−,y− and
                          z−coordinates of the 8 corners.
      planecorners        A matrix of size [6,4] giving the corners for each
                          of the 6 planes.

    [corners,planecorners] = EDmakegeo_shoebox(length_x,length_y,length_z);
```

5.5 EDmakegeo_cylinder

```
    EDmakegeo_cylinder generates the geometry for a polyhedral cylinder.
    The end caps of the cylinder will be parallel to the z=0 plane,
    symmetrically placed. Optionally, the whole cylinder can be translated in
    the z−direction.

    The polygonal shape will be scaled so that regardless of numbers of
    edges, the volume of the polygonal cylinder will be the same as a truly
    circular cylinder with the radius given as input parameter.

    Input parameters:
          radius                  The radius of the equivalent circular cylinder
          width                   The length of the cylinder
          numberofcorners The number of corners approximating the circular cross−section
```

```
     intextgeom              'int' or 'ext' defining if an interior or exterior geometry
                                     should be constructed
     fullcirclefract 1 or 0.5, indicating whether a full or half cylinder should be created.
  angleoffset         (optional) 1 or 0, indicating whether or not the first corner
                             should be placed in y = 0 (angleoffset = 0) or shifted
                                     half an angle step (angleoffset = 1).
  ztranslation    (optional) A value which will be added to all the
                  z-coordinates.

Output parameters:
  corners         Matrix, [2*numberofcorners,3], with the corner
                  coordinates
  planecorners    Matrix, [numberofcorners+2,numberofcorners], with the
                  plane corners. The first numberofcorners rows have the
                  planes of the cylindrical surface, and the two last
                  rows have the flat circular endsurfaces.
  ncornersperplanevec     A vector which gives the number of corners per
                  plane
     radius                  The actual radius of the corners; this will be different
                                     from the (desired) input parameter 'radius', since the
                                     polygonal approximation of the circle is scaled so that
                                     it gets the same cross-section area as the real circle.

[corners,planecorners,ncornersperplanevec,radius] =
EDmakegeo_cylinder(radius,width,numberofcorners,intextgeom,fullcirclefract,angleoffset,ztranslation);
```

6 Example scripts

Some examples of different types are given below. These scripts are hopefully easy to modify to individual needs. In Section 2.3, a loudspeaker simulation was demonstrated, with a single point source representing the loudspeaker element.

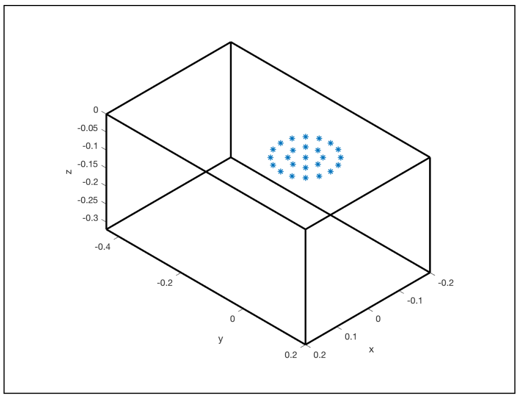
6.1 EDexample_LspKessel_piston.m

This example demonstrates how to simulate a circular piston, with the script EDexample_LspKessel_piston.m (which can be found in the examples folder). Fig. 9 shows the results, and the model (without the receiver position), respectively. In Fig. 9 (b), also the frequency response from Fig. 2 (b) is shown for comparison.

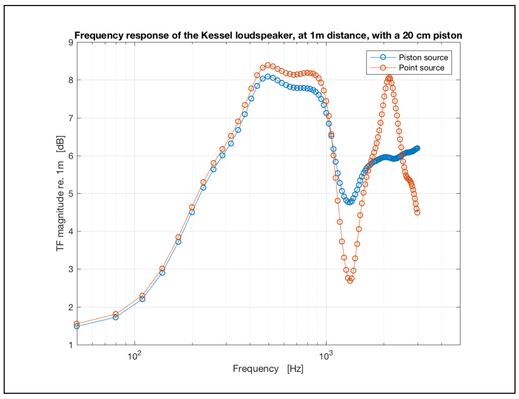
The auxiliary function EDcirclepoints distributes points equally in n concentric circles, and positions them around the origin in the $z = 0$ plane, see Section 5.

```
% EDexample_LspKessel_piston.m
%

mfile = mfilename('fullpath');
[infilepath,filestem] = fileparts(mfile);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Here the size of the shoebox-shaped enclosure is defined
% The function EDmakegeo_shoebox centers the box around the origin but we
% shift it in the x-direction so that the origin (where the source will be
% has 0.2m to all edges)

[corners,planecorners] = EDmakegeo_shoebox(0.4,0.64,0.32);
corners(:,2) = corners(:,2) - 0.12;
geoinputdata = struct('corners',corners,'planecorners',planecorners);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The source (piston-like distribution of monopoles, using the function EDcirclepoints) and receiver are defined

sourcecoordinates = EDcirclepoints(0.1,2);
sourcecoordinates(:,3) = 0.00001;
nsources = size(sourcecoordinates,1);
Sinputdata = struct('coordinates',sourcecoordinates);
Sinputdata.doaddsources = 1;

receivercoordinates = [0 0 1];
Rinputdata = struct('coordinates',receivercoordinates);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some calculation parameters

fvec = linspace(50,3000,100);
controlparameters = struct('frequencies',fvec);
controlparameters.docalctf = 1;
nfrequencies = length(controlparameters.frequencies);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Output file names and location

filehandlingparameters = struct('outputdirectory',[infilepath,filesep,'results']);
filehandlingparameters.filestem = filestem;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run the calculations, for the piston case

filehandlingparameters_piston.filestem = filehandlingparameters.filestem;
EDres_piston = EDmain_convex(geoinputdata,Sinputdata,Rinputdata,struct,controlparameters,filehandlingparameters);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Make the changes for a single monopole source and rerun the calculations

Sinputdata.coordinates = Sinputdata.coordinates(1,:);

filehandlingparameters.filestem = [filehandlingparameters.filestem,'_monopole'];
EDres_monopole = EDmain_convex(geoinputdata,Sinputdata,Rinputdata,struct,controlparameters,filehandlingparameters);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Present the results

tftot_piston = EDres_piston.tftot/nsources;
tftot_point = EDres_monopole.tftot;

figure(2)
h = semilogx(controlparameters.frequencies,20*log10(abs([tftot_piston tftot_point])),'-o');
g = get(h(1),'Parent');
set(g,'FontSize',14);
set(h(1),'LineWidth',2);
set(h(2),'LineWidth',2);
g = xlabel('Frequency   [Hz]');
set(g,'FontSize',14)
```

```
g = ylabel('TF magnitude re. 1m   [dB]');
set(g,'FontSize',14)
g = title('Frequency response of the Kessel loudspeaker, at 1m distance, with a 20 cm piston');
set(g,'FontSize',14)
% axis([50 5000 0 10])
xlim([50 5000])
grid
g = legend('Piston source','Point source');
set(g,'Location','best')
set(g,'FontSize',14)

eddatafile = [infilepath,filesep,'results',filesep,filehandlingparameters_piston.filestem,'_eddata.mat'];
EDplotmodel(eddatafile,1)
```



(a)



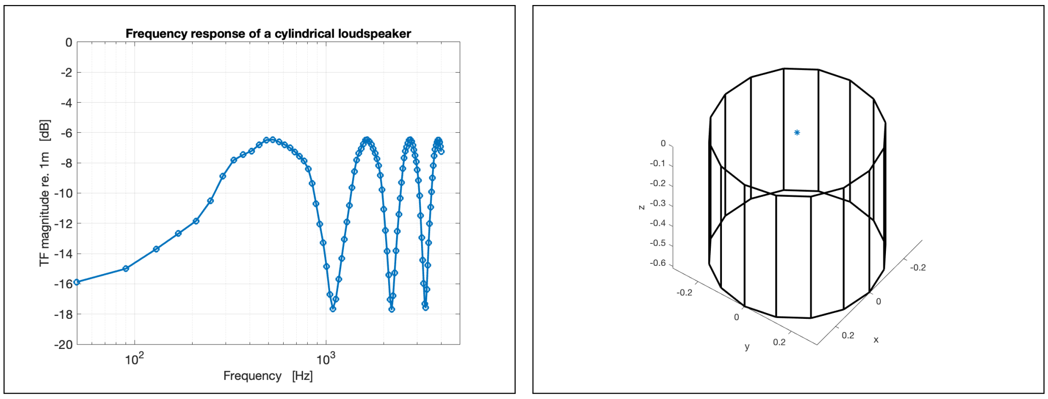
(b)

Figure 9: Results from the `EDexample_LspKessel_piston.m` script, using the "Kessel" loudspeaker example [?], with a piston simulated as a number of monopoles. (a) The loudspeaker model, with source and receiver positions. (b) The frequency response at 1m distance, for the set of monopoles, as well as for a single monopole, computed with `EDmain_convex`. The single-monopole result is identical to the result in Fig. 2(b).

6.2 EDexample_LspCylinder16.m

The shape of a loudspeaker enclosure has much influence on the frequency response, as shown in [H. Olson, "Direct radiator loudspeaker enclosures," J. Audio Eng. Soc. 17, pp. 22-29, 1969]. The enclosure shape with the strongest diffraction effect is the cylinder, with a source centrally placed, and that is demonstrated by the example `EDexample_LspCylinder16.m`. The result diagram in Fig. 10 tries to mimic the results in the paper by Olson. It was not stated in that paper what the distance to the microphone, so it was chosen to generate the same dip frequencies as in the published frequency response. Parts of the script contents are given below. The auxiliary function `EDmakegeo_cylinder` is used to generate the polygonal cylinder shape, see Section 5.

```
% EDexample_LspCylinder16.m
%

mfile = mfilename('fullpath');
[infilepath,filestem] = fileparts(mfile);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define the cylinder geometry using the function EDmakegeo_cylinder

radius = 0.3048;
length = 2*radius;
[corners,planecorners,ncorners,radius] = EDmakegeo_cylinder(radius,length,16,'e',1,0,-radius);
geoinputdata = struct('corners',corners,'planecorners',planecorners);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The source (single point) and receiver are defined

sourcecoordinates = [0 0 0.00001];
Sinputdata = struct('coordinates',sourcecoordinates);

receivercoordinates = [0 0 6.6];
Rinputdata = struct('coordinates',receivercoordinates);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some calculation parameters

fvec = linspace(50,4000,100);
controlparameters = struct('frequencies',fvec);
controlparameters.docalctf = 1;
controlparameters.ngauss = 24;
controlparameters.difforder = 30;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Output file names and location

filehandlingparameters = struct('outputdirectory',[infilepath,filesep,'results']);
filehandlingparameters.filestem = filestem;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run the calculations

EDres = EDmain_convex(geoinputdata,Sinputdata,Rinputdata,struct,controlparameters,filehandlingparameters);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Present the results

figure(2)
clf(2)
h = semilogx(controlparameters.frequencies,20*log10(abs(EDres.tftot)),'-o');
g = get(h(1),'Parent');
set(g,'FontSize',14);
set(h(1),'LineWidth',2);
g = xlabel('Frequency   [Hz]');
set(g,'FontSize',14)
g = ylabel('TF magnitude re. 1m   [dB]');
set(g,'FontSize',14)
g = title('Frequency response of the Kessel loudspeaker, at 1m distance, with a 20 cm piston');
set(g,'FontSize',14)
% axis([50 5000 0 10])
xlim([50 5000])
grid

figure(1)
clf(1)
eddatafile = [infilepath,filesep,'results',filesep,filehandlingparameters_piston.filestem,'_eddata.mat'];
EDplotmodel(eddatafile,1)
```



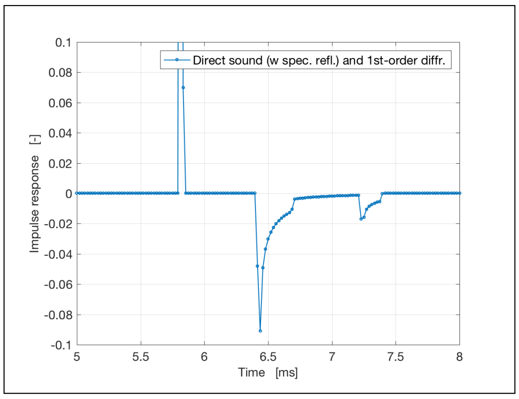
(a) (b)

Figure 10: A cylindrical loudspeaker modeled as a 16-sided polygonal cylinder, with a point source. (a) The frequency response at 6.6 m distance. (b) The loudspeaker model, with the point source marked.

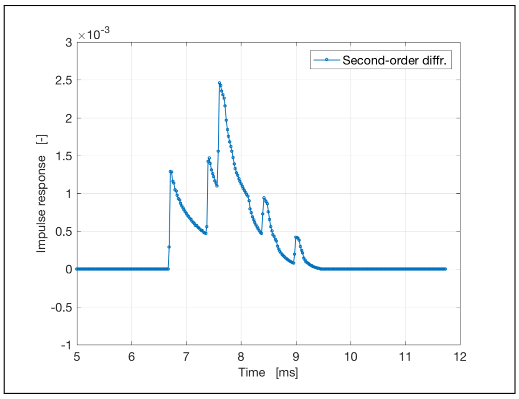
6.3 EDexample_LspKessel_ir.m

This example demonstrates how low-order diffraction impulse responses can be computed, with the script `EDexample_LspKessel_ir.m`. The result diagrams in Fig. 11 show the impulse responses of the individual diffraction orders, and the corresponding frequency responses (transfer functions), including different numbers of diffraction orders. In Fig. 11 (d), a low-pass filtering effect can be observed for the direct sound and specular reflection, which coincide. The cause of this is the simple conversion of the direct sound Dirac pulse to a two-sample impulse response. Such an approach is the simplest possible "fractional delay" version, which gives a very small phase error for the direct sound but a substantial magnitude error. A simple remedy is to doubel or quadrauple the sampling frequency. These impulse responses correspond to the results in [**?**], which used an edge diffraction model with a substantial low-frequency error, but with identical results to the method here for higher frequencies.

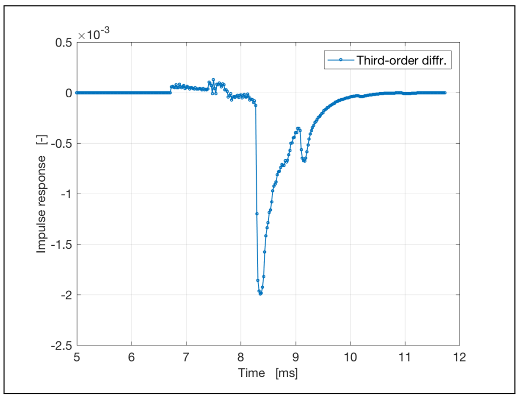
```
% EDexample_LspKessel_ir.m

mfile = mfilename('fullpath');
[infilepath,filestem] = fileparts(mfile);

[corners,planecorners] = EDmakegeo_shoebox(0.4,0.64,0.32);
corners(:,2) = corners(:,2) — 0.12;
geoinputdata = struct('corners',corners,'planecorners',planecorners);
Sinputdata = struct('coordinates',[0 0 0.0001]);
Rinputdata = struct('coordinates',[0 0 2]);
controlparameters = struct('fs',48000);
controlparameters.docalcir = 1;
controlparameters.difforder = 3;
controlparameters.savealldifforders = 1;
filehandlingparameters = struct('outputdirectory',[infilepath,filesep,'results']);
filehandlingparameters.filestem = filestem;
filehandlingparameters.savelogfile = 1;
filehandlingparameters.showtext = 0;

EDres = EDmain_convex(geoinputdata,Sinputdata,Rinputdata,struct,controlparameters,filehandlingparameters);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Present the results

ntotlength = size(EDres.irdirect,1);
allirtots = zeros(ntotlength,4);

allirtots(:,1) = EDres.irdirect + EDres.irgeom;
allirtots(:,2) = EDres.irdiff;
allirtots(:,3) = EDres.irhod{2};
allirtots(:,4) = EDres.irhod{3};

cumsumirtots = cumsum(allirtots.').';

tvec = 1/controlparameters.fs*[0:ntotlength—1].';

figure(1)
clf(1)
h = plot(tvec*1e3,allirtots(:,1)+allirtots(:,2),'—o');
set(h(1),'LineWidth',1,'MarkerSize',3);
g = get(h(1),'Parent');
```



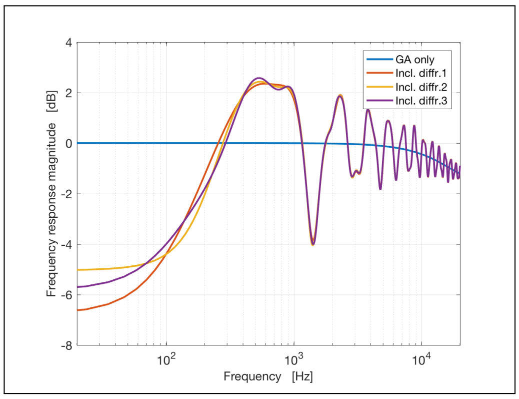
(a)



(b)



(c)



(d)

Figure 11: Diffraction impulse responses and frequency responses for the Kessel loudspeaker case, computed with `EDmain_convex_time`. (a) Direct sound and first-order diffraction components. The direct sound, around 5.8 ms, has an amplitude which is not shown, in order to see the weaker first-order diffraction components. (b) Second-order diffraction components. (c) Third-order diffraction components. (d) The corresponding frequency responses.

```
set(g,'FontSize',14)
g = xlabel('Time   [ms]');
set(g,'FontSize',14)
g = ylabel('Impulse response   [—]');
set(g,'FontSize',14)
grid
axis([5 8 —0.1 0.1])
g = legend('Direct sound (w spec. refl.) and 1st—order diffr.');
set(g,'FontSize',14)

figure(2)
clf(2)
h = plot(tvec*1e3,allirtots(:,3),'—o');
set(h(1),'LineWidth',1,'MarkerSize',3);
g = get(h(1),'Parent');
set(g,'FontSize',14)
g = xlabel('Time   [ms]');
set(g,'FontSize',14)
g = ylabel('Impulse response   [—]');
set(g,'FontSize',14)
grid
axis([5 12 —0.001 0.003])
g = legend('Second—order diffr.');
set(g,'FontSize',14)

figure(3)
clf(3)
h = plot(tvec*1e3,allirtots(:,4),'—o');
set(h(1),'LineWidth',1,'MarkerSize',3);
g = get(h(1),'Parent');
set(g,'FontSize',14)
g = xlabel('Time   [ms]');
set(g,'FontSize',14)
g = ylabel('Impulse response   [—]');
set(g,'FontSize',14)
grid
```

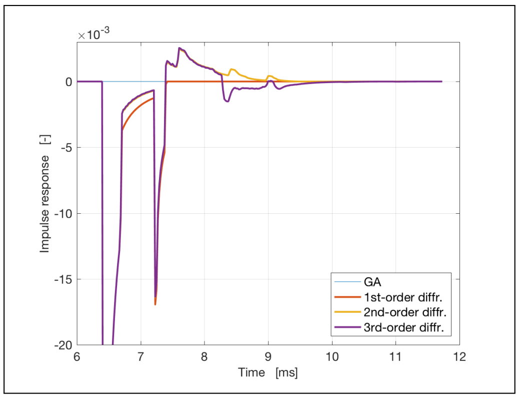
```
axis([5 12 -0.0025 0.0005])
g = legend('Third-order diffr.');
set(g,'FontSize',14)


nfft = 4096;
fvec = controlparameters.fs/nfft*[0:nfft/2-1];
F = fft(cumsumirtots,nfft);

figure(4)
clf(4)
h = semilogx(fvec,20*log10(abs(F(1:nfft/2,:))));
for ii = 1:4
   set(h(ii),'LineWidth',2)
end
g = get(h(1),'Parent');
set(g,'FontSize',14)
grid
g = xlabel('Frequency   [Hz]');
set(g,'FontSize',14)
g = ylabel('Frequency response magnitude   [dB]');
set(g,'FontSize',14)
g = legend('GA only','Incl. diffr.1','Incl. diffr.2','Incl. diffr.3');
axis([20 20000 -8 4])
```

6.4 EDexample_LspKessel_ir_and_tf.m

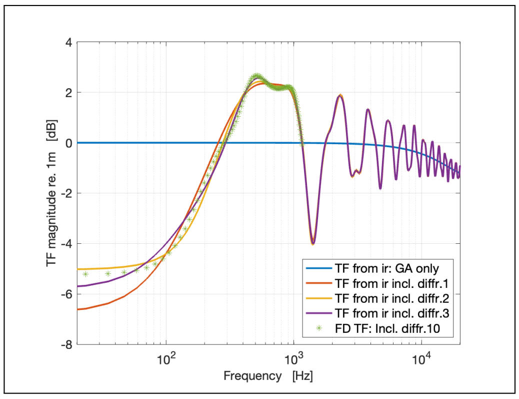
This example expands on the previous one by showing that the impulse response, with low-order diffraction, can be quite accurate for mid to high frequencies, whereas the higher-order diffraction calculation, in the frequency domain, can be employed for the LF range. The script `EDexample_LspKessel_ir_and_tf.m` runs both `EDmain_convex_time` and `EDmain_convexESIE`. The result diagrams in Fig. 12 show the impulse responses of the individual diffraction orders, as in the previous example, and the corresponding frequency responses (transfer functions), including different numbers of diffraction orders. Apparantly, the difference between the results with diffraction orders up to 3 and up to 15, is getting very small above approximately 1 kHz.

```
% EDexample_LspKessel_ir_and_tf.m

mfile = mfilename('fullpath');
[infilepath,filestem] = fileparts(mfile);

[corners,planecorners] = EDmakeshoeboxgeo(0.4,0.64,0.32);
corners(:,2) = corners(:,2) − 0.12;
geoinputdata = struct('corners',corners,'planecorners',planecorners);
Sinputdata = struct('coordinates',[0 0 0.0001]);
Rinputdata = struct('coordinates',[0 0 2]);
controlparameters = struct('fs',48000);
controlparameters.difforder = 3;
controlparameters.docalcir = 1;
controlparameters.savealldifforders = 1;
filehandlingparameters = struct('outputdirectory',[infilepath,filesep,'results']);
filehandlingparameters.filestem = filestem;

EDres_ir = EDmain_convex(geoinputdata,Sinputdata,Rinputdata,struct,controlparameters,filehandlingparameters);

nfft = 4096;
fvec = controlparameters.fs/nfft∗[0:nfft/2−1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

controlparameters.ngauss = 24;
controlparameters.difforder = 10;
controlparameters.docalcir = 0;
controlparameters.docalctf = 1;

fvec_tf = fvec(1:100);
controlparameters.frequencies = fvec_tf;

EDres_tf = EDmain_convex(geoinputdata,Sinputdata,Rinputdata,struct,controlparameters,filehandlingparameters);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Present the results

ntotlength = size(EDres_ir.irdirect,1);
allirtots = zeros(ntotlength,4);
allirtots(:,1) = EDres_ir.irdirect + EDres.irgeom;
allirtots(:,2) = EDres_ir.irdiff;
allirtots(:,3) = EDres_ir.irhod{2};
allirtots(:,4) = EDres_ir.irhod{3};

cumsumirtots = cumsum(allirtots.').';

tftot = EDres_tf.tftot;

tvec = 1/controlparameters.fs∗[0:ntotlength−1];

figure(1)
clf(1)
h = plot(tvec∗1e3,cumsumirtots,'−');
set(h(2),'LineWidth',2);
set(h(3),'LineWidth',2);
set(h(4),'LineWidth',2);
g = get(h(1),'Parent');
set(g,'FontSize',14)
grid
g = xlabel('Time   [ms]');
set(g,'FontSize',14)
g = ylabel('Impulse response   [−]');
set(g,'FontSize',14)
axis([6 12 −0.02 0.003])
g = legend('GA','1st−order diffr.','2nd−order diffr.','3rd−order diffr.');
set(g,'FontSize',14,'Location','SouthEast')


F = fft(cumsumirtots,nfft);

figure(2)
clf(2)
h = semilogx(fvec,20∗log10(abs(F(1:nfft/2,:))),fvec_tf,20∗log10(abs(tftot)),'∗');
for ii = 1:4
   set(h(ii),'LineWidth',2)
```

```
end
g = get(h(1),'Parent');
set(g,'FontSize',14)
grid
g = xlabel('Frequency   [Hz]');
set(g,'FontSize',14)
g = ylabel('Frequency response magnitude   [dB]');
set(g,'FontSize',14)
g = legend('GA only','Incl. diffr.1','Incl. diffr.2','Incl. diffr.3',['Incl. diffr.',int2str(controlparameters.difforder)]);
set(g,'FontSize',14,'Location','SouthEast')
axis([20 20000 −8 4])
```



(a)



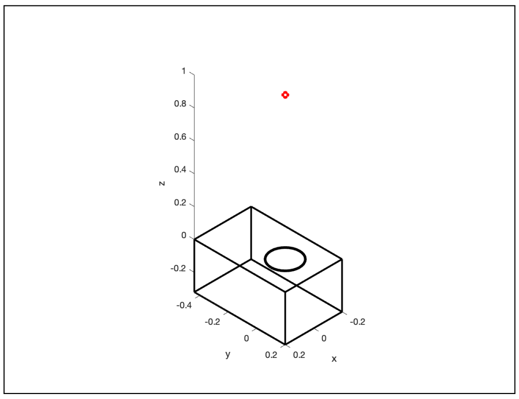
(b)

Figure 12: Diffraction impulse responses and frequency responses for the Kessel loudspeaker case. (a) Direct sound and low-order diffraction impulse responses. The direct sound, around 5.8 ms, is not shown. (b) The corresponding frequency responses (via an FFT of the IRs), and the frequency response computed directly in the frequency domain, including diffraction order 10.

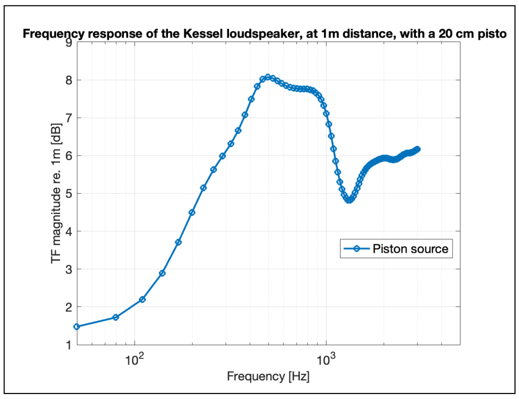
6.5 EDexample_LspKessel_piston_edgewave.m

In this example, a circular piston is represented by a regular, 32-sided polygon. As described in Section 4.3, the direct sound is computed using the piston edge wave representation of the Rayleigh integral, as described in [?], whereas the diffraction terms are computed as a sum of discrete monopoles. The results are given in Fig. 13.

```
% EDexample_LspKessel_pistonedgewave.m
%

mfile = mfilename('fullpath');
[infilepath,filestem] = fileparts(mfile);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Here the size of the shoebox-shaped enclosure is defined
% The function EDmakegeo_shoebox centers the box around the origin but we
% shift it in the x-direction so that the origin (where the source will be)
% has 0.2m to all edges.

[corners,planecorners] = EDmakegeo_shoebox(0.4,0.64,0.32);
corners(:,2) = corners(:,2) - 0.12;
geoinputdata = struct('corners',corners,'planecorners',planecorners);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The source is modeled as a 32-sided regular polygon approximation of a
% circular piston. The "polygonradius" is adjusted so that the regular
% polygon gets the same area as the circular piston that it tries to
% represent.
% For the diffraction terms, a number of monopoles are representing the
% piston.

circpistonradius = 0.1;
npistoncorners = 32;
polygonradius = circpistonradius*sqrt(2*pi/npistoncorners/sin(2*pi/npistoncorners));
phivec = [0:npistoncorners-1].'*2*pi/npistoncorners;
pistoncorners = polygonradius*[cos(phivec) sin(phivec) zeros(size(phivec))];

ngaussorder = 3;

Sinputdata = struct('sourcetype','polygonpiston');
Sinputdata.pistoncornercoordinates = pistoncorners;
Sinputdata.pistoncornernumbers = [1:npistoncorners];
Sinputdata.pistonplanes = 1;
Sinputdata.pistongaussorder = ngaussorder;

receivercoordinates = [0 0 1];
Rinputdata = struct('coordinates',receivercoordinates);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some calculation parameters

fvec = linspace(50,3000,100);
controlparameters = struct('frequencies',fvec);
controlparameters.docalctf = 1;
nfrequencies = length(controlparameters.frequencies);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Output file names and location

filehandlingparameters = struct('outputdirectory',[infilepath,filesep,'results']);
filehandlingparameters.filestem = filestem;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run the calculations, for the piston case

EDres = EDmain_convex(geoinputdata,Sinputdata,Rinputdata,struct,controlparameters,filehandlingparameters);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Present the results

tftot_piston = EDres.tftot;

figure(2)
clf(2)
h = semilogx(controlparameters.frequencies,20*log10(abs([tftot_piston ])),'-o');
g = get(h(1),'Parent');
set(g,'FontSize',14);
set(h(1),'LineWidth',2);
g = xlabel('Frequency [Hz]');
set(g,'FontSize',14)
g = ylabel('TF magnitude re. 1m [dB]');
set(g,'FontSize',14)
g = title('Frequency response of the Kessel loudspeaker, at 1m distance, with a 20 cm piston');
set(g,'FontSize',14)
xlim([50 5000])
grid
g = legend('Piston source');
set(g,'Location','best')
set(g,'FontSize',14)

figure(1)
clf(1)
eddatafile = [infilepath,filesep,'results',filesep,filehandlingparameters.filestem,'_eddata.mat'];
EDplotmodel(eddatafile,3);
```



(a)



(b)

Figure 13: The Kessel loudspeaker, with a 20-cm diameter piston source. (a) The geometrical model. (b) The resulting frequency response, which is practically identical to the response in Fig. 9 (b) The similarity illustrates how the amplitude is scaled for a piston source.

6.6 EDexample_Rayleigh_piston_circpistondirectivity.m

The directivity of a circular piston, as represented by a regular, 32-sided polygon, in an infinite baffle is computed by including only the direct sound. In the same way as for the previous script, the direct sound is computed using the piston edge wave representation of the Rayleigh integral, as described in [?]. The geometry is plotted in Fig. 14 (a) and the resulting directivity function, on a linear magntidue scale, is shown in Fig. 14 (b) for three frequencies.

```
% EDexample_Rayleigh_circpistondirectivity.m
%

mfile = mfilename('fullpath');
[infilepath,filestem] = fileparts(mfile);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% A 10m*10m dummy plate is generated, acting as an infinite baffle

corners = 5*[1 1 0;-1 1 0;-1 -1 0; 1 -1 0];
planecorners = [1 2 3 4;4 3 2 1];
geoinputdata = struct('corners',corners,'planecorners',planecorners);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The source is modeled as a 32-sided regular polygon approximation of a
% circular piston. The "polygonradius" is adjusted so that the regular
% polygon gets the same area as the circular piston that it tries to
% represent.
%
% To study the directivity, we create an arc of receiver

circpistonradius = 0.1;
npistoncorners = 32;
polygonradius = circpistonradius*sqrt(2*pi/npistoncorners/sin(2*pi/npistoncorners));
phivec = [0:npistoncorners-1].'*2*pi/npistoncorners;
pistoncorners = polygonradius*[cos(phivec) sin(phivec) zeros(size(phivec))];

Sinputdata = struct('sourcetype','polygonpiston');
Sinputdata.pistoncornercoordinates = pistoncorners;
Sinputdata.pistoncornernumbers = [1:npistoncorners];
Sinputdata.pistonplanes = 1;

thetavec = linspace(0,0.999*pi/2,100);

Rdistance = 100;
receivercoordinates = Rdistance*[sin(thetavec.') zeros(size(thetavec.')) cos(thetavec.')] ;
Rinputdata = struct('coordinates',receivercoordinates);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some calculation parameters
% No diffraction.
% We choose 3 frequencies to illustrate three degrees of directivity

fvec = [100 1e3 5e3].';
controlparameters = struct('frequencies',fvec);
controlparameters.docalctf = 1;
controlparameters.difforder = 0;
nfrequencies = length(controlparameters.frequencies);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Output file names and location

filehandlingparameters = struct('outputdirectory',[infilepath,filesep,'results']);
filehandlingparameters.filestem = filestem;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run the calculations

EDres = EDmain_convex(geoinputdata,Sinputdata,Rinputdata,struct,controlparameters,filehandlingparameters);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Present the results

tftot_piston = EDres.tftot*Rdistance/2;

figure(2)
clf(2)
h = plot(thetavec*180/pi,(abs([tftot_piston ])),'-o');
g = get(h(1),'Parent');
set(g,'FontSize',14);
set(h(1),'LineWidth',2);
g = xlabel('Radiation angle [deg.]');
set(g,'FontSize',14)
g = ylabel('TF magnitude re. on-axis [-]');
set(g,'FontSize',14)
g = title('Directivity function of a 20 cm piston');
set(g,'FontSize',14)
%xlim([50 5000])
```

```
grid
g = legend('100 Hz','1 kHz','5 kHz');
set(g,'Location','best')
set(g,'FontSize',14)

figure(1)
clf(1)
eddatafile = [infilepath,filesep,'results',filesep,filehandlingparameters.filestem,'_eddata.mat'];
EDplotmodel(eddatafile,3);
```

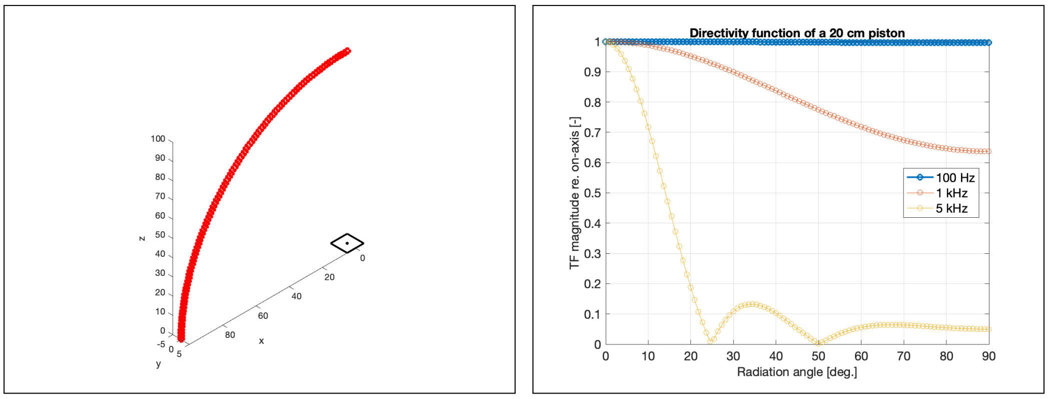


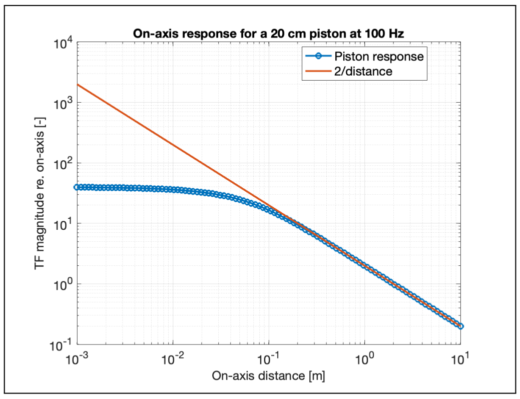
Figure 14: (a) A circular piston with 20 cm diameter is mounted in an infinite baffle, and a number of far-field receiver positions are distributed along a 90-degree arc. The baffle is plotted as a square plate, but the edge plates have no meaning here. (b) The resulting directivity functions, on a linear magnitude scale, for three different frequencies.

6.7 EDexample_Rayleigh_piston_circpistonnearfield.m

In this script, the on-axis response of a circular piston of diameter 20 cm, as represented by a regular, 32-sided polygon, in an infinite baffle is computed by including only the direct sound. A number of receiver positions range from very close to the vibrating surface, to far away. In the same way as for the previous script, the direct sound is computed using the piston edge wave representation of the Rayleigh integral, as described in [?]. The resulting on-axis response, is shown on a log-log scale in Fig. 15 for a single low frequency. It is clear that a transition from the near-field to the farfield occurs around 0.1 m.

```
% EDexample_Rayleigh_circpistonnearfield.m
%

mfile = mfilename('fullpath');
[infilepath,filestem] = fileparts(mfile);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% A 2m*2m dummy plate is generated, acting as an infinite baffle

corners = 1*[1 1 0;-1 1 0;-1 -1 0; 1 -1 0];
planecorners = [1 2 3 4;4 3 2 1];
geoinputdata = struct('corners',corners,'planecorners',planecorners);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The source is modeled as a 32-sided regular polygon approximation of a
% circular piston. The "polygonradius" is adjusted so that the regular
% polygon gets the same area as the circular piston that it tries to
% represent.
%
% The receivers are placed along the on-axis

circpistonradius = 0.1;
npistoncorners = 32;
polygonradius = circpistonradius*sqrt(2*pi/npistoncorners/sin(2*pi/npistoncorners));
phivec = [0:npistoncorners-1].'*2*pi/npistoncorners;
pistoncorners = polygonradius*[cos(phivec) sin(phivec) zeros(size(phivec))];

Sinputdata = struct('sourcetype','polygonpiston');
Sinputdata.pistoncornercoordinates = pistoncorners;
Sinputdata.pistoncornernumbers = [1:npistoncorners];
Sinputdata.pistonplanes = 1;

nreceivers = 100;
zvec = logspace(log10(1e-3),log10(10),nreceivers);
receivercoordinates = [zeros(nreceivers,2) zvec(:)];
Rinputdata = struct('coordinates',receivercoordinates);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some calculation parameters
% No diffraction.
% We choose 1 frequency

fvec = [100 ].';
controlparameters = struct('frequencies',fvec);
controlparameters.docalctf = 1;
controlparameters.difforder = 0;
nfrequencies = length(controlparameters.frequencies);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Output file names and location

filehandlingparameters = struct('outputdirectory',[infilepath,filesep,'results']);
filehandlingparameters.filestem = filestem;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run the calculations

EDres = EDmain_convex(geoinputdata,Sinputdata,Rinputdata,struct,controlparameters,filehandlingparameters);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Present the results

tftot_piston = EDres.tftot;

figure(2)
clf(2)
h = loglog(zvec,(abs([tftot_piston ])),'-o',zvec,2./zvec,'-');
g = get(h(1),'Parent');
set(g,'FontSize',14);
set(h(1),'LineWidth',2);
set(h(2),'LineWidth',2);
g = xlabel('On-axis distance [m]');
set(g,'FontSize',14)
g = ylabel('TF magnitude re. on-axis [-]');
set(g,'FontSize',14)
g = title('On-axis response for a 20 cm piston at 100 Hz');
```

```
set(g,'FontSize',14)
%xlim([50 5000])
grid
g = legend('Piston response','2/distance');
set(g,'Location','best')
set(g,'FontSize',14)

figure(1)
clf(1)
eddatafile = [infilepath,filesep,'results',filesep,filehandlingparameters.filestem,'_eddata.mat'];
EDplotmodel(eddatafile,3);
```



(a)

Figure 15: The on-axis response, on a log-log scale, for a circular piston, diameter 0.2m, in an infinite baffle and a single frequency of 100 Hz.

7 Appendix - Program structure for EDmain_convex

The main functions `EDmain_convex` runs through a sequence of processing blocks, listed in Tables 19 and described briefly below.

Table 19: The processing block sequence of `EDmain_convex`

EDmain_convex	**Comments**
1. `EDcheckinputstructs`	—
2. `EDreadcad`/`EDreadgeomatrices`	—
3. Create a mesh of surface receiver points[1]	If `.docalctf_ESIEBEM` = 1
4. `EDedgeo`	—
5. `EDSorRgeo`(`.Sdata`)	—
6. `EDSorRgeo`(`.Rdata`)	—
7. `EDfindconvexGApaths`	—
8A-B. `EDmakefirstordertfs`	If `controlparameters.docalctf` = 1
8C. `EDmakefirstorderirs`	If `controlparameters.docalcir` = 1
9. `EDed2geo`	If `controlparameters.difforder` > 1
10A-B. `EDinteg_submatrixstructure`	If `controlparameters.difforder` > 1 and (`.docalctf` = 1 or `.docalctf_ESIEBEM` = 1)
11A-B. `EDintegralequation_convex_tf`	If `controlparameters.difforder` > 1 and (`.docalctf` = 1 or 11B. `.docalctf_ESIEBEM` = 1)
12B. `EDpropagateESIEBEM`	If `.docalctf_ESIEBEM` = 1
10C. `EDfindHODpaths`	If `controlparameters.docalcir` = 1
11C. `EDmakeHODirs`	If `controlparameters.docalcir` = 1

[1] This part is in the main code of `EDmain_convex`.

7.1 EDcheckinputstructs

This function checks the input data and assigns default values to input parameters that have not been specified.

```
Input:  geoinputdata,Sinputdata,Rinputdata,envdata,controlparameters,
        filehandlingparameters
Output:  geoinputdata,Sinputdata,Rinputdata,envdata,controlparameters,
        filehandlingparameters
```

7.2 EDreadcad or EDreadgeomatrices

The geometry can either be specified in a separate .cad file, or given as input data matrices. See more on this topic in Section 4.2. The data is stored in a struct called `planedata`, see Table 13 for details.

For `EDreadcad`:
```
Input:  geoinputdata.geoinputfile
Output:  planedata, extraCATTdata
```

For `EDreadgeomatrices`:
```
Input:  geoinputdata.corners,geoinputdata.planecorners
Output:  planedata
```

7.3 EDedgeo

This function identifies all the edges of the polyhedron, and stores data about them in a separate struct called `edgedata`, see Table 15 for details. It is possible to suppress some edges, for special studies.

```
Input:  planedata,geoinputdata.firstcornertoskip (optional), geoinputdata.listofcornerstoskip
Output:  planedata, edgedata
```

7.4 EDSorRgeo

This function is run twice, once to find the visibility data for the source(s), and the second time to find the visibility data for the receiver(s). The visibility data tells what edges and planes each source and receiver can see. The structs `Sdata` and `Rdata` are described in Table 16.

```
Input:  planedata,edgedata,Sinputdata.coordinates or
        planedata,edgedata,Rinputdata.coordinates
Output:  Sdata or
         Rdata
```

7.5 EDfindconvexGApaths

This functions finds all the valid direct sound paths, specular reflection paths, and first-order diffraction paths. This function is specialized for the case of external, convex scattering objects, and for such objects, each source-receiver combination can have maximum one specular reflection. The results are stored in a struct called `firstorderpathdata`, see Table 17.

```
Input:  planedata,edgedata, Sdata.coordinates, .visplanesfromS,.vispartedgesfromS,
        Rdata.coordinates, .visplanesfromR, .vispartedgesfromR,
        controlparameters.difforder, .directsound, Sinputdata.doallSRcombinations
Output:  firstorderpathdata
```

Notes

The specular reflections are found with the image source method: all potentially visible image sources are constructed, using the `Sdata.visplanesfroms` matrix. Then, the visibility for each receiver is determined by checking if the line from the image source to the receiver passes through the intended finite reflection plane. This visibility test employs the same test as the direct sound obscurity test: point-in-polygon. The ray-shooting algorithm in the 2D-flattened polygon version is used for this test. When a hit is very close to an edge, or a corner, the amplitude is reduced from 1 to 0.5, for both the direct sound and the specular reflection.

The first-order diffraction components are found by combining the matrices `Sdata.vispartedgesfroms` and `Rdata.vispartedgesfromr`. Edges that are visible from both the source and the receiver should generate first-order edge diffraction.

7.6 EDmakefirstordertfs

Based on the paths specified in the struct `firstorderpathdata`, the function `EDmakefirstordertfs` generates the transfer functions `tfdirect, tfgeom,` and `tfdiff`.

```
Input:  firstorderpathdata,controlparameters,envdata,Sinputdata,
        Rdata.receivers,edgedata
Output:  tfdirect,tfgeom,tfdiff,timingdata
```

Notes

The direct sound and specular reflection components are straightforward to compute as

$$\begin{cases}\texttt{tfdirect} \\ \texttt{tfgeom}\end{cases} = A\frac{e^{-jkr}}{r}$$

where A is 0.5 if an edge or corner hit occured, and 1 otherwise.

First-order edge diffraction is computed with numerical solution of the integrals presented in [Svensson et al, AAA, 2009]. Matlab's built-in `quadgk` is used for this, yielding high accuracy. For certain source-receiver positions, this integral gets a strong singularity, and then, an analytical integration is carried out for a very small part of the edge, around the apex point. A simplified version of the formulation in [Svensson, Calamia, AAA, 2006] is applied for that analytical integration.

7.7 EDed2geo

This function is run only if difforder > 1. It identifies which edges see which other edges, and stores this information in a struct `edgetoedgedata`.

```
Input:  edgedata,planedata,Sdata,Rdata, controlparameters.specorder, controlparameters.difforder
Output:  edgetoedgedata
```

7.8 EDinteg_submatrix

This function is run only if difforder > 1. It identifies and sets up the submatrix structure for the subsequent integral equation solving, by the function `EDintegral_convex_tf`.

```
Input:   edgedata.edgelengthvec,edgedata.closwedangvec,
         edgedata.planesatedge,edgetoedgedata
         controlparameters.ngauss,controlparameters.discretizationtype,
Output:  Hsubmatrixdata
```

See notes in Section 7.9

7.9 EDintegral_convex_tf

This function is run only if difforder > 1. It computes the ackumulation of higher-order diffraction, from order 2 up to a specified diffraction order. The result is stored in the transfer function `tfinteqdiff`.

The higher-order edge diffraction is computed with the numerical solution of the edge source integral equation (ESIE) presented in [Asheim, JASA, 2013]. The solution involves two stages. In a first stage, so-called edge source amplitudes are computed. In a second stage the diffracted sound pressure at receiver points is computed by "propagating the edge source signal to the external field points" by computing a double integral.

```
Input:   envdata,planedata,edgedata,edgetoedgedata,Hsubmatrixdata
         Sdata,Sinputdata.doaddsources,Sinputdata.sourceamplitudes,
         Sinputdata.doallSRcombinations,Rdata,controlparameters,
         filehandlingparameters
Output:  tfinteqdiff,timingdata
```

Notes

The edge-source amplitudes are found by solving the ESIE. The straightforward Nystrom method is used, implying that the integral equation is sampled in discretization points along the edges, here called "edge points" or "gauss points". A Gauss-Legendre quadrature scheme is very efficient for this computation, which gets formulated as a matrix equation:

$$\mathbf{q} = \mathbf{q_0} + \mathbf{H}\mathbf{q} \tag{3}$$

where $\mathbf{q}$ is a vertical vector of all the edge source amplitudes to compute. The term $\mathbf{q_0}$ gives the contribution from the external monopole source(s), and is computed explicitly without any integration. $\mathbf{H}$ is a huge matrix containing the edge-point-to-edge-point interaction terms, or integral Kernel values. This matrix is computed only if `difforder` > 2, since the $\mathbf{q_0}$ is the exact solution for `difforder` $= 2$. The matrix equation is solved by iteration, and each iteration step corresponds to one diffraction order. Thus, if one iteration step is computed, then the resulting $\mathbf{q}$ will give diffracted sound of maximum diffraction orders 3, etc.

The vector of unknowns to compute, $\mathbf{q}$, consists of sections, where each segment contains all combinations, from all edge points on one edge (the "from-edge"), to all edge points on another edge (the "to-edge"). The locations, and identities, of those sections are described in the matrices `Hsubmatrixdata .edgepairlist`, `Hsubmatrixdata.bigmatrixstartnums`, and `Hsubmatrixdata .bigmatrixendnums`. Each of these matrices has `nedgepairs` rows, see Table 18. Each row in `.edgepairlist` gives the "to-edge" number in column 1 and the "from-edge" number in column 2. The same row in `.bigmatrixstartnums` gives the starting position in the $\mathbf{q}$-vector. Since edges might have different numbers of edge points, each section of $\mathbf{q}$ might have a different length.

This subdivision of the $\mathbf{q}$-vector into sections leads to that the large $\mathbf{H}$-matrix is composed of blocks, or submatrices, and consequently a very sparse structure. These submatrices are stored as individual matrices, `Hsub` (being sparse, they are actually stored as two lists: one of locations, and one of values, see below). The submatrices have different sizes since each submatrix connects one section of the $\mathbf{q}$-vector to another. The matrix `Hsubmatrixdata.Hsubmatrix`, of size `[nedgepairs,nedgepairs]`, gives a zero, or an integer. The zero implies that that part of the $\mathbf{H}$-matrix is zero, whereas an integer refers to the individual `Hsub`-matrix. There will be a total of `nsub` submatrices, and the matrices `Hsubmatrixdata.listofsubmatrices`, `Hsubmatrixdata.edgetripletlist`,
`Hsubmatrixdata.reftoshortlist`, `Hsubmatrixdata.isthinplanetriplet` all have `nsub` rows. Each row in `.edgetripletlist` gives the "to-edge" in pos. 1, the "via-edge" in pos. 2, and the "from-edge" in pos. 3, for the submatrix. The matrix `.listofsubmatrices` contains a re-ordered list of submatrix numbers in column 1. By going through the submatrices in this re-ordered order, the sizes of the submatrix are going through as few changes as possible. This is advantageous for the processing speed, since large matrices don't have to change size more often than necessary. Furthermore, for scattering objects with symmetries, many submatrices will be identical. Therefore, the list `.reftoshortlist` contains the "stand-in" submatrices that can be used: if `.reftoshortlist`(72) == 12, then submatrix 72 can use the contents of submatrix 12, and consequently, no new matrix entries need to be computed for submatrix 72.
Finally, each submatrix is sparse as well, and its contents are stored as two lists: `Hsubdatalistsnn` and `ivuselistsnn`, rather than being stored as a matrices. So, `ivuselistsnn` is a long list of the locations in the submatrix number nn, and `Hsubdatalistsnn` contains the corresponding data values.
The actual solution of Eq. (3) is done iteratingly, that is, with the Neumann approach:

$$\mathbf{q}_N = \sum_{i=0}^{N} \mathbf{q}_i, \qquad \mathbf{q}_i = \mathbf{H}\mathbf{q}_{i-1}, \qquad , i >= 1$$

The value for N has to be set manually through the parameter `controlparameters .difforder`, and N = `.difforder-2`.

As the last step, the sound pressure at the receiver point is computed as a discretized version of a double integral. This is efficently computed as a dot product

$$p = \mathbf{f} \bullet \mathbf{q}_N$$

where $\mathbf{f}$ is a vertical vector containing sampled kernel values for the propagation double integral.

7.10 EDfindHODpaths

This function is run only if difforder > 1, and `controlparameters.docalcir = 1`. It identifies which higher-order diffraction paths are possible, up to a certain diffraction order, and stores this information in a struct `hodpaths`.

```
Input:  edgetoedgedata.edgeseesedge,Sdata.visedgesfroms,
Rdata.visedgesfromr,controlparameters.difforder
Output:  hodpaths
```

7.11 EDmakeHODirs

This function is run only if difforder > 1, and `controlparameters.docalcir = 1`.It computes higher-order diffraction irs, and stores them information in a cell variable `irhod`, that might optionally contain each diffraction order separately.

```
Input:  hodpaths,controlparameters.difforder,elemsize,edgedata,
        edgetoedgedata,Sdata,Sinputdata.doaddsources,Sinputdata.sourceamplitudes,
        Rdata,envdata.cair,controlparameters.fs,controlparameters.Rstart,
        controlparameters.savealldifforders
Output:  irhod
```